Joint Video Experts Team (JVET)
of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29

Document: JVET-AD2002-v1

30th Meeting, Antalya, TR, 21-28 April 2023

Title:	**Algorithm description for Versatile Video Coding and Test Model 20 (VTM20)**
Status:	Output document of JVET
Purpose:	Algorithm description for Versatile Video Coding and Test Model 20

Author(s) or Contact(s):
Adrian Browne — Email: adrian.browne@sony.com
Yan Ye — Email: Yan.Ye@alibaba-inc.com
Seung Hwan Kim — Email: seunghwan3.kim@lge.com

Source: Editors

Abstract

The JVET established the VVC Test Model 21 (VTM21) software at its 30th meeting (21-28 April 2023, Antalya, TR). This document serves as a source of general tutorial information on the VVC design and also provides an algorithm description and encoding method description of VTM21 software. It is noted that, as no update of the algorithm description document was released at the 27th JVET meeting, the numbering of the VTM software version is one higher than that of this description document. In the main body of the text, numbering refers to the software version. 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 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. In addition to the applications that have commonly been addressed by prior video coding standards, some key application areas for the use of this standard include in particular 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.

Ed. Notes:

VVC Test Model 20 (VTM20) algorithm description and encoding method v1

- General improvements to reference picture resampling sections
- Incorporated JVET-AD0169: AHG12: RPR filters for scale factors below 1.5x
- Incorporated JVET-AD0045: AHG10: Encoder MV selections and DMVR revisited

VVC Test Model 19 (VTM19) algorithm description and encoding method v1

- General improvements to neural network-based post filtering SEI messages (section 4.4) to bring the section up to date with the current filtering purposes of the NNPFC and NNPFA messages
- Incorporated JVET-AC0096: AHG10/12: Suggestion for new CTC for RPR in VTM and ECM

VVC Test Model 18 (VTM19) algorithm description and encoding method v2

- Incorporated JVET-AB0072: VTM Encoder Implementation for Green-MPEG SEI Messaging

VVC Test Model 18 (VTM19) algorithm description and encoding method v1

- Incorporated JVET-AA0110 and JVET-AB0267: phase indication SEI
- Incorporated JVET-AB0070: post-filter hint SEI
- Incorporated JVET-AA0102, JVET-AA0101, JVET-AB0051, and JVET-AB0069: processing order SEI
- MTS signaling bug fix in section 3.5.2
- Incorporated JVET-AB0080 and JVET-AB0081; RPR controls and filters
- Incorporated JVET-V0078; QP control for very smooth blocks

VVC Test Model 17 (VTM17) algorithm description and encoding method v2

- Incorporated JVET-Z0047: AHG13: Improvements of film grain analysis
- Incorporated JVET-Z0244: AHG9: NN post-filter SEI

VVC Test Model 17 (VTM17) algorithm description and encoding method v1

- Incorporated JVET-Z0046: VTM Software Implementation for GREEN-MPEG SEI Messaging
- Incorporated JVET-Z0072: AHG10/AHG12: Enhanced reference picture structures for ECM and VTM
- Incorporated JVET-Z0099: AHG10: Deblocking in RDO and beta offset minus 2 for VTM
- Incorporated JVET-Z0111: Adaptively bypass affine ME in VTM
- Incorporated JVET-Z0120: AHG9: Shutter interval information SEI message for VSEI
- General editorial improvements throughout

VVC Test Model 16 (VTM16) algorithm description and encoding method

- Added description of rate control in VTM, incorporated JVET-Y0105: AHG10: An improved VVC rate control scheme
- Incorporated JVET-Y0155: AHG10: Fixes and clean up for temporal prefilter
- Incorporated JVET-Y0077: AHG10: Block importance mapping

VVC Test Model 15 (VTM15) algorithm description and encoding method

- Incorporated JVET-X0128: AHG8: On History-Based Rice Parameter Derivations for Wavefront Parallel Processing
- Incorporated JVET-X0048 CE: Film Grain Synthesis (test CE2.1 and CE2.2)
- Incorporated JVET-X0143 AHG10: VTM Encoder Changes for ALF Usage with Subpicture

VVC Test Model 14 (VTM14) algorithm description and encoding method

- Incorporated JVET-V0047: CE-3.1 and CE-3.2: Transform coefficients range extension for high bit-depth coding
- Incorporated JVET-V0054: CE-2.1: Slice based Rice parameter selection for transform skip residual coding
- Incorporated JVET-V0106: CE-related: On history-enhanced method of Rice parameter derivation for regular residual coding (RRC) at high bit depths
- Incorporated JVET-W0046: CE-1.1: coding of last significant coefficient position for high bit depth and high bit rate extensions
- Incorporated JVET-W0136: Suggested initial profile text for VVC operation range extension

VVC Test Model 13 (VTM13) algorithm description and encoding method

- Updated the description of motion compensated temporal pre-filtering (MCTF)
- Typo Fixes

VVC Test Model 12 (VTM12) algorithm description and encoding method

- Refinement of high precision (1/16 pel) motion compensation and motion vector storage
- Added reference picture resampling

VVC Test Model 11 (VTM11) algorithm description and encoding method

- General editorial improvements
- Added description of encoder configuration with GOP size 32 in random access and GOP size 8 in low delay
- Added description of motion compensated temporal pre-filtering (MCTF)
- Editorial fixes in various sections (introduction, wrap around motion compensation, VTM encoder, etc).
- Fixes in block partitioning signalling and picture boundary forced partitioning

VVC Test Model 10 (VTM10) algorithm description and encoding method

- Updated the description of a number of intra and inter coding tools to match the VVC FDIS spec and improve clarity
- General editorial improvements

VVC Test Model 9 (VTM9) algorithm description and encoding method

- Added description of Profiles, Levels and Tiers.

VVC Test Model 8 (VTM8) algorithm description and encoding method

- Editorial improvements of the JCCR mode

- Added encoder algorithm description for the palette mode, as well as editorial improvements of the palette mode
- Incorporated JVET-Q0291: CE2-related: On maximum palette size of VVC
- Incorporated JVET-Q0503: CE2-related: Encoder improvement for palette mode
- Incorporated JVET-Q0504: CE2-related: Palette mode for non 4:4:4 color format
- Incorporated JVET-Q0712: Non-CE2: Extension of error limit table in JVET-Q0503 to high QP
- Incorporated JVET-Q0493: Non-CE2: Palette encoder improvements for lossless coding
- Incorporated JVET-Q0695: Combined encoder improvements of JVET-Q0101/JVET-Q0408/JVET-Q0514 on JCCR with chroma transform skip
- Incorporated JVET-Q0054: CE1-1.1: Fixes for long luma deblocking filter decision
- Incorporated JVET-Q0150: Fix for ALF virtual boundary processing
- Incorporated JVET-Q0806: Geometric partitioning mode
- Incorporated JVET-Q0495: Clip ranges for NL-ALF
- Incorporated JVET-Q0297: Merge estimation regions
- Incorporated JVET-Q0330: Use QT splitting if other methods are not allowed for picture boundary implicit partitioning
- Incorporated JVET-Q0795: Cross component adaptive loop filter (CC-ALF)
- Incorporated JVET-Q0491: Palette escape binarization
- Incorporated JVET-Q0501: Palette predictor initialization in WPP
- Incorporated JVET-Q0629: Palette mode excluding small blocks
- Incorporated JVET-Q0293: Removal of chroma Nx2 blocks in PDPC
- Incorporated JVET-Q0820: ACT common text for bug fixes and transform change
- Incorporated JVET-Q0516: MTS signaling based on last significant coefficient position
- Incorporated JVET-Q0784: LFNST signalling, latency reduction and scaling process
- Incorporated JVET-Q0512: Enable TS for ACT blocks
- Incorporated JVET-Q0089: TSRC slice-level switch and BDPCM for chroma in 4:2:0
- Incorporated JVET-Q0267: Resetting CuQpOffsets to 0 at the start of each QG

VVC Test Model 7 (VTM7) algorithm description and encoding methodIncorporated

- JVET-O0683: adaptive color transform
- Incorporated JVET-P1000: Transform shift removal in transform skip mode
- Incorporated JVET-P1026: Applying LFNST for ISP Blocks
- Incorporated JVET-P0983: Removal of sps_sbt_max_size_64_flag
- Incorporated JVET-P0058: Enabling transform skip for chroma
- Incorporated JVET-P0168: Removal of 2x2 chroma quantization matrices
- Incorporated JVET-P0365: Disabling scaling matrices for LFNST coded blocks
- Incorporated JVET-P1034: Improved coding of user defined quantization matrices
- Incorporated JVET-P0436: On CU adaptive chroma QP offset signalling
- Incorporated JVET-P0054: Fixed MIP up-sampling order
- Incorporated JVET-P0054: Align MIP matrix multiplication process
- Incorporated JVET-P0803: MIP cleanup
- Incorporated JVET-P0408: MRL to use the same 3 lines as CCLM
- Incorporated JVET-P0059: Enable BDPCM for chroma
- Incorporated JVET-P0077: Line-based Palette mode
- Incorporated JVET-O1038: ALF boundary padding
- Incorporated JVET-P0505: Fixing non-linear ALF clipping values for 8-bit video
- Incorporated JVET-P0254: Fix number of LMCS segments to 32 regardless of bit depth
- Incorporated JVET-P0371: Signalling of corrective values for chroma residual scaling
- Remove bricks
- Added subpicture
- Incorporated JVET-P0325: Change the checking order of the first two spatial merge candidates
- Incorporated JVET-P0057: 1/32-pel precision of PROF motion refinement

- Incorporated JVET-P1023: Reference picture conditions in DMVR and BDOF
- Added virtual boundary for loop filter disabling
- Incorporated JVET-P0641: Removal of 2xN chroma intra blocks
- Incorporated JVET-P0072: JVET-P0298, JVET-P0562, changes related to transform skip residual coding
- Incorporated JVET-P0170: on regular residual coding: Simplified derivation of ZeroPos[n]

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-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: 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 SbTMVP 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

1 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 improve the coding performance.

From the 13th meeting (9–18 January 2019, Marrakech, MA), the JVET experts started 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 a 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. Support for coding of 4:2:2 and 4:4:4 chroma formats were added in addition to 4:2:0 chroma format.

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, support for some new functionalities, e.g. reference picture resampling (also known as adaptive resolution change), scalability coding was added.

At the 16th Meeting (1–11 Oct. 2019, Geneva, CH) the Versatile Video Coding (VVC) working draft 7 and the VVC Test Model 7 (VTM7) algorithm description and encoding method were established.

In the second part of the development of VVC version 1, between VTM6 and VTM10, the JVET group mainly focused on bug fixes, harmonization and clean-up of the existing coding features, and the completion of high level syntax design in order to successfully finalize the standard with the planned timeline. Only three coding tools, namely geometric partition, cross-colour adaptive loop filter and adaptive color transform, were added to the VVC version 1 during this period of time.

At the 19th JVET meeting (22 June – 1 July 2020), VVC version 1 specification was finalized. Normative features in VTM10 correspond to the finalized VVC version 1, and the non-normative features of VTM continued to evolve. Encoding features such as larger group of picture (GOP) sizes and motion compensated temporal pre-filtering (MCTF) were added to VTM11.

VTM13 and VTM14 added normative changes for VVC version 2 to support high bit depth and high frame rate applications. VTM13 added modifications for the precision of transforms, and Golomb-Rice coding modifications for both regular residual coding (RRC) and transform skip residual coding (TSRC). VTM14 added a modification which allows the signalling of the last significant coefficient in a transform block to be reversed. VTM15 added a normative modification to regular residual coding when wavefront parallel processing is enabled. The normative features in VTM15 correspond to the finalized VVC version 2.

At the 25th JVET meeting (12-21 January 2022), and the 26th JVET meeting (20-29 April 2022) no normative changes were made to VTM. However non-normative changes were made to rate control and the temporal pre-filter and included in VTM16. Further changes to motion estimation in the encoder and to the configurations for picture structures, rate distortion optimization were included in VTM17. In

addition three new SEI messages for green metadata, shutter interval information and neural network post filter information were adopted.

VTM18 and VTM19 added the phase indication SEI message, the post-filter hint SEI message, the SEI processing order SEI message, and enhancements to RPR. Updates to the neural network post-filter SEI messages and green metadata SEI message were also integrated.

2 Scope

The normative decoding process for version 1 of Versatile Video Coding is specified in the VVC text specification document [1] and refinement document [2]. The normative extensions to VVC version 1 included in VVC version 2 are specified in the operation range extensions document [3]. The VTM19.0 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 15, which serves as a tutorial for the algorithm and encoding model implemented in the VTM19.0 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 VTM19.0 software. 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-Y2010 [4]. Common test conditions specific to video content with high dynamic range and wide colour gamut are described in JVET-Y2011 [5]. Common test conditions specific to video content for 360° omnidirectional video applications are described in JVET-U1012 [6]. VTM supports the coding of video sequences of 4:4:4 and 4:2:2 colour formats. Common test conditions and reference software configuration specific to non-4:2:0 colour formats are described in JVET-T2013 [7]. Lossless coding design is also supported in VTM. Common test conditions and software reference configurations for lossless, near lossless, and mixed lossy/lossless coding are described in JVET-Q2014 [8]. 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-T1004[9] describes the algorithms used in 360Lib to process, code, and measure quality of 360° omnidirectional video.

Common test conditions for high bit depth and high bit rate applications, which form the operation range extensions in VVC version 2, are described in JVET-U2018 [10].

3 Algorithm description of Versatile Video Coding and Test Model

3.1 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 VTM19 encoder.

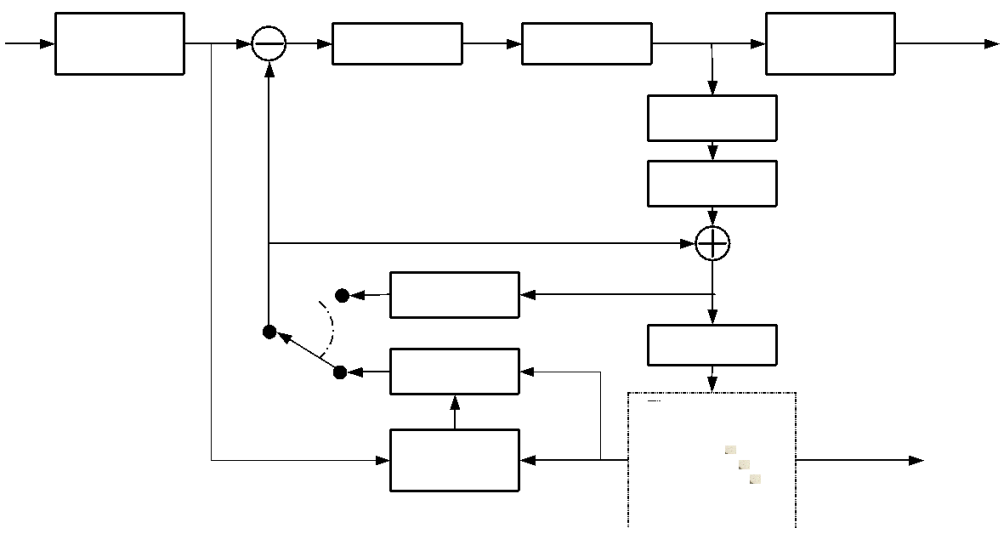


Figure 1 – General block diagram of VTM19 encoder

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 VVC, 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 3.9. The following features have been included in the VVC on top of the block 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
    - Geometric partitioning mode
    - 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
    - Reference picture resampling
- Transform and quantization
    - 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
- Entropy Coding
    - Arithmetic coding engine with adaptive double windows probability update
    - Transform coefficient coding with sign data hiding
- 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
    - Adaptive color transform
    - Block differential pulse coded modulation
    - Transform skip residual coding
- 360-degree video coding
    - Horizontal wrap-around motion compensation
- High-level syntax and parallel processing
    - Reference picture management with direct reference picture list signaling
    - Subpictures, slices and tiles

3.2 Partitioning

3.2.1 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 [11][12]. 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).

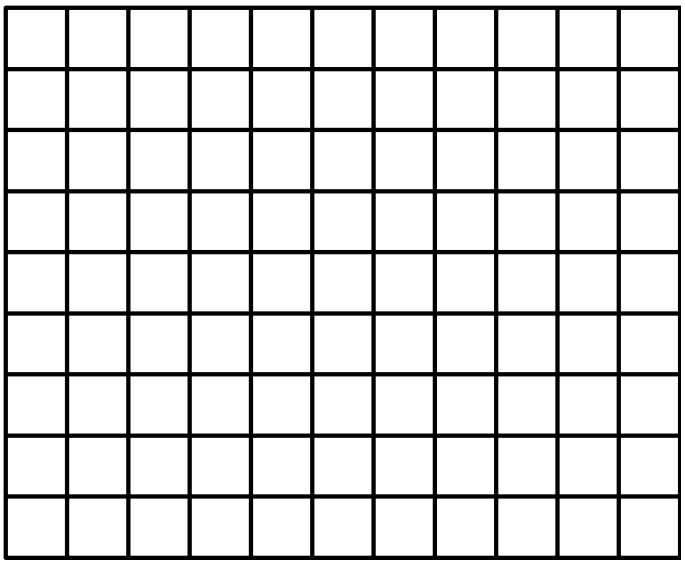


Figure 2 – Example of a picture divided into CTUs

3.2.2 Partitioning of pictures into subpictures, slices, tiles

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 slice consists of an integer number of complete tiles or an integer number of consecutive complete CTU rows within a tile of a picture.

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 complete tiles in a tile raster scan of a picture. In the rectangular slice mode, a slice contains either a number of complete tiles that collectively form a rectangular region of the picture or a number of consecutive complete CTU rows of one tile that collectively form a rectangular region of the picture. Tiles within a rectangular slice are scanned in tile raster scan order within the rectangular region corresponding to that slice.

A subpicture contains one or more slices that collectively cover a rectangular region of a picture.

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.

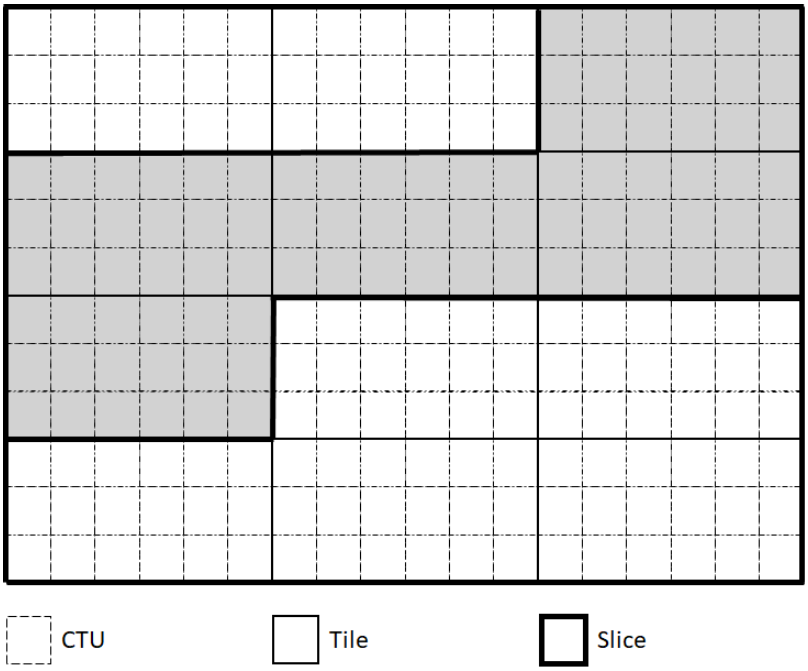


Figure 3 – Example of a picture partitioned into tiles and reaster-scan slices

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.

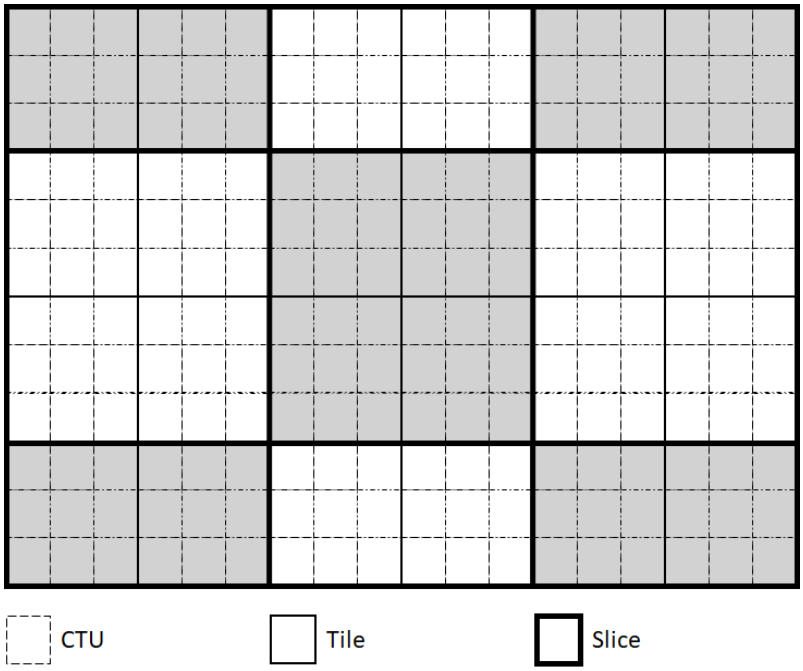


Figure 4 – Example of a picture partitioned into tiles and rectangular slices

Figure 5 shows an example of a picture partitioned into tiles and rectangular slices, where the picture is divided into 4 tiles (2 tile columns and 2 tile rows) and 4 rectangular slices.

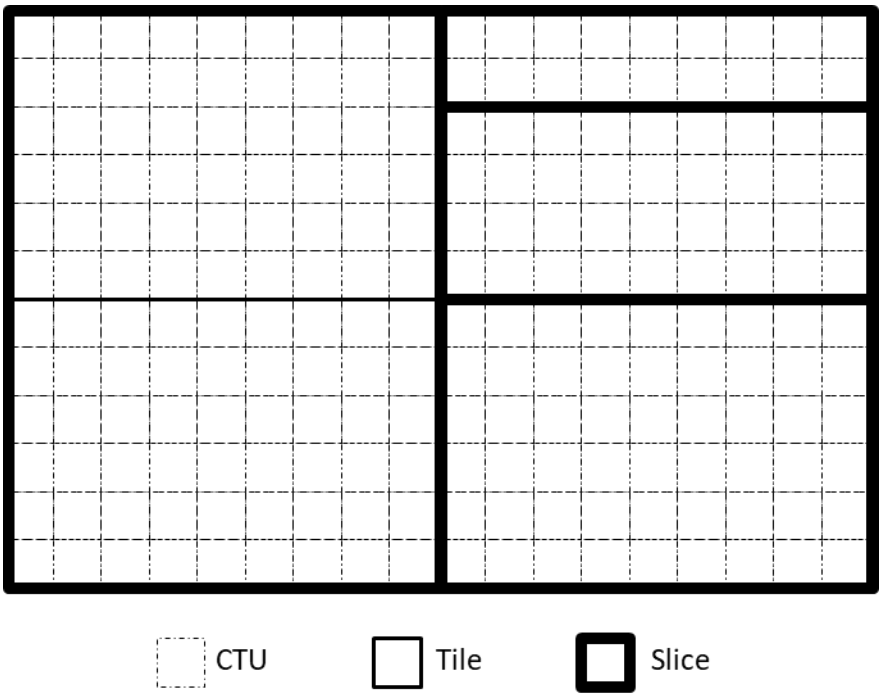


Figure 5 – Example of a picture partitioned into 4 tiles and 4 rectangular slices

Figure 6 shows an example of subpicture partitioning of a picture, where a picture is partitioned into 18 tiles, 12 on the left-hand side each covering one slice of 4 by 4 CTUs and 6 tiles on the right-hand side each covering 2 vertically-stacked slices of 2 by 2 CTUs, altogether resulting in 24 slices and 24 subpictures of varying dimensions (each slice is a subpicture).

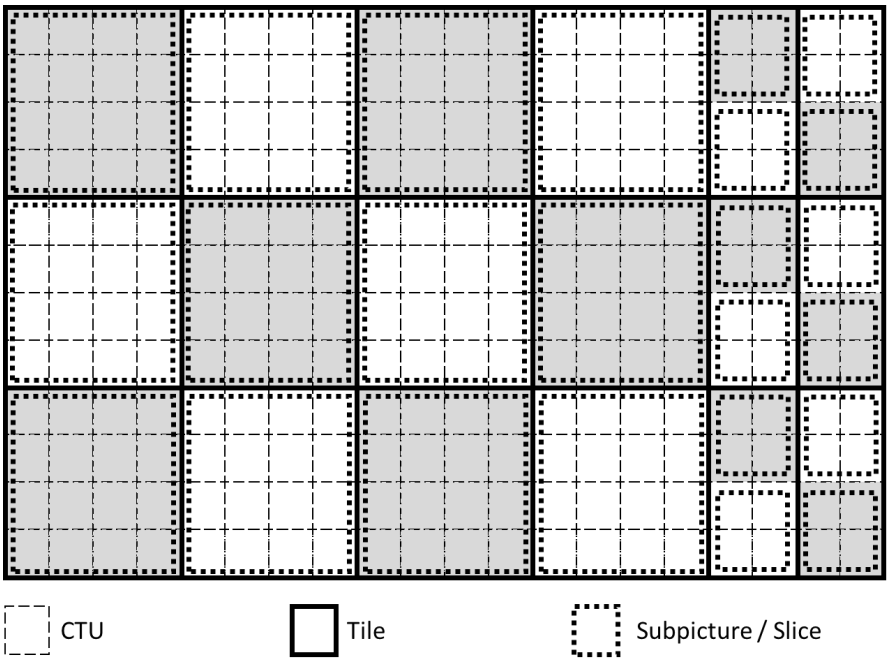


Figure 6 – Example of a picture partitioned into 28 subpictures

3.2.3 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 7, 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.

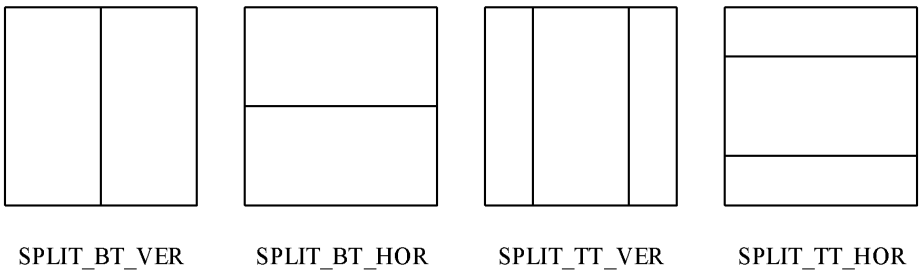


Figure 7 – Multi-type tree splitting modes

Figure 8 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 quadtree with nested multi-type tree coding tree structure, for each CU node, a first flag (split_cu_flag) is signalled to indicate whether the node is further partitioned. If the current CU node is a quadtree CU node, a second flag (split_qt_flag) whether it's a QT partitioning or MTT partitioning mode. When a node is partitioned with MTT partitioning mode, a third flag (mtt_split_cu_vertical_flag) is signalled to indicate the splitting direction, and then a fourth 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.

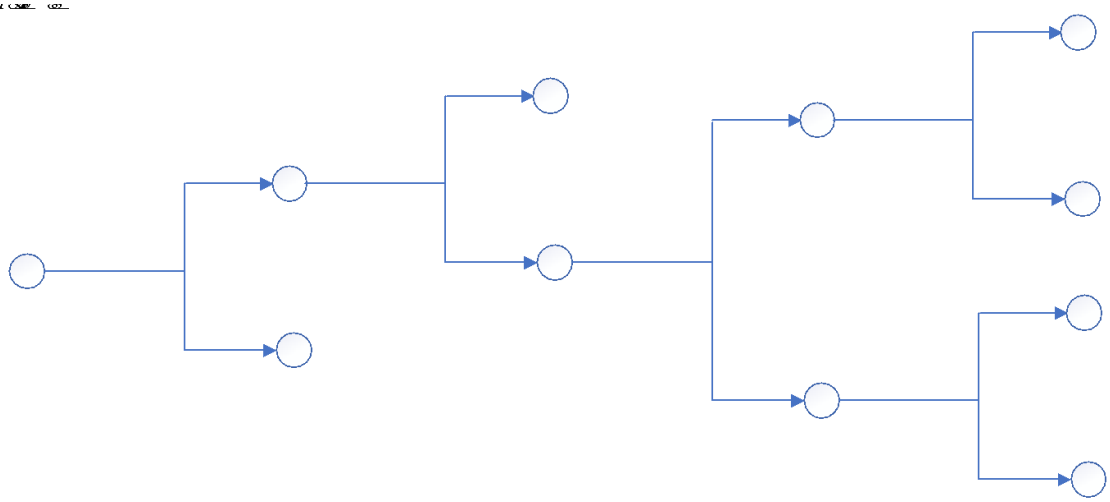


Figure 8 – Splitting flags signalling in quadtree with nested multi-type tree coding tree structure

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 9 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 size chroma CB consist of 16 chroma samples.

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.

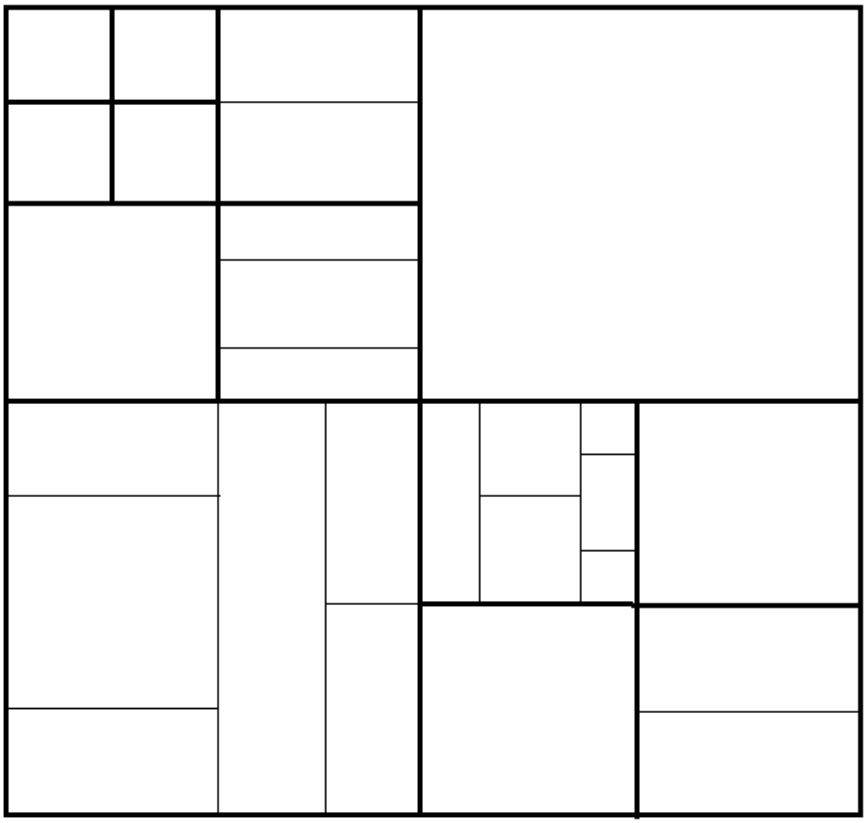


Figure 9– Example of quadtree with nested multi-type tree coding block structure

The following parameters are defined for the quadtree with nested multi-type tree coding tree scheme. These parameters are specified by SPS syntax elements and can be further refined by picture header syntax elements.

– 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

– *MinCbSize*: the minimum allowed coding block 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 128×128 and *MaxTtSize* is set as 64×64, the *MinCbsize* (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 *MinCbsize*, no further horizontal splitting is considered. Similarly, when the multi-type tree node has height equal to *MinCbsize*, no further vertical splitting is considered.

In VVC, the coding tree scheme supports the ability for the luma and chroma to have a separate block tree structure. 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.

3.2.4 CU splits on picture boundaries

Similar to 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 VVC:

– If any portion of a tree node block exceeds the bottom or the right picture boundaries, TT splitting is not allowed.

– If any portion of a tree node block exceeds the bottom or the right picture boundaries, and any of QT and BT splitting is not allowed due to block size restriction, the block is forced to be split with QT split mode.

– Otherwise 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 and a flag *split_qt_flag* is signalled to specify the parititioning mode for this case.
  – 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 and a flag *split_qt_flag* is signalled to specify the parititioning mode for this case.
  – 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.

3.2.5 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.

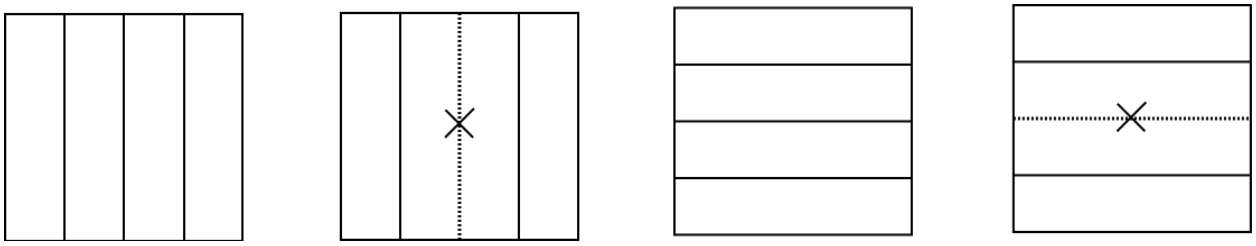


Figure 10–Redundant splitting patterns of binary tree split and ternary tree split cases

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.

3.2.6 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 VTM, 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.

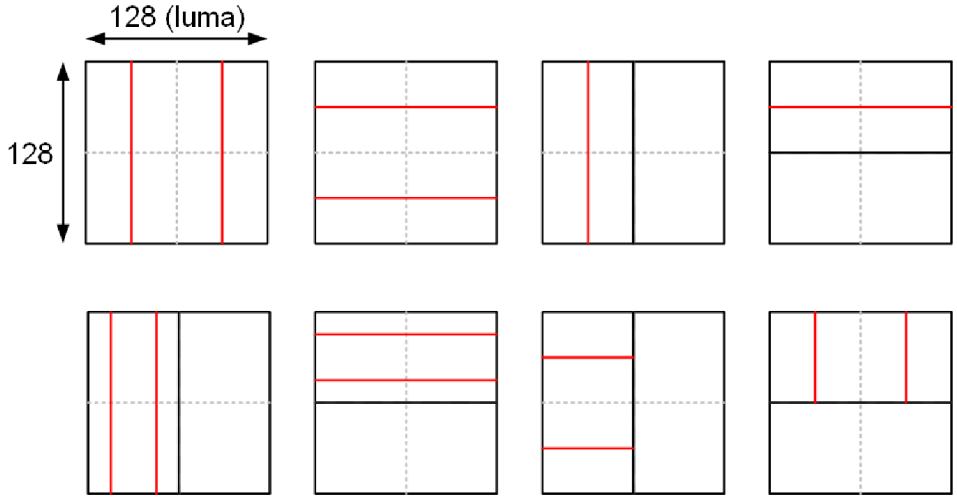


Figure 11 – Examples of disallowed TT and BT partitioning in VTM

3.2.7 Intra chroma partitioning and prediction 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 (size 2x2, 4x2, and 2x4) and chroma intra CBs with width smaller than 4 chroma samples (size 2xN) are 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, or a coding tree node whose chroma block size is not 2xN and has at least one child luma block 4xN 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 small chroma intra CBs with size less than 16 chroma samples or with size 2xN 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.

For the dual tree in intra picture, the 2xN intra chroma blocks are removed by disabling vertical binary and vertical ternary splits for 4xN and 8xN chroma partitions, respectively. The small chroam blocks wtihsize 2x2, 4x2, and 2x4 are also removed by partitioning restrictions.

In addition, a restriction on picture size is considered to avoid 2x2/2x4/4x2/2xN intra chroma blocks at the corner of pictures by considering the picture width and height to be multiple of max (8, MinCbSizeY).

3.3 Intra prediction

3.3.1 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 VVC 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 VVC, 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 VVC, 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.

3.3.1.1 Intra mode coding

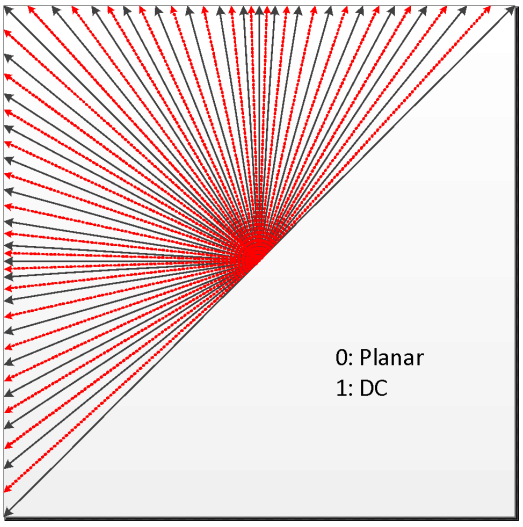


Figure 12 – 67 intra prediction modes

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, Max − 1, Max + 1, Max — 2, 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
  - Set a mode Min as the smaller mode in Left and Above
  - If Max – Min is equal to 1 :
    - MPM list □ {Planar, Left, Above, Min – 1, Max + 1, Min – 2}
  - Otherwise, if Max – Min is greater than or equal to 62 :
    - MPM list □ {Planar, Left, Above, Min + 1, Max – 1, Min + 2}
  - Otherwise, if Max – Min is equal to 2 :
    - MPM list □ {Planar, Left, Above, Min + 1, Min – 1, Max + 1}
  - Otherwise :
    - MPM list □ {Planar, Left, Above, Min – 1, –Min + 1, Max – 1}
- If Left and Above are both angular and they are the same:
  - MPM list □ {Planar, Left, Left − 1, Left + 1, Left – 2, 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.

3.3.1.2 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 VVC, 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.

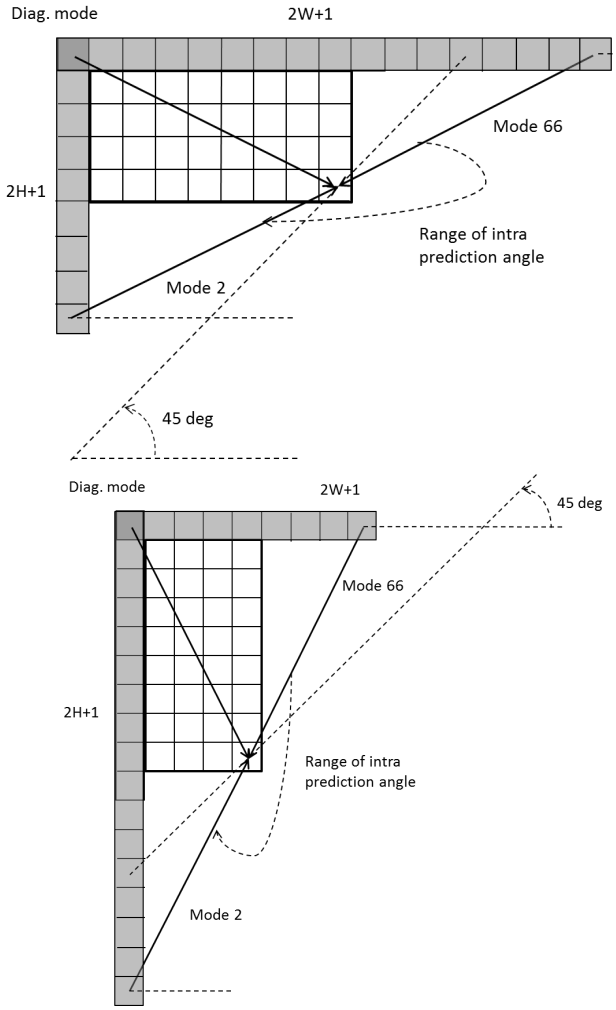


Figure 13 – Reference samples for wide-angular intra prediction

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 2,3,4,5,6,7,8,9,10,11,12, 13,14,15
W / H == 8	Modes 2,3,4,5,6,7,8,9,10,11,12, 13

W / H == 4	Modes 2,3,4,5,6,7,8,9,10,11
W / H == 2	Modes 2,3,4,5,6,7,8,9
W / H == 1	None
W / H == 1/2	Modes 59,60,61,62,63,64,65,66
W / H == 1/4	Mode 57,58,59,60,61,62,63,64,65,66
W / H == 1/8	Modes 55, 56,57,58,59,60,61,62,63,64,65,66
W / H == 1/16	Modes 53, 54, 55, 56,57,58,59,60,61,62,63,64,65,66

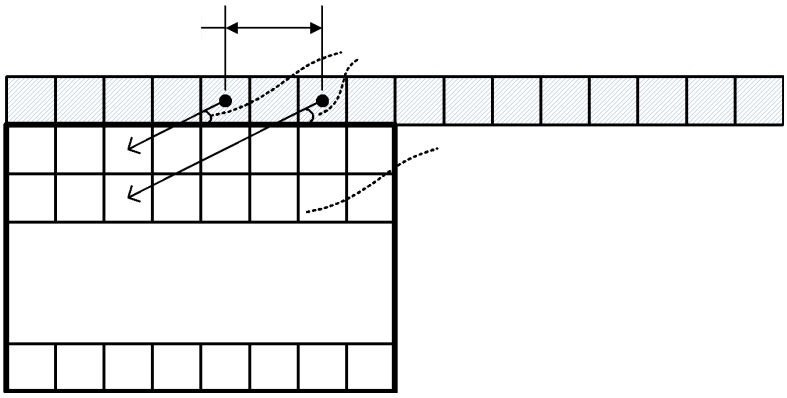


Figure 14 – Problem of discontinuity in case of directions beyond 45°

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 Δp_α. 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.

3.3.1.3 4-tap interpolation filter and reference sample smoothing

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 VVC, the two sets of 4-tap IFs replace lower precision linear interpolation as in HEVC, where one is a DCT-based interpolation filter (DCTIF) and the other one is a 4-tap smoothing interpolation filter (SIF). The DCTIF is constructed in the same way as the one used for chroma component motion compensation in both HEVC and VVC. The SIF is obtained by convolving the 2-tap linear interpolation filter with [1 2 1] /4 filter.

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

- The directional intra-prediction mode is classified into one of the following groups:
  - Group A: vertical or horizontal modes (HOR_IDX, VER_IDX),
  - Group B: directional modes that represent non-fractional angles (−14, −12, −10, −6, 2, 34, 66, 72, 76, 78, 80,) and Planar mode,
  - Group C: remaining directional modes;
- 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;
- Otherwise, if a mode falls into group B and the mode is a directional mode, and all of following conditions are true, 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::
  - refIdx is equal to 0 (no MRL)
  - TU size is greater than 32
  - Luma
  - No ISP block
- Otherwise, if a mode is classified as belonging to group C, MRL index is equal to 0, and the current block is not ISP block, 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). The interpolation filter type is determined as follows :
  - Set minDistVerHor equal to Min(Abs(predModeIntra − 50), Abs(predModeIntra − 18))
  - Set nTbS equal to (Log2 (W) + Log2 (H)) >> 1
  - Set intraHorVerDistThres[nTbS] as specified below :

	nTbS = 2	nTbS = 3	nTbS = 4	nTbS = 5	nTbS = 6	nTbS = 7
intraHorVerDistThres[nTbS]	24	14	2	0	0	0

  - If minDistVerHor is greater than intraHorVerDistThres[nTbS], SIF is used for the interpolation
  - Otherwise, DCTIF is used for the interpolation

3.3.2 Cross-component linear model prediction

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

$$pred_C(i, j) = \alpha \cdot rec_L'(i, j) + \beta \quad (3\text{-}1)$$

where $pred_C(i, j) pred_C(i, j)$ represents the predicted chroma samples in a CU and $rec_L(i, j) rec_L(i, j)$ represents the downsampled reconstructed luma samples of the same CU.

The CCLM parameters (αα and ββ) 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[3 * W' / 4, −1], S[−1, H' / 4], S[−1, 3 * H' / 4] when LM mode is applied and both above and left neighbouring samples are available;

– S[W’ / 8, −1], S[3 * W’ / 8, −1], S[5 * W’ / 8, −1], S[7 * W’ / 8, −1] when LM-A mode is applied or only the above neighbouring samples are available;
– S[−1, H’ / 8], S[−1, 3 * H’ / 8], S[−1, 5 * H’ / 8], S[−1, 7 * H’ / 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 larger values: x^0_A and x^1_A, and two smaller 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 x_A, x_B, y_A and y_B are derived as:

$$X_a=(x^0_A + x^1_A +1)>>1;\ X_b=(x^0_B + x^1_B +1)>>1;\ Y_a=(y^0_A + y^1_A +1)>>1;\ Y_b=(y^0_B + y^1_B +1)>>1 \quad (3\text{-}2)$$

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

$$\alpha = \frac{Y_a - Y_b}{X_a - X_b} \quad (3\text{-}3)$$

$$\beta = Y_b - \alpha \cdot X_b \quad (3\text{-}4)$$

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.

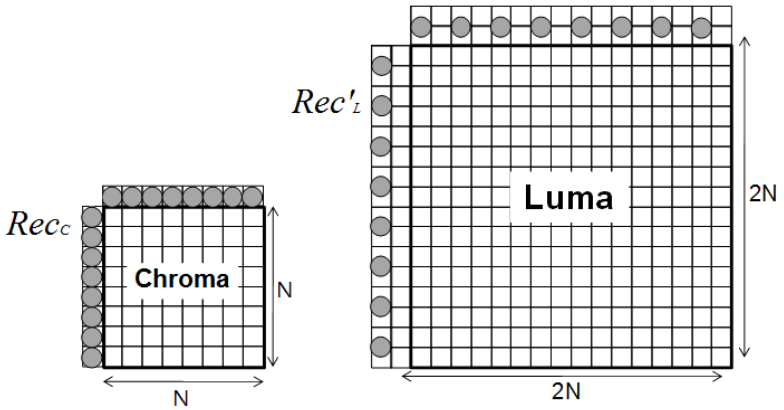


Figure 15 – Locations of the samples used for the derivation of α and β

The division operation to calculate parameter αα 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 αα 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:

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

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_T 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) samples. 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) samples.

In LM_LT mode, left and above templates are used to calculate the linear model coefficients.

To match the chroma sample locations for 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.

$$+ 2\cdot rec_L(2i-1,2j-1) + rec_L(2i+1,2j-1) + rec_L(2i-1,2j) + 2\cdot rec_L(2i,2j) + rec_L(2i+1,2j) + 4\Big] \gg 3 \quad (3\text{-}6)$$

$$rec_L'(i,j) = \Big[rec_L(2i,2j-1) + rec_L(2i-1,2j) + 4\cdot rec_L(2i,2j) + rec_L(2i+1,2j) + rec_L(2i,2j+1) + 4\Big] \gg 3 \quad (3\text{-}7)$$

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 3-4 are 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.

3.3.3 Position dependent intra prediction combination

In VVC, the results of intra prediction of DC, planar and several angular modes are further modified by a position dependent intra prediction combination (PDPC) method. PDPC is an intra prediction method which invokes a combination of the 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, intra angles less than or equal to horizontal, and intra angles greater than or equal to vertical and less than or equal to 80. If the current block is Bdpcm mode or MRL index is larger than 0, PDPC is not applied.

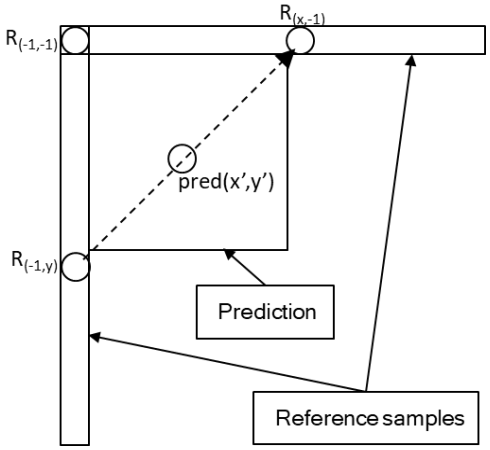
The prediction sample $pred(x',y')$ is predicted using an intra prediction mode (DC, planar, angular) and a linear combination of reference samples according to the Equation 3-8 as follows:

$$pred(x',y') = \text{Clip}(0, (1 << \text{BitDepth}) - 1, (wL \times R_{-1,y'} + wT \times R_{x',-1} + (64 - wL - wT) \times pred(x',y') + 32) >> 6) \quad (3\text{-}8)$$

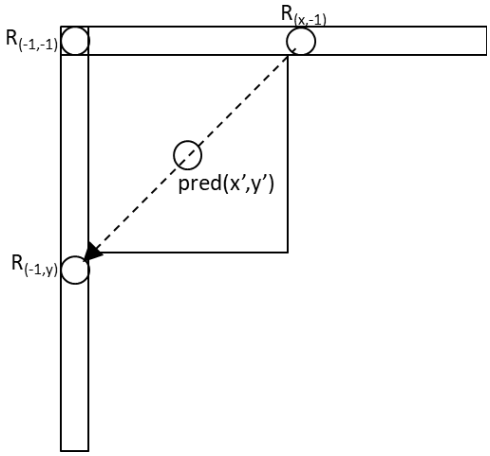
where $R_{x,-1}$, $R_{-1,y}$ represent the reference samples located at the top and left boundaries of current sample (x, y), respectively.

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. For angular modes, if the current angular mode is HOR_IDX or VER_IDX, left or top reference samples is not used, respectively. The PDPC weights and scale factors are dependent on prediction modes and the block sizes. PDPC is applied to the block with both width and height greater than or equal to 4.

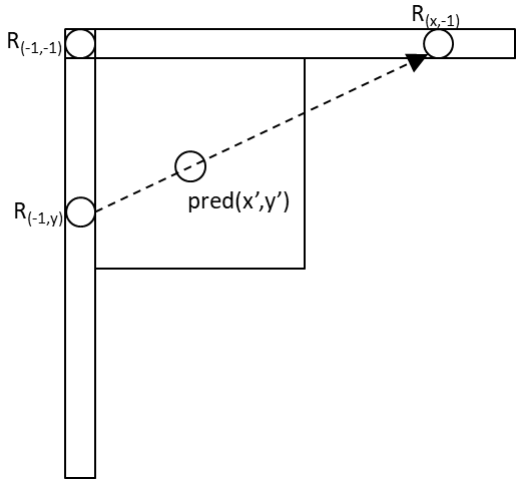
Figure 16 illustrates the definition of reference samples ($R_{x,-1}$ and $R_{-1,y}$) for PDPC applied over various prediction modes. The prediction sample $pred(x', y')$ is located at (x', y') within the prediction block. As an example, the coordinate x of the reference sample $R_{x,-1}$ is given by: $x = x' + y' + 1$, 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 angularg mode, the reference samples $R_{x,-1}$ 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.



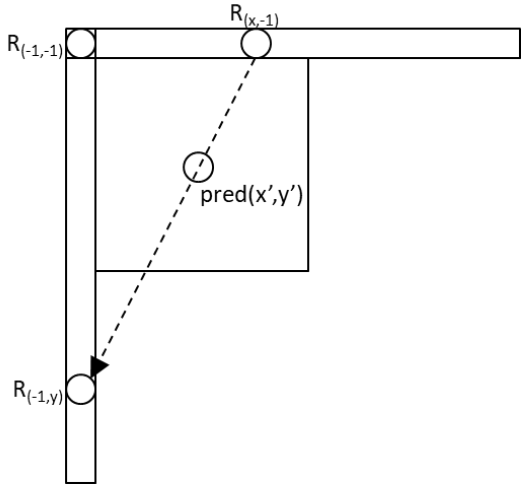
(a) Diagonal top-right mode



(b) Diagonal bottom-left mode



(c) Adjacent diagonal top-right mode



(d) Adjacent diagonal bottom-left mode

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

3.3.4 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 mode is excluded from intra prediction modes in case a nonzero reference line index is signalled.

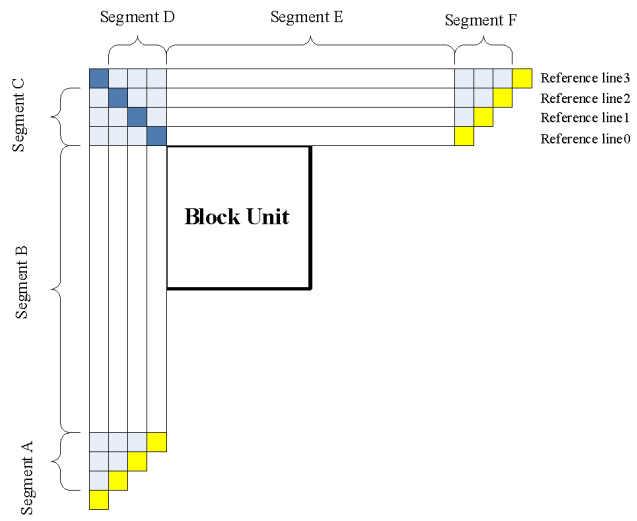


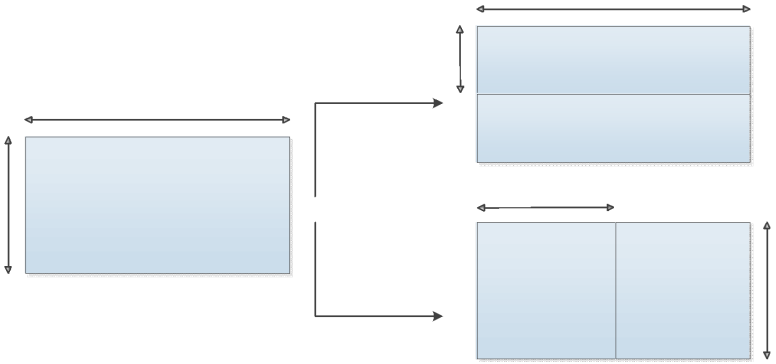
Figure 17 Example of four reference lines neighboring to a prediction block

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. MRL requires the storage of 3 neighboring luma reference lines with a CTU to generate predictions. The Cross-Component Linear Model (CCLM) tool also requires 3 neighboring luma reference lines for its downsampling filters. The definition ofMRL to use the same 3 lines is aligned as CCLM to reduce the storage requirements for decoders.

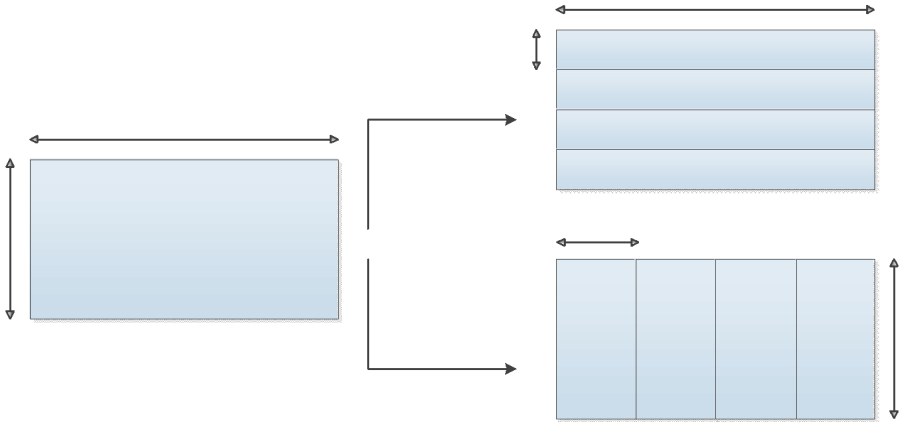
3.3.5 Intra sub-partitions (ISP)

The intra sub-partitions (ISP) 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 $M\times128$ (with $M \leq 64$) and $128\times N$ (with $N \leq 64$) ISP blocks could generate a potential issue with the 64×64 VDPU. For example, an $M\times128$ CU in the single tree case has an $M\times128$ luma TB and two corresponding $\frac{M}{2}\times64$ chroma TBs. If the CU uses ISP, then the luma TB will be divided into four $M\times32$ TBs (only the horizontal split is possible), each of them smaller than a 64×64 block. However, in the current design of ISP chroma blocks are not divided. Therefore, both chroma components will have a size greater than a 32×32 block. Analogously, a similar situation could be created with a $128\times N$ CU using ISP. Hence, these two cases are an issue for the 64×64 decoder pipeline. For this reason, the CU sizes that can use ISP is restricted to a maximum of 64×64. 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 3-5 – Entropy coding coefficient group size

Block Size	Coefficient group Size
$1\times N,\ N\geq 16$	1×16
$N\times 1,\ N\geq 16$	16×1
$2\times N,\ N\geq 8$	2×8
$N\times 2,\ N\geq 8$	8×2
All other possible $M\times N$ cases	4×4

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 3-5. 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 4×4 dimensions.
- CBF coding: it is assumed to have at least one of the sub-partitions has a non-zero CBF. Hence, if n is the number of sub-partitions and the first $n-1$ sub-partitions have produced a zero CBF, then the CBF of the n-th sub-partition is inferred to be 1.
- Transform size restriction: all ISP transforms with a length larger than 16 points uses the DCT-II.
- 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 t_H and t_V be the horizontal and the vertical transforms selected respectively for the $w \times h$ sub-partition, where w is the width and h is the height. Then the transform is selected according to the following rules:
  - If $w = 1$ or $h = 1$, then there is no horizontal or vertical transform respectively.
  - If $w \geq 4$ and $w \leq 16$, t_H = DST-VII, otherwise, t_H = DCT-II
  - If $h \geq 4$ and $h \leq 16$, t_V = DST-VII, otherwise, t_V = DCT-II

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 reference sample filtering process (reference smoothing) and 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.

3.3.6 Matrix weighted Intra Prediction (MIP)

Matrix weighted intra prediction (MIP) method is a newly added intra prediction technique into VVC. For predicting the samples of a rectangular block of width W and height H, matrix weighted intra prediction (MIP) takes one line of H reconstructed neighbouring boundary samples left of the block and one line of W 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 19.

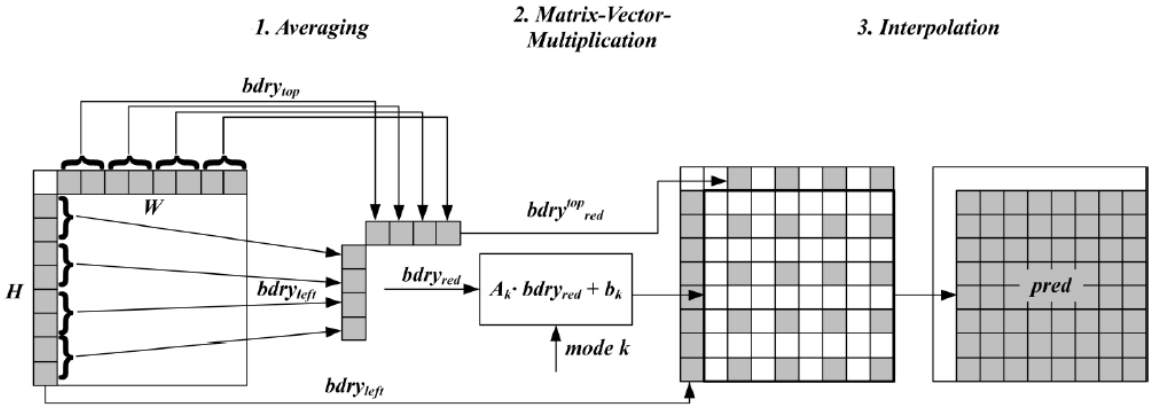


Figure 19 – Matrix weighted intra prediction process

3.3.6.1 Averaging neighboring samples

Among the boundary samples, four samples or eight samples are selected by averaging based on block size and shape. Specifically, the input boundaries $bdry^{top}$ and $bdry^{left}$ are reduced to smaller boundaries

$bdry_{red}^{top}$ and $bdry_{red}^{left}$ by averaging neighboring boundary samples according to predefined rule depends on block size. Then, the two reduced boundaries $bdry_{red}^{top}$ and $bdry_{red}^{left}$ are concatenated to a reduced boundary vector $bdry_{red}$ which is thus of size four for blocks of shape 4×4 and of size eight for blocks of all other shapes. If $mode$ refers to the MIP-mode, this concatenation is defined as follows:

$$d\ mode \geq 10 \left[bdry_{red}^{top},\ bdry_{red}^{left} \right] for (W, H)\ > 8\ and\ mode\ < 6 \left[bdry_{red}^{left},\ bdry_{red}^{top} \right] for (W, H)\ > 8\ and\ mode \geq 6. \tag{3-9}$$

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 $bdry_{red}$ a reduced prediction signal $pred_{red,}$ which is a signal on the downsampled block of width W_{red} and height H_{red} is generated. Here, W_{red} and H_{red} are defined as:

$$W_{red} = \{4\ for\ max(W,\ H) \leq 8\ min(W, 8)\ for\ max(W,\ H) > 8 \tag{3-10}$$

$$H_{red} = \{4\ for\ max(W,\ H) \leq 8\ min(H, 8)\ for\ max(W,\ H) > 8 \tag{3-11}$$

The reduced prediction signal $pred_{red}$ is computed by calculating a matrix vector product and adding an offset:

$$pred_{red} = A \bullet bdry_{red} + b.$$

Here, A is a matrix that has $W_{red} \cdot H_{red}$ rows and 4 columns if $W = H = 4$ and 8 columns in all other cases. b is a vector of size $W_{red} \cdot H_{red}$. The matrix A and the offset vector b are taken from one of the sets S_0, S_1, S_2. One defines an index $idx = idx(W, H)$ as follows:

$$idx(W, H) = \{0\ for\ W = H = 4\ 1\ for\ max(W, H) = 8\ 2\ for (W, H)\ > 8. \tag{3-12}$$

Here, each coefficient of the matrix A is represented with 8 bit precision. The set S_0 consists of 16 matrices A_0^i, $i \in \{0, ..., 15\}$ each of which has 16 rows and 4 columns and 16 offset vectors b_0^i, $i \in \{0, ..., 16\}$ each of size 16. Matrices and offset vectors of that set are used for blocks of size 4×4. The set S_1 consists of 8 matrices A_1^i, $i \in \{0, ..., 7\}$, each of which has 16 rows and 8 columns and 8 offset vectors b_1^i, $i \in \{0, ..., 7\}$ each of size 16. The set S_2 consists of 6 matrices A_2^i, $i \in \{0, ..., 5\}$, each of which has 64 rows and 8 columns and of 6 offset vectors b_2^i, $i \in \{0, ..., 5\}$ of size 64.

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 interpolation is performed firstly in the horizontal direction and then in the vertical direction regardless of block shape or block size.

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 whether an MIP mode is to be applied or not is sent. If an MIP mode is to be applied, MIP mode ($predModeIntra$) is signaled. For an MIP mode, a

transposed flag ($isTransposed$), which determines whether the mode is transposed, and MIP mode Id ($modeId$), which determines which matrix is to be used for the given MIP mode is derived as follows

$$isTransposed = predModeIntra\&1$$
$$modeId = predModeIntra >> 1 \quad (3\text{-}13)$$

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 of 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
- MIP is allowed up to 64x64 regardless of the maximum transform size
- The number of MIP modes is 32 for sizeId=0, 16 for sizeId=1 and 12 for sizeId=2

3.4 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, VVC includes a number of new and refined inter prediction coding tools listed as follows:

- Extended merge prediction
- High precision (1/16 pel) motion compensation and motion vector storage
- Merge mode with MVD (MMVD)
- Symmetric MVD (SMVD) signalling
- Affine motion compensated prediction
- Subblock-based temporal motion vector prediction (SbTMVP)
- Adaptive motion vector resolution (AMVR)
- Bi-prediction with CU-level weight (BCW)
- Bi-directional optical flow (BDOF)
- Decoder side motion vector refinement (DMVR)
- Geometric partitioning mode (GPM)
- Combined inter and intra prediction (CIIP)
- Reference picture resampling

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

3.4.1 Extended merge prediction

In VVC, 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 sequence parameter set header and the maximum allowed size of merge list is 6. 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 derivation process of each category of merge candidates is provided in this session. As done in HEVC, VVC also supports parallel derivation of the merging candidate lists for all CUs within a certain size of area.

3.4.1.1 Spatial candidates derivation

The derivation of spatial merge candidates in VVC is same to that in HEVC except the positions of first two merge candidates are swapped. A maximum of four merge candidates are selected among candidates located in the positions depicted in Figure 20. The order of derivation is B_0, A_0, B_1, A_1 and B_2. Position B_2 is considered only when one or more than one CUs of position B_0, A_0, B_1, A_1 are not available (e.g. because it belongs to another slice or tile) or is intra coded. After candidate at position A_1 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.

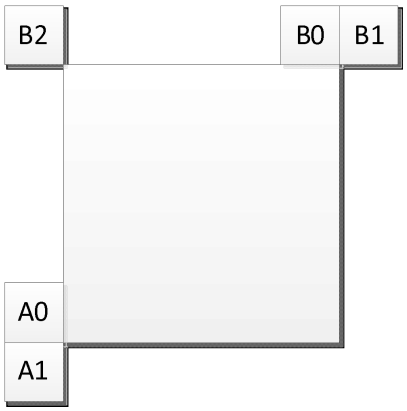


Figure 20– Positions of spatial merge candidate

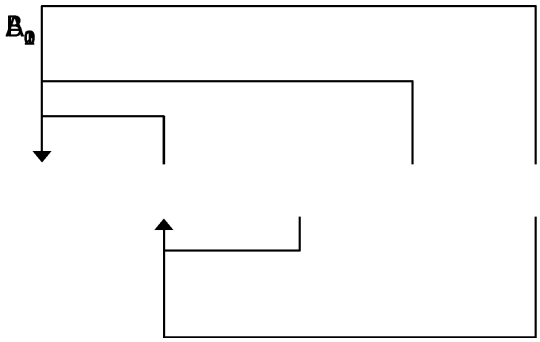


Figure 21 – Candidate pairs considered for redundancy check of spatial merge candidates

3.4.1.2 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 reference picture. The reference picture list and the reference index 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.

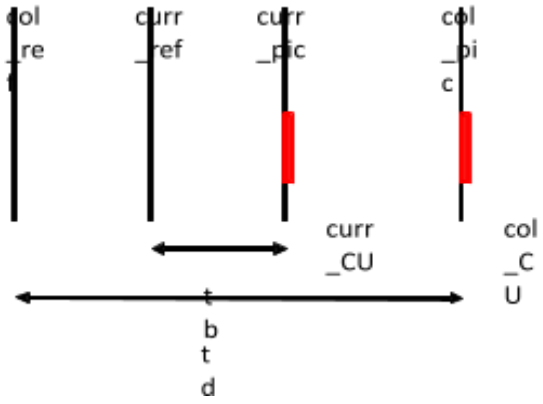


Figure 22 – Illustration of motion vector scaling for temporal merge candidate

The position for the temporal candidate is selected between candidates C_0 and C_1, as depicted in Figure 23. If CU at position C_0 is not available, is intra coded, or is outside of the current row of CTUs, position C_1 is used. Otherwise, position C_0 is used in the derivation of the temporal merge candidate.

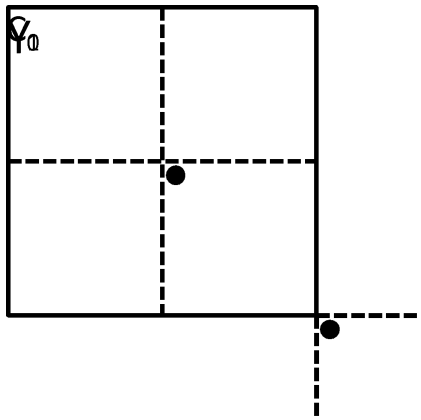


Figure 23 – Candidate positions for temporal merge candidate, C_0 and C_1

3.4.1.3 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.

The HMVP table size *S* is set to be 6, which indicates up to 5 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, and the identical HMVP is inserted to the last entry of the table.

HMVP candidates could be used in the merge candidate list construction process. The latest several HMVP 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. The last two entries in the table are redundancy checked to A_1 and B_1 spatial candidates, respectively.
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.

3.4.1.4 Pair-wise average merge candidate derivation

Pairwise average candidates are generated by averaging predefined pairs of candidates in the existing merge candidate list, using the first two merge candidates. The first merge candidate is defined as p0Cand and the second merge candidate can be defined as p1Cand, respectively. The averaged motion vectors are calculated according to the availability of the motion vector of p0Cand and p1Cand 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, and its reference picture is set to the one of p0Cand; if only one motion vector is available, use the one directly; if no motion vector is available, keep this list invalid. Also, if the half-pel interpolation filter indices of p0Cand and p1Cand are different, it is set to 0.

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.

3.4.1.5 Merge estimation region

Merge estimation region (MER) allows independent derivation of merge candidate list for the CUs in the same merge estimation region (MER). A candidate block that is within the same MER to the current CU is not included for the generation of the merge candidate list of the current CU. In addition, the updating process for the history-based motion vector predictor candidate list is updated only if (xCb + cbWidth) >> Log2ParMrgLevel is greater than xCb >> Log2ParMrgLevel and (yCb + cbHeight) >> Log2ParMrgLevel is great than (yCb >> Log2ParMrgLevel) and where (xCb, yCb) is the top-left luma sample position of the current CU in the picture and (cbWidth, cbHeight) is the CU size. The MER size is selected at encoder side and signalled as log2_parallel_merge_level_minus2 in the sequence parameter set.

3.4.2 High precision (1/16 pel) motion compensation and motion vector storage

VVC increases the MV precision to 1/16 luma sample, to improve the prediction efficiency of slow motion video. This higher motion accuracy is particularly helpful for video contents with locally varying and non-translational motion such as in case of affine mode. For fractional position samples generation of higher MV accuracy, HEVC's 8-tap luma interpolation filters and 4-tap chroma interpolation filters are extended to 16 phases for luma and 32 phases for chroma. This extended filter set is applied in MC process of inter coded CUs except the CUs in affine mode. For affine mode, a set of 6-tap luma interpolation filter with 16 phases is used for lower computational complexity as well as memory bandwidth saving.

In VVC, the highest precision of explicitly signalled motion vectors for non-affine CU is quarter-luma-sample. In some inter prediction modes such as the affine mode, motion vectors can be signalled at 1/16-luma-sample precision. In all inter coded CU with implicitly inferred MVs, the MVs are derived at 1/16-luma-sample precision and motion compensated prediction is performed at 1/16-sample-precision. In terms of internal motion field storage, all motion vectors are stored at 1/16-luma-sample precision.

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

3.4.3 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 regular 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 mmvd candidate flag is signalled to specify which one is used between the first and second merge candidates.

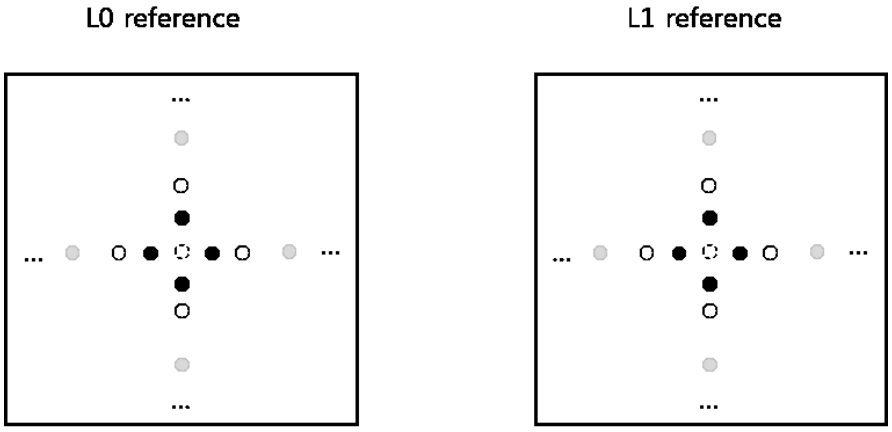


Figure 24 – MMVD Search Point

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 3-6.

Table 3-6 – 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 3-7. 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 3-7 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), and the difference of POC in list 0 is greater than the one in list 1, the sign in Table 3-7 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. Otherwise, if the difference of POC in list 1 is greater than list 0, the sign in Table 3-7 specifies the sign of MV offset added to the list1 MV component of starting MV and the sign for the list0 MV has opposite value.

The MVD is scaled according to the difference of POCs in each direction. If the differences of POCs in both lists are the same, no scaling is needed. Otherwise, if the difference of POC in list 0 is larger than the one of list 1, the MVD for list 1 is scaled, by defining the POC difference of L0 as td and POC difference of L1 as tb, described in Figure 23. If the POC difference of L1 is greater than L0, the MVD for list 0 is scaled in the same way. If the starting MV is uni-predicted, the MVD is added to the available MV.

Table 3-7 – 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	+	–

3.4.4 Symmetric MVD coding

In VVC, 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 nearest reference picture in list-0 and the nearest 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.
2. 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.

$$\{(mvx_0, mvy_0) = (mvpx_0 + mvdx_0, mvpy_0 + mvdy_0) \ (mvx_1, mvy_1) = (mvpx_1 - mvdx_0, mvpy_1 - mvdy_0) \tag{3-14}$$

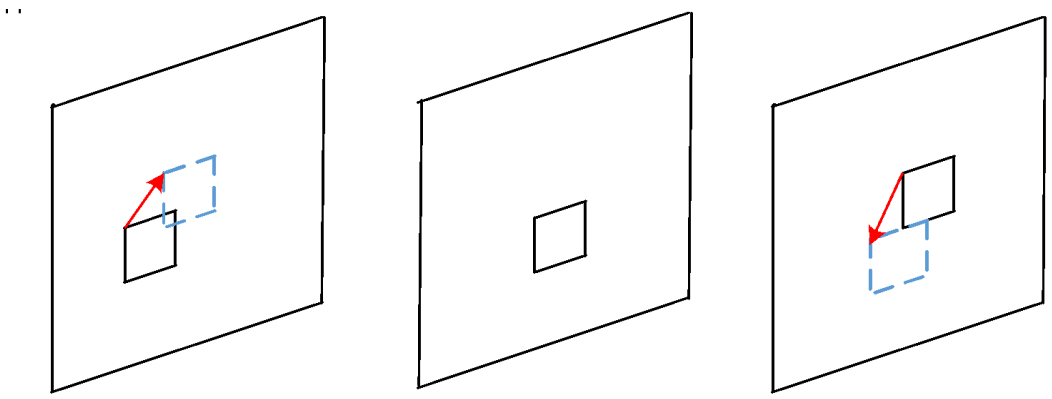
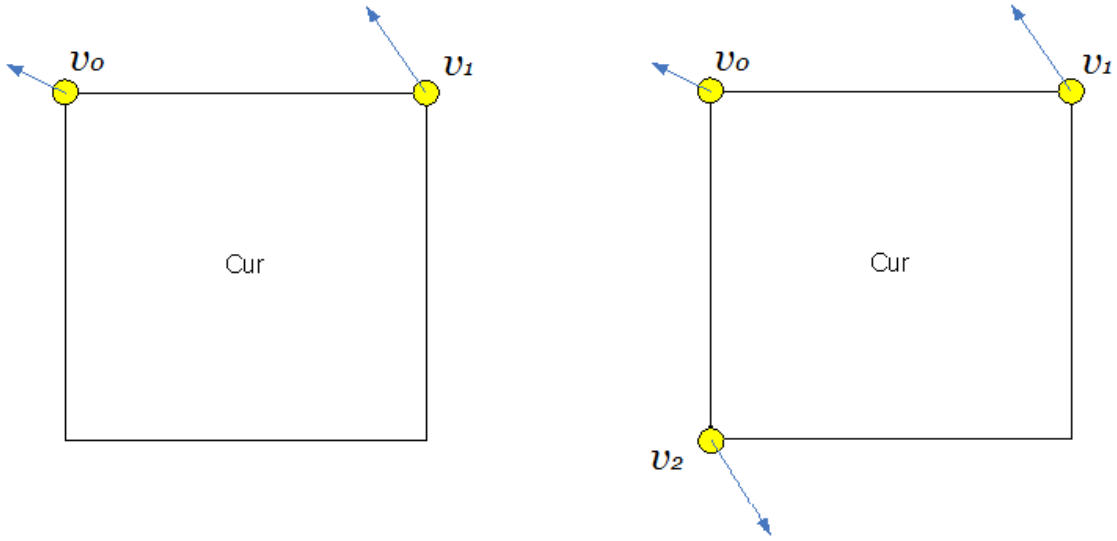


Figure 25 – Illustration for symmetrical MVD mode

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.

3.4.5 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 VVC, 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).



(a) 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:

$$\{mv_x = \frac{mv_{1x}-mv_{0x}}{W}x + \frac{mv_{0y}-mv_{1y}}{W}y + mv_{0x}\ mv_y = \frac{mv_{1y}-mv_{0y}}{W}x + \frac{mv_{1x}-mv_{0x}}{W}y + mv_{0y}$$

(3-15)

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

$$\{mv_x = \frac{mv_{1x}-mv_{0x}}{W}x + \frac{mv_{2x}-mv_{0x}}{H}y + mv_{0x}\ mv_y = \frac{mv_{1y}-mv_{0y}}{W}x + \frac{mv_{2y}-mv_{0y}}{H}y + mv_{0y}$$

(3-16)

Where (mv_{0x}, mv_{0y}) is motion vector of the top-left corner control point, (mv_{1x}, mv_{1y}) is motion vector of the top-right corner control point, and (mv_{2x}, mv_{2y}) 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 top-left and bottom-right luma subblocks in the collocated 8x8 luma region.

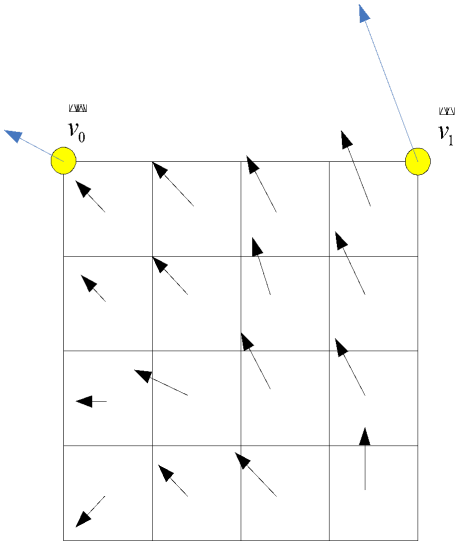


Figure 27 – Affine MVF per subblock

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

3.4.5.1 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:

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

In VVC, 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
Figure 29, if the neighbour left bottom block A is coded in affine mode, the motion vectors v_2 , v_3 and v_4 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 v_2, and v_3. In case that block A is coded with 6-parameter affine model, the three CPMVs of the current CU are calculated according to v_2 , v_3 and v_4.

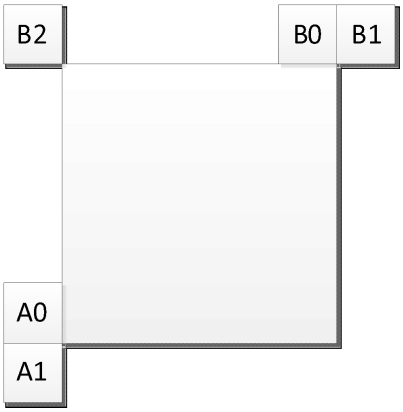


Figure 28 – Locations of inherited affine motion predictors

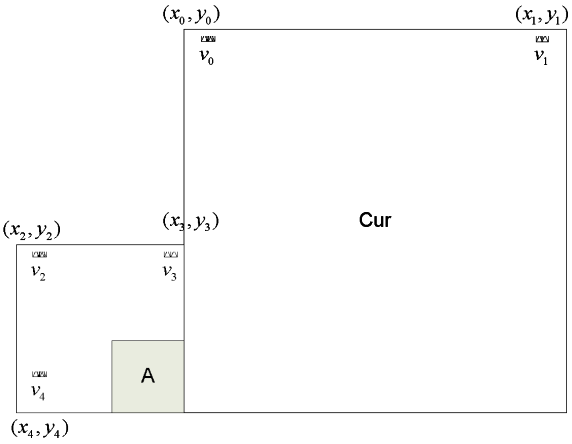


Figure 29 – Control point motion vector inheritance

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. $CPMV_k$ (k=1, 2, 3, 4) represents the k-th control point. For $CPMV_1$, the B2->B3->A2 blocks are checked and the MV of the first available block is used. For $CPMV_2$, the B1->B0 blocks are checked and for $CPMV_3$, the A1->A0 blocks are checked. For TMVP is used as $CPMV_4$ 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:

{$CPMV_1$, $CPMV_2$, $CPMV_3$}, {$CPMV_1$, $CPMV_2$, $CPMV_4$}, {$CPMV_1$, $CPMV_3$, $CPMV_4$},
{$CPMV_2$, $CPMV_3$, $CPMV_4$}, { $CPMV_1$, $CPMV_2$}, { $CPMV_1$, $CPMV_3$}

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.

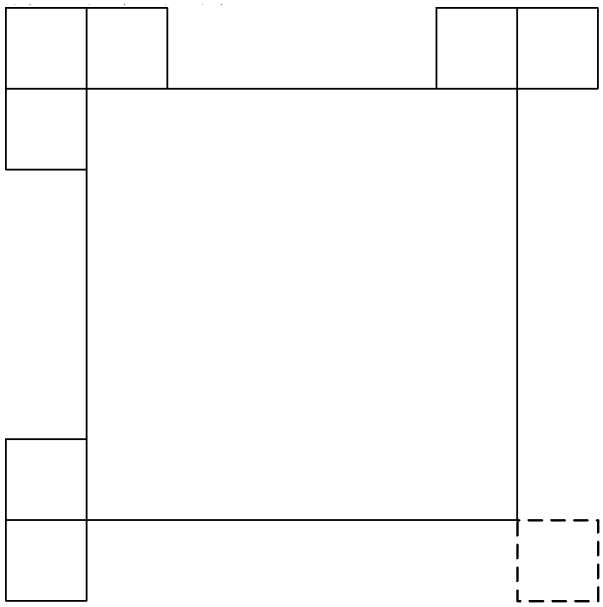


Figure 30 –Locations of Candidates position for constructed affine merge mode

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.

3.4.5.2 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:

- Inherited affine AMVP candidates that extrapolated from the CPMVs of the neighbour CUs
- Constructed affine AMVP candidates CPMVPs that are derived using the translational MVs of the neighbour CUs
- Translational MVs from neighboring CUs
- 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 mv_0 and $mv_1$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 valid inherited affine AMVP candidates and constructed AMVP candidate are inserted, mv_0, mv_1 and mv_2will 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.

3.4.5.3 Affine motion information storage

In VVC, 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 deblocking.

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.

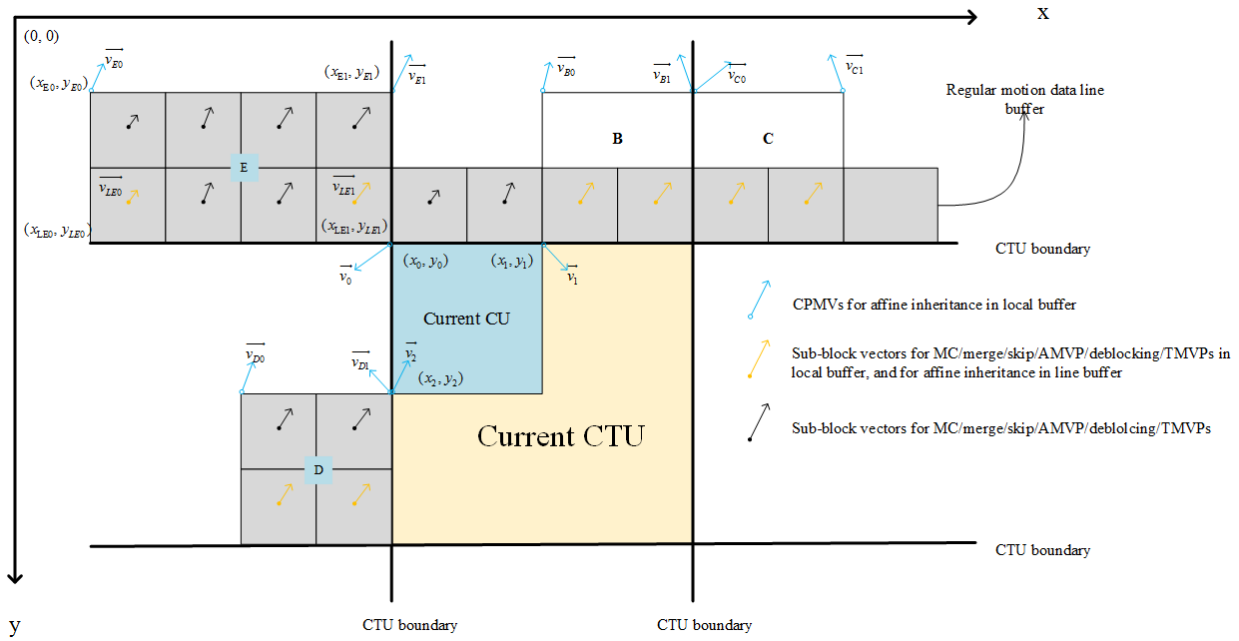


Figure 31 – Illustration of motion vector usage for proposed combined method

3.4.5.4 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 VVC, 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 $I(i,\ j)$.

Step2) The spatial gradients $g_x(i,j)$ and $g_y(i,j)$ 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.

$$g_x(i,\ j) = (I(i\ +\ 1,\ j) \gg shift1) - (I(i\ -\ 1,\ j) \gg shift1) \tag{3-17}$$

$$g_y(i,\ j) = (I(i,\ j\ +\ 1) \gg shift1)\ -\ (I(i,\ j\ -\ 1) \gg shift1) \tag{3-18}$$

$shift1$ 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.

$$\Delta I(i,j)\ =\ g_x(i,j) * \Delta v_x(i,j)\ +\ g_y(i,j) * \Delta v_y(i,j) \tag{3-19}$$

where the $\Delta v(i,j)$ is the difference between sample MV computed for sample location (i,j), denoted by $v(i,j)$, and the subblock MV of the subblock to which sample (i,j) belongs, as shown in Figure 32. The $\Delta v(i,j)$ is quantized in the unit of 1/32 luam sample precision.

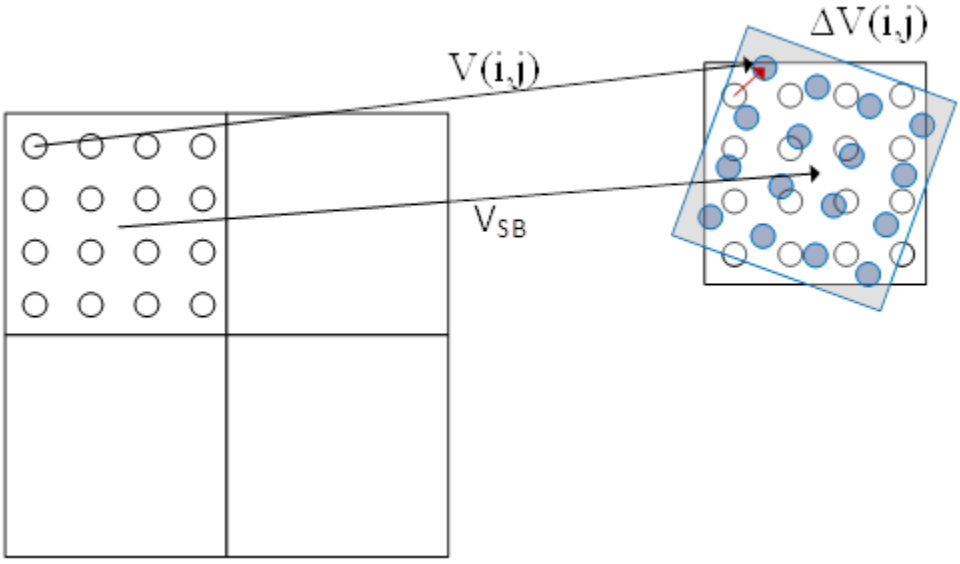


Figure 32 – Subblock MV V_{SB} and pixel $\Delta v(i,j)$ (red arrow)

Since the affine model parameters and the sample location relative to the subblock center are not changed from subblock to subblock, $\Delta v(i,j)$ can be calculated for the first subblock, and reused for other subblocks in the same CU. Let $dx(i,j)$ and $dy(i,j)$ be the horizontal and vertical offset from the sample location (i,j) to the center of the subblock $\left(x_{SB}, y_{SB}\right)$, $\Delta v(x,y)$ can be derived by the following equation,

$$\{dx(i,j) = i\ -\ x_{SB}\ dy(i,\ j) = j\ -\ y_{SB} \tag{3-20}$$

$$\{\Delta v_x(i,j) = C\ *\ dx(i,j)\ +\ D\ *\ dy(i,j)\ \Delta v_y(i,\ j)\ =\ E\ *\ dx(i,j)\ +\ F\ *\ dy(i,j) \tag{3-21}$$

In order to keep accuracy, the enter of the subblock $\left(x_{SB}, y_{SB}\right)$ is calculated as ((W_{SB} − 1)/2, (H_{SB} − 1) / 2), where W_{SB} and H_{SB} are the subblock width and height, respectively.

For 4-parameter affine model,

$$\{C\ =\ F\ =\ \frac{v_{1x}-v_{0x}}{w}\ E\ =-\ D\ =\ \frac{v_{1y}-v_{0y}}{w} \tag{3-22}$$

For 6-parameter affine model,

$$\{C\ =\ \frac{v_{1x}-v_{0x}}{w}\ D\ =\ \frac{v_{2x}-v_{0x}}{h}\ E\ =\ \frac{v_{1y}-v_{0y}}{w}\ F\ =\ \frac{v_{2y}-v_{0y}}{h} \tag{3-23}$$

where (v_{0x}, v_{0y}), (v_{1x}, v_{1y}), (v_{2x}, v_{2y}) are the top-left, top-right and bottom-left control point motion vectors, w and h are the width and height of the CU.

Step 4) Finally, the luma prediction refinement $\Delta I(i,j)$ is added to the subblock prediction $I(i, j)$. The final prediction I' is generated as the following equation.

$$I'(i,j) = I(i,j) + \Delta I(i,j)$$

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.

3.4.5.5 Adaptive bypass of affine ME

If enabled using the VTM encoder parameter AdaptBypassAffineMe, adaptive bypass of affine ME is used as an encoder only operation used to speed up encoding.

Before performing affine ME for a CU, the coding modes of its five spatial neighbours (above, left, above-right, bottom-left, above-left) are checked. If the number of available neighbours is greater than or equal to 4 and none of them are coded as affine or SbTMVP mode, affine ME is bypassed. In addition following two conditions are considered.

1) If a CU is no larger than 16x16, affine ME is not bypassed.

2) If the best mode for a CU is affine merge so far, and the current picture does not have symmetric reference pair (SMVD condition) or the absolute temporal distance between the current picture and SMVD reference is larger than 1, affine ME is not bypassed

3.4.6 Subblock-based temporal motion vector prediction (SbTMVP)

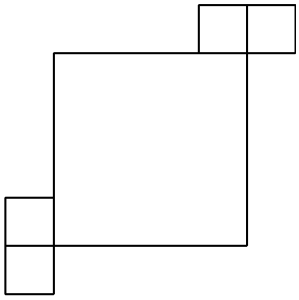
VVC supports the subblock-based temporal motion vector prediction (SbTMVP) method. Similar to the temporal motion vector prediction (TMVP) in HEVC, SbTMVP uses the motion field in the collocated picture to improve motion vector prediction and merge mode for CUs in the current picture. The same collocated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

- TMVP predicts motion at CU level but SbTMVP predicts motion at sub-CU level;
- 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.

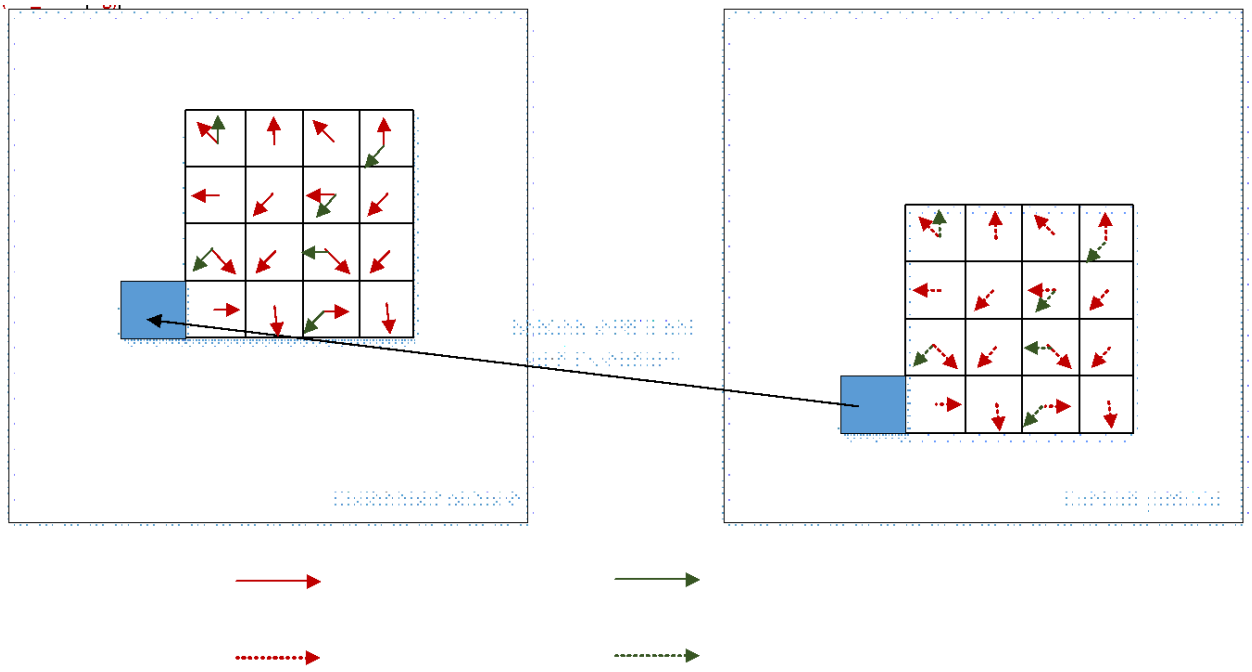
The SbTVMP process is illustrated in Figure 33. SbTMVP predicts the motion vectors of the sub-CUs within the current CU in two steps. In the first step, the spatial neighbor A1 in Figure 33 (a) is examined. If A1 has a motion vector that uses the collocated picture as its reference picture, this motion 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.



(a) Spatial neighboring blocks used by ATVMP



(b) 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 VVC, 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 VVC.

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.

3.4.7 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 VTM, 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.

3.4.8 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 VVC, the bi-prediction mode is extended beyond simple averaging to allow weighted averaging of the two prediction signals.

$$P_{bi-pred} = \left((8 - w) * P_0 + w * P_1 + 4\right) \gg 3 \quad (3\text{-}24)$$

Five weights are allowed in the weighted averaging bi-prediction, $w \in \{-2, 3, 4, 5, 10\}$. 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 ($w \in \{3,4,5\}$) 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 VVC, 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.

3.4.9 Bi-directional optical flow (BDOF)

The bi-directional optical flow (BDOF) tool is included in VVC. BDOF, previously referred to as BIO, was included in the JEM. Compared to the JEM version, the BDOF in VVC 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 distances (i.e. POC difference) from two reference pictures to the current picture are same
- Both reference pictures are short-term reference pictures.
- The CU is not coded using affine mode or the SbTMVP 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

BDOF is only applied to the luma component. 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 (v_x, v_y) 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, $\frac{\partial I^{(k)}}{\partial x}(i, j)$ and $\frac{\partial I^{(k)}}{\partial y}(i, j)$, $k = 0, 1$, of the two prediction signals are computed by directly calculating the difference between two neighboring samples, i.e.,

$$\begin{aligned} \frac{\partial I^{(k)}}{\partial x}(i, j) &= \left((I^{(k)}(i + 1, j) \gg shift1) - (I^{(k)}(i - 1, j) \gg shift1)\right) \\ \frac{\partial I^{(k)}}{\partial y}(i, j) &= \left((I^{(k)}(i, j + 1) \gg shift1) - (I^{(k)}(i, j - 1) \gg shift1)\right) \end{aligned} \tag{3-25}$$

where $I^{(k)}(i, j)$ are the sample value at coordinate (i, j) of the prediction signal in list k, $k = 0, 1$, 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, S_1, S_2, S_3, S_5 and S_6, are calculated as

$$\begin{aligned} S_1 = \sum_{(i,j)\in\Omega} Abs(\psi_x(i,j)), \quad S_3 = \sum_{(i,j)\in\Omega} \theta(i,j) \bullet Sign(\psi_x(i,j)) \\ S_2 = \sum_{(i,j)\in\Omega} \psi_x(i,j) \bullet Sign(\psi_y(i,j)) \\ S_5 = \sum_{(i,j)\in\Omega} Abs(\psi_y(i,j)), \quad S_6 = \sum_{(i,j)\in\Omega} \theta(i,j) \bullet Sign(\psi_y(i,j)) \end{aligned} \tag{3-26}$$

where

$$\begin{aligned} \psi_x(i,j) &= \left(\frac{\partial I^{(1)}}{\partial x}(i, j) + \frac{\partial I^{(0)}}{\partial x}(i, j)\right) \gg n_a \\ \psi_y(i,j) &= \left(\frac{\partial I^{(1)}}{\partial y}(i, j) + \frac{\partial I^{(0)}}{\partial y}(i, j)\right) \gg n_a \\ \theta(i,j) &= \left(I^{(1)}(i, j) \gg n_b\right) - \left(I^{(0)}(i, j) \gg n_b\right) \end{aligned} \tag{3-27}$$

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

The motion refinement (v_x, v_y) is then derived using the cross- and auto-correlation terms using the following:

$$\begin{aligned} v_x &= S_1 > 0 ? clip3\left(-th'_{BIO}, th'_{BIO}, -\left(\left(S_3 \bullet 2^{n_b - n_a}\right) \gg \lfloor S_1 \rfloor\right)\right) : 0 \\ v_y &= S_5 > 0 ? clip3\left(-th'_{BIO}, th'_{BIO}, -\left(\left(\left(S_6 \bullet 2^{n_b - n_a} - \frac{\left(\left(v_x S_{2,m}\right) \ll n_{S_2} + v_x S_{2,s}\right)}{2}\right) \gg \lfloor S_5 \rfloor\right)\right)\right) : 0 \end{aligned} \tag{3-28}$$

where $S_{2,m} = S_2 \gg n_{S_2}, S_{2,s} = S_2 \& \left(2^{n_{S_2}} - 1\right), th'_{BIO} = 2^{max(5, BD-7)}$. $\lfloor \bullet \rfloor$ is the floor function, and $n_{S_2} = 12$.

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

$$b(x,y) = rnd\left(\left(v_x\left(\frac{\partial I^{(1)}(x,y)}{\partial x} - \frac{\partial I^{(0)}(x,y)}{\partial x}\right) + v_y\left(\frac{\partial I^{(1)}(x,y)}{\partial y} - \frac{\partial I^{(0)}(x,y)}{\partial y}\right) + 1\right)/2\right) \quad (3\text{-}29)$$

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

$$pred_{BDOF}(x,y) = \left(I^{(0)}(x,y) + I^{(1)}(x,y) + b(x,y) + o_{offset}\right) \gg shift \quad (3\text{-}30)$$

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 $I^{(k)}(i, j)$ in list k ($k = 0, 1$) outside of the current CU boundaries need to be generated. As depicted in Figure 34, the BDOF in VVC 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.

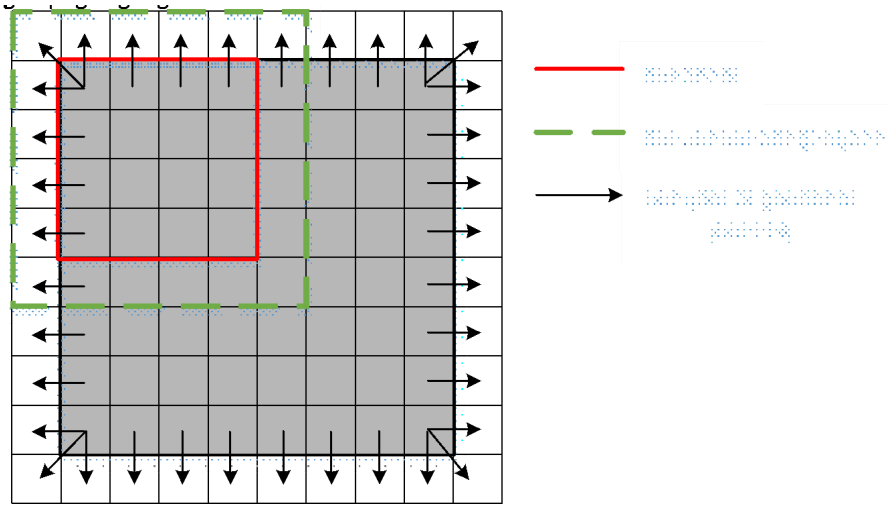


Figure 34 – Extended CU region used in BDOF

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.

3.4.10 Decoder side motion vector refinement (DMVR)

In order to increase the accuracy of the MVs of the merge mode, a bilateral-matching (BM) based decoder side motion vector refinement is applied in VVC. 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.

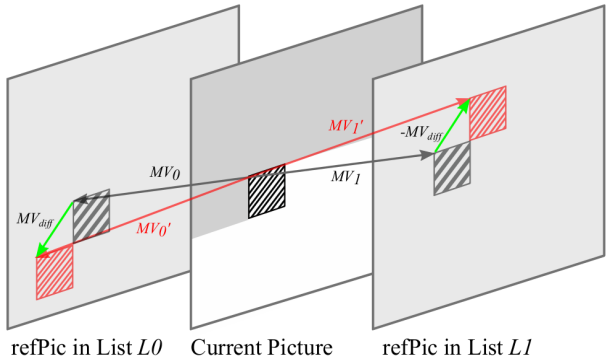


Figure 35 – Decoding side motion vector refinement

In VVC, the application of DMVR is restricted and is only 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 two reference pictures to the current picture are same

– Both reference pictures are short-term reference pictures

– 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 DMVR are mentioned in the following sub-clauses.

3.4.10.1 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:

$$MV0' = MV0 + MV\_offset \quad (3\text{-}31)$$

$$MV1' = MV1 - MV\_offset \quad (3\text{-}32)$$

Where MV_offset represents the refinement offset between the initial MV and the refined MV in one of the reference pictures. 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

$$E(x, y) = A\left(x - x_{min}\right)^2 + B\left(y - y_{min}\right)^2 + C \quad (3\text{-}33)$$

where (x_{min}, y_{min}) 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 (x_{min}, y_{min}) is computed as:

$$x_{min} = (E(-1, 0) - E(1, 0))/(2(E(-1, 0) + E(1, 0) - 2E(0, 0))) \quad (3\text{-}34)$$

$$y_{min} = (E(0, -1) - E(0, 1))/(2((E(0, -1) + E(0, 1) - 2E(0, 0))) \quad (3\text{-}35)$$

The value of x_{min} and y_{min} are automatically constrained to be between −8 and 8 since all cost values are positive and the smallest value is $E(0, 0)$. This corresponds to half peal offset with 1/16th-pel MV accuracy in VVC. The computed fractional (x_{min}, y_{min}) are added to the integer distance refinement MV to get the sub-pixel accurate refinement delta MV.

3.4.10.2 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 need 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.

3.4.10.3 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.

3.4.11 Geometric partitioning mode (GPM)

In VVC, a geometric partitioning mode is supported for inter prediction. The geometric partitioning 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. In total 64 partitions are supported by geometric partitioning mode for each possible CU size $w \times h = 2^m \times 2^n$ with $m, n \in \{3 \cdots 6\}$ excluding 8x64 and 64x8.

When this mode is used, a CU is split into two parts by a geometrically located straight line (Figure 36). The location of the splitting line is mathematically derived from the angle and offset parameters of a specific partition. Each part of a geometric partition in the CU is inter-predicted using its own motion; only uni-prediction is allowed for each partition, that is, each part 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.

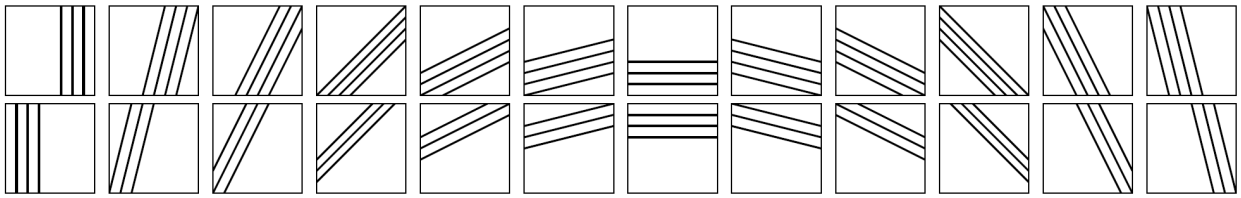


Figure 36 – Examples of the GPM splits grouped by identical angles

If geometric partitioning mode is used for the current CU, then a geometric partition index indicating the partition mode of the geometric partition (angle and offset), and two merge indices (one for each partition) are further signalled. The number of maximum GPM candidate size is signalled explicitly in SPS and specifies syntax binarization for GPM merge indices. After predicting each of part of the geometric partition, the sample values along the geometric partition edge are adjusted using a blending processing with adaptive weights as in 3.4.11.2. 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 geometric partition modes is stored as in 3.4.11.3.

3.4.11.1 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 geometric 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 geometric partitioning 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 geometric partitioning mode.

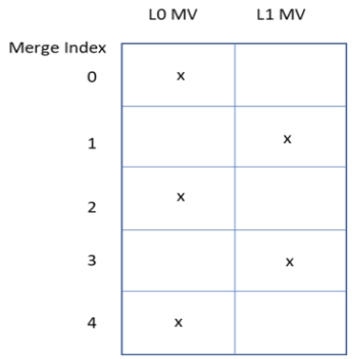


Figure 37 – Uni-prediction MV selection for geometric **partitioning mode**

3.4.11.2 Blending along the geometric partitioning edge

After predicting each part of a geometric partition using its own motion, blending is applied to the two prediction signals to derive samples around geometric partition edge. The blending weight for each position of the CU are derived based on the distance between individual position and the partition edge.

The distance for a position $(x,\ y)$ to the partition edge are derived as:

$$d(x,y) = (2x\ +\ 1\ -\ w)\cos cos\left(\varphi_i\right)\ +(2y\ +\ 1\ -\ h)\sin sin\left(\varphi_i\right)\ -\ \rho_j \tag{3-36}$$

$$\rho_j = \rho_{x,j}\cos cos\left(\varphi_i\right)\ +\rho_{y,j}\sin sin\left(\varphi_i\right) \tag{3-37}$$

$$\rho_{x,j} = \{0\ i\ \%\ 16\ =\ 8\ or\ (i\ \%\ 16\ \neq\ 0\ and\ h\ \geq\ w)\ \pm\ (j\ \times\ w)\gg 2\ otherwise \tag{3-38}$$

$$\rho_{y,j} = \{\pm\ (j\times h)\gg 2\ i\ \%\ 16\ =\ 8\ or\ (i\ \%\ 16 \neq 0\ and\ h \geq w)\ 0\ otherwise \tag{3-39}$$

where i, j are the indices for angle and offset of a geometric partition, which depend on the signaled geometric partition index. The sign of $\rho_{x,j}$ and $\rho_{y,j}$ depend on angle index i.

The weights for each part of a geometric partition are derived as following:

$$wIdxL(x,\ y) = partIdx\ ?\ 32\ +\ d(x,y)\colon 32\ -\ d(x,y) \tag{3-40}$$

$$w_0(x,y)\ = \frac{Clip3(0,\ 8,\ (wIdxL(x,y)+4)\gg 3)}{8} \tag{3-41}$$

$$w_1(x,y) = 1\ -\ w_0(x,y) \tag{3-42}$$

The partIdx depends on the angle index i. One example of weigh w_0 is illustrated in Figure 38.

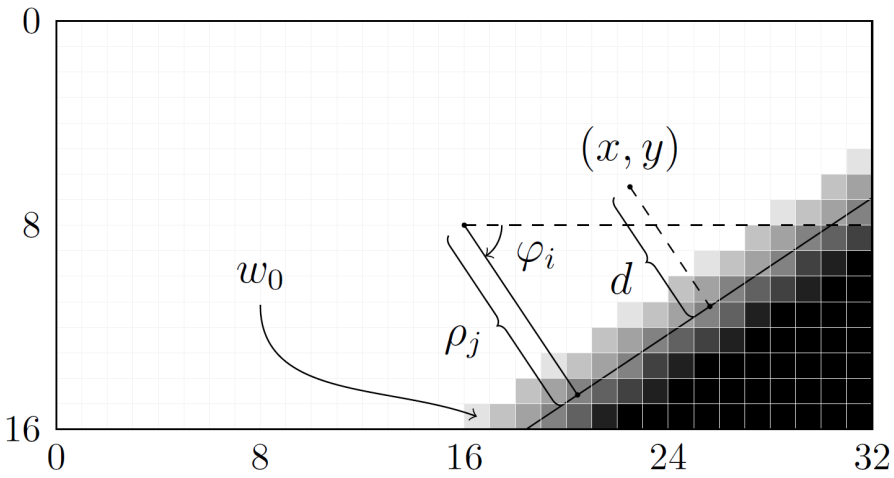


Figure 38 – Exemplified generation of a bending weight w_0 using geometric partitioning mode

3.4.11.3 Motion field storage for geometric partitioning mode

Mv1 from the first part of the geometric partition, Mv2 from the second part of the geometric partition and a combined Mv of Mv1 and Mv2 are stored in the motion filed of a geometric partitioning mode coded CU.

The stored motion vector type for each individual position in the motion filed are determined as:

$$sType = abs(motionIdx) < 32 \,?\, 2 : (motionIdx \leq 0 \,?\, (1 - partIdx) : partIdx) \tag{3-43}$$

where motionIdx is equal to $d(4x + 2, 4y + 2)$, which is recalculated from equation (3-36). The partIdx depends on the angle index i.

If sType is equal to 0 or 1, Mv0 or Mv1 are stored in the corresponding motion field, otherwise if sType is equal to 2, a combined Mv from Mv0 and Mv2 are stored. The combined Mv 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 vectors.
2) Otherwise, if Mv1 and Mv2 are from the same list, only uni-prediction motion Mv2 is stored.

3.4.12 Combined inter and intra prediction (CIIP)

In VVC, 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 P_{inter} is derived using the same inter prediction process applied to regular merge mode; and the intra prediction signal P_{intra} 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 39) 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 + isIntraTop) is equal to 2, then wt is set to 3;
- Otherwise, if (isIntraLeft + isIntraTop) is equal to 1, then wt is set to 2;
- Otherwise, set wt to 1.

The CIIP prediction is formed as follows:

$$P_{CIIP} = \left((4 - wt) * P_{inter} + wt * P_{intra} + 2\right) \gg 2 \tag{3-43}$$

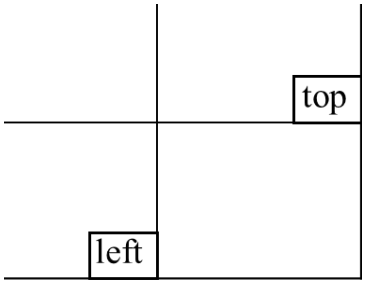


Figure 39 – Top and left neighboring blocks used in CIIP weight derivation

3.4.13 Reference picture resampling (RPR)

In HEVC, the spatial resolution of pictures cannot change unless a new sequence using a new SPS starts, with an IRAP picture. VVC enables picture resolution change within a sequence at a position without encoding an IRAP picture, which is always intra-coded. This feature is sometimes referred to as reference picture resampling (RPR), as the feature needs resampling of a reference picture used for inter prediction when that reference picture has a different resolution than the current picture being decoded. In order to avoid additional processing steps, the RPR process in VVC is designed to be embedded in the motion compensation process and performed at the block level. In the motion compensation stage, the scaling ratio is used together with motion information to locate the reference samples in the reference picture to be used in the interpolation process.

In VVC, the scaling ratio is restricted to be larger than or equal to 1/2 (2 times downsampling from the reference picture to the current picture), and less than or equal to 8 (8 times upsampling). Three sets of resampling filters with different frequency cutoffs are specified to handle various scaling ratios between a reference picture and the current picture. The three sets of resampling filters are applied respectively for the scaling ratio ranging from 1/2 to 1/1.75, from 1/1.75 to 1/1.25, and from 1/1.25 to 8. Each set of resampling filters has 16 phases for luma and 32 phases for chroma which is same to the case of motion compensation interpolation filters. It is worthy noted that the filter set of normal MC interpolation is used in the case of scaling ratio ranging from 1/1.25 to 8. Actually the normal MC interpolation process is a special case of the resampling process with scaling ratio ranging from 1/1.25 to 8. In addition to conventional translational block motion, the affine mode has three sets of 6-tap interpolation filters that are used for the luma component to cover the different scaling ratios in RPR. The horizontal and vertical scaling ratios are derived based on picture width and height, and the left, right, top and bottom scaling offsets specified for the reference picture and the current picture.

3.4.13.1 Signalling of resolution and cropping window

In VVC, the maximum picture resolution and the corresponding conformance cropping window are signalled in the SPS, while in the PPS the picture resolution of each current picture is signalled. Such signalling arrangement can be used to support RPR. When the picture resolution of the current picture (from PPS) is smaller than the maximum picture resolution (from SPS), it is possible to signal conformance cropping offsets in the PPS that are different from the conformance cropping offsets signalled in the SPS.

3.4.14 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.

3.5 Transform and quantization

3.5.1 Large block-size transforms with high-frequency zeroing

In VVC, 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. In addition, transform shift is removed in transform skip mode. The VTM also supports configurable max transform size in SPS, such that encoder has the flexibility to choose up to 32-length or 64-length transform size depending on the need of specific implementation.

3.5.2 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 3-8 shows the basis functions of the selected DST/DCT.

Table 3-8 - Transform basis functions of DCT-II/ VIII and DSTVII for N-point input

Transform Type	Basis function $T_i(j)$, $i, j = 0, 1, \ldots, N-1$
DCT-II	$T_i(j) = \omega_0 \cdot \sqrt{\frac{2}{N}} \cdot \cos cos\left(\frac{\pi \cdot i \cdot (2j+1)}{2N}\right)$ where, $\omega_0 = \{\sqrt{\frac{2}{N}}\, i = 0\ 1\ i \neq 0\ \omega_0 = \{\sqrt{\frac{2}{N}}\, i = 0\ 1\ i \neq 0$
DCT-VIII	$T_i(j) = \sqrt{\frac{4}{2N+1}} \cdot \cos cos\left(\frac{\pi \cdot (2i+1) \cdot (2j+1)}{4N+2}\right)$
DST-VII	$T_i(j) = \sqrt{\frac{4}{2N+1}} \cdot \sin sin\left(\frac{\pi \cdot (2i+1) \cdot (j+1)}{2N+1}\right)$

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 index is signalled to indicate which transform matrix is used for the horizontal and the vertical transform. Here, MTS is applied only for luma. The MTS signaling is skipped when one of the below conditions is applied.

- The position of the last significant coefficient for the luma TB is less than 1 (i.e., DC only)
- The last significant coefficient of the luma TB is located inside the MTS zero-out region

As shown by the transform and signalling mapping table in Table 3-9, the MTS CU index can have 5 values. If it is equal to zero, then DCT-2 is applied in both horizontal and vertical directions. Otherwise, a combination of DCT-8 and DST-7 will be used for the horizontal and vertical directions. Unified 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 3-9 - Transform and signalling mapping table

MTS index	Horizontal transform	Vertical transform
0	DCT2	DCT2
1	DST-7	DST-7
2	DCT-8	DST-7
3	DST-7	DCT-8
4	DCT-8	DCT-8

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. 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.

3.5.3 Extended precision processing (EPP)

In VVC version 2 the flag *sps_extended_precision_flag* is provided to allow increased internal precision of transformed coefficients for bit depths above 10-bit. The number of bits of precision is defined as:

maxLog2TrDynamicRange = *sps_extended_precision_flag* ? max(15, min(20, bitdepth + 6)) : 15

3.5.4 Low-frequency non-separable transform (LFNST)

In VVC, LFNST 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).

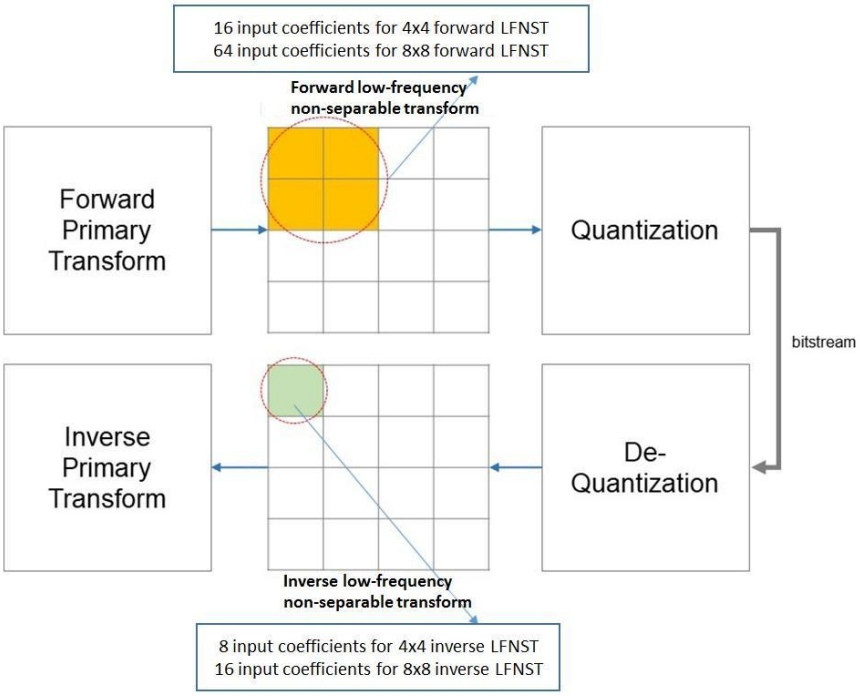


Figure 40 – Low-Frequency Non-Separable Transform (LFNST) process

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

$$X = \begin{bmatrix} X_{00} & X_{01} & X_{10} & X_{11} \\ X_{02} & X_{03} & X_{12} & X_{13} \\ X_{20} & X_{21} & X_{30} & X_{31} \\ X_{22} & X_{23} & X_{32} & X_{33} \end{bmatrix} \tag{3-44}$$

is first represented as a vector $\vec{X}$:

$$\vec{X} = \begin{bmatrix} X_{00} & X_{01} & X_{02} & X_{03} & X_{10} & X_{11} & X_{12} & X_{13} & X_{20} & X_{21} & X_{22} & X_{23} & X_{30} & X_{31} & X_{32} & X_{33} \end{bmatrix}^T \tag{3-45}$$

The non-separable transform is calculated as $\vec{F} = T \cdot \vec{X}$, where $\vec{F}$ indicates the transform coefficient vector, and T is a 16x16 transform matrix. The 16x1 coefficient vector $\vec{F}$ 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.

3.5.4.1 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:

$$T_{RxN} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & \cdots & t_{1N} \\ t_{21} & t_{22} & t_{23} & & t_{2N} \\ & \vdots & & \ddots & \vdots \\ t_{R1} & t_{R2} & t_{R3} & \cdots & t_{RN} \end{bmatrix} \tag{3-46}$$

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.

3.5.4.2 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 3-10. 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 3-10 - 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

3.5.4.3 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.

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. The LFNST index signaling is placed before MTS index signaling.

The use of scaling matrices for perceptual quantization is not evident that the scaling matrices that are specified for the primary matrices may be useful for LFNST coefficients. Hence, the uses of the scaling matrices for LFNST coefficients are not allowed. For single-tree partition mode, chroma LFNST is not applied.

3.5.5 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.

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

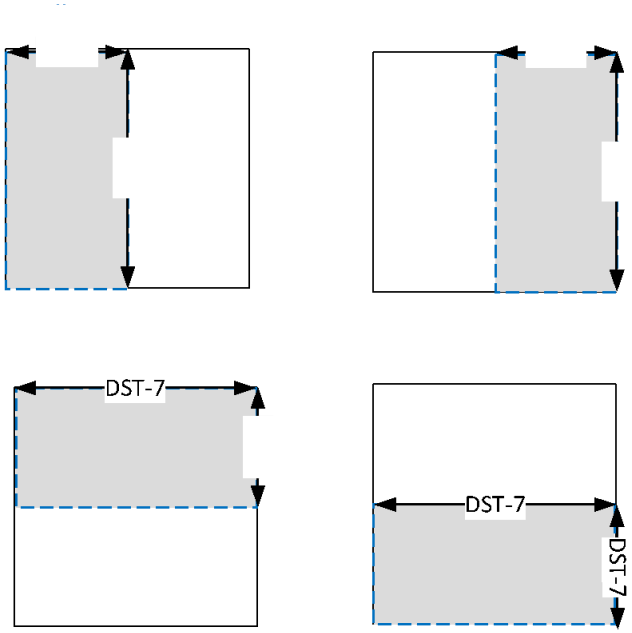


Figure 41 – SBT position, type and transform type

3.5.6 Quantization

3.5.6.1 Quantization parameter control

In VVC, 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 QP_Y to chroma quantization parameter QP_C. 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 [−$QpBdOffset_Y$, 63] and the output range of the piecewise linear model is [−$QpBdOffset_C$, 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. CU chroma QP offset signalling is also consistent with the VPDU CU QP delta availability, and for CU larger than 64x64, send the chroma QP offset with the first transform unit regardless of whether it has non-zero CBF or not

3.5.6.2 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.

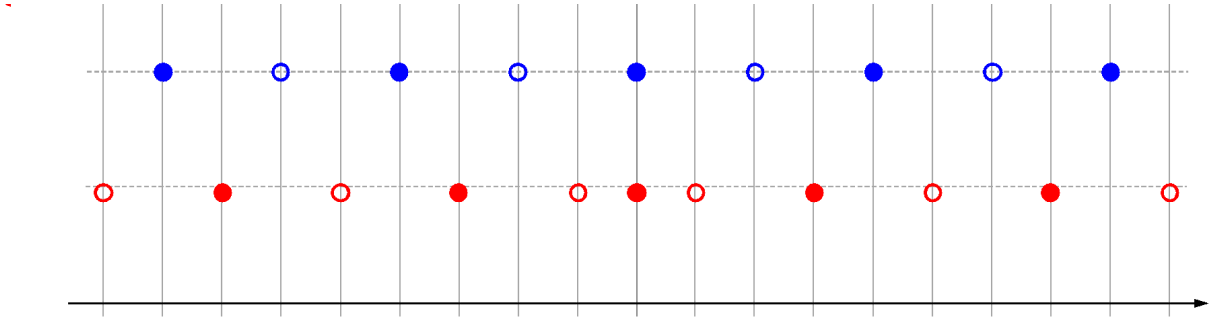


Figure 42 – Illustration of the two scalar quantizers used in the proposed approach of dependent quantization.

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.

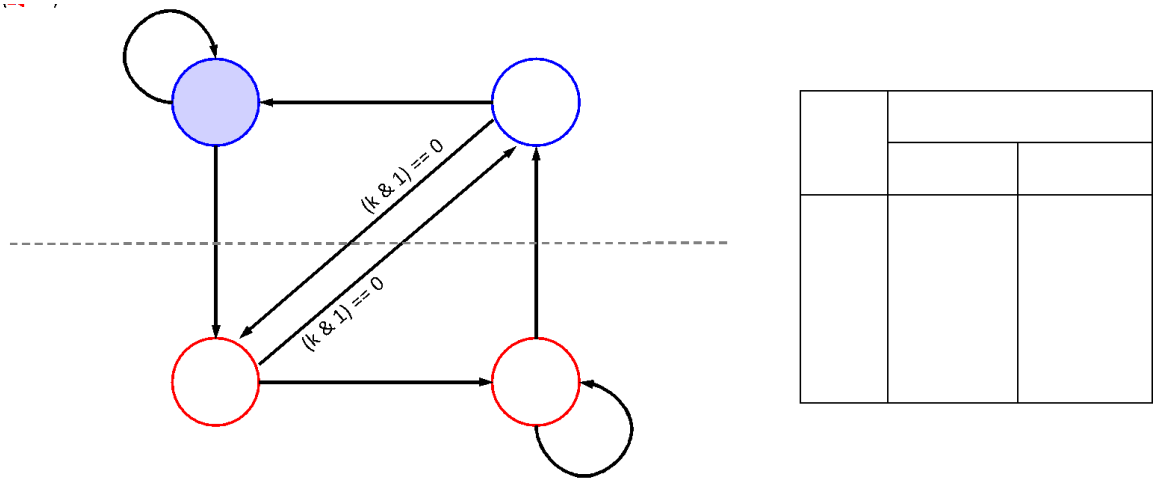


Figure 43 – State transition and quantizer selection for the proposed dependent quantization.

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 43, where k denotes the value of the transform coefficient level.

3.5.6.3 Scaling matrices

VVC supports to use the default scaling matrices or signal 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 4×4 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 upsampled (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. In VVC, 2x2, 2x4, and 4x2 chroma intra coding blocks (CBs) do not exist, and the smallest intra block size is equal to 2x8 and 8x2, as well as the smallest chroma intra block copy (IBC) block size. Furthermore, inter-prediction is disabled for 4x4 luma CBs. Therefore, small 2x2 chroma blocks can be created only by applying a subblock transform (SBT). Considering these essences, 2x2 intra chroma quantization matrices (QMs) are removed from the default QM list, and not code user-defined intra QMs for this size.

In order to improve coding efficiency for user defined quantization matrixes (QM), following approaches are considered.

- Allow referencing a previously coded QM with the same base size as the current QM.
- Allow coding element-to-element differences between the current QM and the reference QM.
- Keep the original DPCM coding of elements within the current QM.
- Use a single matrix identifier scalingListId which combines matrixId and sizeId.

3.5.7 Joint coding of chroma residuals (JCCR)

VVC supports the joint coding of chroma residual (JCCR) tool where the chroma residuals are coded jointly. The usage (activation) of the JCCR 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 JCCR 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 some blocks coded using the JCCR mode. The JCCR mode has 3 sub-modes. When a corresponding JCCR sub-mode (sub-modes 2 in Table 3-11) 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 JCCR sub-modes (sub-modes 1 and 3 in Table 3-11), 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 3-11. When

the JCCR mode is activated, one single joint chroma residual block (resJointC[x][y] in Table 3-11) 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 3-11 - 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 sub-modes described above in Table 3-11 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.

The JCCR mode can be combined with the chroma transform skip (TS) mode (more details of the TS mode can be found in section 3.9.3). To speed up the encoder decision, the JCCR transform selection depends on whether the independent coding of Cb and Cr components selects the DCT-2 or the TS as the best transform, and whether there are non-zero coefficients in independent chroma coding. Specifically, if one chroma component selects DCT-2 (or TS) and the other component is all zero, or both chroma components select DCT-2 (or TS), then only DCT-2 (or TS) will be considered in JCCR encoding. Otherwise, if one component selects DCT-2 and the other selects TS, then both DCT-2 and TS will be considered in JCCR encoding.

3.6 Entropy coding

In the VVC, 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

3.6.1 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 64x4 8-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.

$$ivlLpsRange = rangeTabLps[\ pStateIdx\ ][\ qRangeIdx\ ] \tag{3-47}$$

In VVC, the probability is linearly expressed by the probability index pStateIdx. Therefore, all the calculation can be done with equations without LUT operation. To improve the accuracy of probability estimation, a multi-hypothesis probability update model is applied. The pStateIdx used in the interval subdivision in the binary arithmetic coder is a combination of two probabilities pStateIdx0 and pStateIdx1. The two probabilities are associated with each context model and are updated independently with different adaptation rates. The adaptation rates of pStateIdx0 and pStateIdx1 for each context model are pre-trained based on the statistics of the associated bins. The probability estimate pStateIdx is the average of the estimates from the two hypotheses.

Figure 44 shows the flowchart for decoding a single binary decision in VVC.

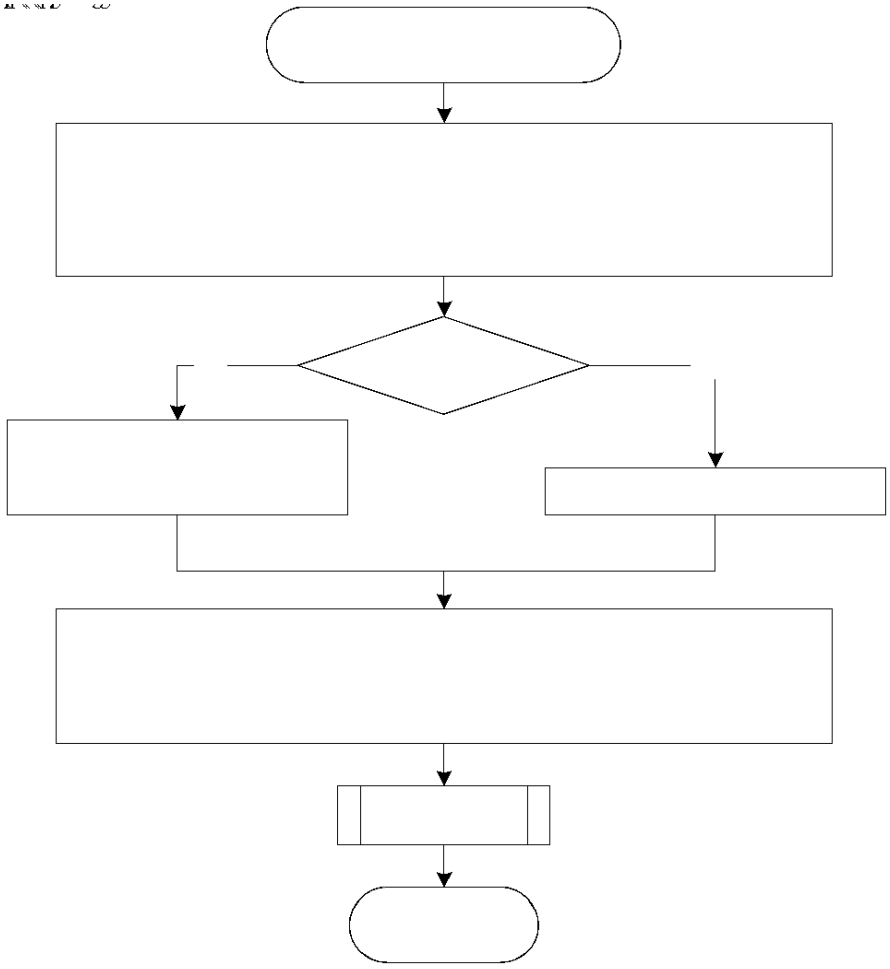


Figure 44 – Flowchart for decoding a bin

As done in HEVC, VVC CABAC also 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, is derived as follows

$$m = slopeIdx \times 5 - 45 \tag{3-48}$$

$$n = (offsetIdx << 3) + 7 \tag{3-49}$$

$$preCtxState = Clip3(1, 127, ((m \times (QP - 32)) >> 4) + n) \quad (3\text{-}50)$$

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.

$$pStateIdx0 = preCtxState << 3 \quad (3\text{-}51)$$

$$pStateIdx1 = preCtxState << 7 \quad (3\text{-}52)$$

3.6.2 Last significant coefficient coding

In VVC version 1, the position (x, y) of the last significant coefficient in the TB in scanning order is coded before coding subblocks of coefficient values. The values x and y are coded using a combination of content-coded and equal probability-coded bins.

In VVC version 2, a new flag, sh_reverse_last_sig_coeff_flag, has been added to support an alternative method of coding the position of the last significant coefficient. If sh_reverse_last_sig_coeff_flag is set to 1, the values coded are ((Log2ZoTbWidth << 1) – 1 – x) and ((Log2ZoTbHeight << 1) − 1 – y), where Log2ZoTbWidth and Log2ZoTbHeight are the log2 of the width and height of the non-zeroed out portion of the TB.

3.6.3 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, 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.

3.6.3.1 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 are 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 45. 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. The Rice parameter is determined not only based on the sum of the absolute levels of five neighboring transform coefficients in the local template as shown in Figure 46, but also the corresponding base level is also taken into consideration as follow:

$$\text{RicePara} = \text{RiceParTable}[\ \max(\min(\ 31, \text{sumAbs} - 5 * \text{baseLevel}), 0)\ ] \qquad (3\text{-}53)$$

When computing the value sumAbs, a value of 0 is used for any of the neighboring coefficients outside the TB.

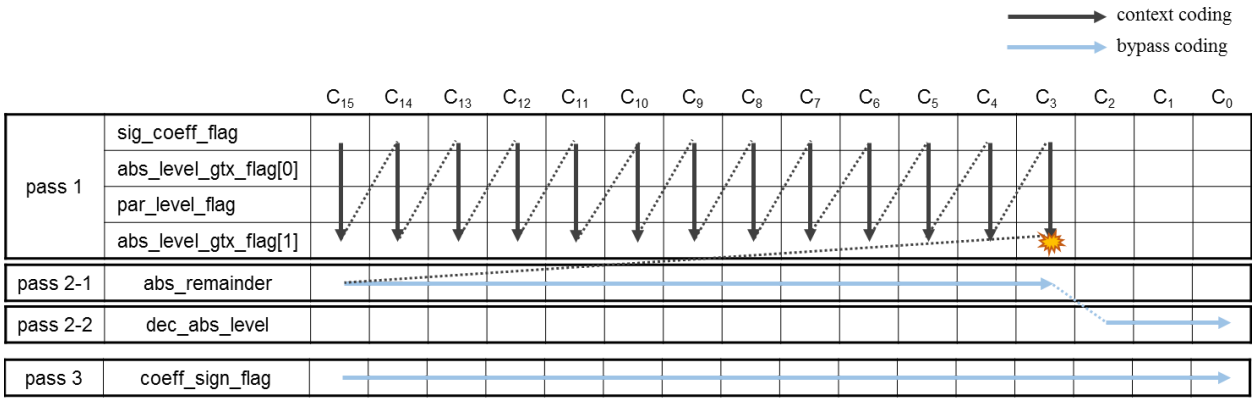


Figure 45 – residual coding structure for transform blocks

After the termination of the 1st subblock coding pass, the absolute value of each of the remaining yet-to-be-coded coefficients is coded by the syntax element dec_abs_level, which corresponds to a modified absolute level value with the zero-level value being conditionally mapped to a nonzero value. At the encoder side, the value of syntax element dec_abs_level is derived from the absolute level (absLevel), dependent quantizer state (QState) and the value of Rice parameter (RicePara) as follows:

```
ZeroPos = ( QState < 2? 1 : 2 ) << RicePara
if (absLevel == 0)
        dec_abs_level = ZeroPos
else
        dec_abs_level = (absLevel <= ZeroPos) ? (absLevel – 1) : absLevel
```

3.6.3.2 Residual coding for transform coefficients in VVC version 2

Version 2 of VVC extends support to bit depths above 10 bits when typically coefficients will have a greater magnitude. To efficiently code such coefficients a wider range of Rice parameters is required. For this purpose two new tools have been included in VVC version 2: Extended Regular Residual Coding (ERRC); and Persistent Regular Residual Coding (PRRC).

ERRC is enabled using sps_rrc_rice_extension_flag. When set equal to 1 the following two changes are made to the algorithm for the residual coding of transform coefficients described above:

1. The baseLevel for Pass 2 is set to the following values:
   - 1 if the bit depth is over 12 and the current slice is an I-slice
   - 2 if the bit depth is over 12 and the current slice is not an I-slice
   - 2 if the bit depth is 12 or less and the current slice is an I-slice
   - 3 if the bit depth is 12 or less and the current slice is not an I-slice
2. Formula (3-53) is modified as below:

$$\text{RicePara} = \text{RiceParTable}[\ \max(\min(\ 31, (\text{sumAbs} >> \text{shiftVal}) - 5 * \text{baseLevel}), 0)\ ] + \text{shiftVal} \qquad (3\text{-}54)$$

where shiftVal adapts sumAbs using a right shift so that the resultant value is more likely in the range 0..31 and compensates for this adaption after using the look up table RiceParTable. The value of shiftVal is computed using a pair of look up tables to compute an even integer which represents an approximation of the log2 value of sumAbs.

PRRC is enabled using sps_persistent_rice_adaptation_enabled_flag. When set, any neighboring coefficients outside the TB are assumed to have the value HistValue when computing sumAbs. HistValue is computed at the beginning of each TB using a value computed from coefficients coded in previous TBs of the current channel in the current slice.

$$HistValue = (1 << StatCoeff) \tag{3-55}$$

where

$$StatCoeff = (StatCoeff + Floor(Log2(abs\_remainder)) + 2) >> 1 \tag{3-56}$$

where StatCoeff is only updated for the first coefficient coded with abs_remainder in the TB.

StatCoeff is initialised as following at the start of the first CTU in a slice or tile:

$$StatCoeff[\ cIdx\ ] = sps\_persistent\_rice\_adaptation\_enabled\_flag\ ? \tag{3-57}$$

$$2 * Floor(Log2(BitDepth - 10) : 0$$

When wavefront parallel processing (WPP) is enabled, StatCoeff is additionally initialised to this value at the start of every CTU row in the slice or tile.

3.6.3.3 Residual coding for transform skip

Similar to HEVC, VVC supports transform skip mode. Transform skip mode is allowed for luma and chroma blocks. 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.

3.6.4 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 46.

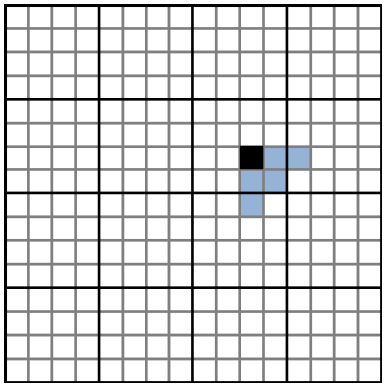


Figure 46: Illustration of the template used for selecting probability models. The black square specifies the current scan position and the blue squares represent the local neighbourhood used.

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 VVC 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:

$$\begin{aligned} ctxInc = {} & 12 * Max(0, QState - 1) + \\ & Min((locSumAbsPass1 + 1) >> 1, 3) + (d < 2 \ ? \ 8 \ : \ (d < 5 \ ? \ 4 \ : \ 0)) \end{aligned} \tag{3-58}$$

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

$$\begin{aligned} ctxInc = {} & 36 + 8 * Max(0, QState - 1) + \\ & Min((locSumAbsPass1 + 1) >> 1, 3) + (d < 2 \ ? \ 4 \ : \ 0) \end{aligned} \tag{3-59}$$

3.7 In-loop filters

There are totally three in-loop filters in VVC. Besides deblocking filter and SAO (the two loop filters in HEVC), adaptive loop filter (ALF) are applied. The ALF comprises of luma ALF, chroma ALF and cross-component ALF (CC-ALF). The ALF filtering process is designed so that luma ALF, chroma ALF and CC-ALF can be executed in parallel. The order of the filtering process in the VVC is the deblocking filter, SAO and ALF. The SAO in VVC is the same as that in HEVC.

In VVC, a new process called the luma mapping with chroma scaling was added (this process was previously known as the adaptive in-loop reshaper). The LMCS modifies the sample values before encoding and after reconstruction by redistributing the codewords across the entire dynamic range. This new process is performed before deblocking.

3.7.1 Adaptive Loop Filter

In VVC, 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.

3.7.1.1 Filter shape

Two diamond filter shapes (as shown in Figure 47) are used. The 7×7 diamond shape is applied for luma component and the 5×5 diamond shape is applied for chroma components.

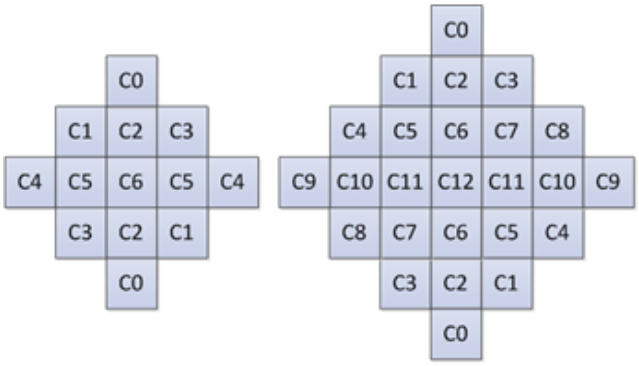


Figure 47 – ALF filter shapes (chroma: 5×5 diamond, luma: 7×7 diamond)

3.7.1.2 Block classification

For luma component, each 4×4 block is categorized into one out of 25 classes. The classification index C is derived based on its directionality D and a quantized value of activity $\hat{A}$, as follows:

$$C = 5D + \hat{A} \tag{3-60}$$

To calculate D and $\hat{A}$, gradients of the horizontal, vertical and two diagonal direction are first calculated using 1-D Laplacian:

$$g_v = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} V_{k,l}, \quad V_{k,l} = |2R(k,l) - R(k,l-1) - R(k,l+1)| \tag{3-61}$$

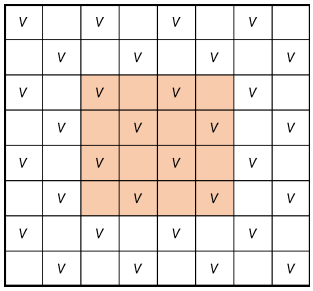
$$g_h = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} H_{k,l}, \quad H_{k,l} = |2R(k,l) - R(k-1,l) - R(k+1,l)| \tag{3-62}$$

$$g_{d1} = \sum_{k=i-2}^{i+3} \sum_{l=j-3}^{j+3} D1_{k,l}, \quad D1_{k,l} = |2R(k,l) - R(k-1,l-1) - R(k+1,l+1)| \tag{3-63}$$

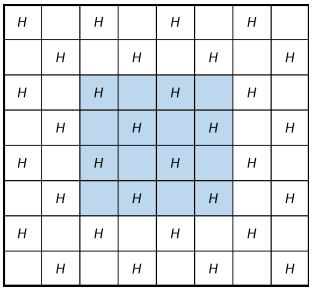
$$g_{d2} = \sum_{k=i-2}^{i+3} \sum_{j=j-2}^{j+3} D2_{k,l}, \quad D2_{k,l} = |2R(k,l) - R(k-1,l+1) - R(k+1,l-1)| \tag{3-64}$$

Where indices i and j refer to the coordinates of the upper left sample within the 4×4 block and $R(i,j)$ indicates a reconstructed sample at coordinate (i,j).

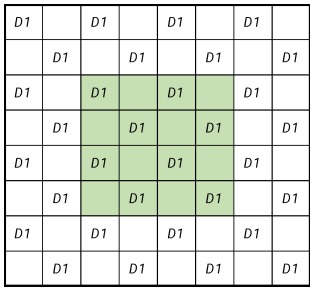
To reduce the complexity of block classification, the subsampled 1-D Laplacian calculation is applied. As shown in Figure 48, the same subsampled positions are used for gradient calculation of all directions.



(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 48 – Subsampled Laplacian calculation

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

$$g_{h,v}^{max} = max\left(g_h, g_v\right), \;\; g_{h,v}^{min} = min\left(g_h, g_v\right) \tag{3-65}$$

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

$$g_{d0,d1}^{max} = max\left(g_{d0}, g_{d1}\right), \;\; g_{d0,d1}^{min} = min\left(g_{d0}, g_{d1}\right) \tag{3-66}$$

To derive the value of the directionality D, these values are compared against each other and with two thresholds t_1 and t_2:

Step 1. If both $g_{h,v}^{max} \le t_1 \bullet g_{h,v}^{min}$ and $g_{d0,d1}^{max} \le t_1 \cdot g_{d0,d1}^{min}$ are true, D is set to 0.

Step 2. If $\frac{g_{h,v}^{max}}{g_{h,v}^{min}} > \frac{g_{d0,d1}^{max}}{g_{d0,d1}^{min}}$, continue from Step 3; otherwise continue from Step 4.

Step 3. If $g_{h,v}^{max} > t_2 \bullet g_{h,v}^{min}$, D is set to 2; otherwise D is set to 1.

Step 4. If $g_{d0,d1}^{max} > t_2 \bullet g_{d0,d1}^{min}$, D is set to 4; otherwise D is set to 3.

The activity value A is calculated as:

$$A = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} \left(V_{k,l} + H_{k,l}\right) \tag{3-67}$$

A is further quantized to the range of 0 to 4, inclusively, and the quantized value is denoted as $\hat{A}$.

For chroma components in a picture, no classification method is applied.

3.7.1.3 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 $f(k, l)$ and to the corresponding filter clipping values $c(k, l)$ 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: $f_D(k,l) = f(l,k), c_D(k,l) = c(l,k),$ (3-68)

Vertical flip: $f_V(k,l) = f(k, K-l-1), c_V(k,l) = c(k, K-l-1)$ (3-69)

Rotation: $f_R(k,l) = f(K-l-1, k), c_R(k,l) = c(K-l-1, k)$ (3-70)

where K is the size of the filter and $0 \leq k, l \leq K-1$ are coefficients coordinates, such that location $(0,0)$ is at the upper left corner and location $(K-1, K-1)$ is at the lower right corner. The transformations are applied to the filter coefficients $f(k, l)$ and to the clipping values $c(k, l)$ depending on gradient values calculated for that block. The relationship between the transformation and the four gradients of the four directions are summarized in the following table.

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

Gradient values	Transformation
$g_{d2} < g_{d1}$ and $g_h < g_v$	No transformation
$g_{d2} < g_{d1}$ and $g_v < g_h$	Diagonal
$g_{d1} < g_{d2}$ and $g_h < g_v$	Vertical flip
$g_{d1} < g_{d2}$ and $g_v < g_h$	Rotation

3.7.1.4 Filtering process

At decoder side, when ALF is enabled for a CTB, each sample $R(i,j)$ within the CU is filtered, resulting in sample value $R'(i,j)$ as shown below,

$$R'(i,j) = R(i,j) + \left(\left(\sum_{k \neq 0} \sum_{l \neq 0} f(k,l) \times K(R(i+k, j+l) - R(i,j), c(k,l)) + 64 \right) \gg 7 \right) \quad (3\text{-}71)$$

where $f(k, l)$ denotes the decoded filter coefficients, $K(x, y)$ is the clipping function and $c(k, l)$ denotes the decoded clipping parameters. The variable k and l varies between $-\frac{L}{2}$ and $\frac{L}{2}$ where L denotes the filter length. The clipping function $K(x, y) = min(y, (-y, x))$ which corresponds to the function $Clip3(-y, y, x)$. The clipping operation introduces non-linearity to make ALF more efficient by reducing the impact of neighbor sample values that are too different with the current sample value.

3.7.1.5 Cross component adaptive loop filter

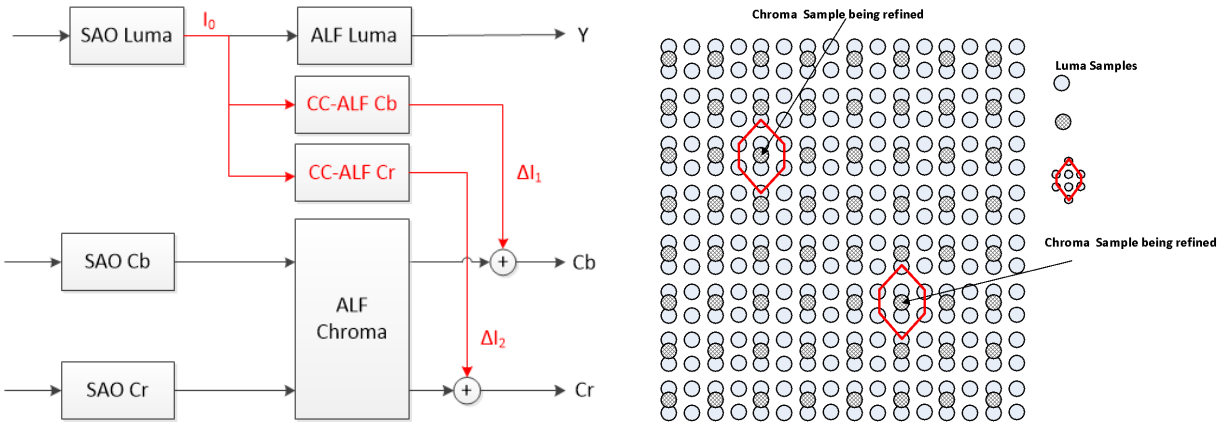
CC-ALF uses luma sample values to refine each chroma component by applying an adaptive, linear filter to the luma channel and then using the output of this filtering operation for chroma refinement. Figure 49 (a) provides a system level diagram of the CC-ALF process with respect to the SAO, luma ALF and chroma ALF processes.

Filtering in CC-ALF is accomplished by applying a linear, diamond shaped filter (Figure 49 (b)) to the luma channel. One filter is used for each chroma channel, and the operation is expressed as

$$\Delta I_i(x,y) = \sum_{(x_0, y_0) \in S_i} I_0\left(x_Y + x_0, y_Y + y_0\right) c_i\left(x_0, y_0\right) \quad (3\text{-}72)$$

where (x, y) is chroma component i location being refined $\left(x_Y, y_Y\right)$ is the luma location based on (x, y), S_i is filter support area in luma component, $c_i\left(x_0, y_0\right)$ $represents\ the\ filter\ coefficients$

As shown in Figure 51b, the luma filter support is the region collocated with the current chroma sample after accounting for the spatial scaling factor between the luma and chroma planes.



(a) **(b)**

Figure 49 – (a) Placement of CC-ALF with respect to other loop filters (b) Diamond shaped filter

In the VVC reference software, CC-ALF filter coefficients are computed by minimizing the mean square error of each chroma channels with respect to the original chroma content. To achieve this, the VTM algorithm uses a coefficient derivation process similar to the one used for chroma ALF. Specifically, a correlation matrix is derived, and the coefficients are computed using a Cholesky decomposition solver in an attempt to minimize a mean square error metric. In designing the filters, a maximum of 8 CC-ALF filters can be designed and transmitted per picture. The resulting filters are then indicated for each of the two chroma channels on a CTU basis.

Additional characteristics of CC-ALF include:

- The design uses a 3x4 diamond shape with 8 taps.
- Seven filter coefficients are transmitted in the APS.
- Each of the transmitted coefficients has a 6-bit dynamic range and is restricted to power-of-2 values.
- The eighth filter coefficient is derived at the decoder such that the sum of the filter coefficients is equal to 0.
- An APS may be referenced in the slice header.
- CC-ALF filter selection is controlled at CTU-level for each chroma component
- Boundary padding for the horizontal virtual boundaries uses the same memory access pattern as luma ALF.

As an additional feature, the reference encoder can be configured to enable some basic subjective tuning through the configuration file. When enabled, the VTM attenuates the application of CC-ALF in regions that are coded with high QP and are either near mid-grey or contain a large amount of luma high frequencies. Algorithmically, this is accomplished by disabling the application of CC-ALF in CTUs where any of the following conditions are true:

- The slice QP value minus 1 is less than or equal to the base QP value
- The number of chroma samples for which the local contrast is greater than (1 << (bitDepth – 2)) – 1 exceeds the CTU height, where the local contrast is the difference between the maximum and minimum luma sample values within the filter support region.
- More than a quarter of chroma samples are in the range between (1 << (bitDepth – 1)) – 16 and (1 << (bitDepth – 1)) + 16

The motivation for this functionality is to provide some assurance that CC-ALF does not amplify artifacts introduced earlier in the decoding path (This is largely due the fact that the VTM currently does not explicitly optimize for chroma subjective quality). It is anticipated that alternative encoder implementations would either not use this functionality or incorporate alternative strategies suitable for their encoding characteristics.

3.7.1.6 Filter parameters signalling

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 table of clipping values for both luma and Chroma components. These clipping values are dependent of the internal bitdepth. More precisely, the clipping values are obtained by the following formula:

$$\text{AlfClip}= \left\{ round\left(2^{B-*n}\right) for\ n \in [0..N\ -\ 1] \right\} \tag{3-73}$$

with B equal to the internal bitdepth, α is a pre-defined constant value equal to 2.35, and N equal to 4 which is the number of allowed clipping values in VVC. The AlfClip is then rounded to the nearest value with the format of power of 2.

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 -2^7 to $2^7 - 1$, inclusive. The central position coefficient is not signalled in the bitstream and is considered as equal to 128.

3.7.1.7 Virtual boundary filtering process for line buffer reduction

In VVC, 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 50, with N equal to 4 for the Luma component and 2 for the Chroma component.

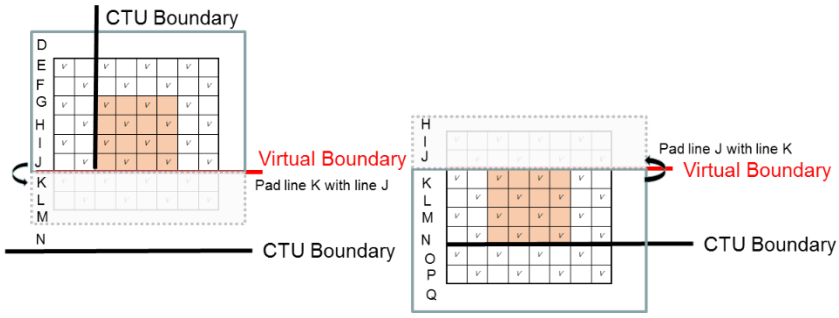


Figure 50 – Modified block classification at virtual boundaries

Modified block classification is applied for the Luma component as depicted in Figure 50. 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 51, 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.

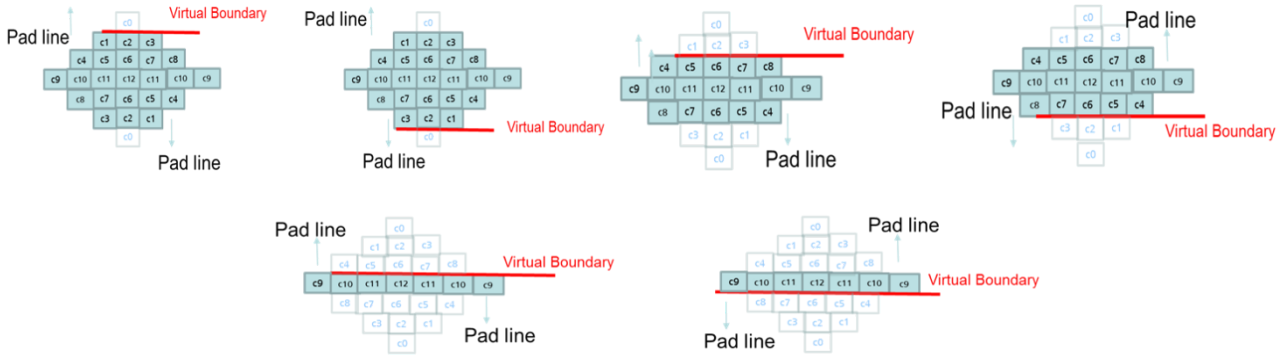


Figure 51 – Modified ALF filtering for Luma component at virtual boundaries

Different to the symmetric padding method used at horizontal CTU boundaries, simple padding process is applied for slice, tile and subpicture boundaries when filter across the boundaries is disabled. The simple padding process is also applied at picture boundary. The padded samples are used for both classification and filering process. To compensate for the extreme padding when filtering samples just above or below the virtual boundary, the filter strength is reduced for those cases for both luma and chroma by increasing the right shift in equation 3-61 [Ed YY: not a linked reference] by 3.

3.7.2 Deblocking filter

Deblocking filtering process is similar to those in HEVC. In VVC, the deblocking filtering process is applied on a CU boundaries, transform subblock boundaries and prediction subblock boundaries. The prediction subblock boundaries include the prediction unit boundaries introduced by the SbTMVP 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. As done in HEVC, the processing order of the deblocking filter is defined as horizontal filtering for vertical edges for the entire picture first, followed by vertical filtering for horizontal edges. This specific order enables either multiple horizontal filtering or vertical filtering processes to be applied in parallel threads, or can still be implemented on a CTB-by-CTB basis with only a small processing latency. Compared to HEVC deblocking, the following modifications are introduced.

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

3.7.2.1 Luma-adaptive deblocking filter strength

As done in HEVC, the filter strength of the deblocking filter in VVC is controlled by the variables β and t_C which are derived from the averaged quantization parameters qP_L of the two adjacent coding blocks. In

VVC, luma-adaptive deblocking can further adjust the filtering strength of deblocking filter based on the averaged luma level of the reconstructed samples. This additional refinement is used to compensate the nonlinear transfer function such as Electro-Optical Transfer Function (EOTF) in the linear light domain.

In this method, deblocking filter controls the strength of the deblocking filter by adding offset to qP_L according to the luma level of the reconstructed samples. The reconstructed luma level LL is derived as follow:

$$LL= ((p_{0,0} + p_{0,3} + q_{0,0} + q_{0,3}) >> 2) / (1 << bitDepth) \quad (3\text{-}74)$$

where, the sample values $p_{i,k}$ and $q_{i,k}$ with i = 0..3 and k = 0 and 3 are derived as shown in Figure 52.

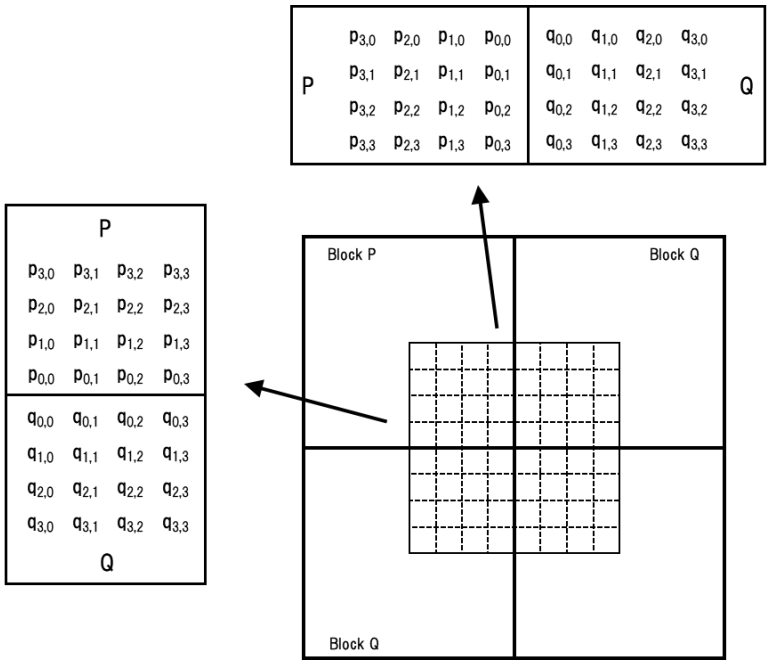


Figure 52 – Sample position of $p_{i,k}$ and $q_{i,k}$

The variable qP_L is derived as follows:

$$qP_L = ((Qp_Q + Qp_P +1) >> 1) + qpOffset \quad (3\text{-}75)$$

where Qp_Q and Qp_P denote the quantization parameters of the coding units containing the sample $q_{0,0}$ and $p_{0,0}$, respectively. The offset qpOffset is dependent on transfer function and the reconstructed luma level LL. The mapping function of qpOffset and the luma level are signalled in the SPS and should be derived according to the transfer characteristics of the contents since the transfer functions vary among video formats.

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

In VVC, 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, 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]

3.7.2.3 4x4 deblocking grid for luma

HEVC uses an 8x8 deblocking grid for both luma and chroma. In VVC, 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.

3.7.2.4 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 p_i for i = 0 to Sp − 1 and q_i for j = 0 to Sq − 1 are then replaced by linear interpolation as follows:

$$p_i' = (f_i * Middle_{s,t} + (64 - f_i) * P_s + 32) \gg 6),\ clipped\ to\ p_i \pm tcPD_i \quad (3\text{-}76)$$

$$q_j' = (g_j * Middle_{s,t} + (64 - g_j) * Q_s + 32) \gg 6),\ clipped\ to\ q_j \pm tcPD_j \quad (3\text{-}77)$$

where $tcPD_i$ and $tcPD_j$ term is a position dependent clipping and g_j, f_i, $Middle_{s,t}$, P_s and Q_s are given below:

Table 3-13 – Derivation of stronger deblocking parameters for luma

Sp, Sq 7, 7 (p side: 7, q side: 7)	$f_i = 59 - i * 9,\ can\ also\ be\ described\ as\ f = \{59, 50, 41, 32, 23, 14, 5\}$ $g_j = 59 - j * 9,\ can\ also\ be\ described\ as\ g = \{59, 50, 41, 32, 23, 14, 5\}$ $Middle_{7,7} = (2 * (p_o + q_o) + p_1 + q_1 + p_2 + q_2 + p_3 + q_3 + p_4 + q_4 + p_5 + q_5 + p_6 +$ $P_7 = (p_6 + p_7 + 1) \gg 1, \quad Q_7 = (q_6 + q_7 + 1) \gg 1$
7, 3 (p side: 7 q side: 3)	$f_i = 59 - i * 9,\ can\ also\ be\ described\ as\ f = \{59, 50, 41, 32, 23, 14, 5\}$ $g_j = 53 - j * 21,\ can\ also\ be\ described\ as\ g = \{53, 32, 11\}$ $Middle_{7,3} = (2 * (p_o + q_o) + q_0 + 2 * (q_1 + q_2) + p_1 + q_1 + p_2 + p_3 + p_4 + p_5 + p_6$ $P_7 = (p_6 + p_7 + 1) \gg 1, \quad Q_3 = (q_2 + q_3 + 1) \gg 1$
3, 7 (p side: 3 q side: 7)	$g_j = 59 - j * 9,\ can\ also\ be\ described\ as\ g = \{59, 50, 41, 32, 23, 14, 5\}$ $f_i = 53 - i * 21,\ can\ also\ be\ described\ as\ f = \{53, 32, 11\}$ $Middle_{3.7} = (2 * (q_o + p_o) + p_0 + 2 * (p_1 + p_2) + q_1 + p_1 + q_2 + q_3 + q_4 + q_5 + q_6$ $Q_7 = (q_6 + q_7 + 1) \gg 1, \quad P_3 = (p_2 + p_3 + 1) \gg 1$
7, 5 (p side: 7 q side: 5)	$g_j = 58 - j * 13,\ can\ also\ be\ described\ as\ g = \{58, 45, 32, 19, 6\}$ $f_i = 59 - i * 9,\ can\ also\ be\ described\ as\ f = \{59, 50, 41, 32, 23, 14, 5\}$ $Middle7,5 = (2 * (p_o + q_o + p_1 + q_1) + q_2 + p_2 + q_3 + p_3 + q_4 + p_4 + q_5 + p_5 +$

	$Q_5 = (q_4 + q_5 + 1) \gg 1, \quad P_7 = (p_6 + p_7 + 1) \gg 1$
5, 7 (p side: 5 q side: 7)	$g_j = 59 - j * 9$, *can also be described as* $g = \{59, 50, 41, 32, 23, 14, 5\}$ $f_i = 58 - i * 13$, *can also be described as* $f = \{58, 45, 32, 19, 6\}$ $Middle5,7 = (2 * (q_o + p_o + p_1 + q_1) + q_2 + p_2 + q_3 + p_3 + q_4 + p_4 + q_5 + p_5 +$ $Q_7 = (q_6 + q_7 + 1) \gg 1, \quad P_5 = (p_4 + p_5 + 1) \gg 1$
5, 5 (p side: 5 q side: 5)	$g_j = 58 - j * 13$, *can also be described as* $g = \{58, 45, 32, 19, 6\}$ $f_i = 58 - i * 13$, *can also be described as* $f = \{58, 45, 32, 19, 6\}$ $Middle5,5 = (2 * (q_o + p_o + p_1 + q_1 + q_2 + p_2) + q_3 + p_3 + q_4 + p_4 + 8) \gg 4$ $Q_5 = (q_4 + q_5 + 1) \gg 1, \quad P_5 = (p_4 + p_5 + 1) \gg 1$
5, 3 (p side: 5 q side: 3)	$g_j = 53 - j * 21$, *can also be described as* $g = \{53, 32, 11\}$ $f_i = 58 - i * 13$, *can also be described as* $f = \{58, 45, 32, 19, 6\}$ $Middle5,3 = (q_o + p_o + p_1 + q_1 + q_2 + p_2 + q_3 + p_3 + 4) \gg 3$ $Q_3 = (q_2 + q_3 + 1) \gg 1, \quad P_5 = (p_4 + p_5 + 1) \gg 1$
3, 5 (p side: 3 q side: 5)	$g_j = 58 - j * 13$, *can also be described as* $g = \{58, 45, 32, 19, 6\}$ $f_i = 53 - i * 21$, *can also be described as* $f = \{53, 32, 11\}$ $Middle3,5 = (q_o + p_o + p_1 + q_1 + q_2 + p_2 + q_3 + p_3 + 4) \gg 3$ $Q_5 = (q_4 + q_5 + 1) \gg 1, \quad P_3 = (p_2 + p_3 + 1) \gg 1$

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 >> 4), sp + sq is less than (3*β >> 5), and Abs($p_0 - q_0$) is less than (5 * t_C + 1) >> 1) ? TRUE : FALSE

where d, dpq, sp and sq are magnitudes of gradient calculations to determine amount of details in comparison to a threshold based on β, a QP dependent coding noise threshold, to avoid removing details by the filtering. Similarly, as HEVC it is also checked that the magnitude of the gradient across the boundary is less than a threshold based on tC, a QP dependent deblocking strength threshold.

3.7.2.5 Strong deblocking filter for chroma

The following strong deblocking filter for chroma is defined:

$$p_2' = (3*p_3+2*p_2+p_1+p_0+q_0+4) >> 3 \quad (3\text{-}78)$$

$$p_1' = (2*p_3+p_2+2*p_1+p_0+q_0+q_1+4) >> 3 \quad (3\text{-}79)$$

$$p_0' = (p_3+p_2+p_1+2*p_0+q_0+q_1+q_2+4) >> 3 \quad (3\text{-}80)$$

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	bS		
		Y	U	V
6	At least one of the adjacent blocks is coded with intra or CIIP mode	2	2	2
5	At least one of the adjacent blocks has non-zero transform coefficients	1	1	1
4	One of the adjacent blocks is coded in IBC prediction mode and the other is coded in inter prediction mode	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	0	0
2	Reference pictures the two adjacent blocks refers to are different	1	0	0
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.

3.7.2.6 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 SbTMVP 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.

3.7.2.7 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 VTM, 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.

3.7.3 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 53 shows the LMCS architecture from decoder's perspective. The light-blue shaded blocks in Figure 53 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 53 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 53 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.

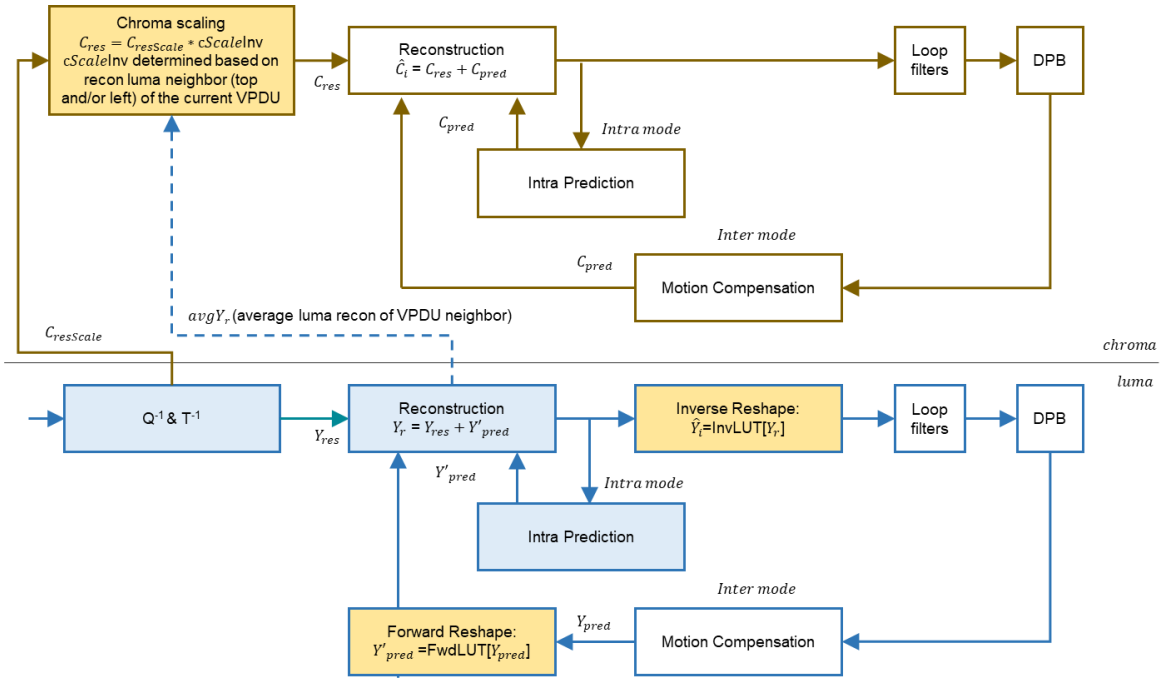


Figure 53 – Luma mapping with chroma scaling architecture

3.7.3.1 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 53 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 53, for an inter-coded block, motion compensated prediction is performed in the mapped domain. In other words, after the motion-compensated prediction block Y_{pred} 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, $Y'_{pred} = FwdMap(Y_{pred})$. For an intra-coded block, the *FwdMap* function is not applied because intra prediction is performed in the mapped domain. After reconstructed block Y_r 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 ($\hat{Y}_i = InvMap(Y_r)$). 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 $FwdMapLUT$ and $InvMapLUT$ can be pre-calculated and pre-stored for use at the tile group level, and forward and inverse mapping can be simply implemented as $FwdMap(Y_{pred}) = FwdMapLUT[Y_{pred}]$ and $InvMap(Y_r) = InvMapLUT[Y_r]$, 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:

$$FwdMap(Y_{pred}) = ((b2 - b1)/(a2 - a1)) * (Y_{pred} - a1) + b1 \quad (3\text{-}81)$$

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, i.e. MappedPivot[i + 1] >> ($BitDepth_Y - 5$) shall not be equal to

MappedPivot[i] >> ($BitDepth_Y - 5$). 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.

3.7.3.2 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 $avgYr$ as the average of the reconstructed neighbouring luma samples (see Figure 53). The value of $C_{ScaleInv}$ is computed in the following steps:

1) Find the index Y_{Idx} of the piecewise linear model to which $avgYr$ belongs based on the *InvMap* function.
2) $C_{ScaleInv}$ = cScaleInv[Y_{Idx}], where cScaleInv[] is a 16-piece LUT pre-computed based on the value of SignalledCW[i] and a offset value sginalled in APS for chroma residual scaling process.

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

Encoder side: $C_{ResScale} = C_{Res} * C_{Scale} = C_{Res} / C_{ScaleInv}$

Decoder side: $C_{Res} = C_{ResScale} / C_{Scale} = C_{ResScale} * C_{ScaleInv}$

3.7.3.3 Encoder-side LMCS parameter estimation

A non-normative reference implementation is provided in the VTM encoder to estimate the LMCS model parameters. Because VTM anchors handle SDR, HDR PQ and HDR HLG differently, the reference algorithm in VTM 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.

3.7.3.3.1 LMCS parameter estimation for SDR

The basic idea of the VTM 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, VTM 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 ($winSize = \lfloor \frac{min(width,height)}{240} \rfloor * 2 + 1$) 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;

   $binCW[i] = round\left(\frac{totalCW}{endIdx-startIdx+1}\right)$, where $binCW[i]$ is the number of codewords allocated to the *i*-th piece, $totalCW$ is the total number of codewords allowed, and $startIdx$ and $endIdx$ are the index values for the first and last valid piece, respectively.

6) 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 $normVar < 1.0$,

$$binCW[i] = \{binCW[i] + delta1,\ 0.8 \leq normVar < 0.9\ binCW[i] + delta2,\ normVar < 0.8 \tag{3-82}$$

   else if $normVar > 1.0$,

$$binCW[i] = \{binCW[i] - delta1,\ 1.1 < normVar \leq 1.2\ binCW[i] - delta2,\ normVar > 1.2 \tag{3-83}$$

   where $normVar = binVar[i]/meanVar$, $delta1 = round(10 * hist)$, $delta2 = round(20 * hist)$, where $binVar[i]$ is the average local spatial variance for the luma values in the *i*-th piece; $meanVar$ is the mean of the average local spatial variances across all valid pieces; and $hist$ 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.

7) 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.
8) 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.
9) 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.

$$w\_lmcs[k] = (SignalledCW[k]/OrgCW)^2 \tag{3-84}$$

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.

3.7.3.3.2 LMCS parameter estimation for HDR

In the JVET HDR CTC, two types of HDR sequences are included: PQ and HLG [5]. 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 [5]. For the HLG sequences, static quantization is used [5]. 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 [5]. 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:

$$dQP(Y) = \max(-3, \min(6, 0.015*Y - 1.5 - 6\ )\ ) \tag{3-85}$$

where Y is the average luma value, $Y \in [0, maxY]$, maxY=1023 for 10-bit video [13]. The weight (W_SSE) used in wPSNR calculation is derived based on dQP values:

$$W\_SSE(Y) = 2\text{^}(dQP(Y)/3) \tag{3-86}$$

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) [13], set slope[Y] = 0 for $Y \in [0, 64)$, or $Y \in (940, 1023]$.
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.

3.8 360-degree video coding tools

3.8.1 Horizontal wrap around motion compensation

The horizontal wrap around motion compensation in the VVC is a 360-specific coding tool designed to improve the visual quality of reconstructed 360-degree video in the equi-rectangular (ERP) projection format [7]. 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 [7], 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 VVC to improve the visual quality of 360-video coded in the ERP projection format.

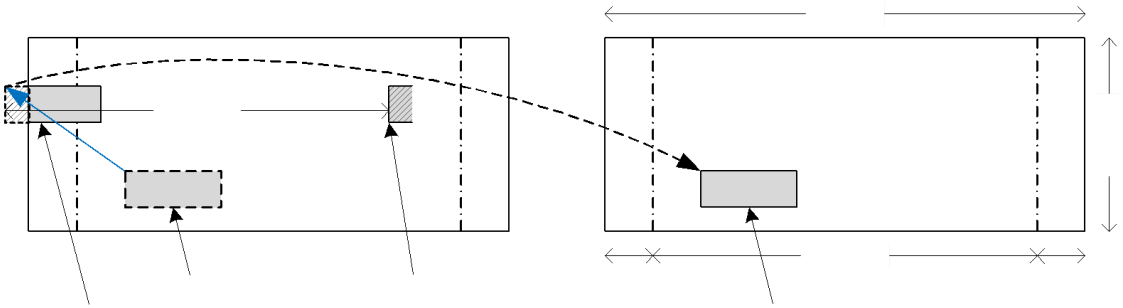


Figure 54 – Horizontal wrap around motion compensation in VVC

The horizontal wrap around motion compensation process is as depicted in Figure 54. 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 54, 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 [7]). In VVC, this is achieved by signaling a high level syntax element to indicate the wrap-around offset, which should be set to the width of padding applied to the ERP picture; 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 [6], 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].

3.8.2 Loop filter disabled across virtual boundaries

For projection formats composed of a plurality of faces, no matter what kind of compact frame packing arrangement is used, discontinuities appear between two or more adjacent faces in the frame packed picture. For example, considering the 3×2 frame packing configuration depicted in Figure 55, the top and bottom halves of the frame packed picture are discontinuous in the 3D geometry. If in-loop filtering operations are performed across this discontinuity, face seam artifacts may become visible in the reconstructed video.

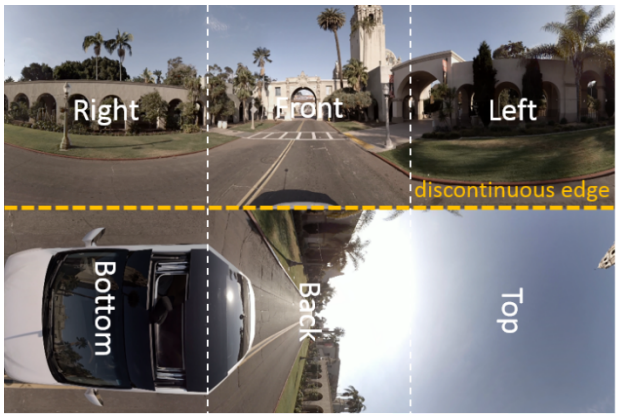


Figure 55 – An example of 3x2 frame packing

To alleviate face seam artifacts and improve the subjective quality of 360-degree video, in-loop filtering operations should be disabled across discontinuities in the frame-packed picture. In VVC, vertical and/or horizontal virtual boundaries, across which the in-loop filtering operations are disabled, are introduced and the positions of those boundaries are signalled in either SPS or Picture Header. Compared to using two tiles, one for each set of continuous faces, and to disable in-loop filtering operations across tiles, this technique is more flexible as it does not require the face size to be a multiple of the CTU size. The signaling is designed to be general purpose, and applicable to other non-360-degree video use cases. The maximum number of vertical virtual boundaries is 3 and the maximum number of horizontal virtual boundaries is also 3. The distance between two virtual boundaries is greater than or equal to the CTU size and the virtual boundary granularity is 8 luma samples.

The virtual boundary could also be used in Gradual Decoding Refresh (GDR) or Progressive Intra Refresh (PIR) which is a technique to limit the large bitrate variations between Intra (I) frames and Inter (P or B) frames while maintaining the same random access period. For this application, the locations of virtual boundaries are signalled in Picture Header since the positions can change frame by frame.

3.9 Screen content coding tools

3.9.1 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.

3.9.1.1 IBC reference region

To reduce memory consumption and decoder complexity, the IBC in VVC allows only the reconstructed portion of the predefined area including the region of current CTU and some region of the left CTU. Figure 56 illustrates the reference region of IBC Mode, where each block represents 64x64 luma sample unit.

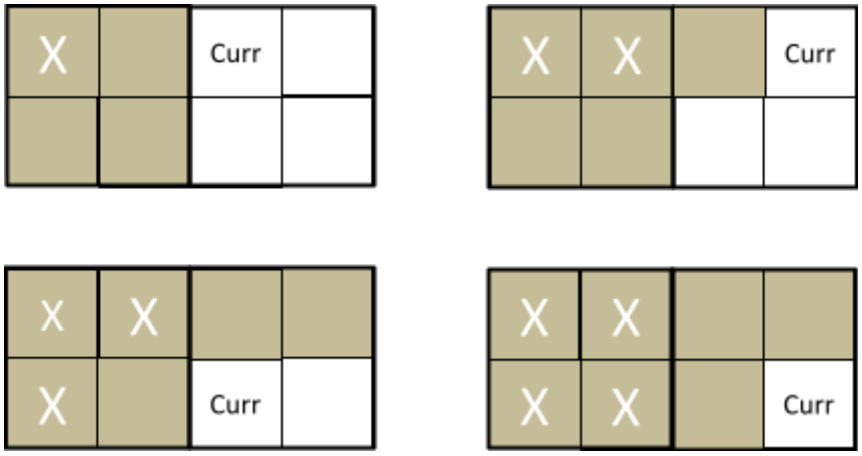


Figure 56 –current CTU processing order and its available reference samples in current and left CTU

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.

3.9.1.2 IBC interaction with other coding tools

The interaction between IBC mode and other inter coding tools in VVC, 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 geometric partitioning mode (GPM) 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 GPM.
– 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 geometric 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 = 128x128/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, W_v = min(ctbSize, 64).

The virtual IBC buffer, ibcBuf is maintained as follows.

– At the beginning of decoding each CTU row, refresh the whole ibcBuf with an invalid value −1.
– 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 + W_v − 1; y = yVPDU%ctbSize, …, yVPDU%ctbSize + W_v − 1.
– 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.

3.9.2 Block differential pulse coded modulation (BDPCM)

VVC supports block differential pulse coded modulation (BDPCM) 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 $r_{i,j},\ 0 \le i \le M-1,\ 0 \le j \le N-1$ be the prediction residual. Let $Q(r_{i,j}),\ 0 \le i \le M-1,\ 0 \le j \le N-1$ denote the quantized version of the residual $r_{i,j}$. BDPCM is applied to the quantized residual values, resulting in a modified M × N array $\widetilde{R}$ with elements $\widetilde{r}_{i,j}$, where $\widetilde{r}_{i,j}$ is predicted from its neighboring quantized residual value. For vertical BDPCM prediction mode, for $0 \le j \le (N-1)$, the following is used to derive $\widetilde{r}_{i,j}$:

$$\widetilde{r}_{i,j} = \{Q(r_{i,j}),\ i = 0\ Q(r_{i,j}) - Q(r_{(i-1),j}),\ 1 \le i \le (M-1) \tag{3-87}$$

For horizontal BDPCM prediction mode, for $0 \le i \le (M-1)$, the following is used to derive $\widetilde{r}_{i,j}$:

$$\widetilde{r}_{i,j} = \{Q(r_{i,j}),\ \ j = 0\ Q(r_{i,j}) - Q(r_{i,(j-1)}),\ 1 \le j \le (N-1) \tag{3-88}$$

At the decoder side, the above process is reversed to compute $Q(r_{i,j}),\ 0 \le i \le M-1,\ 0 \le j \le N-1$, as follows:

$$Q(r_{i,j}) = \sum_{k=0}^{i} \widetilde{r}_{k,j}, \text{ if vertical BDPCM is used} \tag{3-89}$$

$$Q(r_{i,j}) = \sum_{k=0}^{j} \widetilde{r}_{i,k}, \text{ if horizontal BDPCM is used} \tag{3-90}$$

The inverse quantized residuals, $Q^{-1}\left(Q\left(r_{i,j}\right)\right)$, are added to the intra block prediction values to produce the reconstructed sample values.

The predicted quantized residual values $\widetilde{r}_{i,j}$ are sent to the decoder using the same residual coding process as that in transform skip mode residual coding. For lossless coding, if slice_ts_residual_coding_disabled_flag is set to 1, the quantized residual values are sent to the decoder using regular transform residual coding as described in 3.6.2. 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.

3.9.3 Residual coding for transform skip mode

VVC 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. 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:

- 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 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;
- Rice parameter derivation using fixed order = 1 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 57):

- **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 abs_remainder are coded in bypass mode. The remainder of the absolute levels are binarized using a fixed Rice parameter value of 1.

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 1.75*block_width*block_height, or equivalently, 1.75 context coded bins per sample position on average. The bins in the last scan pass (the remainder scan pass) are bypass coded. A variable, RemCcbs, is first set to the maximum number of context-coded bins for the block and is decreased by one each time a context-coded bin is coded. While RemCcbs is larger than or equal to four, syntax elements in the first coding pass, which includes the sig_coeff_flag, coeff_sign_flag, abs_level_gt1_flag and par_level_flag, are coded using context-coded bins. If RemCcbs becomes smaller than 4 while coding the first pass, the remaining coefficients that have yet to be coded in the first pass are coded in the remainder scan pass (pass #3).

After completion of first pass coding, if RemCcbs is larger than or equal to four, syntax elements in the second coding pass, which includes abs_level_gt3_flag, abs_level_gt5_flag, abs_level_gt7_flag, and abs_level_gt9_flag, are coded using context coded bins. If the RemCcbs becomes smaller than 4 while

coding the second pass, the remaining coefficients that have yet to be coded in the second pass are coded in the remainder scan pass (pass #3).

Figure 57 illustrates the transform skip residual coding process. The star marks the position when context coded bins are exhausted, at which point all remaining bins are coded using bypass coding.

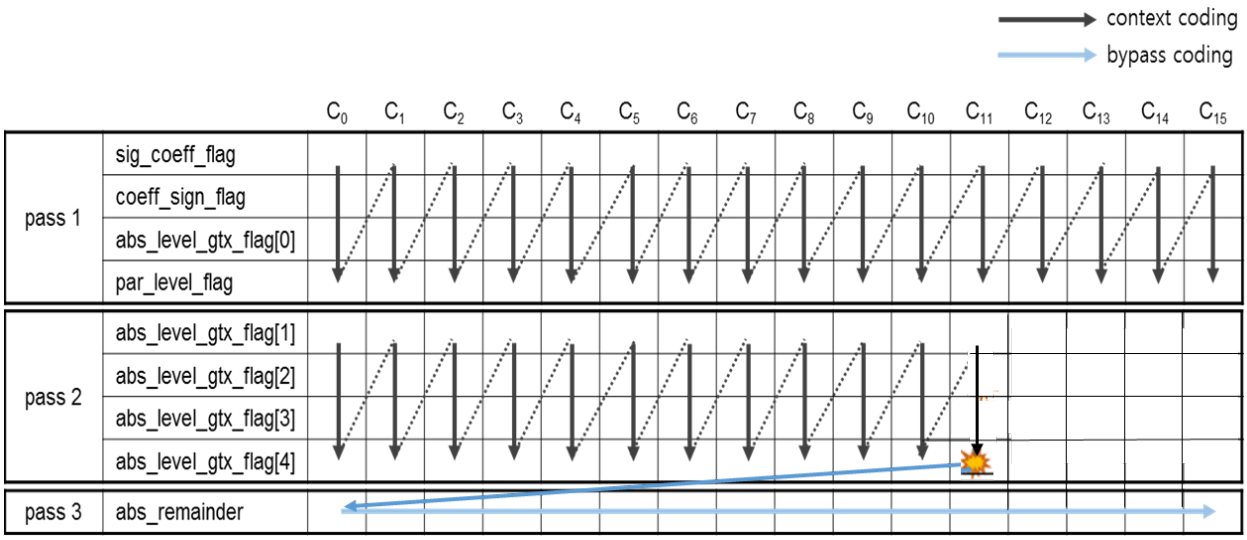


Figure 57 – residual coding passes for transform skip blocks

Further, for a block not coded in the BDPCM mode, a level mapping mechanism is applied to transform skip residual coding until the maximum number of context coded bins has been reached. 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 X_0 denote the absolute coefficient level of the left neighbouring position and let X_1 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;
```

Then, the absCoeffMod value is coded as described above. After all context coded bins have been exhausted, level mapping is disabled for all remaining scan positions in the current block.

3.9.4 Residual coding for transform skip mode in VVC version 2

Version 2 of VVC extends support to bit depths above 10 bits when typically coefficients have a greater magnitude. To efficiently code such coefficients a Rice parameter other than the constant value of 1, as used in VVC version 1, is required.

VVC version 2 includes Extended Transform Skip Residual Coding (ETSRC) which is enabled using sps_ts_residual_coding_rice_present_in_sh_flag. If sh_ts_residual_coding_disabled_flag is set to 0 and sps_ts_residual_coding_rice_present_in_sh_flag is set to 1, the value sh_ts_residual_coding_rice_idx_minus1 is read from the bit slice header. The Rice parameter used for all transform skip residual coding in the slice is then given by:

$$\text{RicePara} = \text{sh\_ts\_residual\_coding\_rice\_idx\_minus1} + 1 \tag{3-91}$$

3.9.5 Palette mode

In VVC, the palette mode is used for screen content coding in all of the chroma formats supported in a 4:4:4 profile (that is, 4:4:4, 4:2:0, 4:2:2 and monochrome). When palette mode is enabled, a flag is transmitted at the CU level if the CU size is smaller than or equal to 64x64, and the number of samples in the CU is greater than 16 to indicate whether palette mode is used. Considering that applying palette mode on small CUs introduces insignificant coding gain and brings extra complexity on the small blocks, palette mode is disabled for CU that are smaller than or equal to 16 samples. 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 set of representative colour values. The set is referred to as the palette. For positions with sample values close to the palette colours, the palette indices are signalled. It is also possible to specify a sample that is outside the palette by signalling an escape symbol. For samples within the CU that are coded using the escape symbol, their component values are signalled directly using (possibly) quantized component values. This is illustrated in Figure 58. The quantized escape symbol is binarized with fifth order Exp-Golomb binarization process (EG5).

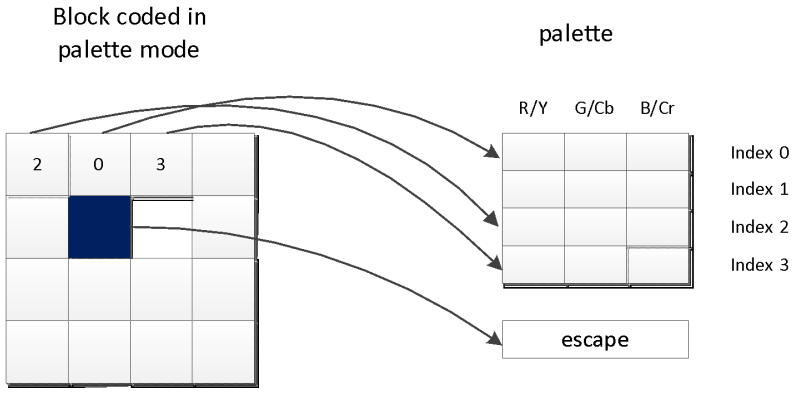


Figure 58: Example of a block coded in palette mode

For coding of the palette, a palette predictor is maintained. The palette predictor is initialized to 0 at the beginning of each slice for non-wavefront case. For WPP case, the palette predictor at the beginning of each CTU row is initialized to the predictor derived from the first CTU in the previous CTU row so that the initialization scheme between palette predictors and CABAC synchronization is unified. 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.

In a similar way as the coefficient group (CG) used in transform coefficient coding, a CU coded with palette mode is divided into multiple line-based coefficient group, each consisting of m samples (i.e., m=16), where index runs, palette index values, and quantized colors for escape mode are encoded/parsed sequentially for each CG. Same as in HEVC, horizontal or vertical traverse scan can be applied to scan the samples, as shown in Figure 59.

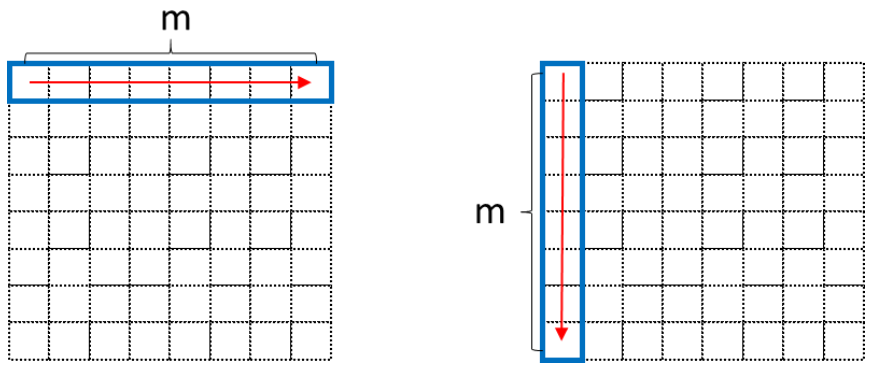


Figure 59: Subblock-based index map scanning for palette, left for horizontal scanning and right for vertical scanning.

The encoding order for palette run coding in each segment is as follows: For each sample position, 1 context coded bin ***run_copy_flag = 0*** is signalled to indicate if the pixel is of the same mode as the previous sample position, i.e., if the previously scanned sample and the current sample are both of run type COPY_ABOVE or if the previously scanned sample and the current sample are both of run type INDEX and the same index value. Otherwise, ***run_copy_flag = 1*** is signaled. If the current sample and the previous sample are of different modes, one context coded bin ***copy_above_palette_indices_flag*** is signaled to indicate the run type, i.e., INDEX or COPY_ABOVE, of the current sample. Here, decoder doesn't have to parse run type if the sample is in the first row (horizontal traverse scan) or in the first column (vertical traverse scan) since the INDEX mode is used by default. With the same way, decoder doesn't have to parse run type if the previously parsed run type is COPY_ABOVE. After palette run coding of samples in one coding pass, the index values (for INDEX mode) and quantized escape colors are grouped and coded in another coding pass using CABAC bypass coding. Such separation of context coded bins and bypass coded bins can improve the throughput within each line CG.

For slices with dual luma/chroma tree, palette is applied on luma (Y component) and chroma (Cb and Cr components) separately, with the luma palette entries containing only Y values and the chroma palette entries containing both Cb and Cr values. 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, unless when a CU is coded using local dual tree, in which case coding of luma and chroma is handled separately. In this case, if the corresponding luma or choma blocks are coded using palette mode, their palette is applied in a way similar to the dual tree case (this is related to non-4:4:4 coding and will be further explained in 3.9.5.1).

For slices coded with dual tree, the maximum palette predictor size is 63, and the maximum palette table size for coding of the current CU is 31. For slices coded with dual tree, the maximum predictor and palette table sizes are halved, i.e., maximum predictor size is 31 and maximum table size is 15, for each of the luma palette and the chroma palette. For deblocking, the palette coded block on the sides of a block boundary is not deblocked.

3.9.5.1 Palette mode for non-4:4:4 content

Palette mode in VVC is supported for all chroma formats in a similar manner as the palette mode in HEVC SCC. For non-4:4:4 content, the following customization is applied:

1. When signaling the escape values for a given sample position, if that sample position has only the luma component but not the chroma component due to chroma subsampling, then only the luma escape value is signaled. This is the same as in HEVC SCC.
2. For a local dual tree block, the palette mode is applied to the block in the same way as the palette mode applied to a single tee block with two exceptions:
    a. The process of palette predictor update is slightly modified as follows. Since the local dual tree block only contains luma (or chroma) component, the predictor update process uses the signalled value of luma (or chroma) component and fills

the "missing" chroma (or luma) component by setting it to a default value of (1 << (component bit depth - 1)).

b. The maximum palette predictor size is kept at 63 (since the slice is coded using single tree) but the maximum palette table size for the luma/chroma block is kept at 15 (since the block is coded using separate palette).

3. For palette mode in monochrome format, the number of colour components in a palette coded block is set to 1 instead of 3.

3.9.5.2 Encoder algorithm for palette mode

At the encoder side, the following steps are used to produce the palette table of the current CU

1. First, to derive the initial entries in the palette table of the current CU, a simplified K-means clustering is applied. The palette table of the current CU is initialized as an empty table. For each sample position in the CU, the SAD between this sample and each palette table entry is calculated and the minimum SAD among all palette table entries is obtained. If the minimum SAD is smaller than a pre-defined error limit, errorLimit, then the current sample is clustered together with the palette table entry with the minimum SAD. Otherwise, a new palette table entry is created. The threshold errorLimit is QP-dependent and is retrieved from a look-up table containing 57 elements covering the entire QP range. After all samples of the current CU have been processed, the initial palette entries are sorted according to the number of samples clustered together with each palette entry, and any entry after the 31st entry is discarded.
2. In the second step, the initial palette table colours are adjusted by considering two options: using the centroid of each cluster from step 1 or using one of the palette colours in the palette predictor. The option with lower rate-distortion cost is selected to be the final colours of the palette table. If a cluster has only a single sample and the corresponding palette entry is not in the palette predictor, the corresponding sample is converted to an escape symbol in the next step.
3. A palette table thus generated contains some new entries from the centroids of the clusters in step 1, and some entries from the palette predictor. So this table is reordered again such that all new entries (i.e. the centroids) are put at the beginning of the table, followed by entries from the palette predictor.

Given the palette table of the current CU, the encoder selects the palette index of each sample position in the CU. For each sample position, the encoder checks the RD cost of all index values corresponding to the palette table entries, as well as the index representing the escape symbol, and selects the index with the smallest RD cost using the following equation:

$$\text{RD cost} = \text{distortion} \times (\text{isChroma? } 0.8 : 1) + \text{lambda} \times \text{bypass coded bits} \tag{3-92}$$

After deciding the index map of the current CU, each entry in the palette table is checked to see if it is used by at least one sample position in the CU. Any unused palette entry will be removed.

After the index map of the current CU is decided, trellis RD optimization is applied to find the best values of run_copy_flag and run type for each sample position by comparing the RD cost of three options: same as the previously scanned position, run type COPY_ABOVE, or run type INDEX. When calculating the SAD values, sample values are scaled down to 8 bits, unless the CU is coded in lossless mode, in which case the actual input bit depth is used to calculate the SAD. Further, in the case of lossless coding, only rate is used in the rate-distortion optimization steps mentioned above (because lossless coding incurs no distortion).

3.9.6 Adaptive colour transform

In HEVC SCC extension, adaptive colour transform (ACT) was applied to reduce the redundancy between three color components in 444 chroma format. The ACT is also adopted into the VVC standard

to enhance the coding efficiency of 444 chroma format coding. Same as in HEVC SCC, the ACT performs in-loop color space conversion in the prediction residual domain by adaptively converting the residuals from the input color space to YCgCo space. Figure 60 illustrates the decoding flowchart with the ACT being applied. Two color spaces are adaptively selected by signaling one ACT flag at CU level. When the flag is equal to one, the residuals of the CU are coded in the YCgCo space; otherwise, the residuals of the CU are coded in the original color space. Additionally, same as the HEVC ACT design, for inter and IBc CUs, the ACT is only enabled when there is at least one non-zero coefficient in the CU. For intra CUs, the ACT is only enabled when chroma components select the same intra prediction mode of luma component, i.e., DM mode.

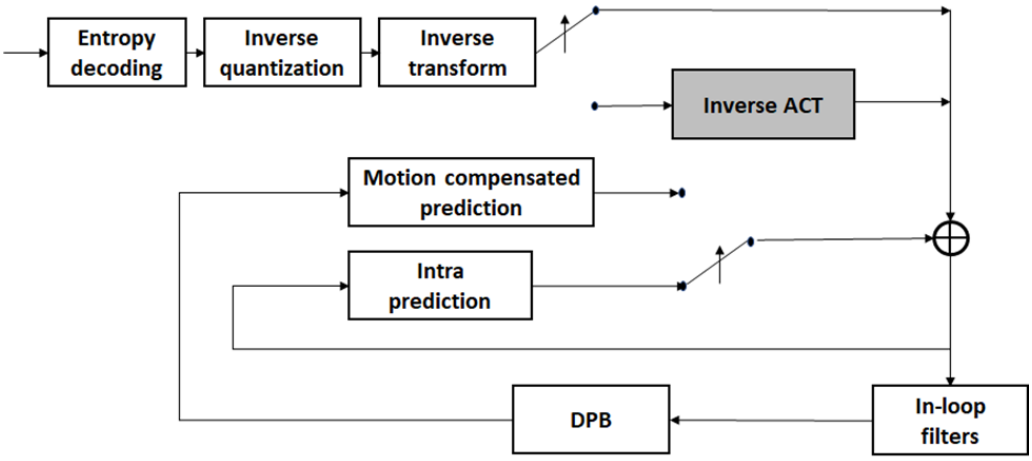


Figure 60: Decoding flowchart with ACT

3.9.6.1 ACT mode

In HEVC SCC extension, the ACT supports both lossless and lossy coding based on lossless flag (i.e.,cu_transquant_bypass_flag). However, there is no flag signalled in the bitstream to indicate whether lossy or lossless coding is applied. Therefore, YCgCo-R transform is applied as ACT to support both lossy and lossless cases. The YCgCo-R reversible colour transform is shown as below.

Forward Conversion: GBR to YCgCo	Backward Conversion: YCgCo to GBR
Co = R-B;	t = Y – (Cg>>1)
t = B + (Co>>1);	G = Cg + t
Cg = G – t;	B = t – (Co>>1)
Y = t + (Cg>>1);	R = Co + B

(3-93)

Since the YCgCo-R transform are not normalized. To compensate the dynamic range change of residuals signals before and after color transform, the QP adjustments of (−5, 1, 3) are applied to the transform residuals of Y, Cg and Co components, respectively. The adjusted quantization parameter only affects the quantization and inverse quantization of the residuals in the CU. For other coding processes (such as deblocking), original QP is still applied.

Additionally, because the forward and inverse color transforms need to access the residuals of all three components, the ACT mode is always disabled for separate-tree partition and ISP mode where the prediction block size of different color component is different. Transform skip (TS) and block differential pulse coded modulation (BDPCM), which are extended to code chroma residuals, are also enabled when the ACT is applied,

3.9.6.2 ACT fast encoding algorithms

To avoid brutal R-D search in both the original and converted color spaces, the following fast encoding algorithms are applied in the VTM reference software to reduce the encoder complexity when the ACT is enabled.

- The order of RD checking of enabling/disabling ACT is dependent on the original color space of input video. For RGB videos, the RD cost of ACT mode is checked first; for YCbCr videos, the RD cost of non-ACT mode is checked first. The RD cost of the second color space is checked only if there is at least one non-zero coefficient in the first color space.
- The same ACT enabling/disabling decision is reused when one CU is obtained through different partition path. Specifically, the selected color space for coding the residuals of one CU will be stored when the CU is coded at the first time. Then, when the same CU is obtained by another partition path, instead of checking the RD costs of the two spaces, the stored color space decision will be directly reused.
- The RD cost of a parent CU is used to decide whether to check the RD cost of the second color space for the current CU. For instance, if the RD cost of the first color space is smaller than that of the second color space for the parent CU, then for the current CU, the second color space is not checked.
- To reduce the number of tested coding modes, the selected coding mode is shared between two color spaces. Specifically, for intra mode, the preselected intra mode candidates based on SATD-based intra mode selection are shared between two color spaces. For inter and IBC modes, block vector search or motion estimation is performed only once. The block vectors and motion vectors are shared by two color spaces.

4 Description of SEI message implementations in VTM

This section describes implementations of SEI messages in VTM.

4.1 Implementation related to film grain characteristics SEI message

The film grain characteristics (FGC) SEI message is specified in H.SEI | ISO/IEC 23002-7. The FGC SEI message may be used to signal the characteristics of film grain that was present in the original source video material and was removed by pre-processing filtering techniques. The FGC SEI message may also be used to simulate film grain on the decoded images when film grain was not present in the original source video material to mask compression or bit depth related artifacts or to create an artistic effect.

VTM reference software is supplied with the following: 1) methods of analyzing source images to determine values of syntax elements of the FGC SEI message; 2) methods of using FGC SEI message syntax values to synthesize grain and blend grain with decoded images; and 3) methods of adding, deleting, and rewriting FGC SEI messages in coded bitstreams.

In VVC, two model types are supported by the FGC SEI message: a frequency-filtering model and an auto-regressive model. In VTM, only the frequency-filtering model is implemented. The analysis and synthesis processes described are based on the SMPTE RDD 5 [14] specification except for two modifications:

1) The 64x64 DCT-2 transform specified in VVC is used instead of the 64x64 DCT-2 transform specified in SMPTE RDD 5
2) The grain block size is increased for for resolutions greater than 1920x1080 (SMPTE RDD 5 restricts grain block size to 8x8, which is not implementation friendly for resolutions above 1920x1080).

All other contrains specified by SMTE RDD 5 are applicable:

- ChromaFormatIdc is equal to 1
- fg_model_id is equal to 0.

– fg_separate_colour_description_present_flag equal is to 0.
– fg_blending_mode_id is equal to 0.
– fg_log2_scale_factor is in the range 2 to 7, inclusive.
– fg_intensity_interval_upper_bound[c][i] is less than fg_intensity_interval_lower_bound[c][i + 1].
– fg_num_model_values_minus1[c] is in the range 0..2, inclusive.
– fg_comp_model_value[c][i][0] is in the range 0.. 255 , inclusive.
– fg_comp_model_value[c][i][1] is in the range 2..14, inclusive.
– fg_comp_model_value[c][i][2] is in the range 2..14, inclusive.

The film grain analysis and synthesis are performed in VTM at an internal bit depth of 8 bits with input and output conversion as needed.

4.1.1 Film grain characteristics analysis

4.1.1.1 Overview

Film grain characteristics analysis is performed at the encoder side. The analysis process determines film grain model parameters to be sent in the FGC SEI message to enable content-aware decoder-side film grain synthesis. The analysis process is not mandatory, and manually tuned parameters can be provided to the encoder based on the input configuration file. If the analysis process determines that the no grain is present in the source, the values of FGC SEI syntax elements are set to 0 to indicate that synthesis should not be performed at the decoder side.

A general framework of film grain analysis and synthesis is illustrated in Figure 61.

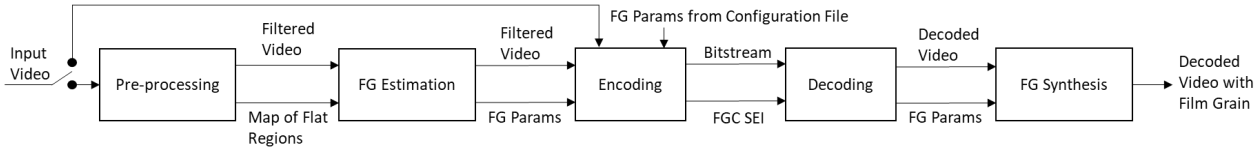


Figure 61: A simplified block diagram of the film grain usage in a video coding framework.

Film grain analysis consists of the following steps:

1) a pre-processing step to produce a film grain image and other information about the characteristics of the input video;
2) a film grain estimation step to determine cut-off frequencies and scaling function.

A more focused illustration of film grain analysis is given in Figure 62.

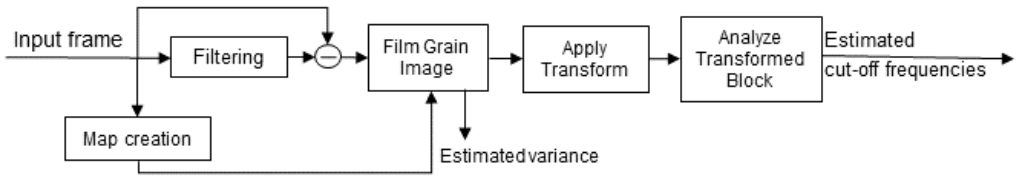


Figure 62: Film grain analysis workflow (pre-processing and film grain estimation).

In the current implementation, film grain analysis is not performed when Qp is less than 17. Also, if IntraPeriod < 1 analysis is performed every 2s, otherwise analysis is performed once per IntraPeriod.

4.1.1.2 Pre-processing

The pre-processing step consists of extracting film grain from the input frame, along with a map indicating where this extraction is reliable.

First, the image is denoised and compared to the original. The film grain image that represents the film grain estimate is calculated as a difference between the original input frame and the denoised frame. The denoised image is either provided externally, using the *SEIFGCExternalDenoised* configuration parameter, or computed internally by low-pass filtering the input frame in which case MCTF is re-used for this purpose. An additional instance of MCTF is created in the current implementation, which gives the flexibility to test MCTF with different parameters without interfering with the initial MCTF used during the encoding process using the *SEIFGCTemporalFilter[…]* configuration parameters. MCTF filtering for film grain analysis uses an internal buffer to store the filtered frame for the film grain analysis module, which means that complete encoding processes is unchanged, i.e., bitrate and decoded frames are the same as without film grain analysis.

In addition, since edges and complex texture can interfere with the analysis process, the pre-processing step creates a map that indicates flat regions of an input frame, unless provided externally using the *SEIFGCExternalMask* parameter (for each colour channel, that external map should contain either zero to indicate flat areas, or max values to indicate textured areas). To create a map of flat regions, analysis is performed on three scales: an original and two subsampled resolutions. One of these is subsampled by a factor of 2 and the other by a factor of 4. At each scale, Canny edge detection is used (except the classical Canny detector blurring step is skipped) followed by morphological dilations (4 for full scale, 3 for half-resolution, and 2 for quarter-resolution). The obtained maps at subsampled scales are upsampled to the original resolution and combined. Additional morphological operations are subsequently applied where two dilations followed by one erosion are applied to get the final map of flat regions of an input frame. The film grain image and map are then provided to the estimation module.

4.1.1.3 Determination of cutoff frequencies

Cut-off frequencies are estimated as follows. The film grain image is scanned block by block at 64x64 granularity (non-overlapping blocks). Only 64x64 blocks within the flat part of an image are processed further. For each block coming from a flat region, a forward 64x64 DCT-2 as defined in VVC is applied. Then, each coefficient resulting from the transform is squared. Then an average 64x64 squared transformed block B_{avg} is computed over all available squared transformed blocks B_i (pixel-wise):

$$B_{avg}(x,y) = \frac{1}{K}\sum_{i=0}^{K-1} B_i(x,y) \tag{4-1}$$

where K is the total number of blocks coming to the calculations. Thereafter, average vectors by columns B_c and by rows B_r are calculated (the DC component for the first row and the first column is discarded from the computation):

$$B_c(y) = \frac{1}{63+(y>0)?1:0}\sum_{i=(y>0)?0:1}^{63} B_{avg}(i,y) \tag{4-2}$$

$$B_r(x) = \frac{1}{63+(x>0)?1:0}\sum_{i=(x>0)?0:1}^{63} B_{avg}(x,i) \tag{4-3}$$

Average vectors B_c and B_r are regularized to suppress peaks. The average vectors are represented as a curve and its intersection points with total average value (avg serves as a threshold) of a B_{avg} block are computed ($avg = \frac{1}{64*64}\sum_{i=0}^{64-1}\sum_{j=0}^{64-1} B_{avg}(i,j)$), as illustrated in Figure 63.

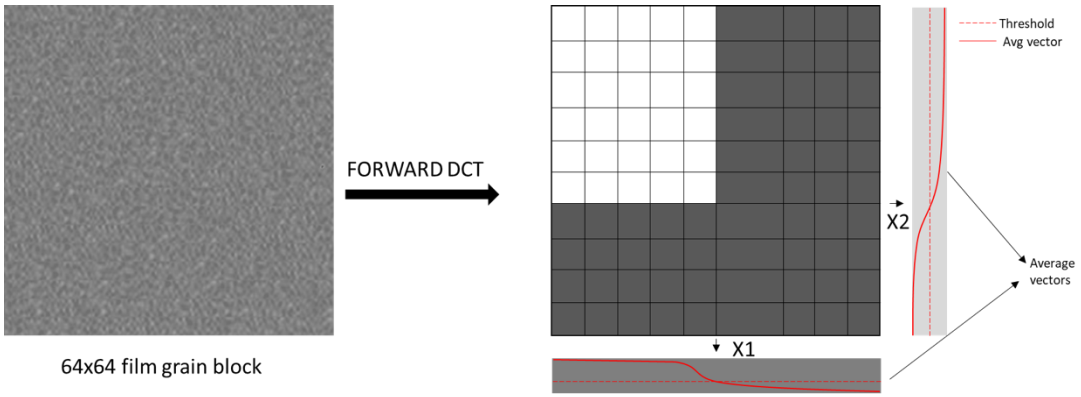


Figure 63: Cut-off frequency estimation (intersection points x1 and x2 represent estimated frequencies)

Based on the analysis of the intersection point(s), cut-off frequencies are obtained. The intersection points are chosen to represent cut-off frequencies of the frequency-filtered film grain model. In such case, model values are set to (according to the SMPTE RDD 5 specification):

- comp_model_value[c][i][1] = Clip3(2,14,(x1-1)>>2) and
- comp_model_value[c][i][2] = Clip3(2,14,(x2-1)>>2),

where *i* is index of the interval, and *c* is color component index. If intersection points are not found, film grain is not present in the input frame.

In the current implementation, only one pair of cut-off frequencies is estimated over the full intensity interval range.

4.1.1.4 Determination of intensity intervals and grain scaling factors

Intensity intervals (fg_intensity_interval_lower_bound[c][i], fg_intensity_interval_upper_bound[c][i]) and corresponding grain scaling values (fg_comp_model_value[c][i][0]) are determined by analyzing the film grain image, filtered image, and flat-region map produced during pre-processing. This leads to estimate a stepwise constant scaling function as illustrated in Figure 64.

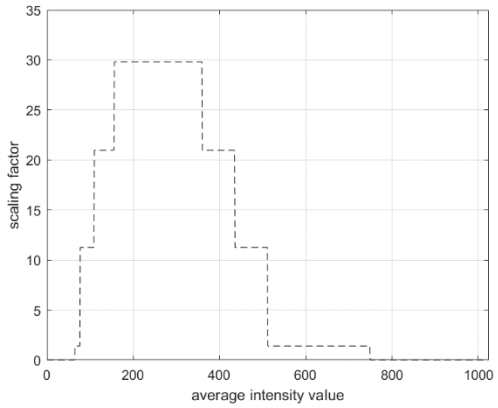


Figure 64: An example of film grain scaling function defined with fg_intensity_interval_upper_bound[c][i], fg_intensity_interval_lower_bound[c][i] and fg_comp_model_value[c][i][0].

The analysis is performed block-by-block with block size, BlockSize, determined based on a frame resolution as follows:

PicSizeinLumaSamples = PicHeightInLumaSamples * PicWidthInLumaSamples

```
if    PicSizeinLumaSamples <= ( 1920 * 1080 )
      BlockSize = 8
else if PicSizeinLumaSamples <= ( 3840 * 2160 )
      BlockSize = 16
else
      BlockSize = 32
```

The average intensity value of the denoised image and the variance *v* of the film grain image is calculated for each block within a flat region of the image. Example results are plotted in Figure 65.

Then, the variance is converted to grain strength with the following formula:

$$s = 3.0 * \sqrt{v}$$

Data points (pairs of intensity & grain strength) with grain strength higher than 16 << (bitDepth – 8) are discarded to limit data points to meaningful values. If strength is higher than the maximum defined value, most likely variance is biased by edges and other high frequency components removed during the filtering (edges or fine details).

The data points are then used to determine intensity intervals and scaling factors in two steps:

- derive a scaling function by two-pass curve fitting
- quantize the fitted curve to produce a stepwise function

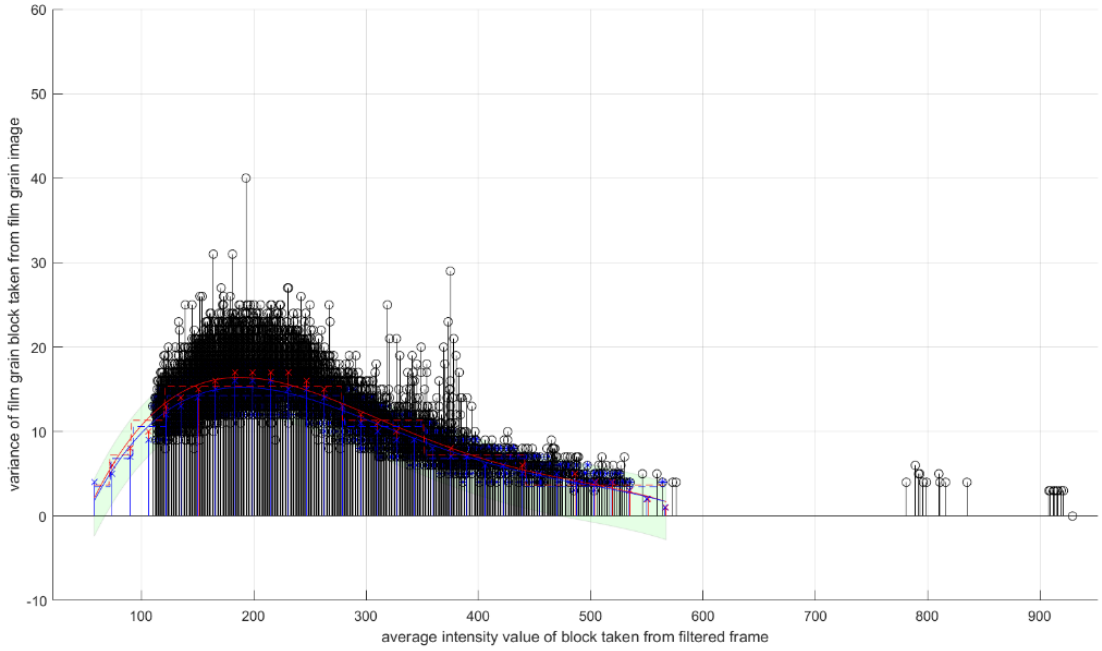


Figure 65: Data-points, 2-pass curve fitting and quantization (black circles: input data points, red: non-filtered fitting curve, blue: filtered fitting curve).

Two-pass curve fitting is performed as follows.

First, the complete intensity dynamic range (horizontal axis) is divided (uniformly quantized) into non-overlapping intervals of size 16. If the number of points falling into an interval is greater than $N_{min} = 8$, the average strength value is calculated for the interval (red points in Figure 65). If there is a single point without any neighbouring points, it is considered as an estimation error and it is removed.

Single points are filtered out (single non-zero interval). Also, points outside the 40 to 950 interval (for 10-bit) are removed. At most 4 additional points are added at each end to extend and smooth curve fitting, avoiding abrupt transitions between areas with and without grain.

Next, the average points per interval are used to fit a 4th order polynomial curve (red curve in Figure 65).

The second pass starts by identifying outliers by calculating the standard deviation σ of the difference of the points and fitted curve. Any point greater than +0.6 * σ and lower than -1.2 * σ from the curve are excluded (see shaded green region in Figure 65). Then, the remaining points are fitted with a new parameterized 4th-order polynomial curve (blue curve in Figure 65).

The final step approximates the scaling function represented by the 4th order polynomial curve by a simplified stepwise scaling function. The stepwise function is derived by using Lloyd max non-uniform quantization with 4 quantization levels (see the blue dotted line in Figure 65). The quantizer is adapted (trained) for each new set of points on-the-fly.

By analyzing the quantized curve the number of intensity intervals, the bounds of intervals (fg_intensity_interval_lower_bound[c][i] and fg_intensity_interval_upper_bound[c][i]) and the scaling factor (fg_comp_model_value[c][i][0]) associated with each estimated interval are obtained. Note that in the given example, even if there are 4 quantization levels, quantization of the curve leads to more intensity intervals. Those intervals and scaling factors are embedded in the SEI message and transmitted to the decoder.

Analysis is performed in the same way for all colour components.

4.1.2 Film grain synthesis

4.1.2.1 Generation of grain-pattern database

A 13x13x64x64 grain-pattern database (GPD) of grain patterns corresponding to each unique set of component model values signalled in an FGC SEI message conforming to SMPTE RDD 5 is determined as follows. For each pair of horizontal and vertical high cutoff frequency (13 each):

1) Generate a 64x64 2-dimensional array of random-value elements having a normalized Gaussian distribution. The 2-dimensional array represents discrete cosine transform (DCT2) coefficients. The row and column indexes of the array represent horizontal and vertical frequencies, respectively.
2) Set the value of all elements of the random-value DCT2 array to 0 when the corresponding row and column indexes are not less than the corresponding high and low horizontal and vertical cutoff.
3) Calculate the inverse discrete cosine transform (IDCT2) of the array produced in step2.
4) Deblock across vertical boundaries of adjacent grain patterns.

Note that the database of grain patterns can be pre-computed and stored.

4.1.2.2 Determination of intensity interval for each grain block

Assign each sample of the decoded picture to a grain block. The array of grain blocks is non-overlapping and cover the entire decoded picture. The grain block size, BlockSize x BlockSize, is determined as follows:

```
PicSizeInLumaSamples = PicHeightInLumaSamples * PicWidthInLumaSamples

if    PicSizeInLumaSamples <= (1920 * 1080)
        BlockSize = 8
else if PicSizeInLumaSamples <= (3840 * 2160)
        BlockSize = 16
else
        BlockSize = 32
```

The applicable intensity interval for each grain block for each applicable colour component is determined as follows:

1) Calculate the average value of the sample values in the block
2) Associate the grain block with the index of the interval value signalled in the FGC SEI message when the block average is within the upper and lower bounds of the intensity interval.

4.1.2.3 Synthesis of grain image and blending with decoded image

The grain image for each applicable colour component is constructed by selecting BlockSize x BlockSize arrays of values from grain patterns in the GPD. For each grain block associated with the *i*th intensity interval, the grain pattern corresponding to the pair of horizontal and vertical cutoff frequencies signalled in the FGC SEI message for the *i*th intensity interval is selected. Each BlockSize x BlockSize array of values is extracted from the selected grain pattern using pseudo-random horizontal and vertical offsets. Deblocking is applied across horizontal boundaries of adjacent grain blocks.

Blending is achieved by summing the sample values of the synthesized grain image with the collocated sample values of the decoded picture for each applicable colour component followed by clipping to prevent out of range sample values.

4.1.3 Film grain characteristics SEI message rewriter

The FGC SEI message rewriter included in VTM allows FGC SEI syntax values to be modified in existing bitstreams without re-encoding. It also supports inserting FGC SEI messages in a VVC bitstream originally encoded without FGC SEI message. The bitstream with the inserted FGC SEI has identical MD5sum value to the bitstream having the FGC SEI enabled during encoding. The tool supports three operational modes, controlled by SEIFilmGrainOption: 0-no change; 1-FGC SEI removal; 2-FGC SEI insertion; 3-FGC SEI rewriter.

1) FGC SEI removal (applied on bitstreams with FGC SEI)
2) FGC SEI insertion (applied on bitstreams without FGC SEI)
3) FGC SEI modification (applied on bitstreams with FGC SEI)

The two typical use cases and examples are given below:

1) A bitstream (test_fg1.bit) is encoded with a pre-configured FGC SEI (fg1.cfg), but the film grain parameters are not producing satisfactory film grain effects. One can use the tool to modify the values (fg2.cfg) of the existing FGC SEI. Example command line:
   ./SEIFilmGrainAppStatic -c fg2.cfg -b test_fg1.bit -o test_fg2.bit --SEIFilmGrainOption=3

2) A bitstream (test.bit) is encoded without FGC SEI so it cannot take advantage of benefit of the film grain synthesis. One can use the tool to easily insert the FGC SEI (fg.cfg) into the bitstream. Example command line:
   ./SEIFilmGrainAppStatic -c fg.cfg -b test.bit -o test_fg.bit --SEIFilmGrainOption=2

The tool manipulates the VVC bitstream at NALU level, so the processing time is extremely fast (e.g., to rewrite FGC SEI in a 4K/8K bitstream, the entire processing time is less than 0.1 sec). The basic workflow is illustrated in Figure 67.

''

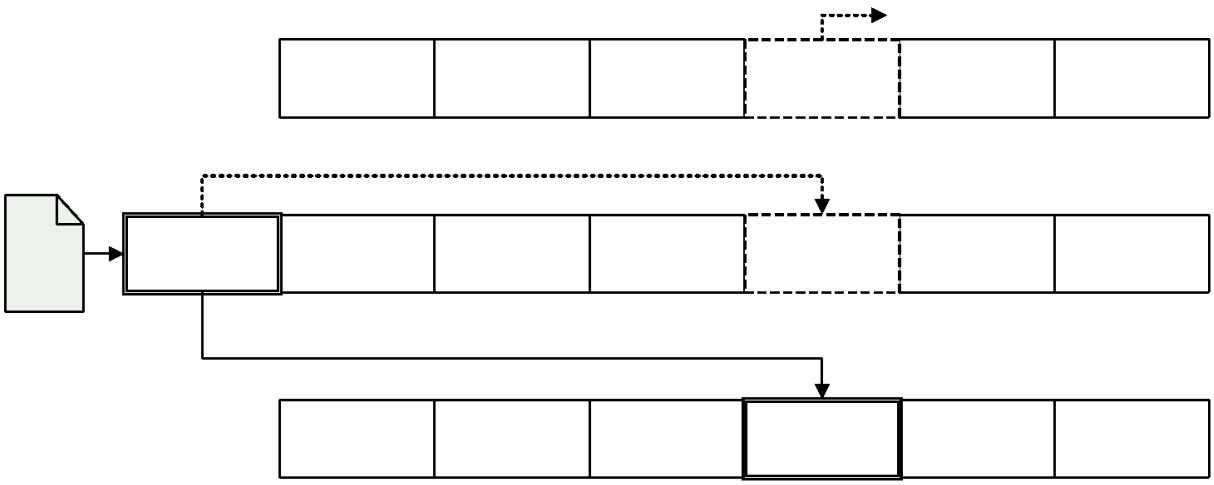


Figure 66: Basic workflow of FGC SEI rewriter

4.2 Implementation related to green metadata SEI messages

Green Metadata SEI messages for signaling complexity metrics for decoder-power reduction, and metrics for quality recovery after low-power encoding, are specified in ISO/IEC 23001-11 3rd edition [16]. The complexity metrics can be used on the receiver side to estimate the complexity of the decoding process, which helps to reduce receiver-side power consumption with tools such as dynamic voltage and frequency scaling. The quality metrics can be used to increase the video quality with suitable quality enhancing post-filtering techniques. Corresponding implementations for the encoder and the decoder are described below.

4.2.1 Encoding of green metadata SEI messages

Green metadata defines two different kinds of SEI messages: metrics for quality recovery after low-power encoding (see Section 4.2.1.1); and complexity metrics for decoder-power reduction (Section 4.2.1.2). When encoding, these can be signalled and handled independently. Both are signalled as prefix-SEI messages. The functionality and usage is explained below.

4.2.1.1 Metrics for quality recovery after low-power encoding

The VTM encoder supports the generation and signalling of metrics for quality recovery after low-power encoding. The supported quality metrics include picture-wise PSNR, WPSNR, and SSIM, which are calculated during encoder runtime. For signaling, the values of these quality metrics are scaled by a factor of 100 and transmitted as 16-bit integer values.

To configure the encoder, the following flags need to be set:

SEIGreenMetadataType specifies the type of green metadata. A value of 1 indicates metrics for quality recovery after low-power encoding.

SEIGreenMetadataPeriodType specifies the period type. Currently, only period type 0 is supported which indicates that metrics are signaled for each picture.

SEIXSDMetricNumber indicates the number of transmitted metrics in the current period. Must correspond to the sum of the flags SEIXSDMetricTypePSNR, SEIXSDMetricTypeSSIM, and SEIXSDMetricTypeWPSNR.

SEIXSDMetricTypePSNR indicates whether the PSNR is signaled (1) or not signaled (0).

SEIXSDMetricTypeSSIM indicates whether the SSIM is signaled (1) or not signaled (0).

SEIXSDMetricTypeWPSNR indicates whether the WPSNR is signaled (1) or not signaled (0).

4.2.1.2 Complexity metrics for decoder power reduction

The VTM encoder supports the generation and signalling of complexity metrics for decoder-power reduction. Both the reduced set and the extended set of complexity metrics as defined in ISO/IEC 23001-11 3rd edition [15], are supported.

To configure the encoder, the following flags need to be set:

SEIGreenMetadataType specifies the type of green metadata. A value of 0 indicates complexity metrics for decoder-power reduction.

SEIGreenMetadataPeriodType specifies the period type. Currently, only period type 0 is supported, which indicates that metrics are signaled for each picture.

SEIGreenMetadataExtendedRepresentation specifies whether the extended set (1) or the reduced set (0) of complexity metrics is signaled.

The values for the complexity metrics are derived by an encoder plugin that counts the occurrence of bit stream features, called the green metadata feature analyzer (GMFA). The GMFA analyzes the encoding information of each CTU after the rate-distortion optimization for the occurrence of features such as inter/intra prediction, transforms, block sizes, motion vectors, motion vector coding, and residual coefficients. Furthermore, at a picture level, statistics on the usage of in-loop filters (ALF, SAO, and deblocking filter) are collected.

4.2.2 Parsing of green metadata SEI messages

The VTM reference software decoder includes functionality for parsing Green Metadata SEI messages. It reads all syntax elements defined in ISO/IEC 23001-11 3rd edition and sends them to standard output. Both forms of metadata, i.e., metrics for quality recovery after low-power encoding and complexity metrics for decoder-power reduction, are supported. The implementation can be used to verify the conformance of bitstreams containing green metadata SEI messages.

4.3 Implementation related to shutter interval information SEI message

The shutter interval information (SII) SEI message is specified in H.274 | ISO/IEC 23002-7. The SII SEI may be used to signal the amount of time an image sensor or photographic film is exposed to light.

The shutter interval has a direct impact on the amount of motion blur in an image. The longer the shutter interval, the more motion blur. The shutter interval also has a direct impact on the amount of noise in a captured image. The shorter the shutter interval, the lower the signal to noise ratio will be. For computer-generated content, a shutter interval value equal to 0 can be used to indicate the complete absence of motion blur and photon noise.

VTM reference software is supplied with methods to illustrate the use of the SII SEI message with backwards-compatible high frame rate (HFR) bit streams as specified in ETSI TS 101 145 [16] and ATSC A/341 [17] and illustrated in Figure 68.

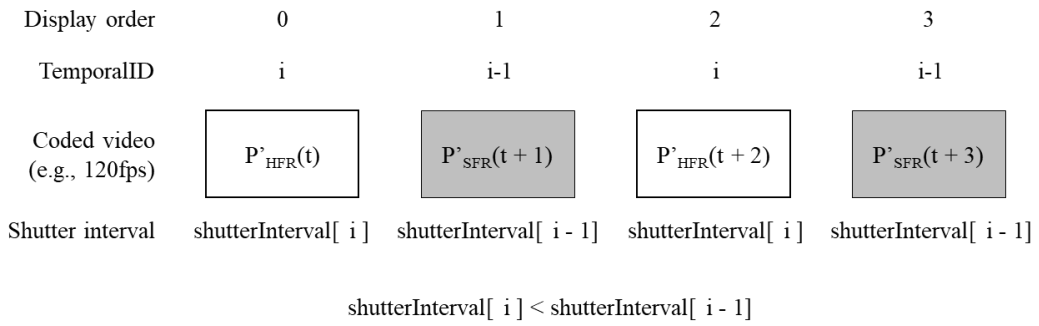


Figure 67. A backwards-compatible HFR bit stream containing SFR, P'$_{SFR}$(t), and HFR, P'$_{HFR}$(t), pictures with different shutter intervals, shutterInterval[i]

When a backwards-compatible HFR bit stream is received by a non-HFR-capable decoder, only the backwards-compatible temporal sublayer with the SFR pictures is decoded, as illustrated in Figure 69A. When the bit stream is received by an HFR-capable decoder, all temporal sublayers are decoded and the following process is used to reconstruct HFR pictures:

$$\hat{P}_{HFR}(t) = \frac{P_{SFR}(t) - w_1 P_{HFR}(t-1)}{w_2} \quad (4\text{-}4)$$

where $P_{HFR}(t-1)$ and $P_{SFR}(t)$ are adjacent decoded pictures in display order, $\hat{P}_{HFR}(t)$ is the current reconstructed HFR picture, and w_1 and w_2 are weighting factors. The reconstruction process is invoked for every other decoded picture in display order, as illustrated in Figure 69B.

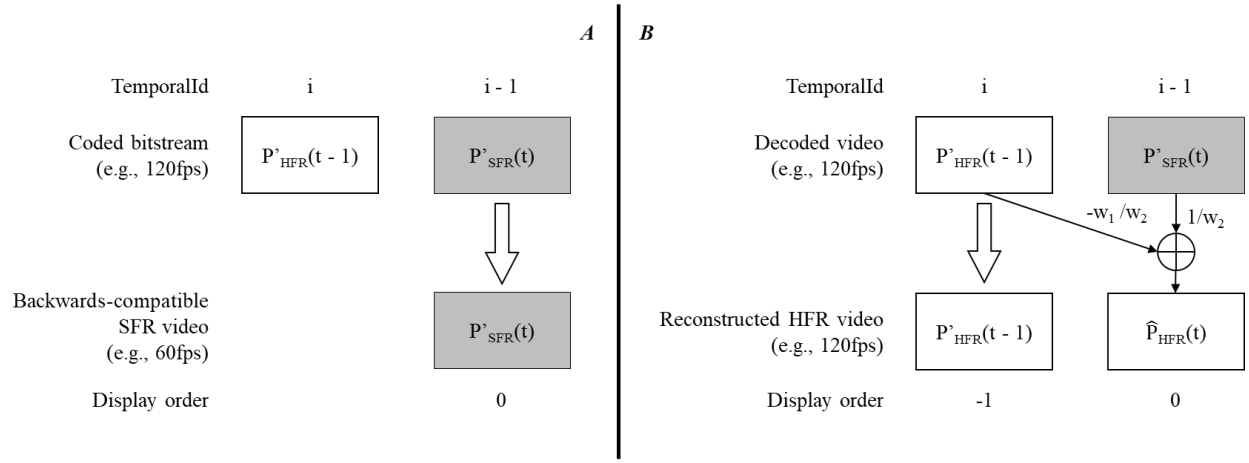


Figure 68. Output process for backwards-compatible HFR bit stream: A, only the SFR video is decoded by a non-HFR-capable receiver; B, post-processing to reconstruct HFR video from the output of an HFR-capable decoder

Weighting parameters, w_1 and w_2, should be selected such that $w_1 + w_2 = 1$ so that average luma and chroma values are preserved. In addition, w_1 and w_2 should be greater than or equal to 0.

When using frame blending to create a backwards-compatible HFR bitstream, ATSC A/341 indicates that setting the values of the weighting parameters to $w_1 = w_2 = 0.5$ produces good results when the shutter interval of the HFR video is equal to the reciprocal of HFR frame rate. In this case, the image sensor would be continuously exposed and the shutter interval of the SFR video would consequently be twice the reciprocal of the HFR frame rate. When the HFR shutter interval is significantly less than the reciprocal

of the frame rate, ATSC A/341 indicates that it can be beneficial to set the values of the weighting parameters such that $w_1 < w_2$.

In the VTM reference software, the weighting parameters, w_1 and w_2, are determined from a blending ratio, M:N, as follows:

$$w_1 = \frac{N}{M+N} \textbf{ and } w_2 = \frac{M}{M+N} \quad (4\text{-}5)$$

The HFR reconstruction process is invoked in the VTM reference software when all the following apply:

- fixed_shutter_interval_within_clvs_flag is equal to 0
- sii_max_sub_layers_minus1 is greater than 0
- subLayerShutterInterval[sii_max_sub_layers_minus1] is less than subLayerShutterInterval [sii_max_sub_layers_minus1 - 1]
- subLayerShutterInterval[i] is equal to subLayerShutterInterval [i – 1] for i = 0 … sii_max_sub_layers_minus1 – 1

The values of *M* and *N* are derived as follows:

$$M = \frac{shutterInterval[i-1]}{shutterInterval[i]} \textbf{ and } N = 1 \quad (4\text{-}6)$$

4.4 Implementation related to neural-network post-filter characteristics and activation SEI messages

The neural-network post-filter characteristics (NNPFC) SEI message specifies the characteristics of a neural network that can be used as post-processing filter. Additionally, the neural-network post-filter activation (NNPFA) SEI message activates the use of a specified post-filter for a specific picture.

An NNPFC SEI message includes the following pieces of information:
- A filter identifier
- A mode indicator (nnpfc_mode_idc), indicating if the neural network used for filtering is provided by external means or included or updated by an ISO/IEC 15938-17 bitstream included in this NNPFC SEI message
- A purpose for filtering (e.g., general visual quality improvement, chroma upsampling, resolution upsampling, picture rate upsampling, bit depth upsampling, colourization, or various combination of these purposes) and purpose-specific syntax elements
- Syntax elements indicating how the input and output tensors are formatted
- Complexity information

An NNPFA SEI message includes the following pieces of information:
- A filter identifier
- A cancel flag (nnpfa_cancel_flag), indicating if the persistence of the target NNPF established by any previous NNPFA SEI message with the same nnpfa_target_id as the current SEI message is cancelled
- A persistence flag (nnpfa_persistence_flag), indicating the persistence of the target NNPFA

The VTM reference software contains implementation of the signaling and parsing of NNPFC and NNPFA SEI messages. Sample configuration files are provided for this purpose. However, VTM reference software does not implement the post filtering process, and it is left up to the decoder implementer to make use of the parsed messages in order to apply the post-filter.

The NNPFC SEI message can be used to carry information about a post-filter for the purpose of general visual quality improvement. For example, five different filters are considered, where four filters, referred to as base filters or pretrained filters, are assumed to be available both at encoder side and at decoder side, and where the fifth filter is referred to as overfitted filter. The four base filters may have been trained on a sufficiently large training dataset. Information about the characteristics of the five filters is signaled via

five different NNPFC SEI messages. In the four NNPFC SEI messages associated to the four base filters, the value of nnpfc_mode_idc is set to 0.

The encoder can select one of the four base filters to be overfitted on the test sequence, for example, based on the PSNR gain provided by the base filters on the first Random Access segment of the test sequence. The overfitted filter is signaled via a NNPFC SEI message, where the filter identifier value is the same as the filter identifier value in one of the four NNPFC SEI messages associated to the four base filters, and where the value of nnpfc_mode_idc is set to 1.

In order to perform the overfitting process, overfitting data needs to be collected. The overfitting data consists of pairs of patches, where each pair consists of one uncompressed patch (used as ground-truth information) and the corresponding patch reconstructed by VTM. The overfitting process is a training process, where some of the parameters of the neural network post-filter are updated, such as a set of multipliers that multiply the output of a convolutional layer. The overfitting process is performed until a stopping criterion is satisfied, such as a predetermined number of iterations is reached, or a predetermined quality is achieved. A weight-update is computed based on the updated parameters. For example, the weight-update can be the difference between the values of the updated parameters and their original values before the overfitting process. The weight-update is compressed by using an encoder that generates a bitstream conforming to ISO/IEC 15938-17. The resulting bitstream, representing the compressed weight-update, is included into the byte sequence nnpfc_payload_byte of the NNPFC SEI message associated to the overfitted filter. The overfitted filter replaces the base filter with the same value of filter identifier.

For each picture, the encoder determines whether filtering is beneficial in terms of rate-distortion performance and, if so, which filter is the optimal one out of the four available filters (three base filters and one overfitted filter that replaced one base filter). If a filter is determined to be beneficial, an NNPFA SEI message is associated to that picture containing the filter identifier.

At the decoder side, the five NNPFC SEI messages are parsed. In particular, the byte sequence nnpfc_payload_byte of the NNPFC SEI message associated to the overfitted filter is decoded by using a decoder that is conformant with ISO/IEC 15938-17. The output of this decoding operation is a reconstructed overfitted filter. A decoded picture that is associated to a NNPFA SEI message is filtered by the filter indicated within that NNPFA message.

The NNPFC and NNPFA SEI messages can also be used to perform picture rate upsampling as a post processing step. Picture rate upsampling allows generating one or more interpolated pictures in between input pictures to the neural network post filter. The input pictures are the decoded output pictures from the decoder. The numbers of input pictures and interpolated pictures are signalled in the NNPFC SEI message. In addition, for the i-th input picture, a flag (nnpfc_input_pic_output_flag[i]) is signalled to indicate whether the NNPF will generate a corresponding output picture with the same output time. Thus, for picture rate upsampling purpose, the total number of pictures resulting from the post filtering process is derived based on the number of interpolated pictures and nnpfc_input_pic_output_flag[i]. In VTM, the following configuration parameters related to picture rate upsampling can be set on the encoder side:

SEINNPostFilterCharacteristicsNumberInputDecodedPicsMinusOnei: Specifies the number of decoded output pictures minus 1 used as input for the i-th neural network post filter. The default value is 0.

SEINNPostFilterCharacteristicsNumberInterpolatedPicsi: Specifies a list, where the j-th entry in the list specifies the number of interpolated pictures generated by the i-th neural network post filter between the j-th and (j+1)-th picture used as inputs for the post processing filter. The default value is an empty list.

SEINNPostFilterCharacteristicsInputPicOutputFlagi: Specifies a list, where the j-th entry in the list indicates whether the i-th neural network post filter generates a corresponding output picture for the j-th input picture. The default value is false.

4.5 Implementation related to phase indication SEI messages

When downsampling video content to generate multiple versions suitable for various bandwidth constraints, a low-pass filter is typically used. A phase shift may be introduced during downsampling, and

the relative location between samples at different resolutions may not be known to a client consuming the video. When switching between multiple resolutions a slight geometric shift may be noticeable if the phase shift used in the upsampler does not mirror the phase shift of the downsampler.

The phase indication SEI message provides the decoder with information about the position of luma sampling locations in cropped decoded pictures relative to a rendering window. This information may be used by a decoder to ensure the correct spatial alignment of rendered pictures, for example when switching between picture resolutions.

A phase indication SEI message includes information for horizontal phase and vertical phase which can be used to adjust the horizontal position and vertical position, respectively, of luma sampling locations relative to a rendering window. Figure 69 illustrates the horizontal and vertical phases for luma sampling location related to a rendering window. Each phase is represented by a numerator and a denominator (i.e., a and b for horizontal phase; c and d for vertical phase).

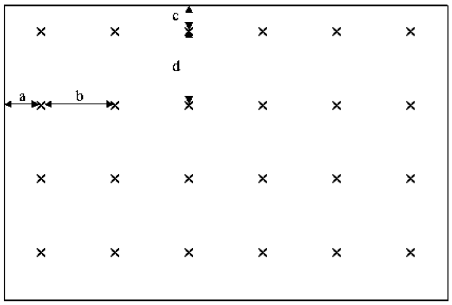


Figure 69: Illustration of horizontal and vertical phases for luma sampling location related to a rendering window. The ratios a/b and c/d represent the horizontal and vertical locations of the luma samples (marked with x) relative to a rendering window.

The VTM reference software includes features to generate phase indication SEI message for encoder and parsing functionality for decoder. The implementation can be used to verify the conformance of bitstreams including SEI processing order SEI messages.

The implementation of phase indication SEI message in the VTM encoder provides users with an ability to include information about the luma sampling location in the bitstream. It is assumed that the values to use for the phase indication are known a priori or are determined in a process outside of the VTM encoding process. Two of the most common approaches for phase positioning in downsampled videos are centered positioning and co-sited positioning. In centered positioning, the samples are positioned with equal distance to all edges of the rendering window, i.e., ½ sample in the resolution of the downsampled video (this is regardless of the downsampling ratio). In co-sited positioning, the top-left sample of the resampled video is aligned with the top-left sample of the original video, i.e., at position s/2 where s is the size of the downsampled video relative to the original video (for the respective dimensions).

The VTM encoder supports encoding with Reference Picture Resampling (RPR) where some input pictures are resampled and encoded in reduced resolution. The resampling process in VTM is performed using the co-sited approach.

Three example configuration files are provided for the phase indication SEI message:

- One that indicates centered positioning for encoding without RPR.
- One that indicates centered positioning for the full resolution and cosited positioning for the reduced resolution to be used together with RPR of scale factor 1.5.
- One that indicates centered positioning for the full resolution and cosited positioning for the reduced resolution to be used together with RPR of scale factor 2.

At decoder side, the phase indication SEI messages are parsed, and the content can be output using the `--OutputDecodedSEIMessagesFilename` parameter. The content of the phase indication SEI messages does not affect the decoding process or the output process in VTM.

4.6 Implementation related to post-filter hint SEI messages

To improve video quality of decoded pictures, it is possible to apply filters after the decoding process to the reconstructed pictures. The filters for such process are delivered (signalled) in post-filter hint SEI message. For example, reference picture resampling (RPR) feature is enabled, the outputted decoded pictures from the VVC decoder could have different resolutions. Prior to displaying, the outputted pictures may be resampled to make them having the same resolution. It is asserted that the resampling of the outputted pictures can be done outside of the decoding loop and the filters used for such a process do not have to the same as the filters used for the RPR during decoding process. To signal such filters for resampling of the decoded pictures, the existing post-filter hint SEI message could be used as it provides the generic way to signal filter coefficients for such purpose.

The post-filter hint SEI message provides the coefficients of a post-filter or correlation information for the design of a post-filter for potential use in post-processing of a set of pictures after they are decoded and output to obtain improved displayed quality.

The design of post-filter hint SEI message in H.274 | ISO/IEC 23002-7 has the following differences from the post-filter hint SEI messages specified in HEVC and AVC as follows:

- Signalling of a flag that specifies whether the SEI includes coefficients for chroma components in additional to the coefficients for luma component.
- Persistence of the SEI message. In HEVC and AVC, the SEI applies to only one picture (i.e., the picture in the access unit that contains the SEI message) whereas the post-filter hint SEI message in H.274 | ISO/IEC 23002-7 allows the SEI message to be applicable to a set of pictures.

4.7 Implementation related to SEI processing order SEI messages

The SEI processing order SEI message carries information indicating the preferred processing order, as determined by the encoder (i.e., the content producer), for different types of SEI messages that may be present in the bitstream. The SEI message signals pairs of values (i.e., SEI payload type and its preferred processing order) for each of two or more SEI messages.

When the signalled processing order of an SEI message is less than the signalled processing order of another SEI message, the SEI message is preferred to be processed before the other SEI message, as illustrated in Figure 70.

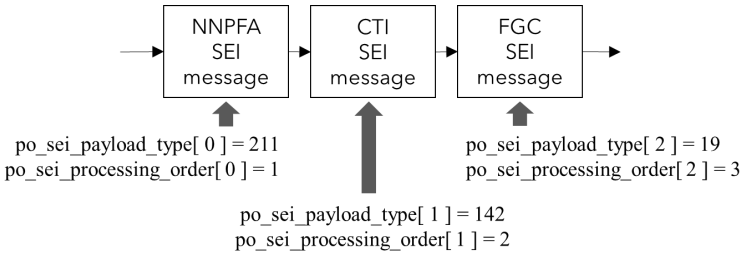


Figure 70: Preferred processing order of two SEI messages with different processing order values.

The SEI message could also be used by an encoder to indicate preference that a specific type of SEI message be processed before or after other SEI messages without indicating preference that those other SEI messages be processed in a particular order. For example, as illustrated in Figure 71, an encoder could indicate preference that a film grain characteristics (FGC) SEI message be processed after any colour transform information (CTI) and neural-network post-filter activation (NNPFA) SEI messages without indicating preference for order of processing between the CTI and NNPFA SEI messages.

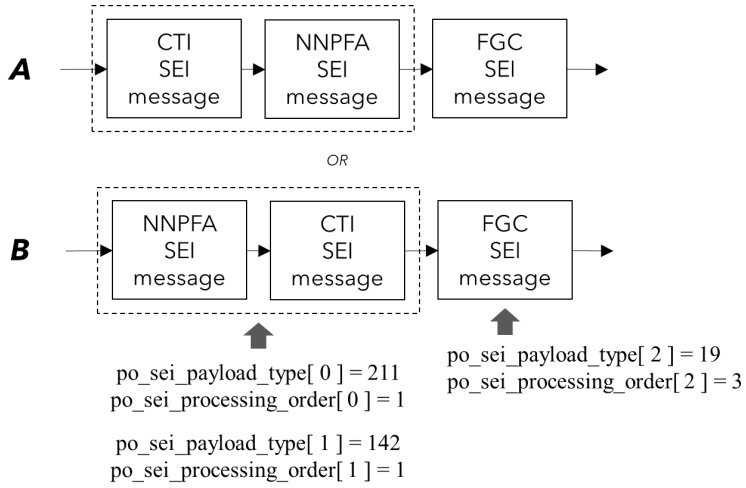


Figure 71: Preferred processing order of a specific SEI message type without indicating a preferred order of other SEI message types (i.e., SEI messages having same processing order values).

The VTM reference software includes features to generate SEI processing order SEI message for encoder and parsing functionality for decoder. The implementation can be used to verify the conformance of bitstreams including SEI processing order SEI messages.

5 Profiles, Levels and Tiers (PTL)

Video coding standards define profiles in order to restrict the feature set for specific applications. In the to-be-published VVC version 1 specification text, six profiles, namely Main 10, Main 10 Still Picture, Main 4:4:4 10, Main 4:4:4 10 Still Picture, Multilayer Main 10 profile and Multilayer Main 10 4:4:4 profile are defined.

The Main 10 profile has the capability to decode video streams with up to 10-bit bit depth and colour format as 4:2:0 or monochrome. Among all the coding features described in this document, palette mode is disallowed to be enabled in the two profiles. ACT can not be used since it's only applicable to 4:4:4 colour format. Main 4:4:4 10 profile supports colour format as 4:4:4, 4:2:2, 4:2:0 or monochrome and up to 10-bit bit depth. All the coding features in VVC version 1 can be enabled in the Main 4:4:4 10 profile. Main 10 Still Picture profile and Main 4:4:4 10 Still Picture profile share the same profile ID with their corresponding Main profiles with a constraint that the bitstream only contains one coded picture. Multilayer Main 10 profile and Multilayer Main 10 4:4:4 profile support multiple layer coding (e.g. containing more than one layer in one coded video sequence (CVS) and having inter-layer prediction), which improves the coding performance of scalability use cases. As single layer coding profiles, all coded pictures shall have same layer ID in one CVS of Main 10 profile and Main 10 4:4:4 profile.

In addition to profiles, VVC also defines levels and tiers, in a similar way as done in HEVC. For purposes of comparison of tier capabilities, the tier with general_tier_flag equal to 0 is considered to be a lower tier than the tier with general_tier_flag equal to 1. For purposes of comparison of level capabilities, a particular level of a specific tier is considered to be a lower level than another level of the same tier when the value of the general_level_idc or sublayer_level_idc[I] of the particular level is less than that of the other level.

Two new aspects regarding PTL in VVC have been introduced: the general constraints and the sub-profile concept. In VVC, almost each tool or feature has a corresponding general constraint flag defined. The main reason for this is to enable third parties, e.g., an application system standards body or even a company, to be able to easily turn off certain tools in case these tools are not conveniently useable by them without the need of going through the time-consuming process for specifying a new VVC profile. The sub-profile concept was introduced for a similar purpose. This enables a third-party to define a sub-profile, which can contain a subset of the tools/features contained in an existing VVC profile, by just going through a registration process as specified by Rec. ITU-T T.35.

In VVC version 2, nine more profiles are defined, namely Main 12, Main 12 Intra, Main 12 Still Picture, Main 12 4:4:4, Main 12 4:4:4 Intra, Main 12 4:4:4 Still Picture, Main 16 4:4:4, Main 16 4:4:4 Intra, and Main 16 4:4:4 Still Picture.

The Main 12 profile has the capability to decode video streams up to 12-bit bit depth in either 4:2:0 or monochrome colour formats, but otherwise the coding features in the Main 12 profile are identical to the Main 10 profile. VVC version 2 coding tools are not available in the Main 12 profile. Similarly, the Main 12 Still Picture profile is identical to the Main 10 Still Picture profile with the exception that it can decode streams of up to 12-bit bit depth. The Main 12 Intra profile is an extension of the Main 12 Still Picture profile which allows a coded video sequence to contain multiple Intra frames but no Inter frames.

The Main 12 4:4:4 profile supports 4:4:4, 4:2:2, 4:2:0, and monochrome colour formats with bit depths up to 12-bit. In addition to supporting the same coding features as the Main 10 4:4:4 profile, the following tools are also supported: Extended Precision Processing (EPP); Extended Regular Residual Coding (ERRC); Persistent Regular Residual Coding (PRRC); Reversed Last Significant Coefficient Position (RLSCP); and Extended Transform Skip Residual Coding (ETSRC). The Main 12 4:4:4 Still Picture profile has the same coding features as Main 12 4:4:4 profile but only supports single I-frames. Similarly, the Main 12 4:4:4 Intra profile has the same coding features as Main 12 4:4:4 profile but supports coded video sequences with multiple I-frames.

The Main 16 4:4:4, Main 16 4:4:4 Still Picture and Main 16 4:4:4 Intra profiles support the same colour formats and tools as the Main 12 4:4:4, Main 12 4:4:4 Still Picture and Main 12 4:4:4 Intra profiles but support bit depths up to 16-bit.

6 Description of VTM encoder and encoding methods

This section describes operation of VTM to select parameters such as coding mode, QP, encoder configurations etc.

6.1 Encoder configurations

6.1.1 Overview of encoder configurations

The VTM encoder is supplied with configuration files supporting three key prediction structures, as used in the JVET common test conditions [2]. These prediction structures are: intra only, random access and low delay. The reference picture list management depends on the temporal configuration.

6.1.1.1 Intra-only configuration

For intra-only (also known as ‘all intra’ and abbreviated as ‘AI’) coding, each picture in the source material is encoded as an IDR picture. No temporal reference pictures are used. One QP value is specified in the configuration file for all pictures and slices.

6.1.1.2 Random access configuration

For the random access configuration, the JVET common test conditions [4] specifies a hierarchical B structure with a GOP size of 32; Figure 72 shows the output and encoding order for each picture in the GOP. Each rectangle in the figure represents a picture. The numbers 0 to 32 at the bottom of the figure shows the POC value, representing the display order, of each picture, and the number inside the rectangles shows the encoding order. The random access configuration defines a hierarchy among different B pictures whereby each of the 6 hierarchical levels used is associated with a temporal identifier as indicated by the numbers 0 to 5 to the left in Figure 72. Pictures with lower temporal id are shown towards the top and they are used for reference more frequently. In this configuration, an intra picture is encoded at approximately one second interval in accordance with the `IntraPeriod` configuration option, configured based on the frame rate of the source material. The pictures located between successive intra pictures in display order are encoded as B-pictures by default.

In VVC, the picture buffer is constrained to ensure that no more than eight pictures (including the picture currently being decoded and pictures already decoded but awaiting output or kept for future reference) are present in the DPB at any one time. The pictures at temporal id 0 reference only pictures of lower POC values. Temporal layers 1-4 consist of referenced B pictures, while the highest temporal layer 5 contains non-referenced B pictures only. The number of active pictures in the reference picture lists RefPicList0 and RefPicList1 vary between 0 and 4. The QP of each inter-coded picture is derived by adding an offset to the base QP; the higher the temporal ID of a picture is, the larger its QP offset.

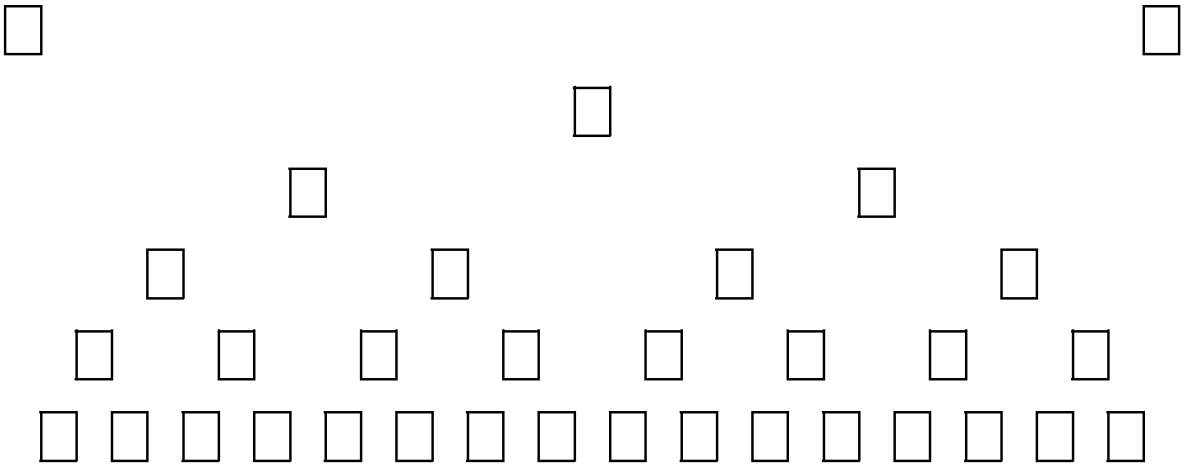


Figure 72: Display order, encoding order and temporal layers in the random access configuration.

6.1.1.3 Low-delay configurations

Two coding configurations have been defined for testing low-delay coding performance, referred to as ‘low-delay P’ and ‘low-delay B’. For the default low-delay configurations in JVET CTC, only the first picture in a video sequence is encoded as an IDR picture. Subsequent pictures are each encoded using a P-slice for low-delay P configuration or a B-slice for low-delay B configuration. For both modes, the P or B slices only reference pictures preceding the current picture in display order. For low-delay B mode, reference lists RefPicList0 and RefPicList1 have 4 reference pictures each.

Figure 73 illustrates the temporal prediction structure of the low-delay B configuration. The encoding/decoding order is the same as the display order. The first picture is intra coded and all other pictures are bi-predictively coded. The QP value of each coded picture is derived by adding an offset to a base QP. The QP offset value used for a picture depends on whether the picture is intra coded or inter coded, and for inter-coded pictures, the QP offset further depends on the location of the picture in the GOP.

Figure 73 should be read as follows. The first picture having a POC value equal to 0 is coded with a QP offset equal to -1 and it is referenced by all the following pictures until and including the picture with POC equal to 32. This is illustrated by the top red horizontal bar. Then a second picture with POC equal to 1 is coded with a QP offset equal to +6. This picture is only referenced by the very next picture. The third picture with POC equal to 2 is coded with a QP offset equal to +4 and it is referenced by the next 3 pictures. The GOP size is equal to 8 which means that the temporal prediction structure repeats after 8 pictures. The QP offset value is the same for all pictures marked with the same colour.

The number of active pictures in each of the RefPicList0 and RefPicList1 reference picture lists is equal to 4 except for the first few pictures in the bitstream that were coded at the beginning when less than 4 reference pictures were available. The closest preceding picture is always used as reference by the current picture, and it is always the first entry in both RefPicList0 and RefPicList1. All pictures are set with temporal layer ID equal to 0, but the QP offset values can be interpreted as indicators of “virtual”

temporal layers. Figure 73 shows that pictures coded with a lower QP value stay longer in the DPB and are referenced by a higher number of pictures compared with pictures coded with a higher QP value.

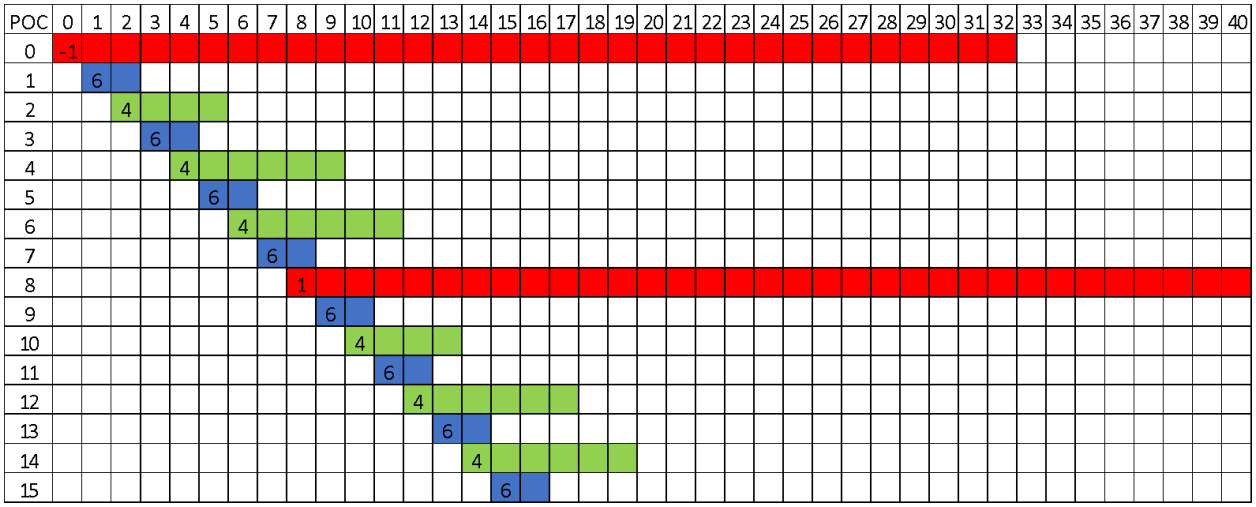


Figure 73: Temporal prediction structure of low-delay configuration.

6.2 Derivation process of coding tree structure

To be added.

6.3 Hash based motion estimation for screen content coding

The VTM reference software 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 74. 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 74) 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.

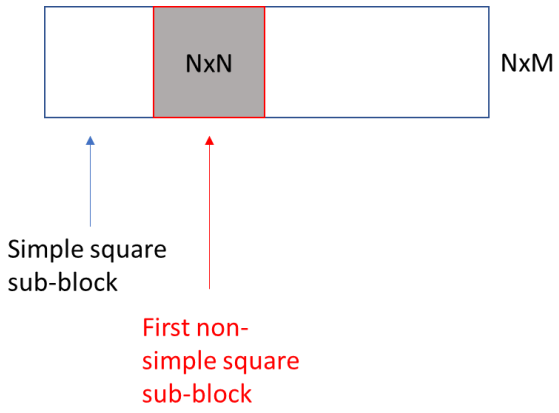


Figure 74 Motion estimation for rectangular block with hash values for square subblocks.

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_GPM 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.

6.4 Pre-encoding GOP-based motion compensated temporal filter (MCTF)

VTM supports MCTF, a temporal filter applied prior to the encoding operation, i.e., directly after reading input pictures, when the `TemporalFilter` configuration option is enabled as in the JVET CTC. The following steps describe this process in more detail:

Step 1: Filter strength determination

The strength of the temporal filter can be adjusted for different pictures in each GOP by using one or more `TemporalFilterStrengthFrame#` configuration options, where '#' is an integer. The specified strength is applied to all pictures whose POC number is exactly divisible by '#'; if this is true for more than one option, the value of the option with the highest '#' is used.

The filter is designed to be applied only to pictures that are low in the coding hierarchy. For example, for a random access configuration, the following two configuration lines can preferably be used:

```
TemporalFilterStrengthFrame8  : 0.95
TemporalFilterStrengthFrame16 : 1.5
```

This would enable the filter for all pictures with (POC % 8) == 0 using the following strengths:

$$s_o(n) = \{1.5,\ n = 0,\ 16,\ 32,\ etc\ 0.95,\ n = 8,\ 24,\ 40,\ etc\quad 0,\ otherwise$$

where n is the POC value of the picture.

For a low-delay configuration, the following configuration line can preferably be used:

```
TemporalFilterStrengthFrame4 : 0.4
```

This would enable the filter for all pictures with (POC % 4) == 0 using the following strengths:

$$s_o(n) = \{0.4,\ n = 0,\ 4,\ 8,\ etc\ 0,\ otherwise$$

The filter is not recommended to be applied in all intra configurations.

Step 2: When available, the `TemporalFilterPastRefs` (4 for random access and low delay configurations) pictures immediately preceding and the `TemporalFilterFutureRefs` (4 for random access and 0 for low delay configurations) pictures immediately following the current picture (in display order) are read directly from the source image file; the pictures available therefore may exist outside the current GOP.

Step 3: Motion is estimated of the available temporally neighboring pictures relative to the current picture per 8x8 sample block.

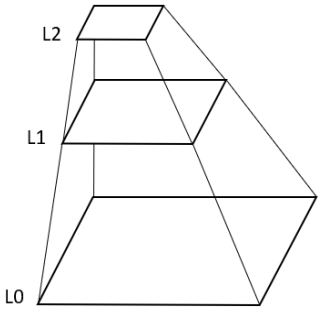
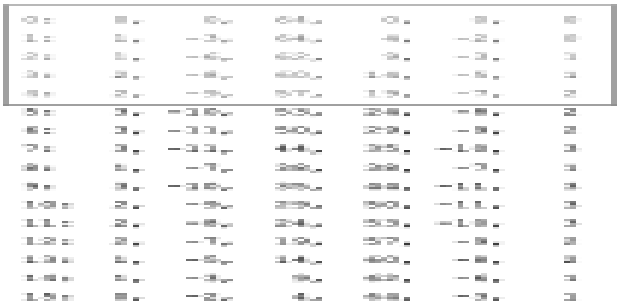


Figure 75: The different layers of the hierarchical motion estimation. L0 is the original resolution. L1 is a subsampled version of L0. L2 is a subsampled version of L1.

A hierarchical motion estimation scheme is used with layers L0, L1 and L2, as illustrated in Figure 75. Each sub-sampled layer is half the width and half the height of the lower layer and sub-sampling is done by computing a rounded average of four corresponding sample values from the lower layer.

First, motion estimation is performed for each 16x16 block in L2. The sum of squared difference is calculated for each selected motion vector and the motion vector corresponding to the smallest difference is selected. The selected motion vector is then used as initial value when estimating the motion in L1. Then the same is done for estimating motion in L0. As a final step, fractional precision motion is estimated for each 8x8 block by using the following 6-tap interpolation filter on L0.



Step 4: Motion compensation is applied on the pictures before and after the current picture according to the best matching motion for each 8×8 block to align the sample coordinates of each block in the current picture with the best matching coordinates in the referenced pictures.

Step 5: The samples of the current picture are then individually filtered for the luma and chroma channels as follows to produce a filtered current picture:

The new sample value, I_n, for the current picture is calculated using the following formula:

$$I_n = \frac{I_o + \sum_{i=-4}^{4} w_r(i,a) I_r(i)}{1 + \sum_{i=-4}^{4} w_r(i,a)} \quad (6\text{-}1)$$

Where I_o is the value of the original sample, $I_r(i)$ is the value of the corresponding sample in motion compensated picture i and $w_r(i,a)$ is the weight of motion compensated picture i given a value a. If `TemporalFilterFutureRefs` is equal to 0, a is set equal to 1 and if `TemporalFilterFutureRefs` is larger than 0, a is equal to 0.

For samples in the luma channel, the weights, $w_r(i,a)$, are calculated as follows:

$$w_r(i,a) = s_l s_o(n) s_r(i,a) w_a e^{-\frac{\Delta I(i)^2}{2\sigma_w \sigma_l(QP)^2}} \quad (6\text{-}2)$$

Where

$$s_l = 0.4$$

$s_r(i,4) = \{0.0,\ \ i = 0\ 1.13,\ \ |i| = 1\ 0.97,\ \ |i| = 2\ 0.81,\ \ |i| = 3\ 0.57,\ \ |i| = 4\ s_r(i,14) = \{0.0,\ \ i = 0\ 1.13,\ \ |i| = 1\ 0.97,\ \ |i|$

$$s_r(i,8)(i,08) = \{0.0,\ \ i = 0\ 0.85,\ \ |i| = 1\ 0.57,\ \ |i| = 2\ 0.41,\ \ |i| = 3\ 0.33,\ \ |i| = 4$$

The adjustment factors w_a and σ_w are calculated for use in computing $w_r(i,a)$, as follows:

$w_a = \frac{min(error)}{error+1} \times \{1.0\ if\ noise < 25\ 0.6\ else\ \times \{0.6\ if\ error > 100\ 1.0\ \ else\ \times \{1.2\ if\ error < 50\ 1.0\ else$ (6-3)

$$\sigma_w = \{1.0\ if\ noise < 25\ 0.8\ else\ \times \{1.0\ if\ error < 50\ 0.8\ else \quad (6\text{-}4)$$

The *noise* and *error* values are computed at a block granularity of 8×8 for luma and 4×4 for chroma.

$$\sigma_l(QP) = 3 * (QP - 10) \quad (6\text{-}5)$$

$$\Delta I(i) = I_r(i) - I_o \quad (6\text{-}6)$$

For the chroma channels, the weights, $w_r(i,a)$, is calculated as follows:

$$w_r(i,a) = s_c s_o(n) s_r(i,a) e^{-\frac{\Delta I(i)^2}{2\sigma_c^2}} \quad (6\text{-}7)$$

Where $s_c = 0.55$ and $\sigma_c = 30$.

Step 6: The filtered current picture is then used for the standard encoding process.

6.5 Rice parameter determination for Extended Transform Skip Residual Coding (ETSRC)

In VVC version 2, if the ETSRC is enabled using sps_ts_residual_coding_rice_present_in_sh_flag, the signalled value sh_ts_residual_coding_rice_idx_minus1 must be determined by the encoder.

In VTM the encoder determines sh_ts_residual_coding_rice_idx_minus1 as follows.

For the first frame, the value is set using the slice QP:

sh_ts_residual_coding_rice_idx_minus1 = Clip3(1, 8, (19 - iQP) / 6) – 1

If the frame consists of a single slice the following apply:

- For frames other than the first, the value of sh_ts_residual_coding_rice_idx_minus1 is based on an estimate of the TSRC Rice coding costs generated when coding the previous I-frame. Thus for the first I-frame the default value of sh_ts_residual_coding_rice_idx_minus1 will be used. For subsequent I, P and B-frames the value of sh_ts_residual_coding_rice_idx_minus1 will be that computed from coding the previous I-frame.
- The default or the value determined using the previous frame I-frame may be overridden. After analysing the percentage of IBC hash hits within such a frame, if the hit ratio is found to be less than 41%, sh_ts_residual_coding_rice_idx_minus1 is reset to 0.

6.6 Determination of last significant coefficient coding direction

In VVC version 2, the signalling of the last significant coefficient in a TB is reversed if sh_reverse_last_sig_coeff_flag is equal to 1.

In VTM, the value signalled by the encoder for sh_reverse_last_sig_coeff_flag is determined as follows:

- If the slice QP is greater than 12, sh_reverse_last_sig_coeff_flag is set to 0.
- Otherwise, if the slice is an intra slice and the intraperiod is 1, sh_reverse_last_sig_coeff_flag is set to 1.
- Otherwise, the value of sh_reverse_last_sig_coeff_flag is determined from the previous I-slice by comparing the distance of the last significant coefficient from the top left and the bottom right of each zeroed out TB. If in this I-slice the number of blocks with a last significant coefficient closer to the bottom right of the TB is greater than number of blocks with a last significant coefficient closer to the top left of the TB, the value of sh_reverse_last_sig_coeff_flag is set to 1.
- Otherwise, the value of sh_reverse_last_sig_coeff_flag is set to 0.

6.7 Encoding for subpicture extraction and merging

The VTM reference encoder can be configured to enable applications where extraction and merging of subpictures occur, e.g., in subpicture-based viewport adaptive streaming (VAS). For this purpose, the following encoder configurations are recommended:

1) defining the maximum number of ALF APSs,
2) defining an offset value to the ALF APS identifiers,
3) using a constant joint chroma coding residual (JCCR) sign,
4) disabling luma mapping with chroma scaling (LMCS),
5) disabling adaptive maximum binary tree size (AMaxBT),
6) disabling the use of scaling lists (ScalingList=0), and
7) enabling slice level delta QP signaling (SliceLevelDeltaQp=true).

Limiting the maximum number of ALF APSs per picture along with ALF identifiers' offset value allows to control the allocation of ALF APS identifiers in sub-streams which may be merged into a single bitstream. The allocation should be configured in such a manner that a unique range of ALF APS identifiers is assigned to each sub-stream so that multiple ALF APSs with the same identifier value but with different content will not be present for the same coded picture in the merged bitstream. The JCCR sign, LMCS, AMaxBT, and the use of scaling list are controlled with picture/sequence-level syntax elements and hence need to be the same in all encoded sub-streams. Furthermore, delta QP should be signaled in slice headers rather than in picture headers.

When the encoder is configured as above, any aggregation of sub-streams results in a conforming bitstream without any ALF APS identifier conflict. For example, in Figure 76 a subpicture partitioning of CMP content (3×3 grid in each CMP face) is shown as high- and low-quality versions in a) and b), respectively. For each version, a set of 4 ALF APSs having a unique identifier range is allocated: the high-quality version is allocated ALF APS identifiers of range {0 ... 3} and the low-quality version is allocated ALF APS identifiers of range {4 ... 7}.

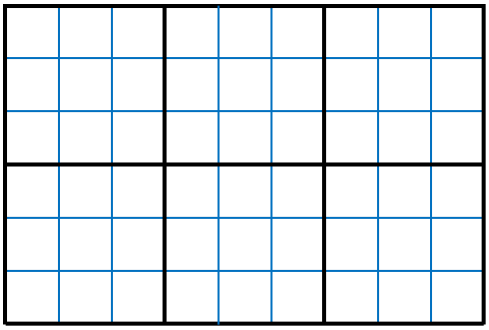
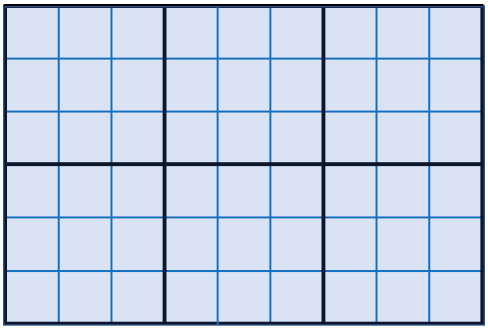


Figure 76: a) CMP subpicture partitioning (high-quality version), 4 ALF APSs are allocated in the range of {0 .. 3}



b) CMP subpicture partitioning (low-quality version), 4 ALF APSs are allocated in the range of {4 .. 7}

As another example, each version of a picture may be divided into several subpicture groups, with each version being allocated a unique range of ALF APS identifiers. In Figure 77, each picture version is divided into 4 subpicture groups (resulting in 8 subpicture groups in total in both versions), for which an ALF APS with a unique identifier is allocated to each subpicture group.

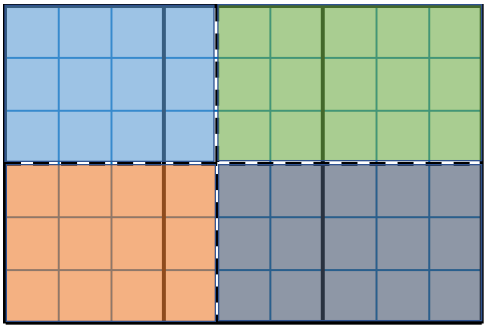
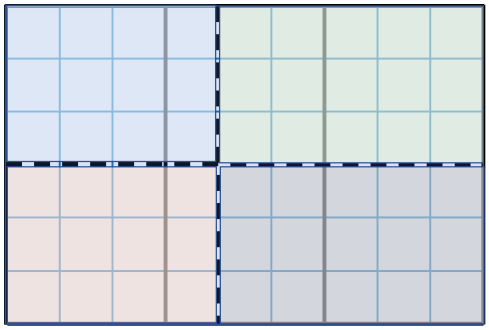


Figure 77: a) CMP subpicture partitioning (high-quality version), black dashed lines show ALF APS subpicture group boundaries, one ALF APS per group in the range of {0 .. 3}



b) CMP subpicture partitioning (low-quality version), black dashed lines show ALF APS subpicture group boundaries, one ALF APS per group in the range of {4 .. 7}

6.8 Rate Control

The VTM encoder implements a single-pass rate control algorithm which provides a constant bit rate (CBR) mode. The algorithm is controlled using the `RateControl` configuration option and is based on a *R*-λ model. This model assumes an exponential relationship between the coding rate and the Lagrange multiplier λ:

$$R = \alpha_1 \bullet \lambda^{\beta_1} \tag{6-8}$$

where α_1 and β_1 are constants dependent on the video source. The rate control process can be summarized into two main steps:

1. Bit allocation,
2. Quantization parameter derivation according to the *R*-λ model.

Bit allocation is performed at the following levels of granularity: GOP, picture and CTU level. Sequence-level information is held in an instance of `EncRCSeq`, GOP-level information is held in an instance of `EncRCGOP` created for each GOP, and picture-level information is held in an instance of

`EncRCPic`. Initially, the rate controller computes the average bits per picture (R_{PicAvg}) from the target bit rate (R_{tar}) and the video frame rate (f):

$$R_{PicAvg} = \frac{R_{tar}}{f} \tag{6-9}$$

Then, bit allocation at the GOP level is performed using an adaptive smoothing window (SW) based on IntraPeriod (IP) and GOP size and taking into account the number of pictures already coded (N_{coded}) and the bits spent on these pictures (R_{coded}). For the current GOP, the target bits (T_{GOP}) are then given as:

$$T_{GOP} = \frac{R_{PicAvg} \bullet \left(N_{coded} + SW\right) - R_{coded}}{SW} \bullet N_{GOP} \tag{6-10}$$

$$SW = \frac{\alpha * N_{GOP}}{(32, IP)} + \beta \tag{6-11}$$

where N_{GOP} denotes the number of pictures in the current GOP, and α and β are set as 20 and 60, respectively. At the picture level, bit allocation is performed taking into account T_{GOP} and the bits spent over the pictures inside the current GOP and already coded ($CODED_{GOP}$). The number of bits allocated for each picture also considers a weight. The weight models the relative number of bits that should be allocated to one picture with respect to the others. As an example, in the random access coding configuration, when a hierarchical GOP structure is used, pictures belonging to lower levels of the hierarchy will receive a larger number of bits since they will serve as reference for subsequent coded pictures (i.e. at higher levels of the hierarchy). Conversely, those located to the higher level of the hierarchy will receive fewer bits. The weights used in VTM are selected differently depending on whether random access or low-delay GOP type is used and depending on the bits per pixel (bpp) associated with the target bit rate (R_{tar}). Table 6-1 and Table 6-2 list the weights used for random access (16-picture GOP) and low-delay GOP configurations, respectively. Random access with a 32-picture GOP is also supported. The target coding rate for current picture ($T_{CurrPic}$) is then given as:

$$T_{CurrPic} = \frac{T_{GOP} - CODED_{GOP}}{\sum_{i \in \{non\ coded\ pictures\}} \omega_i} \bullet \omega_{CurrPic'} \tag{6-12}$$

From the allocated bit budget ($T_{CurrPic}$) the rate controller subtracts the number of bits associated with the header information (e.g. slice header, SPS, PPS, etc.) which is estimated from the header bits spent in previously coded pictures at the same hierarchical level.

Table 6-1. Values for ωCurrPic for random access GOP 16-picture configuration.

Coding order	POC in the GOP structure	bpp > 0.2	0.2 ≥ bpp ≥ 0.1	0.1 ≥ bpp ≥ 0.05	Otherwise
1	16	10	15	40	40
2	8	8	9	17	15
3	4	4	4	7	6
4	2	2	2	2	3
5	1	1	1	1	1
6	3	1	1	1	1
7	6	2	2	2	3
8	5	1	1	1	1

9	7	1	1	1	1
10	12	4	4	7	6
11	10	2	2	2	3
12	9	1	1	1	1
13	11	1	1	1	1
14	14	2	2	2	3
15	13	1	1	1	1
16	15	1	1	1	1

Table 6-2. Values for ωCurrPic for low-delay GOP configuration.

Coding order	**POC in the GOP structure**	**bpp > 0.2**	**0.2 ≥ bpp ≥ 0.1**	**0.1 ≥ bpp ≥ 0.05**	**Otherwise**
1	1	2	2	2	2
2	2	3	3	3	3
3	3	2	2	2	2
4	4	3	3	3	3
5	5	2	2	2	2
6	6	3	3	3	3
7	7	2	2	2	2
8	8	6	10	12	14

Finally, for each CTU the bit rate is allocated taking into account the number of bits allocated for the picture to which the CTU belongs and the number of bits spent while encoding previous CTUs ($CODED_{CTU}$). As for the case of picture bit allocation, CTUs are weighed depending on their position within the picture. More precisely, the weight for the current CTU ($\omega_{CurrCTU}$) is set equal to the sum of absolute transform differences (SATD) for intra coded slices or to the number of bits estimated using the *R-λ* model in (6-8) for inter coded slices. Therefore the target bits allocated for the current CTU ($T_{CurrCTU}$) is given by:

$$T_{CurrCTU} = \frac{\omega_{CurrCTU} - \left(\sum_{i \in \{non\ coded\ CTUs\}} \omega_i - CODED_{CTU} \right)}{SW_{CTU}}, \qquad (6\text{-}13)$$

where SW_{CTU} = min(4, *non coded CTUs*) and denotes a smoothing window used to spread the allocated bits over the next SW_{CTU} CTUs. As may be noted from the formula to compute $T_{CurrCTU}$, the initial estimated bits for the current CTU are adjusted by the amount of bits overspent or underspent in the current picture up to the coding for this CTU.

Once the available bits have been allocated for the current CTU, the quantization parameter QP can be obtained according to the *R-λ* model. More precisely, the Lagrange multiplier is derived from $T_{CurrCTU}$ as follows:

$$\lambda = \alpha \cdot \left(\frac{T_{CurrCTU}}{N}\right)^{\beta} \quad (6\text{-}14)$$

where N denotes the number of luma samples contained in the current CTU. Parameters α and β are initialized to average values of (3.2003 and -1.367, respectively) and then updated by least square regression once a coded picture is available, which considers the previous picture at the same temporal level. The updating strategy is based on the following RD model:

$$D = CR^{-K} \quad (6\text{-}15)$$

With lambda being the slope of the line tangential to the RD curve:

$$\lambda = -\frac{\partial D}{\partial R} = CK \cdot R^{-K-1} \quad (6\text{-}16)$$

where λ, R, D are obtained after encoding a picture and the two variables C, K are estimated with the above two equations. Then, the updated C and K variables are available for use as the new R-D parameters.

The parameter update considers only non-skipped CTUs of the picture, avoiding the very flat RD curve of skipped CTUs from influencing the parameter update.

Moreover, different values are used with respect to level of hierarchy of the GOP structure.

From λ, QP is finally obtained as:

$$QP = \lfloor 4.2005 \cdot \ln ln\,(\lambda) + 13.7122 \rfloor \quad (6\text{-}17)$$

where the $\lfloor \cdot \rfloor$ operator returns the largest integer smaller than or equal to the argument and ln(·) is the natural logarithm.

The VTM rate control algorithm allows to manually set the initial QP value used for the first coded picture using option `--InitialQP`=*value*. If *value* is equal to zero, the algorithm automatically derives the value using the R-λ model otherwise VTM uses the specified value. The remaining parameters associated with the rate control algorithm are listed in Table 6-3 along with a brief description.

Table 6-3. Configuration options for rate control algorithm.

Parameter name	Description
`TargetBitrate`	Target bit rate for the whole sequence measured in bit per second (bps).
`KeepHierarchicalBit`	Determines how bit allocation is done across different pictures. Allowed values are: 0 = uniform, 1 fixed ratio according to the used GOP structure (i.e. random access or low-delay) and 2 = adaptive with respect to the source content.
`LCULevelRateControl`	Switch between CTB-based or picture-based rate control.
`RCLCUSeparateModel`	Selects whether to use α and β parameters on a CTB or picture basis.
`InitialQP`	When specified this initial QP is used for the first (Intra) picture. If `RCForceIntraQP` is enabled, then this value is used for all pictures at temporal layer 0. When this option has effect, λ is set as follows: $\lambda = 0.57 \times \left(1 - Clip\left(0,\ 0.5,\ 0.5 \times GOP_{size} - 1\right)\right) \times 2^{\frac{(QP_{slice}-12)}{3}}$

	Where GOP_{size} is the size of the GOP in frames and QP_{slice} is the value specified with the '`InitialQP`' option.
`RCForceIntraQP`	Force QP value for intra-pictures to be equal to the value specified with `InitialQP`.

The following subsections will describe the rate control workflow for bit allocation at different levels (e.g. GOP, picture and CTB) and updating of the model parameters.

6.8.1 Workflow for bit allocation and lambda estimation

6.8.1.1 Sequence-level operation

The function `init` for class `RateCtrl` and the function `create` for class `EncRCSeq` perform the sequence-level rate allocation. Accordingly, a parameter '*adaptiveBit*' is defined. When `KeepHierarchicalBit` is equal to 2 and a GOP size of 8 is used in a low-delay configuration, *adaptiveBits* is set equal to 1. When `KeepHierarchicalBit` is equal to 2 and a GOP size of 16 is used in a non low-delay configuration, (i.e. random access), *adaptiveBits* is set equal to 2. Otherwise, *adaptiveBits* is set equal to 0.

A list of ratios *bitsRatio* is also set for the above two GOP structures. Initially the ratios of *bitsRatio* are set with the values listed in Table 6-1 and Table 6-2. Then, depending on the value for parameter *adaptiveBit*, these values are updated on a GOP basis with the number of bits spent in the previous GOP.

6.8.1.2 GOP-level operation

The function `create` of class `EncRCGOP` performs the GOP-level rate allocation. Firstly, the target bits for the current GOP are estimated as follows:

$$TargetBits_{GOP} = max\left(GOP_{size} \times \frac{RemainBits_{Seq} - \frac{TargetBits_{Seq}}{TotalFrames_{Seq}} \bullet \left(picsLeft_{Seq} - picCount\right)}{picCount}, 200\right) \quad \text{(6-18)}$$

where *picCount* denotes a smoothing window determined according to GOP size and IntraPeriod. If the value of the parameter *adaptiveBit* (see Section 6.8.1.1) is greater than zero and the number of m_listRCPictures is greater than GOP Size then the values for *bitsRatio* are adjusted as follows. Let λ_{L1} denote the Lagrange multiplier for the picture at hierarchical level equal to 1 in the GOP structure and let *lambdaRatio*(i) for $i = 0,\ldots,GOP_{size} - 1$ denote an array of Lagrange multiplier ratios computed as listed in

Table 6-5. . For non low-delay configuration with GOP size of 16 and 32, *lambdaRatio*(i) is derived based on quality dependency factor (QDF). A GOP in RA configuration contains 16 pictures belonging to 5 coding levels, where the pictures in higher level will choose the pictures in the lower levels as their reference pictures. Thus, there exists quality dependency among different pictures in different coding level, which can be expressed as:

$$\delta_{i,j} = \frac{\partial D_j}{\partial D_i} \quad \text{(6-19)}$$

where D means the distortion (mean square error), and the QDF can be obtained by a linear function with the skip ratio as:

$$\begin{aligned}\delta_{1,2}&=A_{1,2}S_2+B_{1,2}\\ \delta_{1,3}&=A_{1,3}S_3+B_{1,3}\\ \delta_{1,4}&=A_{1,4}S_4+B_{1,4}\\ \delta_{1,5}&=A_{1,5}S_5+B_{1,5}\end{aligned} \tag{6-20}$$

where S_j represents the skip ratio of j th coding levels which can be acquired from last encoded GOP, $A_{1,j}$ and $B_{1,j}$ are the parameters obtained by some analysis of VTM software when rate control is not applied. The parameters $A_{1,j}$ and $B_{1,j}$ are listed in Table 6-4.

Table 6-4. Parameters of relationships between QDF and skip ratio.

Level i	$A_{1,j}$	$B_{1,j}$
2	0.5847	-0.0782
3	0.5468	-0.1364
4	0.6539	-0.203
5	0.8623	-0.4676

The QDF of other coding levels can be obtained with:

$$\delta_{i,j}=\delta_{1,j}\times(1-\delta_{1,i}) \tag{6-21}$$

And $lambdaLev_i$ represents the influence of picture i :

$$\text{, } i\in level\,1\ \frac{1}{1+(3\times\delta_{2,3}+5\times\delta_{2,4}+8\times\delta_{2,5})}\text{, } i\in level\,2\ \frac{1}{1+(2\times\delta_{3,4}+4\times\delta_{3,5})}\text{, } i\in level\,3\ \frac{1}{1+(2\times\delta_{4,5})}\text{, } i\in level\,4\ 1.0\text{, } i\in level\,5 \tag{6-22}$$

By inverting (6-8) to derive λ as a function of R it yields:

$$\lambda = \alpha\cdot R^{\beta} \tag{6-23}$$

Given the computation of *lambdaRatio*(i) values, for each frame i according to coding order in the GOP the associated rate $R(i)$ can be obtained as:

$$R(i)=\left(\frac{lambdaRatio(i)}{\alpha(i)}\right)^{\frac{1}{\beta(i)}}=A(i)^{B(i)} \tag{6-24}$$

where coefficients $A(i)$ and $B(i)$ are computed by the function `xCalEquaCoeff` of class `EncGOP`. Given the bit budget for the current GOP $TargetBits_{GOP}$, coefficients $A(i)$ and $B(i)$ and the number of luma samples $P(i)$, the function `xSolveEqua` of class `EncGOP` computes λ^* by solving the following equation:

$$TargetBits_{GOP}=\sum_{i=0}^{GOP_{size}-1}A(i)\bullet(\lambda^*)^{B(i)}\cdot N(i) \tag{6-25}$$

The equation is solved using the bisection method where the maximum number of iterations is set to 20. After λ^* is obtained, the function `setAllBitRatio` in class `EncRCSeq` sets the weights ω_i for all pictures in the GOP as:

$$\omega_i = \lfloor A(i)\cdot(\lambda^*)^{B(i)} + 0.5 \rfloor \bullet (PicSize) \quad (6\text{-}26)$$

where *PicSize* denotes the number of luma samples in a picture. From the derived ω_i and inter coded pictures, the rate allocation can be performed using formula (6-12) and subtracting the estimated rate for the headers. For intra-coded pictures, the rate ($T_{CurrPic}$) is modified as follows: let *TotalCostIntra* be the picture SATD computed over a non-overlapping grid of 8×8 blocks where for each block the average value of the associated luminance pixels is subtracted to each sample prior to apply the Hadamard transform. Then let α_{INTRA} and β_{INTRA} be defined as:

$$\alpha_{INTRA} = f(x) = \{0.25,\ T_{CurrPic} \bullet 40 < PicSize\ 0.3,\ otherwise\,, \qquad \beta_{INTRA} = 0.5582 \quad (6\text{-}27)$$

The bits allocated for this intra picture ($T_{CurrPic}$) are finally modified as:

$$T_{CurrPic} = \lfloor \alpha_{INTRA} \bullet \left(\frac{4 \bullet TotalCostIntra}{T_{CurrPic}} \right)^{\beta_{INTRA}} \bullet T_{CurrPic} + 0.5 \rfloor \quad (6\text{-}28)$$

Table 6-5. Values for *lambdaRatio(i)* used to derive the *bitsRatio* for a GOP.

		Random access GOP configuration	Low-delay GOP configuration	
Coding Order	POC in the GOP structure		$\lambda_{L1} < 120$	Otherwise
0	16	1	1.3 × *lambdaRatio(1)*	5
1	8	lambdaLev2 / lambdaLev1	0.725 × ln(λ_{L1}) + 0.5793	4
2	4	lambdaLev3 / lambdaLev1	1.3 × *lambdaRatio(1)*	5
3	2	lambdaLev4 / lambdaLev1	0.725 × ln(λ_{L1}) + 0.5793	4
4	1	lambdaLev5 / lambdaLev1	1.3 × *lambdaRatio(1)*	5
5	3	lambdaLev5 / lambdaLev1	0.725 × ln(λ_{L1}) + 0.5793	4
6	6	lambdaLev4 / lambdaLev1	1.3 × *lambdaRatio(1)*	5
7	5	lambdaLev5 / lambdaLev1	1.0	1
8	7	lambdaLev5 / lambdaLev1		
9	12	lambdaLev3 / lambdaLev1		
10	10	lambdaLev4 / lambdaLev1		
11	9	lambdaLev5 / lambdaLev1		
12	11	lambdaLev5 / lambdaLev1		
13	14	lambdaLev4 / lambdaLev1		
14	13	lambdaLev5 / lambdaLev1		
15	15	lambdaLev5 / lambdaLev1		

6.8.1.3 Picture-level operation

At picture level, two main operations take place:

- Estimation of the picture level Lagrange multiplier λ and QP

- Bit allocation for each CTU with subsequent estimation of the associated Lagrange multiplier λ_{CTU} and QP.

For the estimation of the picture level λ_{Pic}, the function `estimatePicLambda` of class `EncRCPic` performs this estimation differently for intra and inter coded slices. For inter coded slices, λ is given as:

$$\lambda_{Pic} = \alpha_{Pic} \bullet \left(\frac{T_{CurrPic}}{PicSize} \right)^{\beta_{Pic}} \tag{6-29}$$

Conversely, for intra-coded slices, the value λ of is given as:

$$\lambda_{Pic} = \frac{\alpha_{Pic}}{256} \bullet \left(\frac{TotalCostIntra}{T_{CurrPic}} \right)^{\beta_{Pic}} \tag{6-30}$$

The obtained λ_{Pic} is then clipped using the values obtained in previously coded pictures belonging to the same hierarchical level in the GOP structure. More precisely, let $\lambda_{SameLevel}$ and $\lambda_{LastPicture}$ denote the Lagrange multiplier value for the last coded picture at the same hierachical level and the last coded picture, respectively. And $\lambda_{LastValidPicture}$ presents the last valid Lagrangian multiplier value (greater than zero) of the coded picture. The current value of λ_{Pic} is clipped by the following four ordered steps:

$$\lambda_{Pic} = Clip3(0.5 \bullet \lambda_{SameLevel}, 2 \bullet \lambda_{SameLevel}, \lambda_{Pic}) \tag{6-31}$$

$$\lambda_{Pic} = Clip3(\lambda_{LastPicture} \bullet 2^{\frac{-10}{3}}, \lambda_{LastPicture} \bullet 2^{\frac{10}{3}}, \lambda_{Pic}) \tag{6-32}$$

$$\lambda_{Pic} = Clip3(\lambda_{LastValidPicture} \bullet 2^{\frac{-10}{3}}, \lambda_{LastValidPicture} \bullet 2^{\frac{10}{3}}, \lambda_{Pic}) \tag{6-33}$$

$$\lambda_{Pic} = Clip3(0.1, 10000 \bullet 2^{bitdepth\_luma\_scale}, \lambda_{Pic}) \tag{6-34}$$

where $bitdepth\_luma\_scale = 2 \cdot (bit\_depth - 8)$. After the clipping, λ_{Pic} is used to compute the quantity $\omega_{CurrCTU}$ for each CTU. This quantity represents an estimate of the number of bits which will be allocated to that CTU using the R- λ model:

$$\omega_{CurrCTU} = \left(\frac{\lambda_{Pic}}{\alpha_{CTU}} \right)^{\frac{1}{\beta_{CTU}}} \tag{6-35}$$

where α_{CTU} and β_{CTU} denote the α and β parameters derived over previously coded CTUs at the same position and hierarchical level of *CurrCTU*. After the computation of each $\omega_{CurrCTU}$, their value is normalized between [0, 1]. Finally the value of the picture level QP (QP_{PIC}) is computed using (6-17) and clipped to avoid large quality fluctuations among frames.

At CTU level, the function `getLCUTargetBpp` of class `EncRCPic` computes the target bits for each CTU ($T_{CurrCTU}$). For inter coded slices, $T_{CurrCTU}$ is computed as specified in (6-13). For intra coded slices, $T_{CurrCTU}$ is computed as follows.

A variable *remainingCostIntra* is initialized to the same value of *TotalCostIntra* in Section 6.8.1.2. A CTU cost estimate *MAD* is derived by summing up the SATD values for all 8×8 blocks belonging to the current CTU. Also, for each CTU at position *idx* in raster scan order inside each picture, the variable *TargetBitsLeft*(*idx*) is defined as:

$$TargetBitsLeft(idx) = \sum_{j=PicSizeInCtu}^{idx} \frac{MAD(idx)}{TotalCostIntra} \bullet T_{CurrPic} \tag{6-36}$$

If *remainingCostIntra* > 0.1 the variable *weightedBitsLeft* is derived as follows:

$$weightedBitsLeft = \frac{T_{CurrPic} \bullet bitrateWindow + (T_{CurrPic} - TargetBitsLeft(idx)) \cdot CTUsLeft)}{bitRateWindow} \quad (6\text{-}37)$$

Where *bitrateWindow* denotes a window to spread the bit budget across the next (up to) 4 CTUs in raster scan order.Then, the target bits allocated for the current CTU ($T_{CurrCTU}$) is given as:

$$T_{CurrCTU} = \lfloor MAD \bullet \frac{weightedBitsLeft}{remainingCostIntra} + 0.5 \rfloor \quad (6\text{-}38)$$

If instead *remainingCostIntra* ≤ 0.1, then $T_{CurrCTU}$ is derived as follows:

$$T_{CurrCTU} = \lfloor \frac{TargetBitsLeft(idx)}{CTUsLeft} + 0.5 \rfloor \quad (6\text{-}39)$$

The function `getLCUTargetBpp` returns $T_{CurrCTU}$ expressed as bits per pixel (bpp) which is computed as:

$$bpp = \frac{T_{CurrCTU}}{CTU_{pixels}} \quad (6\text{-}40)$$

Where CTU_{pixels} includes adjustment for reduces sample count of CTUs along the right edge and the bottom row of a picture.

The function `getLCUEstLambda` in `EncRCPic` computes the Lagrange multiplier for the current CTU $\lambda_{CurrCTU}$ as:

$$\lambda_{CurrCTU} = \alpha_{CTU} \times (bpp)^{\beta_{CTU}} \quad (6\text{-}41)$$

Where, as above, α_{CTU} and β_{CTU} denote the α and β parameters derived over previously coded CTUs at the same spatial position and hierarchical level of *CurrCTU*. As for the picture level case, also at the CTU level $\lambda_{CurrCTU}$is clipped to avoid large quality and coding mode selection swings between CTUs in the same neighbourhood. More precisely, a search for $\lambda_{neighbour}$ is performed by searching back from previous CTUs until a valid value (greater than zero) is found. If $\lambda_{negihbour}$ is found, $\lambda_{CurrCTU}$ is updated as follows:

$$\lambda_{CurrCTU} = Clip3\left(\lambda_{neighbour} \times 2^{\frac{-1}{3}}, \lambda_{neighbour} \times 2^{\frac{-1}{3}}, \lambda_{CurrCTU}\right) \quad (6\text{-}42)$$

If λ_{Pic} is available, then $\lambda_{CurrCTU}$ is constrained as follows:

$$\lambda_{CurrCTU} = Clip3\left(\lambda_{Pic} \times 2^{\frac{-2}{3}}, \lambda_{Pic} \times 2^{\frac{-2}{3}}, \lambda_{CurrCTU}\right) \quad (6\text{-}43)$$

Otherwise a generic constraint is applied:

$$\lambda_{est} = Clip3\left(10 \bullet 2^{bitdepth\_luma\_scale}, 1000 \bullet 2^{bitdepth\_luma\_scale}, \lambda_{CurrCTU}\right) \quad (6\text{-}44)$$

The function `getLCUEstQP` of `EncRCPic` computes the estimated quantization parameter for the current CTU (QP_{CTU}). Let $QP_{neighbour}$ be the QP value obtained by searching back from previously coded CTUs until a valid value (i.e., greater than zero) is found. Using also QP_{PIC}, the value for QP_{CTU} is derived by applying the following ordered steps:

$$if\ QP_{neighbour} > 0\ then\ QP_{CTU} = Clip3(QP_{neighbour} - 1, QP_{neighbour} + 1,\ QP_{CTU}) \quad (6\text{-}45)$$

$$QP_{CTU} = Clip3(QP_{Pic} - 2, QP_{Pic} + 2,\ QP_{CTU}) \quad (6\text{-}46)$$

6.8.2 Workflow for parameters update

After one CTU or picture has finished being encoded, the state of the rate controller needs to be updated so that the model parameters can adjust their value to the characteristics of the video content. In particular, the following parameters are modified:

- Bits spent and frames encoded
- α_{Pic} and β_{Pic} parameters as well as the estimate for the bits spent in the slice header
- α_{CTU} and β_{CTU}

The following subsections will describe how the aforementioned updates are computed.

6.8.2.1 Update of parameters after one picture is encoded

The function `updateAfterPic` of class `EncRCSeq` updates the global counters which store the bits spent so far as well as the total coded frames. The same kind of update is also performed in the function `updateAfterPicture` of class `EncRCGOP`. Conversely, the function `updateAfterPicture` of class `EncRCPic` updates parameters α_{Pic} and β_{Pic} using the number of bits spent to encode the picture. More precisely, the variables ln_{bpp} and λ_{diff} are set as follows:

$$ln_{bpp} = ln\left(\left(\frac{TotalCostIntra}{PicSize}\right)^{1.2517}\right) \quad (6\text{-}47)$$

$$\lambda_{diff} = \beta \bullet \left(ln(R_{Spent}) - ln(T_{CurrPic})\right) \quad (6\text{-}48)$$

$$\lambda_{diff} = Clip3(-0.125,\ 0.125,\ 0.25 \bullet \lambda_{diff}) \quad (6\text{-}49)$$

Where R_{Spent} denotes the bits spent to encode the current picture in units of bits per luma sample. If the current picture is intra coded then parameters α_{Pic} and β_{Pic} are updated as follows:

$$\alpha_{Pic} = \alpha_{Pic} \bullet e^{\lambda_{diff}} \quad (6\text{-}50)$$

$$\beta_{Pic} = \beta_{Pic} + \frac{\lambda_{diff}}{ln_{bpp}} \quad (6\text{-}51)$$

If instead the current picture is inter coded, λ_{non_skip} and D_{non_skip} are the average Lagrange multiplier value and mean square error of non-skip CTU, and R_{Spent} is used to represent the approximation of non-skip CTU consumption bits (assume that the bits consumed by skip CTUs are negligible):

$$K_{update} = \frac{R_{Spent} \bullet \lambda_{non\_skip}}{D_{non\_skip}} \quad (6\text{-}52)$$

$$C_{update} = \frac{D_{non\_skip}}{R_{Spent}^{-K_{update}}} \quad (6\text{-}53)$$

then α_{Pic} and β_{Pic} are updated as follows:

$$\alpha_{Pic} = C\alpha_{updatePic} \bullet K_{update} \quad (6\text{-}54)$$

$$\beta_{pic} = -K_{update} - 1 \quad (6\text{-}55)$$

After computation, α_{Pic} and β_{Pic} are clipped in the range of [0.05, 500• $2^{bitdepth\_luma\_scale}$] and [-3.0, -0.1], respectively.

Along with the update of α_{Pic} and β_{Pic}, the function `setPicPara` stores the updated value in the memory structure `m_picPara` of class `EncRCSeq`. This structure contains α_{Pic} and β_{Pic} for all pictures at different hierarchical levels in the GOP. Finally, the object `EncRCPic` associated with the current picture is copied to the memory structure `m_listRCPictures` which is a C++ list modelling a circular buffer with maximum capacity equal to 32.

The information stored in this buffer is then used to estimate the bits spent in coding the slice header and clip the values for λ_{Pic} and QP_{Pic} (see Section 6.8.1.3).

Finally, if the hierarchical level of the picture being updated is one, the value for the Lagrange multiplier λ_{L1} (see Section 6.8.1.2) is updated using a weighting of 0.5 of the previous λ_{L1} value computed and a weighting of 0.5 of the λ_{calc}.

6.8.2.2 Update of parameters after one CTU is encoded

The function `updateAfterCTU` in class `EncRCPic` is called to update α_{CTU} and β_{CTU}. The update process is similar to picture-level update of inter coded pictures. Firstly, the bits per pixel *bpp* is computed:

$$bpp = \frac{R_{CTU}}{CTU_{pixels}} \quad (6\text{-}56)$$

where R_{CTU} denotes the coding bits spent on the current CTU. As for the picture case, λ_{cal} is then derived using *bpp* and α_{CTU} and β_{CTU}.

$$\lambda_{cal} = \alpha_{CTU} \times bpp^{\beta_{CTU}} \quad (6\text{-}57)$$

And parameter K_{update} and C_{update} is calculated by:

$$K_{update} = \frac{bpp \bullet \lambda_{cal}}{D_{CTU}} \quad (6\text{-}58)$$

$$C_{update} = \frac{D_{CTU}}{bpp^{-K_{update}}} \quad (6\text{-}59)$$

then α_{CTU} and β_{CTU} are updated as follows:

$$\alpha_{CTU} = C_{update} \bullet K_{update} \quad (6\text{-}60)$$

$$\beta_{CTU} = -K_{update} - 1 \quad (6\text{-}61)$$

Once α_{CTU} and β_{CTU} are updated the function `updateAfterCTU` stores their values in the memory structure `m_LCUPara` of class `EncRCSeq`. This structure stores the parameters for all CTUs in a frame at different temporal hierarchy level so that the rate controller can update the statistics according to the video content.

6.8.3 Target bits saturation for rate control

When `RCCpbSaturation` is enabled, a 'target bits saturation' method is used to avoid overflow or underflow of the coded picture buffer (CPB).

The hypothetical reference decoder (HRD) specifies a coded picture buffer characterized by three parameters (*R*, *B*, *F*) where *R* denotes the transmission bit rate, *B* is the CPB size, and *F* is the CPB

fullness. The model assumed by the HRD is the so-called leaky bucket model. Figure 78 shows an example of CPB with the HRD parameters. Then, for frame i, b_i is the number of bits for the frame at time t_i, with frame rate f.

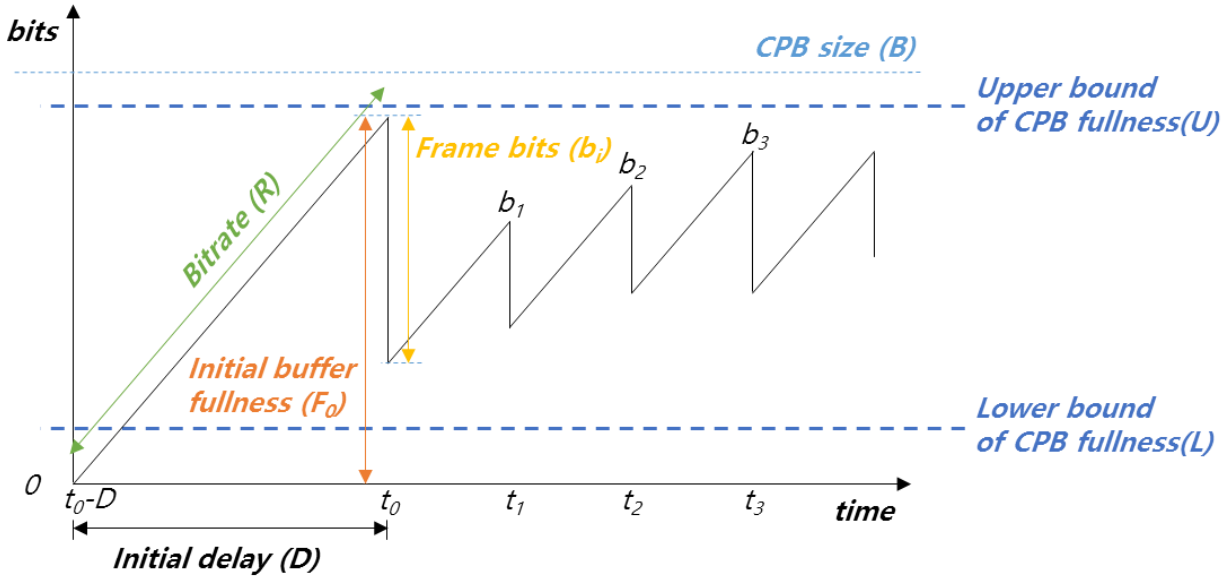


Figure 78 - Example of CPB behavior.

As seen in Figure 78, an upper bound and lower bound of CPB fullness are defined. The upper bound is set at 90% and the lower bound is set adaptively.

To prevent overflow, i.e. to make sure that the buffer fullness will stay below the upper bound when the picture at time instant i is decoded, the target bits allocated for frame i, $b(i)$, is adjusted as follows:

Let $B_e(i)$ be the estimated buffer fullness at time i, given as:

$$B_e(i) = F(i) + \frac{R}{f} \tag{6-62}$$

where $F(i)$ denotes the actual buffer fullness at time i. If the following holds true:

$$B_e(i) - b(i) > B \times 0.9 \tag{6-63}$$

then the target bit $b(i)$, is adjusted as:

$$b(i) = B_e(i) - B \times 0.9 \tag{6-64}$$

Otherwise, $b(i)$ is left with the same value as allocated by the process described in Section 6.8.

To prevent underflow, i.e. to make sure that the buffer fullness will stay above the lower bound when the picture at time instant i is decoded, $b(i)$ is adjusted as follows. The lower bound L(i) is determined by:

$$L(i) = \{ \frac{T_{GOP}}{\sum_{j=0\ldots N-1} \omega_j} \bullet \omega_0 \ if\ i\ is\ the\ last\ picture\ in\ the\ GOP \ \frac{T_{GOP} - CODED_{GOP}}{\sum_{j \in \{non\ coded\ pictures\}} \omega_j} \bullet \omega_{CurrPic} \ otherwise \tag{6-65}$$

where T_{GOP} is the target bit for the current GOP, $CODED_{GOP}$ is coded bits spent for the current GOP, w_j is weighted parameter for the j-th picture, and N is number of pictures in GOP.

The lower bound check is defined by:

$$F(i) - b(i) < L(i) \tag{6-66}$$

If the above relation holds, $b(i)$, is set to $max(200, F(i)\text{-}L(i))$. It should be noted that for underflow control the estimated buffer fullness is set to the actual buffer fullness. This is to guarantee that the buffer does not underflow at any time within the interval whereby picture *i* gets decoded.

6.9 Deblocking in RDO

The VTM encoder has the option to apply deblocking filtering during rate-distortion optimization (RDO) calculations for mode selections to increase compression efficiency in the random access and low delay coding configurations. This feature is controlled by the encoder parameter `EncDbOpt`.

When enabled, the current CU along with the areas affected by deblocking (up to 8 rows or columns) above and to the left are copied to a temporary buffer and the deblocking filter process is applied along the left and top edges of the CU in the temporary buffer. The distortion is then calculated using the deblocked CU. The difference of the distortion before and after applying deblocking of the areas to the left and above are added to the resulting total CU distortion.

The impact of deblocking filtering along the right and bottom edge is not included since those neighbouring CUs are not available when processing the current CU.

6.10 Block Importance Mapping (BIM)

The VTM reference encoder supports a block QP adaptation method called Block Importance Mapping (BIM). The method is enabled by setting the VTM encoder configuration parameter BIM to 1. It is also recommended to set TemporalFilter to 1 and both TemporalFilterPastRefs and TemporalFilterFutureRefs to at least 2. BIM sets delta QP for CTUs depending on an estimation of the difference between pictures and is based on calculations performed in the MCTF. QP is only modified for frames where the temporal filter is active, i.e., where POC is divisible by eight.

The error value *E* is currently computed per 8x8 block for each reference picture in the temporal filter. For BIM, the average *E*-value in a CTU is calculated for the pictures immediately before and after the current picture. These two values are then averaged to form a value *E1* for each CTU in the current picture, and the process is repeated for both pictures at a distance of two, yielding *E2* for every CTU. These values are extrapolated to yield an *E4* value for each CTU:

$$E4 = (E1, E2) + abs(E2 - E1) * 3 \tag{6-67}$$

The *E4* value is designed to take into account how quickly the importance fades. If *E1* indicates high importance, but *E2* indicates low importance, the importance evaporates quickly, and it is not so important to encode the block well. However, if both *E1* and *E2* signal high importance, this indicates that the importance stays for longer and it is more important to code the block well.

For the highest temporal layer that is filtered, where POC is divisible by 8 but not 16, *E4* is modified by

$$\boldsymbol{E4 = 0.6*E4 + 0.4*C_{center}} \tag{6-68}$$

Where C_{center} equals to 45. This sets the importance closer to medium importance to compensate for these pictures being used as a reference for fewer future pictures. Finally, thresholding is performed to decide a delta QP relative to the picture QP for each CTU using the values in Table 6-6.

Table 6-6. BIM *E4* to delta QP thresholds

E4	delta QP
0-14	-2

15-29	-1
30-60	0
61-74	+1
>75	+2

6.11 RPR encoding aspects

6.11.1 RPR functionality testing

RPR can be used to adapt to transmission capability and to improve coding efficiency at low bitrates. Testing of RPR functionality can be configured to switch between the source resolution and one of the following reduced resolutions: 1/2, 2/3, and 4/5, whilst at the same time controlling a QP offset. For each picture in a sequence the resolution and QP offset can be configured individually for luma and chroma.

As an example, the switching of resolution can be set to every Nth picture, where the first N pictures are encoded in a first resolution using a first QP offset for luma and a first QP offset for chroma, followed by N pictures encoded in a second resolution using a second QP offset for luma and a second QP offset for chroma, then N pictures encoded in a third resolution using a third QP offset for luma and a third QP offset for chroma, and finally N pictures encoded in a fourth resolution with a fourth QP offset for luma and a fourth QP offset for chroma. This order of resolutions and QP offset settings repeats until the end of the sequence.

RPR functionality testing can be enabled by setting the encoder configuration parameter RPRFunctionalityTesting to 1.

6.11.2 GOP-based RPR encoder control

Depending on the spatial characteristics and which quantization parameter (QP) is used it can be beneficial to encode pictures in full resolution or in one of the following reduced resolutions 1/2, 2/3 or 4/5 both horizontally and vertically. The encoding resolution is determined by the re-scaling performance of the first picture in each GOP and the base QP. The selected resolution is then used for all pictures in the GOP. The QP used for encoding pictures in lower resolutions is reduced by 6 for 1/2, 4 for 2/3 and 2 for 4/5. The chroma QP is adjusted by the chroma QP slice offsets to have same change in QP as for luma to maintain the same relative ratios of bits for luma and chroma. The method is disabled by default but can be enabled by setting the encoder configuration parameter GOPBasedRPR to 1.

6.11.3 Downscaling of input pictures for reduced resolution coding in RPR

The VTM reference encoder includes functionality for downsampling specific input pictures with downsampling scaling ratios greater than 1 and less than or equal to 2, through the config parameters `ScalingRatioHor` and `ScalingRatioVer`. As shown in Table 6-7, depending on the specific scaling ratio x, one out of four downsampling filter sets is used to generate the downsampled pictures.

Table 6-7. Downsampling filters for different scaling ratios

Scaling ratio	Filter
$1 < x \leq 1.1$	6 tap cos-windowed sinc ratio 1
$1.1 < x \leq 1.35$	12 tap Kaiser(7)-windowed sinc ratio 1.35

$1.35 < x \leq 1.66667$	12 tap cos-windowed sinc ratio 1.5, 0.9π-cutoff
$1.66667 < x \leq 2$	12 tap cos-windowed sinc ratio 2, 0.9π-cutoff

6.11.4 Upscaling of reconstructed pictures to full resolution in RPR

When encoder and decoder configuration parameter UpscaledOutput is equal to 2 reconstructed pictures encoded in a reduced resolution can be upscaled to the original resolution by means of 12-tap filters of 16 phases for luma and 6-tap filters of 32 phases for chroma. Optionally, the VVC 8-tap MC filters for luma and the VVC 4-tap MC filters for chroma or the ECM 12-tap MC filters for luma and the ECM 6-tap MC filters for chroma can be selected to upscale the reconstructed pictures. The optional filter selection can be made using the encoder and decoder configuration parameter UpscaleFilterForDisplay.

6.12 QP control for very smooth blocks

To make sure that very smooth large blocks are encoded with sufficient precision, the QP is reduced to prevent the reduction in subjective quality for such blocks. The granularity of the QP adjustment is 64×64. Smooth blocks are detected by fitting luma source samples to a low order polynomial model in regions of 64×64 samples and then computing the SAD between the source and the predicted samples and comparing the SAD with a threshold. When the distortion is below the threshold the luma QP is reduced according to a linear model. The method is disabled by default but can be enabled by setting the encoder configuration parameter SmoothQPReductionEnable to 1.

6.13 Encoder control for DMVR

On rare occasions, the motion derivation process used by DMVR can result in visible motion discontinuities along the boundary of 16x16 subblocks. This may occur when at least one of the reference blocks used by the bilateral matching process is relatively smooth and when QP is high. For this reason, the VTM encoder supports the following three configuration parameters to restrict the use of DMVR when certain conditions are met:

- DMVREncMvSelect: when this parameter is set to 1, the use of DMVR may be restricted. This is done through penalizing the DMVR mode during RDO for those blocks for which DMVR is considered to be more likely to result in unreliable motion;
- DMVREncMvSelectBaseQpTh: DMVR restriction is only applied if the base QP is equal to or higher than DMVREncMvSelectBaseQpTh. The default value is set to 33.
- DMVREncMvSelectDisableHighestTemporalLayer: for video with frame rate higher than 30Hz, if this parameter is set to 1, then the DMVR restriction is not applied to pictures at the highest temporal layer (regardless of the value of DMVREncMvSelect).

References

[1] B. Bross, J. Chen, S. Liu, and Y.-K. Wang, "Versatile Video Coding (Draft 10)", document JVET-S2001, 19th JVET meeting by teleconference, 22 June – 1 July 2020.

[2] B. Bross, J. Chen, S. Liu, and Y.-K. Wang, "Versatile Video Coding Editorial Refinements on Draft 10", document JVET-T2001, 20th JVET meeting by teleconference, 7 – 16 Oct. 2020.

[3] F. Bossen, B. Bross, T. Ikai, D. Rusanovskyy, G. J. Sullivan, Y.-K. Wang, "VVC operation range extensions (Draft 6)", document JVET-Y2005, 25th JVET meeting by teleconference, 12–21 January 2022.

[4] F. Bossen, X. Li, V. Seregin, K. Sharman, K. Sühring, "JVET common test conditions and software reference configurations for SDR video", document JVET-Y2010, 25th JVET meeting by teleconference, 12–21 January 2022.

[5] A. Segall, E. François, W. Husak, S. Iwamura, D. Rusanovskyy, "VTM and HM common test conditions and evaluation procedures for HDR/WCG video", document JVET-Z2011, 26th JVET meeting by teleconference, 20–29 April 2022.

[6] Y. He, J. Boyce, K. Choi, J.-L. Lin, "JVET common test conditions and evaluation procedures for 360° video", document JVET-U2012, 21st JVET meeting by teleconference, 6 – 15 Jan. 2021.

[7] Y.-H. Chao, Y.-C. Sun, J. Xu, and X. Xu, "JVET common test conditions and software reference configurations for non-4:2:0 colour formats", document JVET-T2013, 20th JVET meeting by teleconference, 7 – 16 Oct. 2020.

[8] T.-C. Ma, A. Nalci, T. Nguyen, "JVET common test conditions and software reference configurations for lossless, near lossless, and mixed lossy/lossless coding", document JVET-Q2014, 17th JVET meeting: Brussels, BE, 7–11 Jan. 2020.

[9] Y. Ye, J. Boyce, "Algorithm descriptions of projection format conversion and video quality metrics in 360Lib (Version 12)," document JVET-T2004, 20th JVET meeting by teleconference, 7 – 16 Oct. 2020.

[10] A. Browne, T. Ikai, D. Rusanovskyy, M. Sarwer, X. Xiu, "Common test conditions for high bit depth and high bit rate video coding", document JVET-U2018, 21st JVET meeting by teleconference, 6 – 15 Jan. 2021.

[11] High Efficiency Video Coding (HEVC), Rec. ITU-T H.265 and ISO/IEC 23008-2, Jan. 2013 (and later editions).

[12] 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.

[13] Supplement ITU-T H.Sup15 (2017), Conversion and Coding Practices for HDR/WCG Y′CbCr 4:2:0 Video with PQ Transfer Characteristics

[14] SMPTE RDD 5:2006 - SMPTE Registered Disclosure Doc - Film Grain Technology — Specifications for H.264 | MPEG-4 AVC Bitstreams, RDD 5:2006, pp. 1–18, Mar. 2006, doi: 10.5594/SMPTE.RDD5.2006.

[15] D. Grois, Y. He, W. Husak, P. de Lagrange, A. Norkin, M. Radosavljević, A. Tourapis, W. Wade "Film grain synthesis technology for video applications (Draft 3)", document JVET-AB2020, 28th JVET Meeting, Mainz, Germany, 10-18 October 2022.

[16] Information technology — MPEG systems technologies — Part 11: Energy-efficient media consumption (green metadata), ISO/IEC 23001-11:19.

[17] ETSI 101 154: Digital Video Broadcasting (DVB); Specification for the use of Video and Audio Coding in Broadcast and Broadband Applications.

[18] ATSC Standard: Video – HEVC, Advanced Television Systems Committee, A/341.