

**ISO/IEC JTC 1/SC 29/WG 11 N 18693**

**ISO/IEC JTC 1/SC 29/WG 11**

**Coding of moving pictures and audio**

**Convenorship: UNI (Italy)**

|  |  |
| --- | --- |
| **Document type:** | **Approved WG 11 document** |
|  |  |
| **Title:** | **Test Model 6 of Versatile Video Coding (VTM 6)** |
|  |  |
| **Status:** | **Approved** |
|  |  |
| **Date of document:** | **2019-07-12** |
|  |  |
| **Source:** | **WG 11 (via JVET)** |
|  |  |
| **Expected action:** | **Study** |
|  |  |
| **No. of pages:** | **88** |
|  |  |
| **Email of convenor:** | **leonardo@chiariglione.org** |
|  |  |
| **Committee URL:** | **https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg11** |

|  |  |
| --- | --- |
| Joint Video Experts Team (JVET)  of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11  15th Meeting: Gothenburg, SE, 3–12 July 2019 | Document: JVET-O2002-v2 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Algorithm description for Versatile Video Coding and Test Model 6 (VTM 6)** | | |
| *Status:* | Output document of JVET | | |
| *Purpose:* | Algorithm description for Versatile Video Coding and Test Model 6 | | |
| *Author(s) or Contact(s):* | Jianle Chen Yan Ye Seung Hwan Kim | Email: | [jianle.chen@futurewei.com](mailto:jianle.chen@futurewei.com) [Yan.Ye@alibaba-inc.com](mailto:Yan.Ye@alibaba-inc.com) seunghwan3.kim@lge.com |
| *Source:* | Editors | | |

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

# Abstract

The JVET established the Versatile Video Coding (VVC) working draft 5 and the VVC Test Model 6 (VTM6) algorithm description and encoding method at its 15th meeting (19–27 3–12 July, Gothenburg, SE). This document serves as a source of general tutorial information on the VVC design and also provides an encoder-side description of VTM6. The VVC has been developed by a joint collaborative team of ITU-T and ISO/IEC experts known as the Joint Video Experts Team (JVET), which is a partnership of ITU-T Study Group 16 Question 6 (known as VCEG) and ISO/IEC JTC 1/SC 29/WG 11 (known as MPEG). This draft new standard has been designed with two primary goals. The first of these is to specify a video coding technology with a compression capability that is substantially beyond that of the prior generations of such standards, and the second is for this technology to be highly versatile for effective use in a broadened range of applications. Some key application areas for the use of this standard particularly include ultra-high-definition video (e.g., with 3840×2160 or 7620×4320 picture resolution and bit depth of 10 or 12 bits as specified in Rec. ITU-R BT.2100), video with a high dynamic range and wide colour gamut (e.g., with the perceptual quantization or hybrid log-gamma transfer characteristics specified in Rec. ITU-R BT.2100), and video for immersive media applications such as 360° omnidirectional video projected using a common projection format such as the equirectangular or cubemap projection format, in addition to the applications that have commonly been addressed by prior video coding standards.

Ed. Notes:

VVC Test Model 6 (VTM6) algorithm description and encoding method

* Editorial improvements in the section on screen content coding tools
* Incorporated JVET-O1124: CCLM restrictions for dualtree to reduce latency
* Incorporated JVET-O0050: Small chroma block size restrictions for shared tree
* Incorporated JVET-O0640: Restriction on small chroma blocks
* Incorporated JVET-O0106: ISP restriction on prediction block size
* Incorporated JVET-O0277: Restriction on small block sizes
* Incorporated JVET-O0364: Intra prediction simplifications
* Incorporated JVET-O0426: MRL reference samples for DC mode
* Incorporated JVET-O0502: 67 modes for ISP
* Incorporated JVET-O0655: Wide-angle in chroma intra angle mapping table for 4:2:2
* Incorporated JVET-O1153: Intra chroma mode coding cleanup
* Incorporated JVET-O0925: MIP 8-bit coefficient and simplifications
* Incorporated JVET-O0315: Intra prediction mode alignment for BDPCM
* Incorporated JVET-O1136: Unified TS and BDPCM signalling
* Incorporated JVET-O0258: Disabling IBC for chroma in case of dual tree
* Incorporated JVET-O0455: Number of IBC merge candidates independent for P/B slices
* Incorporated JVET-O1170: Bitstream conformance with a virtual IBC buffer concept
* Incorporated JVET-O0650: Signalling of chroma QP tables
* Incorporated JVET-O1168: CU level chroma QP control
* Incorporated JVET-O0428: LMCS related clean-ups
* Incorporated JVET-O0272: simplified inverse luma mapping
* Incorporated JVET-O1109: Unification of chroma residual scaling
* Incorporated JVET-O0432: LMCS encoder improvement
* Incorporated JVET-O0057: Half pel AMVR extension with alternative IF
* Incorporated JVET-O0070: Prediction refinement with optical flow for affine mode
* Incorporated JVET-O0119: Palette mode coding
* Incorporated JVET-O0304: unified gradient calculation for BDOF
* Incorporated JVET-O0055: BDOF subblock early termination threshold
* Incorporated JVET-O0108: Disabling DMVR and BDOF for CIIP
* Incorporated JVET-O0634: Unify allowed DMVR and BDOF block sizes
* Incorporated JVET-O0681: Disabling DMVR, BDOF and BCW for CIIP
* Incorporated JVET-O0366: BCW index for constructed affine merge candidate
* Incorporated JVET-O0590: Modified SAD for the center coordinate of DMVR search
* Incorporated JVET-O0265: Simplified MV storage for TPM
* Incorporated JVET-O0414: No SMVD for Long term reference picture
* Incorporated JVET-O0304: Multiplication reduction in BDOF
* Incorporated JVET-O0090: Alternative chroma filters + CTU chroma filter selection
* Incorporated JVET-O0662: Modified ALF filtering for Slice, Brick and Virtual boundaries
* Incorporated JVET- O0625: Apply VB when the bottom CTU boundary is a slice/tile/brick or “360 virtual” boundary
* Incorporated JVET-O0060: CE5-2.1: Deblocking on 4x4 sample grids
* Incorporated JVET-O0061: CE5-3.1 Sub-sample MV threshold for deblocking decisions
* Incorporated JVET-O0159: Non-CE5: Deblocking tC table defined for 10-bit video
* Incorporated JVET-O0094: Simplification of 48x16 LFNST matrices
* Incorporated JVET-O0472: LFNST index signalling depends on last position
* Incorporated JVET-O0368: Disable LFNST for non-DCT2 MTS candidate
* Incorporated JVET-O0529: Disable LFNST and MIP for implicit MTS
* Incorporated JVET-O0219: LFNST transform set selection for a CCLM
* Incorporated JVET-O0213: Limit LFNST up to max TU size
* Incorporated JVET-O0545: Configurable maximum transform size
* Incorporated JVET-O0919: QP clipping in scaling process for transform skip
* Incorporated JVET-O0052: TB-level constraints on context-coded bins
* Incorporated JVET-O0105: Joint chroma residual coding with multiple modes
* Incorporated JVET-O0122: Sign context, level mapping, and bitplane coding for TS residual coding
* Incorporated JVET-O0409: Exclude coded\_subblock\_flag in TSRC max ctx coded bin count
* Incorporated JVET-O0617: Context model reduction for sig\_coeff\_flag
* Incorporated JVET-O0543: Disallow joint chroma coding for non-I CUs
* Incorporated JVET-O0065: QP clipping in scaling process for transform skip
* Incorporated JVET-O0623: Residual coding for transform skip

VVC Test Model 5 (VTM5) algorithm description and encoding method

* Incorporated JVET-N0866: Unification of implicit transform selection
* Incorporated JVET-N0193: LFNST (Low-Frequency Non-Separable Transform)
* Incorporated JVET-N0105: Simplification of LFNST index coding
* Incorporated JVET-N0217: Matrix weighted intra prediction
* Incorporated JVET-N0246: Modified dequantization scaling
* Incorporated JVET-N0847: Support of quantization matrices
* Incorporated JVET-N0188: Unified rice parameter derivation for coefficient level coding
* Incorporated JVET-N0194: Context selection of last non-zero coefficient position in reduced TU
* Incorporated JVET-N0103: Coefficient group size harmonization
* Incorporated JVET-N0185: Unified MPM list for intra mode coding
* Incorporated JVET-N0137: Intra chroma partitioning and prediction restriction
* Incorporated JVET-N0435: Harmonization between WAIP and intra smoothing filters
* Incorporated JVET-N0308: Restriction of the maximum CU size for ISP to 64×64
* Incorporated JVET-N0271: CCLM derived with four neighbouring samples
* Incorporated JVET-N0415: CTU adaptive ALF, and fixed filter set.
* Incorporated JVET-N0242: Non-Linear Adaptive Loop Filtering (NL-ALF)
* Incorporated JVET-N0180: ALF line buffer reduction
* Incorporated JVET-N0473: Deblocking of ISP/SBT TU boundaries
* Incorporated JVET-N0266: Remove 4x4 unipred, and 4x8/8x4 bipred regular inter modes
* Incorporated JVET-N0340: Simplified Merge list construction for TPM
* Incorporated JVET-N0413: quantized residual DPCM
* Incorporated JVET-N0054: joint coding of chroma residuals
* Incorporated JVET-N0251 item 4 on IBC search range.
* Incorporated JVET-M0253 and JVET-N0247 on hash-based motion estimation.
* Incorporated JVET-N0280: residual coding for transform skip mode
* Incorporated JVET-N0325: using 8-bit fixed precision in BDOF
* Incorporated JVET-N0146: disable BDOF if BCW or WP is used
* Incorporated JVET-N0302: CIIP with position-independent weights
* Incorporated JVET-N0483: disallow the combination of subblock transform with triangle mode
* Incorporated JVET-N0286: simplified BCW index coding
* Incorporated JVET-M0140: Subblock transform for inter blocks
* Incorporated JVET-N0481: BCW index inheritance for constructed affine merge candidate
* Incorporated JVET-N0407: Disable 8x8/4xN CUs for DMVR
* Incorporated [JVET-N0868](http://phenix.it-sudparis.eu/jvet/doc_end_user/current_document.php?id=6623): DMVR reconciling with software ticket #214, 25 points SAD full search
* Incorporated JVET-N0178: Implicitly split BDOF application region along 16x16 boundaries
* Incorporated JVET-N0146: Align DMVR with BDOF on the conditions
* Incorporated JVET-N0447/N0400/N0500/N0851, signalling of triangle merge candidate number
* Incorporated JVET-M0444: Symmetric MVD coding

VVC Test Model 4 (VTM4) algorithm description and encoding method

* Incorporated JVET-M0118, JVET-M0328 and JVET-M0883: triangle prediction related changes
* Incorporated JVET-M0487 and JVET-M0063: BDOF related changes
* Incorporated JVET-M0273: SbTMVP related changes
* Incorporated JVET-M0111: BCW related changes:
* Incorporated JVET-M0453: CABAC core engine
* Incorporated JVET-M0142: Alternative CCLM downsampling filter
* Incorporated JVET-M0064: Reduced table size of CCLM parameter derivation
* Incorporated JVET-M0238: Simplification of PDPC reference samples
* Incorporated JVET-M0407: IBC reference region modification
* Incorporated JVET-M0297: 32-length DST-7/DCT-8 using zero-out
* Incorporated JVET-M0464: Unified MTS and transform skip syntax
* Incorporated JVET-M0173: rem\_abs\_gt3\_flag in first coding pass
* Incorporated JVET-M0246: Affine AMVR
* Incorporated JVET-M0427: luma mapping with chroma scaling (previously known as adaptive in-loop reshaper)
* Incorporated JVET-M0102: Intra subpartitions (ISP)
* Incorporated JVET-M0147: Decoder side motion vector refinement
* Incorporated JVET-M0483: Intra block copy
* Incorporated JVET-M0102: Intra Subpartitions
* Incorporated JVET-M 0471: Long tap Deblocking

VVC Test Model 3 (VTM3) algorithm description and encoding method

* Incorporated Adaptive Loop Filter
  + JVET-L0082: 10 b coeffs (instead of 11)
  + JVET-L0147: Subsampled Laplacian calculation
  + JVET-L0083: Reduction of bits for ALF coefficient fractional part
  + JVET-L0392: minor BF
  + JVET-L0664: Remove the signaling of 5x5 as a special case for luma
* JVET-L0081: 64x64 luma size virtual pipeline data units (VPDUs)
* Incorporated Affine related modification, including
  + JVET-L0265: set the chroma subblock size to 4x4 instead of 2x2
  + JVET-L0271: CE4.1.6: Simplification of affine AMVP candidate list construction
  + JVET-L0045: line buffer reduction for affine mode
  + JVET-L0632/L0142: affine merge refinement
  + JVET-L0369/L0055 : moving ATMVP into the affine merge list
* JVET-L0293: CPR mode for screen content coding
* JVET-L0646: bi-prediction with weighted averaging
* JVET-L0256: bi-directional optical flow
* JVET-L0231: horizontal wrap-around motion compensation
* JVET-L0377: Rounding Align of Adaptive Motion Vector Resolution
* JVET-L0198/L0468/L0104: fixed subblock size of 8x8 for SbTMVP mode
* JVET-L0104: disallow 4x4 bi-prediction
* Incorporated JVET-L0191: CCLM parameter derivation
* Incorporated JVET-L0136/JVET-L0085: CCLM with line buffer restriction
* Incorporated JVET-L0338/JVET-L0340: Multi-directional LM (MDLM)
* Incorporated JVET-L0053/JVET-L0272: chroma DM based on center position
* Incorporated JVET-L0279: unification of angular intra prediction
* Incorporated JVET-L0165: intra 6 MPM
* Incorporated JVET-L0059: simplification on MTS kernel derivation
* Incorporated JVET-L0111: transform skip condition on transform block size
* Incorporated JVET-L0285: 8-bit transform matrices
* Incorporated JVET-L0118: unified MTS signaling
* Incorporated JVET-L0553: quantization semantics fix
* Incorporated JVET-L0274: coefficient coding
* Incorporated JVET-L0628: mode dependent intra smoothing
* Incorporated JVET-L0283: multiple reference line intra prediction
* Incorporated JVET-L0414: DF strength dependent on reconstructed luma level
* Incorporated JVET-L0410: Deblocking tC table
* JVET\_L0124/L0208: triangle partition mode
* JVET-L0100: combined intra and inter prediction
* Added merge list generation process, including
  + Spatial MVP and Temporal MVP derivation
  + JVET-L0266/: History-based MVP from an FIFO table
  + JVET-L0090: Pairwise average MVP
* Incorporated JVET-L0054: merge with MVD (MMVD)

VVC Test Model 2 (VTM2) algorithm description and encoding method

* Incorporated JVET-K0230: Separate trees for intra slices (without multi-DMs) with an implicit split to 64x64;
* Incorporated JVET-K0556: Prohibit ternary split of something bigger than 64 in width or height (and not send the bit to indicate ternary type at that level).
* Incorporated JVET-K0351 (test c): Keep only the TT restriction (preventing binary split with same orientation in center partition of the ternary split)
* Incorporated JVET-K0554: Implicit splitting at picture boundaries and ensure MinQTSize at boundary splits
* Incorporated JVET-K0063: Position dependent intra prediction combination (PDPC)
* Incorporated JVET-K0190: CCLM only (test 4.1.8)
* Incorporated JVET-K0122: DC prediction bug fix
* Incorporated JVET-K0529: 67 modes with 3MPM and FLC for non-MPM
* Incorporated JVET-K0500: Wide-angle intra prediction for non-square block
* Incorporated MTS (AMT) modification: Multiple transform selection (MTS)
* Incorporated subblock TMVP
* Incorporated adaptive motion vector resolution
* Incorporated 8x8 and 1/16 pel motion field storage
* Incorporated affine motion

Contents

[Abstract 1](#_Toc18930851)

[1 Introduction 8](#_Toc18930852)

[2 Scope 8](#_Toc18930853)

[3 Algorithm description of Versatile Video Coding 8](#_Toc18930854)

[3.1 VVC coding architecture 8](#_Toc18930855)

[3.2 Partitioning 10](#_Toc18930856)

[3.2.1 Partitioning of the picture into CTUs 10](#_Toc18930857)

[3.2.2 Partitioning of pictures into slices, tiles, and bricks 11](#_Toc18930858)

[3.2.3 Partitioning of the CTUs using a tree structure 13](#_Toc18930859)

[3.2.4 CU splits on picture boundaries 16](#_Toc18930860)

[3.2.5 Restrictions on redundant CU splits 17](#_Toc18930861)

[3.2.6 Virtual pipeline data units (VPDUs) 17](#_Toc18930862)

[3.2.7 Intra chroma partitioning and prediction restriction 18](#_Toc18930863)

[3.3 Intra prediction 19](#_Toc18930864)

[3.3.1 Intra mode coding with 67 intra prediction modes 19](#_Toc18930865)

[3.3.2 Cross-component linear model prediction 22](#_Toc18930866)

[Position dependent intra prediction combination 25](#_Toc18930867)

[3.3.3 Multiple reference line (MRL) intra prediction 26](#_Toc18930868)

[3.3.4 Intra Sub-Partitions (ISP) 27](#_Toc18930869)

[Matrix weighted Intra Prediction (MIP) 30](#_Toc18930870)

[3.4 Inter prediction 32](#_Toc18930871)

[3.4.1 Extended merge prediction 32](#_Toc18930872)

[3.4.2 Merge mode with MVD (MMVD) 35](#_Toc18930873)

[3.4.3 Symmetric MVD coding 36](#_Toc18930874)

[3.4.4 Affine motion compensated prediction 37](#_Toc18930875)

[3.4.5 Subblock-based temporal motion vector prediction (SbTMVP) 43](#_Toc18930876)

[3.4.6 Adaptive motion vector resolution (AMVR) 44](#_Toc18930877)

[3.4.7 Motion field storage 45](#_Toc18930878)

[3.4.8 Bi-prediction with CU-level weight (BCW) 45](#_Toc18930879)

[3.4.9 Bi-directional optical flow (BDOF) 46](#_Toc18930880)

[3.4.10 Decoder side motion vector refinement (DMVR) 49](#_Toc18930881)

[3.4.11 Triangle partition for inter prediction 51](#_Toc18930882)

[3.4.12 Combined inter and intra prediction (CIIP) 52](#_Toc18930885)

[3.4.13 Miscellaneous inter prediction aspects 53](#_Toc18930886)

[3.5 Transform and quantization 53](#_Toc18930887)

[3.5.1 Large block-size transforms with high-frequency zeroing 53](#_Toc18930888)

[3.5.2 Multiple transform selection (MTS) for core transform 53](#_Toc18930889)

[3.5.3 Low-Frequency Non-Separable Transform (LFNST) 55](#_Toc18930890)

[3.5.4 Subblock Transform (SBT) 58](#_Toc18930891)

[3.5.5 Quantization 59](#_Toc18930892)

[3.5.6 Joint coding of chroma residuals 61](#_Toc18930893)

[3.6 Entropy coding 62](#_Toc18930894)

[3.6.1 Core CABAC engine 62](#_Toc18930895)

[3.6.2 Transform coefficient level coding 63](#_Toc18930896)

[3.6.3 Context modeling for coefficient coding 64](#_Toc18930897)

[3.7 In-loop filters 65](#_Toc18930898)

[3.7.1 Adaptive Loop Filter 65](#_Toc18930899)

[3.7.2 Deblocking filter 70](#_Toc18930900)

[3.7.3 Deblocking decision adapted to smaller difference in motion 73](#_Toc18930901)

[3.7.4 Luma mapping with chroma scaling (LMCS) 74](#_Toc18930902)

[3.8 360-degree video coding tools 78](#_Toc18930903)

[3.8.1 Horizontal wrap around motion compensation 78](#_Toc18930904)

[3.9 Screen content coding tools 79](#_Toc18930905)

[3.9.1 Intra block copy (IBC) 79](#_Toc18930906)

[3.9.2 Block differential pulse coded modulation (BDPCM) 81](#_Toc18930907)

[3.9.3 Residual coding for transform skip mode 82](#_Toc18930908)

[3.9.4 Palette mode 84](#_Toc18930909)

[4 Description of VTM6 encoder and encoding methods 86](#_Toc18930910)

[4.1 Derivation process of coding tree structure 86](#_Toc18930911)

[4.2 Hash based motion estimation for screen content coding 86](#_Toc18930912)

[5 References 87](#_Toc18930913)

# Introduction

At the 10th JVET meeting (April 10–20, 2018, San Diego, US), JVET defined the first draft of Versatile Video Coding (VVC) and the VVC Test Model 1 (VTM1) encoding method. It was decided to include a quadtree with nested multi-type tree using binary and ternary splits coding block structure as the initial new coding feature of VVC. Draft reference software to implement the VTM1 encoding method (and the draft VVC decoding process) has also been developed. At the 11th meeting (10–18 July, 2018, Ljubljana, SI), the Versatile Video Coding (VVC) working draft 2 and the VVC Test Model 2 (VTM2) algorithm description and encoding method were established with the inclusion of a group of new coding features as well as some of HEVC coding elements. At the 12th meeting (3–12 October, 2018, Macao, CN), the Versatile Video Coding (VVC) working draft 3 and the VVC Test Model 3 (VTM3) algorithm description and encoding method were established with the inclusion of additional coding tools that improves the coding performance. From the 13th meeting (9–18 January 2019, Marrakech, MA), the JVET experts start working on the simplification and unification of existing coding technologies, besides the new algorithms for additional coding efficiency. The Versatile Video Coding (VVC) working draft 4 and the VVC Test Model 4 (VTM4) algorithm description and encoding method were established with the inclusion of few more coding tools as well as lots of refinements of existing tools. At the 14th meeting (19–27 March 2019, Geneva, CH), the Versatile Video Coding (VVC) working draft 5 and the VVC Test Model 5 (VTM5) algorithm description and encoding method were established with the inclusion of very small number of new coding tools and a large amount of tool simplifications. The coding of 4:2:2 and 4:4:4 chroma formats are supported as well as 4:2:0 chroma formats. At the 15th meeting (3–12 July 2019, Gothenburg, SE), the Versatile Video Coding (VVC) working draft 6 and the VVC Test Model 6 (VTM6) algorithm description and encoding method were established. The major changes include refinement of existing tools, unification of coding tool combination, and HLS design. Also, some new functionalities, e.g. adaptive resolution change, scalability coding are supported.

# Scope

The normative decoding process for Versatile Video Coding is specified in the VVC draft 6 text specification document [1]. The VTM6.0 reference software is provided to demonstrate a reference implementation of non-normative encoding techniques and the normative decoding process for VVC. The reference software can be accessed via

https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware\_VTM.git

This document provides an algorithm description as well as an encoder-side description of the VVC Test Model 5, which serves as a tutorial for the algorithm and encoding model implemented in the VTM6.0 software. The purpose of this document is to share a common understanding of the coding features of VVC and the reference encoding methods supported in the VTM6.0 software, in order to facilitate the assessment of the technical impact of new technologies during the standardization process. Common test conditions and software reference configurations that should be used for experimental work for conventional standard-dynamic range rectangular video content are described in JVET-N1010 [2]. Common test conditions specific to video content with high dynamic range and wide colour gamut are described in JVET-N1011 [3]. Common test conditions specific to video content for 360° omnidirectional video applications are described in JVET-L1012 [4]. When encoding and decoding 360° omnidirectional video, an additional software package called the 360Lib needs to be used together with using the VTM software to process, encode/decode and compute the spherical quality metrics. The 360Lib software is available at:

https://jvet.hhi.fraunhofer.de/svn/svn\_360Lib/

Additionally, document JVET-M1004 [5] describes the algorithms used in 360Lib to process, code, and measure quality of 360° omnidirectional video.

# Algorithm description of Versatile Video Coding

## VVC coding architecture

As in most preceding standards, VVC has a block-based hybrid coding architecture, combining inter-picture and intra-picture prediction and transform coding with entropy coding. Figure 1 shows a general block diagram of the VTM6 encoder.

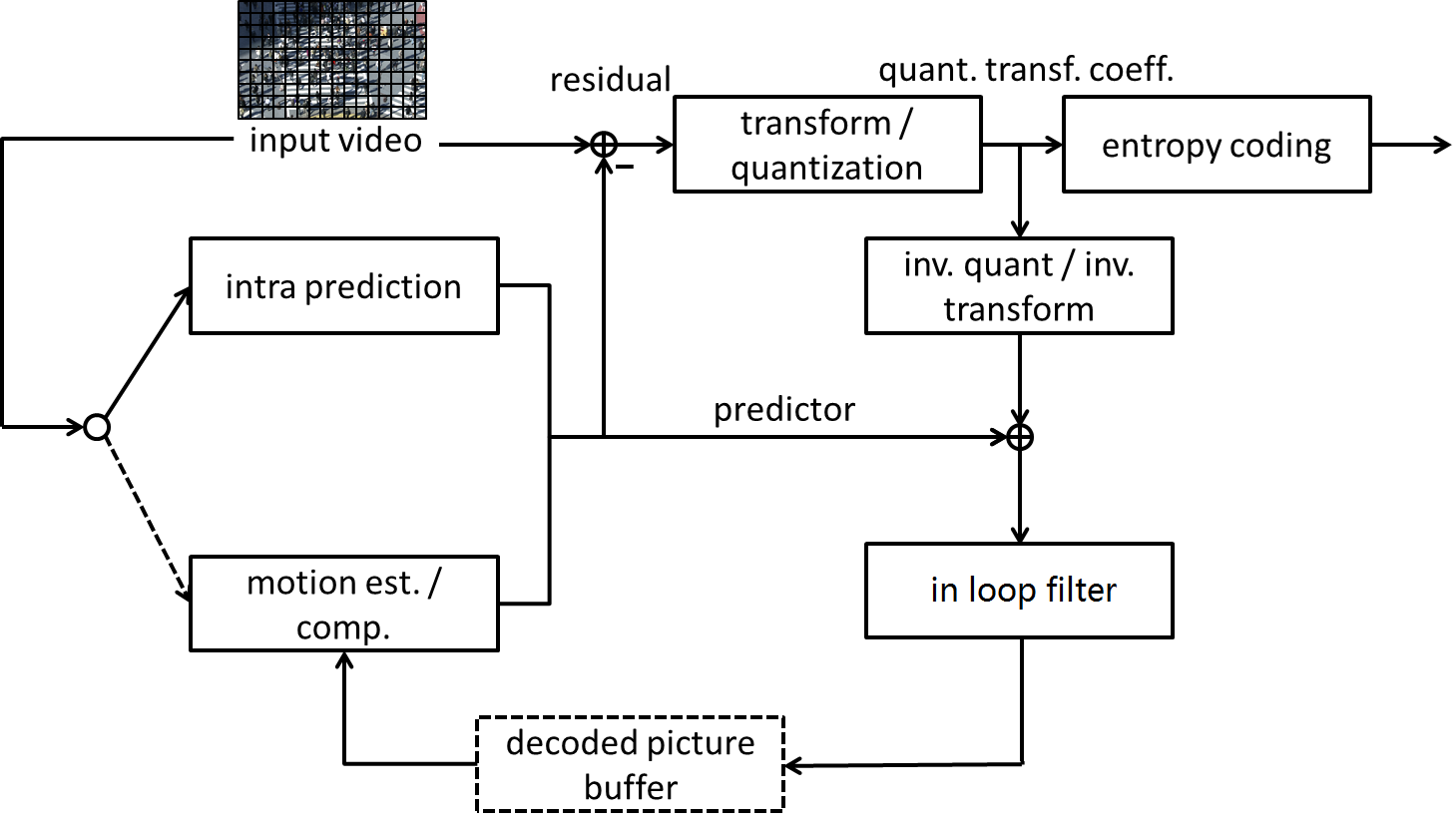


Figure 1 – General block diagram of VTM6 encoder [To be updated]

The picture partitioning structure, which is further described in section 3.2, divides the input video into blocks called coding tree units (CTUs). A CTU is split using a quadtree with nested multi-type tree structure into coding units (CUs), with a leaf coding unit (CU) defining a region sharing the same prediction mode (e.g. intra or inter). In this document, the term ‘unit’ defines a region of an image covering all colour components; the term ‘block’ is used to define a region covering a particular colour component (e.g. luma), and may differ in spatial location when considering the chroma sampling format such as 4:2:0.

The other features of VTM5, including intra prediction processes, inter picture prediction processes, transform and quantization processes, entropy coding processes and in-loop filter processes, are covered in sections 3.3 to 3.7. As agreed in the 11th JVET meeting, the following features have been included in the VVC test model 3 on top of the bock tree structure.

* Intra prediction
  + 67 intra mode with wide angles mode extension
  + Block size and mode dependent 4 tap interpolation filter
  + Position dependent intra prediction combination (PDPC)
  + Cross component linear model intra prediction
  + Multi-reference line intra prediction
  + Intra sub-partitions
  + Weighted intra prediction with matrix multiplication
* Inter-picture prediction
  + Block motion copy with spatial, temporal, history-based, and pairwise average merging candidates
  + Affine motion inter prediction
  + subblock based temporal motion vector prediction
  + Adaptive motion vector resolution
  + 8x8 block based motion compression for temporal motion prediction
  + High precision (1/16 pel) motion vector storage and motion compensation with 8-tap interpolation filter for luma component and 4-tap interpolation filter for chroma component
  + Triangular partitions
  + Combined intra and inter prediction
  + Merge with MVD (MMVD)
  + Symmetrical MVD coding
  + Bi-directional optical flow
  + Decoder side motion vector refinement
  + Bi-prediction with CU-level weight
* Transform, quantization and coefficients coding
  + Multiple primary transform selection with DCT2, DST7 and DCT8
  + Secondary transform for low frequency zone
  + Subblock transform for inter predicted residual
  + Dependent quantization with max QP increased from 51 to 63
  + Transform coefficient coding with sign data hiding
  + Transform skip residual coding
* Entropy Coding
  + Arithmetic coding engine with adaptive double windows probability update
* In loop filter
  + In-loop reshaping
  + Deblocking filter with strong longer filter
  + Sample adaptive offset
  + Adaptive Loop Filter
* Screen content coding:
  + Intra block copy with reference region restriction
  + Palette coding mode
* 360-degree video coding
  + Horizontal wrap-around motion compensation
* High-level syntax and parallel processing
  + Reference picture management with direct reference picture list signaling
  + Tile groups with rectangular shape tile groups

## Partitioning

### Partitioning of the picture into CTUs

Pictures are divided into a sequence of coding tree units (CTUs). The CTU concept is same to that of the HEVC [6][7]. For a picture that has three sample arrays, a CTU consists of an N×N block of luma samples together with two corresponding blocks of chroma samples.Figure 2 shows the example of a picture divided into CTUs.

The maximum allowed size of the luma block in a CTU is specified to be 128×128 (although the maximum size of the luma transform blocks is 64×64).



Figure 2 – Example of a picture divided into CTUs

### Partitioning of pictures into slices, tiles, and bricks

A picture is divided into one or more tile rows and one or more tile columns. A tile is a sequence of CTUs that covers a rectangular region of a picture.

A tile is divided into one or more bricks, each of which consisting of a number of CTU rows within the tile.

A tile that is not partitioned into multiple bricks is also referred to as a brick. However, a brick that is a true subset of a tile is not referred to as a tile.

A slice either contains a number of tiles of a picture or a number of bricks of a tile.

Two modes of slices are supported, namely the raster-scan slice mode and the rectangular slice mode. In the raster-scan slice mode, a slice contains a sequence of tiles in a tile raster scan of a picture. In the rectangular slice mode, a slice contains a number of bricks of a picture that collectively form a rectangular region of the picture. The bricks within a rectangular slice are in the order of brick raster scan of the slice.

Figure 3 shows an example of raster-scan slice partitioning of a picture, where the picture is divided into 12 tiles and 3 raster-scan slices.

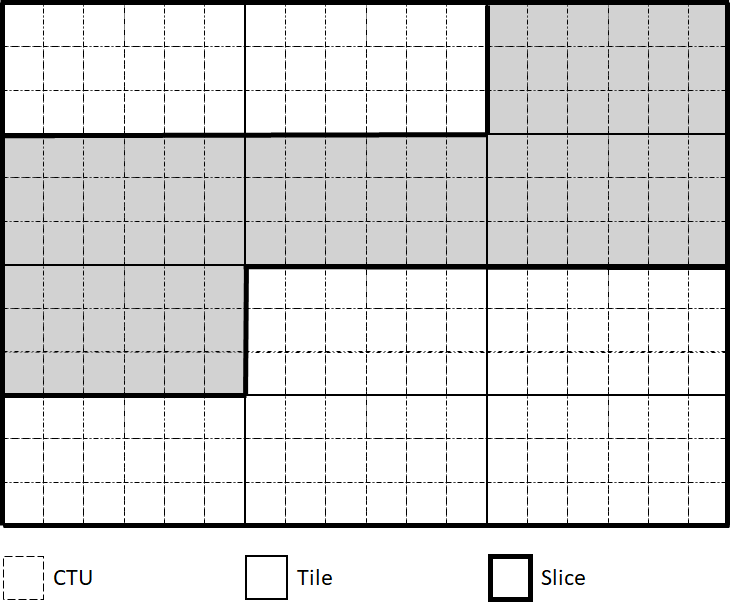


Figure 3 – Example of a picture partitioned into tiles and reaster-scan slices

Figure 4 shows an example of rectangular slice partitioning of a picture, where the picture is divided into 24 tiles (6 tile columns and 4 tile rows) and 9 rectangular slices.

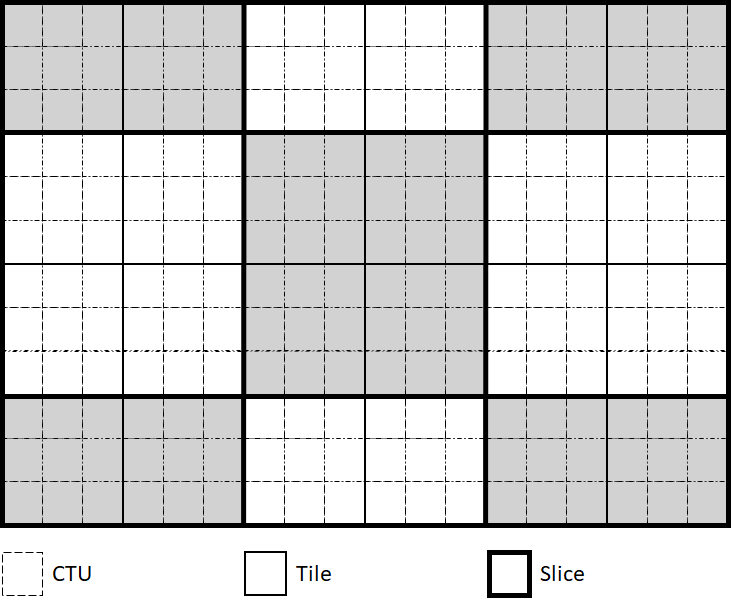


Figure 4 – Example of a picture partitioned into tiles and rectangular slices

Figure 5 shows an example of a picture partitioned into tiles, bricks, and rectangular slices, where the picture is divided into 4 tiles (2 tile columns and 2 tile rows), 11 bricks (the top-left tile contains 1 brick, the top-right tile contains 5 bricks, the bottom-left tile contains 2 bricks, and the bottom-right tile contain 3 bricks), and 4 rectangular slices.



Figure 5 – Example of a picture partitioned into tiles, bricks, and rectangular slices

### Partitioning of the CTUs using a tree structure

In HEVC, a CTU is split into CUs by using a quaternary-tree structure denoted as coding tree to adapt to various local characteristics. The decision whether to code a picture area using inter-picture (temporal) or intra-picture (spatial) prediction is made at the leaf CU level. Each leaf CU can be further split into one, two or four PUs according to the PU splitting type. Inside one PU, the same prediction process is applied and the relevant information is transmitted to the decoder on a PU basis. After obtaining the residual block by applying the prediction process based on the PU splitting type, a leaf CU can be partitioned into transform units (TUs) according to another quaternary-tree structure similar to the coding tree for the CU. One of key feature of the HEVC structure is that it has the multiple partition conceptions including CU, PU, and TU.

In VVC, a quadtree with nested multi-type tree using binary and ternary splits segmentation structure replaces the concepts of multiple partition unit types, i.e. it removes the separation of the CU, PU and TU concepts except as needed for CUs that have a size too large for the maximum transform length, and supports more flexibility for CU partition shapes. In the coding tree structure, a CU can have either a square or rectangular shape. A coding tree unit (CTU) is first partitioned by a quaternary tree (a.k.a. quadtree) structure. Then the quaternary tree leaf nodes can be further partitioned by a multi-type tree structure. As shown in Figure 6, there are four splitting types in multi-type tree structure, vertical binary splitting (SPLIT\_BT\_VER), horizontal binary splitting (SPLIT\_BT\_HOR), vertical ternary splitting (SPLIT\_TT\_VER), and horizontal ternary splitting (SPLIT\_TT\_HOR). The multi-type tree leaf nodes are called coding units (CUs), and unless the CU is too large for the maximum transform length, this segmentation is used for prediction and transform processing without any further partitioning. This means that, in most cases, the CU, PU and TU have the same block size in the quadtree with nested multi-type tree coding block structure. The exception occurs when maximum supported transform length is smaller than the width or height of the colour component of the CU.



Figure 6 – Multi-type tree splitting modes

Figure 7 illustrates the signalling mechanism of the partition splitting information in quadtree with nested multi-type tree coding tree structure. A coding tree unit (CTU) is treated as the root of a quaternary tree and is first partitioned by a quaternary tree structure. Each quaternary tree leaf node (when sufficiently large to allow it) is then further partitioned by a multi-type tree structure. In the multi-type tree structure, a first flag (mtt\_split\_cu\_flag) is signalled to indicate whether the node is further partitioned; when a node is further partitioned, a second flag (mtt\_split\_cu\_vertical\_flag) is signalled to indicate the splitting direction, and then a third flag (mtt\_split\_cu\_binary\_flag) is signalled to indicate whether the split is a binary split or a ternary split. Based on the values of mtt\_split\_cu\_vertical\_flag and mtt\_split\_cu\_binary\_flag, the multi-type tree slitting mode (MttSplitMode) of a CU is derived as shown in Table 3‑1.

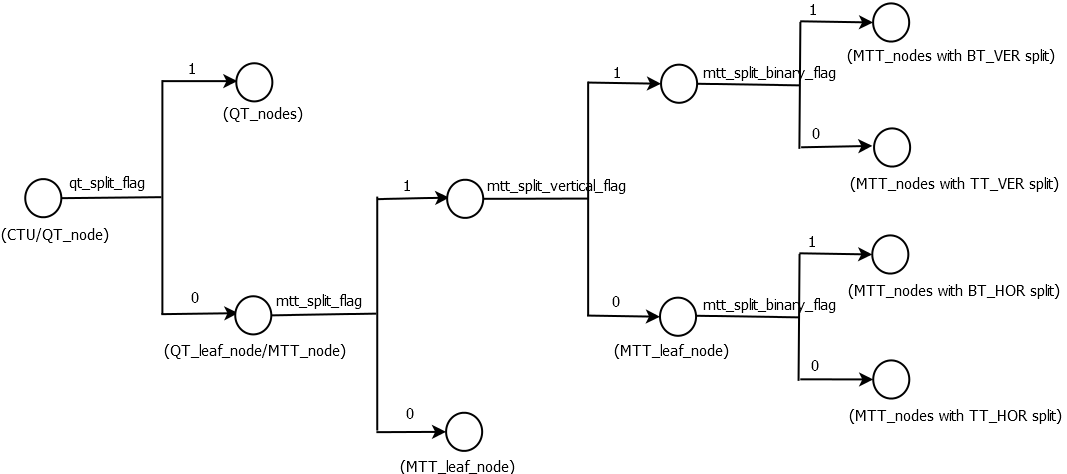


Figure 7 – Splitting flags signalling in quadtree with nested multi-type tree coding tree structure

**Table 3‑1 – MttSplitMode derviation based on multi-type tree syntax elements**

|  |  |  |
| --- | --- | --- |
| **MttSplitMode** | **mtt\_split\_cu\_vertical\_flag** | **mtt\_split\_cu\_binary\_flag** |
| SPLIT\_TT\_HOR | 0 | 0 |
| SPLIT\_BT\_HOR | 0 | 1 |
| SPLIT\_TT\_VER | 1 | 0 |
| SPLIT\_BT\_VER | 1 | 1 |

Figure 8 shows a CTU divided into multiple CUs with a quadtree and nested multi-type tree coding block structure, where the bold block edges represent quadtree partitioning and the remaining edges represent multi-type tree partitioning. The quadtree with nested multi-type tree partition provides a content-adaptive coding tree structure comprised of CUs. The size of the CU may be as large as the CTU or as small as 4×4 in units of luma samples. For the case of the 4:2:0 chroma format, the maximum chroma CB size is 64×64 and the minimum chroma CB size is 2×2.

In VVC, the maximum supported luma transform size is 64×64 and the maximum supported chroma transform size is 32×32. When the width or height of the CB is larger the maximum transform width or height, the CB is automatically split in the horizontal and/or vertical direction to meet the transform size restriction in that direction.

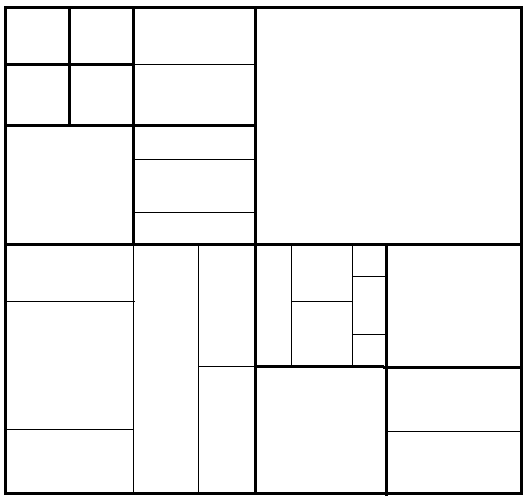


Figure 8– Example of quadtree with nested multi-type tree coding block structure

The following parameters are defined and specified by SPS syntax elements for the quadtree with nested multi-type tree coding tree scheme.

– CTU size: the root node size of a quaternary tree

– *MinQTSize*: the minimum allowed quaternary tree leaf node size

– *MaxBtSize*: the maximum allowed binary tree root node size

– *MaxTtSize*: the maximum allowed ternary tree root node size

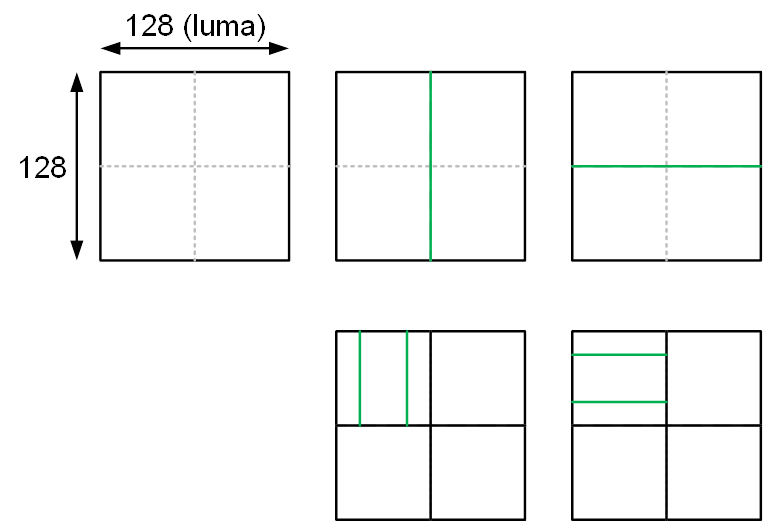
– *MaxMttDepth*: the maximum allowed hierarchy depth of multi-type tree splitting from a quadtree leaf

– *MinBtSize*: the minimum allowed binary tree leaf node size

– *MinTtSize*: the minimum allowed ternary tree leaf node size

In one example of the quadtree with nested multi-type tree coding tree structure, the CTU size is set as 128×128 luma samples with two corresponding 64×64 blocks of 4:2:0 chroma samples, the *MinQTSize* is set as 16×16, the *MaxBtSize* is set as 128×128and *MaxTtSize* is set as 64×64, the *MinBtSize* and *MinTtSize* (for both width and height) is set as 4×4, and the *MaxMttDepth* is set as 4. The quaternary tree partitioning is applied to the CTU first to generate quaternary tree leaf nodes. The quaternary tree leaf nodes may have a size from 16×16 (i.e., the *MinQTSize*) to 128×128 (i.e., the CTU size). If the leaf QT node is 128×128, it will not be further split by the binary tree since the size exceeds the *MaxBtSize* and *MaxTtSize* (i.e., 64×64). Otherwise, the leaf qdtree node could be further partitioned by the multi-type tree. Therefore, the quaternary tree leaf node is also the root node for the multi-type tree and it has multi-type tree depth (mttDepth) as 0. When the multi-type tree depth reaches *MaxMttDepth* (i.e., 4), no further splitting is considered. When the multi-type tree node has width equal to *MinBtSize* and smaller or equal to 2 \* *MinTtSize*, no further horizontal splitting is considered. Similarly, when the multi-type tree node has height equal to *MinBtSize and* smaller or equal to 2 \* *MinTtSize*, no further vertical splitting is considered.

To allow 64×64 Luma block and 32×32 Chroma pipelining design in VVC hardware decoders, TT split is forbidden when either width or height of a luma coding block is larger than 64 , as shown in Figure 9. TT split is also forbidden when either width or height of a chroma coding block is larger than 32.



**Figure 9 – No TT split for 128×128 coding block**

In VTM6, the coding tree scheme supports the ability for the luma and chroma to have a separate block tree structure. Currently, for P and B slices, the luma and chroma CTBs in one CTU have to share the same coding tree structure. However, for I slices, the luma and chroma can have separate block tree structures. When separate block tree mode is applied, luma CTB is partitioned into CUs by one coding tree structure, and the chroma CTBs are partitioned into chroma CUs by another coding tree structure. This means that a CU in an I slice may consist of a coding block of the luma component or coding blocks of two chroma components, and a CU in a P or B slice always consists of coding blocks of all three colour components unless the video is monochrome.

### CU splits on picture boundaries

As done in HEVC, when a portion of a tree node block exceeds the bottom or right picture boundary, the tree node block is forced to be split until the all samples of every coded CU are located inside the picture boundaries. The following splitting rules are applied in the VTM6:

– If a portion of a tree node block exceeds both the bottom and the right picture boundaries,

* + If the block is a QT node and the size of the block is larger than the minimum QT size, the block is forced to be split with QT split mode.
  + Otherwise, the block is forced to be split with SPLIT\_BT\_HOR mode

– Otherwise if a portion of a tree node block exceeds the bottom picture boundaries,

* + If the block is a QT node, and the size of the block is larger than the minimum QT size, and the size of the block is larger than the maximum BT size, the block is forced to be split with QT split mode.
  + Otherwise, if the block is a QT node, and the size of the block is larger than the minimum QT size and the size of the block is smaller than or equal to the maximum BT size, the block is forced to be split with QT split mode or SPLIT\_BT\_HOR mode.
  + Otherwise (the block is a BTT node or the size of the block is smaller than or equal to the minimum QT size), the block is forced to be split with SPLIT\_BT\_HOR mode.

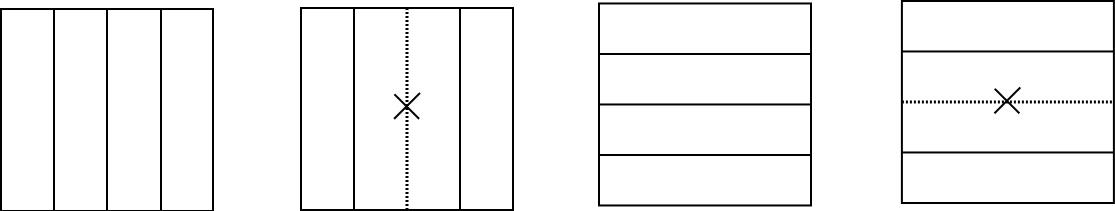
– Otherwise if a portion of a tree node block exceeds the right picture boundaries,

* + If the block is a QT node, and the size of the block is larger than the minimum QT size, and the size of the block is larger than the maximum BT size, the block is forced to be split with QT split mode.
  + Otherwise, if the block is a QT node, and the size of the block is larger than the minimum QT size and the size of the block is smaller than or equal to the maximum BT size, the block is forced to be split with QT split mode or SPLIT\_BT\_VER mode.
  + Otherwise (the block is a BTT node or the size of the block is smaller than or equal to the minimum QT size), the block is forced to be split with SPLIT\_BT\_VER mode.

### Restrictions on redundant CU splits

The quadtree with nested multi-type tree coding block structure provides a highly flexible block partitioning structure. Due to the types of splits supported the multi-type tree, different splitting patterns could potentially result in the same coding block structure. In VVC, some of these redundant splitting patterns are disallowed.

Figure 10 illustrates the redundant splitting patterns of binary tree splits and ternary tree splits. As shown in Figure 10, two levels of consecutive binary splits in one direction could have the same coding block structure as a ternary tree split followed by a binary tree split of the central partition. In this case, the binary tree split (in the given direction) for the central partition of a ternary tree split is prevented by the syntax. This restriction applies for CUs in all pictures.



**Figure 10–Redundant splitting patterns of binary tree split and ternary tree split cases**

When the splits are prohibited as described above, signalling of the corresponding syntax elements is modified to account for the prohibited cases. For example, when any case in Figure 10 is identified (i.e. the binary split is prohibited for a CU of a central partition), the syntax element mtt\_split\_cu\_binary\_flag which specifies whether the split is a binary split or a ternary split is not signalled and is instead inferred to be equal to 0 by the decoder.

### Virtual pipeline data units (VPDUs)

Virtual pipeline data units (VPDUs) are defined as non-overlapping units in a picture. In hardware decoders, successive VPDUs are processed by multiple pipeline stages at the same time. The VPDU size is roughly proportional to the buffer size in most pipeline stages, so it is important to keep the VPDU size small. In most hardware decoders, the VPDU size can be set to maximum transform block (TB) size. However, in VVC, ternary tree (TT) and binary tree (BT) partition may lead to the increasing of VPDUs size.

In order to keep the VPDU size as 64x64 luma samples, the following normative partition restrictions (with syntax signaling modification) are applied in VTM6, as shown in Figure 11:

* TT split is not allowed for a CU with either width or height, or both width and height equal to 128.
* For a 128xN CU with N ≤ 64 (i.e. width equal to 128 and height smaller than 128), horizontal BT is not allowed.
* For an Nx128 CU with N ≤ 64 (i.e. height equal to 128 and width smaller than 128), vertical BT is not allowed.



**Figure 11 – Examples of disallowed TT and BT partitioning in VTM6**

### Intra chroma partitioning and prediction restriction

Since the dual tree in intra picture allows to apply different partitioning in the chroma coding tree compared to the luma coding tree, the dual tree introduces longer coding pipeline and the QTBT MinQTSizeC value range and MinBtSizeY and MinTTSizeY in chroma tree allow small chroma blocks such as 2x2, 4x2, and 2x4. It provides difficulties in practical decoder design. Moreover, several prediction modes such as CCLM, planar and angular mode needs multiplication. In order to alleviate the above mentioned issues, small chroma block sizes (2x2/2x4/4x2) are restricted in dual tree as a partitioning restriction.

In typical hardware video encoders and decoders, processing throughput drops when a picture has more small intra blocks because of sample processing data dependency between neighbouring intra blocks. The predictor generation of an intra block requires top and left boundary reconstructed samples from neighbouring blocks. Therefore, intra prediction has to be sequentially processed block by block.

In HEVC, the smallest intra CU is 8x8 luma samples. The luma component of the smallest intra CU can be further split into four 4x4 luma intra prediction units (PUs), but the chroma components of the smallest intra CU cannot be further split. Therefore, the worst case hardware processing throughput occurs when 4x4 chroma intra blocks or 4x4 luma intra blocks are processed. In VVC, in order to improve worst case throughput, chroma intra CBs smaller than 16 chroma samples is disallowed by constraining the partitioning of chroma intra CBs. In single coding tree, a smallest chroma intra prediction unit (SCIPU) is defined as a coding tree node whose chroma block size is larger than or equal to 16 chroma samples and has at least one child luma block smaller than 64 luma samples. It is required that in each SCIPU, all CBs are inter, or all CBs are non-inter, i.e, either intra or intra block copy (IBC). In case of a non-inter SCIPU, it is further required that chroma of the non-inter SCIPU shall not be further split and luma of the SCIPU is allowed to be further split. In this way, the smallest chroma intra CB size is 16 chroma samples, and 2x2, 2x4, and 4x2 chroma CBs are removed. In addition, chroma scaling is not applied in case of a non-inter SCIPU. Here, no additional syntax is signalled, and whether a SCIPU is non-inter can be derived by the prediction mode of the first luma CB in the SCIPU. The type of a SCIPU is inferred to be non-inter if the current slice is an I-slice or the current SCIPU has a 4x4 luma partition in it after further split one time (because no inter 4x4 is allowed in VVC); otherwise, the type of the SCIPU (inter or non-inter) is indicated by one flag before parsing the CUs in the SCIPU. In addition, a restriction on picture size is considered to avoid 2x2/2x4/4x2 intra chroma blocks at the corner of pictures by considering the picture width and height to be multiple of max (8, MinCbSizeY).

## Intra prediction

### Intra mode coding with 67 intra prediction modes

To capture the arbitrary edge directions presented in natural video, the number of directional intra modes in VTM6 is extended from 33, as used in HEVC, to 65. The new directional modes not in HEVC are depicted as red dotted arrows in Figure 12, and the planar and DC modes remain the same. These denser directional intra prediction modes apply for all block sizes and for both luma and chroma intra predictions.

In VTM6, several conventional angular intra prediction modes are adaptively replaced with wide-angle intra prediction modes for the non-square blocks. Wide angle intra prediction is described in 3.3.1.2.

In HEVC, every intra-coded block has a square shape and the length of each of its side is a power of 2. Thus, no division operations are required to generate an intra-predictor using DC mode. In VTM5, blocks can have a rectangular shape that necessitates the use of a division operation per block in the general case. To avoid division operations for DC prediction, only the longer side is used to compute the average for non-square blocks.

#### Intra mode coding



Figure 12 – 67 intra prediction modes

To keep the complexity of the most probable mode (MPM) list generation low, an intra mode coding method with 6 MPMs is used by considering two available neighboring intra modes. The following three aspects are considered to construct the MPM list:

* + - Default intra modes
    - Neighbouring intra modes
    - Derived intra modes

A unified 6-MPM list is used for intra blocks irrespective of whether MRL and ISP coding tools are applied or not. The MPM list is constructed based on intra modes of the left and above neighboring block. Suppose the mode of the left is denoted as *Left* and the mode of the above block is denoted as *Above*, the unified MPM list is constructed as follows:

* When a neighboring block is not available, its intra mode is set to Planar by default.
* If both modes *Left* and *Above* are non-angular modes:
  + MPM list 🡪 {Planar, DC, V, H, V-4, V+4}
* If one of modes *Left* and *Above* is angular mode, and the other is non-angular:
  + Set a mode *Max* as the larger mode in *Left* and *Above*
  + MPM list 🡪 {Planar, *Max*, DC, *Max -1, Max +1*, *Max -2*}
* If *Left* and *Above* are both angular and they are different:
  + Set a mode *Max* as the larger mode in *Left* and *Above*
  + if the difference of mode *Left* and *Above* is in the range of 2 to 62, inclusive
    - MPM list 🡪 {Planar, *Left*, *Above*, DC, *Max -1*, *Max +1*}
  + Otherwise
    - MPM list 🡪 {Planar, *Left*, *Above*, DC, *Max -2, Max +2*}
* If *Left* and *Above* are both angular and they are the same:
  + MPM list 🡪 {Planar, *Left*, *Left -1, Left +1*, DC, *Left -2*}

Besides, the first bin of the mpm index codeword is CABAC context coded. In total three contexts are used, corresponding to whether the current intra block is MRL enabled, ISP enabled, or a normal intra block.

During 6 MPM list generation process, pruning is used to remove duplicated modes so that only unique modes can be included into the MPM list. For entropy coding of the 61 non-MPM modes, a Truncated Binary Code (TBC) is used.

#### Wide-angle intra prediction for non-square blocks

Conventional angular intra prediction directions are defined from 45 degrees to -135 degrees in clockwise direction. In VTM6, several conventional angular intra prediction modes are adaptively replaced with wide-angle intra prediction modes for non-square blocks. The replaced modes are signalled using the original mode indexes, which are remapped to the indexes of wide angular modes after parsing. The total number of intra prediction modes is unchanged, i.e., 67, and the intra mode coding method is unchanged.

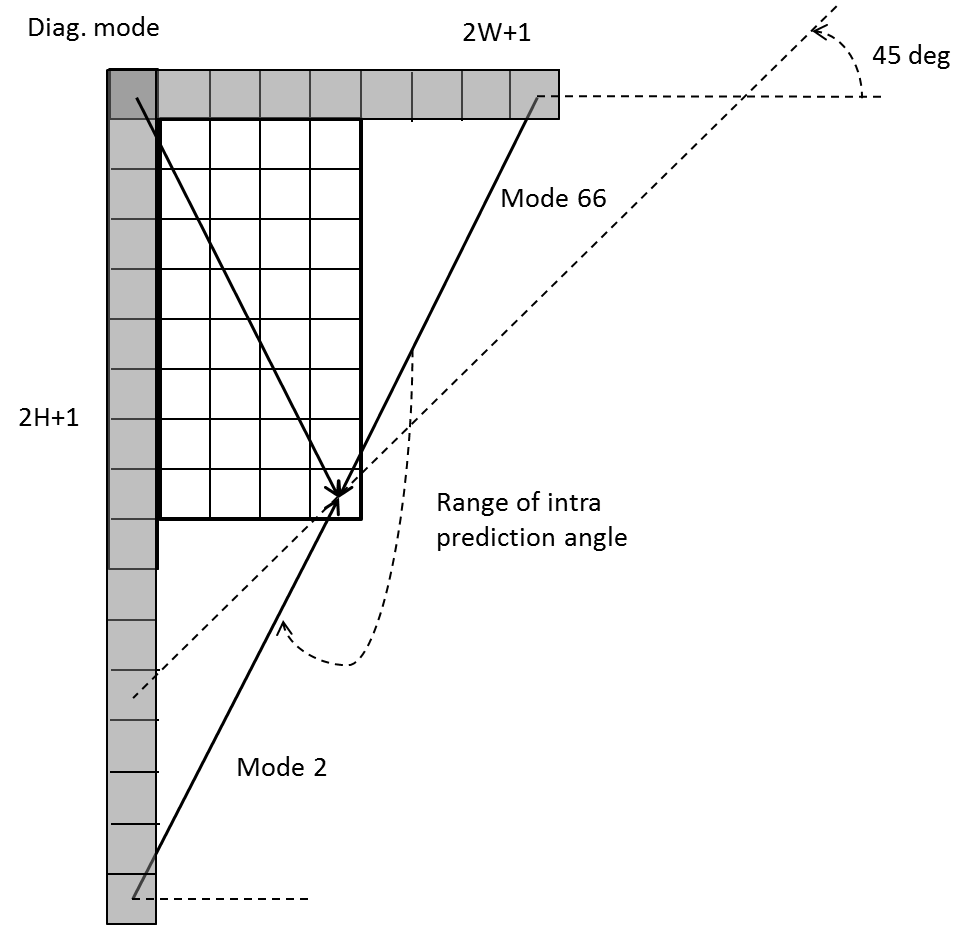
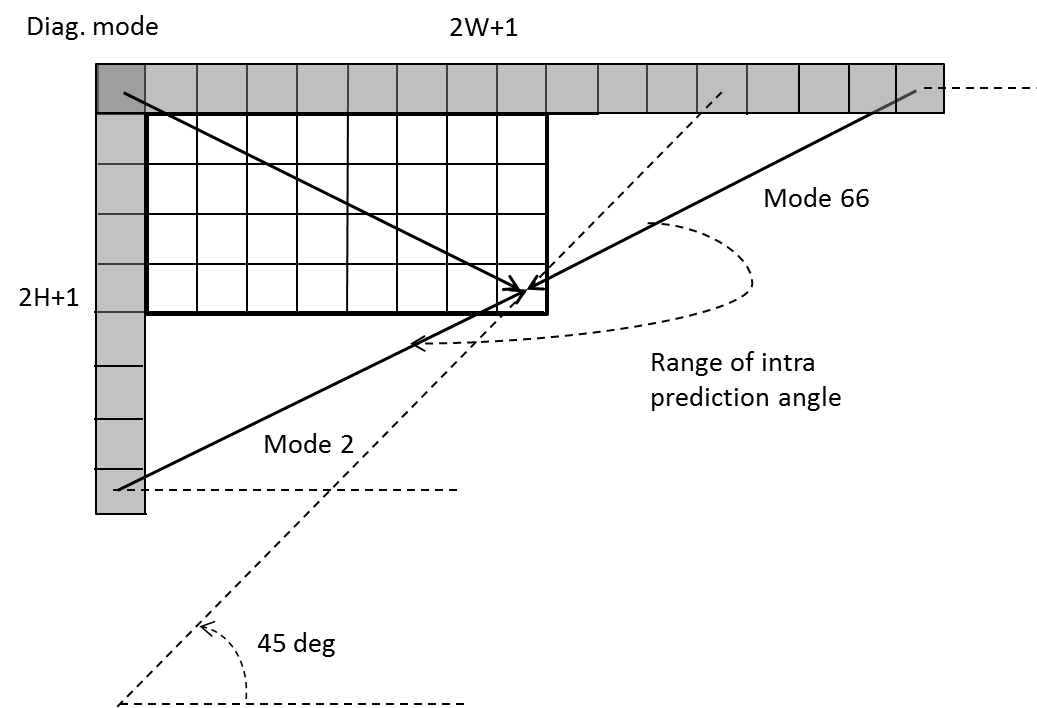


Figure 13 – Reference samples for wide-angular intra prediction

To support these prediction directions, the top reference with length 2W+1, and the left reference with length 2H+1, are defined as shown in Figure 13.

The number of replaced modes in wide-angular direction mode depends on the aspect ratio of a block. The replaced intra prediction modes are illustrated in Table 3‑2

Table 3‑2 - Intra prediction modes replaced by wide-angular modes

|  |  |
| --- | --- |
| Aspect ratio | Replaced intra prediction modes |
| W / H == 16 | Modes 12, 13,14,15 |
| W / H == 8 | Modes 12, 13 |
| W / H == 4 | Modes 2,3,4,5,6,7,8,9,10,11 |
| W / H == 2 | Modes 2,3,4,5,6,7, |
| W / H == 1 | None |
| W / H == 1/2 | Modes 61,62,63,64,65,66 |
| W / H == 1/4 | Mode 57,58,59,60,61,62,63,64,65,66 |
| W / H == 1/8 | Modes 55, 56 |
| W / H == 1/16 | Modes 53, 54, 55, 56 |



Figure 14 - Problem of discontinuity in case of directions beyond 45°

As shown in Figure 14, two vertically-adjacent predicted samples may use two non-adjacent reference samples in the case of wide-angle intra prediction. Hence, low-pass reference samples filter and side smoothing are applied to the wide-angle prediction to reduce the negative effect of the increased gap ∆pα. If a wide-angle mode represents a non-fractional offset. There are 8 modes in the wide-angle modes satisfy this condition, which are [-14, -12, -10, -6, 72, 76, 78, 80]. When a block is predicted by these modes, the samples in the reference buffer are directly copied without applying any interpolation. With this modification, the number of samples needed to be smoothing is reduced. Besides, it aligns the design of non-fractional modes in the conventional prediction modes and wide-angle modes.

In VVC, 4:2:2 and 4:4:4 chroma formats are supported as well as 4:2:0. Chroma derived mode (DM) derivation table for 4:2:2 chroma format was initially ported from HEVC extending the number of entries from 35 to 67 to align with the extension of intra prediction modes. Since HEVC specification does not support prediction angle below -135 degree and above 45 degree, luma intra prediction modes ranging from 2 to 5 are mapped to 2. Therefore chroma DM derivation table for 4:2:2: chroma format is updated by replacing some values of the entries of the mapping table to convert prediction angle more precisely for chroma blocks.

#### Mode Dependent Intra Smoothing (MDIS)

Four-tap intra interpolation filters are utilized to improve the directional intra prediction accuracy. In HEVC, a two-tap linear interpolation filter has been used to generate the intra prediction block in the directional prediction modes (i.e., excluding Planar and DC predictors). In the VTM6, simplified 6-bit 4-tap Gaussian interpolation filter is used for only directional intra modes. Non-directional intra prediction process is unmodified. The selection of the 4-tap filters is performed according to the MDIS condition for directional intra prediction modes that provide non-fractional displacements, i.e. to all the directional modes excluding the following: 2, HOR\_IDX, DIA\_IDX, VER\_IDX, 66.

Depending on the intra prediction mode, the following reference samples processing is performed:

1. The directional intra-prediction mode is classified into one of the following groups:
   * vertical or horizontal modes (HOR\_IDX, VER\_IDX),
   * diagonal modes that represent angles which are multiple of 45 degree (2, DIA\_IDX, VDIA\_IDX),
   * remaining directional modes;
2. If the directional intra-prediction mode is classified as belonging to group A, then then no filters are applied to reference samples to generate predicted samples;
3. Otherwise, if a mode falls into group B, then a [1, 2, 1] reference sample filter may be applied (depending on the MDIS condition) to reference samples to further copy these filtered values into an intra predictor according to the selected direction, but no interpolation filters are applied;
4. Otherwise, if a mode is classified as belonging to group C, then only an intra reference sample interpolation filter is applied to reference samples to generate a predicted sample that falls into a fractional or integer position between reference samples according to a selected direction (no reference sample filtering is performed).

### Cross-component linear model prediction

To reduce the cross-component redundancy, a cross-component linear model (CCLM) prediction mode is used in the VTM6, for which the chroma samples are predicted based on the reconstructed luma samples of the same CU by using a linear model as follows:

(3-1)

where represents the predicted chroma samples in a CU and represents the downsampled reconstructed luma samples of the same CU.

The CCLM parameters ( and ) are derived with at most four neighbouring chroma samples and their corresponding down-sampled luma samples. Suppose the current chroma block dimensions are W×H, then W’ and H’ are set as

* W’=W, H’=H when LM mode is applied;
* W’=W+H when LM-A mode is applied;
* H’=H+W when LM-L mode is applied;

The above neighbouring positions are denoted as S[0, -1]…S[W’-1, -1] and the the left neighbouring positions are denoted as S[-1, 0]…S[-1, H’-1]. Then the four samples are selected as

* S[W’/4, -1], S[3W’/4, -1], S[-1, H’/4], S[-1, 3H’/4] when LM mode is applied and both above and left neighbouring samples are available;
* S[W’/8, -1], S[3W’/8, -1], S[5W’/8, -1], S[7W’/8, -1] when LM-A mode is applied or only the above neighbouring samples are available;
* S[-1, H’/8], S[-1, 3H’/8], S[-1, 5H’/8], S[-1, 7H’/8] when LM-L mode is applied or only the left neighbouring samples are available;

The four neighbouring luma samples at the selected positions are down-sampled and compared four times to find two smaller values: *x*0*A* and *x*1*A*, and two larger values: *x*0*B* and *x*1*B*. Their corresponding chroma sample values are denoted as *y*0*A*, *y*1*A*, *y*0*B* and *y*1*B*. Then *xA*, *xB*, *yA* and *yB* are derived as:

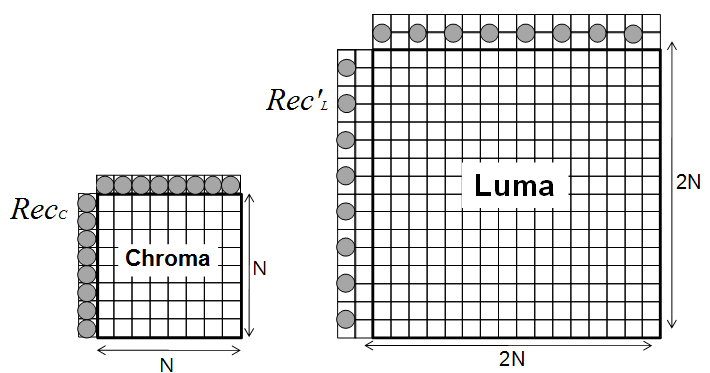
*Xa*=(*x*0*A* + *x*1*A* +1)>>1; *Xb*=(*x*0*B* + *x*1*B* +1)>>1; *Ya*=(*y*0*A* + *y*1*A* +1)>>1; *Yb*=(*y*0*B* + *y*1*B* +1)>>1 (3-2)

Finally, the linear model parameters and are obtained according to the following equations.

(3-3)

(3-4)

Figure 15 shows an example of the location of the left and above samples and the sample of the current block involved in the CCLM mode.



**Figure 15 - Locations of the samples used for the derivation of α and β**

The division operation to calculate parameter is implemented with a look-up table. To reduce the memory required for storing the table, the *diff* value (difference between maximum and minimum values) and the parameter are expressed by an exponential notation. For example, *diff* is approximated with a 4-bit significant part and an exponent. Consequently, the table for 1/diff is reduced into 16 elements for 16 values of the significand as follows:

DivTable [ ] = { 0, 7, 6, 5, 5, 4, 4, 3, 3, 2, 2, 1, 1, 1, 1, 0 } (3-5)

This would have a benefit of both reducing the complexity of the calculation as well as the memory size required for storing the needed tables

Besides the above template and left template can be used to calculate the linear model coefficients together, they also can be used alternatively in the other 2 LM modes, called LM\_A, and LM\_L modes.

In LM\_A mode, only the above template are used to calculate the linear model coefficients. To get more samples, the above template are extended to (W+H). In LM\_L mode, only left template are used to calculate the linear model coefficients. To get more samples, the left template are extended to (H+W).

For a non-square block, the above template are extended to W+W, the left template are extended to H+H.

To match the chroma sample locationsfor 4:2:0 video sequences, two types of downsampling filter are applied to luma samples to achieve 2 to 1 downsampling ratio in both horizontal and vertical directions. The selection of downsampling filter is specified by a SPS level flag. The two downsmapling filters are as follows, which are corresponding to “type-0” and “type-2” content, respectively.

(3-6)

(3-7)

Note that only one luma line (general line buffer in intra prediction) is used to make the downsampled luma samples when the upper reference line is at the CTU boundary.

This parameter computation is performed as part of the decoding process, and is not just as an encoder search operation. As a result, no syntax is used to convey the α and β values to the decoder.

For chroma intra mode coding, a total of 8 intra modes are allowed for chroma intra mode coding. Those modes include five traditional intra modes and three cross-component linear model modes (CCLM, LM\_A, and LM\_L). Chroma mode signalling and derivation process are shown in Table 3‑3. Chroma mode coding directly depends on the intra prediction mode of the corresponding luma block. Since separate block partitioning structure for luma and chroma components is enabled in I slices, one chroma block may correspond to multiple luma blocks. Therefore, for Chroma DM mode, the intra prediction mode of the corresponding luma block covering the center position of the current chroma block is directly inherited.

Table 3‑3 – Derivation of chroma prediction mode from luma mode when cclm\_is enabled

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Chroma prediction mode | Corresponding luma intra prediction mode | | | | |
| 0 | 50 | 18 | 1 | X ( 0  <=  X  <=  66 ) |
| 0 | 66 | 0 | 0 | 0 | 0 |
| 1 | 50 | 66 | 50 | 50 | 50 |
| 2 | 18 | 18 | 66 | 18 | 18 |
| 3 | 1 | 1 | 1 | 66 | 1 |
| 4 | 0 | 50 | 18 | 1 | X |
| 5 | 81 | 81 | 81 | 81 | 81 |
| 6 | 82 | 82 | 82 | 82 | 82 |
| 7 | 83 | 83 | 83 | 83 | 83 |

A single binarization table is used regardless of the value of sps\_cclm\_enabled\_flag as shown in Table 3‑4.

Table 3‑4– Unified binarization table for chroma prediction mode

|  |  |
| --- | --- |
| **Value of intra\_chroma\_pred\_mode** | **Bin string** |
| 4 | 00 |
| 0 | 0100 |
| 1 | 0101 |
| 2 | 0110 |
| 3 | 0111 |
| 5 | 10 |
| 6 | 110 |
| 7 | 111 |

In Table 3‑4, the first bin indicates whether it is regular (0) or LM modes (1). If it is LM mode, then the next bin indicates whether it is LM\_CHROMA (0) or not. If it is not LM\_CHROMA, next 1 bin indicates whether it is LM\_L (0) or LM\_A (1). For this case, when sps\_cclm\_enabled\_flag is 0, the first bin of the binarization table for the corresponding intra\_chroma\_pred\_mode can be discarded prior to the entropy coding. Or, in other words, the first bin is inferred to be 0 and hence not coded. This single binarization table is used for both sps\_cclm\_enabled\_flag equal to 0 and 1 cases. The first two bins in Table 3‑4are context coded with its own context model, and the rest bins are bypass coded.

In addition, in order to reduce luma-chroma latency in dual tree, when the 64x64 luma coding tree node is partitioned with Not Split (and ISP is not used for the 64x64 CU) or QT, the chroma CUs in 32x32 / 32x16 chroma coding tree node are allowed to use CCLM in the following way:

* If the 32x32 chroma node is not split or partitioned QT split, all chroma CUs in the 32x32 node can use CCLM
* If the 32x32 chroma node is partitioned with Horizontal BT, and the 32x16 child node does not split or uses Vertical BT split, all chroma CUs in the 32x16 chroma node can use CCLM.

In all the other luma and chroma coding tree split conditions, CCLM is not allowed for chroma CU.

### Position dependent intra prediction combination

In the VTM6, the results of intra prediction of planar mode are further modified by a position dependent intra prediction combination (PDPC) method. PDPC is an intra prediction method which invokes a combination of the un-filtered boundary reference samples and HEVC style intra prediction with filtered boundary reference samples. PDPC is applied to the following intra modes without signalling: planar, DC, horizontal, vertical, bottom-left angular mode and its *eight* adjacent angular modes, and top-right angular mode and its *eight* adjacent angular modes.

The prediction sample *pred*(*x*,*y*) is predicted using an intra prediction mode (DC, planar, angular) and a linear combination of reference samples according to the Equation 3-8 as follows:

*pred*(*x*,*y*)=(*wL*×*R*-1*,y* + *wT*×*Rx,*-1 – *wTL* ×*R*-1*,*-1+(64 – *wL* – *wT*+*wTL*)×*pred*(*x*,*y*) + 32 )>>6 (3-8)

where *Rx,*-1, *R*-1*,y* represent the reference samples located at the top and left of current sample (*x*, *y*), respectively, and *R*-1*,*-1 represents the reference sample located at the top-left corner of the current block.

If PDPC is applied to DC, planar, horizontal, and vertical intra modes, additional boundary filters are not needed, as required in the case of HEVC DC mode boundary filter or horizontal/vertical mode edge filters. PDPC process for DC and Planar modes is identical and clipping operation is avoided. For angular modes, pdpc scale factor is adjusted such that range check is not needed and condition on angle to enable pdpc is removed (scale >=0 is used). In addition, PDPC weight is based on 32 in all angular mode cases.

Figure 16 illustrates the definition of reference samples (*Rx,*-1, *R*-1*,y* and *R*-1*,*-1) for PDPC applied over various prediction modes. The prediction sample *pred* (*x’*, *y’*) is located at (*x’*, *y’*) within the prediction block. As an example, the coordinate *x* of the reference sample *Rx,*-1 is given by: *x* = *x’* + *y’* + 1, and the coordinate *y* of the reference sample *R*-1*,y* is similarly given by: *y* = *x’* + *y’* + 1 for the diagonal modes. For the other annular mode, the reference samples *Rx,*-1 and*R*-1*,y* could be located in fractional sample position. In this case, the sample value of the nearest integer sample location is used.

|  |  |
| --- | --- |
| 1. Diagonal top-right mode | 1. Diagonal bottom-left mode |
| (c) Adjacent diagonal top-right mode | 1. Adjacent diagonal bottom-left mode |

**Figure 16 - Definition of samples used by PDPC applied to diagonal and adjacent angular intra modes.**

The PDPC weights are dependent on prediction modes and are shown in 4.

**Table 3‑5 - Example of PDPC weights according to prediction modes**

|  |  |  |  |
| --- | --- | --- | --- |
| Prediction modes | wT | wL | wTL |
| Diagonal top-right | 16 >> ( ( *y’*<<1 ) >> *shift*) | 16 >> ( ( *x’*<<1 ) >> *shift*) | 0 |
| Diagonal bottom-left | 16 >> ( ( *y’*<<1 ) >> *shift* ) | 16 >> ( ( *x’*<<1 ) >> *shift* ) | 0 |
| Adjacent diagonal top-right | 32 >> ( ( *y’*<<1 ) >> *shift* ) | 0 | 0 |
| Adjacent diagonal bottom-left | 0 | 32 >> ( ( *x’*<<1 ) >> *shift* ) | 0 |

### Multiple reference line (MRL) intra prediction

Multiple reference line (MRL) intra prediction uses more reference lines for intra prediction. In Figure 17, an example of 4 reference lines is depicted, where the samples of segments A and F are not fetched from reconstructed neighbouring samples but padded with the closest samples from Segment B and E, respectively. HEVC intra-picture prediction uses the nearest reference line (i.e., reference line 0). In MRL, 2 additional lines (reference line 1 and reference line 3) are used.

The index of selected reference line (mrl\_idx) is signalled and used to generate intra predictor. For reference line idx, which is greater than 0, only include additional reference line modes in MPM list and only signal mpm index without remaining mode. The reference line index is signalled before intra prediction modes, and Planar and DC modes are excluded from intra prediction modes in case a nonzero reference line index is signalled.



**Figure 17 Example of four reference lines neighboring to a prediction block**

MRL is disabled for the first line of blocks inside a CTU to prevent using extended reference samples outside the current CTU line. Also, PDPC is disabled when additional line is used. For MRL mode, the derivation of DC value in DC intra prediction mode for non-zero reference line indices is aligned with that of reference line index 0.

### Intra Sub-Partitions (ISP)

The Intra Sub-Partitions (ISP) tool divides luma intra-predicted blocks vertically or horizontally into 2 or 4 sub-partitions depending on the block size. For example, minimum block size for ISP is 4x8 (or 8x4). If block size is greater than 4x8 (or 8x4) then the corresponding block is divided by 4 sub-partitions. It has been noted that the (with ) and (with ) ISP blocks could generate a potential issue with the VDPU. For example, an CU in the single tree case has an luma TB and two corresponding chroma TBs. If the CU uses ISP, then the luma TB will be divided into four TBs (only the horizontal split is possible), each of them smaller than a block. However, in the current design of ISP chroma blocks are not divided. Therefore, both chroma components will have a size greater than a block. Analogously, a similar situation could be created with a CU using ISP. Hence, these two cases are an issue for the decoder pipeline. For this reason, the CU sizes that can use ISP is restricted to a maximum of . Figure 18 shows examples of the two possibilities. All sub-partitions fulfill the condition of having at least 16 samples.

In ISP, the dependence of 1xN/2xN subblock prediction on the reconstructed values of previously decoded 1xN/2xN subblocks of the coding block is not allowed so that the minimum width of prediction for subblocks becomes four samples. For example, an 8xN (N > 4) coding block that is coded using ISP with vertical split is split into two prediction regions each of size 4xN and four transforms of size 2xN. Also, a 4xN coding block that is coded using ISP with vertical split is predicted using the full 4xN block; four transform each of 1xN is used. Although the transform sizes of 1xN and 2xN are allowed, it is asserted that the transform of these blocks in 4xN regions can be performed in parallel. For example, when a 4xN prediction region contains four 1xN transforms, there is no transform in the horizontal direction; the transform in the vertical direction can be performed as a single 4xN transform in the vertical direction. Similarly, when a 4xN prediction region contains two 2xN transform blocks, the transform operation of the two 2xN blocks in each direction (horizontal and vertical) can be conducted in parallel. Thus, there is no delay added in processing these smaller blocks than processing 4x4 regular-coded intra blocks.



a) Examples of sub-partitions for 4x8 and 8x4 CUs



b) Examples of sub-partitions for CUs other than 4x8, 8x4 and 4x4

**Figure 18** ‑ **Sub-partition depending on the block size**

**Table 3‑6 – Entropy coding coefficient group size**

|  |  |
| --- | --- |
| Block Size | Coefficient group Size |
|  |  |
|  |  |
|  |  |
|  |  |
| All other possible cases |  |

For each sub-partition, reconstructed samples are obtained by adding the residual signal to the prediction signal. Here, a residual signal is generated by the processes such as entropy decoding, inverse quantization and inverse transform. Therefore, the reconstructed sample values of each sub-partition are available to generate the prediction of the next sub-partition, and each sub-partition is processed repeatedly. In addition, the first sub-partition to be processed is the one containing the top-left sample of the CU and then continuing downwards (horizontal split) or rightwards (vertical split). As a result, reference samples used to generate the sub-partitions prediction signals are only located at the left and above sides of the lines. All sub-partitions share the same intra mode. The followings are summary of interaction of ISP with other coding tools.

* Multiple Reference Line (MRL): if a block has an MRL index other than 0, then the ISP coding mode will be inferred to be 0 and therefore ISP mode information will not be sent to the decoder.
* Entropy coding coefficient group size: the sizes of the entropy coding subblocks have been modified so that they have 16 samples in all possible cases, as shown in Table 3‑6. Note that the new sizes only affect blocks produced by ISP in which one of the dimensions is less than 4 samples. In all other cases coefficient groups keep the dimensions.
* CBF coding: it is assumed to have at least one of the sub-partitions has a non-zero CBF. Hence, if is the number of sub-partitions and the first sub-partitions have produced a zero CBF, then the CBF of the -th sub-partition is inferred to be 1.
* MPM usage: the MPM flag will be inferred to be one in a block coded by ISP mode, and the MPM list is modified to exclude the DC mode and to prioritize horizontal intra modes for the ISP horizontal split and vertical intra modes for the vertical one.
* Transform size restriction: all ISP transforms with a length larger than 16 points uses the DCT-II.
* PDPC: when a CU uses the ISP coding mode, the PDPC filters will not be applied to the resulting sub-partitions.
* MTS flag: if a CU uses the ISP coding mode, the MTS CU flag will be set to 0 and it will not be sent to the decoder. Therefore, the encoder will not perform RD tests for the different available transforms for each resulting sub-partition. The transform choice for the ISP mode will instead be fixed and selected according the intra mode, the processing order and the block size utilized. Hence, no signalling is required. For example, let and be the horizontal and the vertical transforms selected respectively for the sub-partition, where is the width and is the height. Then the transform is selected according to the following rules:
* If or , then there is no horizontal or vertical transform respectively.
* If or , = DCT-II
* If or , = DCT-II
* Otherwise, the transform is selected as in Table 3‑7.

Table 3‑7 – Transform selection depends on intra mode

|  |  |  |
| --- | --- | --- |
| **Intra mode** |  |  |
| Planar  Ang. 31,32,34,36,37 | DST-VII | DST-VII |
| DC  Ang. 33, 35 | DCT-II | DCT-II |
| Ang. 2, 4, 6…28,30  Ang. 39,41,43…63,65 | DST-VII | DCT-II |
| Ang. 3,5,7…27,29  Ang. 38,40,42…64,66 | DCT-II | DST-VII |

In ISP mode, all 67 intra modes are allowed. PDPC is also applied if corresponding width and height is at least 4 samples long. In addition, the condition for intra interpolation filter selection doesn’t exist anymore, and Cubic (DCT-IF) filter is always applied for fractional position interpolation in ISP mode.

### Matrix weighted Intra Prediction (MIP)

Matrix weighted intra prediction (MIP) method is a newly added intra prediction technique into VTM6. For predicting the samples of a rectangular block of width and height , matrix weighted intra prediction (MIP) takes one line of H reconstructed neighbouring boundary samples left of the block and one line of reconstructed neighbouring boundary samples above the block as input. If the reconstructed samples are unavailable, they are generated as it is done in the conventional intra prediction. The generation of the prediction signal is based on the following three steps, which are averaging, matrix vector multiplication and linear interpolation as shown in Figure 16.

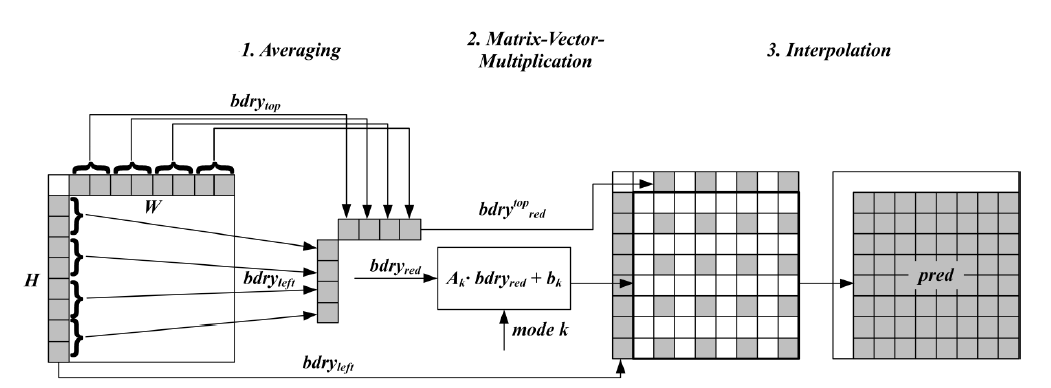


Figure 19 – Matrix weighted intra prediction process

**3.3.6.1 Averaging neighboring samples**

Out of the boundary samples, four samples in the case of and eight samples in all other cases are extracted by averaging. Specifically, the input boundaries and are reduced to smaller boundaries and by averaging neighboring boundary samples according to predefined rule depends on block size. Then, the two reduced boundaries and are concatenated to a reduced boundary vector which is thus of size four for blocks of shape and of size eight for blocks of all other shapes. If refers to the MIP-mode, this concatenation is defined as follows:

(3-9)

**3.3.6.2 Matrix Multiplication**

A matrix vector multiplication, followed by addition of an offset, is carried out with the averaged samples as an input. The result is a reduced prediction signal on a subsampled set of samples in the original block. Out of the reduced input vector a reduced prediction signal which is a signal on the downsampled block of width and height is generated. Here, and are defined as:

(3-10)

(3-11)

The reduced prediction signal is computed by calculating a matrix vector product and adding an offset:

.

Here, is a matrix that has rows and 4 columns if and 8 columns in all other cases. is a vector of size . The matrix and the offset vector are taken from one of the sets , , One defines an index as follows:

(3-12)

Here, each coefficient of the matrix A is represented with 8 bit precision.

**3.3.6.3 Interpolation**

The prediction signal at the remaining positions is generated from the prediction signal on the subsampled set by linear interpolation which is a single step linear interpolation in each direction. The matrices and offset vectors needed to generate the prediction signal are taken from three sets , of matrices. The set consists of 18 matrices each of which has 16 rows and 4 columns and 18 offset vectors each of size 16. Matrices and offset vectors of that set are used for blocks of size The set consists of 10 matrices , each of which has rows and 8 columns and 10 offset vectors each of size 16. Matrices and offset vectors of that set are used for blocks of sizes , and . Finally, the set consists of 6 matrices , each of which has 64 rows and 8 columns and of 6 offset vectors of size 64. Matrices and offset vectors of that set or parts of these matrices and offset vectors are used for all other block-shapes.

**3.3.6.4 Signaling of MIP mode and harmonization with other coding tools**

For each Coding Unit (CU) in intra mode, a flag indicating if an MIP mode is to be applied on the corresponding Prediction Unit (PU) or not is sent. If an MIP mode is to be applied, the MIP modes are directly coded using truncated binary code.

The number of supported MIP modes depends on block size. For example, 35 modes are available for blocks where . And 19 and 11 modes are used for and , respectively. In addition, two modes share the same matrix and offset vector to reduce the memory requirement as follows:

(3-13)

MIP coding mode is harmonized with other coding tools by considering following aspects:

* LFNST is enabled for MIP on large blocks. Here, the LFNST transforms corresponding to the planar mode are used.
* The reference sample derivation for MIP is performed exactly as for the conventional intra prediction modes.
* For the upsampling step used in the MIP-prediction, original reference samples are used instead of downsampled ones.
* Clipping is performed before upsampling and not after upsampling

## Inter prediction

For each inter-predicted CU, motion parameters consisting of motion vectors, reference picture indices and reference picture list usage index, and additional information needed for the new coding feature of VVC to be used for inter-predicted sample generation. The motion parameter can be signalled in an explicit or implicit manner. When a CU is coded with skip mode, the CU is associated with one PU and has no significant residual coefficients, no coded motion vector delta or reference picture index. A merge mode is specified whereby the motion parameters for the current CU are obtained from neighbouring CUs, including spatial and temporal candidates, and additional schedules introduced in VVC. The merge mode can be applied to any inter-predicted CU, not only for skip mode. The alternative to merge mode is the explicit transmission of motion parameters, where motion vector, corresponding reference picture index for each reference picture list and reference picture list usage flag and other needed information are signalled explicitly per each CU.

Beyond the inter coding features in HEVC, the VTM6 includes a number of new and refined inter prediction coding tools listed as follows:

* Extended merge prediction
* Merge mode with MVD (MMVD)
* AMVP mode with symmetric MVD signalling
* Affine motion compensated prediction
* Subblock-based temporal motion vector prediction (SbTMVP)
* Adaptive motion vector resolution (AMVR)
* Motion field storage: 1/16th luma sample MV storage and 8x8 motion field compression
* Bi-prediction with CU-level weight (BCW)
* Bi-directional optical flow (BDOF)
* Decoder side motion vector refinement (DMVR)
* Triangle partition prediction
* Combined inter and intra prediction (CIIP)

The following text provides the details on the inter prediction methods specified in VVC.

### Extended merge prediction

In VTM6, the merge candidate list is constructed by including the following five types of candidates in order:

1. Spatial MVP from spatial neighbour CUs
2. Temporal MVP from collocated CUs
3. History-based MVP from an FIFO table
4. Pairwise average MVP
5. Zero MVs.

The size of merge list is signalled in slice header and the maximum allowed size of merge list is 6 in VTM6. For each CU code in merge mode, an index of best merge candidate is encoded using truncated unary binarization (TU). The first bin of the merge index is coded with context and bypass coding is used for other bins.

The generation process of each category of merge candidates is provided in this session.

#### Spatial candidates derivation

The derivation of spatial merge candidates in VVC is same to that in HEVC. A maximum of four merge candidates are selected among candidates located in the positions depicted in Figure 20. The order of derivation is A0, B0, B1, A1 and B2. Position B2 is considered onlywhen any CU of position A0, B0, B1, A1 is not available (e.g. because it belongs to another slice or tile) or is intra coded. After candidate at position A1 is added, the addition of the remaining candidates is subject to a redundancy check which ensures that candidates with same motion information are excluded from the list so that coding efficiency is improved. To reduce computational complexity, not all possible candidate pairs are considered in the mentioned redundancy check. Instead only the pairs linked with an arrow in Figure 21 are considered and a candidate is only added to the list if the corresponding candidate used for redundancy check has not the same motion information.



Figure 20– Positions of spatial merge candidate



Figure 21 – Candidate pairs considered for redundancy check of spatial merge candidates

#### Temporal candidates derivation

In this step, only one candidate is added to the list. Particularly, in the derivation of this temporal merge candidate, a scaled motion vector is derived based on co-located CU belonging to the collocated referenncee picture. The reference picture list to be used for derivation of the co-located CU is explicitly signalled in the slice header. The scaled motion vector for temporal merge candidate is obtained as illustrated by the dotted line in Figure 22, which is scaled from the motion vector of the co-located CU using the POC distances, tb and td, where tb is defined to be the POC difference between the reference picture of the current picture and the current picture and td is defined to be the POC difference between the reference picture of the co-located picture and the co-located picture. The reference picture index of temporal merge candidate is set equal to zero.

curr\_pic

col\_pic

col\_ref

curr\_ref

td

tb

curr\_CU

col\_CU

Figure 22 – Illustration of motion vector scaling for temporal merge candidate

The position for the temporal candidate is selected between candidates C0 and C1, as depicted in Figure 23. If CU at position C0 is not available, is intra coded, or is outside of the current row of CTUs, position C1 is used. Otherwise, position C0 is used in the derivation of the temporal merge candidate.



Figure 23 – Candidate positions for temporal merge candidate, C0 and C1

#### History-based merge candidates derivation

The history-based MVP (HMVP) merge candidates are added to merge list after the spatial MVP and TMVP. In this method, the motion information of a previously coded block is stored in a table and used as MVP for the current CU. The table with multiple HMVP candidates is maintained during the encoding/decoding process. The table is reset (emptied) when a new CTU row is encountered. Whenever there is a non-subblock inter-coded CU, the associated motion information is added to the last entry of the table as a new HMVP candidate.

In VTM6 the HMVP table size *S* is set to be 6, which indicates up to 6 History-based MVP (HMVP) candidates may be added to the table. When inserting a new motion candidate to the table, a constrained first-in-first-out (FIFO) rule is utilized wherein redundancy check is firstly applied to find whether there is an identical HMVP in the table. If found, the identical HMVP is removed from the table and all the HMVP candidates afterwards are moved forward,

HMVP candidates could be used in the merge candidate list construction process. The latest severalHMVP candidates in the table are checked in order and inserted to the candidate list after the TMVP candidate. Redundancy check is applied on the HMVP candidates to the spatial or temporal merge candidate.

To reduce the number of redundancy check operations, the following simplifications are introduced:

1. Number of HMPV candidates is used for merge list generation is set as (*N* <= 4 ) ? *M*: (8 – *N*), wherein N indicates number of existing candidates in the merge list and M indicates number of available HMVP candidates in the table.
2. Once the total number of available merge candidates reaches the maximally allowed merge candidates minus 1, the merge candidate list construction process from HMVP is terminated.

#### Pair-wise average merge candidates derivation

Pairwise average candidates are generated by averaging predefined pairs of candidates in the existing merge candidate list, and the predefined pairs are defined as {(0, 1), (0, 2), (1, 2), (0, 3), (1, 3), (2, 3)}, where the numbers denote the merge indices to the merge candidate list. The averaged motion vectors are calculated separately for each reference list. If both motion vectors are available in one list, these two motion vectors are averaged even when they point to different reference pictures; if only one motion vector is available, use the one directly; if no motion vector is available, keep this list invalid.

When the merge list is not full after pair-wise average merge candidates are added, the zero MVPs are inserted in the end until the maximum merge candidate number is encountered.

### Merge mode with MVD (MMVD)

In addition to merge mode, where the implicitly derived motion information is directly used for prediction samples generation of the current CU, the merge mode with motion vector differences (MMVD) is introduced in VVC. A MMVD flag is signalled right after sending a skip flag and merge flag to specify whether MMVD mode is used for a CU.

In MMVD, after a merge candidate is selected, it is further refined by the signalled MVDs information. The further information includes a merge candidate flag, an index to specify motion magnitude, and an index for indication of motion direction. In MMVD mode, one for the first two candidates in the merge list is selected to be used as MV basis. The merge candidate flag is signalled to specify which one is used.

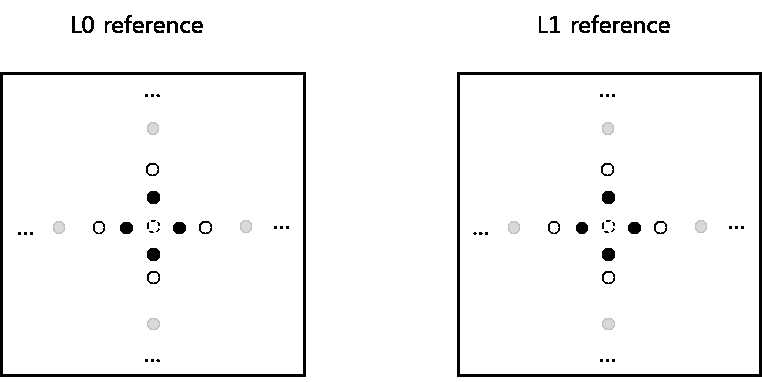


Figure 24 – MMVD Search Point

Distance index specifies motion magnitude information and indicate the pre-defined offset from the starting point. As shown in Figure 24, an offset is added to either horizontal component or vertical component of starting MV. The relation of distance index and pre-defined offset is specified in Table 3‑8

Table 3‑8 – The relation of distance index and pre-defined offset

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Distance IDX** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **Offset (in unit of luma sample)** | 1/4 | 1/2 | 1 | 2 | 4 | 8 | 16 | 32 |

Direction index represents the direction of the MVD relative to the starting point. The direction index can represent of the four directions as shown in Table 3‑9. It’s noted that the meaning of MVD sign could be variant according to the information of starting MVs. When the starting MVs is an un-prediction MV or bi-prediction MVs with both lists point to the same side of the current picture (i.e. POCs of two references are both larger than the POC of the current picture, or are both smaller than the POC of the current picture), the sign in Table 3‑9 specifies the sign of MV offset added to the starting MV. When the starting MVs is bi-prediction MVs with the two MVs point to the different sides of the current picture (i.e. the POC of one reference is larger than the POC of the current picture, and the POC of the other reference is smaller than the POC of the current picture), the sign in Table 3‑9 specifies the sign of MV offset added to the list0 MV component of starting MV and the sign for the list1 MV has opposite value.

Table 3‑9 – Sign of MV offset specified by direction index

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

### Symmetric MVD coding

In VTM6, besides the normal unidirectional prediction and bi-directional prediction mode MVD signalling, symmetric MVD mode for bi-predictional MVD signalling is applied. In the symmetric MVD mode, motion information including reference picture indices of both list-0 and list-1 and MVD of list-1 are not signaled but derived.

The decoding process of the symmetric MVD mode is as follows:

1. At slice level, variables BiDirPredFlag, RefIdxSymL0 and RefIdxSymL1 are derived as follows:

* If mvd\_l1\_zero\_flag is 1, BiDirPredFlag is set equal to 0.
* Otherwise, if the nearst reference picture in list-0 and the nearst reference picture in list-1 form a forward and backward pair of reference pictures or a backward and forward pair of reference pictures, BiDirPredFlag is set to 1, and both list-0 and list-1 reference pictures are short-term reference pictures. Otherwise BiDirPredFlag is set to 0.

1. At CU level, a symmetrical mode flag indicating whether symmetrical mode is used or not is explicitly signaled if the CU is bi-prediction coded and BiDirPredFlag is equal to 1.

When the symmetrical mode flag is true, only mvp\_l0\_flag, mvp\_l1\_flag and MVD0 are explicitly signaled. The reference indices for list-0 and list-1 are set equal to the pair of reference pictures, respectively. MVD1 is set equal to ( – MVD0 ). The final motion vectors are shown in below formula.

(3-14)

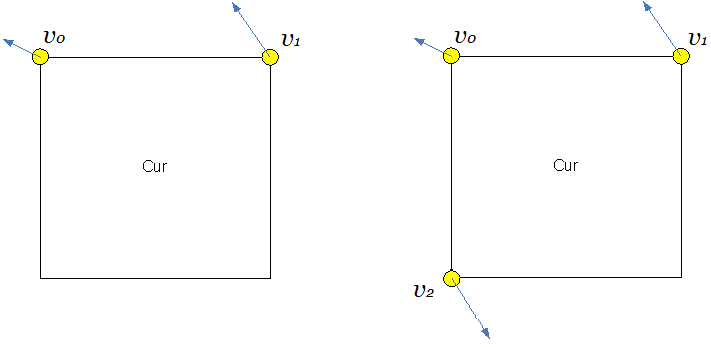


**Figure 25 – Illustration for symmetrical MVD mode**

In the encoder, symmetric MVD motion estimation starts with initial MV evaluation. A set of initial MV candidates comprising of the MV obtained from uni-prediction search, the MV obtained from bi-prediction search and the MVs from the AMVP list. The one with the lowest rate-distortion cost is chosen to be the initial MV for the symmetric MVD motion search.

### Affine motion compensated prediction

In HEVC, only translation motion model is applied for motion compensation prediction (MCP). While in the real world, there are many kinds of motion, e.g. zoom in/out, rotation, perspective motions and the other irregular motions. In the VTM6, a block-based affine transform motion compensation prediction is applied. As shown Figure 26, the affine motion field of the block is described by motion information of two control point (4-parameter) or three control point motion vectors (6-parameter).



1. **4 parameter affine model (b) 6 parameter affine model**

**Figure 26 – control point based affine motion model**

For 4-parameter affine motion model, motion vector at sample location (*x, y*) in a block is derived as:

(3-15)

For 6-parameter affine motion model, motion vector at sample location (*x, y*) in a block is derived as:

(3-16)

Where (*mv0x*, *mv0y*) is motion vector of the top-left corner control point, (*mv1x*, *mv1y*) is motion vector of the top-right corner control point, and (*mv2x*, *mv2y*) is motion vector of the bottom-left corner control point.

In order to simplify the motion compensation prediction, block based affine transform prediction is applied. To derive motion vector of each 4×4 luma subblock, the motion vector of the center sample of each subblock, as shown in Figure 27, is calculated according to above equations, and rounded to 1/16 fraction accuracy. Then the motion compensation interpolation filters are applied to generate the prediction of each subblock with derived motion vector. The subblock size of chroma-components is also set to be 4×4. The MV of a 4×4 chroma subblock is calculated as the average of the MVs of the four corresponding 4×4 luma subblocks.



Figure 27 – Affine MVF per subblock

As done for translational motion inter prediction, there are also two affine motion inter prediction modes: affine merge mode and affine AMVP mode.

#### Affine merge prediction

AF\_MERGE mode can be applied for CUs with both width and height larger than or equal to 8. In this mode the CPMVs of the current CU is generated based on the motion information of the spatial neighboring CUs. . There can be up to five CPMVP candidates and an index is signalled to indicate the one to be used for the current CU. The following three types of CPVM candidate are used to form the affine merge candidate list:

1. Inherited affine merge candidates that extrapolated from the CPMVs of the neighbour CUs
2. Constructed affine merge candidates CPMVPs that are derived using the translational MVs of the neighbour CUs
3. Zero MVs

In VTM6, there are maximum two inherited affine candidates, which are derived from affine motion model of the neighboring blocks, one from left neighboring CUs and one from above neighboring CUs. The candidate blocks are shown in Figure 28. For the left predictor, the scan order is A0->A1, and for the above predictor, the scan order is B0->B1->B2. Only the first inherited candidate from each side is selected. No pruning check is performed between two inherited candidates. When a neighboring affine CU is identified, its control point motion vectors are used to derived the CPMVP candidate in the affine merge list of the current CU. As shown in , if the neighbour left bottom block A is coded in affine mode, the motion vectors , and of the top left corner, above right corner and left bottom corner of the CU which contains the block A are attained. When block A is coded with 4-parameter affine model, the two CPMVs of the current CU are calculated according to , and . In case that block A is coded with 6-parameter affine model, the three CPMVs of the current CU are calculated according to , and .



Figure 28 – Locations of inherited affine motion predictors

  
**Figure 29 – Control point motion vector inheritance**

Constructed affine candidate means the candidate is constructed by combining the neighbor translational motion information of each control point. The motion information for the control points is derived from the specified spatial neighbors and temporal neighbor shown in Figure 30. CPMVk (k=1, 2, 3, 4) represents the k-th control point. For CPMV1, the B2->B3->A2 blocks are checked and the MV of the first available block is used. For CPMV2, the B1->B0 blocks are checked and for CPMV3, the A1->A0 blocks are checked. For TMVP is used as CPMV4 if it’s available.

After MVs of four control points are attained, affine merge candidates are constructed based on those motion information. The following combinations of control point MVs are used to construct in order:

{CPMV1, CPMV2, CPMV3}, {CPMV1, CPMV2, CPMV4}, {CPMV1, CPMV3, CPMV4},  
{CPMV2, CPMV3, CPMV4}, { CPMV1, CPMV2}, { CPMV1, CPMV3}

The combination of 3 CPMVs constructs a 6-parameter affine merge candidate and the combination of 2 CPMVs constructs a 4-parameter affine merge candidate. To avoid motion scaling process, if the reference indices of control points are different, the related combination of control point MVs is discarded.



**Figure 30 –Locations of Candidates position for constructed affine merge mode**

After inherited affine merge candidates and constructed affine merge candidate are checked, if the list is still not full, zero MVs are inserted to the end of the list.

#### Affine AMVP prediction

Affine AMVP mode can be applied for CUs with both width and height larger than or equal to 16. An affine flag in CU level is signalled in the bitstream to indicate whether affine AMVP mode is used and then another flag is signalled to indicate whether 4-parameter affine or 6-parameter affine. In this mode, the difference of the CPMVs of current CU and their predictors CPMVPs is signalled in the bitstream. The affine AVMP candidate list size is 2 and it is generated by using the following four types of CPVM candidate in order:/

1. Inherited affine AMVP candidates that extrapolated from the CPMVs of the neighbour CUs
2. Constructed affine AMVP candidates CPMVPs that are derived using the translational MVs of the neighbour CUs
3. Translational MVs from neighboring CUs
4. Zero MVs

The checking order of inherited affine AMVP candidates is same to the checking order of inherited affine merge candidates. The only difference is that, for AVMP candidate, only the affine CU that has the same reference picture as in current block is considered. No pruning process is applied when inserting an inherited affine motion predictor into the candidate list.

Constructed AMVP candidate is derived from the specified spatial neighbors shown in Figure 30. The same checking order is used as done in affine merge candidate construction. In addition, reference picture index of the neighboring block is also checked. The first block in the checking order that is inter coded and has the same reference picture as in current CUs is used. There is only one When the current CU is coded with 4-parameter affine mode, and and are both availlalbe, they are added as one candidate in the affine AMVP list. When the current CU is coded with 6-parameter affine mode, and all three CPMVs are available, they are added as one candidate in the affine AMVP list. Otherwise, constructed AMVP candidate is set as unavailable.

If affine AMVP list candidates is still less than 2 after inherited affine AMVP candidates and Constructed AMVP candidate are checked, , and will be added, in order, as the translational MVs to predict all control point MVs of the current CU, when available. Finally, zero MVs are used to fill the affine AMVP list if it is still not full.

#### Affine motion information storage

In VTM6, the CPMVs of affine CUs are stored in a separate buffer. The stored CPMVs are only used to generate the inherited CPMVPs in affine merge mode and affine AMVP mode for the lately coded CUs. The subblock MVs derived from CPMVs are used for motion compensation, MV derivation of merge/AMVP list of translational MVs and de-blocking.

To avoid the picture line buffer for the additional CPMVs, affine motion data inheritance from the CUs from above CTU is treated differently to the inheritance from the normal neighboring CUs. If the candidate CU for affine motion data inheritance is in the above CTU line, the bottom-left and bottom-right subblock MVs in the line buffer instead of the CPMVs are used for the affine MVP derivation. In this way, the CPMVs are only stored in local buffer. If the candidate CU is 6-parameter affine coded, the affine model is degraded to 4-parameter model. As shown in Figure 31, along the top CTU boundary, the bottom-left and bottom right subblock motion vectors of a CU are used for affine inheritance of the CUs in bottom CTUs.

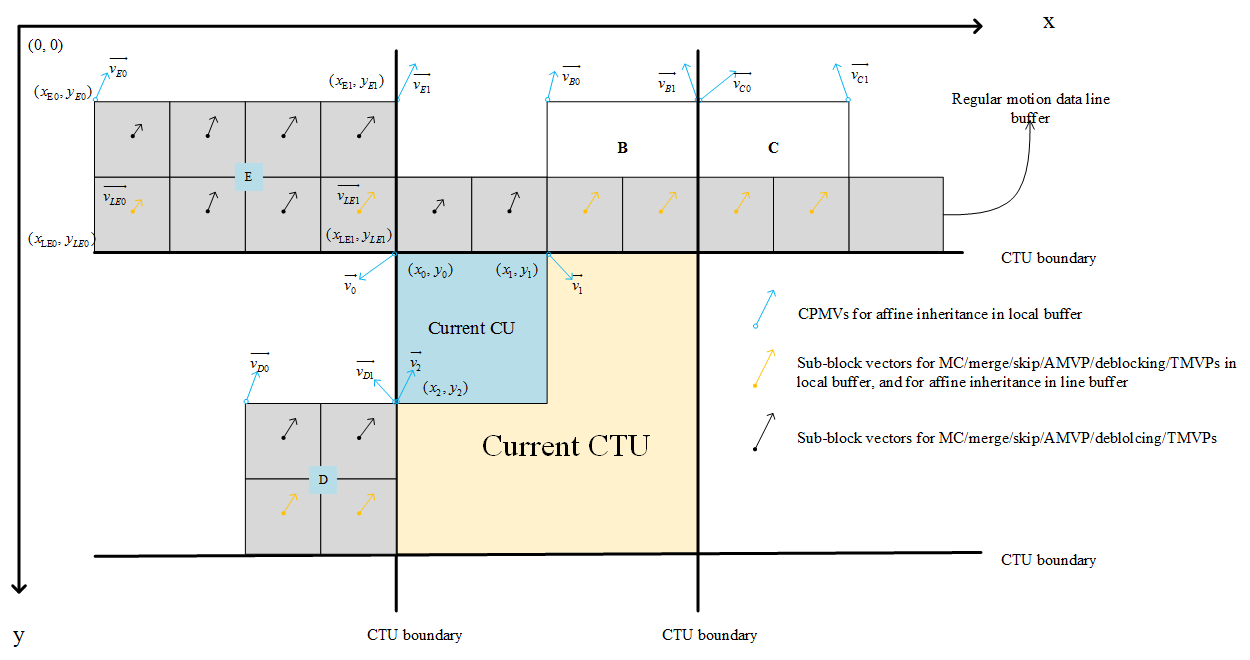


Figure 31 – Illustration of motion vector usage for proposed combined method

#### Prediction refinement with optical flow for affine mode

Subblock based affine motion compensation can save memory access bandwidth and reduce computation complexity compared to pixel based motion compensation, at the cost of prediction accuracy penalty. To achieve a finer granularity of motion compensation, prediction refinement with optical flow (PROF) is used to refine the subblock based affine motion compensated prediction without increasing the memory access bandwidth for motion compensation. In VTM, after the subblock based affine motion compensation is performed, luma prediction sample is refined by adding a difference derived by the optical flow equation. The PROF is described as following four steps:

Step 1) The subblock-based affine motion compensation is performed to generate subblock prediction .

Step2) The spatial gradients and of the subblock prediction are calculated at each sample location using a 3-tap filter [-1, 0, 1]. The gradient calculation is exactly the same as gradient calculation in BDOF.

(3-17)

(3-18)

is used to control the gradient’s precision. The subblock (i.e. 4x4) prediction is extended by one sample on each side for the gradient calculation. To avoid additional memory bandwidth and additional interpolation computation, those extended samples on the extended borders are copied from the nearest integer pixel position in the reference picture.

Step 3) The luma prediction refinement is calculated by the following optical flow equation.

(3-19)

where the is the difference between sample MV computed for sample location , denoted by , and the subblock MV of the subblock to which sample belongs, as shown in Figure 32.

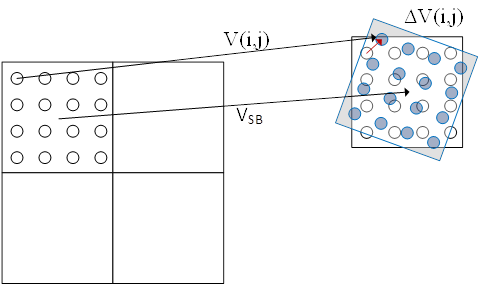


Figure 32 – Subblock MV VSB and pixel (red arrow)

Since the affine model parameters and the sample location relative to the subblock center are not changed from subblock to subblock, can be calculated for the first subblock, and reused for other subblocks in the same CU. Let and be the horizontal and vertical offset from the sample location to the center of the subblock , can be derived by the following equation,

(3-20)

(3-21)

In order to keep accuracy, the enter of the subblock is calculated as ((WSB-1)/2, (HSB-1)/2), where WSB and HSB are the subblock width and height, respectively.

For 4-parameter affine model,

(3-22)

For 6-parameter affine model,

(3-23)

where , , are the top-left, top-right and bottom-left control point motion vectors, and are the width and height of the CU.

Step 4) Finally, the luma prediction refinement is added to the subblock prediction . The final prediction *I’* is generated as the following equation.

PROF is not be applied in two cases for an affine coded CU: 1) all control point MVs are the same, which indicates the CU only has translational motion; 2) the affine motion parameters are greater than a specified limit because the subblock based affine MC is degraded to CU based MC to avoid large memory access bandwidth requirement.

A fast encoding method is applied to reduce the encoding complexity of affine motion estimation with PROF. PROF is not applied at affine motion estimation stage in following two situations: a) if this CU is not the root block and its parent block does not select the affine mode as its best mode, PROF is not applied since the possibility for current CU to select the affine mode as best mode is low; b) if the magnitude of four affine parameters (C, D, E, F) are all smaller than a predefined threshold and the current picture is not a low delay picture, PROF is not applied because the improvement introduced by PROF is small for this case. In this way, the affine motion estimation with PROF can be accelerated.

### Subblock-based temporal motion vector prediction (SbTMVP)

VTM supports the subblock-based temporal motion vector prediction (SbTMVP) method. Similar to the temporal motion vector prediction (TMVP) in HEVC, SbTMVP uses the motion field in the collocated picture to improve motion vector prediction and merge mode for CUs in the current picture. The same collocated picture used by TMVP is used for SbTVMP. SbTMVP differs from TMVP in the following two main aspects:

1. TMVP predicts motion at CU level but SbTMVP predicts motion at sub-CU level;
2. Whereas TMVP fetches the temporal motion vectors from the collocated block in the collocated picture (the collocated block is the bottom-right or center block relative to the current CU), SbTMVP applies a motion shift before fetching the temporal motion information from the collocated picture, where the motion shift is obtained from the motion vector from one of the spatial neighboring blocks of the current CU.

The SbTVMP process is illustrated in Figure 33. SbTMVP predicts the motion vectors of the sub-CUs within the current CU in two steps. In the first step, the spatial neighbor A1 in Figure 33 (a) is examined. If A1 has a motion vector that uses the collocated picture as its reference picture, this motion vector is selected to be the motion shift to be applied. If no such motion is identified, then the motion shift is set to (0, 0).

In the second step, the motion shift identified in Step 1 is applied (i.e. added to the current block’s coordinates) to obtain sub-CU-level motion information (motion vectors and reference indices) from the collocated picture as shown in Figure 33 (b). The example in Figure 33 (b) assumes the motion shift is set to block A1’s motion. Then, for each sub-CU, the motion information of its corresponding block (the smallest motion grid that covers the center sample) in the collocated picture is used to derive the motion information for the sub-CU. After the motion information of the collocated sub-CU is identified, it is converted to the motion vectors and reference indices of the current sub-CU in a similar way as the TMVP process of HEVC, where temporal motion scaling is applied to align the reference pictures of the temporal motion vectors to those of the current CU.



1. **Spatial neighboring blocks used by ATVMP**



1. **Deriving sub-CU motion field by applying a motion shift from spatial neighbor and scaling the motion information from the corresponding collocated sub-CUs**

**Figure 33 – The SbTMVP process in VVC**

In VTM6, a combined subblock based merge list which contains both SbTVMP candidate and affine merge candidates is used for the signalling of subblock based merge mode. The SbTVMP mode is enabled/disabled by a sequence parameter set (SPS) flag. If the SbTMVP mode is enabled, the SbTMVP predictor is added as the first entry of the list of subblock based merge candidates, and followed by the affine merge candidates. The size of subblock based merge list is signalled in SPS and the maximum allowed size of the subblock based merge list is 5 in VTM6.

The sub-CU size used in SbTMVP is fixed to be 8x8, and as done for affine merge mode, SbTMVP mode is only applicable to the CU with both width and height are larger than or equal to 8.

The encoding logic of the additional SbTMVP merge candidate is the same as for the other merge candidates, that is, for each CU in P or B slice, an additional RD check is performed to decide whether to use the SbTMVP candidate.

### Adaptive motion vector resolution (AMVR)

In HEVC, motion vector differences (MVDs) (between the motion vector and predicted motion vector of a CU) are signalled in units of quarter-luma-sample when use\_integer\_mv\_flag is equal to 0 in the slice header. In VVC, a CU-level adaptive motion vector resolution (AMVR) scheme is introduced. AMVR allows MVD of the CU to be coded in different precision. Dependent on the mode (normal AMVP mode or affine AVMP mode) for the current CU, the MVDs of the current CU can be adaptively selected as follows:

* Normal AMVP mode: quarter-luma-sample, half-luma-sample, integer-luma-sample or four-luma-sample.
* Affine AMVP mode: quarter-luma-sample, integer-luma-sample or 1/16 luma-sample.

The CU-level MVD resolution indication is conditionally signalled if the current CU has at least one non-zero MVD component. If all MVD components (that is, both horizontal and vertical MVDs for reference list L0 and reference list L1) are zero, quarter-luma-sample MVD resolution is inferred.

For a CU that has at least one non-zero MVD component, a first flag is signalled to indicate whether quarter-luma-sample MVD precision is used for the CU. If the first flag is 0, no further signaling is needed and quarter-luma-sample MVD precision is used for the current CU. Otherwise, a second flag is signalled to indicate half-luma-sample or other MVD precisions (interger or four-luma sample) is used for normal AMVP CU. In the case of half-luma-sample, a 6-tap interpolation filter instead of the default 8-tap interpolation filter is used for the half-luma sample position. Otherwise, a third flag is signalled to indicate whether integer-luma-sample or four-luma-sample MVD precision is used for normal AMVP CU. In the case of affine AMVP CU, the second flag is used to indicate whether integer-luma-sample or 1/16 luma-sample MVD precision is used. In order to ensure the reconstructed MV has the intended precision (quarter-luma-sample, half-luma-sample, integer-luma-sample or four-luma-sample), the motion vector predictors for the CU will be rounded to the same precision as that of the MVD before being added together with the MVD. The motion vector predictors are rounded toward zero (that is, a negative motion vector predictor is rounded toward positive infinity and a positive motion vector predictor is rounded toward negative infinity).

The encoder determines the motion vector resolution for the current CU using RD check. To avoid always performing CU-level RD check four times for each MVD resolution, in VTM6, the RD check of MVD precisions other than quarter-luma-sample is only invoked conditionally. For normal AVMP mode, the RD cost of quarter-luma-sample MVD precision and integer-luma sample MV precision is computed first. Then, the RD cost of integer-luma-sample MVD precision is compared to that of quarter-luma-sample MVD precision to decide whether it is necessary to further check the RD cost of four-luma-sample MVD precision. When the RD cost for quarter-luma-sample MVD precision is much smaller than that of the integer-luma-sample MVD precision, the RD check of four-luma-sample MVD precision is skipped. Then, the check of half-luma-sample MVD precision is skipped if the RD cost of integer-luma-sample MVD precision is significantly larger than the best RD cost of previously tested MVD precisions. For affine AMVP mode, if affine inter mode is not selected after checking rate-distortion costs of affine merge/skip mode, merge/skip mode, quarter-luma-sample MVD precision normal AMVP mode and quarter-luma-sample MVD precision affine AMVP mode, then 1/16 luma-sample MV precision and 1-pel MV precision affine inter modes are not checked. Furthermore affine parameters obtained in quarter-luma-sample MV precision affine inter mode is used as starting search point in 1/16 luma-sample and quarter-luma-sample MV precision affine inter modes.

### Motion field storage

In VTM6, the highest precision of explicitly signalled motion vectors is quarter-luma-sample. In some inter prediction modes such as the affine mode, motion vectors are derived at 1/16th-luma-sample precision and motion compensated prediction is performed at 1/16th-sample-precision. In terms of internal motion field storage, all motion vectors are stored at 1/16th-luma-sample precision.

For temporal motion field storage used by TMVP and ATVMP, motion field compression is performed at 8x8 granularity in contrast to the 16x16 granularity in HEVC.

### Bi-prediction with CU-level weight (BCW)

In HEVC, the bi-prediction signal is generated by averaging two prediction signals obtained from two different reference pictures and/or using two different motion vectors. In VTM6, the bi-prediction mode is extended beyond simple averaging to allow weighted averaging of the two prediction signals.

|  |  |
| --- | --- |
|  | (3-24) |

Five weights are allowed in the weighted averaging bi-prediction, For each bi-predicted CU, the weight w is determined in one of two ways: 1) for a non-merge CU, the weight index is signalled after the motion vector difference; 2) for a merge CU, the weight index is inferred from neighbouring blocks based on the merge candidate index. BCW is only applied to CUs with 256 or more luma samples (i.e., CU width times CU height is greater than or equal to 256). For low-delay pictures, all 5 weights are used. For non-low-delay pictures, only 3 weights (w∈{3,4,5}) are used.

* At the encoder, fast search algorithms are applied to find the weight index without significantly increasing the encoder complexity. These algorithms are summarized as follows. For further details readers are referred to the VTM software and document JVET-L0646. When combined with AMVR, unequal weights are only conditionally checked for 1-pel and 4-pel motion vector precisions if the current picture is a low-delay picture.
* When combined with affine, affine ME will be performed for unequal weights if and only if the affine mode is selected as the current best mode.
* When the two reference pictures in bi-prediction are the same, unequal weights are only conditionally checked.
* Unequal weights are not searched when certain conditions are met, depending on the POC distance between current picture and its reference pictures, the coding QP, and the temporal level.

The BCW weight index is coded using one context coded bin followed by bypass coded bins. The first context coded bin indicates if equal weight is used; and if unequal weight is used, additional bins are signalled using bypass coding to indicate which unequal weight is used.

Weighted prediction (WP) is a coding tool supported by the H.264/AVC and HEVC standards to efficiently code video content with fading. Support for WP was also added into the VVC standard. WP allows weighting parameters (weight and offset) to be signalled for each reference picture in each of the reference picture lists L0 and L1. Then, during motion compensation, the weight(s) and offset(s) of the corresponding reference picture(s) are applied. WP and BCW are designed for different types of video content. In order to avoid interactions between WP and BCW, which will complicate VVC decoder design, if a CU uses WP, then the BCW weight index is not signalled, and w is inferred to be 4 (i.e. equal weight is applied).For a merge CU, the weight index is inferred from neighbouring blocks based on the merge candidate index. This can be applied to both normal merge mode and inherited affine merge mode. For constructed affine merge mode, the affine motion information is constructed based on the motion information of up to 3 blocks. The BCW index for a CU using the constructed affine merge mode is simply set equal to the BCW index of the first control point MV.

In VTM6, CIIP and BCW cannot be jointly applied for a CU. When a CU is coded with CIIP mode, the BCW index of the current CU is set to 2, e.g. equal weight.

### Bi-directional optical flow (BDOF)

The bi-directional optical flow (BDOF) tool is included in VTM6. BDOF, previously referred to as BIO, was included in the JEM. Compared to the JEM version, the BDOF in VTM6 is a simpler version that requires much less computation, especially in terms of number of multiplications and the size of the multiplier.

BDOF is used to refine the bi-prediction signal of a CU at the 4×4 subblock level. BDOF is applied to a CU if it satisfies all the following conditions:

* The CU is coded using “true” bi-prediction mode, i.e., one of the two reference pictures is prior to the current picture in display order and the other is after the current picture in display order
* The CU is not coded using affine mode or the ATMVP merge mode
* CU has more than 64 luma samples
* Both CU height and CU width are larger than or equal to 8 luma samples
* BCW weight index indicates equal weight
* WP is not enabled for the current CU
* CIIP mode is not used for the current CU

As its name indicates, the BDOF mode is based on the optical flow concept, which assumes that the motion of an object is smooth. For each 4×4 subblock, a motion refinement is calculated by minimizing the difference between the L0 and L1 prediction samples. The motion refinement is then used to adjust the bi-predicted sample values in the 4x4 subblock. The following steps are applied in the BDOF process.

First, the horizontal and vertical gradients, and , , of the two prediction signals are computed by directly calculating the difference between two neighboring samples, i.e.,

|  |  |
| --- | --- |
|  | (3-25) |

where are the sample value at coordinate of the prediction signal in list , , and shift1 is calculated based on the luma bit depth, bitDepth, as shift1 = max( 6, bitDepth-6).

Then, the auto- and cross-correlation of the gradients, , , , and , are calculated as

|  |  |
| --- | --- |
| , | (3-26) |

where

|  |  |
| --- | --- |
|  | (3-27) |

where is a 6×6 window around the 4×4 subblock, and the values of and are set equal to min( 1, bitDepth-11 ) and min( 4, bitDepth-8), respectively.

The motion refinement is then derived using the cross- and auto-correlation terms using the following:

|  |  |
| --- | --- |
|  | (3-28) |

where , , . is the floor function, and .

Based on the motion refinement and the gradients, the following adjustment is calculated for each sample in the 4×4 subblock:

|  |  |
| --- | --- |
|  | (3-29) |

Finally, the BDOF samples of the CU are calculated by adjusting the bi-prediction samples as follows:

|  |  |
| --- | --- |
|  | (3-30) |

These values are selected such that the multipliers in the BDOF process do not exceed 15-bit, and the maximum bit-width of the intermediate parameters in the BDOF process is kept within 32-bit.

In order to derive the gradient values, some prediction samples in list () outside of the current CU boundaries need to be generated. As depicted in Figure 34, the BDOF in VTM6 uses one extended row/column around the CU’s boundaries. In order to control the computational complexity of generating the out-of-boundary prediction samples, prediction samples in the extended area (white positions) are generated by taking the reference samples at the nearby integer positions (using floor() operation on the coordinates) directly without interpolation, and the normal 8-tap motion compensation interpolation filter is used to generate prediction samples within the CU (gray positions). These extended sample values are used in gradient calculation only. For the remaining steps in the BDOF process, if any sample and gradient values outside of the CU boundaries are needed, they are padded (i.e. repeated) from their nearest neighbors.



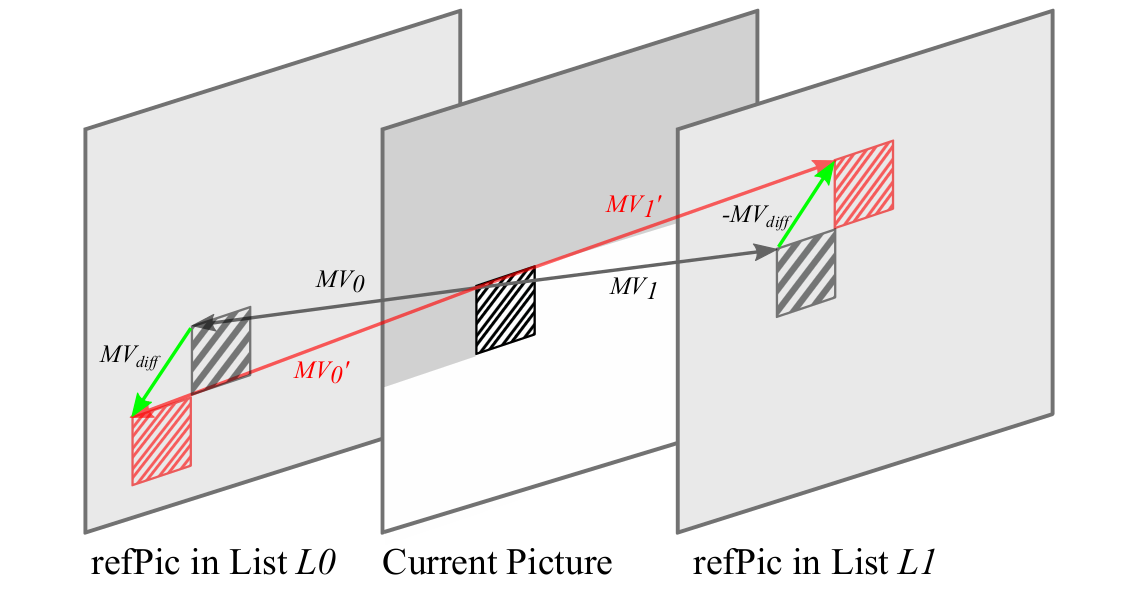
**Figure 34 – Extended CU region used in BDOF**

When the width and/or height of a CU are larger than 16 luma samples, it will be split into subblocks with width and/or height equal to 16 luma samples, and the subblock boundaries are treated as the CU boundaries in the BDOF process. The maximum unit size for BDOF process is limited to 16x16. For each subblock, the BDOF process could skipped. When the SAD of between the initial L0 and L1 prediction samples is smaller than a threshold, the BDOF process is not applied to the subblock. The threshold is set equal to (8 \* W\*( H >> 1 ), where W indicates the subblock width, and H indicates subblock height. To avoid the additional complexity of SAD calculation, the SAD between the initial L0 and L1 prediction samples calculated in DVMR process is re-used here.

If BCW is enabled for the current block, i.e., the BCW weight index indicates unequal weight, then bi-directional optical flow is disabled. Similarly, if WP is enabled for the current block, i.e., the luma\_weight\_lx\_flag is 1 for either of the two reference pictures, then BDOF is also disabled. When a CU is coded with symmetric MVD mode or CIIP mode, BDOF is also disabled.

### Decoder side motion vector refinement (DMVR)

In order to increase the accuracy of the MVs of the merge mode, a bilateral-matching based decoder side motion vector refinement is applied in VTM6. In bi-prediction operation, a refined MV is searched around the initial MVs in the reference picture list L0 and reference picture list L1. The BM method calculates the distortion between the two candidate blocks in the reference picture list L0 and list L1. As illustrated in Figure 35, the SAD between the red blocks based on each MV candidate around the initial MV is calculated. The MV candidate with the lowest SAD becomes the refined MV and used to generate the bi-predicted signal.



**Figure 35 – Decoding side motion vector refinement**

In VTM6, the DMVR can be applied for the CUs which are coded with following modes and features:

* CU level merge mode with bi-prediction MV
* One reference picture is in the past and another reference picture is in the future with respect to the current picture
* The distances (i.e. POC difference) from both reference pictures to the current picture are same
* CU has more than 64 luma samples
* Both CU height and CU width are larger than or equal to 8 luma samples
* BCW weight index indicates equal weight
* WP is not enabled for the current block
* CIIP mode is not used for the current block

The refined MV derived by DMVR process is used to generate the inter prediction samples and also used in temporal motion vector prediction for future pictures coding. While the original MV is used in deblocking process and also used in spatial motion vector prediction for future CU coding.

The additional features of VTM6 DMVR are mentioned in the following sub-clauses.

#### Searching scheme

In DVMR, the search points are surrounding the initial MV and the MV offset obey the MV difference mirroring rule. In other words, any points that are checked by DMVR, denoted by candidate MV pair (MV0, MV1) obey the following two equations:

(3-31)

(3-32)

Where represents the refinement offset between the initial MV and the refined MV in one of the reference pictures. In VTM6, the refinement search range is two integer luma samples from the initial MV. The searching includes the integer sample offset search stage and fractional sample refinement stage.

25 points full search is applied for integer sample offset searching. The SAD of the initial MV pair is first calculated. If the SAD of the initial MV pair is smaller than a threshold, the integer sample stage of DMVR is terminated. Otherwise SADs of the remaining 24 points are calculated and checked in raster scanning order. The point with the smallest SAD is selected as the output of integer sample offset searching stage. To reduce the penalty of the uncertainty of DMVR refinement, it is proposed to favor the original MV during the DMVR process. The SAD between the reference blocks referred by the initial MV candidates is decreased by 1/4 of the SAD value.

The integer sample search is followed by fractional sample refinement. To save the calculational complexity, the fractional sample refinement is derived by using parametric error surface equation, instead of additional search with SAD comparison. The fractional sample refinement is conditionally invoked based on the output of the integer sample search stage. When the integer sample search stage is terminated with center having the smallest SAD in either the first iteration or the second iteration search, the fractional sample refinement is further applied.

In parametric error surface based sub-pixel offsets estimation, the center position cost and the costs at four neighboring positions from the center are used to fit a 2-D parabolic error surface equation of the following form

(3-33)

where ( corresponds to the fractional position with the least cost and C corresponds to the minimum cost value. By solving the above equations by using the cost value of the five search points, the ( is computed as:

(3-34)

(3-35)

The value of and are automatically constrained to be between – 8 and 8 since all cost values are positive and the smallest value is . This corresponds to half peal offset with 1/16th-pel MV accuracy in VTM6. The computed fractional ( are added to the integer distance refinement MV to get the sub-pixel accurate refinement delta MV.

#### Bilinear-interpolation and sample padding

In VVC, the resolution of the MVs is 1/16 luma samples. The samples at the fractional position are interpolated using a 8-tap interpolation filter. In DMVR, the search points are surrounding the initial fractional-pel MV with integer sample offset, therefore the samples of those fractional position needs to be interpolated for DMVR search process. To reduce the calculation complexity, the bi-linear interpolation filter is used to generate the fractional samples for the searching process in DMVR. Another important effect is that by using bi-linear filter is that with 2-sample search range, the DVMR does not access more reference samples compared to the normal motion compensation process. After the refined MV is attained with DMVR search process, the normal 8-tap interpolation filter is applied to generate the final prediction. In order to not access more reference samples to normal MC process, the samples, which is not needed for the interpolation process based on the original MV but is needed for the interpolation process based on the refined MV, will be padded from those available samples.

#### Maximum DMVR processing unit

When the width and/or height of a CU are larger than 16 luma samples, it will be further split into subblocks with width and/or height equal to 16 luma samples. The maximum unit size for DMVR searching process is limit to 16x16.

### Triangle partition for inter prediction

In VTM6, a triangle partition mode is supported for inter prediction. The triangle partition mode is only applied to CUs that are 8x8 or larger. The triangle partition mode is signalled using a CU-level flag as one kind of merge mode, with other merge modes including the regular merge mode, the MMVD mode, the CIIP mode and the subblock merge mode.

When this mode is used, a CU is split evenly into two triangle-shaped partitions, using either the diagonal split or the anti-diagonal split (Figure 36). Each triangle partition in the CU is inter-predicted using its own motion; only uni-prediction is allowed for each partition, that is, each partition has one motion vector and one reference index. The uni-prediction motion constraint is applied to ensure that same as the conventional bi-prediction, only two motion compensated prediction are needed for each CU. The uni-prediction motion for each partition is derived using the process described in 3.4.11.1.



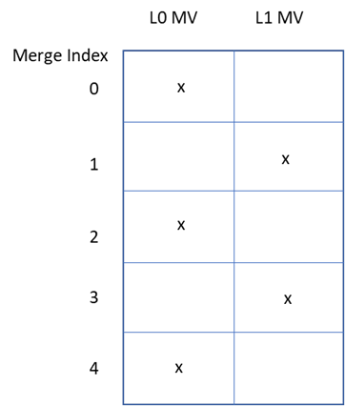
**Figure 36 – Triangle partition based inter prediction**

If triangle partition mode is used for the current CU, then a flag indicating the direction of the triangle partition (diagonal or anti-diagonal), and two merge indices (one for each partition) are further signalled. The number of maximum TPM candidate size is signalled explicitly at slice level and specifies syntax binarization for TMP merge indices. After predicting each of the triangle partitions, the sample values along the diagonal or anti-diagonal edge are adjusted using a blending processing with adaptive weights. This is the prediction signal for the whole CU, and transform and quantization process will be applied to the whole CU as in other prediction modes. Finally, the motion field of a CU predicted using the triangle partition mode is stored as in 3.4.11.3.

The triangle partition mode is not used in combination with SBT, that is, when the signalled triangle mode is equal to 1, the cu\_sbt\_flag is inferred to be 0 without signalling.

#### Uni-prediction candidate list construction

The uni-prediction candidate list is derived directly from the merge candidate list constructed according to the extended merge prediction process in 3.4.1. Denote n as the index of the uni-prediction motion in the triangle uni-prediction candidate list. The LX motion vector of the n-th extended merge candidate, with X equal to the parity of n, is used as the n-th uni-prediction motion vector for triangle partition mode. These motion vectors are marked with “x” in Figure 37. In case a corresponding LX motion vector of the n-the extended merge candidate does not exist, the L(1-X) motion vector of the same candidate is used instead as the uni-prediction motion vector for triangle partition mode.

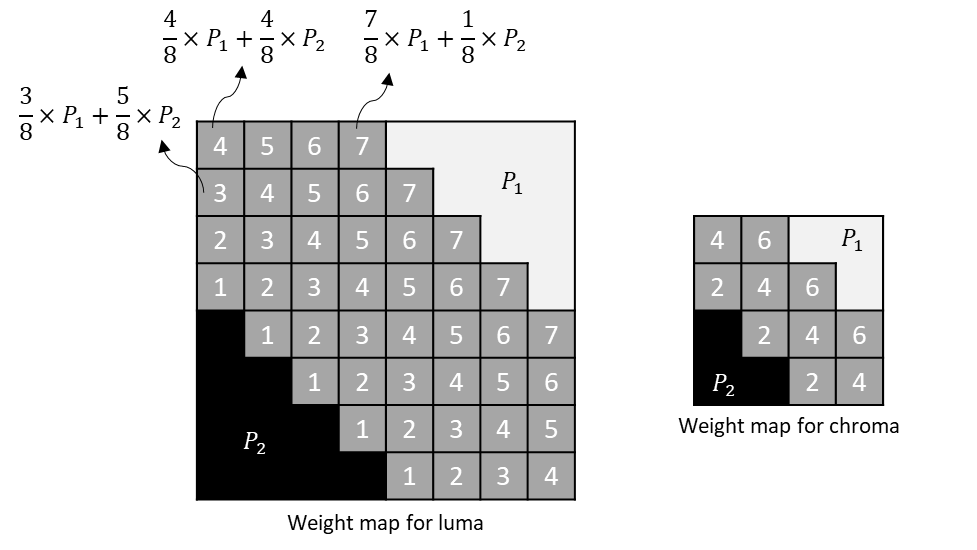


**Figure 37 – Uni-prediction MV selection for triangle partition mode**

#### Blending along the triangle partition edge

After predicting each triangle partition using its own motion, blending is applied to the two prediction signals to derive samples around the diagonal or anti-diagonal edge. The following weights are used in the blending process:

* 7/8, 6/8, 5/8, 4/8, 3/8, 2/8, 1/8} for luma and {6/8, 4/8, 2/8} for chroma, as shown in Figure 38.



**Figure 38 – weights used in the blending process**

#### Motion field storage

The motion vectors of a CU coded in triangle partition mode are generated using the following process:

1. If Mv1 and Mv2 are from different reference picture lists (one from L0 and the other from L1), then Mv1 and Mv2 are simply combined to form the bi-prediction motion vector.
2. Otherwise, if Mv1 and Mv2 are from the same list, only uni-prediction motion Mv2 is stored.

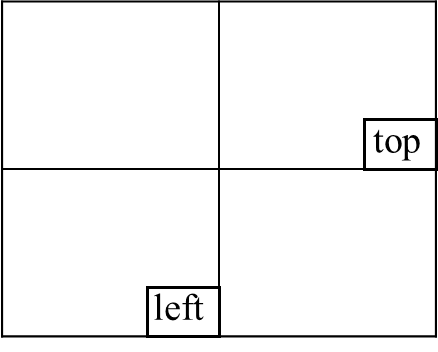
### Combined inter and intra prediction (CIIP)

In VTM6, when a CU is coded in merge mode, if the CU contains at least 64 luma samples (that is, CU width times CU height is equal to or larger than 64), and if both CU width and CU height are less than 128 luma samples, an additional flag is signalled to indicate if the combined inter/intra prediction (CIIP) mode is applied to the current CU. As its name indicates, the CIIP prediction combines an inter prediction signal with an intra prediction signal. The inter prediction signal in the CIIP mode is derived using the same inter prediction process applied to regular merge mode; and the intra prediction signal is derived following the regular intra prediction process with the planar mode. Then, the intra and inter prediction signals are combined using weighted averaging, where the weight value is calculated depending on the coding modes of the top and left neighbouring blocks (depicted in Figure 38) as follows:

* If the top neighbor is available and intra coded, then set isIntraTop to 1, otherwise set isIntraTop to 0;
* If the left neighbor is available and intra coded, then set isIntraLeft to 1, otherwise set isIntraLeft to 0;
* If (isIntraLeft + isIntraLeft) is equal to 2, then wt is set to 3;
* Otherwise, if (isIntraLeft + isIntraLeft) is equal to 1, then wt is set to 2;
* Otherwise, set wt to 1.

The CIIP prediction is formed as follows:

|  |  |
| --- | --- |
|  | (3-36) |



**Figure 39 – Top and left neighboring blocks used in CIIP weight derivation**

### Miscellaneous inter prediction aspects

To reduce memory bandwidth, the inter-coded 4x4 size CU is not allowed in VVC. For inter-coded 4x8/8x4 CU, only uni-directional mode is allowed. When the motion information from merge mode is bi-directional, it is converted to uni-directional by keeping only the list 0 motion information.

## Transform and quantization

### Large block-size transforms with high-frequency zeroing

In VTM6, large block-size transforms, up to 64×64 in size, are enabled, which is primarily useful for higher resolution video, e.g., 1080p and 4K sequences. High frequency transform coefficients are zeroed out for the transform blocks with size (width or height, or both width and height) equal to 64, so that only the lower-frequency coefficients are retained. For example, for an M×N transform block, with M as the block width and N as the block height, when M is equal to 64, only the left 32 columns of transform coefficients are kept. Similarly, when N is equal to 64, only the top 32 rows of transform coefficients are kept. When transform skip mode is used for a large block, the entire block is used without zeroing out any values. The VTM also supports configurable max transform size in SPS, such that encoder has the flexibility to choose up to 16-length, 32-length or 64-length transform size depending on the need of specific implementation.

### Multiple transform selection (MTS) for core transform

In addition to DCT-II which has been employed in HEVC, a Multiple Transform Selection (MTS) scheme is used for residual coding both inter and intra coded blocks. It uses multiple selected transforms from the DCT8/DST7. The newly introduced transform matrices are DST-VII and DCT-VIII. Table 3‑10 shows the basis functions of the selected DST/DCT.

Table 3‑10 - Transform basis functions of DCT-II/ VIII and DSTVII for N-point input

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

In order to keep the orthogonality of the transform matrix, the transform matrices are quantized more accurately than the transform matrices in HEVC. To keep the intermediate values of the transformed coefficients within the 16-bit range, after horizontal and after vertical transform, all the coefficients are to have 10-bit.

In order to control MTS scheme, separate enabling flags are specified at SPS level for intra and inter, respectively. When MTS is enabled at SPS, a CU level flag is signalled to indicate whether MTS is applied or not. Here, MTS is applied only for luma. The MTS CU level flag is signalled when the following conditions are satisfied.

* + - Both width and height smaller than or equal to 32
    - CBF flag is equal to one

If MTS CU flag is equal to zero, then DCT2 is applied in both directions. However, if MTS CU flag is equal to one, then two other flags are additionally signalled to indicate the transform type for the horizontal and vertical directions, respectively. Transform and signalling mapping table as shown in Table 3‑11. Unified the transform selection for ISP and implicit MTS is used by removing the intra-mode and block-shape dependencies. If current block is ISP mode or if the current block is intra block and both intra and inter explicit MTS is on, then only DST7 is used for both horizontal and vertical transform cores. When it comes to transform matrix precision, 8-bit primary transform cores are used. Therefore, all the transform cores used in HEVC are kept as the same, including 4-point DCT-2 and DST-7, 8-point, 16-point and 32-point DCT-2. Also, other transform cores including 64-point DCT-2, 4-point DCT-8, 8-point, 16-point, 32-point DST-7 and DCT-8, use 8-bit primary transform cores.

Table 3‑11 - Transform and signalling mapping table

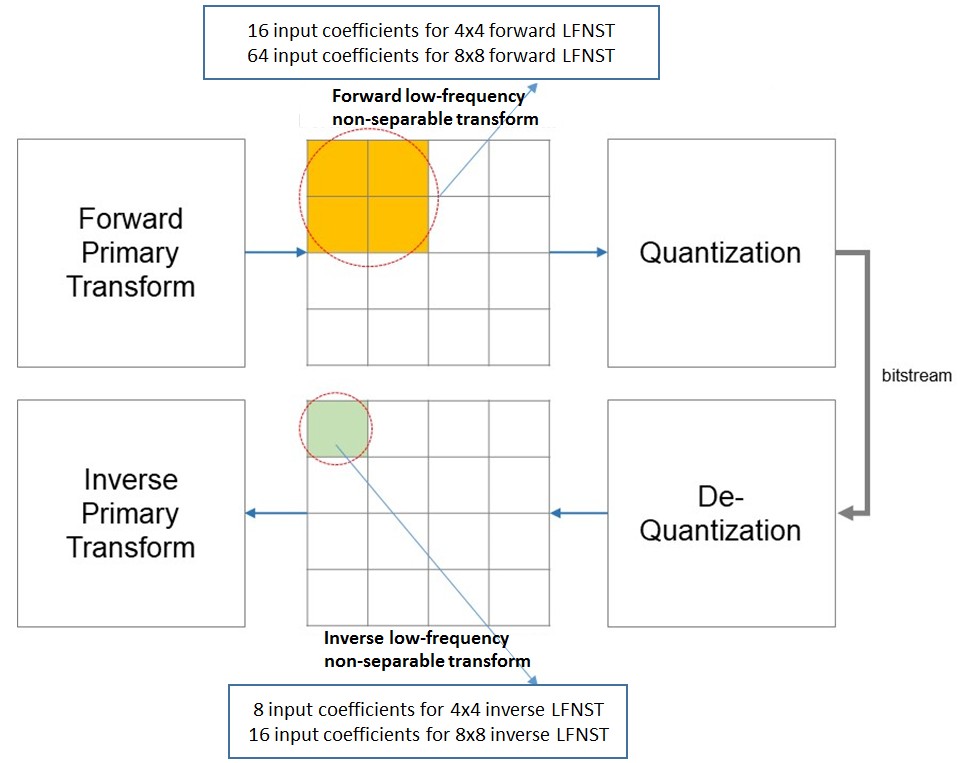
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MTS\_CU\_flag | MTS\_Hor\_flag | MTS\_Ver\_flag | Intra/inter | |
|  |  |  | Horizontal | Vertical |
| 0 |  |  | DCT2 | |
| 1 | 0 | 0 | DST7 | DST7 |
| 0 | 1 | DCT8 | DST7 |
| 1 | 0 | DST7 | DCT8 |
| 1 | 1 | DCT8 | DCT8 |

To reduce the complexity of large size DST-7 and DCT-8, High frequency transform coefficients are zeroed out for the DST-7 and DCT-8 blocks with size (width or height, or both width and height) equal to 32. Only the coefficients within the 16x16 lower-frequency region are retained.

As in HEVC, the residual of a block can be coded with transform skip mode. To avoid the redundancy of syntax coding, the transform skip flag is not signalled when the CU level MTS\_CU\_flag is not equal to zero. The block size limitation for transform skip is the same to that for MTS in JEM4, which indicate that transform skip is applicable for a CU when both block width and height are equal to or less than 32. Note that implicit MTS transform is set to DCT2 when LFNST or MIP is activated for the current CU. Also the implicit MTS can be still enabled when MTS is enabled for inter coded blocks.

### Low-Frequency Non-Separable Transform (LFNST)

In VVC, LFNST (low-frequency non-separable transform), which is known as reduced secondary transform, is applied between forward primary transform and quantization (at encoder) and between de-quantization and inverse primary transform (at decoder side) as shown in Figure 40. In LFNST, 4x4 non-separable transform or 8x8 non-separable transform is applied according to block size. For example, 4x4 LFNST is applied for small blocks (i.e., min (width, height) < 8) and 8x8 LFNST is applied for larger blocks (i.e., min (width, height) > 4).



**Figure 40 – Low-Frequency Non-Separable Transform (LFNST) process**

Application of a non-separable transform, which is being used in LFNST, is described as follows using input as an example. To apply 4x4 LFNST, the 4x4 input block *X*

(3-37)

is first represented as a vector :

(3-38)

The non-separable transform is calculated as , where indicates the transform coefficient vector, and *T* is a 16x16 transform matrix. The 16x1 coefficient vector is subsequently re-organized as 4x4 block using the scanning order for that block (horizontal, vertical or diagonal). The coefficients with smaller index will be placed with the smaller scanning index in the 4x4 coefficient block.

#### Reduced Non-separable transform

LFNST (low-frequency non-separable transform) is based on direct matrix multiplication approach to apply non-separable transform so that it is implemented in a single pass without multiple iterations. However, the non-separable transform matrix dimension needs to be reduced to minimize computational complexity and memory space to store the transform coefficients. Hence, reduced non-separable transform (or RST) method is used in LFNST. The main idea of the reduced non-separable transform is to map an **N** (N is commonly equal to 64 for 8x8 NSST) dimensional vector to an **R** dimensional vector in a different space, where **N/R** (**R** < **N**) is the reduction factor. Hence, instead of NxN matrix, RST matrix becomes an **R**×**N** matrix as follows:

|  |  |  |
| --- | --- | --- |
|  |  | (3-39) |

where the **R** rows of the transform are **R** bases of the **N** dimensional space. The inverse transform matrix for RT is the transpose of its forward transform. For 8x8 LFNST, a reduction factor of 4 is applied, and 64x64 direct matrix, which is conventional 8x8 non-separable transform matrix size, is reduced to16x48 direct matrix. Hence, the 48×16 inverse RST matrix is used at the decoder side to generate core (primary) transform coefficients in 8×8 top-left regions. When16x48 matrices are applied instead of 16x64 with the same transform set configuration, each of which takes 48 input data from three 4x4 blocks in a top-left 8x8 block excluding right-bottom 4x4 block. With the help of the reduced dimension, memory usage for storing all LFNST matrices is reduced from 10KB to 8KB with reasonable performance drop. In order to reduce complexity LFNST is restricted to be applicable only if all coefficients outside the first coefficient sub-group are non-significant. Hence, all primary-only transform coefficients have to be zero when LFNST is applied. This allows a conditioning of the LFNST index signalling on the last-significant position, and hence avoids the extra coefficient scanning in the current LFNST design, which is needed for checking for significant coefficients at specific positions only. The worst-case handling of LFNST (in terms of multiplications per pixel) restricts the non-separable transforms for 4x4 and 8x8 blocks to 8x16 and 8x48 transforms, respectively. In those cases, the last-significant scan position has to be less than 8 when LFNST is applied, for other sizes less than 16. For blocks with a shape of 4xN and Nx4 and N > 8, the proposed restriction implies that the LFNST is now applied only once, and that to the top-left 4x4 region only. As all primary-only coefficients are zero when LFNST is applied, the number of operations needed for the primary transforms is reduced in such cases. From encoder perspective, the quantization of coefficients is remarkably simplified when LFNST transforms are tested. A rate-distortion optimized quantization has to be done at maximum for the first 16 coefficients (in scan order), the remaining coefficients are enforced to be zero.

#### LFNST transform selection

There are totally 4 transform sets and 2 non-separable transform matrices (kernels) per transform set are used in LFNST. The mapping from the intra prediction mode to the transform set is pre-defined as shown in Table 3-11. If one of three CCLM modes (INTRA\_LT\_CCLM, INTRA\_T\_CCLM or INTRA\_L\_CCLM) is used for the current block (81 <= predModeIntra <= 83), transform set 0 is selected for the current chroma block. For each transform set, the selected non-separable secondary transform candidate is further specified by the explicitly signalled LFNST index. The index is signalled in a bit-stream once per Intra CU after transform coefficients.

Table 3-11 Transform selection table

|  |  |
| --- | --- |
| **IntraPredMode** | **Tr. set index** |
| IntraPredMode < 0 | 1 |
| 0 <= IntraPredMode <= 1 | 0 |
| 2 <= IntraPredMode <= 12 | 1 |
| 13 <= IntraPredMode <= 23 | 2 |
| 24 <= IntraPredMode <= 44 | 3 |
| 45 <= IntraPredMode <= 55 | 2 |
| 56 <= IntraPredMode<= 80 | 1 |
| 81 <= IntraPredMode<= 83 | 0 |

#### LFNST index Signaling and interaction with other tools

Since LFNST is restricted to be applicable only if all coefficients outside the first coefficient sub-group are non-significant, LFNST index coding depends on the position of the last significant coefficient. In addition, the LFNST index is context coded but does not depend on intra prediction mode, and only the first bin is context coded. Furthermore, LFNST is applied for intra CU in both intra and inter slices, and for both Luma and Chroma. If a dual tree is enabled, LFNST indices for Luma and Chroma are signaled separately. For inter slice (the dual tree is disabled), a single LFNST index is signaled and used for both Luma and Chroma.

When ISP mode is selected, LFNST is disabled and RST index is not signaled, because performance improvement was marginal even if RST is applied to every feasible partition block. Furthermore, disabling RST for ISP-predicted residual could reduce encoding complexity. LFNST is also disabled and the index is not signaled when MIP mode is selected.

Considering that a large CU greater than 64x64 is implicitly split (TU tiling) due to the existing maximum transform size restriction (64x64), an LFNST index search could increase data buffering by four times for a certain number of decode pipeline stages. Therefore, the maximum size that LFNST is allowed is restricted to 64x64. Note that LFNST is enabled with DCT2 only.

### Subblock Transform (SBT)

In VTM, subblock transform is introduced for an inter-predicted CU. In this transform mode, only a sub-part of the residual block is coded for the CU. When inter-predicted CU with cu\_cbf equal to 1, cu\_sbt\_flag may be signaled to indicate whether the whole residual block or a sub-part of the residual block is coded. In the former case, inter MTS information is further parsed to determine the transform type of the CU. In the latter case, a part of the residual block is coded with inferred adaptive transform and the other part of the residual block is zeroed out.

When SBT is used for an inter-coded CU, SBT type and SBT position information are signaled in the bitstream. There are two SBT types and two SBT positions, as indicated in Figure 41. For SBT-V (or SBT-H), the TU width (or height) may equal to half of the CU width (or height) or 1/4 of the CU width (or height), resulting in 2:2 split or 1:3/3:1 split. The 2:2 split is like a binary tree (BT) split while the 1:3/3:1 split is like an asymmetric binary tree (ABT) split. In ABT splitting, only the small region contains the non-zero residual. If one dimension of a CU is 8 in luma samples, the 1:3/3:1 split along that dimension is disallowed. There are at most 8 SBT modes for a CU.

Position-dependent transform core selection is applied on luma transform blocks in SBT-V and SBT-H (chroma TB always using DCT-2). The two positions of SBT-H and SBT-V are associated with different core transforms. More specifically, the horizontal and vertical transforms for each SBT position is specified in Figure 41. For example, the horizontal and vertical transforms for SBT-V position 0 is DCT-8 and DST-7, respectively. When one side of the residual TU is greater than 32, the transform for both dimensions is set as DCT-2. Therefore, the subblock transform jointly specifies the TU tiling, cbf, and horizontal and vertical core transform type of a residual block.

A variable maxSbtSize is signaled in SPS to specify the max CU size for which SBT can be applied. In the VTM6, for HD and 4K sequences, maxSbtSize is set as 64 by encoder; for other smaller resolution sequences, maxSbtSize is set as 32.

The SBT is not applied to the CU coded with combined inter-intra mode or TPM mode.



Figure 41 – SBT position, type and transform type

### Quantization

#### Quantization parameter control

In VVC Draft 6, Maximum QP was extended from 51 to 63, and the signaling of initial QP was changed accordingly. The initial value of SliceQpY is modified at the slice segment layer when a non-zero value of slice\_qp\_delta is coded. Specifically, the value of init\_qp\_minus26 is modified to be in the range of ( − 26 + QpBdOffsetY ) to +37. When the size of a transform block is not a power of 4, the transform coefficients are processed along with a modification to the QP or QP levelScale table rather than by multiplication by 181/256 (or 181/128), to compensate for an implicit scaling by the transform process. For transform skip block, minimum allowed Quantization Parameter (QP) is defined as 4 because quantization step size becomes 1 when QP is equal to 4.

In HEVC (and also in H.264), a fixed look-up table is used to convert the luma quantization parameter QPY to chroma quantization parameter QPC. In VVC, a more flexible luma-to-chroma QP mapping is used. Instead of having a fixed table, the luma-to-chroma QP mapping relationship is signalled in the SPS using a flexible piecewise linear model, with the only constraint on the linear model being that the slope of each piece cannot be negative (i.e., as luma QP increases, chroma QP must stay flat or increase, but cannot decrease). The piecewise linear model is defined by: 1) number of pieces in the model; 2) input (luma) and output (chroma) delta QPs for that piece. The input range of the piecewise linear model is [−QpBdOffsetY, 63] and the output range of the piecewise linear model is [−QpBdOffsetC, 63]. The QP mapping relationship can be signalled separately for Cb, Cr and joint Cb/Cr coding, or signalled jointly for all three types of residual coding.

Same as in HEVC, CU-level QP adaptation is allowed in VVC. Delta QP values for luma and chroma components can be signalled separately. For the chroma components, the allowed chroma QP offset values are signalled in the form of offset lists in the PPS in a similar manner as in HEVC. The lists are defined separately for Cb, Cr and joint Cb/Cr coding. Up to 6 offset values are allowed for each of Cb, Cr, and joint Cb/Cr lists. At the CU-level, an index is signalled to indicate which one of the offset values in the offset list is used to adjust the chroma QP for that CU.

#### Dependent quantization

In addition, the same HEVC scalar quantization is used with a new concept called dependent scalar quantization. Dependent scalar quantization refers to an approach in which the set of admissible reconstruction values for a transform coefficient depends on the values of the transform coefficient levels that precede the current transform coefficient level in reconstruction order. The main effect of this approach is that, in comparison to conventional independent scalar quantization as used in HEVC, the admissible reconstruction vectors are packed denser in the N-dimensional vector space (N represents the number of transform coefficients in a transform block). That means, for a given average number of admissible reconstruction vectors per N-dimensional unit volume, the average distortion between an input vector and the closest reconstruction vector is reduced. The approach of dependent scalar quantization is realized by: (a) defining two scalar quantizers with different reconstruction levels and (b) defining a process for switching between the two scalar quantizers.



Figure 42 – Illustration of the two scalar quantizers used in the proposed approach of dependent quantization.

The two scalar quantizers used, denoted by Q0 and Q1, are illustrated in Figure 42. The location of the available reconstruction levels is uniquely specified by a quantization step size Δ. The scalar quantizer used (Q0 or Q1) is not explicitly signalled in the bitstream. Instead, the quantizer used for a current transform coefficient is determined by the parities of the transform coefficient levels that precede the current transform coefficient in coding/reconstruction order.



Figure 43 – State transition and quantizer selection for the proposed dependent quantization.

As illustrated in Figure 43, the switching between the two scalar quantizers (Q0 and Q1) is realized via a state machine with four states. The state can take four different values: 0, 1, 2, 3. It is uniquely determined by the parities of the transform coefficient levels preceding the current transform coefficient in coding/reconstruction order. At the start of the inverse quantization for a transform block, the state is set equal to 0. The transform coefficients are reconstructed in scanning order (i.e., in the same order they are entropy decoded). After a current transform coefficient is reconstructed, the state is updated as shown in Figure 18, where k denotes the value of the transform coefficient level.

It is also supported to signal the default and user-defined scaling matrices. The DEFAULT mode scaling matrices are all flat, with elements equal to 16 for all TB sizes. IBC and intra coding modes currently share the same scaling matrices. Thus, for the case of USER\_DEFINED matrices, the number of MatrixType and MatrixType\_DC are updated as follows:

* **MatrixType:** **30 = 2** (2 for intra&IBC/inter) **×** **3** (Y/Cb/Cr components) **×** **5** (square TB size: from 4×4 to 64×64 for luma, from 2×2 to 32×32 for chroma)
* **MatrixType\_DC:** **14 = 2** (2 for intra&IBC/inter × 1 for Y component) **×** **3** (TB size: 16×16, 32×32, 64×64) **+** **4** (2 for intra&IBC/inter × 2 for Cb/Cr components) **×** **2** (TB size: 16×16, 32×32)

The DC values are separately coded for following scaling matrices: 16×16, 32×32, and 64×64. For TBs of size smaller than 8×8, all elements in one scaling matrix are signalled. If the TBs have size greater than or equal to 8×8, only 64 elements in one 8×8 scaling matrix are signalled as a base scaling matrix. For obtaining square matrices of size greater than 8×8, the 8×8 base scaling matrix is up-sampled (by duplication of elements) to the corresponding square size (i.e. 16×16, 32×32, 64×64). when the zeroing-out of the high frequency coefficients for 64-point transform is applied, corresponding high frequencies of the scaling matrices are also zeroed out. That is, if the width or height of the TB is greater than or equal to 32, only left or top half of the coefficients is kept, and the remaining coefficients are assigned to zero. Moreover, the number of elements signalled for the 64×64 scaling matrix is also reduced from 8×8 to three 4×4 submatrices, since the bottom-right 4×4 elements are never used.

### Joint coding of chroma residuals

VVC Draft 6 supports a mode where the chroma residuals are coded jointly. The usage (activation) of a joint chroma coding mode is indicated by a TU-level flag *tu\_joint\_cbcr\_residual\_flag* and the selected mode is implicitly indicated by the chroma CBFs. The flag *tu\_joint\_cbcr\_residual\_flag* is present if either or both chroma CBFs for a TU are equal to 1. In the PPS and slice header, chroma QP offset values are signalled for the joint chroma residual coding mode to differentiate from the usual chroma QP offset values signalled for regular chroma residual coding mode. These chroma QP offset values are used to derive the chroma QP values for those blocks coded using the joint chroma residual coding mode. When a corresponding joint chroma coding mode (modes 2 in Table 3-12) is active in a TU, this chroma QP offset is added to the applied luma-derived chroma QP during quantization and decoding of that TU. For the other modes (modes 1 and 3 in Table 3-12), the chroma QPs are derived in the same way as for conventional Cb or Cr blocks. The reconstruction process of the chroma residuals (resCb and resCr) from the transmitted transform blocks is depicted in Table 3-12. When this mode is activated, one single joint chroma residual block (resJointC[x][y] in Table 3-12) is signalled, and residual block for Cb (resCb) and residual block for Cr (resCr) are derived considering information such as tu\_cbf\_cb, tu\_cbf\_cr, and CSign, which is a sign value specified in the slice header.

At the encoder side, the joint chroma components are derived as explained in the following. Depending on the mode (listed in the tables above), resJointC{1,2} are generated by the encoder as follows:

* If mode is equal to 2 (single residual with reconstruction Cb = C, Cr = CSign \* C), the joint residual is determined according to

resJointC[ x ][ y ] = ( resCb[ x ][ y ] + CSign \* resCr[ x ][ y ] ) / 2.

* Otherwise, if mode is equal to 1 (single residual with reconstruction Cb = C, Cr = (CSign \* C) / 2), the joint residual is determined according to

resJointC[ x ][ y ] = ( 4 \* resCb[ x ][ y ] + 2 \* CSign \* resCr[ x ][ y ] ) / 5.

* Otherwise (mode is equal to 3, i.e., single residual, reconstruction Cr = C, Cb = (CSign \* C) / 2), the joint residual is determined according to

resJointC[ x ][ y ] = ( 4 \* resCr[ x ][ y ] + 2 \* CSign \* resCb[ x ][ y ] ) / 5.

Table 3-12 Reconstruction of chroma residuals. The value CSign is a sign value (+1 or –1), which is specified in the slice header, resJointC[ ][ ] is the transmitted residual.

|  |  |  |  |
| --- | --- | --- | --- |
| tu\_cbf\_cb | tu\_cbf\_cr | reconstruction of Cb and Cr residuals | mode |
| 1 | 0 | resCb[ x ][ y ] = resJointC[ x ][ y ]  resCr[ x ][ y ] = ( CSign \* resJointC[ x ][ y ] ) >> 1 | 1 |
| 1 | 1 | resCb[ x ][ y ] = resJointC[ x ][ y ]  resCr[ x ][ y ] = CSign \* resJointC[ x ][ y ] | 2 |
| 0 | 1 | resCb[ x ][ y ] = ( CSign \* resJointC[ x ][ y ] ) >> 1 resCr[ x ][ y ] = resJointC[ x ][ y ] | 3 |

The three joint chroma coding modes described above are only supported in I slices. In P and B slices, only mode 2 is supported. Hence, in P and B slices, the syntax element *tu\_joint\_cbcr\_residual\_flag* is only present if both chroma cbfs are 1. Note that transform depth is removed in the context modeling of tu\_cbf\_luma and tu\_cbf\_cb.

## Entropy coding

In the VVC draft 6, CABAC contains the following major changes compared to the design in HEVC:

* Core CABAC engine
* Separate residual coding structure for transform block and transform skip block.
* Context modeling for transform coefficients

### Core CABAC engine

The CABAC engine in HEVC uses a table-based probability transition process between 64 different representative probability states. In HEVC, the range ivlCurrRange representing the state of the coding engine is quantized to a set of 4 values prior to the calculation of the new interval range. The HEVC state transition can be implemented using a table containing all 64x4x8-bit pre-computed values to approximate the values of ivlCurrRange \* pLPS( pStateIdx ), where pLPS is the probability of the least probable symbol (LPS) and pStateIdx is the index of the current state. Also, a decode decision can be implemented using the pre-computed LUT. First ivlLpsRange is obtained using the LUT as follows. Then, ivlLpsRange is used to update ivlCurrRange and calculate the output binVal.

ivlLpsRange = rangeTabLps[ pStateIdx ][ qRangeIdx ] (3-40)

In VVC draft 6, CABAC has a QP dependent initialization process invoked at the beginning of each slice. Given the initial value of luma QP for the slice, the initial probability state of a context model, denoted as preCtxState, isderived as follows

m = slopeIdx × 5 – 45 (3-41)

n = (offsetIdx << 3) +7 (3-42)

preCtxState = Clip3(1, 127, ((m × (QP – 32)) >> 4) + n) (3-43)

where slopeIdx and offsetIdx are restricted to 3 bits, and total initialization values are represented by 6-bit precision. The probability state preCtxState represents the probability in the linear domain directly. Hence, preCtxState only needs proper shifting operations before input to arithmetic coding engine, and the logarithmic to linear domain mapping as well as the 256-byte table is saved.

pStateIdx0 = preCtxState << 3 (3-44)

pStateIdx1 = preCtxState << 7 (3-45)

### Transform coefficient level coding

In HEVC, transform coefficients of a coding block are coded using non-overlapped coefficient groups (CGs or subblocks), and each CG contains the coefficients of a 4x4 block of a coding block. In VVC Draft 6, the selection of coefficient group sizes becomes dependent upon TB size only, i.e., remove the dependency on channel type. As a consequence, various CGs (1x16, 2x8, 8x2, 2x4, 4x2 and 16x1) become available. The CGs inside a coding block, and the transform coefficients within a CG, are coded according to pre-defined scan orders. In order to restrict the maximum number of context coded bins per pixel, the area of the TB and the colour component are used to derive the maximum number of context-coded bins for a TB. For a luma TB, the maximum number of context-coded bins is equal to TB\_zosize\*1.75. For a chroma TB, the maximum number of context-coded bins (CCB) is equal to TB\_zosize\*1.25. Here, TB\_zosize indicates the number of samples within a TB after coefficient zero-out. Note that the coded\_sub\_block\_flag in transform skip residual mode is not considered for CCB count. Unlike HEVC where residual coding is designed for the statistics and signal characteristics of transform coefficient levels, two separate residual coding structures are employed for transform coefficients and transform skip coefficients, respectively.

#### Residual coding for transform coefficients

In transform coefficient coding, a variable, remBinsPass1, is first set to the maximum number of context-coded bins and is decreased by one when a context-coded bin is signalled. While the remBinsPass1 is larger than or equal to four, the first coding pass, which includes the sig\_coeff\_flag, abs\_level\_gt1\_flag, par\_level\_flag, and abs\_level\_gt3\_flag, is coded by using context-coded bins. If the number of context coded bin is not greater than Mccb in the first pass coding, the rest part of level information, which is indicated to be further coded in the first pass, is coded with syntax element of abs\_remainder by using Golomb-rice code and bypass-coded bins. When the remBinsPass1 becomes smaller than 4 while coding the first pass, the rest part of coefficients, which are indicated to be further coded in the first pass, are coded with a syntax element of abs\_remainder, and coefficients which are not coded in the first pass is directly coded in the second pass with the syntax element of dec\_abs\_level by using Golomb-Rice code and bypass-coded bins as depicted in Figure 43. The remBinsPass1 is reset for every TB. The transition of using context-coded bins for the sig\_coeff\_flag, abs\_level\_gt1\_flag, par\_level\_flag, and abs\_level\_gt3\_flag to using bypass-coded bins for the rest coefficients only happens at most once per TB. For a coefficient subblock, if the remBinsPass1 is smaller than 4, the entire coefficient subblock is coded by using bypass-coded bins. After all the above mentioned level coding, the signs (sign\_flag) for all scan positions with sig\_coeff\_flag equal to 1 is finally bypass coded.

The unified (same) rice parameter (ricePar) derivation is used for Pass 2 and Pass 3. The only difference is that baseLevel is set to 4 and 0 for Pass 2 and Pass 3, respectively. Rice parameter is determined not only based on sum of absolute levels of neighboring five transform coefficients in local template, but the corresponding base level is also taken into consideration as follow:

RicePara = RiceParTable[ max(min( 31, sumAbs – 5 \* baseLevel), 0) ] (3-46)

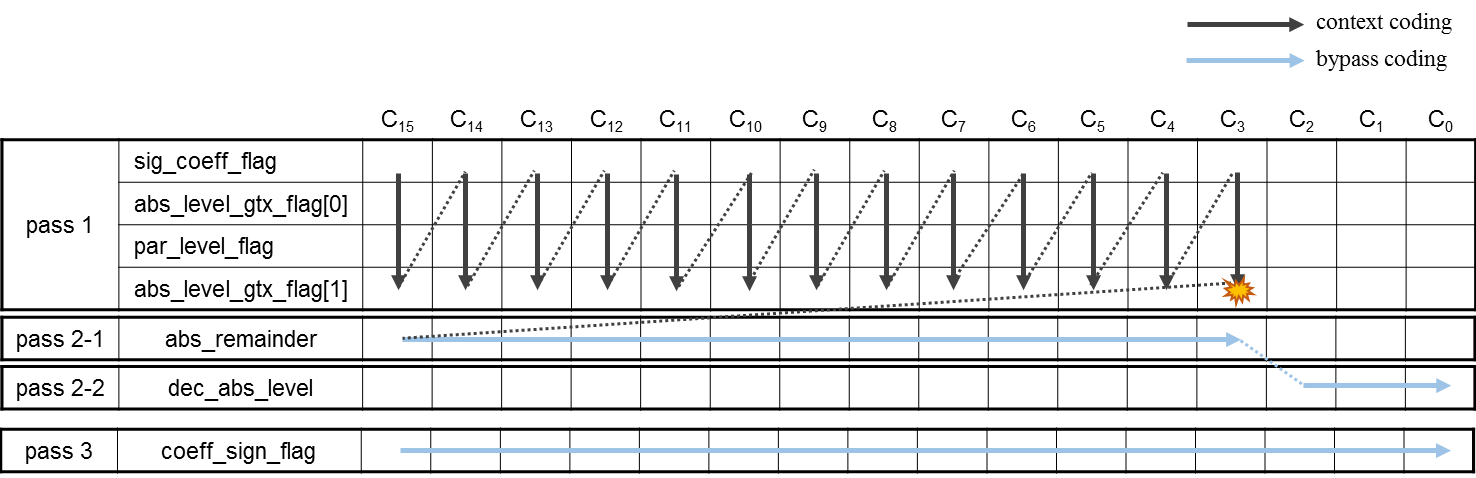


Figure 44 – residual coding structure for transform blocks

#### Residual coding for transform skip

Similar to HEVC, VVC supports transform skip mode. In transform skip mode, the statistical characteristics of the signal are different from those of transform coefficients, and applying transform to such residual in order to achieve energy compaction around low-frequency components is generally less effective. Residuals with such characteristics are often found in screen content as opposed to natural camera captured content. Therefore, detailed description of transform coefficient coding is described as part of the screen content coding tools in section 3.9.

### Context modeling for coefficient coding

The selection of probability models for the syntax elements related to absolute values of transform coefficient levels depends on the values of the absolute levels or partially reconstructed absolute levels in a local neighbourhood. The template used is illustrated in Figure 45.



Figure 45: Illustration of the template used for selecting probability models. The black square specifies the current scan position and the blue squares represent the local neighbourhood used.

The selected probability models depend on the sum of the absolute levels (or partially reconstructed absolute levels) in a local neighbourhood and the number of absolute levels greater than 0 (given by the number of sig\_coeff\_flags equal to 1) in the local neighbourhood. The context modelling and binarization depends on the following measures for the local neighbourhood:

* numSig: the number of non-zero levels in the local neighbourhood;
* sumAbs1: the sum of partially reconstructed absolute levels (absLevel1) after the first pass in the local neighbourhood;
* sumAbs: the sum of reconstructed absolute levels in the local neighbourhood
* diagonal position (d): the sum of the horizontal and vertical coordinates of a current scan position inside the transform block

Based on the values of numSig, sumAbs1, and d, the probability models for coding sig\_flag, par\_flag, gt1\_flag, and gt2\_flag are selected. The Rice parameter for binarizing abs\_remainder is selected based on the values of sumAbs and numSig.

In Draft 6 reduced 32-point MTS (RMTS32) based on skipping high frequency coefficients is used to reduce computational complexity of 32-point DST-7/DCT-8. And, it accompanies coefficient coding changes considering all types of zero-out (i.e., RMTS32 and the existing zero out for high frequency components in DCT2). Specifically, binarization of last non-zero coefficient position coding is coded based on reduced TU size, and the context model selection for the last non-zero coefficient position coding is determined by the original TU size. In addition, 60 context models are used to encode the sig\_coeff\_flag of transform coefficients. The selection of context model index is based on a sum of a maximum of five previously partially reconstructed absolute level called locSumAbsPass1 as follows

* If cIdx is equal to 0, ctxInc is derived as follows:

ctxInc = 12 \* Max( 0, QState – 1 ) +   
Min( ( locSumAbsPass1 + 1 ) >> 1, 3 ) + ( d < 2  ?  8  :  ( d < 5  ?  4  :  0 ) ) (3-47)

* Otherwise (cIdx is greater than 0), ctxInc is derived as follows:

ctxInc = 36 + 8 \* Max( 0, QState − 1) +   
Min( ( locSumAbsPass1 + 1 ) >> 1, 3 ) + ( d < 2  ?  4  :  0 ) (3-48)

## In-loop filters

There are totally three in-loop filters in VTM6. Besides deblocking filter and SAO (the two loop filters in HEVC), adaptive loop filter (ALF) are applied in the VTM6. The order of the filtering process in the VTM6 is the deblocking filter, SAO and ALF.

In the VTM6, the SAO and deblocking filtering processes are almost same as those in HEVC.

Starting from the VTM5, a new process called the luma mapping with chroma scaling was added (this process was previously known as the adaptive in-loop reshaper). This new process is performed before deblocking.

### Adaptive Loop Filter

In the VTM6, an Adaptive Loop Filter (ALF) with block-based filter adaption is applied. For the luma component, one among 25 filters is selected for each 4×4 block, based on the direction and activity of local gradients.

#### Filter shape

In the VTM6, two diamond filter shapes (as shown in Figure 46) are used. The 7×7 diamond shape is applied for luma component and the 5×5 diamond shape is applied for chroma components.

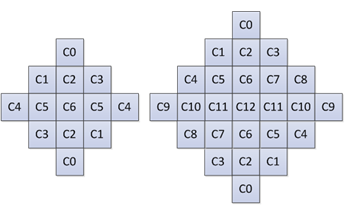


Figure 46 – ALF filter shapes (chroma: 5×5 diamond, luma: 7×7 diamond)

#### Block classification

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

(3-49)

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

(3-50)

(3-51)

(3-52)

(3-53)

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

To reduce the complexity of block classification, the subsampled 1-D Laplacian calculation is applied. As shown in Figure 47, the same subsampled positions are used for gradient calculation of all directions.

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

**Figure 47 – Subsampled Laplacian calculation**

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

, (3-54)

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

, (3-55)

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

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

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

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

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

The activity value is calculated as:

(3-56)

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

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

#### Geometric transformations of filter coefficients and clipping values

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

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

Diagonal: (3-57)

Vertical flip: , (3-58)

Rotation: , (3-59)

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

**Table 3‑12 - Mapping of the gradient calculated for one block and the transformations**

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

#### Filter parameters signalling

In the VTM6, ALF filter parameters are signalled in Adaptation Parameter Set (APS). In one APS, up to 25 sets of luma filter coefficients and clipping value indexes, and up to eight sets of chroma filter coefficients and clipping value indexes could be signalled. To reduce bits overhead, filter coefficients of different classification for luma component can be merged. In slice header, the indices of the APSs used for the current slice are signaled.

Clipping value indexes, which are decoded from the APS, allow determining clipping values using a Luma table of clipping values and a Chroma table of clipping values. These clipping values are dependent of the internal bitdepth. More precisely, the Luma table of clipping values and Chroma table of clipping values are obtained by the following formulas:

AlfClipL, (3-60)

AlfClipC (3-61)

with B equal to the internal bitdepth and N equal to 4 which is the number of allowed clipping values in VTM6.0.

In slice header, up to 7 APS indices can be signaled to specify the luma filter sets that are used for the current slice. The filtering process can be further controlled at CTB level. A flag is always signalled to indicate whether ALF is applied to a luma CTB. A luma CTB can choose a filter set among 16 fixed filter sets and the filter sets from APSs. A filter set index is signaled for a luma CTB to indicate which filter set is applied. The 16 fixed filter sets are pre-defined and hard-coded in both the encoder and the decoder.

For chroma component, an APS index is signaled in slice header to indicate the chroma filter sets being used for the current slice. At CTB level, a filter index is signaled for each chroma CTB if there is more than one chroma filter set in the APS.

The filter coefficients are quantized with norm equal to 128. In order to restrict the multiplication complexity, a bitstream conformance is applied so that the coefficient value of the non-central position shall be in the range of −27 to 27 − 1, inclusive. The central position coefficient is not signalled in the bitstream and is considered as equal to 128.

#### Filtering process

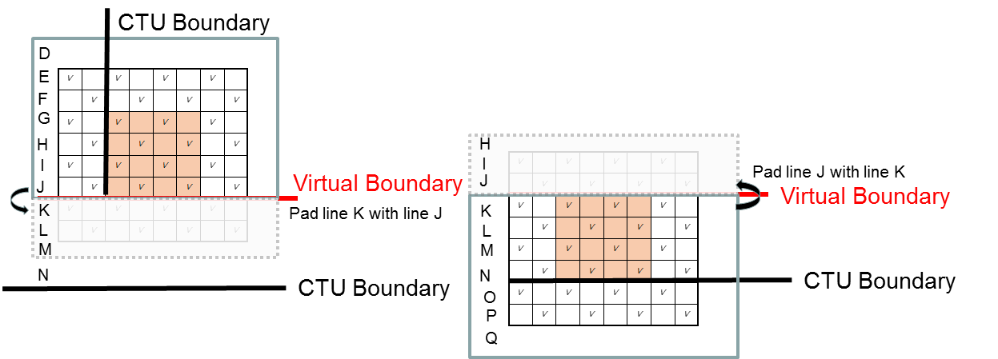
At decoder side, when ALF is enabled for a CTB, each sample within the CU is filtered, resulting in sample value as shown below,

(3-62)

where denotes the decoded filter coefficients, is the clipping function and denotes the decoded clipping parameters. The variable k and l varies between and where *L* denotes the filter length. The clipping function which corresponds to the function

#### Virtual boundary filtering process for line buffer reduction

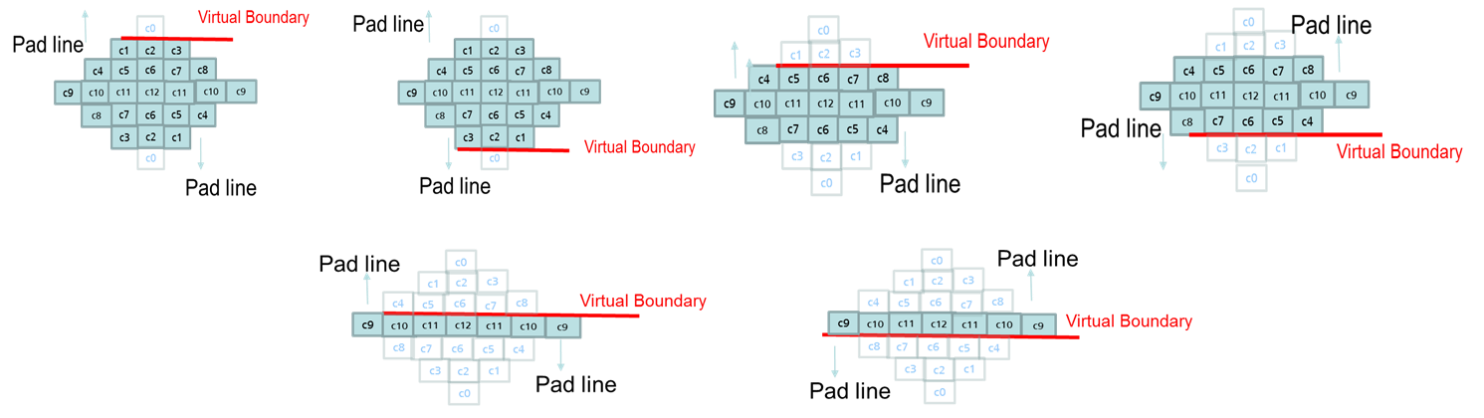
In VTM6, to reduce the line buffer requirement of ALF, modified block classification and filtering are employed for the samples near horizontal CTU boundaries. For this purpose, a virtual boundary is defined as a line by shifting the horizontal CTU boundary with “N” samples as shown in Figure 48, with N equal to 4 for the Luma component and 2 for the Chroma component.



**Figure 48 – Modified block classification at virtual boundaries**

Modified block classification is applied for the Luma component as depicted in Figure 48. For the 1D Laplacian gradient calculation of the 4x4 block above the virtual boundary, only the samples above the virtual boundary are used. Similarly for the 1D Laplacian gradient calculation of the 4x4 block below the virtual boundary, only the samples below the virtual boundary are used. The quantization of activity value A is accordingly scaled by taking into account the reduced number of samples used in 1D Laplacian gradient calculation.

For filtering processing, symmetric padding operation at the virtual boundaries are used for both Luma and Chroma components. As shown in Figure 49, when the sample being filtered is located below the virtual boundary, the neighboring samples that are located above the virtual boundary are padded. Meanwhile, the corresponding samples at the other sides are also padded, symmetrically.



**Figure 49 – Modified ALF filtering for Luma component at virtual boundaries**

The process is also applied for slice, brick and tile boundaries when filter across the boundaries is disabled. For the ALF block classification, only the samples which are in the same slice, brick or tile are used and the activity value is scaled accordingly. For the ALF filtering, symmetric padding is applied both in horizontal and vertical directions for horizontal and vertical boundaries respectively.

### Deblocking filter

In the VTM6, deblocking filtering process is mostly the same to those in HEVC. However, the following modifications are added.

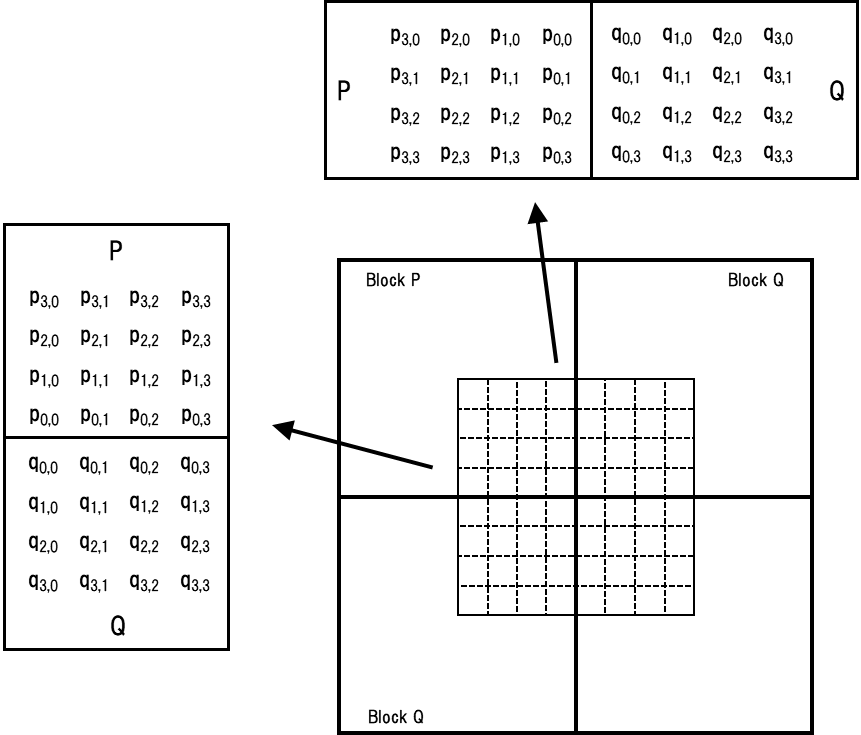
* 1. The filter strength of the deblocking filter dependent of the averaged luma level of the reconstructed samples.
  2. Deblocking tC table extension and adaptation to 10-bit video
  3. 4x4 grid deblocking for luma
  4. Stronger deblocking filter for luma
  5. Stronger deblocking filter for chroma
  6. Deblocking filter for subblock boundary
  7. Deblocking decision adapted to smaller difference in motion

#### Filter strength dependent on reconstructed average luma level

In HEVC, the filter strength of the deblocking filter is controlled by the variables β and tC which are derived from the averaged quantization parameters qPL. In the VTM6, deblocking filter controls the strength of the deblocking filter by adding offset to qPL according to the luma level of the reconstructed samples. The reconstructed luma level LL is derived as follow:

LL= ( ( p0,0 + p0,3 + q0,0 + q0,3 ) >> 2 ) / ( 1 << bitDepth ) (3-63)

where, the sample values pi,k and qi,k with i = 0..3 and k = 0 and 3 are derived as shown in Figure 50.



**Figure 50 – Sample position of pi,k and qi,k**

The variable qPL is derived as follows:

qPL = ( ( QpQ + QpP +1 ) >> 1 ) + qpOffset (3-64)

where QpQ and QpP denote the quantization parameters of the coding units containing the sample q0,0 and p0,0, respectively. The offset qpOffset dependent on transfer function, the values are signalled in the SPS.

#### Deblocking tC table extension and adaptation to 10-bit video

In VTM6, Maximum QP was changed from 51 to 63, and it is desired to reflect corresponding change to deblocking table, which derive values of deblocking parameters tC based on the block QP. The table was also adapted to 10-bit video instead of 8-bit video as was the case for HEVC. The following is updated tC table to accommodate the extension of the QP range and 10-bit video.

tC = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3,4,4,4,4,5,5,5,5,7,7,8,9,10,10,11,13,14,15,17,19,21,24,25,29,33,36,41,45,51,57,64,71,80,89,100,112,125,141,157,177,198,222,250,280,314,352,395]

#### 4x4 deblocking grid for luma

#### HEVC uses an 8x8 deblocking grid for both luma and chroma. In VTM6, deblocking on a 4x4 grid for luma boundaries was introduced to handle blocking artifacts from rectangular transform shapes. Parallel friendly luma deblocking on a 4x4 grid is achieved by restricting the number of samples to be deblocked to 1 sample on each side of a vertical luma boundary where one side has a width of 4 or less or to 1 sample on each side of a horizontal luma boundary where one side has a height of 4 or less.

#### Stronger deblocking filter for luma

A bilinear filter (stronger deblocking filter) is used when samples at either one side of a boundary belong to a large block. A sample belonging to a large block is defined as when the width is larger than or equal to 32 for a vertical edge, and when height is larger than or equal to 32 for a horizontal edge. Block boundary samples pi for i=0 to Sp-1 and qi for j=0 to Sq-1 are then replaced by linear interpolation as follows:

(3-65)

(3-66)

where and term is a position dependent clipping and , , , and are given below:

**Table 3‑13 – Derivation of stronger deblocking parameters for luma**

|  |  |
| --- | --- |
| Sp, Sq  7, 7  (p side: 7,  q side: 7) |  |
| 7, 3  (p side: 7  q side: 3) |  |
| 3, 7  (p side: 3  q side: 7) |  |
| 7, 5  (p side: 7  q side: 5) |  |
| 5, 7  (p side: 5  q side: 7) |  |
| 5, 5  (p side: 5  q side: 5) |  |
| 5, 3  (p side: 5  q side: 3) |  |
| 3, 5  (p side: 3  q side: 5) |  |

Above mentioned stronger luma filters are used only if all of the **Condition1**, **Condition2** and **Condition 3** are TRUE. The condition 1 is the “large block condition”. This condition detects whether the samples at P-side and Q-side belong to large blocks. The condition 2 and condition 3 are determined by:

**Condition2** = (d < β) ? TRUE: FALSE

**Condition3 =** StrongFilterCondition = (dpq is less than ( β  >>  2 ), sp3 + sq3 is less than ( 3\*β  >>  5 ), and Abs( p0 − q0 ) is less than ( 5 \* tC + 1 )  >>  1) ? TRUE : FALSE

#### Strong deblocking filter for chroma

The following strong deblocking filter for chroma is defined:

p2′= (3\*p3+2\*p2+p1+p0+q0+4) >> 3 (3-67)

p1′= (2\*p3+p2+2\*p1+p0+q0+q1+4) >> 3 (3-68)

p0′= (p3+p2+p1+2\*p0+q0+q1+q2+4) >> 3 (3-69)

The above chroma filter performs deblocking on a 8x8 chroma sample grid. The chroma strong filters are used on both sides of the block boundary. Here, the chroma filter is selected when both sides of the chroma edge are greater than or equal to 8 (in unit of chroma sample), and the following decision with three conditions are satisfied. The first one is for decision of boundary strength as well as large block. The second and third one are basically the same as for HEVC luma decision, which are on/off decision and strong filter decision, respectively. In the first decision, boundary strength (bS) is modified for chroma filtering as shown in **Table 3‑14**. The conditions in **Table 3‑14** are checked sequentially. If a condition is satisfied then the remaining conditions with lower priorities are skipped.

**Table 3‑14 – The modified boundary strength**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Priority | Conditions | Y | U | V |
| 5 | At least one of the adjacent blocks is intra | 2 | 2 | 2 |
| 4 | At least one of the adjacent blocks has non-zero transform coefficients | 1 | 1 | 1 |
| 3 | Absolute difference between the motion vectors that belong to the adjacent blocks is greater than or equal to one half luma sample | 1 | N/A | N/A |
| 2 | Motion prediction in the adjacent blocks refers to vectors is different | 1 | N/A | N/A |
| 1 | Otherwise | 0 | 0 | 0 |

Chroma deblocking is performing when bS is equal to 2, or bS is equal to 1 when a large block boundary is detected. The second and third condition is basically the same as HEVC luma strong filter decision.

#### Deblocking filter for subblock boundary

The deblocking filtering process are applied on a 4x4 grid for CU boundaries and transform subblock boundaries and on an 8x8 grid for prediction subblock boundaries. The prediction subblock boundaries include the prediction unit boundaries introduced by STMVP and affine modes, and the transform subblock boundaries include the transform unit boundaries introduced by SBT and ISP modes, and transforms due to implicit split of large CUs.

For SBT and ISP subblocks, similar to the logic in TU in HEVC deblocking filter, the deblocking filter is applied on TU boundary when there are non-zero coefficients in either transform subblock across the edge.

For SbTMVP and affine prediction subblocks, similar to the logic in PU in HEVC, the deblocking filter is applied on 8x8 grid with the consideration of the difference between motion vectors and reference pictures of the neighboring prediction subblock.

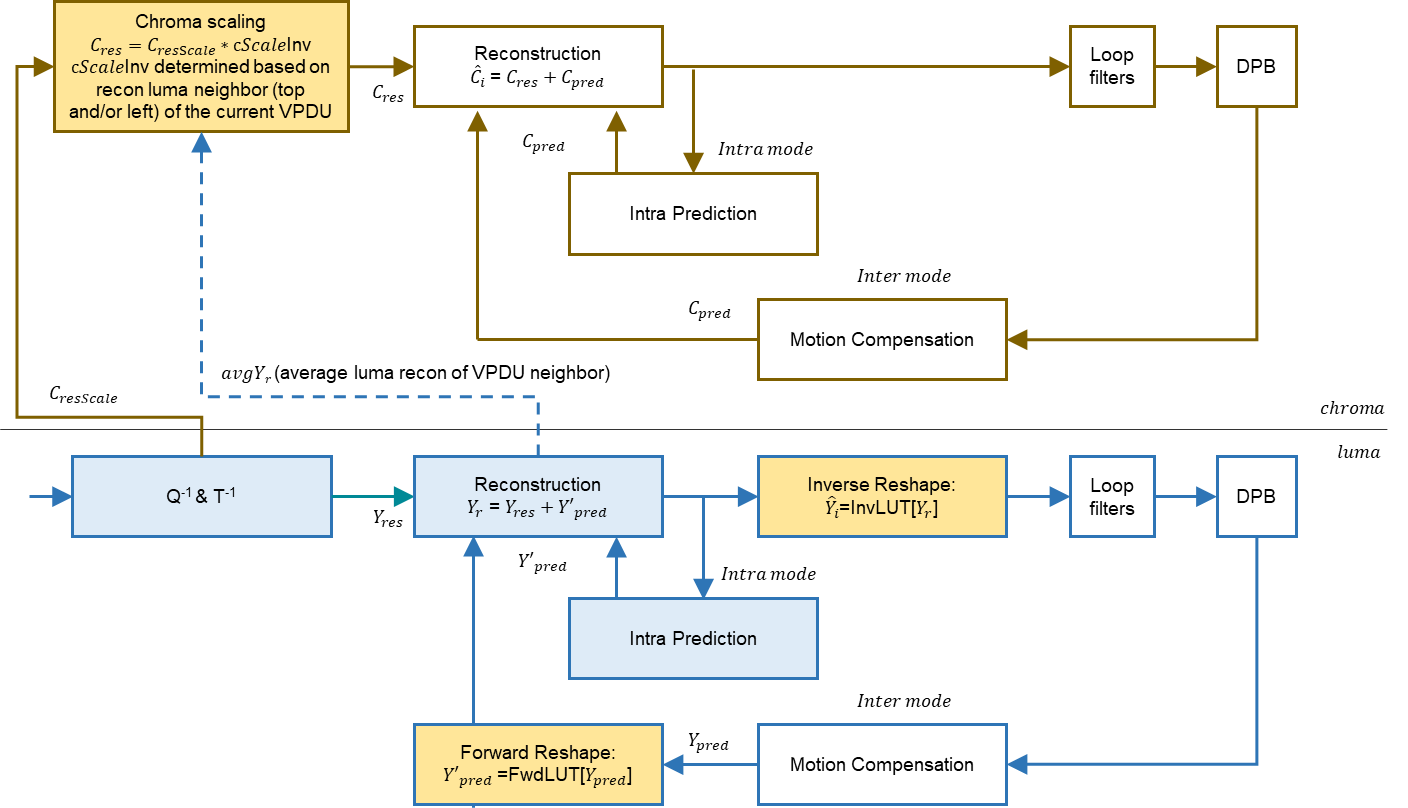
Transform block boundaries can at most be deblocked with 5 samples on a side of transform boundary which also is part of a coding block where SbTMVP or affine is used to enable parallel friendly deblocking. Internal prediction subblock boundaries 4 samples from a transform block boundary are at most deblocked by 1 sample on each side, internal prediction subblock boundaries 8 samples away from a transform block boundary are at most deblocked by 2 samples on each side of the boundary and other internal prediction subblock boundaries are at most deblocked with 3 samples on each side of the boundary.

### Deblocking decision adapted to smaller difference in motion

HEVC enables deblocking of a prediction unit boundary when the difference in at least one motion vector component between blocks on respective side of the boundary is equal to or larger than a threshold of 1 sample. In VTM6, a threshold of a half luma sample is introduced to also enable removal of blocking artifacts originating from boundaries between inter prediction units that have a small difference in motion vectors.

### Luma mapping with chroma scaling (LMCS)

In VVC, a coding tool called the luma mapping with chroma scaling (LMCS) is added as a new processing block before the loop filters. LMCS has two main components: 1) in-loop mapping of the luma component based on adaptive piecewise linear models; 2) for the chroma components, luma-dependent chroma residual scaling is applied. Figure 51 shows the LMCS architecture from decoder’s perspective. The light-blue shaded blocks in Figure 51 indicate where the processing is applied in the mapped domain; and these include the inverse quantization, inverse transform, luma intra prediction and adding of the luma prediction together with the luma residual. The unshaded blocks in Figure 51 indicate where the processing is applied in the original (i.e., non-mapped) domain; and these include loop filters such as deblocking, ALF, and SAO, motion compensated prediction, chroma intra prediction, adding of the chroma prediction together with the chroma residual, and storage of decoded pictures as reference pictures. The light-yellow shaded blocks in Figure 51 are the new LMCS functional blocks, including forward and inverse mapping of the luma signal and a luma-dependent chroma scaling process. Like most other tools in VVC, LMCS can be enabled/disabled at the sequence level using an SPS flag.

****

**Figure 51 – Luma mapping with chroma scaling architecture**

#### Luma mapping with piecewise linear model

The in-loop mapping of the luma component adjusts the dynamic range of the input signal by redistributing the codewords across the dynamic range to improve compression efficiency. Luma mapping makes use of a forward mapping function, *FwdMap*, and a corresponding inverse mapping function, *InvMap*. The *FwdMap* function is signalled using a piecewise linear model with 16 equal pieces. *InvMap* function does not need to be signalled and is instead derived from the *FwdMap* function.

The luma mapping model is signalled in the adaptation parameter set (APS) syntax structure with aps\_params\_type set equal to 1 (LMCS\_APS). Up to 4 LMCS APS’s can be used in a coded video sequence. Only 1 LMCS APS can be used for a picture.. The luma mapping model is signalled using piecewise linear model. The piecewise linear model partitions the input signal’s dynamic range into 16 equal pieces, and for each piece, its linear mapping parameters are expressed using the number of codewords assigned to that piece. Take 10-bit input as an example. Each of the 16 pieces will have 64 codewords assigned to it by default. The signalled number of codewords is used to calculate the scaling factor and adjust the mapping function accordingly for that piece. At the slice level, an LMCS enable flag is signalled to indicate if the LMCS process as depicted in Figure 51 is applied to the current slice. If LMCS is enabled for the current slice, an aps\_id is signalled in the slice header to identify the APS that carries the luma mapping parameters.

Each i-th piece, i = 0 … 15, of the *FwdMap* piecewise linear model is defined by two input pivot points InputPivot[] and two output (mapped) pivot points MappedPivot[].

The InputPivot[] and MappedPivot[] are computed as follows (assuming 10-bit video):

1. OrgCW = 64
2. For i = 0:16, InputPivot[ i ] = i \* OrgCW
3. For i=0:16, MappedPivot[i] is calculated as follows:   
   MappedPivot[ 0 ] = 0;  
   for( i = 0; i <16 ; i++)  
    MappedPivot[ i + 1 ] = MappedPivot[ i ] + SignalledCW[ i ]

where SignalledCW[ i ] is the signalled number of codewords for the i-th piece.

As shown in Figure 51, for an inter-coded block, motion compensated prediction is performed in the mapped domain. In other words, after the motion-compensated prediction block is calculated based on the reference signals in the DPB, the *FwdMap* function is applied to map the luma prediction block in the original domain to the mapped domain, . For an intra-coded block, the *FwdMap* function is not applied because intra prediction is performed in the mapped domain. After reconstructed block is calculated, the *InvMap* function is applied to convert the reconstructed luma values in the mapped domain back to the reconstructed luma values in the original domain (). The *InvMap* function is applied to both intra- and inter-coded luma blocks.

The luma mapping process (forward and/or inverse mapping) can be implemented using either look-up-tables (LUT) or using on-the-fly computation. If LUT is used, then and can be pre-calculated and pre-stored for use at the tile group level, and forward and inverse mapping can be simply implemented as and , respectively. Alternatively, on-the-fly computation may be used. Take forward mapping function *FwdMap* as an example. In order to figure out the piece to which a luma sample belongs, the sample value is right shifted by 6 bits (which corresponds to 16 equal pieces). Then, the linear model parameters for that piece are retrieved and applied on-the-fly to compute the mapped luma value. Let i be the piece index, a1, a2 be InputPivot[i] and InputPivot[i+1], respectively, and b1, b2 be MappedPivot[i] and MappedPivot[i+1], respectively. The FwdMap function is evaluated as follows:

(3-70)

The InvMap function can be computed on-the-fly in a similar manner. Generally, the pieces in the mapped domain are not equal sized, therefore the most straightforward inverse mapping process would require comparisons in order to figure out to which piece the current sample value belongs. Such comparisons increase decoder complexity. For this reason, VVC imposes a bistream constraint on the values of the output pivot points MappedPivot[ i ] as follows. Assume the range of the mapped domain (for 10-bit video, this range is [0, 1023]) is divided into 32 equal pieces. If MappedPivot[ i ] is not a multiple of 32, then MappedPivot[ i + 1 ] and MappedPivot[ i ] cannot belong to the same piece of the 32 equal-sized pieces. Thanks to such bitstream constraint, the InvMap function can also be carried out using a simple right bit-shift by 5 bits (which corresponds 32 equal-sized pieces) in order to figure out the piece to which the sample value belongs.

#### Luma-dependent chroma residual scaling

Chroma residual scaling is designed to compensate for the interaction between the luma signal and its corresponding chroma signals. Whether chroma residual scaling is enabled or not is also signalled at the slice level. If luma mapping is enabled, an additional flag is signalled to indicate if luma-dependent chroma residual scaling is enabled or not. When luma mapping is not used, luma-dependent chroma residual scaling is disabled. Further, luma-dependent chroma residual scaling is always disabled for the chroma blocks whose area is less than or equal to 4.

Chroma residual scaling depends on the average value of top and/or left reconstructed neighbouring luma samples of the current VPDU. If the current CU is inter 128x128, inter 128x64 and inter 64x128, then the chroma residual scaling factor derived for the CU associated with the first VPDU is used for all chroma transform blocks in that CU. Denote as the average of the reconstructed neighbouring luma samples (see Figure 51). The value of is computed in the following steps:

1. Find the index of the piecewise linear model to which belongs based on the *InvMap* function.

1. = cScaleInv[], where cScaleInv[] is a pre-computed 16-piece LUT.

Unlike luma mapping, which is performed on the sample basis, is a constant value for the entire chroma block. With , chroma residual scaling is applied as follows:

*Encoder side:*

*Decoder side:*

#### Encoder-side LMCS parameter estimation

A non-normative reference implementation is provided in the VTM6.0 encoder to estimate the LMCS model parameters. Because VTM anchors handle SDR, HDR PQ and HDR HLG differently, the reference algorithm in VTM6.0 is designed differently for SDR, HDR PQ and HDR HLG sequences. For SDR and HDR HLG sequences, the encoder algorithm is based on local luma variance and optimized for PSNR metrics. For HDR PQ sequences, the encoder algorithm is based on luma values and optimized for wPSNR (weighted PSNR) metrics.

##### LMCS parameter estimation for SDR

The basic idea of the VTM6.0 reference implementation for SDR is to assign pieces with more codewords to those dynamic range segments that have lower than average variance, and to assign fewer codewords to those dynamic range segments that have higher than average variance. In this way, smooth areas of the picture will be coded with more codewords than average, and vice versa. For SDR, VTM6.0 provides a reference algorithm and also configurable LMCS parameters for user tuning.

For SDR test sequences, the reference algorithm performs the following signal analysis:

1. Statistics of the input video are collected and analyzed assuming 10-bit internal coding bit-depth is used. If the internal coding bit-depth is not 10-bit, then bit-depth is first normalized to 10-bit.
2. Divide the dynamic range of [0, 1023] into 16 equal pieces.
3. For each luma sample location in the picture, the local spatial variance of luma sample values is calculated using a winSize x winSize () neighbourhood centered on the current position. Denote the specific piece (out of the 32 pieces) to which the current luma sample value belongs as *p*. This local variance is thus associated with the *p*-th piece.
4. For each of the 16 pieces, calculate the average local spatial variance (bin\_var)
5. For all valid pieces, an equal number of codewords per piece is allocated;

, where is the number of codewords allocated to the *i-*th piece, is the total number of codewords allowed, and and are the index values for the first and last valid piece, respectively.

1. The allocation of codewords is adjusted such that more codewords are allocated to pieces with lower average local spatial variance and fewer codewords are allocated to pieces with higher average local variance;

if

(3-71)

else if ,

(3-72)

where , , where is the average local spatial variance for the luma values in the *i-*th piece; is the mean of the average local spatial variances across all valid pieces; and is the percentage of samples in the *i-*th piece over the total number of samples, clipped to the range of [0, 0.4] to avoid aggressive codeword assignment.

1. If the total number of codewords allocated exceeds the maximum number of allowed codewords. the total number of codewords is reduced by equal amount starting from the first piece.
2. Adaptation decisions are made to set LMCS slice type, high bit rate, and chroma adjust adaptation parameters based on the relative histogram and average local spatial variances of the luma signal before and after reshaping. Slice type adaptation refers to enabling LMCS for the follow slice type combinations: intra and inter; pictures with temporal-ID 0 only; or inter only. High bit rate adaptation refers to adjusting the number of codewords allocated to pieces for QP values less than or equal to 22. Chroma adjustment adaptation refers to disabling or enabling chroma residue scaling. All these adaptation decisions are based on a series of threshold comparisons.
3. The SignalledCW [i] values are signalled in an LMCS APS.

When LMCS is applied, SSE is used for luma for intra (I) slices and weighted SSE is used for luma for inter (P or B) slices. The weight, w\_lmcs(k), is derived as follows based on the codeword assignment of the k-th piece in the piecewise linear model.

w\_lmcs[k] = (SignalledCW[k]/OrgCW)^2 (3-73)

SSE is always used for chroma mode decision.

In terms of picture-level decision whether to enable LMCS or not, different considerations are given to the different coding configurations. For the Random Access (RA) test conditions, picture analysis is performed for each IRAP picture as explained above. Then, if the average local spatial variance of the original picture is large, or if the average local variance of the mapped picture is large compared to the original, then LMCS is disabled for intra. For the other inter-coded pictures in the same IRAP period, if LMCS is enabled for the IRAP picture, then LMCS is enabled only for the pictures with temporal layer ID equal to 0. Otherwise, if LMCS is disabled for the IRAP picture, then LMCS is enabled for all the inter-coded pictures.

For All Intra (AI) and low delay (LD) test conditions, LMCS is enabled for all pictures. For AI, the LMCS parameter estimation is performed for all coded pictures, and the model parameters are updated in LMCS APS for all coded pictures. For LD, the LMCS parameters are estimated at every second interval, and the model parameters are updated in the LMCS APS at the instances of those pictures.

##### LMCS parameter estimation for HDR

In the JVET HDR CTC, two types of HDR sequences are included: PQ and HLG [3]. These two types of sequences are treated differently in the VTM reference encoder. For the PQ sequences, the VTM reference encoder applies luma-based QP adaptation and allows the QP value to vary spatially [3]. For the HLG sequences, static quantization is used [3]. Correspondingly, LMCS is applied differently for these two types of sequences as well. For PQ, LMCS is applied using a default LMCS mapping fucntion calculated as explained below. For HLG, LMCS parameter estimation algorithm similar to that for SDR is applied.

The VTM reference encoder uses wPSNR (weighted PSNR) instead of the conventional PSNR as an objective quality metric in the HDR PQ CTC [3]. The default HDR PQ LMCS curve is calculated to match the dQP function to maximize the wPSNR metric.

The luma-based QP adaptation derives a local delta QP (dQP) value per CTU based on the average of luma sample values:

dQP(Y) = max(-3, min(6, 0.015\*Y - 1.5 – 6 ) ) (3-74)

where Y is the average luma value, , maxY=1023 for 10-bit video [8]. The weight (W\_SSE) used in wPSNR calculation is derived based on dQP values:

W\_SSE(Y) = 2^(dQP(Y)/3) (3-75)

The default LMCS curve is calculated based on luma sample value as follows:

1. Compute the slope of the reshaping curve: slope[Y] = sqrt(W\_SSE(Y)) = 2^(dQP(Y)/6).
2. If signal is in narrow range (also called a standard range) [8], set slope[Y] = 0 for , or .
3. Calculate F[Y] by integrating slope[Y], F[Y+1] = F[Y] + slope[Y], Y =0…maxY-1
4. *FwdLUT*[Y] is calculated by normalizing F[Y] to [0 maxY], *FwdLUT*[Y] = clip3(0, maxY, round(F[Y]\*maxY/F[maxY]))
5. Calculate the number of codewords for the 16 equal pieces SignalledCW[i], i=0…15, as follows;

SignalledCW[15] = *FwdLUT*[1023] – *FwdLUT*[960];

for( i = 14; i >=0 ; i – –)

SignalledCW[ i  ] = FwdLUT[(i + 1) \* OrgCW] – FwdLUT[i \* OrgCW];

In terms of rate distortion optimized mode decision at the encoder, when LMCS is applied, for an intra (I) slice, SSE is used for luma and weighted SSE is used for chroma as the distortion measure. For an inter (P or B) slice, weighted SSE is used for both luma and chroma. LMCS is applied to all slices.

## 360-degree video coding tools

### Horizontal wrap around motion compensation

The horizontal wrap around motion compensation in the VTM6 is a 360-specific coding tool designed to improve the visual quality of reconstructed 360-degree video in the equi-rectangular (ERP) projection format [5]. In conventional motion compensation, when a motion vector refers to samples beyond the picture boundaries of the reference picture, repetitive padding is applied to derive the values of the out-of-bounds samples by copying from those nearest neighbors on the corresponding picture boundary. For 360-degree video, this method of repetitive padding is not suitable, and could cause visual artefacts called “seam artefacts” in a reconstructed viewport video. Because a 360-degree video is captured on a sphere and inherently has no “boundary,” the reference samples that are out of the boundaries of a reference picture in the projected domain can always be obtained from neighboring samples in the spherical domain. For a general projection format, it may be difficult to derive the corresponding neighboring samples in the spherical domain, because it involves 2D-to-3D and 3D-to-2D coordinate conversion [5], as well as sample interpolation for fractional sample positions. This problem is much simpler for the left and right boundaries of the ERP projection format, as the spherical neighbors outside of the left picture boundary can be obtained from samples inside the right picture boundary, and vice versa. Given the wide usage of the ERP projection format, and the relative ease of implementation, the horizontal wrap around motion compensation was adopted in the VTM6 to improve the visual quality of 360-video coded in the ERP projection format.



**Figure 52 – Horizontal wrap around motion compensation in VVC**

The horizontal wrap around motion compensation process is as depicted in Figure 52. When a part of the reference block is outside of the reference picture’s left (or right) boundary in the projected domain, instead of repetitive padding, the “out-of-boundary” part is taken from the corresponding spherical neighbors that are located within the reference picture toward the right (or left) boundary in the projected domain. Repetitive padding is only used for the top and bottom picture boundaries. As depicted in Figure 52, the horizontal wrap around motion compensation can be combined with the non-normative padding method often used in 360-degree video coding (see padded ERP in [5]). In VVC, this is achieved by signaling a high level syntax element to indicate the wrap-around offset, which should be set to the ERP picture width before padding; this syntax is used to adjust the position of horizontal wrap around accordingly. This syntax is not affected by the specific amount of padding on the left and right picture boundaries, and therefore naturally supports asymmetric padding of the ERP picture, i.e., when left and right padding are different. The horizontal wrap around motion compensation provides more meaningful information for motion compensation when the reference samples are outside of the reference picture’s left and right boundaries. Under the 360 video CTC [4], this tool improves compression performance not only in terms of rate-distortion performance, but also in terms of reduced seam artefacts and improved subjective quality of the reconstructed 360-degree video. The horizontal wrap around motion compensation can also be used for other single face projection formats with constant sampling density in the horizontal direction, such as adjusted equal-area projection in 360Lib [5].

## Screen content coding tools

### Intra block copy (IBC)

Intra block copy (IBC) is a tool adopted in HEVC extensions on SCC. It is well known that it significantly improves the coding efficiency of screen content materials. Since IBC mode is implemented as a block level coding mode, block matching (BM) is performed at the encoder to find the optimal block vector (or motion vector) for each CU. Here, a block vector is used to indicate the displacement from the current block to a reference block, which is already reconstructed inside the current picture. The luma block vector of an IBC-coded CU is in integer precision. The chroma block vector rounds to integer precision as well. When combined with AMVR, the IBC mode can switch between 1-pel and 4-pel motion vector precisions. An IBC-coded CU is treated as the third prediction mode other than intra or inter prediction modes. The IBC mode is applicable to the CUs with both width and height smaller than or equal to 64 luma samples.

At the encoder side, hash-based motion estimation is performed for IBC. The encoder performs RD check for blocks with either width or height no larger than 16 luma samples. For non-merge mode, the block vector search is performed using hash-based search first. If hash search does not return valid candidate, block matching based local search will be performed.

In the hash-based search, hash key matching (32-bit CRC) between the current block and a reference block is extended to all allowed block sizes. The hash key calculation for every position in the current picture is based on 4x4 subblocks. For the current block of a larger size, a hash key is determined to match that of the reference block when all the hash keys of all 4×4 subblocks match the hash keys in the corresponding reference locations. If hash keys of multiple reference blocks are found to match that of the current block, the block vector costs of each matched reference are calculated and the one with the minimum cost is selected.

In block matching search, the search range is set to cover both the previous and current CTUs.

At CU level, IBC mode is signalled with a flag and it can be signaled as IBC AMVP mode or IBC skip/merge mode as follows:

* IBC skip/merge mode: a merge candidate index is used to indicate which of the block vectors in the list from neighboring candidate IBC coded blocks is used to predict the current block. The merge list consists of spatial, HMVP, and pairwise candidates.
* IBC AMVP mode: block vector difference is coded in the same way as a motion vector difference. The block vector prediction method uses two candidates as predictors, one from left neighbor and one from above neighbor (if IBC coded). When either neighbor is not available, a default block vector will be used as a predictor. A flag is signaled to indicate the block vector predictor index.

#### IBC reference region

To reduce memory consumption and decoder complexity, the IBC in VTM6 allows only the reconstructed portion of the predefined area including the region of current CTU and some region of the left CTU. Figure 53 illustrates the reference region of IBC Mode, where each block represents 64x64 luma sample unit.

X

Curr

X

X

Curr

X

X

X

X

Curr

X

X

Curr

X

**Figure 53 –current CTU processing order and its available reference samples in current and left CTU**

Depending on the location of the current coding CU location within the current CTU, the following applies:

* If current block falls into the top-left 64x64 block of the current CTU, then in addition to the already reconstructed samples in the current CTU, it can also refer to the reference samples in the bottom-right 64x64 blocks of the left CTU, using CPR mode. The current block can also refer to the reference samples in the bottom-left 64x64 block of the left CTU and the reference samples in the top-right 64x64 block of the left CTU, using CPR mode.
* If current block falls into the top-right 64x64 block of the current CTU, then in addition to the already reconstructed samples in the current CTU, if luma location (0, 64) relative to the current CTU has not yet been reconstructed, the current block can also refer to the reference samples in the bottom-left 64x64 block and bottom-right 64x64 block of the left CTU, using CPR mode; otherwise, the current block can also refer to reference samples in bottom-right 64x64 block of the left CTU.
* If current block falls into the bottom-left 64x64 block of the current CTU, then in addition to the already reconstructed samples in the current CTU, if luma location (64, 0) relative to the current CTU has not yet been reconstructed, the current block can also refer to the reference samples in the top-right 64x64 block and bottom-right 64x64 block of the left CTU, using CPR mode. Otherwise, the current block can also refer to the reference samples in the bottom-right 64x64 block of the left CTU, using CPR mode.
* If current block falls into the bottom-right 64x64 block of the current CTU, it can only refer to the already reconstructed samples in the current CTU, using CPR mode.

This restriction allows the IBC mode to be implemented using local on-chip memory for hardware implementations.

#### IBC interaction with other coding tools

The interaction between IBC mode and other inter coding tools in VTM6, such as pairwise merge candidate, history based motion vector predictor (HMVP), combined intra/inter prediction mode (CIIP), merge mode with motion vector difference (MMVD), and triangle partition are as follows:

* IBC can be used with pairwise merge candidate and HMVP. A new pairwise IBC merge candidate can be generated by averaging two IBC merge candidates. For HMVP, IBC motion is inserted into history buffer for future referencing.
* IBC cannot be used in combination with the following inter tools: affine motion, CIIP, MMVD, and triangle partition.
* IBC is not allowed for the chroma coding blocks when DUAL\_TREE partition is used.

Unlike in the HEVC screen content coding extension, the current picture is no longer included as one of the reference pictures in the reference picture list 0 for IBC prediction. The derivation process of motion vectors for IBC mode excludes all neighboring blocks in inter mode and vice versa. The following IBC design aspects are applied:

* IBC shares the same process as in regular MV merge including with pairwise merge candidate and history based motion predictor, but disallows TMVP and zero vector because they are invalid for IBC mode.
* Separate HMVP buffer (5 candidates each) is used for conventional MV and IBC.
* Block vector constraints are implemented in the form of bitstream conformance constraint, the encoder needs to ensure that no invalid vectors are present in the bitsream, and merge shall not be used if the merge candidate is invalid (out of range or 0). Such bitstream conformance constraint is expressed in terms of a virtual buffer as described below.
* For deblocking, IBC is handled as inter mode.
* If the current block is coded using IBC prediction mode, AMVR does not use quarter-pel; instead, AMVR is signaled to only indicate whether MV is inter-pel or 4 integer-pel.
* The number of IBC merge candidates can be signalled in the slice header separately from the numbers of regular, subblock, and triangle merge candidates.

A virtual buffer concept is used to describe the allowable reference region for IBC prediction mode and valid block vectors. Denote CTU size as ctbSize, the virtual buffer, ibcBuf, has width being wIbcBuf = 128\*128/ctbSize and height hIbcBuf = ctbSize. For example, for a CTU size of 128x128, the size of ibcBuf is also 128x128; for a CTU size of 64x64, the size of ibcBuf is 256x64; and a CTU size of 32x32, the size of ibcBuf is 512x32.

The size of a VPDU is min(ctbSize, 64) in each dimension, Wv = min(ctbSize, 64).

The virtual IBC buffer, ibcBuf is maintained as follows.

1. At the beginning of decoding each CTU row, refresh the whole ibcBuf with an invalid value -1.
2. At the beginning of decoding a VPDU (xVPDU, yVPDU) relative to the top-left corner of the picture, set the ibcBuf[ x ][ y ] = -1, with x = xVPDU%wIbcBuf, …, xVPDU% wIbcBuf + Wv – 1; y = yVPDU%ctbSize, …, yVPDU%ctbSize + Wv – 1.
3. After decoding a CU contains (x, y) relative to the top-left corner of the picture, set

ibcBuf[ x % wIbcBuf ][ y % ctbSize ] = recSample[ x ][ y ]

For a block covering the coordinates (x, y), if the following is true for a block vector *bv = (bv[0], bv[1])*, then it is valid; otherwise, it is not valid:

*ibcBuf[ (x + bv[0])% wIbcBuf] [ (y + bv[1]) % ctbSize ] shall not be equal to -1.*

### Block differential pulse coded modulation (BDPCM)

VTM6 supports block differential pulse coded modulation (BBDPCM) for screen content coding. At the sequence level, a BDPCM enable flag is signalled in the SPS; this flag is signalled only if the transform skip mode (described in the next section) is enabled in the SPS.

When BDPCM is enabled, a flag is transmitted at the CU level if the CU size is smaller than or equal to MaxTsSize by MaxTsSize in terms of luma samples and if the CU is intra coded, where MaxTsSize is the maximum block size for which the transform skip mode is allowed. This flag indicates whether regular intra coding or BDPCM is used. If BDPCM is used, a BDPCM prediction direction flag is transmitted to indicate whether the prediction is horizontal or vertical. Then, the block is predicted using the regular horizontal or vertical intra prediction process with unfiltered reference samples. The residual is quantized and the difference between each quantized residual and its predictor, i.e. the previously coded residual of the horizontal or vertical (depending on the BDPCM prediction direction) neighbouring position, is coded.

For a block of size M (height) × N (width), let be the prediction residual. Let denote the quantized version of the residual . BDPCM is applied to the quantized residual values, resulting in a modified M × N array with elements , where is predicted from its neighboring quantized residual value. For vertical BDPCM prediction mode, for , the following is used to derive :

(3-74)

For horizontal BDPCM prediction mode, for , the following is used to derive :

(3-75)

At the decoder side, the above process is reversed to compute , as follows:

if vertical BDPCM is used (3-76)

if horizontal BDPCM is used (3-77)

The inverse quantized residuals, , are added to the intra block prediction values to produce the reconstructed sample values.

The predicted quantized residual values are sent to the decoder using the same residual coding process as that in transform skip mode residual coding. In terms of the MPM mode for future intra mode coding, horizontal or vertical prediction mode is stored for a BDPCM-coded CU if the BDPCM prediction direction is horizontal or vertical, respectively. For deblocking, if both blocks on the sides of a block boundary are coded using BDPCM, then that particular block boundary is not deblocked.

### Residual coding for transform skip mode

VTM6 allows the transform skip mode to be used for luma blocks of size up to MaxTsSize by MaxTsSize, where the value of MaxTsSize is signaled in the PPS and can be at most 32 in VTM6. When a CU is coded in transform skip mode, its prediction residual is quantized and coded using the transform skip residual coding process. This process is modified from the transform coefficient coding process described in 3.6.2. In transform skip mode, the residuals of a TU are also coded in units of non-overlapped subblocks of size 4x4. For better coding efficiency, some modifications are made to customize the residual coding process towards the residual signal’s characteristics. The following summarizes the differences between transform skip residual coding and regular transform residual coding in VTM6:

* + - Forward scanning order is applied to scan the subblocks within a transform block and also the positions within a subblock;
    - no signalling of the last (x, y) position;
    - coded\_sub\_block\_flag is coded for every subblock except for the last subblock when all previous flags are equal to 0;
    - sig\_coeff\_flag context modelling uses a reduced template, and context model of sig\_coeff\_flag depends on top and left neighbouring values;
    - context model of abs\_level\_gt1 flag also depends on the left and top sig\_coeff\_flag context modelling uses a reduced template, and context model of sig\_coeff\_flag depends on top and left neighbouring values;
    - values.
    - par\_level\_flag using only one context model;
    - additional greater than 3, 5, 7, 9 flags are signalled to indicate the coefficient level, one context for each flag;
    - modified rice parameter derivation for the binarization of the remainder values;
    - context model of the sign flag is determined based on left and above neighbouring values and the sign flag is parsed after sig\_coeff\_flag to keep all context coded bins together;

For each subblock, if the coded\_subblock\_flag is equal to 1 (i.e., there is at least one non-zero quantized residual in the subblock), coding of the quantized residual levels is performed in three scan passes (see Figure 54):

* **First scan pass**: significance flag (sig\_coeff\_flag), sign flag (coeff\_sign\_flag), absolute level greater than 1 flag (abs\_level\_gtx\_flag[0]), and parity (par\_level\_flag) are coded. For a given scan position, if sig\_coeff\_flag is equal to 1, then coeff\_sign\_flag is coded, followed by the abs\_level\_gtx\_flag[0] (which specifies whether the absolute level is greater than 1). If abs\_level\_gtx\_flag[0] is equal to 1, then the par\_level\_flag is additionally coded to specify the parity of the absolute level.
* **Greater than x scan pass**: for each scan position whose absolute level is greater than 1, up to four abs\_level\_gtx\_flag[i] for i = 1...4 are coded to indicate if the absolute level at the given position is greater than 3, 5, 7, or 9, respectively.
* **Remainder scan pass**: The remainder of the absolute level are coded for all scan positions with abs\_level\_gtx\_flag[4] equal to 1 (that is, the absolute level is greater than 9). The remainder of the absolute levels are binarized using reduced rice parameter derivation template.

The bins in scan passes #1 and #2 (the first scan pass and the greater than x scan pass) are context coded until the maximum number of context coded bins in the TU have been exhausted. The maximum number of context coded bins in a residual block is limited to 2\*block\_width\*block\_height, or equivalently, 2 context coded bins per sample position on average. The bins in the last scan pass (the remainder scan pass) are bypass coded.

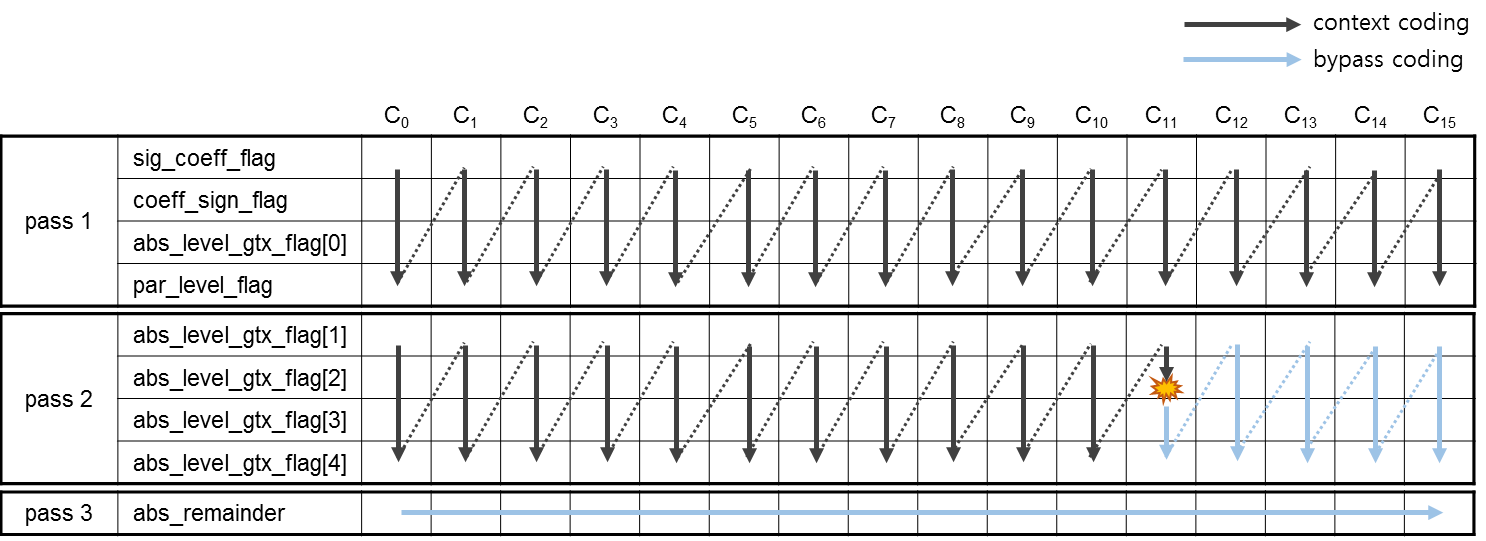


Figure 54 – residual coding passes for transform skip blocks

Further, for a block not coded in the BDPCM mode, a level mapping mechanism is applied to transform skip residual coding. Level mapping uses the top and left neighbouring coefficient levels to predict the current coefficient level in order to reduce signalling cost. For a given residual position, denote *absCoeff* as the absolute coefficient level before mapping and *absCoeffMod* as the coefficient level after mapping. Let X0 denote the absolute coefficient level of the left neighbouring position and let X1 denote the absolute coefficient level of the above neighbouring position. The level mapping is performed as follows:

pred = max(X0, X1);

if (absCoeff == pred)

absCoeffMod = 1;

else

absCoeffMod = (absCoeff < pred) ? absCoeff + 1 : absCoeff;

### Palette mode

VTM6 supports the palette mode to be used for screen content coding in 4:4:4 color format. When palette mode is enabled, a flag is transmitted at the CU level if the CU size is smaller than or equal to 64x64 indicating whether palette mode is used. A palette coded coding unit (CU) is treated as a prediction mode other than intra prediction, inter prediction, and intra block copy (IBC) mode.

If the palette mode is utilized, the sample values in the CU are represented by a small set of representative colour values. The set is referred to as the palette. For pixels with values close to the palette colors, the palette indices are signalled. It is also possible to specify a sample that is outside the palette by signalling an escape symbol followed by quantized component values. This is illustrated in Figure 55.



Figure 55: Example of a block coded in palette mode

For coding of the palette, a palette predictor is maintained. The predictor is initialized to 0 at the beginning of each slice for non-wavefront case and at the beginning of each CTU row for wavefront case. For each entry in the palette predictor, a reuse flag is signalled to indicate whether it is part of the current palette in the CU. The reuse flags are sent using run-length coding of zeros. After this, the number of new palette entries and the component values for the new palette entries are signalled. After encoding the palette coded CU, the palette predictor will be updated using the current palette, and entries from the previous palette predictor that are not reused in the current palette will be added at the end of the new palette predictor until the maximum size allowed is reached. An escape flag is signaled for each CU to indicate if escape symbols are present in the current CU. If escape symbols are present, the palette table is augmented by one and the last index is assigned to be the escape symbol.

Palette indices of samples in a CU form a palette index map. The index map is coded using horizontal and vertical traverse scans as shown in Figure 56. The scan order is explicitly signalled in the bitstream using the palette\_transpose\_flag.



Figure 56: Horizontal and vertical traverse scans

The palette indices are coded using two main palette sample modes: 'INDEX' and 'COPY\_ABOVE'. The mode is signalled using a flag except for the top row when horizontal scan is used, the first column when the vertical scan is used, or when the previous mode was 'COPY\_ABOVE'. In the 'COPY\_ABOVE' mode, the palette index of the sample in the row above is copied. In the 'INDEX' mode, the palette index is explicitly signalled. For both 'INDEX' and 'COPY\_ABOVE' modes, a run value is signalled which specifies the number pixels that are coded using the same mode.

The encoding order for index map is as follows: First, the number of index values associated to ‘INDEX’ runs is signalled. This is followed by signalling of the actual index values associated to ‘INDEX’ runs for the entire CU using truncated binary coding. Then the palette mode (INDEX or COPY\_ABOVE) and run length for each run are signalled in an interleaved manner. Finally, the quantized escape mode colors for the entire CU are grouped together and coded with exponential Golomb coding.

For slices with dual luma/chroma tree, palette is applied on luma (Y component) and chroma (Cb and Cr components) separately. For slices of single tree, palette will be applied on Y, Cb, Cr components jointly, i.e., each entry in the palette contains Y, Cb, Cr values.

For deblocking, the palette coded block on the sides of a block boundary is not deblocked.

# Description of VTM6 encoder and encoding methods

## Derivation process of coding tree structure

To be added.

## Hash based motion estimation for screen content coding

VTM6 uses hash-based motion estimation to handle the sometimes large and irregular motion in screen content. For each reference picture, hash tables corresponding to 4x4 to 64x64 block sizes are generated using a bottom-up approach as follows:

* For each 2x2 block, the block hash value is calculated directly from the original sample values (luma samples are used if 4:2:0 chroma format and both luma and chroma sample values are used if 4:4:4 chroma format). The cyclic redundancy check (CRC) value is used as the hash value.
* For 4x4, 8x8, 16x16, 32x32 and 64x64 blocks, the hash value of the current block is the CRC value calculated from the CRC values of its four subblocks.

To enable efficient search for matched blocks, the structure of inverted index is used, where hash values are used as to index into a table, and the table entries contain all the blocks with the same hash value as the corresponding table index. The blocks corresponding a given table index are stored as a linked list. Two CRC values, one 16-bit hash and the other 24-bit hash, are calculated for each block. The two hash values are calculated in a similar way but using different CRC truncated polynomials. The first 16-bit CRC value is used as the inverted index. The second 24-bit hash value is stored together with the blocks to resolve hash conflicts in the case more than one matching blocks are found. To reduce the length of the hash table, the hash values of all “simple” blocks (defined as a block with only one sample value in each row or column) are excluded from the hash table.

In motion estimation, if the current block is a square block (except for 128x128 blocks), its hash values are calculated. Then, the encoder queries the corresponding hash table. If hash match is found, the matched block is used as the reference. If the current block is a rectangle block of size NxM (and without loss of generality assume M > N), it will be divided into several non-overlapping square subblocks of size NxN. An example is shown in Figure 57. The encoder will find the first non-simple square subblock and calculate its hash values. Encoder queries the hash values of this NxN square subblock on the hash table corresponding to NxN block size. The one or more matched reference blocks are considered reference block candidates. For each matched reference block candidate, encoder will continue to check whether the hash values of the remaining square subblocks (namely the white region that follows the first non-simple square subblock depicted in Figure 57) are equal to those of the square subblocks adjacent to that reference block candidate. If the hash values of all square subblocks are matched, the reference block candidate will be regarded as a valid reference block.

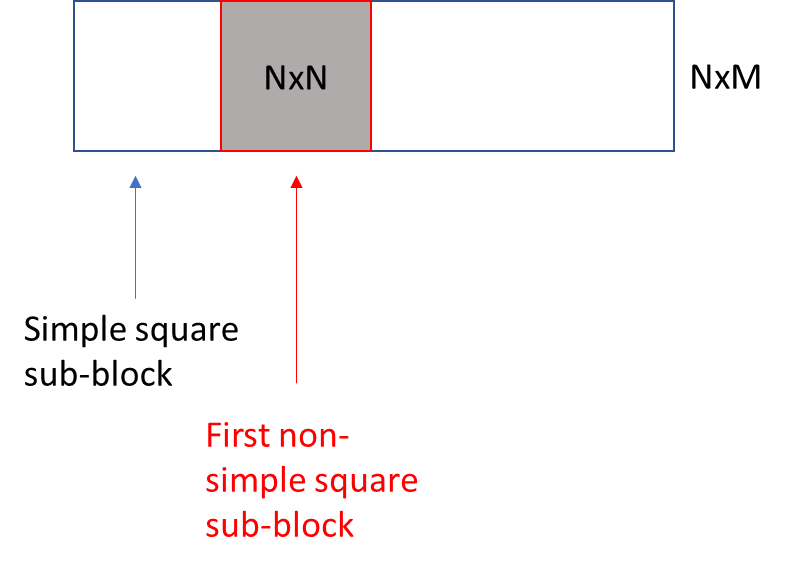


Figure 57 Motion estimation for rectangular block with hash values for square subblocks.

For inter coding, the hash-based motion search is performed before testing all coding modes. In addition, encoder will reuse the MVs of the hash mode as the starting point candidates in the normal motion estimation process. If the hash-based motion vector exists, which indicates that the block most likely contains screen content, fractional motion estimation is skipped.

To accelerate the encoder, coding modes other than the skip and merge part of ETM\_MERGE\_SKIP, ETM\_AFFINE, and ETM\_MERGE\_TRIANGLE modes and finer-granularity block splitting are skipped if all of the following conditions are satisfied:

* Current block size is 64x64, 128x64 or 64x128.
* An identical reference block is found in a reference picture.
* The QP of reference picture is not larger than that of current picture.

# References

1. B. Bross, J. Chen, S. Liu, “Versatile Video Coding (Draft 6),” document JVET-O2001, 15th JVET meeting: Gothenburg, SE, 3–12 July 2019.
2. F. Bossen, J. Boyce, X. Li, and V. Seregin, K. Sühring, “JVET common test conditions and software reference configurations for SDR video,” document JVET-N1010, 14th JVET meeting: Geneva, MA, 19–27 Mar. 2019.
3. A. Segall, E. François, W. Husak, S. Iwamura, D. Rusanovskyy, “JVET common test conditions and evaluation procedures for HDR/WCG video,” document JVET-O2011, 15th JVET meeting: Gothenburg, SE, 3–12 July 2019.
4. P. Hanhart, J. Boyce, and K. Choi, “JVET common test conditions and evaluation procedures for 360° video,” document JVET-K1012, 11th JVET meeting: Macao, CN, 3–12 Oct. 2018.
5. Y. Ye, J. Boyce, “Algorithm descriptions of projection format conversion and video quality metrics in 360Lib Version 7,” document JVET-M1004, 13th JVET meeting: Marrakech, MA, 9–18 Jan. 2019.
6. High Efficiency Video Coding (HEVC), Rec. ITU-T H.265 and ISO/IEC 23008-2, Jan. 2013 (and later editions).
7. G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard”, *IEEE Trans. Circuits and Systems for Video Technology*, Vol. 22, No. 12, pp. 1649‒1668, Dec. 2012.
8. Supplement ITU-T H.Sup15 (2017), Conversion and Coding Practices for HDR/WCG Y′CbCr 4:2:0 Video with PQ Transfer Characteristics