Joint Collaborative Team on 3D Video Coding Extension Development
of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11
6th Meeting: Geneva, CH, 25 Oct. – 1 Nov. 2013

Document:
JCT3V-F1001-v4

Title: **3D-HEVC Draft Text 2**

Status: Output Document of JCT-3V

Purpose: 3D-HEVC working draft

Author(s) or Contact(s):

Gerhard Tech
Fraunhofer HHI
Email: gerhard.tech@hhi.fraunhofer.de

Krzysztof Wegner
Poznan University of Technology
Email: kwegner@multimedia.edu.pl

Ying Chen
Qualcomm Incorporated
Email: cheny@qti.qualcomm.com

Sehoon Yea
LG Electronics
Email: sehoon.yea@lge.com

Source: Editor

ABSTRACT

Draft 2 of 3D-HEVC

- ----------- Release v4 -----------
- Accepted change marks.
- ----------- Release v3 -----------
- (3DC-41) Disabling Sub PU Iv MVP for depth
- (3DC-40) Review GT
- (3DC-39) Review CY
- (3DC-38) Fix related to F0105.
- (3DC-37) Fix #49 Ticket Mismatch in VSP horSplitFlag derivation between WD and HTM
- (3DC-36) Review GT
- (3DC-35/JCT3V-F0123, JCT3V-F0108) Harmonize F0108/F0123 according to the above for inter-view ARP.
- (3DC-34/JCT3V-F0131, JCT3V-F0139, JCT3V-F0138) DLT related: including F0131, F0139 and moving DLT from VPS to PPS.
- (3DC-33) Review CY
- ----------- Release v2 -----------
- (3DC-32) Review GT
- (3DC-31) Review GT
- (3DN-30/JCT3V-F0093) CE3.h: Results on simple merge candidate list construction for 3DV. It was agreed to add the condition on numMergeCand from F0129.Decision: Adopt F0093 with modifications of F0129
- (3DE-29) Cleanup merge list generation 2: Removed VSP flag list
- (3DE-28) Cleanup merge list generation 1: Introduced equal motion function.
- (3DC-27) Fixes related to F1001.
- (3DC-26) Fix vps_inter_sdc_flag.
- (3DC-25) Fix number extra merge candidates F1001.
- ----------- Release v1 -----------
- (3DN-24/JCT3V-F0160) Non-CE: Illumination compensation flag coding ; Decision: Adopt
- (3DN-23/JCT3V-F0151) HLS: Removal of IC in depth coding and IC flag signalling in 3D-HEVC; Decision: Remove IC for depth map coding, no change for texture coding.
- (3DN-22/JCT3V-F0082) HLS: On slice-level camera parameter signaling ;Decision: Adopt the second solution: the cp_in_slice_segment_layer_flag to be view specific and used as a condition of the presence of slice header level camera parameters.
- (3DN-21/JCT3V-F0136) Comments on camera parameters in 3D-HEVC, Decision: Adopt (harmonized F0136/F0045)
- (3DN-20/JCT3V-F0044) HLS: HEVC compatible slice segment header in 3D-HEVC; Decision: Adopt the proposal to move the camera parameters from slice header extension to some place before the slice header extension in slice header under the condition of nuh_layer_id unequal to 0. If changes are to be made, the MVCompatibleFlag should also be part of the condition.
- (3DN-19/JCT3V-F0045) HLS: Constraints on camera parameter signaling; Decision (BF): Add missing brackets in the loop related to the camera parameter signaling.
- (3DN-18/JCT3V-F0105) CE4: ARP reference picture selection and its availability check ;One aspect suggests to use the first temporal reference picture instead of the first entry in each reference picture list (same as F0123). Another aspect proposes to check whether ARP fixed reference picture is in DPB marked as “used for reference”, which is explicitly indicated in reference layer’s RPS. The text was revised from the original proposal to have a slice level check. Decision: Adopt
- (3DE-17) Added General decoding process for prediction units in inter prediction mode
- (3DN-16/JCT3V-F0110) CE3: Sub-PU level inter-view motion prediction. Decision: Adopt, specify 8x8 in CTC
- (3DN-15/JCT3V-F0125) CE3: Inter-view motion vector prediction for depth coding Decision: Adopt F0125.
- (3DN-14/JCT3V-F0111) CE1: Simplified view synthesis prediction Decision: Adopt both simplifications suggested in F0111.
- (3DN-13/JCT3V-F0104) CE3: Removal of redundancy on VSP, ARP and IC Decision: Adopt F0104 (without IC_ARP_DEPEND)Item 2 has already been decided to study in CE as per the discussion in CE4.
- (3DN-12/JCT3V-F0161) CE4: Coding of weighting factor of advanced residual prediction; Decision: Adopt.
- (3DE-11 Ed. five_minus_max_num_merge_cand) Added updated semantics of five_minus_max_num_merge_cand from HEVC version 1.
- (3DN-10/JCT3V-F0150) CE3: MPI candidate in depth merge mode list construction Decision: Adopt (option 1)

- (3DN-09/JCT3V-F0109,JCT3V-F0120) CE1: A simplified block partitioning method for view synthesis prediction Decision: Adopt (F0109/F0120) (Identical)
- (3DN-08/JCT3V-F0102) CE1: VSP partitioning for AMP Decision: Adopt F0102
- (3DN-07/JCT3V-F0115) CE2: Problem fix of the DV derivation in 3D-HEVC Decision: Adopt the suggested solution which aligns the text with the software bug fix.
- (3DN-06/JCT3V-F0149) CE5: Simplified depth inter mode coding; Adoption (BF): Align the text with software as suggested in F0149.
- (3DN-05/JCT3V-F0147) CE5: DMM simplification and signalling Decision: Adopt (remove DMM3 and RBC).
- (3DN-04/JCT3V-F0159) CE5: Fast depth lookup table application method to intra modes for depth data ;Decision: Adopt F0159 method 3. Implement an enabling flag at (position t.b.d.).
- (3DN-03/JCT3V-F0132) CE5: Unification of delta DC coding for depth intra modes.
- (3DN-01/JCT3V-F0171) CE5: Fix for DMM/RBC reference sample filtering.
- (3DC-00 Review GT)

Draft 1 of 3D-HEVC

Ed. Notes (Draft 1) (changes compare to JCT3V-D1005)

- ----------- Release v3 -----------
- Accepted change marks.
- ----------- Release v2 -----------
- (3DC-04) Fix disparity derivation
- (3D-GT6) Review, cleanups.
- (3DC-03) Update definitions.
- (3DC-02/JCT3V-E0159) Removal further unused parts of overlap between DMM1 and DMM3.
- (3DC-01 Fix refIdx for VSP)
- (3DN-22/JCT3V-E0172) Added missing dv-scaling of item 3.
- (3DC-GT5) Revised editors comments. Cleanups. Fixed ticket #41, #42, #43
- (3DE-02) Update to HEVC version 1
- ----------- Release v1 -----------
- (3DN-GT3) Cleanups
- (3DE-CY1): Review and editorial improvements.
- (3DN-23/JCT3V-E0163) Camera parameter presence indication.
- (3DN-22/JCT3V-E0172/Items 3+4) CE2: VSP Fix
- (3DC-GT2) Fix tickets #35, #30, #32, #33, #34, #37
- (3DN-21/JCT3V-E0126) CE3: Merge candidates derivation from vector shifting.
- (3DN-20/JCT3V-E0142,JCT3V-E0190) CE2: Simplified NBDV and improved disparity vector derivation
- (3DN-19/JCT3V-E0207) + JCT3V-E0208 CE1: Adaptive block partitioning for VSP and clipping.
- (3DN-18/JCT3V-E0141) CE2: Clipping in depth-based disparity vector derivation
- (3DN-17/JCT3V-E0156) CE6: Simplified Inter Mode Coding of Depth Decision
- (3DE-01) Added Decoding process for the residual signal of coding units coded in inter predmode from base spec
- (3DN-16/JCT3V-E0034) HLS: Revision of the Alternative Depth Info SEI message
- (3DN-15/JCT3V-E0160) HLS: Make 3D-HEVC Compatible with MV-HEVC Adopt (solution 2)
- (3DN-14/JCT3V-E0134) HLS: Signalling of camera parameters.
- (3DN-13/JCT3V-E0057) HLS: On parameter sets. Adopt View Id aspect
- (3DN-12/JCT3V-E0104) HLS: Only portion that swaps multiview and depth flag in scalability dimension
- (3DN-11/JCT3V-E0182) CE3: A bug-fix for the texture merging candidate
- (3DC-GT1) Review and editorial improvements
- (3DN-10/JCT3V-E0172/Item 5) CE2: Disparity inter-view motion vector derivation
- (3DN-10/JCT3V-E0172/Item 7) CE2: DVMCP Fix
- (3DN-09/JCT3V-E0170) CE3: Motion data buffer reduction for 3D-HEVC Decision: Adopt (first scheme)
- (3DN-08/JCT3V-E0117) CE6: Simplified DC calculation for SDC
- (3DN-07/JCT3V-E0242) CE5: On DMM simplification
- (3DN-06/JCT3V-E0204) CE5: Simplified Binarization for depth_intra_mode
- (3DN-05/JCT3V-E0159) CE5: Removal of Overlap between DMM1 and DMM3
- (3DN-04/JCT3V-E0158) CE6: Removal of DC from SDC Mode
- (3DN-03/JCT3V-E0146) CE5: DMM simplification and signalling. Remove DMM2.
- (3DN-02/JCT3V-E0168) CE4: Complexity reduction of bi-prediction for illumination compensation
- (3DN-01/JCT3V-E0046) CE4: Resampling in IC parameter derivation and 4x4 Chroma removal

Decision: Adopt: JCT3V-E0046

Ed. Notes (TM4) (changes compare to JCT3V-C1005):

- ----------- Release v4 -----------
- Accepted all change marks.
- ----------- Release v3 -----------
- (3DC-GT4) Review and editorial improvements
- ----------- Release v2 -----------
- (3DN-22/JCT3V-D0166) On reference view selection in NBDV and VSP
- (3DC-01 availableFlagDV) Fixed availability flag for disparity vector.
- (3DC-GT3) Review and editorial improvements
- (3DC-CY1) Review and editorial improvements
- ----------- Release v1 -----------
- (3DC-GT2) Clean-up of variables not used any more (related to disparity derivation)
- (3DN-21/D0220/ViewId) ViewId not reflecting coding order any more.
- (3DN-20/JCT3V-D0272) Signaling Global View and Depth
- (3DN-19/JCT3V-D0103) Signaling Warp Maps as an Alternative 3D Format
- (3DN-18/JCT3V-D0032/JCT3V-D0141/JCT3V-D0034) SDC Residual CABAC contexts.
- (3DN-17/JCT3V-D0035) DLT for DMM deltaDC coding
- (3DN-16/JCT3V-D0195) Unification of new intra modes in 3D-HEVC
- (3DN-15/JCT3V-D0193) Clean-up for 64x64 SDC
- (3DN-14/JCT3V-D0183) Simplified DC predictor for depth intra modes
- (3DN-13/JCT3V-D0110) Sample-based simplified depth coding.
- (3DN-12/JCT3V-D0060) Removal of parsing dependency for illumination compensation
- (3DN-11/JCT3V-D0122) AMVP candidate list construction
- (3DN-10/JCT3V-D0091) Inter-view SAO process in 3DV coding
- (3DN-09/JCT3V-D0177) Advanced residual prediction for multiview coding
- (3DN-08/JCT3V-D0138) Simplified DV derivation for DoNBDV and BVSP
- (3DN-07/JCT3V-D0112) Default disparity vector derivation
- (3DN-06/JCT3V-D0105) BVSP NBDV
- (3DN-05/JCT3V-D0191) Clean-ups for BVSP in 3D-HEVC.
- (3DN-04/JCT3V-D0092) CE1.h related: BVSP mode inheritance
- (Incorporated 8.5.2.1.3 from base spec) Derivation process for combined bi-predictive merging candidates.
- (3DN-03/JCT3V-D0181) CE2.h related: CU-based Disparity Vector Derivation
- (Incorporated 8.5 from base spec)
- (3DN-02/JCT3V-D0135) CE5: Unification of disparity vector rounding
- (3DN-01/JCT3V-D0156): HLS for stereo compatibility.(Also covers disabling of VSP for depth as proposed in D0105 and D0139).
- (3DC-GT1) Editorial improvements, small corrections.

Ed. Notes (TM3) (changes compare to JCT3V-B1005):

- (3Dc-04) Revised text related to edge intra.
- (3DC-GT2) Editorial improvements, small corrections.(among others tickets #21 #22)
- ----------- Release d0 -----------
- Converted to .doc- File
- Split of Test Model text and specification text
- (3DE-05) Alignment with MV-HEVC draft 3.
- (3DE-01) Reordered sub-clauses related to disparity estimation and additional motion candidates.
- (3DN-20) Alignment of JCT3V-C0152 + JCT3V-C0137.
- (3DN-07/JCT3V-C0137) Texture motion vector candidate for depth.
- (3DN-07/JCT3V-C0137) Removal of MPI.
- (3DN-19) Camera parameters
- (3Dn−03) Wedgelet pattern generation process.
- (3Dn-01) Incorporated missing intra-predicted wedgelet partition mode
- (3DN-08/JCT3V-C0138) Removal of parsing dependency for inter-view residual prediction.
- (3DN-18/JCT3V-C0160) QTL disabled for RAP.
- (3DN-17/JCT3V-C0154) Reference sub-sampling for SDC and DMM.
- (3Dc-03) Fix SDC
- (3DN-16/JCT3V-C0096) Removal of DMM 2 from SDC.
- (3DN-15/JCT3V-C0034) Delta DC processing for DMMs.
- (3DN-14/JCT3V-C0044) Signalling of wedgeIdx for DMM3.
- (3DN-02/JCT3V-C0152) View synthesis prediction (without disparity derivation part).

- (3DN-03/JCT3V-C0112) Restricted search of max disparity.
- (3DN-01,02/JCT3V-C0131,JCT3V-C0152) Disparity derivation from depth maps.
- (3Dc-02) Incorporated missing conditions of long/short-term pictures in AMVP (related to JCT3V-B0046).
- (3DN-13/JCT3V-C0116) Inter-view vector scaling for AMVP.
- (3DE-03) Incorporated derivation process for AMVP from base spec.
- (3DN-12/JCT3V-C0115) Signalling of inter-view motion vector scaling.
- (3DE-02) Incorporated TMVP text from base spec.
- (3DN-09/JCT3V-C0047) Alternative reference index for TMVP.
- (3DN-10/JCT3V-C0051) Unification of inter-view candidate derivation.
- (3DE-01) Revised text related to residual prediction.
- (3DN-06/JCT3V-C0129) Vertical component in residual prediction.
- (3DN-05/JCT3V-C0097/JCT3V-C0141) Temporal blocks first in DV derivation.
- (3DC-GT1) Editorial improvements, small corrections.
- (3DN-04/JCT3V-C0135) Restriction on the temporal blocks for memory bandwidth reduction in DV derivation.
- (3Dn-02) Full sample MV accuracy for depth.
- (3DN-11/JCT3V-C0046) Extension of illumination compensation to depth.
- (3Dc-01) Fix Illumination compensation (including ic_flag for skip).

Ed. Notes (TM2) (changes compared to JCT3V-A1005)

- Accepted changes and marked delta to base spec
- (3DC-GT2) Editorial improvements, small corrections
- (3DC-CY) Editorial improvements, small corrections
- (MVS-02/JCT3V-B0046) Treatment of inter-view pictures as long term- reference pictures
- (3DE-11) Revised text related to 3Dn-01
- (3Dn-01/m23639) Results on motion parameter prediction
- (3DE-12) Revised text related residual prediction
- (3DE-10) Revised text Related to Illumination compensation.
- (3DN-01/JCT3V-B0045) Illumination compensation for inter-view prediction.
- (3Dn-02/m24766) Restricted Inter-View Residual Prediction
- (3DE-09) Revised text related to depth intra: Edge Intra
- (3DE-09) Revised text related to depth intra: SDC
- (3DE-09) Revised text related to depth intra: DMMs
- (3DO-01/JCT3V-B0131) Depth distortion metric with a weighted depth fidelity term
- (3DN-12/JCT3V-B0036) Simplified Depth Coding with an optional Depth LUT
- (3DN-13/JCT3V-B0039) Simplified Wedgelet search for DMM modes 1 and 3
- (3DN-03/JCT3V-B0083) Unconstrained motion parameter inheritance
- (3DE-08) Incorporated context tables for SDC
- (3DE-07) Improved MPI text.
- (3DN-02/JCT3V-B0068) Incorporated Depth Quadtree Prediction.
- (3DE-06) Incorporated parsing process, including tables for DMMs.
- (3DE-05) Added missing initialization of invalid motion/disparity parameters
- (3DC-03) Added missing pruning of collocated merge candidate due to number of total candidates.
- (3DE-04) Moved pruning of spatial merge candidate B2 due to number of total candidates.
- (3DE-03) Moved derivation of disparity one level higher in process hierarchy.
- (3DE-02) Inserted "Derivation process for motion vector components and reference indices" from base spec
- (3DC-02) Fixed storage of IvpMvFlagLX and IvpMvDisp.
- (3DN-09-10-11/JCT3V-B0048,B0069,B0086) Modification inter-view merge candidates
- (3DC-01) Fixed derivation of inter-view merge candidates.
- (3DE-01) Revised derivation of disparity from temporal candidates
- (3DN-04/JCT3V-B0047) Improvements for disparity vector derivation)
- (3DN-08/JCT3V-B0136) Support of parallel merge in disparity vector derivation
- (3DN-05/JCT3V-B0135) Modified disparity vector derivation process for memory reduction
- (3DN-04/JCT3V-B0111) Decoupling inter-view candidate for AMVP
- (3DN-07/JCT3V-B0096) Removal of dependency between multiple PUs in a CU for DV-derivation
- (3DC-GT) Small corrections, editorial improvements

Ed. Notes (TM1) (changes compare to N12744)

- (3D08/JCT3V-A0126) (T,N) Simplified disparity derivation
- (3D16) Moved 3D-tool related flags from SPS to VPS, removal camera parameters
- (3D09/JCT3V-A0049) (N) Inter-view motion prediction modification
- (3D13/JCT3V-A0119) (T) VSO depth fidelity
- (3D07/JCT3V-A0070) (T,N) Region boundary chain coding for depth maps
- (3D06/JCT3V-A0087) (T) RDO selection between Non-Zero Residual and All-Zero Residual Intra
- (3D12) (T) Depth Quadtree Prediction
- (3D15) (N) Fix references
- (3D11) (T,N) Improvement of text of already adopted tools
- (3D10/JCT3V-A0097) (T;N) Disparity vector generation
- (3D02) (N) Removed MV-Part and update to Annex F
- (3D03) (T) Labelling of tools not in CTC/Software. Removal?
- (3D05/JCT3V-A0093) (T) VSO early skip
- (3D04/JCT3V-A0033) (T) VSO model based estimation
- (3D14) (N) Update of low level specification to match HEVC text specification 8(d7)
- (3D01) (N): Removed HEVC text specification

CONTENTS

Abstract 2

Contents 7

List of Figures 8

List of Tables 8

Annex H 3D High Efficiency Video Coding 10
- H.1 Scope 10
- H.2 Normative references 10
- H.3 Definitions 10
- H.4 Abbreviations 10
- H.5 Conventions 11
- H.6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships 11
- H.7 Syntax and semantics 11
  - H.7.1 Method of specifying syntax in tabular form 11
  - H.7.2 Specification of syntax functions, categories, and descriptors 11
  - H.7.3 Syntax in tabular form 11
    - H.7.3.1 NAL unit syntax 11
    - H.7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax 11
    - H.7.3.3 Profile, tier and level syntax 15
    - H.7.3.4 Scaling list data syntax 16
    - H.7.3.5 Supplemental enhancement information message syntax 16
    - H.7.3.6 Slice segment header syntax 17
    - H.7.3.7 Short-term reference picture set syntax 20
    - H.7.3.8 Slice segment data syntax 20
  - H.7.4 Semantics 25
    - H.7.4.1 General 25
    - H.7.4.2 NAL unit semantics 25
    - H.7.4.3 Raw byte sequence payloads, trailing bits, and byte alignment semantics 25
    - H.7.4.4 Profile, tier and level semantics 30
    - H.7.4.5 Scaling list 30
    - H.7.4.6 Supplemental enhancement information message semantics 30
    - H.7.4.7 Slice segment header semantics 30
    - H.7.4.8 Short-term reference picture set semantics 31
    - H.7.4.9 Slice data semantics 31
- H.8 Decoding process 35
  - H.8.1 General decoding process 35
    - H.8.1.1 Decoding process for a coded picture with nuh_layer_id greater than 0 35
  - H.8.2 NAL unit decoding process 35
  - H.8.3 Slice decoding process 35
    - H.8.3.1 Decoding process for picture order count 35
    - H.8.3.2 Decoding process for reference picture set 35
    - H.8.3.3 Decoding process for generating unavailable reference pictures 35
    - H.8.3.4 Decoding process for reference picture lists construction 36
    - H.8.3.5 Derivation process for the candidate picture list for disparity vector derivation 36
    - H.8.3.6 Decoding process for a depth lookup table 36
    - H.8.3.7 Derivation process for the alternative target reference index for TMVP in merge mode 37
    - H.8.3.8 Derivation process for the default reference view order index for disparity derivation 37
    - H.8.3.9 Derivation process for the target reference index for residual prediction 37
  - H.8.4 Decoding process for coding units coded in intra prediction mode 38
    - H.8.4.1 General decoding process for coding units coded in intra prediction mode 38
    - H.8.4.2 Derivation process for luma intra prediction mode 38
    - H.8.4.3 Derivation process for chroma intra prediction mode 40
    - H.8.4.4 Decoding process for intra blocks 40
  - H.8.5 Decoding process for coding units coded in inter prediction mode 49
    - H.8.5.1 General decoding process for coding units coded in inter prediction mode 49
    - H.8.5.2 Inter prediction process 50
    - H.8.5.3 Decoding process for prediction units in inter prediction mode 50
    - H.8.5.4 Decoding process for the residual signal of coding units coded in inter prediction mode 78
    - H.8.5.5 Derivation process for disparity vectors 80
    - H.8.5.6 Derivation process for disparity vectors from neighbouring depth samples 83

H.8.6 Scaling, transformation and array construction process prior to deblocking filter process 84
H.8.7 In-loop filter process 84
H.9 Parsing process 84
H.9.1 General 84
H.9.2 Parsing process for 0-th order Exp-Golomb codes 84
H.9.2.1 General 84
H.9.2.2 Mapping process for signed Exp-Golomb codes 84
H.9.3 CABAC parsing process for slice segment data 84
H.9.3.1 General 84
H.9.3.2 Initialization process 85
H.9.3.3 Binarization process 87
H.9.3.4 Decoding process flow 89
H.9.3.5 Arithmetic encoding process (informative) 90
H.10 Sub-bitstream extraction process 91
H.11 Profiles and levels 91
H.12 Byte stream format 91
H.13 Hypothetical reference decoder 91
H.14 Supplemental enhancement information 91
H.14.1 General 91
H.14.2 SEI payload syntax 91
H.14.2.1 Alternative depth information SEI message syntax 91
H.14.3 SEI payload semantics 92
H.14.3.1 Alternative depth information SEI message semantics 92
H.15 Video usability information 95

LIST OF FIGURES

Figure H-1 Fractional sample position dependent variables in bi-linear interpolation and surrounding integer position samples A, B, C, and D 74

Figure H-2 Relation between camera ID and GVD texture/depth and packing of texture/depth views to the base and residual texture/depth layer 95

LIST OF TABLES

Table H-1 – Mapping of ScalabiltyId to scalability dimensions 25

Table H-2 – Specification of DepthIntraMode and associated name depending on depthIntraModeSet and depth_intra_mode and specification of and depthIntraModeMaxLen depending on depthIntraModeSet 33

Table H-3 – Specification of intra prediction mode and associated names 38

Table H−4 – Specification of resShift 45

Table H−5 – Specification of xPosS, yPosS, xPosE, yPosE, xIncS, yIncS, xIncE, yIncE 45

Table H−6 – Specification of xS, yS, xE, yE 46

Table H−7 Specification of xOff, yOff 47

Table H-8 – Specification of divCoeff depending on sDenomDiv 72

Table H-9 – Specification of the variables xOff, yOff, and interSdcResiIdx[i] 79

Table H-10 – Association of ctxIdx and syntax elements for each initializationType in the initialization process 85

Table H-11 – Values of initValue for wedge_full_tab_idx ctxIdx 86

Table H-12 – Values of initValue for depth_dc_abs ctxIdx 86

Table H-13 – Values of initValue for iv_res_pred_weight_idx ctxIdx 86

Table H-14 – Values of initValue for ic_flag ctxIdx 86

Table H-15 – Values of initValue for depth_intra_mode ctxIdx 86

Table H-16 – Values of initValue for depth_dc_flag ctxIdx 87

Table H-17 – Values of initValue for inter_sdc_flag ctxIdx 87

Table H-18 – Values of initValue for inter_sdc_resi_abs_minus1 ctxIdx 87

Table H-19 – Values of initValue for inter_sdc_resi_sign_flag ctxIdx 87

Table H-20 – Syntax elements and associated binarizations 88

Table H-21 –Values of wedgeFullTabIdxBits[log2PUSize] 88

Table H-22 –Assignment of ctxInc to syntax elements with context coded bins 90

Table H-23 – Specification of ctxInc using left and above syntax elements 90

Table H-24 – Interpretation of depth_type 93

Table H-25– Top-left corner co-ordinates of GVD sub-residual views packed in a residual layer of width = W and height=H. 95

3D High Efficiency Video Coding Draft Specification

The specifications made in this section are based on HEVC version 1 and MV-HEVC draft 6 (JCT3V-F1004).

In text blocks copied from HEVC version 1 or MV-HEVC draft 6 changes are highlighted.

Modifications of HEVC specification:

In Foreword replace paragraph that start with "In this Recommendation | International Standard Annexes":

In this Recommendation | International Standard Annexes A through H contain normative requirements and are an integral part of this Recommendation | International Standard.

In 0.7, add the following paragraph after the paragraph that starts with "Annex F":

Annex H specifies multiview and depth video coding, referred to as 3D High Efficiency Video Coding (3D-HEVC). The reader is referred to Annex H for the entire decoding process for 3D-HEVC, which is specified there with references being made to clauses 1-9 and Annexes A-G.

Annex H 3D High Efficiency Video Coding

[Ed. (GT) Annex character and title formatting need to be updated.]

This annex specifies 3D high efficiency video coding, referred to as 3D-HEVC.

H.1 Scope

Bitstreams and decoders conforming to the profile specified in this annex are completely specified in this annex with reference made to clauses 2-9 and Annexes A-G.

[Ed. (GT): Some references to Annex F might be replaced to references to Annex G and vice versa. This should be fixed when MV-HEVC structure is finalized. When a referenced subclause does not exist in Annex G, the corresponding subclause in Annex F is valid or vice versa.]

H.2 Normative references

The specifications in clause 2 apply.

H.3 Definitions

For the purpose of this annex, the following definitions apply in addition to the definitions in clause F.3 and G.3. These definitions are either not present in clause F.3 and G.3 or replace definitions in clause F.3 and G.3.

H.3.1 **depth view:** A sequence of pictures associated with the same value of ViewOrderIdx and DepthFlag equal to 1.

H.3.2 **depth view component:** A *coded representation* of a the depth view.

H.3.3 **texture view:** A sequence of pictures associated with the same value of ViewOrderIdx and DepthFlag equal to 0.

H.3.4 **texture view component:** A *coded representation* of a the texture view.

H.3.5 **view component:** A coded *representation* of a *view* that may contain a *depth view component* and a *texture view component*.

H.4 Abbreviations

The specification in clause 4 apply.

H.5 Conventions

The specification in clause 5 apply.

H.6 Source, coded, decoded and output data formats, scanning processes, and neighbouring relationships

The specification in clause 6 apply.

H.7 Syntax and semantics

This clause specifies syntax and semantics for coded video sequences that conform to one or more of the profiles specified in this annex.

H.7.1 Method of specifying syntax in tabular form

The specifications in subclause 7.1 apply.

H.7.2 Specification of syntax functions, categories, and descriptors

The specifications in subclause 7.2 apply.

H.7.3 Syntax in tabular form

H.7.3.1 NAL unit syntax

The specifications in subclause G.7.3.1 and all its subclauses apply.

H.7.3.2 Raw byte sequence payloads and RBSP trailing bits syntax

H.7.3.2.1 Video parameter set RBSP

The specifications in subclause G.7.3.2.1 apply.

[Ed. (GT): Inclusion of potential VPS extension 3 needs to be specified.]

H.7.3.2.1.1 Video parameter set extension syntax

The specifications in subclause G.7.3.2.1.1 apply.

H.7.3.2.1.2 Video parameter set extension 2 syntax

vps_extension2( ) {	**Descriptor**
while(!byte_aligned( ))	
vps_extension_byte_alignment_reserved_one_bit	u(1)
for(i = 0; i <= vps_max_layers_minus1; i++) {	
layerId = layer_id_in_nuh[i]	
if (layerId ! = 0) {	
iv_mv_pred_flag[layerId]	u(1)
log2_sub_pb_size_minus2[layerId]	ue(v)
if (!VpsDepthFlag[layerId]) {	
iv_res_pred_flag[layerId]	u(1)
depth_refinement_flag[layerId]	u(1)
view_synthesis_pred_flag[layerId]	u(1)
} else {	
mpi_flag[layerId]	u(1)
vps_depth_modes_flag[layerId]	u(1)
lim_qt_pred_flag[layerId]	u(1)
vps_inter_sdc_flag[layerId]	u(1)
}	
}	
}	
cp_precision	ue(v)
for(i = 0; i < NumViews; i++) {	
cp_present_flag[i]	u(1)
if(cp_present_flag[i]) {	
cp_in_slice_segment_header_flag[i]	u(1)
if(!cp_in_slice_segment_header_flag[i])	
for(j = 0; j < i; j++) {	
vps_cp_scale[i][j]	se(v)
vps_cp_off[i][j]	se(v)
vps_cp_inv_scale_plus_scale[i][j]	se(v)
vps_cp_inv_off_plus_off[i][j]	se(v)
}	
}	
}	
iv_mv_scaling_flag	u(1)
}	

H.7.3.2.2 Sequence parameter set RBSP syntax

The specifications in subclause G.7.3.2.2 apply.

[Ed. (GT): Inclusion of a potential further SPS extension needs to be specified.]

H.7.3.2.2.1 Sequence parameter set extension syntax

The specifications in subclause G.7.3.2.2.1 apply.

H.7.3.2.3 Picture parameter set RBSP syntax

pic_parameter_set_rbsp( ) {	**Descriptor**
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	ue(v)
dependent_slice_segments_enabled_flag	u(1)
output_flag_present_flag	u(1)
num_extra_slice_header_bits	u(3)
sign_data_hiding_flag	u(1)
cabac_init_present_flag	u(1)
num_ref_idx_l0_default_active_minus1	ue(v)
num_ref_idx_l1_default_active_minus1	ue(v)
init_qp_minus26	se(v)
constrained_intra_pred_flag	u(1)
transform_skip_enabled_flag	u(1)
cu_qp_delta_enabled_flag	u(1)
if (cu_qp_delta_enabled_flag)	
diff_cu_qp_delta_depth	ue(v)
pps_cb_qp_offset	se(v)
pps_cr_qp_offset	se(v)
pps_slice_chroma_qp_offsets_present_flag	u(1)
weighted_pred_flag	u(1)
weighted_bipred_flag	u(1)
transquant_bypass_enabled_flag	u(1)
tiles_enabled_flag	u(1)
entropy_coding_sync_enabled_flag	u(1)
if(tiles_enabled_flag) {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
uniform_spacing_flag	u(1)
if(!uniform_spacing_flag) {	
for(i = 0; i < num_tile_columns_minus1; i++)	
column_width_minus1[i]	ue(v)
for(i = 0; i < num_tile_rows_minus1; i++)	
row_height_minus1[i]	ue(v)
}	
loop_filter_across_tiles_enabled_flag	u(1)
}	
loop_filter_across_slices_enabled_flag	u(1)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_disable_deblocking_filter_flag	u(1)
if(!pps_disable_deblocking_filter_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
}	
}	
pps_scaling_list_data_present_flag	u(1)
if(pps_scaling_list_data_present_flag)	

scaling_list_data( )	
lists_modification_present_flag	u(1)
log2_parallel_merge_level_minus2	ue(v)
slice_segment_header_extension_present_flag	u(1)
pps_extension_flag	u(1)
if(pps_extension_flag) {	
pps_extension( )	
pps_extension2_flag	u(1)
if(pps_extension2_flag)	
while(more_rbsp_data( ))	
pps_extension_data_flag	u(1)
}	
rbsp_trailing_bits( )	
}	

H.7.3.2.3.1 Picture parameter set extension syntax

pps_extension( ) {	Descriptor
dlt_present_flag	u(1)
if(dlt_present_flag) {	
pps_depth_layers_minus1	u(6)
pps_bit_depth_for_depth_views_minus8	u(4)
for(i=0; i <= pps_depth_layers_minus1; i++) {	
dlt_flag[i]	u(1)
if(dlt_flag[i]) {	
inter_view_dlt_pred_enable_flag[i]	u(1)
if(!inter_view_dlt_pred_enable_flag[i])	
dlt_bit_map_rep_flag[i]	u(1)
if(dlt_bit_map_rep_flag[i])	
for(j = 0; j <= depthMaxValue; j++)	
dlt_bit_map_flag[i][j]	u(1)
else	
entry_table(i)	
}	
}	
}	
}	

H.7.3.2.3.2 Entry table syntax

entry_table(i) {	Descriptor
num_entry	u(v)
if(num_entry > 0) {	
if(num_entry > 1)	
max_diff	u(v)
if(num_entry > 2)	
min_diff_minus1	u(v)
entry0	u(v)

if(max_diff > (min_diff_minus1 + 1))	
for(k = 1; k < num_entry; k++)	
entry_value_diff_minus_min[k]	u(v)
}	
}	

H.7.3.2.4 Supplemental enhancement information RBSP syntax

The specifications in subclause G.7.3.2.4 apply.

H.7.3.2.5 Access unit delimiter RBSP syntax

The specifications in subclause G.7.3.2.5 apply.

H.7.3.2.6 End of sequence RBSP syntax

The specifications in subclause G.7.3.2.6 apply.

H.7.3.2.7 End of bitstream RBSP syntax

The specifications in subclause G.7.3.2.7 apply.

H.7.3.2.8 Filler data RBSP syntax

The specifications in subclause G.7.3.2.8 apply.

H.7.3.2.9 Slice layer RBSP syntax

The specifications in subclause G.7.3.2.9 apply.

H.7.3.2.10 RBSP slice trailing bits syntax

The specifications in subclause G.7.3.2.10 apply.

H.7.3.2.11 RBSP trailing bits syntax

The specifications in subclause G.7.3.2.11 apply.

H.7.3.2.12 Byte alignment syntax

The specifications in subclause G.7.3.2.12 apply.

H.7.3.3 Profile, tier and level syntax

The specifications in subclause G.7.3.3 apply.

H.7.3.4 Scaling list data syntax

The specifications in subclause G.7.3.4 apply.

H.7.3.5 Supplemental enhancement information message syntax

The specifications in subclause G.7.3.5 apply.

H.7.3.6 Slice segment header syntax

H.7.3.6.1 General slice segment header syntax

slice_segment_header( ) {	**Descriptor**
first_slice_segment_in_pic_flag	u(1)
if(nal_unit_type >= BLA_W_LP && nal_unit_type <= RSV_IRAP_VCL23)	
no_output_of_prior_pics_flag	u(1)
slice_pic_parameter_set_id	ue(v)
if(!first_slice_segment_in_pic_flag) {	
if(dependent_slice_segments_enabled_flag)	
dependent_slice_segment_flag	u(1)
slice_segment_address	u(v)
}	
if(!dependent_slice_segment_flag) {	
i = 0	
if(num_extra_slice_header_bits > i) {	
i++	
poc_reset_flag	u(1)
}	
if(num_extra_slice_header_bits > i) {	
i++	
discardable_flag	u(1)
}	
for(~~i = 1~~; i < num_extra_slice_header_bits; i++)	
slice_reserved_flag[i]	u(1)
slice_type	ue(v)
if(output_flag_present_flag)	
pic_output_flag	u(1)
if(separate_colour_plane_flag = = 1)	
colour_plane_id	u(2)
if(nuh_layer_id > 0 | | (nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP)) {	
slice_pic_order_cnt_lsb	u(v)
if(nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP) {	
short_term_ref_pic_set_sps_flag	u(1)
if(!short_term_ref_pic_set_sps_flag)	
short_term_ref_pic_set(num_short_term_ref_pic_sets)	
else if(num_short_term_ref_pic_sets > 1)	
short_term_ref_pic_set_idx	u(v)
if(long_term_ref_pics_present_flag) {	
if(num_long_term_ref_pics_sps > 0)	
num_long_term_sps	ue(v)
num_long_term_pics	ue(v)
for(i = 0; i < num_long_term_sps + num_long_term_pics; i++) {	
if(i < num_long_term_sps) {	
if(num_long_term_ref_pics_sps > 1)	
lt_idx_sps[i]	u(v)
} else {	
poc_lsb_lt[i]	u(v)
used_by_curr_pic_lt_flag[i]	u(1)

}	
delta_poc_msb_present_flag[i]	u(1)
if(delta_poc_msb_present_flag[i])	
delta_poc_msb_cycle_lt[i]	ue(v)
}	
}	
if(sps_temporal_mvp_enabled_flag)	
slice_temporal_mvp_enabled_flag	u(1)
}	
if(nuh_layer_id > 0 && all_ref_layers_active_flag && NumDirectRefLayers[nuh_layer_id] > 0) {	
inter_layer_pred_enabled_flag	u(1)
if(inter_layer_pred_enabled_flag && NumDirectRefLayers[nuh_layer_id] > 1) {	
if(!max_one_active_ref_layer_flag)	
num_inter_layer_ref_pics_minus1	u(v)
if(NumActiveRefLayerPics != NumDirectRefLayers[nuh_layer_id])	
for(i = 0; i < NumActiveRefLayerPics; i++)	
inter_layer_pred_layer_idc[i]	u(v)
}	
}	
if(sample_adaptive_offset_enabled_flag) {	
slice_sao_luma_flag	u(1)
slice_sao_chroma_flag	u(1)
}	
if(slice_type = = P || slice_type = = B) {	
num_ref_idx_active_override_flag	u(1)
if(num_ref_idx_active_override_flag) {	
num_ref_idx_l0_active_minus1	ue(v)
if(slice_type = = B)	
num_ref_idx_l1_active_minus1	ue(v)
}	
if(lists_modification_present_flag && NumPicTotalCurr > 1)	
ref_pic_lists_modification( )	
if(slice_type = = B)	
mvd_l1_zero_flag	u(1)
if(cabac_init_present_flag)	
cabac_init_flag	u(1)
if(slice_temporal_mvp_enabled_flag) {	
if(slice_type = = B)	
collocated_from_l0_flag	u(1)
if((collocated_from_l0_flag && num_ref_idx_l0_active_minus1 > 0) || (!collocated_from_l0_flag && num_ref_idx_l1_active_minus1 > 0))	
collocated_ref_idx	ue(v)
}	
if((weighted_pred_flag && slice_type = = P) || (weighted_bipred_flag && slice_type = = B))	
pred_weight_table( )	
else if(nuh_layer_id > 0 && !DepthFlag && !MvHevcCompatibilityFlag) {	
slice_ic_enable_flag	u(1)
if(slice_ic_enable_flag **)**	

slice_ic_disable_merge_zero_idx_flag	u(1)
}	
five_minus_max_num_merge_cand	ue(v)
}	
slice_qp_delta	se(v)
if(pps_slice_chroma_qp_offsets_present_flag) {	
slice_cb_qp_offset	se(v)
slice_cr_qp_offset	se(v)
}	
if(deblocking_filter_override_enabled_flag)	
deblocking_filter_override_flag	u(1)
if(deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
}	
}	
if(pps_loop_filter_across_slices_enabled_flag && (slice_sao_luma_flag || slice_sao_chroma_flag || !slice_deblocking_filter_disabled_flag))	
slice_loop_filter_across_slices_enabled_flag	u(1)
}	
if(tiles_enabled_flag || entropy_coding_sync_enabled_flag) {	
num_entry_point_offsets	ue(v)
if(num_entry_point_offsets > 0) {	
offset_len_minus1	ue(v)
for(i = 0; i < num_entry_point_offsets; i++)	
entry_point_offset_minus1[i]	u(v)
}	
}	
if(nuh_layer_id > 0 && cp_in_slice_segment_header_flag[ViewIdx])	
for (j = 0; j < ViewIdx; j++) {	
cp_scale[j]	se(v)
cp_off[j]	se(v)
cp_inv_scale_plus_scale[j]	se(v)
cp_inv_off_plus_off[j]	se(v)
}	
if(slice_segment_header_extension_present_flag) {	
slice_segment_header_extension_length	ue(v)
slice_segment_header_extension( )	u(1)
for(i = 0; i < slice_segment_header_extension_length; i++)	
slice_segment_header_extension_data_byte[i]	u(8)
}	
byte_alignment( )	
}	

H.7.3.6.2 Reference picture list modification syntax

The specifications in subclause 7.3.6.2 apply.

H.7.3.6.3 Weighted prediction parameters syntax

The specifications in subclause 7.3.6.3 apply.

H.7.3.7 Short-term reference picture set syntax

The specifications in subclause 7.3.5.2 apply.

H.7.3.8 Slice segment data syntax

The specifications in subclause 7.3.8 apply.

H.7.3.8.1 General slice segment data syntax

The specifications in subclause 7.3.8.1 apply.

H.7.3.8.2 Coding tree unit syntax

The specifications in subclause 7.3.8.2 apply.

H.7.3.8.3 Sample adaptive offset syntax

The specifications in subclause 7.3.8.3 apply.

H.7.3.8.4 Coding quadtree syntax

coding_quadtree(x0, y0, log2CbSize, cqtDepth) {	**Descriptor**
if(x0 + (1 << log2CbSize) <= pic_width_in_luma_samples && y0 + (1 << log2CbSize) <= pic_height_in_luma_samples && log2CbSize > MinCbLog2SizeY && !predSplitCuFlag)	
split_cu_flag[x0][y0]	ae(v)
if(cu_qp_delta_enabled_flag && log2CbSize >= Log2MinCuQpDeltaSize) {	
IsCuQpDeltaCoded = 0	
CuQpDeltaVal = 0	
}	
if(split_cu_flag[x0][y0]) {	
x1 = x0 + (1 << (log2CbSize − 1))	
y1 = y0 + (1 << (log2CbSize − 1))	
coding_quadtree(x0, y0, log2CbSize − 1, cqtDepth + 1)	
if(x1 < pic_width_in_luma_samples)	
coding_quadtree(x1, y0, log2CbSize − 1, cqtDepth + 1)	
if(y1 < pic_height_in_luma_samples)	
coding_quadtree(x0, y1, log2CbSize − 1, cqtDepth + 1)	
if(x1 < pic_width_in_luma_samples && y1 < pic_height_in_luma_samples)	
coding_quadtree(x1, y1, log2CbSize − 1, cqtDepth + 1)	
} else	
coding_unit(x0, y0, log2CbSize, cqtDepth)	
}	

H.7.3.8.5 Coding unit syntax

coding_unit(x0, y0, log2CbSize , ctDepth) {	**Descriptor**
if(transquant_bypass_enabled_flag)	
cu_transquant_bypass_flag	ae(v)
if(slice_type != I)	
cu_skip_flag[x0][y0]	ae(v)
nCbS = (1 << log2CbSize)	
if(cu_skip_flag[x0][y0]) {	
prediction_unit(x0, y0, nCbS, nCbS)	
if (iv_res_pred_flag[nuh_layer_id] && RpRefPicAvailFlag)	
iv_res_pred_weight_idx	ae(v)
if (icEnableFlag)	
ic_flag	ae(v)
}	
else {	
if(slice_type != I)	
pred_mode_flag	ae(v)
if((CuPredMode[x0][y0] != MODE_INTRA || log2CbSize = = MinCbLog2SizeY) && !predPartModeFlag)	
part_mode	ae(v)
if(CuPredMode[x0][y0] = = MODE_INTRA) {	
if(PartMode = = PART_2Nx2N && pcm_enabled_flag && log2CbSize >= Log2MinIpcmCbSizeY && log2CbSize <= Log2MaxIpcmCbSizeY)	
pcm_flag[x0][y0]	ae(v)

```
    if( pcm_flag[ x0 ][ y0 ] ) {
      while( !byte_aligned( ) )
        pcm_alignment_zero_bit                                                 f(1)
      pcm_sample( x0, y0, log2CbSize )
    } else {
      pbOffset = ( PartMode  = =  PART_NxN ) ? ( nCbS / 2 ) : nCbS
      for( j = 0; j < nCbS; j = j + pbOffset )
        for( i = 0; i < nCbS; i = i + pbOffset ) {
          if( vps_depth_modes_flag[ nuh_layer_id ] )
            depth_mode_parameters( x0 + i ,  y0+ j , log2CbSize )
          if( DepthIntraMode[ x0 + i ][ y0 + j ]  = =  INTRA_DEP_NONE )
            prev_intra_luma_pred_flag[ x0 + i ][ y0 + j ]                      ae(v)
        }
      for( j = 0; j < nCbS; j = j + pbOffset )
        for( i = 0; i < nCbS; i = i + pbOffset )
          if( DepthIntraMode[ x0 + i ][ y0 + j ]  = =  INTRA_DEP_NONE) {
            if( prev_intra_luma_pred_flag[ x0 + i ][ y0 + j ] )
              mpm_idx[ x0 + i ][ y0 + j ]                                      ae(v)
            else
              rem_intra_luma_pred_mode[ x0 + i ][ y0 + j ]                     ae(v)
          }
      intra_chroma_pred_mode[ x0 ][ y0 ]                                       ae(v)
    }
  } else {
    if( PartMode  = =  PART_2Nx2N ) {
      prediction_unit( x0, y0, nCbS, nCbS )
      if ( iv_res_pred_flag[ nuh_layer_id ]  &&  RpRefPicAvailFlag )
        iv_res_pred_weight_idx                                                 ae(v)
    } else if( PartMode  = =  PART_2NxN ) {
      prediction_unit( x0, y0, nCbS, nCbS / 2 )
      prediction_unit( x0, y0 + ( nCbS / 2 ), nCbS, nCbS / 2 )
    } else if( PartMode  = =  PART_Nx2N ) {
      prediction_unit( x0, y0, nCbS / 2, nCbS )
      prediction_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS )
    } else if( PartMode  = =  PART_2NxnU ) {
      prediction_unit( x0, y0, nCbS, nCbS / 4 )
      prediction_unit( x0, y0 + ( nCbS / 4 ), nCbS, nCbS * 3 / 4 )
    } else if( PartMode  = =  PART_2NxnD ) {
      prediction_unit( x0, y0, nCbS, nCbS * 3 / 4 )
      prediction_unit( x0, y0 + ( nCbS * 3 / 4 ), nCbS, nCbS / 4 )
    } else if( PartMode  = =  PART_nLx2N ) {
      prediction_unit( x0, y0, nCbS / 4, nCbS )
      prediction_unit( x0 + ( nCbS / 4 ), y0, nCbS * 3 / 4, nCbS )
    } else if( PartMode  = =  PART_nRx2N ) {
      prediction_unit( x0, y0, nCbS * 3 / 4, nCbS )
      prediction_unit( x0 + ( nCbS * 3 / 4 ), y0, nCbS / 4, nCbS )
    } else { /* PART_NxN */
      prediction_unit( x0, y0, nCbS / 2, nCbS / 2 )
      prediction_unit( x0 + ( nCbS / 2 ), y0, nCbS / 2, nCbS / 2 )
      prediction_unit( x0, y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 )
      prediction_unit( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), nCbS / 2, nCbS / 2 )
    }
  }
```

if (icEnableFlag)	
ic_flag	ae(v)
if(vps_inter_sdc_flag && PredMode[x0][y0] ! = MODE_INTRA && !skip_flag[x0][y0])	
inter_sdc_flag	ae(v)
if(inter_sdc_flag) {	
puNum = (PartMode = = PART_2Nx2N) ? 1 : (PartMode = = PART_NxN ? 4 : 2)	
for(i = 0; i < puNum; i++) {	
inter_sdc_resi_abs_minus1[x0][y0][i]	ae(v)
inter_sdc_resi_sign_flag[x0][y0][i]	ae(v)
}	
}	
if(!pcm_flag[x0][y0]) {	
if(CuPredMode[x0][y0] != MODE_INTRA && !(PartMode = = PART_2Nx2N && merge_flag[x0][y0]))	
rqt_root_cbf	ae(v)
if(rqt_root_cbf && !inter_sdc_flag) {	
MaxTrafoDepth = (CuPredMode[x0][y0] = = MODE_INTRA ? (max_transform_hierarchy_depth_intra + IntraSplitFlag) : max_transform_hierarchy_depth_inter)	
transform_tree(x0, y0, x0, y0, log2CbSize, 0, 0)	
}	
}	
}	
}	

H.7.3.8.5.1 Depth mode parameter syntax

depth_mode_parameters(x0 , y0 , log2CbSize) {	**Descriptor**
depth_intra_mode[x0][y0]	ae(v)
if (DepthIntraMode[x0][y0] = = INTRA_DEP_DMM_WFULL || DepthIntraMode[x0][y0] = = INTRA_DEP_SDC_DMM_WFULL)	
wedge_full_tab_idx[x0][y0]	ae(v)
if(DmmFlag[x0][y0] || SdcFlag[x0][y0]) {	
dcNumSeg = (DepthIntraMode[x0][y0] = = INTRA_DEP_SDC_PLANAR) ? 1 : 2	
depth_dc_flag[x0][y0]	ae(v)
if (depth_dc_flag[x0][y0])	
for(i = 0; i < dcNumSeg; i ++) {	
depth_dc_abs[x0][y0][i]	ae(v)
if (depth_dc_abs[x0][y0][i])	
depth_dc_sign_flag[x0][y0][i]	ae(v)
}	
}	
}	

H.7.3.8.6 Prediction unit syntax

The specifications in subclause 7.3.8.6 apply.

H.7.3.8.7 PCM sample syntax

The specifications in subclause 7.3.8.7 apply.

H.7.3.8.8 Transform tree syntax

The specifications in subclause 7.3.8.8 apply.

H.7.3.8.9 Motion vector difference coding syntax

The specifications in subclause 7.3.8.9 apply.

H.7.3.8.10 Transform unit syntax

The specifications in subclause 7.3.8.10 apply.

H.7.3.8.11 Residual coding syntax

The specifications in subclause 7.3.8.11 apply.

H.7.4 Semantics

H.7.4.1 General

H.7.4.2 NAL unit semantics

H.7.4.2.1 General NAL unit semantics

The specifications in subclause G.7.4.2.1 apply.

H.7.4.2.2 NAL unit header semantics

The specification in subclause G.7.4.2.2 apply with the following modifications and additions.

The variable RapPicFlag is derived as specified in the following:

RapPicFlag = (nal_unit_type >= BLA_W_LP && nal_unit_type <= RSV_IRAP_VCL23) (H-1)

H.7.4.2.3 Encapsulation of an SODB within an RBSP (informative)

The specifications in subclause G.7.4.2.3 apply.

H.7.4.2.4 Order of NAL units and association to coded pictures, access units, and video sequences

The specifications in subclause G.7.4.2.4 apply.

H.7.4.3 Raw byte sequence payloads, trailing bits, and byte alignment semantics

H.7.4.3.1 Video parameter set RBSP semantics

The specifications in subclause G.7.4.3.1 apply, with the following modifications and additions:

vps_extension2_flag equal to 0 specifies that no vps_extension2( ) syntax structure is present in the VPS RBSP syntax structure. vps_extension_flag equal to 1 specifies that the vps_extension2( ) syntax structure is present in the VPS RBSP syntax structure. The variable MvHevcCompatibilityFlag is set equal to !vps_extension2_flag. [Ed.(GT): At some stage this might be changed to profile Idc. Moreover, vps_extensions for different HEVC extensions need to be harmonized.]

vps_extension3_flag equal to 0 specifies that no vps_extension3_data_flag syntax elements are present in the VPS RBSP syntax structure. vps_extension2_flag shall be equal to 1 in bitstreams conforming to Annex H of this Recommendation | International Standard. The value of 1 for vps_extension3_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all data that follow the value 1 for vps_extension3_flag in a VPS NAL unit.

H.7.4.3.1.1 Video parameter set extension semantics

The specifications in subclause G.7.4.3.1.1 apply, with the following modifications and additions:

– Table F-1 is replaced by Table H-1.

Table H-1 – Mapping of ScalabiltyId to scalability dimensions

scalability mask index	Scalability dimension	ScalabilityId mapping
0	Depth	Depth Flag
1	Multiview	View Order Index
2-15	Reserved	

The variable ScalabilityId[i][smIdx] specifying the identifier of the smIdx-th scalability dimension type of the i-th layer, the variable ViewOrderIdx[layer_id_in_nuh[i]] specifying the view order index of the i-th layer, the variable VpsDepthFlag[layer_id_in_nuh[i]] specifying the depth flag of the i-th layer, and the variable ViewScalExtLayerFlag specifying whether the i-th layer is a view scalability extension layer are derived as follows:

```
NumViews = 1
for( i = 0; i <= vps_max_layers_minus1; i++ ) {
    lId = layer_id_in_nuh[ i ]
    for( smIdx= 0, j = 0; smIdx < 16; smIdx++ )
        if( scalability_mask_flag[ smIdx ] )
            ScalabilityId[ i ][ smIdx ] = dimension_id[ i ][ j++ ]
```

```
        VpsDepthFlag[ lId ] = ScalabilityId[ i ][ 0 ]
        ViewOrderIdx[ lId ] = ScalabilityId[ i ][ 1 ]
        if( i > 0  &&  ( ViewOrderIdx[ lId ]  !=  ScalabilityId[ i – 1][ 1 ] ) )
            NumViews++
        ViewScalExtLayerFlag[ lId ] = ( ViewOrderIdx[ lId ] > 0 )
    }
```

The function ViewIdx(picX) is specified as follows:

ViewIdx(picX) = ViewOrderIdx[nuh_layer_id of the picture picX] (H-2)

The function DepthFlag(picX) is specified as follows:

DepthFlag(picX) = VpsDepthFlag[nuh_layer_id of the picture picX] (H-3)

The function ViewId(picX) is specified as follows:

ViewId(picX) = ViewId[nuh_layer_id of the picture picX] (H-4)

The function DiffViewId(picA, picB) is specified as follows:

DiffViewId(picA, picB) = ViewId(picA) − ViewId(picB) (H-5)

H.7.4.3.1.2 Video parameter set extension 2 semantics

iv_mv_pred_flag[layerId] indicates whether inter-view motion parameter prediction is used in the decoding process of the layer with nuh_layer_id equal to layerId. iv_mv_pred_flag[layerId] equal to 0 specifies that inter-view motion parameter prediction is not used for the layer with nuh_layer_id equal to layerId. iv_mv_pred_flag[layerId] equal to 1 specifies that inter-view motion parameter prediction may be used for the layer with nuh_layer_id equal to layerId. When not present, the value of iv_mv_pred_flag[layerId] is inferred to be equal to 0.

log2_sub_pb_size_minus2[layerId] specifies the value of the variable SubPbSize[layerId] that is used in the decoding of prediction units using the inter-view merge candidate. The value of log2_sub_pb_size_minus2 shall be in the range of 0 to 4, inclusive.

[Ed. (CY): There sounds to be no need and no agreement to send this syntax element for each view. In addition, sub-PU doesn’t apply to depth views.]

The variable SubPbSize[layerId] is derived as specified in the following:

SubPbSize[layerId] = VpsDepthFlag(layerId) ? 64 : 1 << (log2_sub_pb_size_minus2[layerId]+ 2) (H-6)

[Ed. (GT): The derivation of SubPbSize corresponds to the fixed derivation process in HTM-9.0r1. Further discussions might be required.]

iv_res_pred_flag[layerId] indicates whether inter-view residual prediction is used in the decoding process of the layer with nuh_layer_id equal to layerId. iv_res_pred_flag[layerId] equal to 0 specifies that inter-view residual prediction is not used for the layer with nuh_layer_id equal to layerId. iv_res_pred_flag[layerId] equal to 1 specifies that inter-view residual prediction may be used for the layer with nuh_layer_id equal to layerId. When not present, the value of iv_res_pred_flag[layerId] is to be equal to 0.

view_synthesis_pred_flag[layerId] equal to 0 specifies that view synthesis prediction merge candidates are not used for the layer with nuh_layer_id equal to layerId. view_synthesis_pred_flag[layerId] equal to 1 specifies that view synthesis prediction merge candidates might be used for the layer with nuh_layer_id equal to layerId. When not present, the value of view_synthesis_pred_flag[layerId] is inferred to be equal to 0.

depth_refinement_flag[layerId] equal to 0 specifies that depth view components are not used in the derivation process for a disparity vector for the layer with nuh_layer_id equal to layerId. depth_refinement_flag[layerId] equal to 1 specifies that depth components are used in the derivation process for a disparity vector for the layer with nuh_layer_id equal to layerId. When not present, the value of depth_refinement_flag[layerId] is inferred to be equal to 0.

mpi_flag[layerId] equal to 0 specifies that motion parameter inheritance is not used for the layer with nuh_layer_id equal to layerId. mpi_flag[layerId] equal to 1 specifies that motion parameter inheritance may be used for the layer with nuh_layer_id equal to layerId. When not present, the value of mpi_flag[layerId] is inferred to be equal to 0.

vps_depth_modes_flag[layerId] equal to 1 specifies that depth map modelling modes, the chain coding mode and simplified depth coding modes may be used in the decoding process of the layer with layer_id equal to layerId. vps_depth_modes_flag[layerId] equal to 0 specifies that depth map modelling modes, the chain coding mode and simplified depth coding modes are not used in the decoding process of the layer with layer_id equal to layerId. When not present, vps_depth_modes_flag[layerId] is inferred to be equal to 0.

lim_qt_pred_flag[layerId] equal to 1 specifies that prediction of a limited quadtree is used for the layer with nuh_layer_id equal to layerId.. lim_qt_pred_flag[layerId] equal to 0 specifies that prediction of a limited quadtree is not used for the layer with nuh_layer_id equal to layerId. When not present, the value of lim_qt_pred_flag[layerId] is inferred to be equal to 0.

vps_inter_sdc_flag[layerId] equal to 1 specifies that inter SDC coding is used for the layer with nuh_layer_id equal to layerId. vps_inter_sdc_flag[layerId] equal to 0 specifies that inter SDC coding is not used for the layer with nuh_layer_id equal to layerId. When not present, the value of vps_inter_sdc_flag[layerId] is inferred to be equal to 0.

cp_precision specifies the precision of vps_cp_scale[i][j], vps_cp_off[i][j], vps_cp_inv_scale_plus_scale[i][j], and vps_cp_inv_off_plus_off[i][j] in the VPS and cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] in the slice segment header. The value of cp_precision shall be in the range of 0 to 5, inclusive.

cp_present_flag[i] equal to 1 specifies that the syntax elements vps_cp_scale[i][j], vps_cp_off[i][j], vps_cp_inv_scale_plus_scale[i][j], and vps_cp_inv_off_plus_off[i][j] are present in the VPS or that cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] are present in slice segment headers with nuh_layer_id equal to layerId and VpsViewIdx[layerId] equal to i. cp_present_flag[i] equal to 1 indicates that camera parameters are not present.

For layerId in the range of 0 to MaxLayersMinus1, inclusive, the following applies:

$$\text{cpRequiredFlag[layerId]} = \text{depth\_refinement\_flag[layerId]} \;||\; \text{view\_synthesis\_pred\_flag[layerId]} \;||\; (\text{iv\_mv\_pred\_flag[layerId]} \;\&\&\; \text{VpsDepthFlag[layerId]}) \quad \text{(H-7)}$$

When, for any value of layerId, cpRequiredFlag[layerId] is equal to 1, the value of cp_present_flag[VpsViewIdx[layerId]] shall be equal to 1. When not present, the value of cp_present_flag[i] is inferred to be equal to 0.

cp_in_slice_segment_header_flag[i] equal to 1 specifies that the syntax elements vps_cp_scale[i][j], vps_cp_off[i][j], vps_cp_inv_scale_plus_scale[i][j], and vps_cp_inv_off_plus_off[i][j] are not present in the VPS and that the syntax elements cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] are present in slice segment headers with nuh_layer_id equal to layerId and VpsViewIdx[layerId] equal to i. cp_in_slice_segment_header_flag equal to 0 specifies that the vps_cp_scale[i][j], vps_cp_off[i][j], vps_cp_inv_scale_plus_scale[i][j], and vps_cp_inv_off_plus_off[i][j] syntax elements are present in the VPS and that the syntax elements cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] are not present in slice segment headers with nuh_layer_id equal to layerId and VpsViewIdx[layerId] equal to i. When not present, the value of cp_in_slice_segment_header_flag[i] is inferred to be equal to 0.

vps_cp_scale[i][j], **vps_cp_off**[i][j], **vps_cp_inv_scale_plus_scale**[i][j], and **vps_cp_inv_off_plus_off**[i][j] specify conversion parameters for converting a depth value to a disparity value and might be used to infer the values of cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] for the i-th view specified in VPS. When the i-th view contains both a texture view and a depth view, the conversion parameters are associated with the texture view.

iv_mv_scaling_flag equal to 1 specifies that motion vectors used for inter-view prediction in a layer with nuh_layer_id equal to layerId may be scaled based on ViewId**[** layerId **]** values. iv_mv_scaling_flag equal to 0 specifies that motion vectors used for inter-view prediction in a layer with nuh_layer_id equal to layerId are not scaled based on ViewId**[** layerId **]** values. When not present, the value of iv_mv_scaling_flag is inferred to be equal to 0.

H.7.4.3.2 Sequence parameter set RBSP semantics

The specifications in subclause G.7.4.3.2 and its subclauses apply, with the following modifications and additions:

sps_extension2_flag equal to 0 specifies that no sps_extension2( ) syntax structure is present in the SPS RBSP syntax structure. sps_extension2_flag equal to 1 specifies that the sps_extension2( ) syntax structure is present in the SPS RBSP syntax structure.

sps_extension3_flag equal to 0 specifies that no sps_extension_data_flag syntax elements are present in the SPS RBSP syntax structure. sps_extension3_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. The value of 1 for sps_extension3_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all sps_extension_data_flag syntax elements that follow the value 1 for sps_extension3_flag in an SPS NAL unit.

H.7.4.3.3 Picture parameter set RBSP semantics

The specifications in subclause G.7.4.3.3 apply with the following modifications and additions:

pps_extension2_flag equal to 0 specifies that no pps_extension_data_flag syntax elements are present in the PPS RBSP syntax structure. pps_extension2_flag shall be equal to 0 in the bitstream conforming to this version of this specification. The value 1 for pps_extension2_flag is reserved for future use by ITU-T | ISO/IEC. Decoders shall ignore all pps_extension_data_flag syntax elements that follow the value 1 for pps_extension2_flag in a PPS NAL unit.

H.7.4.3.3.1 Picture parameter set extension semantics

dlt_present_flag equal to 1 specifies the depth lookup tables for the depth views are present in this PPS. dlt_present_flag equal to 0 specifies the depth lookup tables for the depth views are not present in this PPS.

For variables NumDepthLayers and DepthIdxToLayerIdInNuh are derived as specified in the following:

```
j = 0
for ( i = 0; i  <=  MaxNumLayersMinus1; i++) {
    layerId = LayerIdInNuh[ i ]
    if( VpsDepthFlag[ layerId ] )
        DepthIdxToLayerIdInNuh[ j++ ] = layerId
NumDepthLayers = j
```

pps_depth_layers_minus1 plus 1 specifies the number of depth layers. pps_depth_layers_minus1 shall be equal to NumDepthLayers − 1.

pps_bit_depth_for_depth_views_minus8 plus 8 specifies the bit depth of the samples of the depth layer.

The variable depthMaxValue is set equal to (1 << (pps_bit_depth_for_depth_views_minus8 + 8)) − 1.

dlt_flag[i] equal to 1 specifies that the i-th depth lookup table is present in the PPS and used and for the coding of the layer with nuh_layer_id equal to DepthIdxToLayerIdInNuh[i]. dlt_flag[i] equal to 0 specifies that a depth lookup table is not present for the layer with nuh_layer_id equal to DepthIdxToLayerIdInNuh[i]. When not present, the value of dlt_flag[i] is inferred to be equal to 0.

For i in the range of 0 to NumDepthLayers −1, the variable DltFlag[DepthIdxToLayerIdInNuh[i]] is set equal to dlt_flag[i].

inter_view_dlt_pred_enable_flag[i] equal to 1 indicates the i-th depth lookup table is predicted from the depth lookup table of the 0-th depth lookup table. [Ed. (CY): a more generic solution is to signal the reference view for DLT prediction.] inter_view_dlt_pred_enable_flag[i] equal to 0 indicates the i-th depth lookup table is not predicted from any other depth lookup table. The value of inter_view_dlt_pred_enable_flag[0] shall be equal to 0.

[Ed. (GT): Since flexible coding order is not allowed the 0-th DLT always belongs to the base view. See comments on derivation process below.]

dlt_bit_map_rep_flag[i] equal to 1 specifies the i-th depth lookup table is represented as bit map by the syntax elements dlt_bit_map_flag[j]. dlt_bit_map_rep_flag[i] equal to 0 specifies the i-th depth lookup table is derived from the entry table. When not present, the value of dlt_bit_map_rep_flag[i] is inferred to be equal to 0.

dlt_bit_map_flag[i][j] equal to 1 specifies that the depth value equal to j is present in the i-th depth lookup table as one entry. dlt_bit_map_flag[i][j] equal to 0 specifies that the depth value equal to j is not an entry of the depth lookup table as one entry.

The variable layerId is set equal to DepthIdxToLayerIdInNuh[i] and when dlt_bit_map_rep_flag[i] is equal to 1, the variables DltDepthValue[layerId][k]] and NumDepthValuesInDlt[layerId] of the i-th depth lookup table are derived as follows:

```
k = 0
for( j = 0; j <=  depthMaxValue; j++ )
    if( dlt_bit_map_flag[ i ][ j ] )                                        (H-8)
        DltDepthValue[ layerId ][ k++ ] = j
NumDepthValuesInDlt[ layerId ] = k
```

H.7.4.3.3.2 Entry table semantics

num_entry specifies the number of entries in the i-th depth lookup table. The length of num_entry syntax element is pps_bit_depth_for_depth_views_minus8 + 8 bits.

max_diff specifies the maximum difference between two consecutive entries of the i-th depth lookup table. The length of max_diff syntax element is pps_bit_depth_for_depth_views_minus8 + 8 bits. When not present, the value of max_diff is inferred to be equal to 0.

min_diff_minus1 specifies the minimum difference between two consecutive entries of the i-th depth lookup table, min_diff_minus1 is in the range of 0 to max_diff – 1, inclusive. The length of the min_diff_minus1 syntax element is Ceil(Log2(max_diff + 1)) bits. When not present, the value of min_diff_minus1 is inferred to be equal to (max_diff − 1).

The variable minDiff is set equal to (min_diff_minus1 + 1).

entry0 specifies the 0-th entry of the i-th depth lookup table. The length of the entry0 syntax element is pps_bit_depth_for_depth_views_minus8 + 8 bits

entry_value_diff_minus_min[k] plus minDiff specifies the difference between the k-th entry and the (k − 1)-th entry in the i-th depth lookup table. The length of entry_value_diff_minus_min[k] syntax element is Ceil(Log2(max_diff − minDiff + 1)) bits.

The variable entry[k] is derived as specified in the following:

```
entry[ 0 ]= entry0
for( k = 1; k < num_entry; k++ )                                                    (H-9)
    entry[ k ] = entry[ k − 1 ] + entry_value_diff_minus_min[ k ] + minDiff
```

The variable layerId is set equal to DepthIdxToLayerIdInNuh[i] and the variable baseDepthLayerId is set equal to DepthIdxToLayerIdInNuh[0].

Depending on inter_view_dlt_pred_enable_flag[i] the variables DltDepthValue[layerId][k]] and NumDepthValueInDlt[layerId] of the i-th depth lookup table are derived as specified in the following:

- If inter_view_dlt_pred_enable_flag[i] is equal to 0, the following applies: :

```
NumDepthValueInDlt[ layerId ] = num_entry
for( j = 0; j < num_entry; j++ )                                                   (H-10)
    DltDepthValue[ layerId ][ j ] = entry[ j ]
```

- Otherwise (inter_view_dlt_pred_enable_flag[i] is equal to 1), the following applies:

```
for( j = 0, k = 0; j < depthMaxValue  &&  k < NumDepthValueInDlt[ baseDepthLayerId ]; j++ ) {
    dltRefBitMapFlag[ j ] = 0
    if( DltDepthValue[ baseDepthLayerId ][ k ]  = =  j ) {
        dltRefBitMapFlag[ j ] = 1
        k++
    }
}
```

[Ed. (CY): the above calculations assume that the 0-th DLT is the DLT table of the base view.]
[Ed. (GT): Since flexible coding order is not allowed (Although text for this seems to missing in the draft), this assumption is right.]

```
for( j = 0, k = 0; j < depthMaxValue  &&  k< num_entry; j++ ) {
    dltSignalBitMapFlag[ j ] = 0                                                    (H-11)
    if( entry[ k ]  = =  j ) {
        dltSignalBitMapFlag[ j ] = 1
        k++
    }
}
for( j = 0; j < depthMaxValue; j++ )
    dltBitMapCurrFlag[ j ] = dltRefBitMapFlag[ j ] ^ dltSignalBitMapFlag[ j ]
for( j = 0, k = 0; j  <=  depthMaxValue; j++ )
    if( dltBitMapCurrFlag[ j ] )
        DltDepthValue[ layerId ][ k++ ] = j
NumDepthValueInDlt[ layerId ] = k
```

H.7.4.3.4 Supplemental enhancement information RBSP semantics

The specifications in subclause G.7.4.3.4 apply.

H.7.4.3.5 Access unit delimiter RBSP semantics

The specifications in subclause G.7.4.3.5 apply.

H.7.4.3.6 End of sequence RBSP semantics

The specifications in subclause G.7.4.3.6 apply.

H.7.4.3.7 End of bitstream RBSP semantics

The specifications in subclause G.7.4.3.7 apply.

H.7.4.3.8 Filler data RBSP semantics

The specifications in subclause G.7.4.3.8 apply.

H.7.4.3.9 Slice layer RBSP semantics

The specifications in subclause G.7.4.3.9 apply.

H.7.4.3.10 RBSP slice trailing bits semantics

The specifications in subclause G.7.4.3.10 apply.

H.7.4.3.11 RBSP trailing bits semantics

The specifications in subclause G.7.4.3.11 apply.

H.7.4.3.12 Byte alignment semantics

The specifications in subclause G.7.4.3.12 apply.

H.7.4.4 Profile, tier and level semantics

The specifications in subclause G.7.4.4 apply.

H.7.4.5 Scaling list

The specifications in subclause G.7.4.5 apply.

H.7.4.6 Supplemental enhancement information message semantics

The specifications in subclause 7.4.6 apply.

H.7.4.7 Slice segment header semantics

H.7.4.7.1 General slice segment header semantics

The specification in subclause G.7.4.7.1 apply with the following modifications and additions.

The variable DepthFlag is set equal to VpsDepthFlag[nuh_layer_id] and the variable ViewIdx is set equal to ViewOrderIdx[nuh_layer_id].

five_minus_max_num_merge_cand specifies the maximum number of merging MVP candidates supported in the slice subtracted from 5.

The variable NumExtraMergeCand is derived as specified in the following:

$$\text{NumExtraMergeCand} = \text{iv\_mv\_pred\_flag}[\ \text{nuh\_layer\_id}\ ]\ |\,|\ \text{mpi\_flag}[\ \text{nuh\_layer\_id}\ ] \qquad \text{(H-12)}$$

The maximum number of merging MVP candidates, MaxNumMergeCand is derived as follows:

$$\text{MaxNumMergeCand} = 5 - \text{five\_minus\_max\_num\_merge\_cand} + \text{NumExtraMergeCand} \qquad \text{(H-13)}$$

The value of MaxNumMergeCand shall be in the range of 1 to (5 + NumExtraMergeCand), inclusive.

slice_ic_enable_flag equal to 1 specifies illumination compensation is enabled for the current slice. slice_ic_enable_flag equal to 0 specifies that illumination compensation is disabled for the current slice, When not present, slice_ic_enable_flag is inferred to be equal to 0.

slice_ic_disable_merge_zero_idx_flag equal to 1 specifies that ic_flag is not present in the coding units with partitioning mode equal to PART_2Nx2N of the current slice when merge_flag is equal to 1 and merge_idx of the prediction unit in the coding unit is equal to 0. slice_ic_disable_merge_zero_idx_flag equal to 0 specifies that ic_flag might be present in the coding units with partitioning mode equal to PART_2Nx2N of the current slice when merge_flag is equal to 1 and merge_idx of the prediction unit in the coding unit is equal to 0. When not present, slice_ic_disable_merge_zero_idx_flag is inferred to be equal to 0.

cp_scale[j], **cp_off**[j], **cp_inv_scale_plus_scale**[j], and **cp_inv_off_plus_off**[j] specify conversion parameters for converting a depth value to a disparity value. When not present, the values of cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j], are inferred to be equal to vps_cp_scale[ViewIdx][j], vps_cp_off[ViewIdx][j], vps_cp_inv_scale_plus_scale[ViewIdx][j], and vps_cp_inv_off_plus_off[ViewIdx][j], respectively. It is a requirement of bitstream conformance, that the values of cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] in a slice segment header having a ViewIdx equal to viewIdxA and the values of cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] in a slice segment header having a ViewIdx equal to viewIdxB shall be the same, when viewIdxA is equal to viewIdxB.

[Ed. (GT): Consider adding range limitations for values of above syntax elements.]

The array DepthToDisparityB[j][d] specifying the disparity between the current view and the view with ViewIdx equal j corresponding to the depth value d in the view with ViewIdx equal to j and the array DepthToDisparityF[j][d] specifying the disparity between the view with ViewIdx equal j and the current view corresponding to the depth value d in the current view is derived as specified in the following:

- The variable log2Div is set equal to $\text{BitDepth}_Y - 1 + \text{cp\_precision}$.
- For d in range of 0 to $((1 \ll \text{BitDepth}_Y) - 1)$, inclusive, the following applies:
  - For i in the range of 0 to $\text{ViewIdx} - 1$, inclusive, the following applies:

$$\text{offset} = (\text{cp\_off}[j] \ll \text{BitDepth}_Y) + ((1 \ll \text{log2Div}) \gg 1) \quad \text{(H-14)}$$

$$\text{scale} = \text{cp\_scale}[j] \quad \text{(H-15)}$$

$$\text{DepthToDisparityB}[j][d] = (\text{scale} * d + \text{offset}) \gg \text{log2Div} \quad \text{(H-16)}$$

$$\text{invOffset} = ((\text{cp\_inv\_off\_plus\_off}[j] - \text{cp\_off}[j]) \ll \text{BitDepth}_Y) + ((1 \ll \text{log2Div}) \gg 1) \quad \text{(H-17)}$$

$$\text{invScale} = (\text{cp\_inv\_scale\_plus\_scale}[j] - \text{cp\_scale}[j]) \quad \text{(H-18)}$$

$$\text{DepthToDisparityF}[j][d] = (\text{invScale} * d + \text{invOffset}) \gg \text{log2Div} \quad \text{(H-19)}$$

H.7.4.7.2 Reference picture list modification semantics

The specifications in subclause G.7.4.5.2 apply.

H.7.4.8 Short-term reference picture set semantics

The specifications in subclause G.7.4.8 apply.

H.7.4.9 Slice data semantics

H.7.4.9.1 Slice data semantics

The specifications in subclause 7.4.9.1 apply.

H.7.4.9.2 Coding tree unit semantics

The specifications in subclause 7.4.9.2 apply, with the following modifications and additions.

Let DepthPic be the picture in the current access unit with ViewIdx(DepthPic) equal to ViewIdx and DepthFlag(DepthPic) equal to 1.

Let TexturePic be the picture in the current access unit with ViewIdx(TexturePic) equal to ViewIdx and DepthFlag(TexturePic) equal to 0.

The arrays TextureCtDepth TexturePartMod, TexturePredMode, and TextureIntraPredModeY are set equal to the arrays CtDepth, CtPartMode, PredMode, and IntraPredModeY of Texture Pic, respectively...

H.7.4.9.3 Sample adaptive offset semantics

The specification in subclause 7.4.9.3 apply.

H.7.4.9.4 Coding quadtree semantics

The specifications in subclause 7.4.9.4 apply.

The variable predSplitCuFlag specifying whether the split_cu_flag is predicted by inter-component prediction is derived as specified in the following.

  - If slice_type is not equal to I, RapPicFlag is equal to 0, and lim_qt_pred_flag[nuh_layer_id] is equal to 1, predSplitCuFlag is set equal to (TextureCtDepth[x0][y0] <= ctDepth)
  - Otherwise (slice_type is equal to I or RapPicFlag is equal to 1 or lim_qt_pred_flag[nuh_layer_id] is equal to 0), predSplitCuFlag is set equal to 0.

split_cu_flag[x0][y0] specifies whether a coding unit is split into coding units with half horizontal and vertical size. The array indices x0, y0 specify the location (x0, y0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

When split_cu_flag[x0][y0] is not present, the following applies:

- If log2CbSize is greater than MinCbLog2SizeY and predSplitCuFlag is equal to 0 , the value of split_cu_flag[x0][y0] is inferred to be equal to 1.

– Otherwise (log2CbSize is equal to MinCbLog2SizeY or predSplitCuFlag is equal to 1), the value of split_cu_flag[x0][y0] is inferred to be equal to 0.

H.7.4.9.5 Coding unit semantics

The specification in subclause 7.4.9.5 apply with the following modifications and additions:

– If slice_type is not equal to I, RapPicFlag is equal to 0 and lim_qt_pred_flag[nuh_layer_id] is equal to 1, the variable predPartModeFlag specifying whether part_mode is predicted by inter-component prediction is derived as follows.

$$\begin{aligned}&\text{predPartModeFlag} =\\&(\,\text{TextureCtDepth}[\,x0\,][\,y0\,] == \text{ctDepth}\,)\ \&\&\ (\,\text{TexturePartMode}[\,x0\,][\,y0\,]\ != \text{PART\_NxN}\,)\end{aligned}\tag{H-20}$$

– Otherwise (slice_type is equal to I or RapPicFlag is equal to 1 or lim_qt_pred_flag[nuh_layer_id] is equal to 0), predPartModeFlag is set equal to 0.

iv_res_pred_weight_idx specifies the index of the weighting factor used for residual prediction. iv_res_pred_weight_idx equal to 0 specifies that residual prediction is not used for the current coding unit. iv_res_pred_weight_idx not equal to 0 specifies that residual prediction is used for the current coding unit. When not present, the value of iv_res_pred_weight_idx is inferred to be equal to 0.

When DefaultDispFlag[x0][y0] is equal to 1, iv_res_pred_weight_idx shall be equal to 0.

The variable icEnableFlag is set equal to 0 and when slice_ic_enable_flag is equal to 1 and PartMode is equal to 2Nx2N and PredMode[x0][y0] is not equal to MODE_INTRA, the following applies:

– If merge_flag[x0][y0] is equal to 1, the following applies:

$$\text{icEnableFlag} = (\,\text{merge\_idx}[\,x0\,][\,y0\,]\ != 0\,)\ ||\ !\text{slice\_ic\_disable\_merge\_zero\_idx\_flag}\tag{H-21}$$

– Otherwise (merge_flag[x0][y0] is equal to 0), the following applies:

  – With X being replaced by 0 and 1, the variable refViewIdxLX is set equal to ViewIdx(RefPicListLX[ref_idx_lX[x0][y0]]).

  – The flag icEnableFlag is derived as specified in the following:

$$\begin{aligned}&\text{icEnableFlag} =\\&(\text{inter\_pred\_idc}[\,x0\,][\,y0\,]\ != \text{Pred\_L0}\ \&\&\ \text{refViewIdxL1}\ != \text{ViewIdx})\ ||\\&(\text{inter\_pred\_idc}[\,x0\,][\,y0\,]\ != \text{Pred\_L1}\ \&\&\ \text{refViewIdxL0}\ != \text{ViewIdx})\end{aligned}\tag{H-22}$$

ic_flag equal to 1 specifies illumination compensation is used for the current coding unit. ic_flag equal to 0 specifies illumination compensation is not used for the current coding unit. When not present, ic_flag is inferred to be equal to 0.

inter_sdc_flag equal to 1 specifies simplified depth coding of residual blocks is used for the current coding unit. inter_sdc_flag equal to 0 specifies simplified depth coding of residual blocks is not used for the current coding unit. When not present, inter_sdc_flag is inferred to be equal to 0.

inter_sdc_resi_abs_minus1[x0][y0][i], **inter_sdc_resi_sign_flag**[x0][y0][i] are used to derive InterSdcResi[x0][y0][i] as follows:

$$\begin{aligned}\text{InterSdcResi}[\,x0\,][\,y0\,][\,i\,] = (\,1 - 2 * \text{inter\_sdc\_resi\_sign\_flag}[\,x0\,][\,y0\,][\,i\,]\,)\,* \\ (\,\text{inter\_sdc\_resi\_abs\_minus1}[\,x0\,][\,y0\,][\,i\,] + 1\,)\end{aligned}\tag{H-23}$$

rqt_root_cbf equal to 1 specifies that the transform_tree( ) syntax structure is present for the current coding unit. rqt_root_cbf equal to 0 specifies that the transform_tree( ) syntax structure is not present for the current coding unit.. When not present, the value of rqt_root_cbf is inferred to be equal to !SdcFlag[x0][y0].

When DepthFlag is equal to 0, for use in derivation processes of variables invoked later in the decoding process, the following assignments are made for $x = x0..x0 + (1 \ll \text{log2cbSize}) - 1$, $y = y0..y0 + (1 \ll \text{log2cbSize}) - 1$:

$$\text{CtDepth}[\,x\,][\,y\,] = \text{ctDepth}\tag{H-24}$$

$$\text{CtPartMode}[\,x\,][\,y\,] = \text{PartMode}\tag{H-25}$$

$$\text{CbSize}[\,x\,][\,y\,] = (\,1 \ll \text{log2cbSize}\,)\tag{H-26}$$

$$\text{CbPosX}[\,x\,][\,y\,] = x0\tag{H-27}$$

$$\text{CbPosY}[\,x\,][\,y\,] = y0\tag{H-28}$$

H.7.4.9.5.1 Depth mode parameter semantics

The variable Log2MaxDmmCbSize is set equal to 5.

The variables depthIntraModeSet is derived as specified in the following:

- If log2CbSize is equal to 6, depthIntraModeSet is set equal to 0.
- Otherwise, if log2CbSize is equal to 3 and PartMode[xC][yC] is equal to PART_NxN, depthIntraModeSet is set equal to 1.
- Otherwise, depthIntraModeSet is set equal to 2.

depth_intra_mode[x0][y0] specifies the depth intra mode of the current prediction unit. Table H-2 specifies the value of the variable depthIntraModeMaxLen depending on depthIntraModeSet and the value of the variable DepthIntraMode and the associated name depending on the on depth_intra_mode and depthIntraModeSet.

The variable SdcFlag[x0][y0] is derived as specified in the following:

SdcFlag[x0][y0] = (DepthIntraMode[x0][y0] = = INTRA_DEP_SDC_PLANAR) | | (H-29)
(DepthIntraMode[x0][y0] = = INTRA_DEP_SDC_DMM_WFULL)

The variable DmmFlag[x0][y0] is derived as specified in the following:

DmmFlag[x0][y0] = (DepthIntraMode[x0][y0] = = INTRA_DEP_DMM_WFULL) | | (H-30)
(DepthIntraMode[x0][y0] = = INTRA_DEP_DMM_CPREDTEX)

Table H-2 – Specification of DepthIntraMode and associated name depending on depthIntraModeSet and depth_intra_mode and specification of and depthIntraModeMaxLen depending on depthIntraModeSet

	depthIntraModeSet	0	1	2	
	depthIntraModeMaxLen	1	3	3	
DepthIntraMode	**Associated name**	**depth_intra_mode**			
0	INTRA_DEP_SDC_PLANAR	0	-	0	
1	INTRA_DEP_NONE	1	0	1	
2	INTRA_DEP_SDC_DMM_WFULL	-	-	2	
3	INTRA_DEP_DMM_WFULL	-	1	3	
4	INTRA_DEP_DMM_CPREDTEX	-	-	4	

wedge_full_tab_idx[x0][y0] specifies the index of the wedgelet pattern in the corresponding pattern list when DepthIntraMode[x0][y0] is equal to INTRA_DEP_DMM_WFULL.

depth_dc_flag[x0][y0] equal to 1 specifies that depth_dc_abs[x0][y0][i] and depth_dc_sign_flag[x0][y0][i] are present. depth_dc_flag[x0][y0] equal to 0 specifies that depth_dc_abs[x0][y0][i] and depth_dc_sign_flag[x0][y0][i] are not present.

depth_dc_abs[x0][y0][i], **depth_dc_sign_flag**[x0][y0][i] are used to derive DcOffset[x0][y0][i] as follows:

DcOffset[x0][y0][i] =
(1 − 2 *depth_dc_sign_flag[x0][y0][i]) * (depth_dc_abs[x0][y0][i] − dcNumSeg +2) (H-31)

H.7.4.9.6 Prediction unit semantics

The specifications in subclause 7.4.9.6 apply.

H.7.4.9.7 PCM sample semantics

The specifications in subclause 7.4.9.7 apply.

H.7.4.9.8 Transform tree semantics

The specifications in subclause 7.4.9.8 apply.

H.7.4.9.9 Motion vector difference coding semantics

The specifications in subclause 7.4.9.9 apply.

H.7.4.9.10 Transform unit semantics

The specifications in subclause 7.4.9.10 apply.

H.7.4.9.11 Residual coding semantics

The specifications in subclause 7.4.9.11 apply.

H.8 Decoding process

H.8.1 General decoding process

The specifications in subclause F.8.1 apply with the following modifications:.

- "ViewScalExtLayerFlag[nuh_layer_id] is equal to 1" is replaced by "ViewScalExtLayerFlag[nuh_layer_id] is equal to 1 or VpsDepthFlag[nuh_layer_id] is equal to 1"
- All invocations of the process specified in subclause G.8.1 are replaced with invocations of the process specified in subclause H.8.1.1.

H.8.1.1 Decoding process for a coded picture with nuh_layer_id greater than 0

The decoding process operates as follows for the current picture CurrPic:

1. The decoding of NAL units is specified in subclause G.8.2.
2. The processes in subclause G.8.1.2 and G.8.3.4 specify the following decoding processes using syntax elements in the slice segment layer and above:
   - Prior to decoding the first slice of the current picture, subclause G.8.1.2 is invoked.
   - At the beginning of the decoding process for each P or B slice, the decoding process for reference picture lists construction specified in subclause G.8.3.4 is invoked for derivation of reference picture list 0 (RefPicList0), and when decoding a B slice, reference picture list 1 (RefPicList1).
   - When iv_mv_pred_flag[nuh_layer_id] is equal to 1 or iv_res_pred_flag[nuh_layer_id] is equal to 1, the decoding process for candidate picture list for disparity vector derivation in subclause H.8.3.5is invoked at the beginning of the decoding process for each P or B slice. [Ed. (GT): VSP should be added here as condition.]
   - At the beginning of the decoding process for each P or B slice, the derivation process for the alternative target reference index for TMVP in merge mode as specified in subclause H.8.3.7 is invoked.
   - At the beginning of the decoding process for each P or B slice, the derivation process for the default reference view order index for disparity derivation as specified in subclause H.8.3.8 is invoked.
   - When iv_res_pred_flag[layerId] is equal to 1, the derivation process for the for the target reference index for residual prediction as specified in subclause H.8.3.9 is invoked, at the beginning of the decoding process for each P or B slice.
   - When DltFlag[nuh_layer_id] is equal to 1, the decoding process for the depth lookup table in subclause H.8.3.6 is invoked at the beginning of the decoding process of first slice.
3. The processes in subclauses H.8.3.8, H.8.5, H.8.5.6, and H.8.7 specify decoding processes using syntax elements in all syntax structure layers. It is a requirement of bitstream conformance that the coded slices of the picture shall contain slice segment data for every coding tree unit of the picture, such that the division of the picture into slices, the division of the slices into slice segments, and the division of the slice segments into coding tree units each form a partitioning of the picture.
4. After all slices of the current picture have been decoded, the marking process for ending the decoding of a coded picture with nuh_layer_id greater than 0 specified in subclause G.8.1.3 is invoked.

H.8.2 NAL unit decoding process

The specification in subclause G.8.2 apply.

H.8.3 Slice decoding process

H.8.3.1 Decoding process for picture order count

The specifications in subclause G.8.3.1 apply.

H.8.3.2 Decoding process for reference picture set

The specifications in subclause G.8.3.2 apply.

H.8.3.3 Decoding process for generating unavailable reference pictures

The specifications in subclause G.8.3.3 apply.

H.8.3.4 Decoding process for reference picture lists construction

The specifications in subclause G.8.3.4 apply.

H.8.3.5 Derivation process for the candidate picture list for disparity vector derivation

[Ed. (GT): This algorithm is different from the algorithm in software. However, it seems to be equivalent.]

The variable NumDdvCandPics is set equal to 0 and the candidate picture list DdvCandPicList with a number of NumDdvCandPics elements is constructed as follows.

When slice_temporal_mvp_enabled_flag is equal to 1 the following ordered steps apply:

1. DdvCandPicList[0] is set equal to RefPicListX[collocated_ref_idx], with X equal to (1 – collocated_from_l0_flag), and NumDdvCandPics is set equal to 1.

2. The variable lowestTemporalIdRefs is set equal to 7.

3. NumDdvCandPics, DdvCandPicList[1] and lowestTemporalIdRefs are derived as specified in the following:

```
for ( dir = 0; dir < 2 ; dir++) {
    X = dir ? collocated_from_l0_flag : (1 – collocated_from_l0_flag)
    for( i = 0; i  <=  num_ref_idx_lX_default_active_minus1; i++ ) {
        if(  ViewIdx  = =  ViewIdx( RefPicListX[ i ] )
            &&  ( X  = =  collocated_from_l0_flag  | |  i  !=  collocated_ref_idx )
            &&  ( NumDdvCandPics  ! =  2 ) ) {
            if( RefPicListX[ i ] is a random access view component ) {
                DdvCandPicsList[ 1 ] = RefPicListX[ i ]
                NumDdvCandPics = 2
            }
            else if( lowestTemporalIdRefs > TemporalId of RefPicListX[ i ] )
                lowestTemporalIdRefs = TemporalId of RefPicListX[ i ]
        }
    }
}
```

4. When NumDdvCandPics is equal to 1, the following applies:

```
pocDistance =255
for( dir = 0; dir < 2 ; dir++) {
    X = dir ? collocated_from_l0_flag : ( 1 – collocated_from_l0_flag )
    for( i =0; i <= num_ref_idx_lX_default_active_minus1; i++ ) {
        if( ViewIdx  = =  ViewIdx( RefPicListX[ i ] )
            &&  ( X  = =  collocated_from_l0_flag  | |  i != collocated_ref_idx )
            &&  TemporalId of RefPicListX[ i ] = = lowestTemporalIdRefs
            &&  ( Abs( PicOrderCntVal – PicOrderCnt( RefPicListX[ i ] ) ) < pocDistance ) ) {
            pocDistance = Abs( PicOrderCntVal – PicOrderCnt( RefPicListX[ i ] )
            Z = X
            idx = i
        }
    }
}
if( pocDistance < 255 ) {
    DdvCandPicsList[ 1 ] = RefPicListZ[ idx ]
    NumDdvCandPics =2
}
```

H.8.3.6 Decoding process for a depth lookup table

This process is only invoked when DltFlag[nuh_layer_id] is equal to 1.

The list elements Idx2DepthValue[i] specifying the depth value of the i-th index in the lookup table with i ranging from 0 to NumDepthValuesInDlt[nuh_layer_id] – 1, inclusive is derived as follows.

- For i in the range of 0 to NumDepthValuesInDlt[nuh_layer_id] – 1, inclusive, the elements in Idx2DepthValue are derived as follows:
    - Idx2DepthValue[i] is set equal to DltDepthValue[nuh_layer_id][i]

The list elements DepthValue2Idx[d] specifying the index of depth values d in the lookup table with d ranging from 0 to BitDepth_Y – 1, inclusive are derived as specified in the following:

```
for( d = 0; d < BitDepth_Y; d++ ) {
    idxLower = 0
    for( iL = 1, foundFlag = 0; iL < NumDepthValuesInDlt[ nuh_layer_id ]  &&  !foundFlag; iL++ )
        if( Idx2DepthValue[ iL ] > d ) {
            idxLower = iL − 1
            foundFlag = 1
        }
    idxUpper = NumDepthValuesInDlt[ nuh_layer_id ] − 1
    for( iU = NumDepthValuesInDlt[ nuh_layer_id ] − 2, foundFlag = 0; iU >= 0  &&  !foundFlag; iU++ )
        if ( Idx2DepthValue[ iU ] < d ) {
            idxUpper = iU + 1
            foundFlag = 1
        }
    if( Abs( d − Idx2DepthValue[ idxLower ] ) < Abs ( d − Idx2DepthValue[ idxUpper ] ) )
        DepthValue2Idx[ d ] = idxLower
    else
        DepthValue2Idx[ d ] = idxUpper
```

H.8.3.7 Derivation process for the alternative target reference index for TMVP in merge mode

This process is invoked when the current slice is a P or B slice.

The variables AltRefIdxL0 and AltRefIdxL1 are set equal to –1 and the following applies for X in the range of 0 to 1, inclusive:

- When X is equal to 0 or the current slice is a B slice the following applies:

```
zeroIdxLtFlag = RefPicListX[ 0 ] is a short-term reference picture ? 0 : 1
for( i = 1; i  <=  num_ref_idx_lX_active_minus1  &&  AltRefIdxLX  = =  –1; i++)
    if( ( zeroIdxLtFlag && RefPicListX[ i ] is a short-term reference picture)  | |
        ( !zeroIdxLtFlag  &&  RefPicListX[ i ] is a long-term reference picture ) )
        AltRefIdxLX = i
```

H.8.3.8 Derivation process for the default reference view order index for disparity derivation

This process is invoked when the current slice is a P or B slice.

The variable DefaultViewIdx is set equal to –1, and the following applies for curViewIdx in the range of 0 to (ViewIdx – 1), inclusive:

- The following applies for X in the range of 0 to 1, inclusive:
  - When X is equal to 0 or the current slice is a B slice, the following applies for i in the range of 0 to NumRefPicsLX, inclusive:
    - When all of the following conditions are true, DefaultViewIdx is set equal to curViewIdx.
      - DefaultViewIdx is equal to –1.
      - ViewIdx(RefPicListX[i]) is equal to curViewIdx.
      - PicOrderCnt(RefPicListX[i]) is equal to PicOrderCntVal.

H.8.3.9 Derivation process for the target reference index for residual prediction

This process is invoked when the current slice is a P or B slice.

The variables RpRefIdxL0 and RpRefIdxL1 are set equal to –1, the variables RpRefPicAvailFlagL0 and RpRefPicAvailFlagL1 are set equal to 0.

The following applies for X in the range of 0 to 1, inclusive:

- When X is equal to 0 or the current slice is a B slice the following applies:
  - For i in the range of 0 to num_ref_idx_lX_active_minus1, inclusive, the following applies:
    - When PicOrderCnt(RefPicListX[i]) is not equal to PicOrderCntVal and RpRefPicAvailFlagLX is equal to 0, the following applies:

$$\text{RpRefIdxLX} = i \quad \text{(H-32)}$$

RpRefPicAvailFlagLX = 1 (H-33)

[Ed. (GT): There might be pictures present in the DPB fulfilling the above conditions, but having e.g. a different value of DepthFlag compared to the current layer.].

The variable RpRefPicAvailFlag is set equal to (RpRefPicAvailFlagL0 | | RpRefPicAvailFlagL1).

When RpRefPicAvailFlag is equal to 1, the following applies for X in the range of 0 to 1, inclusive:

- When X is equal to 0 or the current slice is a B slice the following applies:
  - For i in the range of 0 to NumActiveRefLayerPics − 1, inclusive, the following applies:
    - The variable refViewIdx is set equal to ViewIdx(RefPicListX[i]).
    - The variable RefRpRefAvailFlagLX[refViewIdx] is set equal to 0.
    - When RpRefPicAvailFlagLX is equal to 1 and there is a picture picA in the DPB with PicOrderCnt(picA) equal to PicOrderCnt(RefPicListX[RpRefIdxLX]), ViewIdx(picA) equal to refViewIdx, DepthFlag(picA) equal to 0 and marked as "used for reference", RefRpRefAvailFlagLX[refViewIdx] is set equal to 1.

H.8.4 Decoding process for coding units coded in intra prediction mode

H.8.4.1 General decoding process for coding units coded in intra prediction mode

The specifications in subclause 8.4.1 apply with the following modification:

- All invocations of the process specified in subclause 8.4.2 are replaced with invocations of the process specified in subclause H.8.4.2.
- All invocations of the process specified in subclause 8.4.4.1 are replaced with invocations of the process specified in subclause H.8.4.4.1.

H.8.4.2 Derivation process for luma intra prediction mode

Input to this process is a luma location (xPb, yPb) specifying the top-left sample of the current luma prediction block relative to the top left luma sample of the current picture.

In this process, the luma intra prediction mode IntraPredModeY[xPb][yPb] is derived.

Table H-3 specifies the value for the intra prediction mode and the associated names.

Table H-3 – Specification of intra prediction mode and associated names

Intra prediction mode	Associated name
0	INTRA_PLANAR
1	INTRA_DC
2..34	INTRA_ANGULAR2..INTRA_ANGULAR34
35	INTRA_DMM_WFULL
36	INTRA_DMM_CPREDTEX

IntraPredModeY[xPb][yPb] labelled 0..34 represents directions of predictions as illustrated in Figure 8 1.

- If DepthIntraMode[xPb][yPb] is equal to INTRA_DEP_SDC_PLANAR, IntraPredModeY[xPb][yPb] is set equal to INTRA_PLANAR.
- Otherwise, if DepthIntraMode[xPb][yPb] is equal to INTRA_DEP_SDC_DMM_WFULL, IntraPredModeY[xPb][yPb] is set equal to INTRA_DMM_WFULL.
- Otherwise, if DepthIntraMode[xPb][yPb] is equal to INTRA_DEP_DMM_WFULL, IntraPredModeY[xPb][yPb] is set equal to INTRA_DMM_WFULL.
- Otherwise if DepthIntraMode[xPb][yPb] is equal to INTRA_DEP_DMM_CPREDTEX, IntraPredModeY[xPb][yPb] is set equal to INTRA_DMM_CPREDTEX.
- Otherwise (DepthIntraMode[xPb][yPb] is equal to INTRA_DEP_NONE), IntraPredModeY[xPb][yPb] is

derived as the following ordered steps:

1. The neighbouring locations (xNbA, yNbA) and (xNbB, yNbB) are set equal to (xPb − 1, yPb) and (xPb, yPb − 1), respectively.

2. For X being replaced by either A or B, the variables candIntraPredModeX are derived as follows:

   - The availability derivation process for a block in z-scan order as specified in subclause 6.4.2 is invoked with the location (xCurr, yCurr) set equal to (xPb, yPb) and the neighbouring location (xNbY, yNbY) set equal to (xNbX, yNbX) as inputs, and the output is assigned to availableX.

   - The candidate intra prediction mode candIntraPredModeX is derived as follows:

     - If availableX is equal to FALSE, candIntraPredModeX is set equal to INTRA_DC.

     - Otherwise, if CuPredMode[xNbX][yNbX] is not equal to MODE_INTRA or pcm_flag[xNbX][yNbX] is equal to 1, candIntraPredModeX is set equal to INTRA_DC,

     - Otherwise, if X is equal to B and yPb − 1 is less than ((yPb >> CtbLog2SizeY) << CtbLog2SizeY), candIntraPredModeB is set equal to INTRA_DC.

     - Otherwise, if candIntraPredModeX is larger than 34, candIntraPredModeX is set equal to INTRA_DC.

     - Otherwise, candIntraPredModeX is set equal to IntraPredModeY[xNbX][yNbX].

3. The candModeList[x] with x = 0..2 is derived as follows:

   - If candIntraPredModeB is equal to candIntraPredModeA, the following applies:

     - If candIntraPredModeA is less than 2 (i.e. equal to INTRA_PLANAR or INTRA_DC), candModeList[x] with x = 0..2 is derived as follows:

       candModeList[0] = INTRA_PLANAR (H-34)

       candModeList[1] = INTRA_DC (H-35)

       candModeList[2] = INTRA_ANGULAR26 (H-36)

     - Otherwise, candModeList[x] with x = 0..2 is derived as follows:

       candModeList[0] = candIntraPredModeA (H-37)

       candModeList[1] = 2 + ((candIntraPredModeA + 29) % 32) (H-38)

       candModeList[2] = 2 + ((candIntraPredModeA − 2 + 1) % 32) (H-39)

   - Otherwise (candIntraPredModeB is not equal to candIntraPredModeA), the following applies:

     - candModeList[0] and candModeList[1] are derived as follows:

       candModeList[0] = candIntraPredModeA (H-40)

       candModeList[1] = candIntraPredModeB (H-41)

     - If neither of candModeList[0] and candModeList[1] is equal to INTRA_PLANAR, candModeList[2] is set equal to INTRA_PLANAR,

     - Otherwise, if neither of candModeList[0] and candModeList[1] is equal to INTRA_DC, candModeList[2] is set equal to INTRA_DC,

     - Otherwise, candModeList[2] is set equal to INTRA_ANGULAR26.

4. IntraPredModeY[xPb][yPb] is derived by applying the following procedure:

   - If prev_intra_luma_pred_flag[xPb][yPb] is equal to 1, the IntraPredModeY[xPb][yPb] is set equal to candModeList[mpm_idx].

   - Otherwise, IntraPredModeY[xPb][yPb] is derived by applying the following ordered steps:

     1) The array candModeList[x], x = 0..2 is modified as the following ordered steps:

        i. When candModeList[0] is greater than candModeList[1], both values are swapped as follows:

           (candModeList[0], candModeList[1]) = Swap(candModeList[0], candModeList[1]) (H-42)

        ii. When candModeList[0] is greater than candModeList[2], both values are swapped as follows:

(candModeList[0], candModeList[2]) = Swap(candModeList[0], candModeList[2]) (H-43)

iii. When candModeList[1] is greater than candModeList[2], both values are swapped as follows:

(candModeList[1], candModeList[2]) = Swap(candModeList[1], candModeList[2]) (H-44)

2) IntraPredModeY[xPb][yPb] is derived by the following ordered steps:

i. IntraPredModeY[xPb][yPb] is set equal to rem_intra_luma_pred_mode[xPb][yPb].

ii. For i equal to 0 to 2, inclusive, when IntraPredModeY[xPb][yPb] is greater than or equal to candModeList[i], the value of IntraPredModeY[xPb][yPb] is incremented by one.

H.8.4.3 Derivation process for chroma intra prediction mode

The specifications in subclause 8.4.3 apply.

H.8.4.4 Decoding process for intra blocks

H.8.4.4.1 General decoding process for intra blocks

Inputs to this process are:

- a sample location (xTb0, yTb0) specifying the top-left sample of the current transform block relative to the top-left sample of the current picture,
- a variable log2TrafoSize specifying the size of the current transform block,
- a variable trafoDepth specifying the hierarchy depth of the current block relative to the coding unit,
- a variable predModeIntra specifying the intra prediction mode,
- a variable cIdx specifying the colour component of the current block.

Output of this process is a modified reconstructed picture before deblocking filtering.

The luma sample location (xTbY, yTbY) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture is derived as follows:

(xTbY, yTbY) = (cIdx = = 0) ? (xTb0, yTb0) : (xTb0 << 1, yTb0 << 1) (H-45)

The variable splitFlag is derived as follows:

- If cIdx is equal to 0, splitFlag is set equal to split_transform_flag[xTbY][yTbY][trafoDepth].
- Otherwise, if all of the following conditions are true, splitFlag is set equal to 1.
  - cIdx is greater than 0
  - split_transform_flag[xTbY][yTbY][trafoDepth] is equal to 1
  - log2TrafoSize is greater than 2
- Otherwise, splitFlag is set equal to 0.

Depending on the value of splitFlag, the following applies:

- If splitFlag is equal to 1, the following ordered steps apply:
  1. The variables xTb1 and yTb1 are derived as follows:
     - The variable xTb1 is set equal to xTb0 + (1 << (log2TrafoSize − 1)).
     - The variable yTb1 is set equal to yTb0 + (1 << (log2TrafoSize − 1)).
  2. The general decoding process for intra blocks as specified in this subclause is invoked with the location (xTb0, yTb0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.
  3. The general decoding process for intra blocks as specified in this subclause is invoked with the location (xTb1, yTb0), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.

4. The general decoding process for intra blocks as specified in this subclause is invoked with the location (xTb0, yTb1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.

5. The general decoding process for intra blocks as specified in this subclause is invoked with the location (xTb1, yTb1), the variable log2TrafoSize set equal to log2TrafoSize − 1, the variable trafoDepth set equal to trafoDepth + 1, the intra prediction mode predModeIntra, and the variable cIdx as inputs, and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (splitFlag is equal to 0), the following ordered steps apply:

1. The variable nTbS is set equal to 1 << log2TrafoSize.

2. The general intra sample prediction process as specified in subclause 8.4.4.2.1 is invoked with the transform block location (xTb0, yTb0), the intra prediction mode predModeIntra, the transform block size nTbS, and the variable cIdx as inputs, and the output is an (nTbS)x(nTbS) array predSamples.

3. Depending on SdcFlag[xTb0][xTb0], the following applies:

   – If SdcFlag[xTb0][yTb0] is equal to 0, following applies:

     – The scaling and transformation process as specified in subclause 8.6.2 is invoked with the luma location (xTbY, yTbY), the variable trafoDepth, the variable cIdx, and the transform size trafoSize set equal to nTbS as inputs, and the output is an (nTbS)x(nTbS) array resSamples.

     [Ed. (GT):Meeting notes say: "Implement an enabling flag at (position t.b.d.)". However there seems to be no decision on the position yet.]

     – If DltFlag[nuh_layer_id] is equal to 1 and predModeIntra is equal to INTRA_DC, INTRA_ANGULAR10, INTRA_ANGULAR26, INTRA_DMM_WFULL, INTRA_DMM_WPREDTEX, INTRA_DMM_CPREDTEX, or INTRA_CHAIN, the following applies, for i in the range of 0 to nTbS − 1, inclusive, and j in the range of 0 to nTbS − 1, inclusive:

$$\mathrm{idx} = \mathrm{DepthValue2Idx}[\ \mathrm{predSamples}[\ i\ ][\ j\ ]\ ] + \mathrm{resSamples}[\ i\ ][\ j\ ] \qquad \text{(H-46)}$$

$$S_L[\ xTb0 + i\ ][\ yTb0 + j\ ] = \mathrm{Idx2DepthValue}[\ \mathrm{clip3}(\ 0, \mathrm{NumDepthValuesInDlt}[\ \mathrm{nuh\_layer\_id}\ ] - 1, \mathrm{idx}\ )\ ] \qquad \text{(H-47)}$$

     – Otherwise, the following applies:

       – The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the transform block location (xTb0, yTb0), the transform block size nTbS, the variable cIdx, the (nTbS)x(nTbS) array predSamples, and the (nTbS)x(nTbS) array resSamples as inputs.

   – Otherwise (SdcFlag[xTb0][yTb0] is equal to 1) the following ordered steps apply:

     – The depth value reconstruction process as specified in subclause H.8.4.4.3 is invoked with the location (xTb0, yTb0), the transform size trafoSize set equal to nTbS, the (nTbS)x(nTbS) array predSamples, and the intra prediction mode predModeIntra, as the inputs and the output is a (nTbS)x(nTbS) array resSamples.

4. For x in the range of 0 to nTbS − 1 and y in the range of 0 to nTbS − 1, the following applies:

   – When cIdx is equal to 0, $\mathrm{ResSamples}_L$[xTb0 + x][yTb0 + y] is set equal to 0.

   – When cIdx is equal to 1, $\mathrm{ResSamples}_{Cb}$[xTb0 + x][yTb0 + y] is set equal to 0.

   – When cIdx is equal to 2, $\mathrm{ResSamples}_{Cr}$[xTb0 + x][yTb0 + y] is set equal to 0.

H.8.4.4.2 Intra sample prediction

H.8.4.4.2.1 General intra sample prediction

Inputs to this process are:

– a sample location (xTbCmp, yTbCmp) specifying the top-left sample of the current transform block relative to the top left sample of the current picture,

– a variable predModeIntra specifying the intra prediction mode,

– a variable nTbS specifying the transform block size,

- a variable cIdx specifying the colour component of the current block.

Output of this process is the predicted samples predSamples[x][y], with x, y = 0..nTbS − 1.

The nTbS * 4 + 1 neighbouring samples p[x][y] that are constructed samples prior to the deblocking filter process, with x = −1, y = −1..nTbS * 2 − 1 and x = 0..nTbS * 2 − 1, y = −1, are derived as follows:

- The neighbouring location (xNbCmp, yNbCmp) is specified by:

  $$(xNbCmp, yNbCmp) = (xTbCmp + x, yTbCmp + y) \qquad (8\ 27)$$

- The current luma location (xTbY, yTbY) and the neighbouring luma location (xNbY, yNbY) are derived as follows:

  $$(xTbY, yTbY) = (cIdx == 0) ? (xTbCmp, yTbCmp) : (xTbCmp << 1, yTbCmp << 1) \qquad (8\ 28)$$

  $$(xNbY, yNbY) = (cIdx == 0) ? (xNbCmp, yNbCmp) : (xNbCmp << 1, yNbCmp << 1) \qquad (8\ 29)$$

- The availability derivation process for a block in z-scan order as specified in subclause 6.4.1 is invoked with the current luma location (xCurr, yCurr) set equal to (xTbY, yTbY) and the neighbouring luma location (xNbY, yNbY) as inputs, and the output is assigned to availableN.
- Each sample p[x][y] is derived as follows:
  - If one or more of the following conditions are true, the sample p[x][y] is marked as "not available for intra prediction":
    - The variable availableN is equal to FALSE.
    - CuPredMode[xNbY][yNbY] is not equal to MODE_INTRA and constrained_intra_pred_flag is equal to 1.
  - Otherwise, the sample p[x][y] is marked as "available for intra prediction" and the sample at the location (xNbCmp, yNbCmp) is assigned to p[x][y].

When at least one sample p[x][y] with x = −1, y = −1..nTbS * 2 − 1 and x = 0..nTbS * 2 − 1, y = −1 is marked as "not available for intra prediction", the reference sample substitution process for intra sample prediction in subclause 8.4.4.2.2 is invoked with the samples p[x][y] with x = −1, y = −1..nTbS * 2 − 1 and x = 0..nTbS * 2 − 1, y = −1, nTbS, and cIdx as inputs, and the modified samples p[x][y] with x = −1, y = −1..nTbS * 2 − 1 and x = 0..nTbS * 2 − 1, y = −1 as output.

Depending on the value of predModeIntra, the following ordered steps apply:

1. When cIdx is equal to 0 and predModeIntra is in the range of 0 to 34, the filtering process of neighbouring samples specified in subclause 8.4.4.2.3 is invoked with the sample array p and the transform block size nTbS as inputs, and the output is reassigned to the sample array p.
2. The intra sample prediction process according to predModeIntra applies as follows:
   - If predModeIntra is equal to INTRA_PLANAR, the corresponding intra prediction mode specified in subclause 8.4.4.2.4 is invoked with the sample array p and the transform block size nTbS as inputs, and the output is the predicted sample array predSamples.
   - Otherwise, if predModeIntra is equal to INTRA_DC, the corresponding intra prediction mode specified in subclause 8.4.4.2.5 is invoked with the sample array p, the transform block size nTbS, and the colour component index cIdx as inputs, and the output is the predicted sample array predSamples.
   - Otherwise, if predModeIntra is in the range of INTRA_ANGULAR2..INTRA_ANGULAR34, the corresponding intra prediction mode specified in subclause 8.4.4.2.6 is invoked with the intra prediction mode predModeIntra, the sample array p, the transform block size nTbS, and the colour component index cIdx as inputs, and the output is the predicted sample array predSamples.
   - Otherwise, if predModeIntra is equal to INTRA_DMM_WFULL, the corresponding intra prediction mode specified in subclause H.8.4.4.2.7 is invoked with the location (xTbY, yTbY), the sample array p and the transform block size nTbS as the inputs and the output are the predicted sample array predSamples.
   - Otherwise, if predModeIntra is equal to INTRA_DMM_CPREDTEX, the corresponding intra prediction mode specified in subclause H.8.4.4.2.8 is invoked with the location (xTbY, yTbY), with the sample array p and the transform block size nTbS as the inputs and the output are the predicted sample array predSamples.

H.8.4.4.2.2 Reference sample substitution process for intra sample prediction

The specifications in subclause 8.4.4.2.2 apply.

H.8.4.4.2.3 Filtering process of neighbouring samples

The specifications in subclause 8.4.4.2.3 apply.

H.8.4.4.2.4 Specification of intra prediction mode INTRA_PLANAR

The specifications in subclause 8.4.4.2.4 apply.

H.8.4.4.2.5 Specification of intra prediction mode INTRA_DC

The specifications in subclause 8.4.4.2.5 apply.

H.8.4.4.2.6 Specification of intra prediction mode in the range of INTRA_ANGULAR2.. INTRA_ANGULAR34

The specifications in subclause 8.4.4.2.6 apply.

H.8.4.4.2.7 Specification of intra prediction mode INTRA_DMM_WFULL

Inputs to this process are:

- a sample location (xTb, yTb) specifying the top-left sample of the current block relative to the top-left sample of the current picture,
- the neighbouring samples p[x][y], with x = −1, y = −1..nTbS * 2 − 1 and x = 0..nTbS * 2 − 1, y = −1,
- a variable nTbS specifying the transform block size,

Output of this process is:

- the predicted samples predSamples[x][y], with x, y = 0..nTbS − 1.

The values of the prediction samples predSamples[x][y], with x, y = 0..nTbS − 1, are derived as specified by the following ordered steps:

1. The variable wedgePattern[x][y] with x, y =0..nTbS − 1, specifying a binary partition pattern is derived as.

   wedgePattern = WedgePatternTable[Log2(nTbS)][wedge_full_tab_idx[xTb][yTb]] (H-48)

2. The depth partition value derivation and assignment process as specified in subclause H.8.4.4.2.9 is invoked with the neighbouring samples p[x][y], the binary pattern wedgePattern[xTb][yTb], the transform size nTbS, the dcOffsetAvailFlag set equal to depth_dc_flag[xTb][yTb], and the DC Offsets DcOffset[xTb][yTb][0], and DcOffset[xTb][yTb][1] as inputs and the output is assigned to predSamples[x][y].
3. For x, y = 0..nTbs − 1, inclusive the following applies:
   - WedgeIdx[xTb + x][yTb + y] is set equal to wedge_full_tab_idx[xTb][yTb].

H.8.4.4.2.8 Specification of intra prediction mode INTRA_DMM_CPREDTEX

Inputs to this process are:

- a sample location (xTb, yTb) specifying the top-left sample of the current block relative to the top-left sample of the current picture,
- the neighbouring samples p[x][y], with x = −1, y = −1..nTbS * 2 − 1 and x = 0..nTbS * 2 − 1, y = −1,
- a variable nTbS specifying the transform block size,

Output of this process is:

- the predicted samples predSamples[x][y], with x, y = 0..nTbS − 1.

The values of the prediction samples predSamples[x][y], with x, y = 0..nTbS − 1, are derived as specified by the following ordered steps:

1. The variable recTextPic is set equal to the array of the reconstructed luma picture samples of TexturePic..
2. The variable textThresh specifying a threshold for the segmentation of recTextPic is derived as specified in the following.
   - The variable sumTextPicVals is set equal to 0.

- For x = 0..nTbS − 1 the following applies
  - For y = 0..nTbS − 1 the following applies

    sumTextPicVals += recTextPic[xTb + x][yTb + y] (H-49)

- The variable textThresh is set equal to (sumTextPicVals >> (2 * log2(nTbS)))

3. The variable wedgeletPattern[x][y] with x, y =0..nTbS − 1 specifying a binary partition pattern is derived as specified in the following.

   - For x = 0..nTbS − 1 the following applies
     - For y = 0..nTbS − 1 the following applies

       wedgeletPattern[x][y] = (recTextPic[xTb + x][yTb + y] > textThresh) (H-50)

4. The depth partition value derivation and assignment process as specified in subclause H.8.4.4.2.9 is invoked with the neighbouring samples p[x][y], the binary pattern wedgeletPattern[x][y], the transform size nT, the dcOffsetAvailFlag set equal to depth_dc_flag[xTb][yTb], and the DC Offsets DcOffset[xTb][yTb][0], and DcOffset[xTb][yTb][1] as inputs and the output is assigned to predSamples[x][y].

H.8.4.4.2.9 Depth partition value derivation and assignment process

Inputs to this process are:

- the neighbouring samples p[x][y], with x = −1, y = −1..nTbS * 2 − 1 and x = 0..nTbS * 2 − 1, y = −1,
- a binary array partitionPattern[x][y], with x, y =0..nTbS − 1, specifying a partitioning of the prediction block in a partition 0 and a partition 1.
- a variable nTbS specifying the transform block size,
- a flag dcOffsetAvailFlag, specifying whether DC Offset values are available,
- the variables dcOffsetP0 and dcOffsetP1, specifying the DC offsets for the block partitions

Output of this process is:

- the predicted samples predSamples[x][y], with x, y = 0..nTbS − 1.

The variables vertEdgeFlag and horEdgeFlag are derived as specified in the following:

vertEdgeFlag = (partitionPattern[0][0] != partitionPattern[nTbS − 1][0]) ? 1 : 0 (H-51)

horEdgeFlag = (partitionPattern[0][0] != partitionPattern[0][nTbS − 1]) ? 1 : 0 (H-52)

The variables dcVal0 and dcVal1 are derived as specified in the following:

- If vertEdgeFlag is equal to horEdgeFlag, the following applies:

  $$\text{dcValBR} = \text{horEdgeFlag} \; ? \; ((p[-1][\text{nTbS} - 1] + p[\text{nTbS} - 1][-1]) >> 1) : (1 << (\text{BitDepth}_Y - 1)) \quad \text{(H-53)}$$

  dcValLT = (p[−1][0] + p[0][−1]) >> 1 (H-54)

- Otherwise (horEdgeFlag is not equal to vertEdgeFlag), the following applies:

  dcValBR = horEdgeFlag ? p[−1][nTbS − 1] : p[nTbS − 1][−1] (H-55)

  dcValLT = horEdgeFlag ? p[(nTbS – 1) >> 1][−1] : p[−1][(nTbS – 1) >> 1] (H-56)

The predicted sample values predSamples[x][y] are derived as specified in the following:

- For x in the range of 0 to (nTbS − 1), inclusive the following applies:
  - For y in the range of 0 to (nTbS − 1), inclusive the following applies:
    - The variables predDcVal and dcOffset are derived as specified in the following:

      predDcVal = (partitionPattern[x][y] = = partitionPattern[0][0]) ? dcValLT : dcValBR (H-57)

      dcOffset = dcOffsetAvailFlag ? (partitionPattern[x][y] = = 0 ? dcOffsetP0 : dcOffsetP1) : 0 (H-58)

    - If DltFlag[nuh_layer_id] is equal to 0, the following applies:

      predSamples[x][y] = predDcVal + dcOffset (H-59)

- Otherwise (intraChainFlag is equal to 0), the following applies:

$$\text{predSamples}[\ x\ ][\ y\ ] = \text{Idx2DepthValue}[\ \text{DepthValue2Idx}[\ \text{predDcVal}\ ] + \text{dcOffset}\ ] \quad \text{(H-60)}$$

H.8.4.4.2.10 Specification of tables WedgePatternTable

NOTE 1 − Tables and values resulting from the processes specified in the following are independent of any information contained in the bitstream. Therefore the derivation process described in this subclause can be carried out once as part of the initialization of the decoding process. Alternatively, the tables and values can be stored within the decoder (read-only) memory as fixed lookup tables, such that the derivation process described in this section does not need to be implemented in the decoder at all.

The list WedgePatternTable[log2BlkSize] of binary partition patterns of size (1 << log2BlkSize)x(1 << log2BlkSize), the variable NumWedgePattern[log2BlkSize] specifying the number of binary partition patterns in list WedgePatternTable[log2BlkSize] are derived as specified in the following:

- For log2BlkSize ranging from 2 to Log2MaxDmmCbSize, inclusive, the following applies:
  - Depending on log2BlkSize, the variable resShift is derived as specified in Table H−4.

Table H−4 – Specification of resShift

log2BlkSize	**resShift**
2,3	1
4	0
Otherwise (5… Log2MaxDmmCbSize)	−1

  - The variable wBlkSize is set equal to (1 << (log2BlkSize + resShift))
  - For wedgeOri in the range of 0 to 5, inclusive, the following ordered steps apply.
    - Depending on wedgeOri the variables xPosS, yPosS, xPosE, yPosE, xIncS, yIncS, xIncE, yIncE are derived as specified in Table H−5.

Table H−5 – Specification of xPosS, yPosS, xPosE, yPosE, xIncS, yIncS, xIncE, yIncE

wedgeOri	0	1	2	3	4	5
xPosS	0	wBlkSize − 1	wBlkSize − 1	0	0	wBlkSize − 1
yPosS	0	0	wBlkSize − 1	wBlkSize − 1	0	0
xPosE	0	wBlkSize − 1	wBlkSize− 1	0	0	0
yPosE	0	0	wBlkSize − 1	wBlkSize − 1	wBlkSize − 1	0
xIncS	1	0	−1	0	1	0
yIncS	0	1	0	−1	0	1
xIncE	0	−1	0	1	1	0
yIncE	1	0	−1	0	0	1

    - For m in the range of 0 to wBlkSize − 1, inclusive, the following applies:
      - For n in the range of 0 to wBlkSize − 1, inclusive, the following applies:
        - The Wedgelet pattern generation process as specified in subclause H.8.4.4.2.10.1 is invoked with patternSize being equal to (1 << log2BlkSize), the variable resShift, variable wedgeOri, xS being equal to (xPosS + m * xIncS), yS being equal to (yPosS + m * yIncS), xE being equal to (xPosE + n * xIncE) and yE being equal to (yPosE + n * yIncE) as inputs and the output is the binary array curWedgePattern.
        - The wedgelet pattern list insertion process as specified in subclause H.8.4.4.2.10.2 is invoked with log2BlkSize, and the binary partition pattern curWedgePattern as inputs.

H.8.4.4.2.10.1 Wedgelet pattern generation process

Inputs to this process are:

- a variable patternSize specifying the binary partition pattern size,
- a resolution shift value resShift specifying the precision of the wedgelet partition start and end positions relative to patternSize,
- a variable wedgeOri specifying the orientation identifier of the wedgelet pattern,
- a variable xS specifying the partition line start horizontal position,
- a variable yS specifying the partition line start vertical position,
- a variable xE specifying the partition line end horizontal position,
- a variable yE specifying the partition line end vertical position.

Output of this process is:

- binary array wedgePattern[x][y] of size (patternSize)x(patternSize)

The variable curSize specifying the size of the current partition pattern is derived as follows:.

curSize = (resShift = = 1) ? (patternSize << 1) : patternSize (H-61)

When resShift is equal to −1 variables xS, yS, xE and yE are modified as specified in Table H−6.

Table H−6 – Specification of xS, yS, xE, yE

wedgeOri	xS	yS	xE	yE
0	xS << 1	yS << 1	xE << 1	yE << 1
1	curSize − 1	yS << 1	xE << 1	yE << 1
2	xS << 1	curSize − 1	curSize − 1	yE << 1
3	xS << 1	yS << 1	xE << 1	curSize − 1
4	xS << 1	yS << 1	xE << 1	curSize − 1
5	curSize − 1	yS << 1	xE << 1	yE << 1

The values of variable curPattern[x][y], are derived as specified by the following ordered steps.

1. For x, y = 0..curSize − 1, curPattern[x][y] is set equal to 0.
2. The samples of the array curPattern that form a line between (xS, yS) and (xE, yE) are set equal to 1 as specified in the following:

```
x0 = xS
y0 = yS
x1 = xE
y1 = yE
if( abs( yE − yS ) > abs( xE − xS ) ) {
    ( x0, y0 ) = Swap( x0, y0 )
    ( x1, y1 ) = Swap( x1, y1 )
}
if( x0 > x1 ) {
    ( x0, x1 ) = Swap( x0, x1 )
    ( y0, y1 ) = Swap( y0, y1 )
}
sumErr = 0
posY = y0
for( posX = x0; posX <= x1; posX +  + ) {
    if( abs( yE − yS ) > abs( xE − xS ) )
        curPattern[ posY ][ posX ] = 1
    else
        curPattern[ posX ][ posY ] = 1
    sumErr + = ( abs( y1 − y0 )  <<  1 )
    if( sumErr >= ( x1 − x0 ) ) {
        posY + = ( y0 < y1 ) ? 1 : −1
```

```
            sumErr −= ( x1 − x0 )  <<  1
        }
    }
```

3. The samples of curPattern belonging to the smaller partition are set equal to 1 as specified in the following:

```
if( wedgeOri  = =  0 )
    for( iX = 0; iX < xS; iX ++ )
        for( iY = 0; curPattern[ iX ][ iY ]  = =  0; iY++ )
            curPattern[ iX ][ iY ] = 1
else if( wedgeOri  = =  1 )
    for( iY = 0; iY < yS; iY++ )
        for( iX = curSize − 1; curPattern[ iX ][ iY ]  = =  0 ; iX−− )
            curPattern[ iX ][ iY ] = 1
else if( wedgeOri  = =  2 )
    for( iX = curSize − 1; iX > xS; iX−− )
        for( iY = curSize − 1; curPattern[ iX ][ iY ]  = =  0 ; iY−−)
            curPattern[ iX ][ iY ] = 1
else if( wedgeOri  = =  3 )
    for( iY = curSize − 1; iY > yS; iY−− )
        for( iX = 0; curPattern[ iX ][ iY ]  = =  0 ; iX++)
            curPattern[ iX ][ iY ] = 1
else if( wedgeOri  = =  4 ) && ( ( xS + xE ) < curSize ) )
    for( iY = 0; iY < curSize; iY ++ )
        for( iX = 0; curPattern[ iX ][ iY ]  = = 0  ; iX +)
            curPattern[ iX ][ iY ] = 1
else if( wedgeOri  = =  4 )
    for( iY = 0; iY < curSize; iY++ )
        for( iX = curSize − 1; curPattern[ iX ][ iY ]  = =  0 ; iX−−)
            curPattern[ iX ][ iY ] = 1
else if( wedgeOri  = =  5 )  &&  ( ( yS + yE ) < curSize ) )
    for( iX = 0; iX < curSize; iX ++ )
        for( iY = 0; curPattern[ iX ][ iY ] = = 0 ; iY++ )
            curPattern[ iX ][ iY ] = 1
else if( wedgeOri  = =  5 )
    for( iX = 0; iX < curSize; iX++ )
        for( iY = curSize − 1; curPattern[ iX ][ iY ]  = =  0 ; iY−− )
            curPattern[ iX ][ iY ] = 1
```

4. The binary partition pattern wedgePattern[x][y], with x, y = 0..patternSize − 1, is derived as specified in the following.

- If resShift is equal to 1, the following applies.
  - Depending on wedgeOri, the variables xOff and yOff are set as specified in Table H−7.

Table H−7 Specification of xOff, yOff

wedgeOri	(xS + xE) < curSize	xOff	yOff
0		0	0
1		1	0
2		1	1
3		0	1
4	0	1	0
4	1	0	0
5	0	0	1
5	1	0	0

  - For x, y = 0 ..patternSize − 1 the following applies:

$$wedgePattern[\ x\ ][\ y\ ] = curPattern[\ (\ x\ \ll\ 1\ ) + xOff\ ][\ (\ y\ \ll\ 1\ ) + yOff\ ] \quad \text{(H-62)}$$

- Otherwise (resShift is not equal to 1), wedgePattern is set equal to curPattern.

H.8.4.4.2.10.2 Wedgelet pattern list insertion process

Inputs to this process are:

- a variable log2BlkSize specifying the binary partition pattern size as (1 << log2BlkSize),
- binary partition pattern wedgePattern[x][y], with x, y =0..(1 << log2BlkSize) − 1.

The variable isValidFlag specifying whether the binary partition pattern wedgePattern is added to the list WedgePatternTable[log2BlkSize] not is set equal to 0.

The value of isValidFlag is derived as specified by the following ordered steps.

1. For x, y = 0..(1 << log2BlkSize) − 1 the following applies.
   - When wedgePattern[x][y] is not equal to wedgePattern[0][0] the flag isValidFlag is set to 1.
2. For k = 0..NumWedgePattern[log2BlkSize] − 1 the following applies.
   - The flag patIdenticalFlag is set equal to 1.
   - For x, y =0..(1 << log2BlkSize)−1 the following applies.
     - When wedgePattern[x][y] is not equal to WedgePatternTable[log2BlkSize][k][x][y], patIdenticalFlag is set to 0.
   - When patIdenticalFlag is equal to 1, isValidFlag is set to 0.
3. For k = 0..NumWedgePattern[log2BlkSize] − 1 the following applies.
   - The flag patInvIdenticalFlag is set to 1.
   - For x, y =0..(1 << log2BlkSize) − 1 the following applies.
     - When wedgePattern[x][y] is equal to WedgePatternTable[log2BlkSize][k][x][y], patInvIdenticalFlag is set to 0.
   - When patInvIdenticalFlag is equal to 1, isValidFlag is set to 0.

When isValidFlag is equal to 1, the following applies.

- The pattern WedgePatternTable[log2BlkSize][NumWedgePattern[log2BlkSize]] is set equal to wedgePattern.
- The value of NumWedgePattern[log2BlkSize] is increased by one.

H.8.4.4.3 Depth value reconstruction process

Inputs to this process are:

- a luma location (xTb, yTb) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current picture,
- a variable nTbS specifying the transform block size,
- predicted samples predSamples[x][y], with x, y =0..nTbS − 1
- the intra prediction mode predModeIntra,

Output of this process is:

- reconstructed depth value samples resSamples[x][y], with x, y = 0.. nTbS − 1.

Depending on predModeIntra the array wedgePattern[x][y] with x, y =0..nTbS − 1 specifying the binary segmentation pattern is derived as follows.

- If predModeIntra is equal to INTRA_DMM_WFULL, the following applies.

  wedgePattern = WedgePatternTable[Log2(nTbS)][wedge_full_tab_idx[xTb][yTb]]

- Otherwise (predModeIntra is not equal to INTRA_DMM_WFULL), the following applies.
  - For x, y = 0..nTbS − 1 wedgePattern[x][y] is set equal to 0.

Depending on DltFlag[nuh_layer_id] the reconstructed depth value samples resSamples[x][y] are derived as specified in the following:

- If DltFlag[nuh_layer_id] is equal to 0, the following applies:
  - For x, y = 0..nTbS − 1, the reconstructed depth value samples resSamples[x][y] are derived as specified in the following:

$$\text{resSamples}[\,x\,][\,y\,] = \text{predSamples}[\,x\,][\,y\,] + \text{DcOffset}[\,\text{xTb}\,][\,\text{yTb}\,][\,\text{wedgePattern}[\,x\,][\,y\,]\,] \quad \text{(H-63)}$$

- Otherwise (DltFlag[nuh_layer_id] is equal to 1), the following applies:
  - The variables dcPred[0] and dcPred[1] are derived as specified in the following:
    - If predModeIntra is equal to INTRA_DC, the following applies:

$$\text{dcPred}[\,0\,] = \text{predSamples}[\,\text{nTbS} - 1\,][\,\text{nTbS} - 1\,] \quad \text{(H-64)}$$

    - Otherwise, if predModeIntra is equal to INTRA_PLANAR, the following applies:

$$\text{dcPred}[\,0\,] = (\,\text{predSamples}[\,0\,][\,0\,] + \text{predSamples}[\,0\,][\,\text{nTbS} - 1\,] + \text{predSamples}[\,\text{nTbS} - 1\,][\,0\,] + \text{predSamples}[\,\text{nTbS} - 1\,][\,\text{nTbS} - 1\,] + 2\,) >> 2 \quad \text{(H-65)}$$

    - Otherwise, (predModeIntra is equal to INTRA_DMM_WFULL), the following applies.

$$\text{dcPred}[\,\text{wedgePattern}[\,0\,][\,0\,]\,] = \text{predSamples}[\,0\,][\,0\,] \quad \text{(H-66)}$$

$$\text{dcPred}[\,\text{wedgePattern}[\,\text{nTbS} - 1\,][\,0\,]\,] = \text{predSamples}[\,\text{nTbS} - 1\,][\,0\,] \quad \text{(H-67)}$$

$$\text{dcPred}[\,\text{wedgePattern}[\,0\,][\,\text{nTbS} - 1\,]\,] = \text{predSamples}[\,0\,][\,\text{nTbS} - 1\,] \quad \text{(H-68)}$$

$$\text{dcPred}[\,\text{wedgePattern}[\,\text{nTbS} - 1\,][\,\text{nTbS} - 1\,]\,] = \text{predSamples}[\,\text{nTbS} - 1\,][\,\text{nTbS} - 1\,] \quad \text{(H-69)}$$

  - For x, y = 0..nTbS − 1, the reconstructed depth value samples resSamples[x][y] are derived as specified in the following:

$$\text{dltIdxPred} = \text{DepthValue2Idx}[\,\text{dcPred}[\,\text{wedgePattern}[\,x\,][\,y\,]\,]\,] \quad \text{(H-70)}$$

$$\text{dltIdxResi} = \text{DcOffset}[\,\text{xTb}\,][\,\text{yTb}\,][\,\text{wedgePattern}[\,x\,][\,y\,]\,] \quad \text{(H-71)}$$

$$\text{resSamples}[\,x\,][\,y\,] = \text{predSamples}[\,x\,][\,y\,] + \text{Idx2DepthValue}[\,\text{dltIdxPred} + \text{dltIdxResi}\,] - \text{dcPred}[\,\text{wedgePattern}[\,x\,][\,y\,]\,] \quad \text{(H-72)}$$

H.8.5 Decoding process for coding units coded in inter prediction mode

H.8.5.1 General decoding process for coding units coded in inter prediction mode

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable log2CbSize specifying the size of the current coding block.

Output of this process is a modified reconstructed picture before deblocking filtering.

The derivation process for quantization parameters as specified in subclause 8.6.1 is invoked with the luma location (xCb, yCb) as input.

The variable nCbS_L is set equal to 1 << log2CbSize and the variable nCbS_C is set equal to 1 << (log2CbSize − 1).

The decoding process for coding units coded in inter prediction mode consists of following ordered steps:

1. When iv_mv_pred_flag[nuh_layer_id] is equal to 1, or iv_res_pred_flag[nuh_layer_id] is equal to 1 or view_synthesis_pred_flag[nuh_layer_id] is equal to 1, following applies:
   - If DepthFlag is equal to 0 the derivation process for disparity vectors as specified in subclause H.8.5.5 is invoked with the luma locations (xCb, yCb), the coding block size nCbS_L as the inputs.
   - Otherwise (DepthFlag is equal to 1), the derivation process for disparity vectors from neighbouring depth samples as specified in subclause H.8.5.6 is invoked with the luma locations (xCb, yCb), the coding block size nCbS_L as the inputs.
2. The inter prediction process as specified in subclause H.8.5.3 is invoked with the luma location (xCb, yCb) and the luma coding block size log2CbSize as inputs, and the outputs are three arrays predSamples_L, predSamples_{Cb}, and predSamples_{Cr}.
3. The decoding process for the residual signal of coding units coded in inter prediction mode specified in subclause H.8.5.3.3.9 is invoked with the luma location (xCb, yCb) and the luma coding block size log2CbSize

as inputs, and the outputs are three arrays resSamples_L, resSamples_{Cb}, and resSamples_{Cr}.

4. The reconstructed samples of the current coding unit are derived as follows:
   - The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the luma coding block location (xCb, yCb), the variable nCurrS set equal to nCbS_L, the variable cIdx set equal to 0, the $(\text{nCbS}_L)\text{x}(\text{nCbS}_L)$ array predSamples set equal to predSamples_L, and the $(\text{nCbS}_L)\text{x}(\text{nCbS}_L)$ array resSamples set equal to resSamples_L as inputs.
   - The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the chroma coding block location (xCb / 2, yCb / 2), the variable nCurrS set equal to nCbS_C, the variable cIdx set equal to 1, the $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array predSamples set equal to predSamples_{Cb}, and the $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array resSamples set equal to resSamples_{Cb} as inputs.
   - The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the chroma coding block location (xCb / 2, yCb / 2), the variable nCurrS set equal to nCbS_C, the variable cIdx set equal to 2, the $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array predSamples set equal to predSamples_{Cr}, and the $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array resSamples set equal to resSamples_{Cr} as inputs.

H.8.5.2 Inter prediction process

The specifications in subclause 8.5.2 apply with the following modification:

– All invocations of the process specified in subclause 8.5.3 are replaced with invocations of the process specified in subclause H.8.5.3.

H.8.5.3 Decoding process for prediction units in inter prediction mode

H.8.5.3.1 General

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xBl, yBl) specifying the top-left sample of the current luma prediction block relative to the top-left sample of the current luma coding block,

– a variable nCbS specifying the size of the current luma coding block,

– a variable nPbW specifying the width of the current luma prediction block,

– a variable nPbH specifying the width of the current luma prediction block,

– a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are:

– an (nCbSL)x(nCbSL) array predSamplesL of luma prediction samples, where nCSL is derived as specified below,

– an (nCbSC)x(nCbSC) array predSamplesCb of chroma prediction samples for the component Cb, where nCSC is derived as specified below,

– an (nCbSC)x(nCbSC) array predSamplesCr of chroma prediction samples for the component Cr, where nCSC is derived as specified below.

The variable nCbSL is set equal to nCbS and the variable nCbSC is set equal to nCbS >> 1.

The decoding process for prediction units in inter prediction mode consists of the following ordered steps:

1. The derivation process for motion vector components and reference indices as specified in subclause H.8.5.3.2 is invoked with the luma coding block location (xCb, yCb), the luma prediction block location (xBl, yBl), the luma coding block size block nCbS, the luma prediction block width nPbW, the luma prediction block height nPbH, and the prediction unit index partIdx as inputs, and the luma motion vectors mvL0 and mvL1, the chroma motion vectors mvCL0 and mvCL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1 as outputs.
2. Depending on subPbMotionFlag, the following applies:
   - If subPbMotionFlag is equal to 0, the decoding process for inter sample prediction as specified in subclause H.8.5.3.3.1 is invoked with the luma coding block location (xCb, yCb), the luma prediction block location (xBl, yBl), the luma coding block size block nCbS, the luma prediction block width nPbW, the luma prediction block height nPbH, the luma motion vectors mvL0 and mvL1, the chroma motion

vectors mvCL0 and mvCL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1 as inputs, and the inter prediction samples (predSamples) that are an (nCbSL)x(nCbSL) array predSamplesL of prediction luma samples and two (nCbSC)x(nCbSC) arrays predSamplesCr and predSamplesCr of prediction chroma samples, one for each of the chroma components Cb and Cr, as outputs.

- Otherwise (subPbMotionFlag is equal to 1), the decoding process for sub prediction block wise inter sample prediction as specified in subclause H.8.5.3.3.9 is invoked with the luma coding block location (xCb, yCb), the luma prediction block location (xBl, yBl), the luma coding block size block nCbS, the luma prediction block width nPbW, the luma prediction block height nPbH as inputs, and the inter prediction samples (predSamples) that are an (nCbSL)x(nCbSL) array predSamplesL of prediction luma samples and two (nCbSC)x(nCbSC) arrays predSamplesCr and predSamplesCr of prediction chroma samples, one for each of the chroma components Cb and Cr, as outputs.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made for x = xBl..xBl + nPbW − 1 and y = yBl..yBl + nPbH − 1:

MvL0[xCb + x][yCb + y] = subPbMotionFlag ? SubPbMvL0[xCb + x][yCb + y] : mvL0 (H-73)

MvL1[xCb + x][yCb + y] = subPbMotionFlag ? SubPbMvL1[xCb + x][yCb + y] : mvL1 (H-74)

RefIdxL0[xCb + x][yCb + y] = subPbMotionFlag ? SubPbRefIdxL0[xCb + x][yCb + y] : refIdxL0 (H-75)

RefIdxL1[xCb + x][yCb + y] = subPbMotionFlag ? SubPbRefIdxL1[xCb + x][yCb + y] : refIdxL1 (H-76)

PredFlagL0[xCb + x][yCb + y] = subPbMotionFlag ? SubPbPredFlagL0[xCb + x][yCb + y] : predFlagL0(H-77)

PredFlagL1[xCb + x][yCb + y] = subPbMotionFlag ? SubPbPredFlagL1[xCb + x][yCb + y] : predFlagL1(H-78)

H.8.5.3.2 Derivation process for motion vector components and reference indices

Inputs to this process are:

- a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a luma location (xBl, yBl) of the top-left sample of the current luma prediction block relative to the top-left sample of the current luma coding block,
- a variable nCbS specifying the size of the current luma coding block,
- two variables nPbW and nPbH specifying the width and the height of the luma prediction block,
- a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are:

- the luma motion vectors mvL0 and mvL1,
- the chroma motion vectors mvCL0 and mvCL1,
- the reference indices refIdxL0 and refIdxL1,
- the prediction list utilization flags predFlagL0 and predFlagL1.
- the flag subPbMotionFlag, specifying, whether the motion data of the current PU has sub prediction block size motion accuracy.

Let (xPb, yPb) specify the top-left sample location of the current luma prediction block relative to the top-left luma sample of the current picture where xPb = xCb + xBl and yPb = yCb + yBl.

Let the variable currPic and ListX be the current picture and RefPicListX, with X being 0 or 1, of the current picture, respectively.

The function LongTermRefPic(aPic, aPb, refIdx, LX), with X being 0 or 1, is defined as follows:

- If the picture with index refIdx from reference picture list LX of the slice containing prediction block aPb in the picture aPic was marked as "used for long term reference" at the time when aPic was the current picture, LongTermRefPic(aPic, aPb, refIdx, LX) is equal to 1.
- Otherwise, LongTermRefPic(aPic, aPb, refIdx, LX) is equal to 0.

The variables vspModeFlag, ivpMvFlagL0, ivpMvFlagL1 and subPbMotionFlag are set equal to 0.

For the derivation of the variables mvL0 and mvL1, refIdxL0 and refIdxL1, as well as predFlagL0 and predFlagL1, the following applies:

- If merge_flag[xPb][yPb] is equal to 1, the derivation process for luma motion vectors for merge mode as specified in subclause H.8.5.3.2.1 is invoked with the luma location (xCb, yCb), the luma location (xPb, yPb), the variables nCbS, nPbW, nPbH, and the partition index partIdx as inputs, and the output being the luma motion vectors mvL0, mvL1, the reference indices refIdxL0, refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1,, the disparity vector availability flags ivpMvFlagL0 and ivpMvFlagL1, the flag vspModeFlag, and the flag subPbMotionFlag.
- Otherwise, for X being replaced by either 0 or 1 in the variables predFlagLX, mvLX, and refIdxLX, in PRED_LX, and in the syntax elements ref_idx_lX and MvdLX, the following applies:
  1. The variables refIdxLX and predFlagLX are derived as follows:
     - If inter_pred_idc[xPb][yPb] is equal to PRED_LX or PRED_BI,

       $$\text{refIdxLX} = \text{ref\_idx\_lX[xPb][yPb]} \quad \text{(H-79)}$$

       $$\text{predFlagLX} = 1 \quad \text{(H-80)}$$
     - Otherwise, the variables refIdxLX and predFlagLX are specified by:

       $$\text{refIdxLX} = -1 \quad \text{(H-81)}$$

       $$\text{predFlagLX} = 0 \quad \text{(H-82)}$$
  2. The variable mvdLX is derived as follows:

     $$\text{mvdLX[0]} = \text{MvdLX[xPb][yPb][0]} \quad \text{(H-83)}$$

     $$\text{mvdLX[1]} = \text{MvdLX[xPb][yPb][1]} \quad \text{(H-84)}$$
  3. When predFlagLX is equal to 1, the derivation process for luma motion vector prediction in subclause 8.5.3.2.5 is invoked with the luma coding block location (xCb, yCb), the coding block size nCbS, the luma prediction block location (xPb, yPb), the variables nPbW, nPbH, refIdxLX, and the partition index partIdx as inputs, and the output being mvpLX.
  4. When predFlagLX is equal to 1, the luma motion vector mvLX is derived as follows:

     $$\text{uLX[0]} = (\text{mvpLX[0]} + \text{mvdLX[0]} + 2^{16}) \,\%\, 2^{16} \quad \text{(H-85)}$$

     $$\text{mvLX[0]} = (\text{uLX[0]} >= 2^{15}) \,?\, (\text{uLX[0]} - 2^{16}) : \text{uLX[0]} \quad \text{(H-86)}$$

     $$\text{uLX[1]} = (\text{mvpLX[1]} + \text{mvdLX[1]} + 2^{16}) \,\%\, 2^{16} \quad \text{(H-87)}$$

     $$\text{mvLX[1]} = (\text{uLX[1]} >= 2^{15}) \,?\, (\text{uLX[1]} - 2^{16}) : \text{uLX[1]} \quad \text{(H-88)}$$

     NOTE – The resulting values of mvLX[0] and mvLX[1] as specified above will always be in the range of -2^{15} to $2^{15} - 1$, inclusive.

When ChromaArrayType is not equal to 0 and predFlagLX, with X being 0 or 1, is equal to 1, the derivation process for chroma motion vectors in subclause 8.5.3.2.9 is invoked with mvLX as input, and the output being mvCLX.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made for x = xPb.. (xPb + nPbW − 1), y = yPb..(yPb + nPbH− 1) (with X being either 0 or 1):

$$\text{IvpMvFlagLX[x][y]} = \text{ivpMvFlagLX} \quad \text{(H-89)}$$

$$\text{VspModeFlag[x][y]} = \text{vspModeFlag} \quad \text{(H-90)}$$

H.8.5.3.2.1 Derivation process for luma motion vectors for merge mode

This process is only invoked when merge_flag[xPb][yPb] is equal to 1, where (xPb, yPb) specify the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture.

Inputs to this process are:

- a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a luma location (xPb, yPb) of the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,
- a variable nCbS specifying the size of the current luma coding block,

- two variables nPbW and nPbH specifying the width and the height of the luma prediction block,
- a variable partIdx specifying the index of the current prediction unit within the current coding unit.

Outputs of this process are:

- the luma motion vectors mvL0 and mvL1,
- the reference indices refIdxL0 and refIdxL1,
- the prediction list utilization flags predFlagL0 and predFlagL1,
- the disparity vector availability flags ivpMvFlagL0 and ivpMvFlagL1,
- the flag vspModeFlag, specifying, whether the current PU is coded using view synthesis prediction,
- the flag subPbMotionFlag, specifying, whether the motion data of the current PU has sub prediction block size motion accuracy.

[Ed. (GT): In particular two things need to be check in this process: 1.) Are the limits on candidates in the list correct (e.g. MaxNumMergeCand vs. 5 + NumExtraMergeCand) 2.) Is (xOrigP, yOrigP) and (xPb, yPb) used correctly in all places?]

The function differentMotion(N, M) is specified as follows:

- If one of the following conditions is true, differentMotion(N, M) is equal to 1:
  - predFlagLXN != predFlagLXM (with X being replaced by 0 and 1),
  - mvLXN != mvLXM (with X being replaced by 0 and 1),
  - refIdxLXN != refIdxLXM (with X being replaced by 0 and 1),
- Otherwise, differentMotion(N, M) is equal to 0.

The motion vectors mvL0 and mvL1, the reference indices refIdxL0 and refIdxL1, and the prediction utilization flags predFlagL0 and predFlagL1 are derived by the following ordered steps:

1. The derivation process for the base merge candidate list as specified in subclause H.8.5.3.2.18 is invoked with the luma location (xCb, yCb), the luma location (xPb, yPb), the variables nCbS, nPbW, nPbH, and the partition index partIdx as inputs, and the output being a modified luma location (xPb, yPb), the modified variables nPbW and nPbH, the modified variable partIdx, the luma location (xOrigP, yOrigP), the variables nOrigPbW and nOrigPbH, the merge candidate list baseMergeCandList, the luma motion vectors mvL0N and mvL1N, the reference indices refIdxL0N and refIdxL1N, and the prediction list utilization flags predFlagL0N and predFlagL1N, with N being replaced by all elements of baseMergeCandList.
2. For N being replaced by A_1, B_1, B_0, A_0 and B_2, the following applies:
   - If N is an element in baseMergeCandList, availableFlagN is set equal to 1.
   - Otherwise (N is not an element in baseMergeCandList), availableFlagN is set equal to 0.
3. Depending on iv_mv_pred_flag[nuh_layer_id], the following applies:
   - If iv_mv_pred_flag[nuh_layer_id] is equal to 0, the flags availableFlagIvMC, availableIvMCShift and availableFlagIvDC are set equal to 0.
   - Otherwise (iv_mv_pred_flag[nuh_layer_id] is equal to 1), the derivation process for the inter-view merge candidates as specified in subclause H.8.5.3.2.10 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, as the inputs and the output is assigned to the availability flags availableFlagIvMC, availableIvMCShift and availableFlagIvDC, the reference indices refIdxLXIvMC, refIdxLXIvMCShift and refIdxLXIvDC, the prediction list utilization flags predFlagLXIvMC, predFlagLXivMCShift and predFlagLXIvDC, and the motion vectors mvLXIvMC, mvLXIvMCShift and mvLXIvDC (with X being 0 or 1, respectively).
4. Depending on view_synthesis_pred_flag[nuh_layer_id], the following applies:
   - If view_synthesis_pred_flag[nuh_layer_id] is equal to 0, the flag availableFlagVSP is set equal to 0.
   - Otherwise (view_synthesis_pred_flag[nuh_layer_id] is equal to 1), the derivation process for a view synthesis prediction merge candidate as specified in subclause H.8.5.3.2.13 is invoked with the luma locations (xCb, yCb) as input and the outputs are the availability flag availableFlagVSP, the reference indices refIdxL0VSP and refIdxL1VSP, the prediction list utilization flags predFlagL0VSP and predFlagL1VSP, and the motion vectors mvL0VSP and mvL1VSP.

5. Depending on mpi_flag[nuh_layer_id], the following applies:
   - If mpi_flag[nuh_layer_id] is equal to 0, the variable availableFlagT is set equal to 0.
   - Otherwise (mpi_flag[nuh_layer_id] is equal to 1), the derivation process for the texture merging candidate as specified in subclause H.8.5.3.2.14 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH as the inputs and the outputs are the flag availableFlagT, the prediction utilization flags predFlagL0T and predFlagL1T, the reference indices refIdxL0T and refIdxL1T, and the motion vectors mvL0T and mvL1T.

6. The merging candidate list, extMergeCandList, is constructed as follows:

```
i = 0
if( availableFlagT )
    extMergeCandList[ i++ ] = T
if( availableFlagIvMC  &&  ( !availableFlagT  ||  differentMotion( T, IvMC ) ) )
    extMergeCandList[ i++ ] = IvMC
N = DepthFlag ? T : IvMC
if( $availableFlagA_1$  &&  ( !availableFlagN  ||  differentMotion( N, $A_1$ ) ) )
    extMergeCandList[ i++ ] = $A_1$
if( $availableFlagB_1$  &&  ( !availableFlagN  ||  differentMotion( N, $B_1$ ) ) )
    extMergeCandList[ i++ ] = $B_1$
if( $availableFlagB_0$ )
    extMergeCandList[ i++ ] = $B_0$                                        (H-91)
if( availableFlagIvDC  &&  ( !$availableFlagA_1$  ||  differentMotion( $A_1$, IvDC ) )  &&
                    ( !$availableFlagB_1$  ||  differentMotion( $B_1$, IvDC ) ) )
    extMergeCandList[ i++ ] = IvDC
if( availableFlagVSP  &&  !ic_flag  &&  iv_res_pred_weight_idx  = =  0 )
    extMergeCandList[ i++ ] = VSP
if( $availableFlagA_0$ )
    extMergeCandList[ i++ ] = $A_0$
if( $availableFlagB_2$ )
    extMergeCandList[ i++ ] = $B_2$
if( availableFlagIvMCShift  &&  i < ( 5 + NumExtraMergeCand )  &&
                    ( !availableFlagIvMC  ||  differentMotion( IvMC, IvMCShift ) ) )
    extMergeCandList[ i++ ] = IvMCShift
```

7. The variable availableFlagIvDCShift is set equal to 0, and when availableFlagIvMCShift is equal to 0, and i is less than (5 + NumExtraMergeCand), the derivation process for the shifted disparity merging candidate as specified in subclause H.8.5.3.2.15 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, and the availability flags availableFlagN, the reference indices refIdxL0N and refIdxL1N, the prediction list utilization flags predFlagL0N and predFlagL1N, the motion vectors mvL0N and mvL1N, of every candidate N being in extMergeCandList, extMergeCandList, and i as the inputs and the outputs are the flag availableFlagIvDCShift, the prediction utilization flags predFlagL0IvDCShift and predFlagL1IvDCShift, the reference indices refIdxL0IvDCShift and refIdxL1IvDCShift, and the motion vectors mvL0IvDCShift and mvL1IvDCShift.

8. The merging candidate list, extMergeCandList, is constructed as follows:

```
if( availableFlagIvDCShift )
    extMergeCandList[ i++ ] = IvDCShift
j = 0
while( i < MaxNumMergeCand ) {                                             (H-92)
    N = baseMergeCandList[ j++ ]
    if( N  !=  $A_1$  &&  N  !=  $B_1$  &&  N  !=  $B_0$  &&  N  !=  $A_0$  &&  N  !=  $B_2$ )
        extMergeCandList[ i++ ] = N
}
```

9. The variable N is set equal to extMergeCandList[merge_idx[xOrigP][yOrigP]].

10. The variable subPbMotionFlag is set equal to (N = = IvMC).

11. The following assignments are made with X being replaced by 0 or 1:

mvLX = subPbMotionFlag ? 0 : mvLXN (H-93)

refIdxLX = subPbMotionFlag ? −1 : refIdxLXN (H-94)

predFlagLX = subPbMotionFlag ? 0 : predFlagLXN (H-95)

12. When predFlagL0 is equal to 1 and predFlagL1 is equal to 1, and (nOrigPbW + nOrigPbH) is equal to 12, the following applies

    refIdxL1 = −1 (H-96)

    predFlagL1 = 0 (H-97)

13. The derivation process for a view synthesis prediction flag as specified in subclause H.8.5.3.2.17 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, the merge candidate indicator N as the inputs, and the output is the mergeCandIsVspFlag.

14. The variable vspModeFlag is derived as specified in the following:

    vspModeFlag = mergeCandIsVspFlag && !ic_flag && (iv_res_pred_weight_idx = = 0) (H-98)

15. The disparity availability flag ivpMvFlagLX is derived as follows (with X being replace by 0 or 1).

    - If DepthFlag is equal to 0 and one of the following conditions is true, ivpMvFlagLX is set equal to 1

      [Ed. (GT) There is some redundancy in draft and software since disparities equal for both lists.(#7)]

      - predFlagLXIvMC && extMergeCandList[merge_idx[xPb][yPb]] = = IvMC
      - predFlagLXIvMCShift && extMergeCandList[merge_idx[xPb][yPb]] = = IvMCShift

      [Ed. (GT): PredMode[xCb][yCb] = = MODE_SKIP might be added here instead of testing it in the disparity vector derivation process]

    - Otherwise, ivpMvFlagLX is set equal to 0.

H.8.5.3.2.2 Derivation process for spatial merging candidates

The specifications in subclause 8.5.3.2.2 apply.

H.8.5.3.2.3 Derivation process for combined bi-predictive merging candidates

The specifications in subclause 8.5.3.2.3 apply.

H.8.5.3.2.4 Derivation process for zero motion vector merging candidates

The specifications in subclause 8.5.3.2.4 apply.

H.8.5.3.2.5 Derivation process for luma motion vector prediction

The specifications in subclause 8.5.3.2.5 apply.

H.8.5.3.2.6 Derivation process for motion vector predictor candidates

The specifications in subclause 8.5.3.1.6 apply.

H.8.5.3.2.7 Derivation process for temporal luma motion vector prediction

The specifications in subclause 8.5.3.2.7 apply, with the following modifications:

- All invocations of the process specified in subclause 8.5.3.2.8 are replaced with invocations of the process specified in subclause H.8.5.3.2.8.

H.8.5.3.2.8 Derivation process for collocated motion vectors

Inputs to this process are:

- a variable currPb specifying the current prediction block,
- a variable colPic specifying the collocated picture,
- a variable colPb specifying the collocated prediction block inside the collocated picture specified by colPic,
- a luma location (xColPb, yColPb) specifying the top-left sample of the collocated luma prediction block specified by colPb relative to the top-left luma sample of the collocated picture specified by colPic,
- a reference index refIdxLX, with X being 0 or 1.

Outputs of this process are:

- the motion vector prediction mvLXCol,

- the availability flag availableFlagLXCol.

The variable currPic specifies the current picture.

The arrays predFlagLXCol[x][y], mvLXCol[x][y], and refIdxLXCol[x][y] are set equal to the corresponding arrays of the collocated picture specified by colPic, PredFlagLX[x][y], MvLX[x][y], and RefIdxLX[x][y], respectively, with X being the value of X this process is invoked for.

The variables mvLXCol and availableFlagLXCol are derived as follows:

- If colPb is coded in an intra prediction mode, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.
- Otherwise, the motion vector mvCol, the reference index refIdxCol, and the reference list identifier listCol are derived as follows:
  - If predFlagL0Col[xColPb][yColPb] is equal to 0, mvCol, refIdxCol, and listCol are set equal to mvL1Col[xColPb][yColPb], refIdxL1Col[xColPb][yColPb], and L1, respectively.
  - Otherwise, if predFlagL0Col[xColPb][yColPb] is equal to 1 and predFlagL1Col[xColPb][yColPb] is equal to 0, mvCol, refIdxCol, and listCol are set equal to mvL0Col[xColPb][yColPb], refIdxL0Col[xColPb][yColPb], and L0, respectively.
  - Otherwise (predFlagL0Col[xColPb][yColPb] is equal to 1 and predFlagL1Col[xColPb][yColPb] is equal to 1), the following assignments are made:
    - If DiffPicOrderCnt(aPic, currPic) is less than or equal to 0 for every picture aPic in every reference picture list of the current slice, mvCol, refIdxCol, and listCol are set equal to mvLXCol[xColPb][yColPb], refIdxLXCol[xColPb][yColPb] and LX, respectively.
    - Otherwise, mvCol, refIdxCol, and listCol are set equal to mvLNCol[xColPb][yColPb], refIdxLNCol[xColPb][yColPb], and LN, respectively, with N being the value of collocated_from_l0_flag.

  and mvLXCol and availableFlagLXCol are derived as follows:

  - The variables curIvFlag and colIvFlag, specifying whether inter-view prediction is utilized for the current and collocated PU are derived as:

$$\text{curIvFlag} = \text{LongTermRefPic}(\text{ currPic, currPb, refIdxLX, LX }) \tag{H-99}$$

$$\text{colIvFlag} = \text{LongTermRefPic}(\text{ colPic, colPb, refIdxCol, listCol }) \tag{H-100}$$

  - When MvHevcCompatibilityFlag is equal to 0, curIvFlag is not equal to colIvFlag, and AltRefIdxLX is not equal to −1, the variables refIdxCol and colIvFlag are modified as follows:

$$\text{refIdxLX} = \text{AltRefIdxLX} \tag{H-101}$$

$$\text{curIvFlag} = \text{LongTermRefPic}(\text{ currPic, currPb, refIdxLX, LX }) \tag{H-102}$$

  - The motion vector mvLXCol is modified as follows.
  - If curIvFlag is not equal to colIvFlag, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.
  - Otherwise, the variable availableFlagLXCol is set equal to 1, refPicListCol[refIdxCol] is set to be the picture with reference index refIdxCol in the reference picture list listCol of the slice containing prediction block currPb in the picture colPic, and the following applies:
    - The variables colDiff and currDiff specifying a POC or ViewId difference are derived as follows.
      - If curIvFlag is equal to 0 or ((ViewIdx != 0) && iv_mv_scaling_flag) is equal to 0, the following applies.

$$\text{colDiff} = \text{DiffPicOrderCnt}(\text{ colPic, refPicListCol[refIdxCol] }) \tag{H-103}$$

$$\text{currDiff} = \text{DiffPicOrderCnt}(\text{ currPic, RefPicListX[refIdxLX] }) \tag{H-104}$$

      - Otherwise, (curIvFlag is equal to 1 and ((ViewIdx != 0) && iv_mv_scaling_flag) is equal to 1), the following applies.

$$\text{colDiff} = \text{DiffViewId}(\text{ colPic, refPicListCol[refIdxCol] }) \tag{H-105}$$

$$\text{currDiff} = \text{DiffViewId}(\text{ currPic, RefPicListX[refIdxLX] }) \tag{H-106}$$

- If colDiff is equal to currDiff, mvLXCol is derived as follows:

$$\text{mvLXCol} = \text{mvCol} \quad \text{(H-107)}$$

- Otherwise, mvLXCol is derived as a scaled version of the motion vector mvCol as follows:

$$\text{tx} = (16384 + (\text{Abs}(\text{td}) >> 1)) / \text{td} \quad \text{(H-108)}$$

$$\text{distScaleFactor} = \text{Clip3}(-4096, 4095, (\text{tb} * \text{tx} + 32) >> 6) \quad \text{(H-109)}$$

$$\text{mvLXCol} = \text{Clip3}(-32768, 32767, \text{Sign}(\text{distScaleFactor} * \text{mvCol}) * ((\text{Abs}(\text{distScaleFactor} * \text{mvCol}) + 127) >> 8)) \quad \text{(H-110)}$$

where td and tb are derived as follows:

$$\text{td} = \text{Clip3}(-128, 127, \text{colDiff}) \quad \text{(H-111)}$$

$$\text{tb} = \text{Clip3}(-128, 127, \text{currDiff}) \quad \text{(H-112)}$$

H.8.5.3.2.9 Derivation process for chroma motion vectors

The specifications in subclause 8.5.2.1.9 apply.

H.8.5.3.2.10 Derivation process for inter-view merge candidates

This process is not invoked when iv_mv_pred_flag[nuh_layer_id] is equal to 0.

Inputs to this process are:

- a luma location (xPb, yPb) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit,

Outputs of this process are (with X being 0 or 1, respectively)

- the availability flags availableFlagIvMC, availableFlagIvMCShift and availableFlagIvDC specifying whether the inter-view merge candidates are available,
- the reference indices refIdxLXIvMC, refIdxLXIvMCShift and refIdxLXIvDC,
- the prediction list utilization flags predFlagLXIvMC, predFlagLXIvMCShift and predFlagLXIvDC,
- the motion vectors mvLXIvMC, mvLXIvMCShift and mvLXIvDC,

The temporal inter-view motion vector merging candidate is derived by the following ordered steps.

1. The derivation process for a sub prediction block temporal inter-view motion vector candidate as specified in subclause H.8.5.3.2.16 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, the view order index RefViewIdx[xPb][yPb] and the disparity vector MvRefinedDisp[xPb][yPb] as the inputs and the outputs are, with X being in the range of 0 to 1, inclusive, the flag availableFlagLXIvMC, the motion vector mvLXIvMC and the reference index refIdxLXIvMC.
2. The availability flag availableFlagIvMC, and the prediction utilization flags predFlagL0IvMC and predFlagL1IvMC are derived by

$$\text{availableFlagIvMC} = \text{availableFlagL0IvMC} \,||\, \text{availableFlagL1IvMC} \quad \text{(H-113)}$$

$$\text{predFlagL0IvMC} = \text{availableFlagL0IvMC} \quad \text{(H-114)}$$

$$\text{predFlagL1IvMC} = \text{availableFlagL1IvMC} \quad \text{(H-115)}$$

The shifted temporal inter-view motion vector merging candidate is derived by the following ordered steps.

1. For the prediction list indication X being 0 and 1 the following applies.
   - The derivation process for a temporal inter-view motion vector candidate as specified in subclause H.8.5.3.2.11 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, the prediction list indication X , the view order index RefViewIdx[xPb][yPb], the disparity vector MvRefinedDisp[xPb][yPb] + (nPbW *2 + 4, nPbH *2 + 4), and the reference index refIdxLX being equal to −1, and as the inputs and the outputs are the flag availableFlagLXIvMCShift, the motion vector mvLXIvMCShift and the reference index refIdxLXIvMCShift.
2. The availability flag availableFlagIvMCShift, and the prediction utilization flags predFlagL0IvMCShift and predFlagL1IvMCShift are derived by

$$\text{availableFlagIvMCShift} = \text{availableFlagL0IvMCShift} \,||\, \text{availableFlagL1IvMCShift} \quad \text{(H-116)}$$

$$\text{predFlagL0IvMCShift} = \text{availableFlagL0IvMCShift} \quad \text{(H-117)}$$

$$\text{predFlagL1IvMCShift} = \text{availableFlagL1IvMCShift} \quad \text{(H-118)}$$

The disparity inter-view motion vector merging candidate is derived by the following ordered steps.

1. For the prediction list indication X being 0 and 1 the following applies.
   - The derivation process for a disparity inter-view motion vector candidate as specified in subclause H.8.5.3.2.12 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, the view order index RefViewIdx[xPb][yPb], the disparity vector MvRefinedDisp[xPb][yPb], and the prediction list indication X, as the inputs and the outputs are the flag availableFlagLXIvDC, the motion vector mvLXIvDC, and the reference index refIdxLXIvDC.
2. The availability flag availableFlagIvDC, and the prediction utilization flags predFlagL0IvDC and predFlagL1IvDC are derived by

$$\text{availableFlagIvDC} = \text{availableFlagL0IvDC} \,||\, \text{availableFlagL1IvDC} \quad \text{(H-119)}$$

$$\text{predFlagL0IvDC} = \text{availableFlagL0IvDC} \quad \text{(H-120)}$$

$$\text{predFlagL1IvDC} = \text{availableFlagL1IvDC} \quad \text{(H-121)}$$

H.8.5.3.2.11 Derivation process for a temporal inter-view motion vector candidate

This process is not invoked when iv_mv_pred_flag[nuh_layer_id] is equal to 0.

Inputs to this process are:

- a luma location (xPb, yPb) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit,
- a prediction list indication X,
- a reference view index refViewIdx.
- a disparity vector mvDisp,

Outputs of this process are:

- a flag availableFlagLXInterView specifying whether the temporal inter-view motion vector candidate is available,
- a temporal inter-view motion vector candidate mvLXInterView,
- a reference index refIdxLX specifying a reference picture in the reference picture list RefPicListLX,

The flag availableFlagLXInterView is set equal to 0, the variable refIdxLX is set equal to –1, and both components of mvLXInterView are set equal to 0.

When X is equal to 1 and the current slice is not a B slice the whole decoding process specified in this subclause terminates.

The reference layer luma location (xRef, yRef) is derived by

$$\text{xRefFull} = \text{xPb} + (\ \text{nPbW} >> 1\ ) + (\ (\ \text{mvDisp}[\ 0\ ] + 2\ ) >> 2\ ) \quad \text{(H-122)}$$

$$\text{yRefFull} = \text{yPb} + (\ \text{nPbH} >> 1\ ) + (\ (\ \text{mvDisp}[\ 1\ ] + 2\ ) >> 2\ ) \quad \text{(H-123)}$$

$$\text{xRef} = \text{Clip3}(\ 0, \text{PicWidthInSamples}_L - 1, (\ \text{xRefFull} >> 3\ ) << 3\ ) \quad \text{(H-124)}$$

$$\text{yRef} = \text{Clip3}(\ 0, \text{PicHeightInSamples}_L - 1, (\ \text{yRefFull} >> 3\ ) << 3\ ) \quad \text{(H-125)}$$

The variable ivRefPic is set equal to the picture with ViewIdx equal to refViewIdx in the current access unit.

The variable ivRefPb specifies the luma prediction block covering the location given by (xRef, yRef) inside the inter-view reference picture specified by ivRefPic.

The luma location (xIvRefPb, yIvRefPb) is set equal to the top-left sample of the inter-view reference luma prediction block specified by ivRefPb relative to the top-left luma sample of the inter-view reference picture specified by ivRefPic.

When ivRefPb is not coded in an intra prediction mode, the following applies, for Y in the range of X to (1 – X), inclusive:

- The variables refPicListLYIvRef, predFlagLYIvRef[x][y], mvLYIvRef[x][y], and refIdxLYIvRef[x][y] are set equal to the corresponding variables of the inter-view reference picture specified by ivRefPic, RefPicListLY,, PredFlagLY[x][y], MvLY[x][y], and RefIdxLY[x][y], respectively.
- When predFlagLYIvRef[xIvRefPb][yIvRefPb] is equal to 1, the following applies for each i from 0 to num_ref_idx_lX_active_minus1, inclusive:
  - When PicOrderCnt(refPicListLYIvRef[refIdxLYIvRef[xIvRefPb][yIvRefPb]]) is equal to PicOrderCnt(RefPicListLX[i]) and availableFlagLXInterView is equal to 0, the following applies.

availableFlagLXInterView = 1 (H-126)

mvLXInterView = mvLYIvRef[xIvRefPb][yIvRefPb] (H-127)

refIdxLX = i (H-128)

H.8.5.3.2.12 Derivation process for a disparity inter-view motion vector candidate

This process is not invoked when iv_mv_pred_flag[nuh_layer_id] is equal to 0.

Inputs to this process are:

- a luma location (xPb, yPb) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit,
- a prediction list indication X,
- a reference view index refViewIdx,
- a disparity vector mvDisp,

Outputs of this process are:

- a flag availableFlagLXInterView specifying whether the disparity inter-view motion vector candidate is available,
- a disparity inter-view motion vector candidate mvLXInterView,
- a reference index refIdxLX specifying a reference picture in the reference picture list RefPicListLX.

The flag availableFlagLXInterView is set equal to 0, both components of mvLXInterView are set equal to 0.

When X is equal to 1 and the current slice is not a B slice the whole decoding process specified in this subclause terminates.

For each i from 0 to num_ref_idx_lX_active_minus1, inclusive, the following applies:

- When PicOrderCnt(RefPicListX[i]) is equal to the PicOrderCntVal, ViewIdx(RefPicListX[i]) is equal to refViewIdx and availableFlagLXInterView is equal to 0 the following applies:

availableFlagLXInterView = 1 (H-129)

mvLXInterView[0] = DepthFlag ? (mvDisp[0] + 2) >> 2 : mvDisp[0] (H-130)

mvLXInterView[1] = 0 (H-131)

refIdxLX = i (H-132)

H.8.5.3.2.13 Derivation process for a view synthesis prediction merge candidate

Inputs to this process are:

- a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

Outputs of this process are

- the availability flag availableFlagVSP whether the VSP merge candidate is available,
- the reference indices refIdxL0VSP and refIdxL1VSP ,
- the prediction list utilization flags predFlagL0VSP and predFlagL1VSP,
- the motion vectors mvL0VSP and mvL1VSP.

The variable availableFlagVSP is set equal to 1, the variables predFlagL0VSP and predFlagL1VSP are set equal to 0, the variables refIdxL0VSP and refIdxL1VSP are set equal to −1 and the variable refViewAvailableFlag is set equal to 0.

- For X in the range of 0 to 1, inclusive, the following applies:
  - For i in the range of 0 to NumRefPicsLX – 1, inclusive, the following applies:
    - When refViewAvailableFlag is equal to 0 and ViewIdx(RefPicListX[i]) is equal to RefViewIdx[xCb][yCb], the following applies:

$$\text{refViewAvailableFlag} = 1 \tag{H-133}$$

$$\text{predFlagLXVSP} = 1 \tag{H-134}$$

$$\text{mvLXVSP} = \text{MvDisp}[\ \text{xCb}\ ][\ \text{yCb}\ ] \tag{H-135}$$

$$\text{refIdxLXVSP} = \text{i} \tag{H-136}$$

$$\text{Y} = 1 - \text{X} \tag{H-137}$$

When the current slice is a B slice and refViewAvailableFlag is equal to 1, refViewAvailableFlag is set equal to 0 and the following applies:

- For i in the range of 0 to NumRefPicsLY – 1, inclusive, the following applies.
  - When refViewAvailableFlag is equal to 0 and ViewIdx(RefPicListY[i]) is not equal to RefViewIdx[xCb][yCb] and ViewIdx(RefPicListY[i]) is not equal to ViewIdx, the following applies:
    - The variables refViewAvailableFlag, predFlagLYVSP, mvLYVSP, and refIdxLYVSP are derived as specified in the following:

$$\text{refViewAvailableFlag} = 1 \tag{H-138}$$

$$\text{predFlagLYVSP} = 1 \tag{H-139}$$

$$\text{mvLYVSP} = \text{MvDisp}[\ \text{xCb}\ ][\ \text{yCb}\ ] \tag{H-140}$$

$$\text{refIdxLYVSP} = \text{i} \tag{H-141}$$

    - When iv_mv_scaling_flag is equal to 1, mvLYVSP is modified as specified in the following:

$$\text{td} = \text{Clip3}(\ -128, 127, \text{ViewId} - \text{view\_id\_val}[\ \text{RefViewIdx}[\ \text{xCb}\ ][\ \text{yCb}\ ]\ )\ ] \tag{H-142}$$

$$\text{tb} = \text{Clip3}(\ -128, 127, \text{ViewId} - \text{ViewId}(\ \text{RefPicListY}[\ \text{i}\ ]\ )\ ) \tag{H-143}$$

$$\text{tx} = (\ 16384 + (\ \text{Abs}(\ \text{td}\ ) \gg 1\ )\ ) / \text{td} \tag{H-144}$$

$$\text{distScaleFactor} = \text{Clip3}(\ -4096, 4095, (\ \text{tb} * \text{tx} + 32\ ) \gg 6\ ) \tag{H-145}$$

$$\begin{aligned}\text{mvLYVSP}[\ 0\ ] = \text{Clip3}(\ &-32768, 32767, \text{Sign2}(\ \text{distScaleFactor} * \text{mvLYVSP}[\ 0\ ]\ ) \\ &* (\ (\text{Abs}(\ \text{distScaleFactor} * \text{mvLYVSP}[\ 0\ ]\ ) + 127\ ) \gg 8\ )\ )\end{aligned} \tag{H-146}$$

$$\begin{aligned}\text{mvLYVSP}[\ 1\ ] = \text{Clip3}(\ &-32768, 32767, \text{Sign2}(\ \text{distScaleFactor} * \text{mvLYVSP}[\ 1\ ]\ ) \\ &* (\ (\text{Abs}(\ \text{distScaleFactor} * \text{mvLYVSP}[\ 1\ ]\ ) + 127\ ) \gg 8\ )\ )\end{aligned} \tag{H-147}$$

H.8.5.3.2.14 Derivation process for the texture merging candidate

This process is not invoked when mpi_flag[nuh_layer_id] is equal to 0.

Inputs to this process are:

- a luma location (xPb, yPb) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit,

Outputs of this process are:

- a flag availableFlagT specifying whether the texture merging candidate is available,
- the prediction utilization flags predFlagL0T and predFlagL1T,
- the reference indices refIdxL0T and refIdxL1T (when availableFlagT is equal to 1),
- the motion vectors mvL0T and mvL1T (when availableFlagT is equal to 1).

The variable availableFlagT is set equal to 0. The variables predFlagL0T and predFlagL1T are set equal to 0. The variables refIdxL0T and refIdxL1T are set equal to −1. Both components of the motion vectors mvL0T and mvL1T are set equal to 0.

The texture luma location (xRef, yRef) is derived by:

$$\text{xRefFull} = \text{xPb} + ((\text{nPbW} - 1) >> 1) \quad \text{(H-148)}$$

$$\text{yRefFull} = \text{yPb} + ((\text{nPbH} - 1) >> 1) \quad \text{(H-149)}$$

$$\text{xRef} = \text{Clip3}(0, \text{PicWidthInSamples}_L - 1, (\text{xRefFull} >> 3) << 3) \quad \text{(H-150)}$$

$$\text{yRef} = \text{Clip3}(0, \text{PicHeightInSamples}_L - 1,(\text{yRefFull} >> 3) << 3) \quad \text{(H-151)}$$

[Ed. (GT): Is clipping necessary?]

Let textPic be the picture with PicOrderCntVal and ViewIdx equal to PicOrderCnt and ViewIdx of the current picture and DepthFlag being equal to 0 and let textPU be the PU at covering the position (xRef, yRef) in textPic.

For X in the range of 0 to 1, inclusive, the following applies:

1. The variable textPredFlagLX is set equal to PredFlagLX of textPU. The variable textRefIdxLX is set equal to RefIdxLX of textPU. The variable textMvLX is set equal to the MvLX of textPU. The variable availableFlag is set equal to 0.
2. When X is equal to 0 or the current slice is a B slice, for i in the range of 0 to NumRefPicsLX − 1, inclusive, the following applies:
   - When all of the following conditions are true, availableFlag is set equal to 1,
     - textPredFlagLX[xRef][yRef] is equal to 1
     - PicOrderCnt(RefPicListX[i]) is equal to PicOrderCnt(textPic)
     - ViewIdx(RefPicListX[i]) is equal to ViewIdx(textPic)
   - When predFlagLXT is equal to 0 and availableFlag is equal to 1, the following applies:

$$\text{mvLXT}[0] = (\text{textMvLX}[\text{xRef}][\text{yRef}][0] + 2) >> 2 \quad \text{(H-152)}$$

$$\text{mvLXT}[1] = (\text{textMvLX}[\text{xRef}][\text{yRef}][1] + 2) >> 2 \quad \text{(H-153)}$$

$$\text{refIdxLX} = \text{i} \quad \text{(H-154)}$$

$$\text{predFlagLXT} = 1 \quad \text{(H-155)}$$

$$\text{availableFlagT} = 1 \quad \text{(H-156)}$$

H.8.5.3.2.15 Derivation process for the shifted disparity merging candidate

This process is not invoked when DepthFlag is equal to 1.

Inputs to this process are:

- a luma location (xPb, yPb) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit,
- the availability flags availableFlagN,
- the reference indices refIdxL0N and refIdxL1N,
- the prediction list utilization flags predFlagL0N and predFlagL1N,
- the motion vectors mvL0N and mvL1N,
- a merging candidate list mergeCandList,
- the variable numMergeCand specifying the number of merge candidates in list mergeCandList

Outputs of this process are:

- the flag availableFlagIvDCShift, specifying whether shifted disparity merging candidate is available
- the prediction utilization flags predFlagL0IvDCShift and predFlagL1IvDCShift,
- the reference indices refIdxL0IvDCShift and refIdxL1IvDCShift,
- the motion vectors mvL0IvDCShift and mvL1IvDCShift.

The variable availableFlagIvDCShift is set equal to 0 and for i in the range of 0 to numMergeCand - 1, inclusive, the following applies:

- The variable N is set equal to mergeCandList[i].
- The derivation process for a view synthesis prediction flag as specified in subclause H.8.5.3.2.17 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, the merge candidate indicator N as the inputs, and the output is the mergeCandIsVspFlag.
- When availableFlagIvDCShift is equal to 0 and availableFlagN is equal to 1, the candidate N is not equal to IvMC or IvDC, and mergeCandIsVspFlag is not equal to 0, predFlagL0N is equal to 1 and ViewIdx(RefPicList0[refIdxL0N]) is not equal to ViewIdx, the following applies:
  - availableFlagIvDCShift is set equal to 1
  - predFlagLXIvDCShift is set equal to predFlagLXN, (with X being replaced by 0 and 1)
  - refIdxLXIvDCShift is set equal to refIdxLXN, (with X being replaced by 0 and 1)
  - mvL0IvDCShift[0] is set equal to mvL0N[0] + 4
  - mvL0IvDCShift[1] is set equal to (view_synthesis_pred_flag[nuh_layer_id] ? 0 : mvL0N[1])
  - mvL1IvDCShift = mvL1N

When availableFlagIvDCShift is equal to 0 and availableFlagIvDC is equal to 1, availableFlagIvDCShift is set to 1 and the following applies for X being 0 to 1, inclusive:

- predFlagLXIvDCShift is set equal to predFlagLXIvDC,
- refIdxLXIvDCShift is set equal to refIdxLXIvDC,
- mvLXIvDCShift[0] is set equal to mvL0IvDC[0] + 4
- mvLXIvDCShift[1] is set equal to mvL0IvDC[1]

H.8.5.3.2.16 Derivation process for a sub prediction block temporal inter-view motion vector candidate

This process is not invoked when iv_mv_pred_flag[nuh_layer_id] is equal to 0.

Inputs to this process are:

- a luma location (xPb, yPb) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit,
- a reference view index refViewIdx.
- a disparity vector mvDisp,

Outputs of this process are:

- the flags availableFlagLXInterView, with X in the range of 0 to 1, inclusive, specifying whether the temporal inter-view motion vector candidate is available,
- the temporal inter-view motion vector candidate mvLXInterView, with X in the range of 0 to 1, inclusive.
- the reference index refIdxLXInterView, with X in the range of 0 to 1, inclusive, specifying a reference picture in the reference picture list RefPicListLX,

For X in the range of 0 to 1, inclusive, the following applies:

- The flag availableFlagLXInterView is set equal to 0.
- The motion vector mvLXInterView is set equal to (0, 0).
- The reference index refIdxLXInterView is set equal to –1.

The variables nSbW and nSbH are derived as:

$$\text{nSbW} = \text{nPbW} / \text{SubPbSize}[\ \text{nuh\_layer\_id}\ ] \ <= \ 1\ ?\ \text{nPbW} : \text{SubPbSize}[\ \text{nuh\_layer\_id}\ ] \tag{H-157}$$

$$\text{nSbH} = \text{nPbH} / \text{SubPbSize}[\ \text{nuh\_layer\_id}\ ] \ <= \ 1\ ?\ \text{nPbH} : \text{SubPbSize}[\ \text{nuh\_layer\_id}\ ] \tag{H-158}$$

The variable ivRefPic is set equal to the picture with ViewIdx equal to refViewIdx in the current access unit, the variable curSubBlockIdx is set equal to 0 and the variable lastAvailableFlag is set equal to 0.

For yBlk in the range of 0 to (nPbH / nSbH – 1), inclusive, the following applies:

- For xBlk in the range of 0 to (nPbW / nSbW – 1), inclusive, the following applies:
  - For X in the range of 0 to 1, inclusive, the derivation process for a temporal inter-view motion vector candidate as specified in subclause H.8.5.3.2.11 is invoked with the luma location (xPb + xBlk*nSbW, yPb + yBlk * nSbH), the variables nSbW and nSbH, the prediction list indication X, the view order index refViewIdx, and the disparity vector mvDisp as the inputs and the outputs are the flag spPredFlagLX[xBlk][yBlk], the motion vector spMvLX[xBlk][yBlk] and the reference index spRefIdxLX[xBlk][yBlk].
  - The variable curAvailableFlag is set equal to (spRefIdxL0[xBlk][yBlk] || spRefIdxL1[xBlk][yBlk]).
  - Depending on curAvailableFlag, the following applies:
    - If curAvailableFlag is equal to 1, the following ordered steps apply:
      1. When lastAvailableFlag is equal to 0, the following applies:
         - For X in the range of 0 to 1, inclusive, the following applies:

           mvLXInterView = spMvLX[xBlk][yBlk] (H-159)

           refIdxLXInterView = spRefIdxLX[xBlk][yBlk] (H-160)

           availableFlagLXInterView = spPredFlagLX[xBlk][yBlk] (H-161)

         - When curSubBlockIdx is greater than 0, the following applies for k in the range of 0 to (curSubBlockIdx − 1), inclusive:
           - The variables i and k are derived as specified in the following:

             i = k % (nPbW / nSbW) (H-162)

             j = k / (nPbW / nSbW) (H-163)

           - For X in the range of 0 to 1, inclusive, the following applies:

             spMvLX[i][j] = spMvLX[xBlk][yBlk] (H-164)

             spRefIdxLX[i][j] = spRefIdxLX[xBlk][yBlk] (H-165)

             spPredFlagLX[i][j] = spPredFlagLX[xBlk][yBlk] (H-166)

      2. The variable lastAvailableFlag is set equal to 1.
      3. The variables xLastAvail and yLastAvail are set equal to xBlk and yBlk, respectively.
    - Otherwise (curAvailableFlag is equal to 0), when lastAvailable Flag is equal to 1, the following applies for X in the range of 0 to 1, inclusive:

      spMvLX[xBlk][yBlk] = spMvLX[xLastAvail][yLastAvail] (H-167)

      spRefIdxLX[xBlk][yBlk] = spRefIdxLX[xLastAvail][yLastAvail] (H-168)

      spPredFlagLX[xBlk][yBlk] = spPredFlagLX[xLastAvail][yLastAvail] (H-169)

  - The variable curSubBlockIdx is set equal to curSubBlockIdx + 1.

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made for x = 0.. nPbW − 1 and y = 0.. nPbH − 1:

- For X in the range of 0 to 1, inclusive, the following applies:
  - The variables SubPbPredFlagLX, SubPbMvLX and SubPbRefIdxLX are derived as specified in following:

    SubPbPredFlagLX[xPb + x][yPb + y] = spPredFlagLX[x / nSbW][y / nSbW] (H-170)

    SubPbMvLX[xPb + x][yPb + y] = spMvLX[x / nSbW][y / nSbW] (H-171)

    SubPbRefIdxLX[xPb + x][yPb + y] = spRefIdxLX[x / nSbW][y / nSbW] (H-172)

  - The derivation process for chroma motion vectors in subclause 8.5.3.2.9 is invoked with SubPbMvLX[xPb + x][yPb + y] as input and the output is SubPbMvCLX[xPb + x][yPb + y].

H.8.5.3.2.17 Derivation process for a view synthesis prediction flag

Inputs to this process are:

- a luma location (xPb, yPb) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit,
- a merge candidate indicator N, specifying the merge candidate.

Outputs of this process are:

- a variable mergeCandIsVspFlag specifying, whether the merge candidate is a view synthesis prediction merge candidate.

The variable mergeCandIsVspFlag is derived as specified in the following:

- If N is equal to VSP, mergeCandIsVspFlag is set equal to 1,
- Otherwise, if N is equal to A_1, mergeCandIsVspFlag is set equal to VspModeFlag[xPb − 1][yPb + nPbH − 1],
- Otherwise, if N is equal to A_0, mergeCandIsVspFlag is set equal to VspModeFlag[xPb − 1][yPb + nPbH],
- Otherwise, mergeCandIsVspFlag is set equal to 0.

H.8.5.3.2.18 Derivation process for the base merge candidate list

The specifications in subclause 8.5.3.2.1 apply, with the following modifications:

- Steps 9 and 10 are removed.
- “When slice_type is equal to B, the derivation process for combined bi-predictive merging candidates” is replaced by “When slice_type is equal to B and numMergeCand is less than 5, the derivation process for combined bi-predictive merging candidates”
- “temporal luma motion vector prediction in subclause 8.5.3.2.7 is invoked” is replaced by “temporal luma motion vector prediction in subclause H.8.5.3.2.7 is invoked”
- The outputs of the process are replaced by:
  - a modified luma location (xPb, yPb) of the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,
  - two variables nPbW and nPbH specifying the modified width and the height of the luma prediction block,
  - a modified variable partIdx specifying the modified index of the current prediction unit within the current coding unit.
  - an original luma location (xOrigP, yOrigP) of the top-left sample of the current luma prediction block relative to the top-left luma sample of the current picture,
  - two variables nOrigPbW and nOrigPbH specifying the original width and the height of the luma prediction block,
  - the merge candidate list, mergeCandList,
  - the luma motion vectors mvL0N and mvL1N, with N being replaced by all entries of mergeCandList
  - the reference indices refIdxL0N and refIdxL1N, with N being replaced by all entries of mergeCandList
  - the prediction list utilization flags predFlagL0N and predFlagL1N, with N being replaced by all elements of mergeCandList

H.8.5.3.3 Decoding process for inter prediction samples

H.8.5.3.3.1 General

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a luma location (xBl, yBl) specifying the top-left sample of the current luma prediction block relative to the top-left sample of the current luma coding block,

– a variable nCbS specifying the size of the current luma coding block,

– two variables nPbW and nPbH specifying the width and the height of the luma prediction block,

– the luma motion vectors mvL0 and mvL1,

– the chroma motion vectors mvCL0 and mvCL1,

– the reference indices refIdxL0 and refIdxL1,

– the prediction list utilization flags, predFlagL0, and predFlagL1.

Outputs of this process are:

– an $(nCbS_L)x(nCbS_L)$ array $predSamples_L$ of luma prediction samples, where $nCbS_L$ is derived as specified below,

– an $(nCbS_C)x(nCbS_C)$ array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where $nCbS_C$ is derived as specified below,

– an $(nCbS_C)x(nCbS_C)$ array $predSamples_{Cr}$ of chroma residual samples for the component Cr, where $nCbS_C$ is derived as specified below.

The variable $nCbS_L$ is set equal to nCbS and the variable $nCbS_C$ is set equal to nCbS >> 1.

– If VspModeFlag[xCb + xBl][yCb + yBl] is equal to 0, the following ordered steps apply:

1. Let $predSamplesL0_L$ and $predSamplesL1_L$ be (nPbW)x(nPbH) arrays of predicted luma sample values and $predSampleL0_{Cb}$, $predSampleL1_{Cb}$, $predSampleL0_{Cr}$, and $predSampleL1_{Cr}$ be (nPbW / 2)x(nPbH / 2) arrays of predicted chroma sample values.

2. For X being each of 0 and 1, when predFlagLX is equal to 1, the following applies:

   – When predFlagLX is equal to 1, the following applies.

     – The variable resPredFlag is derived as specified in the following: [Ed. (CY): Based on F0123, the only check for resPredFlag is the iv_res_pred_weight_idx, however F105 introduces other checks for ARP, which may apply to temporal residual prediction. The additional checks in H-195 need to be closely inspected.]

       resPredFlag = (iv_res_pred_weight_idx != 0) && RpRefPicAvailFlagLX &&
       RefRpRefAvailFlagLX[RefViewIdx[xP][yP]] (H-173)

     – If resPredFlag is equal to 1, the bilinear sample interpolation and residual prediction process as specified in subclause H.8.5.3.3.7 is invoked with the luma locations (xCb, yCb), (xBl, yBl), the size of the current luma coding block nCbS, the width and the height of the current luma prediction block nPbW, nPbH, the prediction list indication X, the prediction list utilization flag predFlagLX, the reference index refIdxLX, and the motion vectors mvLX, mvCLX, as the inputs and the outputs are the arrays $predSamplesLX_L$, $predSamplesLX_{Cb}$, and $predSamplesLX_{Cr}$.

     – Otherwise, (resPredFlag is equal to 0), the following applies:

       – The reference picture consisting of an ordered two-dimensional array $refPicLX_L$ of luma samples and two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in subclause 8.5.3.3.2 with refIdxLX as input.

       – If DepthFlag is equal to 0, the arrays $predSamplesLX_L$, $predSamplesLX_{Cb}$, and $predSamplesLX_{Cr}$ are derived by invoking the fractional sample interpolation process specified in subclause 8.5.3.3.3 with the luma locations (xCb, yCb) and (xBl, yBl), the luma prediction block width nPbW, the luma prediction block height nPbH, the motion vectors mvLX and mvCLX, and the reference arrays $refPicLX_L$, $refPicLX_{Cb}$, and $refPicLX_{Cr}$ as inputs.

       – Otherwise, (DepthFlag is equal to 1), arrays $predSamplesLX_L$, $predSamplesLX_{Cb}$, and $predSamplesLX_{Cr}$ are derived by invoking the full sample interpolation process specified in subclause H.8.5.3.3.5 with the luma locations (xCb, yCb), (xBl, yBl), the width and the height of the current luma prediction block nPbW, nPbH, the motion vectors mvLX, mvCLX, and the reference arrays with $refPicLX_L$, $refPicLX_{Cb}$ and $refPicLX_{Cr}$ given as input.

3. Depending on ic_flag, the array $predSamples_L$ is derived as specified in the following:

   – If ic_flag is equal to 0, the following applies.

     – The array $predSample_L$ of the prediction samples of luma component is derived by invoking the weighted sample prediction process specified in subclause 8.5.3.3.4 with the luma prediction block width nPbW, the luma prediction block height nPbH, and the sample arrays predSamplesL0$_L$ and predSamplesL1$_L$, and the variables predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 0 as inputs. [Ed. (GT): There seems to be an issue with the base spec. In this subclause predSample$_{L\,s}$ is of size (nCbS$_L$)x(nCbS$_L$), whereas the output of 8.5.3.3.4 is of size (nPbW)x(nPbH).]

– Otherwise (ic_flag is equal to 1), the following applies.

  – The array predSample$_L$ of the prediction samples of luma component is derived by invoking the illumination compensated sample prediction process specified in subclause H.8.5.3.3.6, with the luma location (xCb, yCb), the size of the current luma coding block nCbS, the luma location (xBl, yBl), the width and the height of the current luma prediction block nPbW, nPbH, and the sample arrays predSamplesL0$_L$ and predSamplesL1$_L$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1 and cIdx equal to 0 given as input.

4. Depending on ic_flag and nPbW, the arrays predSample$_{Cb}$, and predSample$_{Cr}$ are derived as specified in the following:

– If ic_flag is equal to 0 or nPbW is not greater than 8, the following applies:

  – The array predSample$_{Cb}$ of the prediction samples of component Cb is derived by invoking the weighted sample prediction process specified in subclause 8.5.3.3.4 with the chroma prediction block width nPbW$_{Cb}$ set equal to nPbW / 2, the chroma prediction block height nPbH$_{Cb}$ set equal to nPbH / 2, the sample arrays predSamplesL0$_{Cb}$ and predSamplesL1$_{Cb}$, and the variables predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 1 as inputs.

  – The array predSample$_{Cr}$ of the prediction samples of component Cr is derived by invoking the weighted sample prediction process specified in subclause 8.5.3.3.4 with the chroma prediction block width nPbW$_{Cr}$ set equal to nPbW / 2, the chroma prediction block height nPbH$_{Cr}$ set equal to nPbH / 2, the sample arrays predSamplesL0$_{Cr}$ and predSamplesL1$_{Cr}$, and the variables predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 2 as inputs.

– Otherwise (ic_flag is equal to 1 and nPbW is greater than 8), the following applies:

  – The array predSample$_{Cb}$ of the prediction samples of component Cb is derived by invoking the illumination compensated sample prediction process specified in subclause H.8.5.3.3.6, with the luma location (xCb, yCb), the size of the current luma coding block nCbS, with the chroma location (xBl/2, yBl/2), the width and the height of the current chroma prediction block nPbW$_{Cb}$ set equal to nPbW / 2, nPbH$_{Cb}$ set equal to nPbH / 2, and the sample arrays predSamplesL0$_{Cb}$ and predSamplesL1$_{Cb}$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvCL0, mvCL1, and cIdx equal to 1 given as input.

  – The array predSample$_{Cr}$ of the prediction samples of component Cr is derived by invoking the illumination compensated sample prediction process specified in subclause H.8.5.3.3.6, with the luma location (xCb, yCb), the size of the current luma coding block nCbS, with the chroma location (xBl / 2, yBl / 2), the width and the height of the current chroma prediction block nPbW$_{Cr}$ set equal to nPbW / 2, nPbH$_{Cr}$ set equal to nPbH / 2, and the sample arrays predSamplesL0$_{Cr}$ and predSamplesL1$_{Cr}$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvCL0, mvCL1, and cIdx equal to 2 given as input.

– Otherwise, (VspModeFlag[xCb + xBl][yCb + yBl] is equal to 1), the following applies:

  – For X in the range of 0 to 1, inclusive, the following applies.

    – When predFlagLX is equal to 1, the arrays predSamples$_L$,predSample$_{Cb}$, and predSample$_{Cr}$ are derived by invoking the view synthesis prediction process as specified in subclause H.8.5.3.3.7.3, with the luma locations (xCb, yCb), (xBl, yBl), the width and the height of the current luma prediction block nPbW, nPbH, the prediction list indicator X and the reference index refIdxLX as the inputs and the outputs are the sample arrays predSamplesLX$_L$, predSamplesLX$_{Cb}$, and predSamplesLX$_{Cr}$.

  – The array predSample$_L$ of the prediction samples of luma component is derived by invoking the weighted sample prediction process specified in subclause 8.5.2.2.3 with the luma location (xBl, yBl), the width and the height of the current luma prediction block nPbW, nPbH, and the sample arrays predSamplesL0$_L$ and predSamplesL1$_L$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1 and cIdx equal to 0 given as input.

  – The array predSample$_{Cb}$ of the prediction samples of component Cb is derived by invoking the weighted sample prediction process specified in subclause 8.5.2.2.3 with the chroma location (xBl/2, yBl/2), the width and the height of the current chroma prediction block nPbW$_{Cb}$ set equal to nPbW / 2, nPbH$_{Cb}$ set equal to nPbH / 2, and the sample arrays predSamplesL0$_{Cb}$ and predSamplesL1$_{Cb}$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 1 given as input.

- The array $predSample_{Cr}$ of the prediction samples of component Cr is derived by invoking the weighted sample prediction process specified in subclause 8.5.2.2.3 with the chroma location (xBl / 2, yBl / 2), the width and the height of the current chroma prediction block $nPbW_{Cr}$ set equal to nPbW / 2, $nPbH_{Cr}$ set equal to nPbH / 2, and the sample arrays $predSamplesL0_{Cr}$ and $predSamplesL1_{Cr}$ as well as predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 2 given as input.

H.8.5.3.3.2 Reference picture selection process

The specifications in subclause 8.5.3.3.2 apply.

H.8.5.3.3.3 Fractional sample interpolation process

The specifications in subclause 8.5.3.3.3 apply.

H.8.5.3.3.4 Weighted sample prediction process

The specifications in subclause 8.5.3.3.4 apply.

H.8.5.3.3.5 Full sample interpolation process

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,
- a luma location (xBl, yBl) specifying the top-left sample of the current luma prediction block relative to the top left sample of the current luma coding block,
- the width and height of the prediction block, nPbW and nPbH, in luma-sample units,
- a luma motion vector mvLX given in quarter-luma-sample units,
- a chroma motion vector mvCLX given in eighth-chroma-sample units,
- the reference picture sample arrays $refPicLX_L$, $refPicLX_{Cb}$, and $refPicLX_{Cr}$.

Outputs of this process are:

- a (nPbW)x(nPbH) array $predSampleLX_L$ of prediction luma sample values,
- two (nPbW/2)x(nPbH/2) arrays $predSampleLX_{Cb}$, and $predSampleLX_{Cr}$ of prediction chroma sample values.

The location (xP, yP) given in full-sample units of the upper-left luma samples of the current prediction block relative to the upper-left luma sample location of the given reference sample arrays is derived by

$$xP = xCb + xBl \tag{H-174}$$

$$yP = yCb + yBl \tag{H-175}$$

Let ($xInt_L$, $yInt_L$) be a luma location given in full-sample units specifying sample locations inside the reference sample arrays $refPicLX_L$.

For each luma sample location ($x_L = 0..(nPbW - 1)$, $y_L = 0..(nPbH-1)$) inside the prediction luma sample array $predSampleLX_L$, the corresponding prediction luma sample value $predSampleLX_L[x_L][y_L]$ is derived as follows:

- The variables $xInt_L$, $yInt_L$, are derived as specified in the following:

$$xInt_L = xP + mvLX[0] + x_L \tag{H-176}$$

$$yInt_L = yP + mvLX[1] + y_L \tag{H-177}$$

- The prediction luma sample value $predSampleLX_L[x_L][y_L]$ is derived as specified in the following:

$$predSampleLX_L[x_L][y_L] = refPicLX_L[xInt_L][yInt_L] \tag{H-178}$$

For each chroma sample location ($x_C = 0..(nPbW / 2 - 1)$, $y_C = 0..(nPbH / 2 -1)$) inside the prediction chroma sample arrays $predSampleLX_{Cb}$ and $predSampleLX_{Cr}$, the corresponding prediction chroma sample values $predSampleLX_{Cb}[x_C][y_C]$ and $predSampleLX_{Cr}[x_C][y_C]$ are set to be equal to $(1 << (BitDepth_C - 1))$.

[Ed. (GT): In current software and draft chroma planes are also present for depth. A general discussion is needed to specify how chroma planes are handled. (#12)]

H.8.5.3.3.6 Illumination compensated sample prediction process

Inputs to this process are:

- a location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top left sample of the current picture,
- the size of current luma coding block nCbS,
- a location (xBl, yBl) specifying the top-left sample of the current prediction block relative to the top left sample of the current coding block,
- the width and height of this prediction block, nPbW and nPbH,
- two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1,
- prediction list utilization flags, predFlagL0 and predFlagL1,
- reference indices, refIdxL0 and refIdxL1,
- motion vector mvL0 and mvL1
- colour component index, cIdx,

Outputs of this process are:

- the (nPbW)x(nPbH) array predSamples of prediction sample values.

Variables shift1, shift2, offset1 and offset2 are derived as follows.

- The variable shift1 is set equal to 14 − bitDepth and the variable shift2 is set equal to 15 − bitDepth,
- The variable offset1 is derived as follows.
  - If shift1 is greater than 0, offset1 set equal to 1 << (shift1 − 1).
  - Otherwise (shift1 is equal to 0), offset1 is set equal to 0.
- The variable offset2 is set equal to 1 << (shift2 − 1).

The variable bitDepth is derived as follows.

- If cIdx is equal to 0, bitDepth is set equal to $\text{BitDepth}_{Y.}$.
- Otherwise (cIdx is equal to 1 or 2), bitDepth is set equal to BitDepth_{C}.

The derivation process for illumination compensation mode availability and parameters as specified in subclause H.8.5.3.3.6.1 is invoked with the luma location (xCb, yCb), the size of the current luma coding block nCbS, prediction list utilization flags, predFlagL0 and predFlagL1, reference indices refIdxL0 and refIdxL1, motion vectors mvL0 and mvL1, the bit depth of samples, bitDepth, a variable cIdx specifying colour component index as the inputs and the outputs are the flags puIcFlagL0 and puIcFlagL1 and the variables icWeightL0 and icWeightL1 specifying weights for illumination compensation, the variables icOffsetL0 and icOffsetL1 specifying offsets for illumination compensation.

Depending on the value of predFlagL0 and predFlagL1, the prediction samples predSamples[x][y] with x = 0..(nPbW) − 1 and y = 0..(nPbH) − 1 are derived as follows:

- For X in the range of 0 to 1, inclusive, the following applies:
  - When predFlagLX is equal to 1 the following applies:

$$\text{clipPredVal} = \text{Clip3}(\,0,\,(\,1 \ll \text{bitDepth}\,) - 1,\,(\,\text{predSamplesLX}[\,x\,][\,y\,] + \text{offset1}\,) \gg \text{shift1}\,) \tag{H-179}$$

$$\text{predValX} = !\text{puIcFlagLX}\ ?\ \text{clipPredVal} :\ (\,\text{Clip3}(\,0,\,(\,1 \ll \text{bitDepth}\,) - 1,\,(\,\text{clipPredVal} * \text{icWeightLX}\,) \gg 5\,) + \text{icOffsetLX}\,) \tag{H-180}$$

- If predFlagL0 is equal to 1 and predFlagL1 is equal to 1, the following applies:

$$\text{predSamples}[\,x\,][\,y\,] = \text{Clip3}(\,0,\,(\,1 \ll \text{bitDepth}\,) - 1,\,(\,\text{predVal0} + \text{predVal1} + \text{offset2}\,) \gg \text{shift2}\,) \tag{H-181}$$

- Otherwise (predFlagL0 is equal to 0 or predFlagL1 is equal to 0), the following applies:

$$\text{predSamples}[\,x\,][\,y\,] = \text{predFlagL0}\ ?\ \text{predVal0} : \text{predVal1} \tag{H-182}$$

H.8.5.3.3.6.1 Derivation process for illumination compensation mode availability and parameters

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top left sample of the current picture,

- the size of the current luma coding block, nCbS,
- prediction list utilization flags, predFlagL0 and predFlagL1,
- reference indices refIdxL0 and refIdxL1,
- motion vectors mvL0 and mvL1
- a bit depth of samples, bitDepth.
- a variable cIdx specifying colour component index.

Outputs of this process are:

- flags puIcFlagL0 and puIcFlagL1 specifying whether illumination compensation is enabled.
- variables icWeightL0 and icWeightL1 specifying weights for illumination compensation
- variables icOffsetL0 and icOffsetL1 specifying offsets for illumination compensation

The variables puIcFlagL0 and PuIcFlagL1 are set equal to 0, the variables icWeightL0 and icWeightL1 are set equal to 1, and the variables icOffsetL0 and icOffsetL1 are set equal to 0.

The variables nCS specifying the current luma or chroma coding block size, and the location (xC, yC) specifying the top left sample of the current luma or chroma coding block is derived as follows.

$$\mathrm{nCS} = (\mathrm{cIdx} == 0) \; ? \; \mathrm{nCbS} : \mathrm{nCbS} / 2 \qquad \text{(H-183)}$$

$$(\mathrm{xC}, \mathrm{yC}) = (\mathrm{cIdx} == 0) \; ? \; (\mathrm{xCb}, \mathrm{yCb}) : (\mathrm{xCb} / 2 , \mathrm{yCb} / 2) \qquad \text{(H-184)}$$

The variable availFlagCurAboveRow specifying the availability of above neighbouring row samples is derived by invoking the availability derivation process for a block in z-scan order as specified in subclause 6.4.1 with the location (xCurr, yCurr) set equal to (xCb, yCb) and the neighbouring location (xN, yN) set equal to (xCb, yCb − 1) as the input and the output is assigned to availFlagCurAboveRow.

The variable availFlagCurLeftCol specifying the availability of left neighbouring column samples is derived by invoking the availability derivation process for a block in z-scan order as specified in subclause 6.4.1 with the location (xCurr, yCurr) set equal to (xCb, yCb) and the neighbouring location (xN, yN) set equal to (xCb − 1, yCb) as the input and the output is assigned to availFlagCurLeftCol.

[Ed. (GT) The availability derivation is specified as performed in the software. However, the check of the availability of left and above PU might not be sufficient to guarantee the availability of the whole left column or above row. A check of availability similar to that used for intra prediction might be a better solution.]

When availFlagCurAboveRow is equal to 0 and availFlagCurLeftCol is equal to 0 the whole derivation process of this subclause terminates.

For X being replaced by 0 and 1, when predFlagLX is equal to 1, the variable puIcFlagLX is derived by the following ordered steps.

1. The variable refPicLX specifying the reference picture from reference picture list X is set equal to RefPicListX[refIdxLX] .
2. If ViewIdx(refPicLX) is not equal to ViewIdx, the variable puIvPredFlagLX specifying whether inter-view prediction from list X is utilized is set equal to 1, otherwise (predFlagLX is equal to 0 or ViewIdx(RefPicListX[refIdxLX]) is equal to ViewIdx), puIvPredFlagLX is set equal to 0.
3. If puIvPredFlagLX is equal to 0, the variable puIcFlagLX is set equal to 0, otherwise (puIvPredFlagLX is equal to 1) the following applies:
    - The luma location (xRLX, yRLX) specifying the top-left sample of the reference block in refPicLX is derived as

      $$\mathrm{xRLX} = \mathrm{xC} + ((\mathrm{mvLX}[0] + (\mathrm{cIdx} \; ? \; 4 : 2)) \gg (2 + (\mathrm{cIdx} \; ? \; 1 : 0))) \qquad \text{(H-185)}$$

      $$\mathrm{yRLX} = \mathrm{yC} + ((\mathrm{mvLX}[1] + (\mathrm{cIdx} \; ? \; 4 : 2)) \gg (2 + (\mathrm{cIdx} \; ? \; 1 : 0))) \qquad \text{(H-186)}$$

    - The variable availFlagAboveRowLX specifying whether the above neighbouring row samples of the current block and the reference block are available is derived as specified in the following:

      $$\mathrm{availFlagAboveRowLX} = (\mathrm{yRLX} > 0) \; \&\& \; \mathrm{availFlagCurAboveRow} \qquad \text{(H-187)}$$

    - The variable availFlagLeftColLX specifying whether the left neighbouring column samples of the current block and the reference block are available is derived as specified in the following:

$$\text{availFlagLeftColLX} = (\ \text{xRLX} > 0\ )\ \ \&\&\ \ \text{availFlagCurLeftCol} \tag{H-188}$$

– The variable puIcFlagLX is derived as follows:

$$\text{puIcFlagLX} = \text{availFlagAboveRowLX}\ \ ||\ \ \text{availFlagLeftColLX} \tag{H-189}$$

Depending on the colour component cIdx, the variable curRecSamples specifying the reconstructed picture samples of the current picture is derived as

$$\text{curRecSamples} = (\ !\text{cidx}\ )\ ?\ \text{RecSamplesL} : (\ (\ \text{icdx} == 1\ )\ ?\ \text{RecSamplesCb} : \text{RecSamplesCr}\ ) \tag{H-190}$$

[Ed. (GT). The reconstructed samples before deblocking filter RecSamplesL, RecSamplesCb and RecSamplesCr as used above although not explicitly defined. However, they should be defined in the base spec.]

For X being replaced by 0 and 1, when puIcFlagLX is equal to 1, the variables icWeightLX, and icOffsetLX are derived by the following ordered steps:

1. Depending on the colour component cIdx, the variable refRecSamples specifying the reconstructed picture samples of the reference picture is derived as specified in the following.
   - If cIdx is equal to 0, refRecSamples is set equal to reconstructed picture sample array S_L of picture refPicLX.
   - Otherwise, if cIdx is equal to 1, refRecSamples is set equal to the reconstructed chroma sample array S_{Cb} of picture refPicLX.
   - Otherwise (cIdx is equal to 2), refRecSamples is set equal to the reconstructed chroma sample array S_{Cr} of picture refPicLX.
2. The lists curNeighSampleListLX and refNeighSampleListLX specifying the neighbouring samples in the current picture and the reference picture are derived as specified in the following.
   - The variable numNeighSamplesLX specifying the number of elements in curNeighSampleListLX and in refNeighSampleLX is set equal to 0.
   - The variable leftNeighOffLX specifying the offset of the left neighbouring samples in curNeighSampleListLX and refNeighSampleLX is derived as

$$\text{leftNeighOffLX} = \text{availFlagAboveRowLX}\ ?\ 0 : \text{nCS} \tag{H-191}$$

   - For i ranging from 0 to nCS − 1, inclusive the following applies.
     - When availFlagAboveRowLX is equal to 1 the following applies.

$$\text{curNeighSampleListLX}[\ i\ ] = \text{curRecSamples}[\ \text{xC} + i][\ \text{yC} - 1\ ] \tag{H-192}$$

$$\text{refNeighSampleListLX}[\ i\ ] = \text{refRecSamples}[\ \text{xRLX} + i\ ][\ \text{yRLX} - 1\ ] \tag{H-193}$$

$$\text{numNeighSamplesLX}\ \ \mathrel{+}=\ \ 1 \tag{H-194}$$

     - When availFlagLeftColLX is equal to 1 the following applies

$$\text{curNeighSampleListLX}[\ i + \text{leftNeighOffLX}\ ] = \text{curRecSamples}[\ \text{xC} - 1][\ \text{yC} + i\ ] \tag{H-195}$$

$$\text{refNeighSampleListLX}[\ i + \text{leftNeighOffLX}\ ] = \text{refRecSamples}[\ \text{xRLX} - 1][\ \text{yRLX} + i\ ] \tag{H-196}$$

$$\text{numNeighSamplesLX}\ \ \mathrel{+}=\ \ 1 \tag{H-197}$$

3. The derivation process for illumination compensation parameters as specified in subclause H.8.5.3.3.6.2 is invoked, with the list of neighbouring samples in the current picture curNeighSampleList, the list of neighbouring samples in the reference picture refNeighSample list, the number of neighbouring samples numNeighSamlesLX and the size of the current luma coding block nCSl as inputs and the illumination parameters icWeightLX, and icOffsetLX as outputs.

H.8.5.3.3.6.2 Derivation process for illumination compensation parameters

Inputs to this process are:

- a list curSampleList specifying the current samples, ,
- a list refSampleList specifying the reference samples,
- a variable numSamples specifying the number of elements in curSampleList and refSampleList.
- a bit depth of samples, bitDepth.

- the size of the current luma coding block nCSl

Outputs of this process are:

- a variable icWeight specifying a weight for illumination compensation,
- a variable icOffset specifying a offset for illumination compensation,

The variable precShift is set equal to Max(0, bitDepth − 12).

The variables sumRef, sumCur, sumRefSquare and sumProdRefCur are set equal to 0 and the following applies for i ranging from 0 to numSamples / 2− 1, inclusive:

$$\mathrm{sumRef} \mathrel{+}= \mathrm{refSampleList}[\,2 * i\,] \tag{H-198}$$

$$\mathrm{sumCur} \mathrel{+}= \mathrm{curSampleList}[\,2 * i\,] \tag{H-199}$$

$$\mathrm{sumRefSquare} \mathrel{+}= (\,\mathrm{refSampleList}[\,2 * i\,] * \mathrm{refSampleList}[\,2 * i\,]\,) \gg \mathrm{precShift} \tag{H-200}$$

$$\mathrm{sumProdRefCur} \mathrel{+}= (\,\mathrm{refSampleList}[\,2 * i\,] * \mathrm{curSampleList}[\,2 * i\,]\,) \gg \mathrm{precShift} \tag{H-201}$$

The variable avgShift and avgOffset are derived as follows:

$$\mathrm{avgShift} = \mathrm{Log2}(\,\mathrm{numSamples} / 2) \tag{H-202}$$

$$\mathrm{avgOffset} = 1 \ll (\,\mathrm{avgShift} - 1\,) \tag{H-203}$$

The variables numerDiv and denomDiv are derived as follows:

$$\begin{aligned}\mathrm{denomDiv} = {} & (\,(\,\mathrm{sumRefSquare} + (\,\mathrm{sumRefSquare} \gg 7\,)\,) \ll \mathrm{avgShift}\,) \\ & - (\,\mathrm{sumRef} * \mathrm{sumRef}\,) \gg \mathrm{precShift}\end{aligned} \tag{H-204}$$

$$\begin{aligned}\mathrm{numerDiv} = {} & \mathrm{Clip3}(\,0, 2 * \mathrm{denomDiv}, (\,(\,\mathrm{sumProdRefCur} + (\,\mathrm{sumRefSquare} \gg 7\,)\,) \ll \mathrm{avgShift}\,) \\ & - (\,\mathrm{sumRef} * \mathrm{sumCur}\,) \gg \mathrm{precShift}\,)\end{aligned} \tag{H-205}$$

The variables shiftNumer and shiftDenom are derived as follows:

$$\mathrm{shiftDenom} = \mathrm{Max}(\,0, \mathrm{Floor}(\,\mathrm{Log2}(\,\mathrm{Abs}(\,\mathrm{denomDiv}\,)\,)\,) - 5\,) \tag{H-206}$$

$$\mathrm{shiftNumer} = \mathrm{Max}(\,0, \mathrm{shiftDenom} - 12\,) \tag{H-207}$$

The variables sNumerDiv and sDenomDiv are derived as follows:

$$\mathrm{sDenomDiv} = \mathrm{denomDiv} \gg \mathrm{shiftDenom} \tag{H-208}$$

$$\mathrm{sNumerDiv} = \mathrm{numerDiv} \gg \mathrm{shiftNumer} \tag{H-209}$$

The value of variable divCoeff is derived from Table H-8 depending on sDenomDiv and the variables icWeight, and icOffset are derived as follows:

$$\mathrm{icWeight} = (\,\mathrm{sNumerDiv} * \mathrm{divCoeff}\,) \gg (\,\mathrm{shiftDenom} - \mathrm{shiftNumer} + 10\,) \tag{H-210}$$

$$\mathrm{icOffset} = (\,\mathrm{sumCur} - (\,(\,\mathrm{icWeight} * \mathrm{sumRef}\,) \gg 5\,) + \mathrm{avgOffset}\,) \gg \mathrm{avgShift} \tag{H-211}$$

Table H-8 – Specification of divCoeff depending on sDenomDiv

sDenomDiv	0	1	2	3	4	5	6	7	8	9	10	11	12
divCoeff	0	32768	16384	10923	8192	6554	5461	4681	4096	3641	3277	2979	2731
sDenomDiv	13	14	15	16	17	18	19	20	21	22	23	24	25
divCoeff	2521	2341	2185	2048	1928	1820	1725	1638	1560	1489	1425	1365	1311
sDenomDiv	26	27	28	29	30	31	32	33	34	35	36	37	38
divCoeff	1260	1214	1170	1130	1092	1057	1024	993	964	936	910	886	862
sDenomDiv	39	40	41	42	43	44	45	46	47	48	49	50	51
divCoeff	840	819	799	780	762	745	728	712	697	683	669	655	643
sDenomDiv	52	53	54	55	56	57	58	59	60	61	62	63	
divCoeff	630	618	607	596	585	575	565	555	546	537	529	520	

H.8.5.3.3.7 Bilinear sample interpolation and residual prediction process

The process is only invoked if res_pred_flag is equal to 1.

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,
- a luma location (xBl, yBl) specifying the top-left sample of the current luma prediction block relative to the top-left sample of the current luma coding block,
- a variable nCbS specifying the size of the current luma coding block,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit, prediction list utilization flags, predFlagL0 and predFlagL1,
- the prediction list indication X,
- the prediction list utilization flag predFlagLX,
- the reference index refIdxLX,
- the motion vectors mvLX, mvCLX

Outputs of this process are:

- the (nPbW)x(nPbH) array $\text{predSamplesLX}_{\text{L}}$,
- the (nPbW / 2)x(nPbH / 2) arrays $\text{predSamplesLX}_{\text{Cb}}$ and $\text{predSamplesLX}_{\text{Cr}}$.

The location (xP, yP) is derived by:

$$\text{xP} = \text{xCb} + \text{xBl} \qquad \text{(H-212)}$$

$$\text{yP} = \text{yCb} + \text{yBl} \qquad \text{(H-213)}$$

The variable ivRefFlag is set equal to (DiffPicOrderCnt(currPic, RefPicListX[refIdxLX]) = = 0), and the variable availFlag is set equal to 0.

Depending on ivRefFlag and RpRefIdxLX, the following applies:

- If ivRefFlag is equal to 0 and RpRefIdxLX is not equal to −1, the variable availFlag is set equal to 1, the variable refIdxLX is set equal to RpRefIdxLX and the residual prediction motion vector scaling process as specified in subclause H.8.5.3.3.7.3 is invoked with the prediction list utilization variable equal to X, the motion vector mvLX, and the RefPicListX[refIdxLX] and as inputs and modified mvLX as output.
- Otherwise, when ivRefFlag is equal to 1, the following applies:
  - The derivation process for a motion vector from a reference block for residual prediction as specified in subclause H.8.5.3.3.7.4 is invoked with (xP, yP), nPbW and nPbH, RefPicListX[refIdxLX], and mvLX, as inputs, and availFlag, motion vector mvT and prediction list utilization variable Y as outputs.
  - When availFlag is equal to 0 and RpRefIdxLX is not equal to −1, availFlag is set equal to 1, mvT is set equal to

(0, 0), Y is set equal to X.

The motion vector mvCLX is set equal to mvLX.

The arrays predSamplesLX$_{L}$, predSamplesLX$_{Cb}$, and predSamplesLX$_{Cr}$ are derived as specified in the following:

- The reference picture consisting of an ordered two-dimensional array refPicLX$_{L}$ of luma samples and two ordered two-dimensional arrays refPicLX$_{Cb}$ and refPicLX$_{Cr}$ of chroma samples is derived by invoking the process specified in subclause 8.5.2.2.1 with currRefIdx as input.
- The arrays predSamplesLX$_{L}$, predSamplesLX$_{Cb}$, and predSamplesLX$_{Cr}$ are derived by invoking the bilinear sample interpolation process specified in subclause H.8.5.3.3.7.1 with the luma locations (xCb, yCb), (xBl, yBl), , the luma prediction block width nPbW, the luma prediction block height nPbH,, the motion vectors mvLX, mvCLX, and the reference arrays with refPicLX$_{L}$, refPicLX$_{Cb}$ and refPicLX$_{Cr.}$ as the inputs.

When availFlag is equal to 1 and iv_res_pred_weight_idx is not equal to 0, the following applies:

- Depending on ivRefFlag, the variables rpPic, rpRefPic, mvRp and curRefIdx are derived as specified in the following:
  - If ivRefFlag is equal to 0, the following applies:
    - Let rpPic be the picture with PicOrderCnt(rpPic) equal to PicOrderCntVal and ViewIdx equal to RefViewIdx[xP][yP].
    - Let rpRefPic be the picture with PicOrderCnt(rpRefPic) equal to RefPicListX[RpRefIdxLX]) and ViewIdx equal to RefViewIdx[xP][yP],
    - The variable mvRp is set equal to MvDisp[xP][yP].
    - The variable curRefIdx is set equal to RpRefIdxLX.
  - Otherwise (ivRefFlag is equal to 1), the following applies:
    - Let rpPic be the picture RefPicListY[RpRefIdxLY]. [Ed. (CY): here the interaction with F0105 needs to be further studied.]
    - Let rpRefPic be the picture with PicOrderCnt(rpRefPic) equal to PicOrderCnt(rpPic) and ViewIdx equal to RefViewIdx[xP][yP]
    - The variable mvRp is set equal to mvT.
    - The variable currRefIdx is set equal to RpRefIdxLY.
- The arrays rpSamplesLXL, rpSamplesLXCb, and rpSamplesLXCr are derived as specified in the following:
  - Let the reference picture sample arrays rpPicLXL, rpPicLXCb, and rpPicLXCr corresponding to decoded sample arrays SL, SCb, SCr derived in subclause 8.7 for the previously-decoded picture rpPic.
  - The arrays rpSamplesLXL, rpSamplesLXCb, and rpSamplesLXCr are derived by invoking the bilinear sample interpolation process specified in subclause H.8.5.3.3.7.1 with the luma locations (xCb, yCb) and (xBl, yBl), the luma prediction block width nPbW, the luma prediction block height nPbH, the motion vectors mvLX equal to mvRp and mvCLX equal to mvRp, and the reference arrays with rpPicLXL, rpPicLXCb and rpPicLXCr as the inputs.
- The arrays rpRefSamplesLXL, rpRefSamplesLXCb, and rpRefSamplesLXCr are derived as specified in the following:
  - Let the reference picture sample arrays rpRefPicLXL, rpRefPicLXCb, and rpRefPicLXCr corresponding to decoded sample arrays SL, SCb, SCr derived in subclause 8.7 for the previously-decoded picture rpRefPic.
  - The arrays rpRefSamplesLXL, rpRefSamplesLXCb, and rpRefSamplesLXCr are derived by invoking the bilinear sample interpolation process specified in subclause H.8.5.3.3.7.1 with the luma locations (xCb, yCb), (xBl, yBl), the luma prediction block width nPbW, the luma prediction block height nPbH,, the motion vector mvLX equal to (mvLX + mvRp) and the motion vector mvCLX equal to (mvCLX + mvRp), and the reference arrays with rpRefPicLXL, rpRefPicLXCb and rpRefPicLXCr as the inputs.
- The variable shiftVal is set equal to (iv_res_pred_weight_idx − 1).
- The modified prediction samples predSamplesLX$_{L}$[x][y] with x = 0..(nPbW) − 1 and y = 0..(nPbH) − 1 are derived as specified in the following:

  predSamplesLX$_{L}$[x][y] = predSamplesLX$_{L}$[x][y] +

$$((\text{rpSamplesLX}_L[x][y] - \text{rpRefSamplesLX}_L[x][y]) >> \text{shiftVal}) \qquad \text{(H-214)}$$

- The modified prediction samples $\text{predSamplesLX}_{Cb}[x][y]$ with x = 0..(nPbW /2) − 1 and y = 0..(nPbH /2)−1 are derived as specified in the following:

$$\text{predSamplesLX}_{Cb}[x][y] = \text{predSamplesLX}_{Cb}[x][y] + ((\text{rpSamplesLX}_{Cb}[x][y] - \text{rpRefSamplesLX}_{Cb}[x][y]) >> \text{shiftVal}) \qquad \text{(H-215)}$$

- The modified prediction samples $\text{predSamplesLX}_{Cr}[x][y]$ with x = 0..(nPbW /2) − 1 and y = 0..(nPbH /2) − 1 are derived as specified in the following:

$$\text{predSamplesLX}_{Cr}[x][y] = \text{predSamplesLX}_{Cr}[x][y] + ((\text{rpSamplesLX}_{Cr}[x][y] - \text{rpRefamplesLX}_{Cr}[x][y]) >> \text{shiftVal}) \qquad \text{(H-216)}$$

H.8.5.3.3.7.1 Bilinear sample interpolation process

The specifications in subclause 8.5.3.3.3.1 apply with the following modifications:

- All invocations of the process specified in subclause 8.5.3.3.3.2 are replaced with invocations of the process specified in subclause H.8.5.3.3.7.2 with chromaFlag equal to 0 as additional input.
- All invocations of the process specified in subclause 8.5.3.3.3.3 are replaced with invocations of the process specified in subclause H.8.5.3.3.7.2 with chromaFlag equal to 1 as additional input.

H.8.5.3.3.7.2 Bilinear luma and chroma sample interpolation process

Inputs to this process are:

- a location in full-sample units (xInt, yInt),
- a location offset in fractional-sample units (xFrac, yFrac),
- a sample reference array refPicLX,
- a flag chromaFlag.

Output of this process is a predicted sample value predPartLX[x][y].

In Figure H-1, the positions labelled with A, B, C, and D represent samples at full-sample locations inside the given two-dimensional array refPicLX of samples.

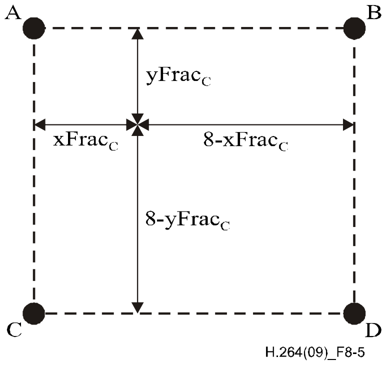

Figure H-1 Fractional sample position dependent variables in bi-linear interpolation and surrounding integer position samples A, B, C, and D

The variable picWidthInSamples is set equal to pic_width_in_luma_samples and the variable picHeightInSamples is set equal to pic_height_in_luma_samples.

- If chromaFlag is equal 0, xFrac is set equal to (xFrac << 1) and yFrac is set equal to (yFrac << 1).
- Otherwise (chromaFlag is equal to 1), picWidthInSamples is set equal to (picWidthInSamples / SubWidthC) and picHeightInSamples is set equal to (picHeightInSamples / SubHeightC).

The coordinates of positions A, B, C and D are derived as follows:.

$$xA = \text{Clip3}(0, \text{picWidthInSamples} - 1, \text{xInt}) \qquad \text{(H-217)}$$

$$xB = \text{Clip3}(0, \text{picWidthInSamples} - 1, \text{xInt} + 1) \qquad \text{(H-218)}$$

$$xC = \text{Clip3}(0, \text{picWidthInSamples} - 1, \text{xInt}) \qquad \text{(H-219)}$$

$$xD = \text{Clip3}(\ 0, \text{picWidthInSamples} - 1, \text{xInt} + 1\ ) \tag{H-220}$$

$$yA = \text{Clip3}(\ 0, \text{picHeightInSamples} - 1, \text{yInt}\ ) \tag{H-221}$$

$$yB = \text{Clip3}(\ 0, \text{picHeightInSamples} - 1, \text{yInt}\ ) \tag{H-222}$$

$$yC = \text{Clip3}(\ 0, \text{picHeightInSamples} - 1, \text{yInt} + 1\ ) \tag{H-223}$$

$$yD = \text{Clip3}(\ 0, \text{picHeightInSamples} - 1, \text{yInt} + 1\ ) \tag{H-224}$$

The value of predPartLX[x][y] is derived as specified in the following:

$$\begin{aligned}\text{predPartLX}[\ x\ ][\ y\ ] = (\,&\text{refPicLX}[\ xA\ ][\ yA\ ] * (\ 8 - \text{xFrac}\ ) * (\ 8 - \text{yFrac}\ ) + \\ &\text{refPicLX}[\ xB\ ][\ yB\ ] * (\ 8 - \text{yFrac}\ ) * \text{xFrac} + \\ &\text{refPicLX}[\ xC\ ][\ yC\ ] * (\ 8 - \text{xFrac}\ ) * \text{yFrac} + \\ &\text{refPicLX}[\ xD\ ][\ yD\ ] * \text{xFrac} * \text{yFrac}\ ) >> 6\end{aligned} \tag{H-225}$$

H.8.5.3.3.7.3 Residual prediction motion vector scaling process

Inputs to this process are:

- A prediction list utilization variable X,
- A motion vector mvLX,
- A reference picture (associated with the motion vector mvLX) refPicLX,

Output of this process is a scaled motion vector mvLX.

The motion vector mvLX is scaled as specified in the following:

$$tx = (\ 16384 + (\ \text{Abs}(\ td\ )\ >>\ 1\ )\ ) / td \tag{H-226}$$

$$\text{distScaleFactor} = \text{Clip3}(\ -4096, 4095, (\ tb * tx + 32\ ) >> 6\ ) \tag{H-227}$$

$$\begin{aligned}mv = \ &\text{Clip3}(\ -32768, 32767, \text{Sign}(\ \text{distScaleFactor} * \text{mvLX}\ ) * \\ &(\ (\ \text{Abs}(\ \text{distScaleFactor} * \text{mvLX}\ ) + 127\ ) >> 8\ )\ )\end{aligned} \tag{H-228}$$

where td and tb are derived as:

$$td = \text{Clip3}(\ -128, 127, \text{DiffPicOrderCnt}(\ \text{currPic}, \text{refPicLX}\ )\ ) \tag{H-229}$$

$$tb = \text{Clip3}(\ -128, 127, \text{DiffPicOrderCnt}(\ \text{currPic}, \text{RefPicListX}[\ \text{RpRefIdxLX}\ ])\ ) \tag{H-230}$$

[Ed. (CY): need to update the equation numbers for those from H-241 to this sub-clause.]

H.8.5.3.3.7.4 Derivation process for a motion vector from a reference block for residual prediction

Inputs to this process are:

- a luma location (xP, yP) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
- variables nPbW and nPbH specifying the width and the height, respectively, of the current prediction unit,
- a reference picture refPic,
- a motion vector mvDisp

Outputs of this process are:

- a flag availFlag
- a motion vector mvT
- prediction list utilization variable Y.

The variable availFlag is set to 0 and the reference luma location (xRef, yRef) in refPicLX is derived by

$$xRef = \text{Clip3}(\ 0, \text{PicWidthInSamples}_L - 1, xP + (\ \text{nPSW}\ >>\ 1\ ) + (\ (\ \text{mvDisp}[\ 0\ ] + 2\ )\ >>\ 2\ )\ ) \tag{H-231}$$
$$yRef = \text{Clip3}(\ 0, \text{PicHeightInSamples}_L - 1, yP + (\ \text{nPSH}\ >>\ 1\ ) + (\ (\ \text{mvDisp}[\ 1\ ] + 2\ )\ >>\ 2\ )\ ) \tag{H-232}$$

Let variable refCU and refPU be the coding unit and prediction unit that cover the luma location (xRef, yRef) in refPic, respectively.

When the variable PredMode for the coding unit refCU is equal to MODE_SKIP or MODE_INTER, the following ordered steps apply for X in the range of 0 to 1, inclusive:

- The variable refPredFlagLX is set equal to the prediction utilization flag predFlagLX of the prediction unit refPU.
- When availFlag is equal to 0 and refPredFlagLX is equal to 1, the following applies:
  - Let refPicListRefX be the reference picture list X of refPic.
  - Let mvLX and refIdxLX be the motion vector and reference index of the prediction unit refPU corresponding to refPicListRefX, respectively. [Ed. (GT): What happens when predFlagLX is equal to 0? Ed. (CY): motion information not available, if both predFlagL0 and predFlagL1 are 0, the zero motion vector with ref index equal to RpRefIdxLX is used.]
  - When refPicListRefX[refIdxLX] is a temporal reference picture of refPic and RpRefIdxLX is not equal to –1, availFlag is set to 1, Y is set equal to X and the residual prediction motion vector scaling process as specified in subclause H.8.5.3.3.7.3 is invoked with the prediction list utilization variable equal to X, the motion vector mvLX, and the reference picture refPicListRefX[refIdxLX] as the inputs, and the output being mvT.

H.8.5.3.3.8 View synthesis prediction process

Inputs to this process are:

- a location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top left sample of the current picture,
- a location (xBl, yBl) specifying the top-left sample of the current prediction block relative to the top left sample of the current coding block,
- the width and height of this prediction block, nPbW and nPbH,
- the prediction list indicator X
- the reference index refIdxLX

Outputs of this process are:

- an array $predSamples_L$ of luma prediction samples,
- an array $predSamples_{Cb}$ of chroma prediction samples for the component Cb
- an array $predSamples_{Cr}$ of chroma prediction samples for the component Cr

The location (xP, yP) given in full-sample units of the upper-left luma samples of the current prediction block relative to the upper-left luma sample location of the current picture is derived by:

$$xP = xCb + xBl \qquad \text{(H-233)}$$

$$yP = yCb + yBl \qquad \text{(H-234)}$$

The reference picture consisting of an ordered two-dimensional array $refPic_L$ of luma samples and two ordered two-dimensional arrays $refPic_{Cb}$ and $refPic_{Cr}$ of chroma samples is derived by invoking the process specified in subclause 8.5.2.2.1 with refIdxLX as input.

The variable refViewIdx is set equal to the ViewIdx(RefPicListX[refIdxLX]) and the variable depthViewIdx is set equal to RefViewIdx[xCb + xBl][yCb + yBl]. The variable mvDisp is set equal to MvDisp[xCb + xBl][yCb + yBl]

The derivation process for a disparity sample array as specified in section H.8.5.5.2 is invoked with the luma location (xP, yP), the disparity vector mvDisp, the variable refViewIdx, the variable depthViewIdx, the variable nPSW, the variable nPSH, and the variable splitFlag equal to 1 as the inputs, and the output is the array disparitySamples of size (nPSW)x(nPSH).

Let ($xInt_L$, $yInt_L$) be a luma location given in full-sample units and ($xFrac_L$, $yFrac_L$) be an offset given in quarter-sample units.

For each luma sample location ($x_L = 0..nPbW - 1$, $y_L = 0..nPbH - 1$) inside the prediction luma sample array $predSamples_L$, the corresponding prediction luma sample value $predSamples_L[x_L][y_L]$ is derived as follows:

- The variables $xInt_L$, $yInt_L$, $xFrac_L$, and $yFrac_L$ are derived by

$$xInt_L = xP + x_L + disparitySamples[x_L][y_L] \qquad \text{(H-235)}$$

$$yInt_L = yP + y_L \qquad \text{(H-236)}$$

$$xFrac_L = disparitySamples[x_L][y_L] \;\&\; 3 \qquad \text{(H-237)}$$

$\text{yFrac}_L = 0$ (H-238)

– The prediction luma sample value $\text{predSamples}_L[\ x_L][\ y_L\ ]$ is derived by invoking the process specified in subclause 8.5.3.3.3.2 with (xInt_L, yInt_L), (xFrac_L, yFrac_L) and refPic_L given as input.

[Ed. (GT): As for inter prediction the treatment of colour planes for depth needs to be discussed. In software colour planes are set to 128 in VSP process. (#12)]

Let (xInt_C, yInt_C) be a chroma location given in full-sample units and (xFrac_C, yFrac_C) be an offset given in one-eighth sample units.

For each chroma sample location ($x_C = 0..\text{nPbW} / 2-1$, $y_C = 0..\text{nPbH} / 2 - 1$) inside the prediction chroma sample arrays predSamples_{Cb} and predSamples_{Cr}, the corresponding prediction chroma sample values $\text{predSampleLX}_{Cb}[\ x_C\ ][\ y_C\ ]$ and $\text{predSamples}_{Cr}[\ x_C\ ][\ y_C\ ]$ are derived as follows:

– The variables xInt_C, yInt_C, xFrac_C, and yFrac_C are derived by

$\text{xInt}_C = (\ \text{xP} / 2\ ) + x_C + \text{disparitySamples}[\ x_C << 1\ ][\ y_C << 1\ ]$ (H-239)

$\text{yInt}_C = (\ \text{yP} / 2\ ) + y_C$ (H-240)

$\text{xFrac}_C = \text{disparitySamples}[\ x_C << 1][\ y_C << 1\ ]\ \&\ 7$ (H-241)

$\text{yFrac}_C = 0$ (H-242)

– The prediction sample value $\text{predSamples}_{Cb}[\ x_C\ ][\ y_C\ ]$ is derived by invoking the process specified in subclause 8.5.3.3.3.3 with (xInt_C, yInt_C), (xFrac_C, yFrac_C) and refPic_{Cb} given as input.

– The prediction sample value $\text{predSamples}_{Cr}[\ x_C\ ][\ y_C\ ]$ is derived by invoking the process specified in subclause 8.5.3.3.3.3 with (xInt_C, yInt_C), (xFrac_C, yFrac_C) and refPic_{Cr} given as input.

H.8.5.3.3.9 Decoding process for sub prediction block wise inter sample prediction

Inputs to this process are:

– a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,

– a luma location (xBl, yBl) specifying the top-left sample of the current luma prediction block relative to the top-left sample of the current luma coding block,

– a variable nCbS specifying the size of the current luma coding block,

– two variables nPbW and nPbH specifying the width and the height of the luma prediction block,

Outputs of this process are:

– an $(\text{nCbS}_L)\text{x}(\text{nCbS}_L)$ array predSamples_L of luma prediction samples, where nCbS_L is derived as specified below,

– an $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array predSamples_{Cb} of chroma prediction samples for the component Cb, where nCbS_C is derived as specified below,

– an $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array predSamples_{Cr} of chroma residual samples for the component Cr, where nCbS_C is derived as specified below.

The variables nSbW and nSbH are derived as:

nSbW = nPbW / SubPbSize[nuh_layer_id] <= 1 ? nPbW : SubPbSize[nuh_layer_id] (H-243)

nSbH = nPbH / SubPbSize[nuh_layer_id] <= 1 ? nPbH : SubPbSize[nuh_layer_id] (H-244)

For x in the range of 0 to (nPbW / nSbW − 1), inclusive, the following applies:

– For y in the range of 0 to (nPbH / nSbH − 1), inclusive, the following applies:

  – The luma location (xSb, ySb) specifying the top-left sample of the current luma sub prediction block relative to the top-left sample of the current luma coding block is derived as specified in the following:

    xSb = xBl + x * nSbW (H-245)

    ySb = yBl + y * nSbH (H-246)

  – For X in the range of 0 to 1, inclusive, the variables mvLX, mvCLX, refIdxLX, and predFlagLX are derived as specified in the following:

    mvLX = SubPbMvLX[xSb][ySb] (H-247)

$$\text{mvCLX} = \text{SubPbMvCLX}[\ \text{xSb}\ ][\ \text{ySb}\ ] \tag{H-248}$$

$$\text{refIdxLX} = \text{SubPbRefIdxLX}[\ \text{xSb}\ ][\ \text{ySb}\ ] \tag{H-249}$$

$$\text{predFlagLX} = \text{SubPbPredFlagLX}[\ \text{xSb}\ ][\ \text{ySb}\ ] \tag{H-250}$$

  - The decoding process for inter sample prediction as specified in subclause H.8.5.3.3.1 is invoked with the luma coding block location (xCb, yCb), the luma prediction block location (xBl, yBl) equal to (xSb, ySb), the luma coding block size block nCbS, the luma prediction block width nPbW equal to nSbW, the luma prediction block height nPbH equal to nSbH, the luma motion vectors mvL0 and mvL1, the chroma motion vectors mvCL0 and mvCL1, the reference indices refIdxL0 and refIdxL1, and the prediction list utilization flags predFlagL0 and predFlagL1 as inputs, and the inter prediction samples (predSamples) that are an (nCbSL)x(nCbSL) array predSamplesL of prediction luma samples and two (nCbSC)x(nCbSC) arrays predSamplesCr and predSamplesCr of prediction chroma samples, one for each of the chroma components Cb and Cr, as outputs.

H.8.5.4 Decoding process for the residual signal of coding units coded in inter prediction mode

H.8.5.4.1 General

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable log2CbSize specifying the size of the current luma coding block.

Outputs of this process are:

- an $(\text{nCbS}_L)\text{x}(\text{nCbS}_L)$ array resSamples_L of luma residual samples, where nCbS_L is derived as specified below,
- an $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array resSamples_{Cb} of chroma residual samples for the component Cb, where nCbS_C is derived as specified below,
- an $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array resSamples_{Cr} of chroma residual samples for the component Cr, where nCbS_C is derived as specified below.

The variable nCbS_L is set equal to $1 \ll \text{log2CbSize}$ and the variable nCbS_C is set equal to $\text{nCbS}_L \gg 1$.

Let resSamples_L be an $(\text{nCbS}_L)\text{x}(\text{nCbS}_L)$ array of luma residual samples and let resSamples_{Cb} and resSamples_{Cr} be two $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ arrays of chroma residual samples.

- If inter_sdc_flag is equal to 0, the following applies, depending on the value of rqt_root_cbf, the following applies:
  - If rqt_root_cbf is equal to 0 or skip_flag[xCb][yCb] is equal to 1, all samples of the $(\text{nCbS}_L)\text{x}(\text{nCbS}_L)$ array resSamples_L and all samples of the two $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ arrays resSamples_{Cb} and resSamples_{Cr} are set equal to 0.
  - Otherwise (rqt_root_cbf is equal to 1), the following ordered steps apply:
    1. The decoding process for luma residual blocks as specified in subclause H.8.5.4.2 below is invoked with the luma location (xCb, yCb), the luma location (xB0, yB0) set equal to (0, 0), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable nCbS set equal to nCbS_L, and the $(\text{nCbS}_L)\text{x}(\text{nCbS}_L)$ array resSamples_L as inputs, and the output is a modified version of the $(\text{nCbS}_L)\text{x}(\text{nCbS}_L)$ array resSamples_L.
    2. The decoding process for chroma residual blocks as specified in subclause H.8.5.4.3 below is invoked with the luma location (xCb, yCb), the luma location (xB0, yB0) set equal to (0, 0), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable cIdx set equal to 1, the variable nCbS set equal to nCbS_C, and the $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array resSamples_{Cb} as inputs, and the output is a modified version of the $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array resSamples_{Cb}.
    3. The decoding process for chroma residual blocks as specified in subclause H.8.5.4.3 below is invoked with the luma location (xCb, yCb), the luma location (xB0, yB0) set equal to (0, 0), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable cIdx set equal to 2, the variable nCbS set equal to nCbS_C, and the $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array resSamples_{Cr} as inputs, and the output is a modified version of the $(\text{nCbS}_C)\text{x}(\text{nCbS}_C)$ array resSamples_{Cr}.
- Otherwise (inter_sdc_flag is equal to 1), the decoding process for simplified depth coded residual blocks as specified in subclause H.8.5.4.4 is invoked with the luma location (xCb, yCb), the luma location (xB0, yB0) set equal to (0, 0), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable nCbS

set equal to nCbS$_L$, and the (nCbS$_L$)x(nCbS$_L$) array resSamples$_L$ as inputs, and the output is a modified version of the (nCbS$_L$)x(nCbS$_L$) array resSamples$_L$.

For x in the range of 0 to nCbS$_L$ − 1 and y in the range of 0 to nCbS$_L$ − 1, the following applies:

- ResSamples$_L$[xCb + x][yCb + y] is set equal to resSamples$_L$[x][y].

For x in the range of 0 to nCbS$_C$ − 1 and y in the range of 0 to nCbS$_C$ − 1, the following applies:

- ResSamples$_{Cb}$[xCb /2 + x][yCb /2 + x] is set equal to resSamples$_{Cb}$[x][y].
- ResSamples$_{Cr}$[xCb /2 + x][yCb /2 + x] is set equal to resSamples$_{Cr}$[x][y].

H.8.5.4.2 Decoding process for luma residual blocks

The specification in subclause 8.5.4.2 applies.

H.8.5.4.3 Decoding process for chroma residual blocks

The specification in subclause 8.5.4.3 applies.

H.8.5.4.4 Decoding process for simplified depth coded residual blocks

Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable nCbS specifying the size of the current luma coding block,
- a (nCbS)x(nCbS) array resSamples of luma residual samples.

Output of this process is:

- a modified version of the (nCbS)x(nCbS) array of luma residual samples.

The values of the variables xOff, yOff, and interSdcResiIdx[i] for i in the range of 0 to 3, inclusive, depending on the value of PartMode are specified in Table H-9.

Table H-9 – Specification of the variables xOff, yOff, and interSdcResiIdx[i]

PartMode	**xOff**	**yOff**	**interSdcResiIdx**[i]			
			i = 0	i = 1	i = 2	i = 3
PART_2Nx2N	nCbS	nCbS	0	0	0	0
PART_2NxN	nCbS	(nCbS >> 1)	0	0	1	1
PART_2NxnU	nCbS	(nCbS >> 2)	0	0	1	1
PART_2NxnD	nCbS	(nCbS >> 1) + (nCbS >> 2)	0	0	1	1
PART_Nx2N	(nCbS >> 1)	nCbS	0	1	0	1
PART_nLx2N	(nCbS >> 2)	nCbS	0	1	0	1
PART_nRx2N	(nCbS >> 1) + (nCbS >> 2)	nCbS	0	1	0	1
PART_NxN	(nCbS >> 1)	(nCbS >> 1)	0	1	2	3

For x in the range of 0 to nCbS the following applies:

- For y in the range of 0 to nCbS the following applies:
  - The variable i is derived as specified in the following:
    - If x is less than xOff and y is less than yOff, i is set equal to 0.
    - Otherwise, if x greater than or equal to xOff and y is less than to yOff, i is set equal to 1.
    - Otherwise, if x less than xOff and y is greater than or equal to yOff, i is set equal to 2.
    - Otherwise, (x is greater than or equal to xOff and y is greater than or equal to yOff), i is set equal to 3.

  - The value of resSamples[x][y] is set equal to InterSdcResi[xCb][yCb][interSdcResiIdx[i]]

H.8.5.5 Derivation process for disparity vectors

Inputs to this process are:

- a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable nCbS specifying the size of the current luma coding block,

The flag availableDV is set equal to 0, and both components of the disparity vector mvDisp are set equal to 0.

The variable checkParallelMergeFlag is derived as follows:

- If one or more of the following conditions are true, checkParallelMergeFlag is set equal to 1.
  - PredMode[xCb][yCb] is equal to MODE_SKIP.
  - PredMode[xCb][yCb] is equal to MODE_INTER and merge_flag[xCb][yCb] is equal to 1.
- Otherwise, checkParallelMergeFlag is set equal to 0.

The derivation process for a disparity vector from temporal neighbour block as specified in subclause H.8.5.5.1 is invoked with the luma location (xCb, yCb), and the variable nCbS as inputs, and the outputs are the flag availableDV, the disparity vector mvDisp and the reference view order index refViewIdx.

When availableDV is equal to 0, for each N being A_1, B_1 and (xN, yN) being (xCb − 1, yCb + nCbS − 1), (xCb + nCbS − 1, yCb − 1), respectively, the following ordered steps apply.

1. When yCb − 1 is less than ((yCb >> Log2CtbSizeY) << Log2CtbSizeY), the following applies.

   $$xB_1 = ((xB_1 >> 3) << 3) + ((xB_1 >> 3) \& 1) * 7 \quad \text{(H-251)}$$

2. The derivation process for z-scan order block availability as specified in subclause 6.4.1 is invoked with (xCurr, yCurr) set equal to the (xCb, yCb) and the luma location (xN, yN) as the input and the output assigned to availableN.
3. When availableN is equal to 1 and PredMode[xN][yN] is equal to MODE_INTRA, availableN is set equal to 0. [Ed. (GT): 2+3 correspond to 6.4.2 for CU when (xN, yN) outside CU. Cross-check appreciated.]
4. When all of the following conditions are true, availableN is set equal to 0.
   - checkParallelMergeFlag is equal to 1
   - (xCb >> (log2_parallel_merge_level_minus2 + 2)) is equal to (xN >> (log2_parallel_merge_level_minus2 + 2))
   - (yCb >> (log2_parallel_merge_level_minus2 + 2)) is equal to (yN >> (log2_parallel_merge_level_minus2 + 2)).
5. The flag availableIvpMvSearchFlagN is set equal to availableN.
6. When one of the following conditions is true, N is equal to B_1 and ((yN >> Log2CtbSizeY) << Log2CtbSizeY) is less than ((yCb >> Log2CtbSizeY) << Log2CtbSizeY), availableIvpMvSearchFlagN is set equal to 0.
7. The flag availableFlagIvpMvN is set equal to 0.
8. For each X from 0 to 1, the following applies:
   - When availableDV is equal to 0, availableN is equal to 1, RefIdxLX[xN][yN] is greater than or equal to 0, and PredFlagLX[xN][yN] is equal to 1, the following applies:
     - If RefPicListX[RefIdxLX[xN][yN]] is an inter-view reference picture of the current picture, the following applies:

       $$\text{refViewIdx} = \text{ViewIdx}(\text{RefPicListX}[\text{RefIdxLX}[xN][yN]]) \quad \text{(H-252)}$$

       $$\text{mvDisp} = \text{MvLXN}[xN][yN] \quad \text{(H-253)}$$

       $$\text{availableDV} = 1 \quad \text{(H-254)}$$

     - Otherwise (RefPicListX[RefIdxLX[xN][yN]] is not an inter-view reference picture), the following applies:

  - When availableIvpMvSearchFlagN is equal to 1, availableFlagIvpMvN is equal to 0, and PredMode[xN][yN] is equal to MODE_SKIP and IvpMvFlagLX[xN][yN] is equal to 1, the following applies:

    ivpMvDispN = MvRefinedDisp[xN][yN] (H-255)

    refViewIdxN = RefViewIdx[xN][yN] (H-256)

    availableFlagIvpMvN = 1 (H-257)

When availableDV is equal to 0 for each N being A_1 and B_1, the following applies.

- When availableDV is equal to 0 and availableFlagIvpMvN is equal to 1, the following applies:

  mvDisp = ivpMvDispN (H-258)

  refViewIdx = refViewIdxN (H-259)

  availableDV = 1 (H-260)

When availableDV is equal to 0, refViewIdx is set equal to DefaultViewIdx, and mvDisp is set equal to (0, 0). The variable mvRefinedDisp is set equal to mvDisp.

When depth_refinement_flag[nuh_layer_id] is equal to 1, the following ordered steps apply:

1. The derivation process for a disparity sample array as specified in subclause H.8.5.5.2 is invoked with the luma locations xCb, yCb, the disparity vector mvDisp, the view identifier refViewIdx, the variable nPSW equal to nCbS, the variable nPSH equal to nCbS, and the variable splitFlag equal to 0 as the inputs, and the output is the array disparitySamples of size (nCbS)x(nCbS).
2. The horizontal component of the disparity vector mvRefinedDisp[0] is set equal to disparitySamples[0][0].

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made for x = xCb.. (xCb + nCbS − 1), y = yCb..(yCb + nCbS− 1):

MvDisp[x][y] = mvDisp (H-261)

MvRefinedDisp[x][y] = mvRefinedDisp (H-262)

RefViewIdx[x][y] = refViewIdx (H-263)

DefaultDispFlag[x][y] = !availableDV (H-264)

H.8.5.5.1 Derivation process for a disparity vector from temporal neighbour blocks

Inputs to this process are

- a luma location (xCb, yCb) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable nCbS specifying the size of the current luma coding block.

Outputs of this process are

- the disparity vector mvDisp,
- the reference view order index refViewIdx,
- the availability flag availableFlag.

The luma location (xCCtr , yCCtr) specifying the centre position of the current luma coding block is derived as follows:

xCCtr = xCb + (nCbS >> 1) (H-265)

yCCtr = yCb + (nCbS >> 1) (H-266)

The flag availableFlag is set equal to 0, and mvDisp is set equal to (0, 0).

For i from 0 to NumDdvCandPics − 1, inclusive, the following ordered steps apply and the whole decoding process of this sub-clause terminates once availableFlag is set to 1.

1. Let colPu the prediction unit in DdvCandPicsList[i] covering the position ((xCCtr >> 4) << 4 , (yCCtr >> 4) << 4).
2. The position (xPCol, yPCol) is set equal to the position of the top-left sample of colPu relative to the top-left luma sample of the DdvCandPicsList[i].

3. If slice_type is equal to B, the variable dir is set equal to collocated_from_l0_flag, otherwise, dir is set equal to 1 – collocated_from_l0_flag. [Ed. (GT) In software L0 is always checked first. Moreover the number of checks depends on the slice_type the of collocated slice.]

4. For each X from dir to 1 – dir, inclusive, the following applies:
   - The variables candPicRefPicList, candPredFlag, candRefIdx, and candMV are set equal to the variables RefPicListX, RefIdxLX, and MvLX of DdvCandPicsList[i], respectively.
   - When colPu is not coded in an intra prediction mode and candPredFlag[xPCol][yPCol] is equal to 1, the following applies:
     - The variable candRefViewIdx is set equal to the ViewIdx(candPicRefPicList[candRefIdx[xPCol][yPCol]]).
     - When candRefViewIdx is not equal to the ViewIdx(DdvCandPicsList[i]) and there is an inter-view reference picture with ViewIdx equal to candViewIdx in RefPicList0 or RefPicList1, the following applies:

$$\text{refViewIdx} = \text{candRefViewIdx} \qquad \text{(H-267)}$$

$$\text{mvDisp} = \text{candMV[xPCol][yPCol]} \qquad \text{(H-268)}$$

$$\text{availableFlag} = 1 \qquad \text{(H-269)}$$

H.8.5.5.2 Derivation process for a disparity sample array

Inputs to this process are:

- a luma location (xP, yP) relative to the top-left luma sample of the current picture,
- a disparity vector mvDisp,
- a view order index refViewIdx specifying a reference view,
- a view order index depthViewIdx specifying the view the depth should be derived from
- variables nPSW and nPSH specifying a width and a height, respectively
- a variable splitFlag.

Outputs of this process are:

- a (nPSW)x(nPSH) array disparitySamples of disparities values.

Let refDepPic the picture in the current access unit with ViewIdx(refDepPic) equal to ViewIdx and DepthFlag(refDepPic) equal to 1.

Let refDepPels be an array of reconstructed depth samples refDepPic. The luma location (x_{TL}, y_{TL}) of top-left luma sample of a block in refDepPels is derived by

$$x_{TL} = xP + ((mvDisp[0] + 2) >> 2) \qquad \text{(H-270)}$$

$$y_{TL} = yP + ((mvDisp[1] + 2) >> 2) \qquad \text{(H-271)}$$

The variables nSubBlkW and nSubBlkH are set equal to nPSW and nPSH, respectively.

When splitFlag is equal to 1, nSubBlkW, nSubBlkH are modified as specified in the following:

- The variable minSubBlkSizeFlag is derived as specified in the following:

$$\text{minSubBlkSizeFlag} = (nPSW \% 8 \;!=\; 0) \;||\; (nPSH \% 8 \;!=\; 0) \qquad \text{(H-272)}$$

- Depending on the value of minSubBlkSizeFlag, the following applies.
  - If minSubBlkSizeFlag is equal to 1, the following applies:

$$\text{horSplitFlag} = (nPSH \% 8 \;!=\; 0) \qquad \text{(H-273)}$$

  - Otherwise (minSubBlkSizeFlag is equal to 0), the following applies:

$$xP0 = \text{Clip3}(0, \text{pic\_width\_in\_luma\_samples} - 1, x_{TL}) \qquad \text{(H-274)}$$

$$yP0 = \text{Clip3}(0, \text{pic\_height\_in\_luma\_samples} - 1, y_{TL}) \qquad \text{(H-275)}$$

$$xP1 = \text{Clip3}(0, \text{pic\_width\_in\_luma\_samples} - 1, x_{TL} + nPSW - 1) \qquad \text{(H-276)}$$

$yP1 = Clip3(0, pic\_height\_in\_luma\_samples - 1, y_{TL} + nPSH - 1)$ (H-277)

$horSplitFlag = (refDepPels[xP0][yP0] < refDepPels[xP1][yP1]) == (refDepPels[xP1][yP0] < refDepPels[xP0][yP1]))$ (H-278)

- The variables nSubBlkW and nSubBlkH are modified as specified in the following:

  $nSubBlkW = horSplitFlag ? 8 : 4$ (H-279)

  $nSubBlkH = horSplitFlag ? 4 : 8$ (H-280)

The array disparitySamples is derived as specified in the following:

- For sBy in the range of 0 to ((nPSH / nSubBlkH) –1), inclusive, the following applies:
  - For sBx in the range of 0 to ((nPSW / nSubBlkW) –1), inclusive, the following applies:
    - The variable maxDep is set equal to –1 and modified as specified in the following.

```
xSubB = sBx * nSubBlkW
ySubB = sBy * nSubBlkH
xP0 = Clip3( 0, pic_width_in_luma_samples – 1, $x_{TL}$ + xSubB )
yP0 = Clip3( 0, pic_height_in_luma_samples – 1, $y_{TL}$ + ySubB )
xP1 = Clip3( 0, pic_width_in_luma_samples – 1, $x_{TL}$ + xSubB + nSubBlkW – 1 )
yP1 = Clip3( 0, pic_height_in_luma_samples – 1, $y_{TL}$ + ySubB + nSubBlkH – 1 )
maxDep = Max( maxDep, refDepPels[ xP0 ][ yP0 ] )
maxDep = Max( maxDep, refDepPels[ xP0 ][ yP1] )
maxDep = Max( maxDep, refDepPels[ xP1][ yP0 ] )
maxDep = Max( maxDep, refDepPels[ xP1][ yP1 ] )
```

    - The values of the array depthSamples are modified as specified in the following:

```
for ( yOff = 0; yOff < nSubBlkH; yOff++ )
    for( xOff = 0; xOff < nSubBlkW; xOff++ ) {
        x = xSubB + xOff
        y = ySubB + yOff
        disparitySamples[ x ][ y ] = DepthToDisparityB[ refViewIdx ][ maxDep ]
    }
```

H.8.5.6 Derivation process for disparity vectors from neighbouring depth samples

Inputs to this process are:

- a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable nCbS specifying the size of the current luma coding block.

Let p be the array of constructed samples prior to the deblocking filter process.

The disparity vector mvDisp is set equal to (0, 0). and modified as specified in the following:

[Ed. (GT): The derivation process below is e.g. not compatible with tiles, since a reasonable availability check (as e.g. for intra sample prediction is not applied.]

1. The variable avgDep is derived as follows:
   - If xCb is greater than 0 and yCb is greater than 0, the following applies:

     $$avgDep = (5 * p[xCb - 1][yCb - 1] + 5 * p[xCb - 1][yCb + nCbS - 1] + 6 * p[xCb + nCbS - 1][yCb - 1] + 8) >> 4) \quad (H-281)$$

   - Otherwise, if xCb is greater than 0 and yCb is equal to 0, the following applies:

     $$avgDep = p[xCb - 1][yCb + nCbS - 1] \quad (H-282)$$

   - Otherwise, if xCb is equal to 0 and yCb is greater than 0, the following applies:

     $$avgDep = p[xCb + nCbS - 1][yCb - 1] \quad (H-283)$$

   - Otherwise, (xCb is equal to 0 and yCb is equal to 0), the following applies:

     $$avgDep = -1 \quad (H-284)$$

2. When avgDep is not equal to −1, mvDisp[0] is modified as specified in the following:

mvDisp[0] = DepthToDisparityB[0][avgDep]

For use in derivation processes of variables invoked later in the decoding process, the following assignments are made for x = xC..(xCb + nCbS − 1), y = yCb..(yCb + nCbS− 1):

MvRefinedDisp[x][y] = mvDisp (H-285)

RefViewIdx[x][y] = 0 (H-286)

H.8.6 Scaling, transformation and array construction process prior to deblocking filter process

The specifications in subclause 8.6 apply.

H.8.7 In-loop filter process

The specifications in subclause 8.7 apply.

H.9 Parsing process

H.9.1 General

The specifications in clause 9.1 apply with the following modifications

- All references to the process specified in subclause 7.3 are replaced with references to the process specified in subclause H.7.3 .
- All invocations of the process specified in subclause 9.1 are replaced with invocations of the process specified in subclause H.9.1.

H.9.2 Parsing process for 0-th order Exp-Golomb codes

H.9.2.1 General

The specifications in subclause 9.2.1 apply with the following modifications .

- All references to the process specified in subclause 7.3 are replaced with references to the process specified in subclause H.7.3.
- All invocations of the process specified in subclause 9.2.2 are replaced with invocations of the process specified in subclause H.9.2.2.

H.9.2.2 Mapping process for signed Exp-Golomb codes

The specifications in subclause 9.2.1 apply with the following modifications.

- All references to the process specified in subclause 9.2 are replaced with references to the process specified in subclause H.9.2.1. [Ed. (GT) Reference in base spec to 9.2 is wrong.]

H.9.3 CABAC parsing process for slice segment data

H.9.3.1 General

The specifications in subclause 9.3.1 apply with the following modifications.

- All references to the process specified in subclauses 7.3.8.1 to through 7.3.8.11 are replaced with references to the process specified in subclauses H.7.3.8.1 to H.7.3.8.11
- All invocations of the process specified in subclause 9.3.2 are replaced with invocations of the process specified in subclause H.9.3.2.
- All invocations of the process specified in subclause 9.3.3 are replaced with invocations of the process specified in subclause H.9.3.3.
- All invocations of the process specified in subclause 9.3.4 are replaced with invocations of the process specified in subclause H.9.3.4 .
- All invocations of the process specified in subclause 9.3.2.3 are replaced with invocations of the process specified in subclause H.9.3.2.3.

H.9.3.2 Initialization process

H.9.3.2.1 General

The specifications in subclause 9.3.1 apply with the following modifications.

- All invocations of the process specified in subclause 9.3.2.2 are replaced with invocations of the process specified in subclause H.9.3.2.2.
- All invocations of the process specified in subclause 9.3.2.4 are replaced with invocations of the process specified in subclause H.9.3.2.4.

H.9.3.2.2 Initialization process for context variables

The specifications in subclause 9.3.2.2 apply with the following modifications.

- All references to the process specified in subclauses7.3.8.1 through 7.3.8.11 are replaced with references to the process specified in subclauses H.7.3.8.1 to H.7.3.8.11.
- Table H-10 is appended to the end of Table 9-4.
- Table H-11 to Table H-19 are appended to the end of the subclause.

Table H-10 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

Syntax structure	**Syntax element**	**ctxTable**	**initType**		
			0	**1**	**2**
coding_unit( ) depth_mode_parameters( )	depth_intra_mode	Table H-15	0..7	8..15	16..23
	wedge_full_tab_idx	Table H-11	0	1	2
	depth_dc_flag	Table H-16	0	1	2
	depth_dc_abs	Table H-12	0	1	2
	iv_res_pred_weight_idx	Table H-13		0..3	4..7
	ic_flag	Table H-14		0..2	3..5
	inter_sdc_flag	Table H-17	0	1	2
	inter_sdc_resi_abs_minus1	Table H-18	0	1	2
	inter_sdc_resi_sign_flag	Table H-19	0	1	2

[Ed (GT). Tables need to be sorted.].

Table H-11 – Values of initValue for wedge_full_tab_idx ctxIdx

Initialization variable	**ctxIdx of wedge_full_tab_idx**		
	0	**1**	**2**
initValue	154	154	154

Table H-12 – Values of initValue for depth_dc_abs ctxIdx

Initialization variable	**ctxIdx of depth_dc_abs**		
	0	**1**	**2**
initValue	154	154	154

Table H-13 – Values of initValue for iv_res_pred_weight_idx ctxIdx

Initialization variable	**ctxIdx of iv_res_pred_weight_idx**							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
initValue	162	153	154	162	162	153	154	162

Table H-14 – Values of initValue for ic_flag ctxIdx

Initialization variable	**ctxIdx of ic_flag**					
	0	**1**	**2**	**3**	**4**	**5**
initValue	154	154	154	154	154	154

Table H-15 – Values of initValue for depth_intra_mode ctxIdx

Initialization variable	**ctxIdx of depth_intra_mode**							
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
initValue	0	0	64	168	168	124	154	0
	8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
initValue	0	64	0	183	154	108	0	0
	16	**17**	**18**	**19**	**20**	**21**	**22**	**23**
initValue	64	0	154	154	168	109	0	0

Table H-16 – Values of initValue for depth_dc_flag ctxIdx

Initialization variable	ctxIdx of depth_dc_flag		
	0	1	2
initValue	0	0	64

Table H-17 – Values of initValue for inter_sdc_flag ctxIdx

Initialization variable	ctxIdx of inter_sdc_flag		
	0	1	2
initValue	154	154	154

Table H-18 – Values of initValue for inter_sdc_resi_abs_minus1 ctxIdx

Initialization variable	ctxIdx of inter_sdc_resi_abs_minus1		
	0	1	2
initValue	154	154	154

Table H-19 – Values of initValue for inter_sdc_resi_sign_flag ctxIdx

Initialization variable	ctxIdx of inter_sdc_resi_sign_flag		
	0	1	2
initValue	154	154	154

H.9.3.2.3 Storage process for context variables

The specifications in subclause 9.3.2.3 apply with the following modifications

- All references to the process specified in subclauses 7.3.8.1 through 7.3.8.11 are replaced with references to the process specified in subclauses H.7.3.8.1 to H.7.3.8.11

H.9.3.2.4 Synchronization process for context variables

The specifications in subclause 9.3.2.4 apply with the following modifications

- All references to the process specified in subclauses 7.3.8.1 through 7.3.8.11 are replaced with references to the process specified in subclauses H.7.3.8.1 to H.7.3.8.11

H.9.3.2.5 Initialization process for the arithmetic decoding engine

The specifications in subclause 9.3.2.5 apply.

H.9.3.3 Binarization process

H.9.3.3.1 General

The specifications in subclause 9.3.3.1 apply with the following modifications.

- Table H-20 is appended to the end of Table 9-32.

Table H-20 – Syntax elements and associated binarizations

Syntax structure	Syntax element	Binarization	
		Process	Input parameters
coding_unit( )	iv_res_pred_weight_idx	TR	cMax = 2, cRiceParam = 0
	ic_flag	FL	cMax = 1
	inter_sdc_flag	FL	cMax = 1
	inter_sdc_resi_abs_minus1	UEG0	[Ed. (GT) To be specified]
	inter_sdc_resi_sign_flag	FL	cMax = 1
depth_mode_parameters( )	depth_intra_mode	TR	cMax = depthIntraModeMaxLen, cRiceParam = 0
	wedge_full_tab_idx	FL	cMax = wedgeFullTabIdxBits[log2PbSize] (defined in Table H-21)
	depth_dc_flag	FL	cMax = 1
	depth_dc_abs	UEG0	[Ed. (GT) To be specified]
	depth_dc_sign_flag	FL	cMax = 1

Table H-21 –Values of wedgeFullTabIdxBits[log2PUSize]

Initialization variable	wedgeFullTabIdxBits				
log2PbSize	2	3	4	5	6
Value	7	10	11	11	13

H.9.3.3.2 Truncated Rice (TR) binarization process

The specifications in subclause 9.3.3.2 apply.

H.9.3.3.3 k-th order Exp-Golomb (EGk) binarization process

The specifications in subclause 9.3.3.3 apply.

H.9.3.3.4 Fixed-length (FL) binarization process

The specifications in subclause 9.3.3.4 apply.

H.9.3.3.5 Binarization process for part_mode

The specifications in subclause 9.3.3.5 apply.

H.9.3.3.6 Binarization process for intra_chroma_pred_mode

The specifications in subclause 9.3.3.6 apply.

H.9.3.3.7 Binarization process for inter_pred_idc

The specifications in subclause 9.3.3.7 apply.

H.9.3.3.8 Binarization process for cu_qp_delta_abs

The specifications in subclause 9.3.3.8 apply.

H.9.3.3.9 Binarization process for coeff_abs_level_remaining

The specifications in subclause 9.3.3.9 apply.

H.9.3.3.10 Binarization process for sdc_residual_abs_minus1

Input to this process is a request for the a syntax element sdc_residual_abs_minus1,

Output of this process is the binarization of the syntax element.

The bin string is a concatenation of a prefix bin string and, when present, a suffix bin string.

The variable numDepthValues is derived as follows:

$$\text{numDepthValues} = \text{DltFlag[nuh\_layer\_id]} \ ? \ \text{num\_depth\_values\_in\_dlt[nuh\_layer\_id]} : (1 \ll \text{BitDepth}_Y) - 1 \quad \text{(H-287)}$$

The variable cMaxPrefix is derived as follows:

$$\text{cMaxPrefix} = (\text{numDepthValues} * 3) \gg 2)$$

For the derivation of the prefix bin string, the following applies:

- If sdc_residual_abs_minus1 is less than cMaxPrefix, the prefix bin string is a bit string of length sdc_residual_abs_minus1+ 1 indexed by binIdx. The bins for binIdx less than sdc_residual_abs_minus1 are equal to 1. The bin with binIdx equal to sdc_residual_abs_minus1 is equal to 0.
- Otherwise, the prefix bin string is a bit string of length cMaxPrefix with all bins being equal to 1.

When sdc_residual_abs_minus1 is greater than cMaxPrefix, the suffix of the bin string is present and it is derived as follows:

- The suffix value suffixVal, is derived as follows:

  $$\text{suffixVal} = \text{sdc\_residual\_abs\_minus1} - \text{cMaxPrefix} \quad \text{(H-288)}$$

- The suffix of the bin string is specified by Fixed-length (FL) binarization process as specified in subclause with suffixVal and cMax equal to (numDepthValues − cMaxPrefix) as the inputs.

H.9.3.4 Decoding process flow

H.9.3.4.1 General

The specifications in subclause 9.3.4.1 apply. with the following modifications.

- All references to the process specified in subclause 9.3.3 are replaced with references to the process specified in subclause H.9.3.3.
- All invocations of the process specified in subclause 9.3.4.2 are replaced with invocations of the process specified in subclause H.9.3.4.2.
- All invocations of the process specified in subclause 9.3.4.3 are replaced with invocations of the process specified in subclause H.9.3.4.3.

H.9.3.4.2 Derivation process for ctxTable, ctxIdx and bypassFlag

H.9.3.4.2.1 General

The specifications in subclause 9.3.4.2.1 apply with the following modifications:

- Table H-22 is appended to the end of Table 9-37.

Table H-22 –Assignment of ctxInc to syntax elements with context coded bins

Syntax element	binIdx					
	0	1	2	3	4	>=5
wedge_full_tab_idx	0	0	0	0	0	0
depth_dc_flag	0	na	na	na	na	na
depth_dc_abs	0	0	0	0	0	0
depth_dc_sign_flag	bypass	0	0	0	0	0
res_pred_flag	0	na	na	na	na	na
ic_flag	0	na	na	na	na	na
inter_sdc_flag	0	0	0	0	0	0
inter_sdc_resi_abs_minus1	0	0	0	0	0	0
inter_sdc_resi_sign_flag	0	0	0	0	0	0

H.9.3.4.2.2 Derivation process of ctxInc using left and above syntax elements

The specifications in subclause 9.3.4.2.2 apply with the following modifications.

– Table H-23 is appended to the end of Table 9-38.

Table H-23 – Specification of ctxInc using left and above syntax elements

Syntax element	condL	condA	ctxIdxInc
iv_res_pred_weight_idx	iv_res_pred_weight_idx [xL][yL]	iv_res_pred_weight_idx [xA][yA]	(condL && availableL) + (condA && availableA)
ic_flag	ic_flag[xL][yL]	ic_flag[xA][yA]	(condL && availableL) + (condA && availableA)

H.9.3.4.2.3 .Derivation process of ctxInc for the syntax elements last_sig_coeff_x_prefix and last_sig_coeff_y_prefix

The specifications in subclause 9.3.4.2.3 apply.

H.9.3.4.2.4 Derivation process of ctxInc for the syntax element coded_sub_block_flag

The specifications in subclause 9.3.4.2.4 apply.

H.9.3.4.2.5 Derivation process of ctxInc for the syntax element sig_coeff_flag

The specifications in subclause 9.3.4.2.5 apply.

H.9.3.4.2.6 Derivation process of ctxInc for the syntax element coeff_abs_level_greater1_flag

The specifications in subclause 9.3.4.2.6 apply.

H.9.3.4.2.7 Derivation process of ctxInc for the syntax element coeff_abs_level_greater2_flag

The specifications in subclause 9.3.4.2.7 apply.

H.9.3.4.3 Arithmetic decoding process

The specifications in subclause 9.3.4.3 apply. with the following modifications.

– All references to the process specified in subclause 9.3.4.2 are replaced with references to the process specified in subclause H.9.3.4.2.

H.9.3.5 Arithmetic encoding process (informative)

The specifications in subclause 9.3.5 apply with the following modifications.

– All references to the process specified in subclause 9.3.4.3 are replaced with references to the process specified in subclause H.9.3.4.3.

H.10 Sub-bitstream extraction process

The specifications in clause 10 apply.

H.11 Profiles and levels

The specifications in Annex A apply, with the following modification:

TBD

H.12 Byte stream format

The specifications in Annex B apply.

H.13 Hypothetical reference decoder

The specifications in clause F.13 apply.

H.14 Supplemental enhancement information

H.14.1 General

The specifications in clause F.14 apply.

H.14.2 SEI payload syntax

The specifications in subclause G.14.2 together with the extensions and modifications specified in this subclause apply.

H.14.2.1 Alternative depth information SEI message syntax

alternative_depth_info (payloadSize) {	**Descriptor**
alternative_depth_info_cancel_flag	u(1)
if(alternative_depth_info_cancel_flag = = 0) {	
depth_type	u(2)
if(depth_type = = 1) {	
min_offset_x_int	se(v)
min_offset_x_frac	u(8)
max_offset_x_int	se(v)
max_offset_x_frac	u(8)
offset_y_present_flag	u(1)
if(offset_y_present_flag){	
min_offset_y_int	se(v)
min_offset_y_frac	u(8)
max_offset_y_int	se(v)
max_offset_y_frac	u(8)
}	
warp_map_size_present_flag	u(1)
if(warp_map_size_present_flag) {	
warp_map_width_minus2	ue(v)
warp_map_height_minus2	ue(v)
}	
}	
if(depth_type = = 0) {	
num_residual_texture_views_minus1	ue(v)
residual_depth_flag	u(1)
}	
}	
}	

H.14.3 SEI payload semantics

The specifications in subclause G.14.3 together with the extensions and modifications specified in this subclause apply.

H.14.3.1 Alternative depth information SEI message semantics

The alternative depth information SEI message indicates that decoded depth samples have to be interpreted as an alternative depth format. To discriminate different alternative depth formats, a depth_type syntax element is used. The information of the alternative depth information SEI message persists in output order until any of the following are true:

- A new CVS begins.
- The bitstream ends.
- A picture in an access unit containing an alternative depth information SEI message is output having PicOrderCntVal greater than PicOrderCnt(CurrPic).

alternative_depth_info_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous alternative depth information SEI message in output order. alternative_depth_info_cancel_flag equal to 0 indicates that alternative depth information follows.

depth_type identifies an alternative depth type according to Table H-24. A value of depth_type is equal to 0 indicates that this SEI message signals Global View and Depth (GVD) information. A value of depth_type is equal to 1 indicates that decoded depth samples can be used to derive a warp map and view synthesis can be performed by image-domain warping. .

Values of depth_type that are not listed in Table H-24 are reserved for future use by ITU-T | ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification. Decoders shall ignore alternative depth information

SEI messages that contain reserved values of depth_type.

Table H-24 – Interpretation of depth_type

Value	Description
0	Global view and depth
1	Warp map

NOTE 2 − When depth_type is equal to 0, decoding processes for inter-view prediction and decoding processes with depth-texture interaction should be disabled.

min_offset_x_int, **min_offset_x_frac** specify the integer and the fractional part of the minimum offset for the horizontal direction of a warp map.

The variable minOffsetX is derived as follows:

$$\text{minOffsetX} = \text{min\_offset\_x\_int} + \text{min\_offset\_x\_frac} \div 256 \quad \text{(H-289)}$$

max_offset_x_int, **max_offset_x_frac** specify the integer and the fractional part of the maximum offset for the horizontal direction of a warp map.

The variable maxOffsetX value is derived as follows:

$$\text{maxOffsetX} = \text{max\_offset\_x\_int} + \text{max\_offset\_x\_frac} \div 256$$

offset_y_present_flag equal to 1 specifies that min_offset_y_int, min_offset_y_frac, max_offset_y_int and max_offset_y_frac are present. offset_y_present_flag equal to 0 specifies that min_offset_y_int, min_offset_y_frac, max_offset_y_int and max_offset_y_frac are not present.

min_offset_y_int, **min_offset_y_frac** specify the integer and the fractional part of the minimum offset for the vertical direction of a warp map. When not present, min_offset_y_int and min_offset_y_frac are inferred to be equal to 0.

The variable minOffsetY value is derived as follows:

$$\text{minOffsetY} = \text{min\_offset\_y\_int} + \text{min\_offset\_y\_frac} \div 256 \quad \text{(H-290)}$$

max_offset_y_int, **max_offset_y_frac** specify the integer and the fractional part of the maximum offset for the vertical direction of a warp map. When not present, max_offset_y_int and max_offset_y_frac are inferred to be equal to 0.

The variable maxOffsetY value is derived as follows:

$$\text{maxOffsetY} = \text{max\_offset\_y\_int} + \text{max\_offset\_y\_frac} \div 256 \quad \text{(H-291)}$$

warp_map_size_present_flag equal to 1 specifies that a new warp map size is present, which is valid for the current and all following warp maps in output order until a new message with warp_map_size_present_flag equal to 1 is received or alternative_depth_info_cancel_flag is equal to.. warp_map_size_present_flag equal to 0 specifies that the warp map size is not changed.

warp_map_width_minus2 plus 2 specifies the width of the warp map. The value of warp_map_width_minus2 shall be in the range of 0 to pic_width_in_luma_samples − 2, inclusive. The variable warpMapWidth is set equal to (warp_map_width_minus2 + 2)

warp_map_height_minus2 plus 2 specifies the height of the warp map. The value of warp_map_height_minus2 shall be in the range of 0 to (pic_height_in_luma_samples >> offset_y_present_flag) − 2, inclusive. The variable warpMapHeight is set equal to (warp_map_height_minus2 + 2)

The variables deltaX, deltaY, scaleX and scale Y are derived as specified in the following:

$$\text{deltaX} = \text{pic\_width\_in\_luma\_samples} \div (\text{warpMapWidth} - 1) \quad \text{(H-292)}$$

$$\text{deltaY} = \text{pic\_height\_in\_luma\_samples} \div (\text{warpMapHeight} - 1) \quad \text{(H-293)}$$

$$\text{scaleX} = (\text{maxOffsetX} - \text{minOffsetX}) / ((1 \ll \text{BitDepth}_Y) - 1) \quad \text{(H-294)}$$

$$\text{scaleY} = (\text{maxOffsetY} - \text{minOffsetY}) / ((1 \ll \text{BitDepth}_Y) - 1) \quad \text{(H-295)}$$

Let recSamples[x][y] correspond to the reconstructed sample array S_L of a depth view component. The corresponding horizontal warp map component w[x][y][0] and the corresponding vertical warp map component w[x][y][1] for recSamples[x][y] are derived as specified in the following:

```
for( x = 0; x < warpMapWidth ; x++ )
```

```
    for( y = 0; y < warpMapHeight; y++){
        w[ x ][ y ][ 0 ] = x * deltaX + minOffsetX + scaleX * recSamples[ x ][ y ]
        if( offset_y_present_flag )
            w[ x ][ y ][ 1 ] =  y * deltaY + minOffsetY +
                                scaleY * recSamples[ x ][ y + pic_height_in_luma_samples / 2 ]
        else
            w[ x ][ y ][ 1 ] =  y * deltaY
    }
```

A warp map w[x][y] is derived for each input view using reconstructed samples of its corresponding depth view component, i.e. each input view has an associated warp map and vice versa. A warp map specifies a sparse set of positional correspondences. These correspondences identify semantically corresponding image locations between two views of the same time instance, i.e. the associated input view and a neighbouring input view which is identified as follows.

When the warp map w[x][y] is associated with the leftmost input view, then the warp map specifies for each sub-pel position (x*deltaX, y*deltaY) in this input view a corresponding sub-pel position (2 * w[x][y][0], 2 * w[x][y][1]) in the closest input view on the right.

When the warp map w[x][y] is associated with an input view different to the leftmost input view, then the warp map specifies for each sub-pel position (x * deltaX, y*deltaY) in this input view a corresponding sub-pel position (2 * w[x][y][0], 2 * w[x][y][1]) in the closest input view on the left.

NOTE 3 − A sample dense set of positional correspondences can be derived e.g. by bilinear interpolation.

num_residual_texture_views_minus1 plus1 specifies the number of sub-residual texture views packed in the residual texture layer. num_residual_texture_views_minus1 shall be in the range of 0 to 3, inclusive.

residual_depth_flag equal to 1 specifies that the number of sub-residual depth views packed in the residual depth layer is equal to num_residual_texture_views_minus1 + 1. residual_depth_flag equal to 0 specifies the number of sub-residual depth views is equal to 0 and that no residual depth layer is present in the bitstream.

When GVD information are signalled by the SEI message, information of multiple texture views are packed into two layers and information of multiple depth views is packed into one or two layers.

The two layers containing the texture views are the base texture layer and the residual texture layer. The base texture layer contains a base texture view in full resolution (e.g. the view from a central camera position) and is the layer with ViewIdx equal to 0 and DepthFlag equal to 0. The residual texture layer contains packed information of up to four additional views in quarter resolution (sub-residual texture views) and is the layer with ViewIdx equal to 1 and DepthFlag equal to 0.

Each sub-residual texture view is derived by applying the following to an additional texture input view:

– Project the base texture view to the position of the additional texture input view using the decoded base depth view (and the decoded residual depth view, when residual_depth_flag is equal to 1).

– Create a picture containing samples of the additional texture view that are located at positions not covered by projected sample positions of the base texture view.

– Decimate the created picture by a factor of two in horizontal and vertical direction by discarding odd sample positions.

With increasing order of camera IDs (which are specified by external means), the order of sub-residual texture views within the residual texture layer is top-right, top-left, bottom-left, bottom-right.

NOTE 4 − An example is shown in Figure H-2. The variable N is set equal to (num_residual_texture_views_minus1 + 2). The base texture view (B) is the input from central camera (camera ID = 3). In the case N = 2, camera ID = 2 and 3 are used. In the case N = 3, camera ID = 2, 3 and 4 are used. In the case N = 4, camera ID = 1, 2, 3 and 4 are used. In the case N = 5, camera ID = 1, 2, 3, 4 and 5 are used. The N − 1 input texture views with camera ID not equal to 3 are converted to N − 1 quarter-size sub-residual texture views (R-x) and packed in the residual texture layer in order top-left (R-2), bottom-left (R-4), top-right (R-1) and bottom-right (R-5). Their top-left co-ordinates in the residual texture layer of width = W and height = H is shown in Table H-25. The residual texture layer represents occluded area or out-of-frame area of the base texture view when it is projected to the N − 1 input texture views by GVD process.

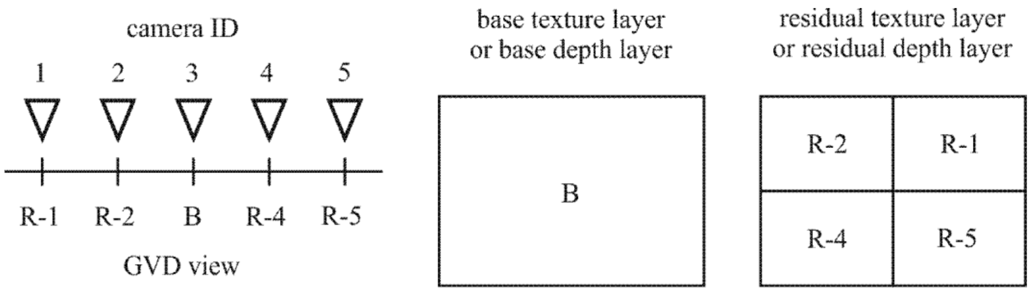

Figure H-2 Relation between camera ID and GVD texture/depth and packing of texture/depth views to the base and residual texture/depth layer

Table H-25– Top-left corner co-ordinates of GVD sub-residual views packed in a residual layer of width = W and height=H.

R-1	R-2	R-4	R-5
(W / 2, 0)	(0, 0)	(0, H / 2)	(W/2, H / 2)

The two layers containing the depth views are the base depth layer and the residual depth layer. The base depth layer contains a depth map in full resolution (base depth view) and is the layer with ViewIdx equal to 0 and DepthFlag equal to 1. The base depth view is generated as specified in the following:

– Project all depth maps associated with the additional texture views to the position of the base view.

– Derive the median of samples values projected to the same position.

When residual_depth_flag is equal to 0 and the number of input texture views is even (2 or 4), the position of the base depth view is the center of input views (e.g. camera ID = 2.5 when only cameras 2 and 3, or cameras 1, 2, 3 and 4 are present in the above example) and all depth maps are projected to this position, when the base depth view is generated.

When residual_depth_flag is equal to 1, the residual depth layer contains packed information derived from up to four depth views associated with the additional texture views in quarter resolution (sub-residual depth views). The residual depth layer is the layer with ViewIdx equal to 1 and DepthFlag equal to 1.

Each sub-residual depth view is derived by applying the following to an additional depth input view:

– Project the base depth view to the position of the additional depth input view using the decoded base depth view.

– Create a picture containing samples of the additional depth view that are located at positions not covered by projected sample positions of the base depth view.

– Decimate the created picture by a factor of two in horizontal and vertical direction by discarding odd sample positions.

The order of sub-residual depth views within the residual depth layer corresponds to the order of sub-residual texture layers in the residual texture layer.

H.15 Video usability information

The specifications in clause G.15 apply.