INTERNATIONAL STANDARD

ISO/IEC 23008-2

Second edition
2015-05-01

AMENDMENT 1
2015-10-01

Information technology — High efficiency coding and media delivery in heterogeneous environments —

Part 2: High efficiency video coding

AMENDMENT 1: 3D video extensions

Technologies de l'information — Codage à haute efficacité et livraison des medias dans des environnements hétérogènes —

Partie 2: Codage vidéo à haute efficacité

AMENDEMENT 1: Extensions vidéo 3D

Reference number
ISO/IEC 23008-2:2015/Amd.1:2015(E)

CONTENTS

Page

LIST OF TABLES

Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding — Amendment 1

Introduction

Replace the text of 0.2 with the following:

As the costs for both processing power and memory have reduced, network support for coded video data has diversified, and advances in video coding technology have progressed, the need has arisen for an industry standard for compressed video representation with substantially increased coding efficiency and enhanced robustness to network environments. Toward these ends the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) formed a Joint Collaborative Team on Video Coding (JCT-VC) in 2010 and a Joint Collaborative Team on 3D Video Coding Extension Development (JCT-3V) in 2012 for development of a new Recommendation | International Standard. This Recommendation | International Standard was developed in the JCT-VC and the JCT-3V.

In 0.3 add the following sentence to the end of the clause:

Support for 3D enables joint representation of video content and depth information with multiple camera views.

In 0.5 add the following paragraph to the end of the clause:

Rec. ITU-T H.265 | ISO/IEC 23008-2 version 3 refers to the integrated text containing 3D extensions, additional supplement enhancement information, and corrections to various minor defects in the prior content of the specification.

In 0.8, replace "Annexes A through H" with "Annexes A through I".

In 0.8, add the following sentence after the sentence that starts with "Annex H":

Annex I contains support for 3D coding.

Sequence parameter set RBSP

In 7.3.2.2.1 and F.7.3.2.2.1 replace the row containing "if(sps_extension_present_flag)" and all following rows in the syntax table by the following:

if(sps_extension_present_flag) {	
sps_range_extension_flag	u(1)
sps_multilayer_extension_flag	u(1)
sps_3d_extension_flag	u(1)
sps_extension_5bits	u(5)
}	
if(sps_range_extension_flag)	
sps_range_extension( )	
if(sps_multilayer_extension_flag)	
sps_multilayer_extension( ) /* specified in Annex F */	
if(sps_3d_extension_flag)	
sps_3d_extension( ) /* specified in Annex I */	
if(sps_extension_5bits)	
while(more_rbsp_data( ))	
sps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

In 7.4.3.2.1 replace the semantics of sps_extension_present_flag with the following semantics:

sps_extension_present_flag equal to 1 specifies that the syntax elements sps_range_extension_flag, sps_multilayer_extension_flag, sps_3d_extension_flag, and sps_extension_5bits are present in the SPS RBSP syntax structure. sps_extension_present_flag equal to 0 specifies that these syntax elements are not present.

In 7.4.3.2.1 add the following semantics after semantics of sps_multilayer_extension_flag:

sps_3d_extension_flag equal to 1 specifies that the sps_3d_extension( ) syntax structure (specified in Annex I) is present in the SPS RBSP syntax structure. sps_3d_extension_flag equal to 0 specifies that the sps_3d_extension( ) syntax structure is not present. When not present, the value of sps_3d_extension_flag is inferred to be equal to 0.

In 7.4.3.2.1 replace all occurrences of "sps_extension_6bits" with "sps_extension_5bits".

Picture parameter set RBSP

In 7.3.2.3.1 replace the row containing "if(pps_extension_present_flag)" and all following rows in the syntax table with the following:

if(pps_extension_present_flag) {	
pps_range_extension_flag	u(1)
pps_multilayer_extension_flag	u(1)
pps_3d_extension_flag	u(1)
pps_extension_5bits	u(5)
}	
if(pps_range_extension_flag)	
pps_range_extension( )	
if(pps_multilayer_extension_flag)	
pps_multilayer_extension( ) /* specified in Annex F */	
if(pps_3d_extension_flag)	
pps_3d_extension( ) /* specified in Annex I */	
if(pps_extension_5bits)	
while(more_rbsp_data( ))	
pps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

In 7.4.3.3.1 replace the semantics of pps_extension_present_flag with the following semantics:

pps_extension_present_flag equal to 1 specifies that the syntax elements pps_range_extension_flag, pps_multilayer_extension_flag, pps_3d_extension_flag, and pps_extension_5bits are present in the picture parameter set RBSP syntax structure. pps_extension_present_flag equal to 0 specifies that these syntax elements are not present.

In 7.4.3.3.1 add the following semantics after semantics of pps_multilayer_extension_flag:

pps_3d_extension_flag equal to 1 specifies that the pps_3d_extension( ) syntax structure (specified in Annex I) is present in the PPS RBSP syntax structure. pps_3d_extension_flag equal to 0 specifies that the pps_3d_extension( ) syntax structure is not present. When not present, the value of pps_3d_extension_flag is inferred to be equal to 0.

In 7.4.3.3.1 replace all occurrences of "pps_extension_6bits" with "pps_extension_5bits".

General SEI message syntax

In D.2.1 add the following rows to the syntax table after the row containing "multiview_view_position(payloadSize) / specified in Annex G */":*

else if(payloadType = = 181)	
alternative_depth_info(payloadSize) /* specified in Annex I */	

Common decoding process for a coded picture

In F.8.1.3 add the following paragraph before the paragraph starting with "After all slices of the current picture have been decoded,":

– Otherwise, general_profile_idc in the profile_tier_level( ) syntax structure VpsProfileTierLevel[profile_tier_level_idx[TargetOlsIdx][lIdx]] is equal to 8, the decoding process for the current picture takes as inputs the syntax elements and upper-case variables from clause I.7 and the decoding process of clause I.8.1.2 is invoked.

Video parameter set RBSP semantics

In F.7.4.3.1 replace the semantics of vps_extension2_flag with the following semantics:

vps_extension2_flag equal to 0 specifies that no vps_extension_data_flag syntax elements are present in the VPS RBSP syntax structure. vps_extension2_flag equal to 1 specifies vps_extension_data_flag syntax elements are present in the VPS RBSP syntax structure. Decoders conforming to a profile specified in Annexes A, G, or H shall ignore all data that follow the value 1 for vps_extension2_flag in a VPS NAL unit.

In F.7.4.3.1 add the following semantics:

vps_extension_data_flag may have any value. Its presence and value do not affect decoder conformance to profiles specified in Annexes A, G, or H. Decoders conforming to a profile specified in Annexes A, G, or H shall ignore all vps_extension_data_flag syntax elements.

In F.7.4.3.1.1 replace Table F.1 with the following table:

Table F.1 – Mapping of ScalabiltyId to scalability dimensions

scalability mask index	Scalability dimension	ScalabilityId mapping
0	Texture or depth	DepthLayerFlag
1	Multiview	ViewOrderIdx
2	Spatial/quality scalability	DependencyId
3	Auxiliary	AuxId
4-15	Reserved	

In F.7.4.3.1.1 remove Note 2.

In F.7.4.3.1.1 replace the paragraph starting with "The variable ScalabilityId[i][smIdx] specifying the identifier" and the following equation (F-2), with the following:

The variable ScalabilityId[i][smIdx] specifying the identifier of the smIdx-th scalability dimension type of the i-th layer, and the variables DepthLayerFlag[lId], ViewOrderIdx[lId], DependencyId[lId], and AuxId[lId] specifying the depth flag, view order index, the spatial/quality scalability identifier, and the auxiliary identifier, respectively, of the layer with nuh_layer_id equal to lId are derived as follows:

```
NumViews = 1
for( i = 0; i  <=  MaxLayersMinus1; 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++ ]
        else
            ScalabilityId[ i ][ smIdx ] = 0
    }
    DepthLayerFlag[ lId ] = ScalabilityId[ i ][ 0 ]
    ViewOrderIdx[ lId ] = ScalabilityId[ i ][ 1 ]
    DependencyId[ lId ] = ScalabilityId[ i ][ 2 ]                                  (F-2)
    AuxId[ lId ] = ScalabilityId[ i ][ 3 ]
    if( i > 0 ) {
        newViewFlag = 1
```

```
            for( j = 0; j < i; j++ )
                if( ViewOrderIdx[ lId ] = = ViewOrderIdx[ layer_id_in_nuh[ j ] ] )
                    newViewFlag = 0
            NumViews += newViewFlag
        }
    }
```

In F.7.4.3.1.1 replace the semantics of direct_dep_type_len_minus2 with the following:

direct_dep_type_len_minus2 plus 2 specifies the number of bits of the direct_dependency_type[i][j] and the direct_dependency_all_layers_type syntax elements. In bitstreams conforming to this version of this Specification the value of direct_dep_type_len_minus2 shall be equal 0 or 1. Although the value of direct_dep_type_len_minus2 shall be equal to 0 or 1 in this version of this Specification, decoders shall allow other values of direct_dep_type_len_minus2 in the range of 0 to 30, inclusive, to appear in the syntax.

In F.7.4.3.1.1 replace the semantics of direct_dependency_all_layers_type with the following:

direct_dependency_all_layers_type, when present, specifies the inferred value of direct_dependency_type[i][j] for all combinations of i-th and j-th layers. The length of the direct_dependency_all_layers_type syntax element is direct_dep_type_len_minus2 + 2 bits. Although the value of direct_dependency_all_layers_type is required to be in the range of 0 to 6, inclusive, in this version of this Specification, decoders shall allow values of direct_dependency_all_layers_type in the range of 0 to $2^{32} - 2$, inclusive, to appear in the syntax.

In F.7.4.3.1.1 replace the semantics of direct_dependency_type with the following:

direct_dependency_type[i][j] indicates the type of dependency between the layer with nuh_layer_id equal layer_id_in_nuh[i] and the layer with nuh_layer_id equal to layer_id_in_nuh[j]. direct_dependency_type[i][j] equal to 0 specifies that the layer with nuh_layer_id equal to layer_id_in_nuh[j] may be used for inter-layer sample prediction but is not used for inter-layer motion prediction of the layer with nuh_layer_id equal layer_id_in_nuh[i]. direct_dependency_type[i][j] equal to 1 specifies that the layer with nuh_layer_id equal to layer_id_in_nuh[j] may be used for inter-layer motion prediction but is not used for inter-layer sample prediction of the layer with nuh_layer_id equal layer_id_in_nuh[i]. direct_dependency_type[i][j] equal to 2 specifies that the layer with nuh_layer_id equal to layer_id_in_nuh[j] may be used for both inter-layer motion prediction and inter-layer sample prediction of the layer with nuh_layer_id equal layer_id_in_nuh[i]. The length of the direct_dependency_type[i][j] syntax element is direct_dep_type_len_minus2 + 2 bits. Although the value of direct_dependency_type[i][j] shall be in the range of 0 to 2, inclusive, when the layer with nuh_layer_id equal to layer_id_in_nuh[i] conforms to a profile specified in Annexes A, G, or H, and in the range of 0 to 6, inclusive, when the layer with nuh_layer_id equal to layer_id_in_nuh[i] conforms to a profile specified in Annex I, decoders shall allow values of direct_dependency_type[i][j] in the range of 0 to $2^{32} - 2$, inclusive, to appear in the syntax.

Profiles, tiers, and levels

In F.11.2 add the following row to the end of Table F.3:

3D Main	3D Main, Multiview Main, Main, Main Still Picture

In G.11.1.1 and H.11.1.1 remove " and sps_extension_6bits equal to 0 only".

In G.11.1.1 and H.11.1.1 remove " and pps_extension_6bits equal to 0 only".

Add a new Annex I as follows:

Annex I

3D high efficiency video coding

(This annex forms an integral part of this Recommendation | International Standard.)

I.1 Scope

This annex specifies syntax, semantics, and decoding processes for 3D high efficiency video coding that use the syntax, semantics, and decoding processes specified in clauses 2-10 and Annexes A-G. This annex also specifies profiles, tiers, and levels for 3D high efficiency video coding.

I.2 Normative references

The list of normative references in clause G.2 apply.

I.3 Definitions

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

I.3.1 **depth intra contour prediction**: A *prediction* of a *partition pattern* for a *prediction block* in a *picture* of a *depth layer* derived from samples of a *picture* included in the same *access unit* and in the *texture layer* of the same *view*.

I.3.2 **depth layer**: A *layer* with a nuh_layer_id value equal to i, such that DepthLayerFlag[i] is equal to 1 and DependencyId[i] and AuxId[i] are equal to 0.

I.3.3 **depth look-up table**: A list containing *depth values*.

I.3.4 **depth value**: A sample value of a *decoded picture* of a *depth layer*.

I.3.5 **disparity vector**: A *motion vector* used for inter-view prediction.

I.3.6 **inter-component prediction**: An *inter-layer prediction* where the *reference pictures* are associated with a DepthFlag value different from the DepthFlag value of the current *picture*.

I.3.7 **inter-view prediction**: An *inter-layer prediction* where the *reference pictures* are associated with *reference view order index* values different from the ViewIdx value of the current *picture*.

I.3.8 **intra prediction**: A *prediction* derived from only data elements (e.g., sample values) of the same decoded *slice* and additionally *may* be using *depth intra contour prediction*.

I.3.9 **partition pattern**: An MxM (M-column by M-row) array of *flags* defining two *sub-block partitions* of an MxM *prediction block*.

I.3.10 **prediction block**: A rectangular MxN *block* of samples on which either the same *prediction* or *partitioning* in *sub-block partitions* is applied.

I.3.11 **reference view order index**: A ViewIdx value associated with a *reference picture* used for *inter-view prediction*.

I.3.12 **sub-block partition**: A subset of samples of a *prediction block* on which the same *prediction* is applied.

I.3.13 **texture layer**: A *layer* with a nuh_layer_id value equal to i, such that DepthLayerFlag[i], DependencyId[i], and AuxId[i] are equal to 0.

I.4 Abbreviations

The specifications in clause G.4 apply.

I.5 Conventions

The specifications in clause G.5 apply.

I.6 Bitstream and picture formats, partitionings, scanning processes, and neighbouring relationships

I.6.1 Bitstream formats

The specifications in clause 6.1 apply.

I.6.2 Source, decoded, and output picture formats

The specifications in clause 6.2 apply.

I.6.3 Partitioning of pictures, slices, slice segments, tiles, coding tree units, and coding tree blocks

The specifications in clause 6.3 and its subclauses apply.

I.6.4 Availability processes

The specifications in clause 6.4 apply.

I.6.5 Scanning processes

The specifications in clause 6.5 and its subclauses apply.

I.6.6 Derivation process for a wedgelet partition pattern table

NOTE – Tables and values resulting from this process are independent of any information contained in the bitstream.

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

- For log2BlkSize in the range of 2 to 4, inclusive, the following applies:
  - NumWedgePattern[log2BlkSize] is set equal to 0.
  - The variable resShift is set equal to (log2BlkSize = = 4) ? 0 : 1.
  - The variable wBlkSize is set equal to (1 << (log2BlkSize + resShift)).
  - For wedgeOri in the range of 0 to 5, inclusive, the following applies:
    - The variable posEnd is set equal to NumWedgePattern[log2BlkSize].
    - If wedgeOri is equal to 0 or 4, the following applies:
      - The variables sizeScaleS and sizeScaleE are derived as follows:

$$\text{sizeScaleS} = (\text{log2BlkSize} > 3) \; ? \; 2 : 1 \tag{I-1}$$

$$\text{sizeScaleE} = (\text{wedgeOri} < 4 \;\&\&\; \text{log2BlkSize} > 3) \; ? : 2 : 1 \tag{I-2}$$

      - For m in the range of 0 to (wBlkSize / sizeScaleS − 1), inclusive, the following applies:
        - For n in the range of 0 to (wBlkSize / sizeScaleE − 1), inclusive, the following applies:
          - The wedgelet partition pattern generation process as specified in clause I.6.6.1 is invoked with patternSize equal to (1 << log2BlkSize), the variable resShift, the variable wedgeOri, the variable (xS, yS) equal to (m * sizeScaleS , 0), the variable (xE, yE) equal to (wedgeOri = = 0) ? (0, n * sizeScaleE) : (n * sizeScaleE, wBlkSize − 1) as inputs, and the output is the partition pattern curWedgePattern.
          - The wedgelet partition pattern table insertion process as specified in clause I.6.6.2 is invoked with the variable log2BlkSize and the partition pattern curWedgePattern as inputs.
    - Otherwise (wedgeOri is equal to 1, 2, 3, or 5), the following applies:
      - For curPos in the range of posStart to posEnd − 1, inclusive, the following applies:
        - The partition pattern curWedgePattern[x][y] is derived as follows:

```
for( y = 0; y < ( 1 << log2BlkSize ); y++ )
    for( x = 0; x < ( 1 << log2BlkSize ); x++ )                                    (I−3)
        curWedgePattern[ x ][ y ] = 1 −
            WedgePatternTable[ log2BlkSize ][ curPos ][ y ][ ( 1 << log2BlkSize ) − 1 − x ]
```

    - The variable posStart is set equal to posEnd.

– NumWedgePattern[5] is set equal to NumWedgePattern[4].

– For k = 0..NumWedgePattern[5] − 1, the following applies:

  – For x, y = 0..(1 << 5) − 1, the following applies:

```
WedgePatternTable[ 5 ][ k ][ x ][ y ] = WedgePatternTable[ 4 ][ k ][ x >> 1 ][ y >> 1 ]      (I−4)
```

I.6.6.1 Wedgelet partition pattern generation process

Inputs to this process are:

– a variable patternSize specifying the partition pattern size,

– a variable resShift specifying the precision of the partition pattern start and end locations relative to patternSize,

– a variable wedgeOri specifying the orientation of the partition pattern,

– a location (xS, yS) specifying the boundary start of a sub-block partition,

– a location (xE, yE) specifying the boundary end of a sub-block partition.

Output of this process is the partition pattern wedgePattern[x][y] of size (patternSize)x(patternSize).

The values of the partition pattern wedgePattern[x][y] are derived as specified by the following ordered steps:

1. For x, y = 0..patternSize − 1, wedgePattern[x][y] is set equal to 0.

2. The samples of the partition pattern wedgePattern that form a line between (xS, yS) and (xE, yE) are set equal to 1 as follows:

```
( x0, y0 ) = ( xS, yS )
( x1, y1 ) = ( xE, 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                                                        (I−5)
posY = y0
for( posX = x0; posX <= x1; posX++ ) {
    if( abs( yE − yS ) > abs( xE − xS ) )
        wedgePattern[ posY >> resShift ][ posX >> resShift ] = 1
    else
        wedgePattern[ posX >> resShift ][ posY >> resShift ] = 1
    sumErr += ( abs( y1 − y0 ) << 1 )
    if( sumErr >= ( x1 − x0 ) ) {
        posY += ( y0 < y1 ) ? 1 : −1
        sumErr −= ( x1 − x0 ) << 1
    }
}
```

3. The samples of wedgePattern are modified as follows:

```
for( y = 0; y <= ( yE >> resShift ); y++ )
    for( x = 0; ( x <= patternSize − 1 ) && ( wedgePattern[ x ][ y ] = = 0 ); x++ )      (I−6)
        wedgePattern[ x ][ y ] = 1
```

I.6.6.2 Wedgelet partition pattern table insertion process

Inputs to this process are:

– a variable log2BlkSize specifying the partition pattern size,

– a partition pattern wedgePattern[x][y], with x, y = 0..(1 << log2BlkSize) − 1.

The variable validPatternFlag is set equal to 0 and the following applies:

1. For x, y = 0..(1 << log2BlkSize) − 1, the following applies:

– When wedgePattern[x][y] is not equal to wedgePattern[0][0], validPatternFlag is set equal to 1.

2. For k = 0..NumWedgePattern[log2BlkSize] − 1, the following applies:
   – The variable 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 equal to 0.
   – When patIdenticalFlag is equal to 1, validPatternFlag is set equal to 0.
3. For k = 0..NumWedgePattern[log2BlkSize] − 1, the following applies:
   – The variable patInvIdenticalFlag is set equal 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 equal to 0.
   – When patInvIdenticalFlag is equal to 1, validPatternFlag is set equal to 0.

When validPatternFlag is equal to 1, the following applies:

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

I.7 Syntax and semantics

I.7.1 Method of specifying syntax in tabular form

The specifications in clause F.7.1 apply.

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

The specifications in clause F.7.2 apply.

I.7.3 Syntax in tabular form

I.7.3.1 NAL unit syntax

The specifications in clause F.7.3.1 and all its subclauses apply.

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

I.7.3.2.1 Video parameter set RBSP

video_parameter_set_rbsp( ) {	**Descriptor**
vps_video_parameter_set_id	u(4)
vps_base_layer_internal_flag	u(1)
vps_base_layer_available_flag	u(1)
vps_max_layers_minus1	u(6)
vps_max_sub_layers_minus1	u(3)
vps_temporal_id_nesting_flag	u(1)
vps_reserved_0xffff_16bits	u(16)
profile_tier_level(1, vps_max_sub_layers_minus1)	
vps_sub_layer_ordering_info_present_flag	u(1)
for(i = (vps_sub_layer_ordering_info_present_flag ? 0 : vps_max_sub_layers_minus1); i <= vps_max_sub_layers_minus1; i++) {	
vps_max_dec_pic_buffering_minus1[i]	ue(v)
vps_max_num_reorder_pics[i]	ue(v)
vps_max_latency_increase_plus1[i]	ue(v)
}	

vps_max_layer_id	u(6)
vps_num_layer_sets_minus1	ue(v)
for(i = 1; i <= vps_num_layer_sets_minus1; i++)	
for(j = 0; j <= vps_max_layer_id; j++)	
layer_id_included_flag[i][j]	u(1)
vps_timing_info_present_flag	u(1)
if(vps_timing_info_present_flag) {	
vps_num_units_in_tick	u(32)
vps_time_scale	u(32)
vps_poc_proportional_to_timing_flag	u(1)
if(vps_poc_proportional_to_timing_flag)	
vps_num_ticks_poc_diff_one_minus1	ue(v)
vps_num_hrd_parameters	ue(v)
for(i = 0; i < vps_num_hrd_parameters; i++) {	
hrd_layer_set_idx[i]	ue(v)
if(i > 0)	
cprms_present_flag[i]	u(1)
hrd_parameters(cprms_present_flag[i], vps_max_sub_layers_minus1)	
}	
}	
vps_extension_flag	u(1)
if(vps_extension_flag) {	
while(!byte_aligned( ))	
vps_extension_alignment_bit_equal_to_one	u(1)
vps_extension( )	
vps_extension2_flag	u(1)
if(vps_extension2_flag) {	
vps_3d_extension_flag	u(1)
if(vps_3d_extension_flag) {	
while(!byte_aligned( ))	
vps_3d_extension_alignment_bit_equal_to_one	u(1)
vps_3d_extension( )	
}	
vps_extension3_flag	u(1)
if(vps_extension3_flag)	
while(more_rbsp_data( ))	
vps_extension_data_flag	u(1)
}	
}	
rbsp_trailing_bits( )	
}	

I.7.3.2.1.1 Video parameter set extension syntax

The specifications in clause F.7.3.2.1.1 apply.

I.7.3.2.1.2 Representation format syntax

The specifications in clause F.7.3.2.1.2 apply.

I.7.3.2.1.3 DPB size syntax

The specifications in clause F.7.3.2.1.3 apply.

I.7.3.2.1.4 VPS VUI syntax

The specifications in clause F.7.3.2.1.4 apply.

I.7.3.2.1.5 Video signal info syntax

The specifications in clause F.7.3.2.1.5 apply.

I.7.3.2.1.6 VPS VUI bitstream partition HRD parameters syntax

The specifications in clause F.7.3.2.1.6 apply.

I.7.3.2.1.7 Video parameter set 3D extension syntax

vps_3d_extension( ) {	**Descriptor**
cp_precision	ue(v)
for(n = 1; n < NumViews; n++) {	
i = ViewOIdxList[n]	
num_cp[i]	u(6)
if(num_cp[i] > 0) {	
cp_in_slice_segment_header_flag[i]	u(1)
for(m = 0; m < num_cp[i]; m++) {	
cp_ref_voi[i][m]	ue(v)
if(!cp_in_slice_segment_header_flag[i]) {	
j = cp_ref_voi[i][m]	
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)
}	
}	
}	
}	
}	

I.7.3.2.2 Sequence parameter set RBSP syntax

I.7.3.2.2.1 General sequence parameter set RBSP syntax

The specifications in clause F.7.3.2.2.1 apply.

I.7.3.2.2.2 Sequence parameter set range extensions syntax

The specifications in clause F.7.3.2.2.2 apply.

I.7.3.2.2.3 Sequence parameter set multilayer extension syntax

The specifications in clause F.7.3.2.2.3 apply.

I.7.3.2.2.4 Sequence parameter set 3D extension syntax

sps_3d_extension( ) {	**Descriptor**
for(d = 0; d <= 1; d++) {	
iv_di_mc_enabled_flag[d]	u(1)
iv_mv_scal_enabled_flag[d]	u(1)
if(d = = 0) {	
log2_ivmc_sub_pb_size_minus3[d]	ue(v)
iv_res_pred_enabled_flag[d]	u(1)
depth_ref_enabled_flag[d]	u(1)
vsp_mc_enabled_flag[d]	u(1)
dbbp_enabled_flag[d]	u(1)
} else {	
tex_mc_enabled_flag[d]	u(1)
log2_texmc_sub_pb_size_minus3[d]	ue(v)
intra_contour_enabled_flag[d]	u(1)
intra_dc_only_wedge_enabled_flag[d]	u(1)
cqt_cu_part_pred_enabled_flag[d]	u(1)
inter_dc_only_enabled_flag[d]	u(1)
skip_intra_enabled_flag[d]	u(1)
}	
}	
}	

I.7.3.2.3 Picture parameter set RBSP syntax

I.7.3.2.3.1 General picture parameter set RBSP syntax

The specifications in clause F.7.3.2.3.1 apply.

I.7.3.2.3.2 Picture parameter set range extensions syntax

The specifications in clause F.7.3.2.3.2 apply.

I.7.3.2.3.3 Picture parameter set multilayer extension syntax

The specifications in clause F.7.3.2.3.3 apply.

I.7.3.2.3.4 General colour mapping table syntax

The specifications in clause F.7.3.2.3.4 apply.

I.7.3.2.3.5 Colour mapping octants syntax

The specifications in clause F.7.3.2.3.5 apply.

I.7.3.2.3.6 Picture parameter set 3D extension syntax

pps_3d_extension( ) {	**Descriptor**
dlts_present_flag	u(1)
if(dlts_present_flag) {	
pps_depth_layers_minus1	u(6)
pps_bit_depth_for_depth_layers_minus8	u(4)
for(i = 0; i <= pps_depth_layers_minus1; i++) {	
dlt_flag[i]	u(1)

if(dlt_flag[i]) {	
dlt_pred_flag[i]	u(1)
if(!dlt_pred_flag[i])	
dlt_val_flags_present_flag[i]	u(1)
if(dlt_val_flags_present_flag[i])	
for(j = 0; j <= depthMaxValue; j++)	
dlt_value_flag[i][j]	u(1)
else	
delta_dlt(i)	
}	
}	
}	
}	

I.7.3.2.3.7 Delta depth look-up table syntax

delta_dlt(i) {	**Descriptor**
num_val_delta_dlt	u(v)
if(num_val_delta_dlt > 0) {	
if(num_val_delta_dlt > 1)	
max_diff	u(v)
if(num_val_delta_dlt > 2 && max_diff > 0)	
min_diff_minus1	u(v)
delta_dlt_val0	u(v)
if(max_diff > (min_diff_minus1 + 1))	
for(k = 1; k < num_val_delta_dlt; k++)	
delta_val_diff_minus_min[k]	u(v)
}	
}	

I.7.3.2.4 Supplemental enhancement information RBSP syntax

The specifications in clause F.7.3.2.4 apply.

I.7.3.2.5 Access unit delimiter RBSP syntax

The specifications in clause F.7.3.2.5 apply.

I.7.3.2.6 End of sequence RBSP syntax

The specifications in clause F.7.3.2.6 apply.

I.7.3.2.7 End of bitstream RBSP syntax

The specifications in clause F.7.3.2.7 apply.

I.7.3.2.8 Filler data RBSP syntax

The specifications in clause F.7.3.2.8 apply.

I.7.3.2.9 Slice segment layer RBSP syntax

The specifications in clause F.7.3.2.9 apply.

I.7.3.2.10 RBSP slice segment trailing bits syntax

The specifications in clause F.7.3.2.10 apply.

I.7.3.2.11 RBSP trailing bits syntax

The specifications in clause F.7.3.2.11 apply.

I.7.3.2.12 Byte alignment syntax

The specifications in clause F.7.3.2.12 apply.

I.7.3.3 Profile, tier and level syntax

The specifications in clause F.7.3.3 apply.

I.7.3.4 Scaling list data syntax

The specifications in clause F.7.3.4 apply.

I.7.3.5 Supplemental enhancement information message syntax

The specifications in clause F.7.3.5 apply.

I.7.3.6 Slice segment header syntax

I.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++	
discardable_flag	u(1)
}	
if(num_extra_slice_header_bits > i) {	
i++	
cross_layer_bla_flag	u(1)
}	
for(; 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 && !poc_lsb_not_present_flag[LayerIdxInVps[nuh_layer_id]]) || (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)	
st_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 && !default_ref_layers_active_flag && NumRefListLayers[nuh_layer_id] > 0) {	
inter_layer_pred_enabled_flag	u(1)
if(inter_layer_pred_enabled_flag && NumRefListLayers[nuh_layer_id] > 1) {	
if(!max_one_active_ref_layer_flag)	
num_inter_layer_ref_pics_minus1	u(v)
if(NumActiveRefLayerPics != NumRefListLayers[nuh_layer_id])	
for(i = 0; i < NumActiveRefLayerPics; i++)	
inter_layer_pred_layer_idc[i]	u(v)
}	
}	
if(inCmpPredAvailFlag)	
in_comp_pred_flag	u(1)
if(sample_adaptive_offset_enabled_flag) {	
slice_sao_luma_flag	u(1)
if(ChromaArrayType != 0)	
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(!DepthFlag && NumRefListLayers[nuh_layer_id] > 0) {	
slice_ic_enabled_flag	u(1)
if(slice_ic_enabled_flag)	
slice_ic_disabled_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(chroma_qp_offset_list_enabled_flag)	
cu_chroma_qp_offset_enabled_flag	u(1)
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(cp_in_slice_segment_header_flag[ViewIdx])	
for(m = 0; m < num_cp[ViewIdx]; m++) {	
j = cp_ref_voi[ViewIdx][m]	

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(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(slice_segment_header_extension_present_flag) {	
slice_segment_header_extension_length	ue(v)
if(poc_reset_info_present_flag)	
poc_reset_idc	u(2)
if(poc_reset_idc != 0)	
poc_reset_period_id	u(6)
if(poc_reset_idc = = 3) {	
full_poc_reset_flag	u(1)
poc_lsb_val	u(v)
}	
if(!PocMsbValRequiredFlag && vps_poc_lsb_aligned_flag)	
poc_msb_cycle_val_present_flag	u(1)
if(poc_msb_cycle_val_present_flag)	
poc_msb_cycle_val	ue(v)
while(more_data_in_slice_segment_header_extension( ))	
slice_segment_header_extension_data_bit	u(1)
}	
byte_alignment( )	
}	

I.7.3.6.2 Reference picture list modification syntax

The specifications in clause F.7.3.6.2 apply.

I.7.3.6.3 Weighted prediction parameters syntax

The specifications in clause F.7.3.6.3 apply.

I.7.3.7 Short-term reference picture set syntax

The specifications in clause F.7.3.7 apply.

I.7.3.8 Slice segment data syntax

I.7.3.8.1 General slice segment data syntax

The specifications in clause F.7.3.8.1 apply.

I.7.3.8.2 Coding tree unit syntax

The specifications in clause F.7.3.8.2 apply.

I.7.3.8.3 Sample adaptive offset syntax

The specifications in clause F.7.3.8.3 apply.

I.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(cu_chroma_qp_offset_enabled_flag &&	
log2CbSize >= Log2MinCuChromaQpOffsetSize)	
IsCuChromaQpOffsetCoded = 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)	
}	

I.7.3.8.5 Coding unit syntax

coding_unit(x0, y0, log2CbSize) {	**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)	
else if(SkipIntraEnabledFlag)	
skip_intra_flag[x0][y0]	ae(v)
if(!cu_skip_flag[x0][y0] && !skip_intra_flag[x0][y0]) {	
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	
log2PbSize = log2CbSize − ((PartMode = = PART_NxN) ? 1 : 0)	
for(j = 0; j < nCbS; j = j + pbOffset)	
for(i = 0; i < nCbS; i = i + pbOffset) {	
if(IntraDcOnlyWedgeEnabledFlag || IntraContourEnabledFlag)	
intra_mode_ext(x0 + i , y0 + j , log2PbSize)	
if(no_dim_flag[x0 + i][y0 + j])	
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(no_dim_flag[x0 + i][y0 + j]) {	
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)
}	
if(ChromaArrayType = = 3)	
for(j = 0; j < nCbS; j = j + pbOffset)	
for(i = 0; i < nCbS; i = i + pbOffset)	
intra_chroma_pred_mode[x0 + 1][y0 + j]	ae(v)
else if(ChromaArrayType != 0)	
intra_chroma_pred_mode[x0][y0]	ae(v)
}	
} else {	
if(PartMode = = PART_2Nx2N)	
prediction_unit(x0, y0, nCbS, nCbS)	
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)	
}	
}	
}	
cu_extension(x0, y0, log2CbSize)	
if(DcOnlyFlag[x0][y0] | | (!skip_intra_flag[x0][y0] && CuPredMode[x0][y0] = = MODE_INTRA))	
depth_dcs(x0, y0, log2CbSize)	
if(!cu_skip_flag[x0][y0] && !skip_intra_flag[x0][y0] && !dc_only_flag[x0][y0] && !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) {	
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)	
}	
}	
}	

I.7.3.8.5.1 Intra mode extension syntax

intra_mode_ext(x0, y0, log2PbSize) {	**Descriptor**
if(log2PbSize < 6)	
no_dim_flag[x0][y0]	ae(v)
if(!no_dim_flag[x0][y0] && IntraDcOnlyWedgeEnabledFlag && IntraContourEnabledFlag)	
depth_intra_mode_idx_flag[x0][y0]	ae(v)
if(!no_dim_flag[x0][y0] && !depth_intra_mode_idx_flag[x0][y0])	
wedge_full_tab_idx[x0][y0]	ae(v)
}	

I.7.3.8.5.2 Coding unit extension syntax

cu_extension(x0, y0, log2CbSize) {	**Descriptor**
if(skip_intra_flag[x0][y0])	
skip_intra_mode_idx[x0][y0]	ae(v)
else {	
if(!cu_skip_flag[x0][y0]) {	
if(DbbpEnabledFlag && DispAvailFlag && log2CbSize > 3 && (PartMode = = PART_2NxN || PartMode = = PART_Nx2N))	
dbbp_flag[x0][y0]	ae(v)
if((CuPredMode[x0][y0] = = MODE_INTRA ? IntraDcOnlyWedgeEnabledFlag : InterDcOnlyEnabledFlag) && PartMode = = PART_2Nx2N)	
dc_only_flag[x0][y0]	ae(v)
}	
if(CuPredMode[x0][y0] != MODE_INTRA && PartMode = = PART_2Nx2N) {	
if(IvResPredEnabledFlag && RpRefPicAvailFlag)	
iv_res_pred_weight_idx[x0][y0]	ae(v)
if(slice_ic_enabled_flag && icCuEnableFlag && iv_res_pred_weight_idx[x0][y0] = = 0)	
illu_comp_flag[x0][y0]	ae(v)
}	
}	
}	

I.7.3.8.5.3 Depth DCs syntax

depth_dcs(x0, y0, log2CbSize) {	**Descriptor**
nCbS = (1 << log2CbSize)	
pbOffset = (PartMode = = PART_NxN && CuPredMode[x0][y0] = = MODE_INTRA) ? (nCbS / 2) : nCbS	
for(j = 0; j < nCbS; j = j + pbOffset)	
for(k = 0; k < nCbS; k = k + pbOffset)	
if(DimFlag[x0 + k][y0 + j] || DcOnlyFlag[x0][y0]) {	
if(CuPredMode[x0][y0] = = MODE_INTRA && DcOnlyFlag[x0][y0])	
depth_dc_present_flag[x0 + k][y0 + j]	ae(v)
dcNumSeg = DimFlag[x0 + k][y0 + j] ? 2 : 1	
if(depth_dc_present_flag[x0 + k][y0 + j])	
for(i = 0; i < dcNumSeg; i++) {	
depth_dc_abs[x0 + k][y0 + j][i]	ae(v)
if((depth_dc_abs[x0 + k][y0 + j][i] − dcNumSeg + 2) > 0)	
depth_dc_sign_flag[x0 + k][y0 + j][i]	ae(v)
}	
}	
}	

I.7.3.8.6 Prediction unit syntax

The specifications in clause F.7.3.8.6 apply.

I.7.3.8.7 PCM sample syntax

The specifications in clause F.7.3.8.7 apply.

I.7.3.8.8 Transform tree syntax

The specifications in clause F.7.3.8.8 apply.

I.7.3.8.9 Motion vector difference coding syntax

The specifications in clause F.7.3.8.9 apply.

I.7.3.8.10 Transform unit syntax

The specifications in clause F.7.3.8.10 apply.

I.7.3.8.11 Residual coding syntax

The specifications in clause F.7.3.8.11 apply.

I.7.3.8.12 Cross-component prediction syntax

The specifications in clause F.7.3.8.12 apply.

I.7.4 Semantics

I.7.4.1 General

I.7.4.2 NAL unit semantics

I.7.4.2.1 General NAL unit semantics

The specifications in clause F.7.4.2.1 apply.

I.7.4.2.2 NAL unit header semantics

The specifications in clause F.7.4.2.2 apply.

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

The specifications in clause F.7.4.2.3 apply.

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

The specifications in clause F.7.4.2.4 and all its subclauses apply.

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

I.7.4.3.1 Video parameter set RBSP semantics

The specifications in clause F.7.4.3.1 apply with the following modifications and additions:

vps_extension2_flag equal to 0 specifies that no vps_3d_extension( ) syntax structure and no vps_extension_data_flag syntax elements are present in the VPS RBSP syntax structure. vps_extension2_flag equal to 1 specifies that the vps_3d_extension( ) syntax structure and vps_extension_data_flag syntax elements may be present in the VPS RBSP syntax structure. When MaxLayersMinus1 is greater than 0, vps_extension2_flag shall be equal to 1.

vps_3d_extension_flag equal to 0 specifies that no vps_3d_extension( ) syntax structure is present in the VPS RBSP syntax structure. vps_3d_extension_flag equal to 1 specifies that the vps_3d_extension( ) syntax structure is present in the VPS RBSP syntax structure. When MaxLayersMinus1 is greater than 0, vps_3d_extension_flag shall be equal to 1.

vps_3d_extension_alignment_bit_equal_to_one shall be equal to 1.

vps_extension3_flag equal to 0 specifies that no vps_extension_data_flag syntax elements are present in the VPS RBSP syntax structure. vps_extension3_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. The value of 1 for vps_extension3_flag is reserved for future use by ITU-T | ISO/IEC. Decoders conforming to this version of this Specification shall ignore all data that follow the value 1 for vps_extension3_flag in a VPS RBSP.

vps_extension_data_flag may have any value. Its presence and value do not affect decoder conformance to profiles specified in Annexes A, G, H, or I. Decoders conforming to a profile specified in Annexes A, G, H, or I shall ignore all vps_extension_data_flag syntax elements.

I.7.4.3.1.1 Video parameter set extension semantics

The specifications in clause F.7.4.3.1.1 apply with the following additions and modifications:

direct_dependency_type[i][j] indicates the type of dependency between the layer predLayer with nuh_layer_id equal layer_id_in_nuh[i] and the layer refLayer with nuh_layer_id equal to layer_id_in_nuh[j]. ((direct_dependency_type[i][j] +1) & 0x1) greater than 0 specifies that samples of refLayer may be used for inter-layer prediction of predLayer. ((direct_dependency_type[i][j] +1) & 0x1) equal to 0 specifies that samples of refLayer are not used for inter-layer prediction of predLayer. ((direct_dependency_type[i][j] +1) & 0x2) greater than 0 specifies that motion vectors of refLayer may be used for inter-layer prediction of predLayer. ((direct_dependency_type[i][j] +1) & 0x2) equal to 0 specifies that motion vectors of refLayer are not used for inter-layer prediction of predLayer. ((direct_dependency_type[i][j] +1) & 0x4) greater than 0 specifies that coding quadtree and coding unit partitioning information of refLayer may be used for inter-layer prediction of predLayer. ((direct_dependency_type[i][j] +1) & 0x4) equal to 0 specifies that coding quadtree and coding unit partitioning information of refLayer are not used for inter-layer prediction of the predLayer.

The length of the direct_dependency_type[i][j] syntax element is direct_dep_type_len_minus2 + 2 bits. Although the value of direct_dependency_type[i][j] shall be in the range of 0 to 2, inclusive, when predLayer conforms to a profile specified in Annexes A, G, or H, and in the range of 0 to 6, inclusive, when the predLayer conforms to a profile specified in Annex I, decoders shall allow values of direct_dependency_type[i][j] in the range of 0 to $2^{32} - 2$, inclusive, to appear in the syntax.

The list ViewOIdxList[idx] is derived as follows:

```
idx = 0
ViewOIdxList[ idx++ ] = 0
for( i = 1; i  <=  MaxLayersMinus1; i++ ) {
    newViewFlag = 1
    for( j = 0; j < i; j++ )                                                        (I−7)
        if( ViewOrderIdx[ layer_id_in_nuh[ i ] ]  = =  ViewOrderIdx[ layer_id_in_nuh[ j ] ] )
            newViewFlag = 0
    if( newViewFlag )
        ViewOIdxList[ idx++ ] = ViewOrderIdx[ lId ]
}
```

The variables NumRefListLayers[iNuhLId] and IdRefListLayer[iNuhLId] are derived as follows:

```
for( i = 0; i  <=  MaxLayersMinus1; i++ ) {
    iNuhLId = layer_id_in_nuh[ i ]
    NumRefListLayers[ iNuhLId ] = 0
    for( j = 0; j < NumDirectRefLayers[ iNuhLId ]; j++ ) {
        jNuhLId = IdDirectRefLayer[ iNuhLId ][ j ]                                   (I−8)
        if( DepthLayerFlag[ iNuhLId ]  = =  DepthLayerFlag[ jNuhLId ] )
            IdRefListLayer[ iNuhLId ][ NumRefListLayers[ iNuhLId ]++ ] = jNuhLId
    }
}
```

The variables ViewCompLayerPresentFlag[iViewOIdx][depFlag] and ViewCompLayerId[iViewOIdx][depFlag] are derived as follows:

```
for( depFlag = 0; depFlag  <=  1; depFlag++ )
    for( i = 0; i < NumViews; i++ ) {
        iViewOIdx = ViewOIdxList[ i ]
        layerId = −1
        for( j = 0; j  <=  MaxLayersMinus1; j++ ) {
            jNuhLId = layer_id_in_nuh[ j ]                                             (I−9)
            if( DepthLayerFlag[ jNuhLId ]  = =  depFlag  &&  ViewOrderIdx[ jNuhLId ]  = =  iViewOIdx
                &&  DependencyId[ jNuhLId ]  = =  0  &&  AuxId[ jNuhLId ]  = =  0 )
                layerId = jNuhLId
        }
        ViewCompLayerPresentFlag[ iViewOIdx ][ depFlag ] = ( layerId  !=  −1 )
        ViewCompLayerId[ iViewOIdx ][ depFlag ] = layerId
    }
```

The function ViewIdx(picX) is specified as follows:

ViewIdx(picX) = ViewIdx of the picture picX (I−10)

The function ViewIdVal(picX) is specified as follows:

ViewIdVal(picX) = view_id_val[ViewIdx(picX)] (I−11)

I.7.4.3.1.2 Representation format semantics

The specifications in clause F.7.4.3.1.2 apply.

I.7.4.3.1.3 DPB size semantics

The specifications in clause F.7.4.3.1.3 apply.

I.7.4.3.1.4 VPS VUI semantics

The specifications in clause F.7.4.3.1.4 apply.

I.7.4.3.1.5 Video signal info semantics

The specifications in clause F.7.4.3.1.5 apply.

I.7.4.3.1.6 VPS VUI bitstream partition HRD parameters semantics

The specifications in clause F.7.4.3.1.6 apply.

I.7.4.3.1.7 Video parameter set 3D extension semantics

cp_precision + $BitDepth_Y - 1$ specifies the precision of the vps_cp_scale[i][j] and vps_cp_inv_scale_plus_scale[i][j] syntax elements present in the VPS and the cp_scale[j] and cp_inv_scale_plus_scale[j] syntax elements present in slice headers. The value of cp_precision shall be in the range of 0 to 5, inclusive.

num_cp[i], when cp_in_slice_segment_header_flag[i] is equal to 0, specifies the number 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] syntax elements present for the view with ViewIdx equal to i in the VPS. num_cp[i], when cp_in_slice_segment_header_flag[i] is equal to 1, specifies the number of cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] syntax elements present in slice headers of layers with ViewIdx equal to i.

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] for the view with ViewIdx equal to i 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] may be present in slice headers of layers with ViewIdx equal to i. cp_in_slice_segment_header_flag[i] 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 for the view with ViewIdx equal to i 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 headers of layers with ViewIdx equal to i. When not present, the value of cp_in_slice_segment_header_flag[i] is inferred to be equal to 0.

cp_ref_voi[i][m], when cp_in_slice_segment_header_flag[i] is equal to 0, specifies the ViewIdx value j of the view to which the m-th 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 element present for the view with ViewIdx equal to i in the VPS is related to. cp_ref_voi[i][m], when cp_in_slice_segment_header_flag[i] is equal to 1, specifies the ViewIdx value j of the view to which the m-th cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] syntax element present in the slice headers of layers with ViewIdx equal to i is related to. The value of cp_ref_voi[i][m] shall be in the range of 0 to 65535, inclusive. It is a requirement of bitstream conformance that cp_ref_voi[i][x] is not equal to cp_ref_voi[i][y] for any values of x and y in the range of 0 to num_cp[i] − 1, inclusive, when x is not equal to y.

For n and m in the range of 0 to NumViews − 1, inclusive, the variable CpPresentFlag[ViewOIdxList[n]][ViewOIdxList[m]] is set equal to 0 and modified as follows:

```
for( n = 1; n < NumViews; n++ ) {
    i = ViewOIdxList[ n ]
    for( m = 0; m < num_cp[ i ]; m++ )                                  (I−12)
        CpPresentFlag[ i ][ cp_ref_voi[ i ][ m ] ] = 1
}
```

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 parameters for derivation of a horizontal component of a disparity vector from a depth value and may be used to infer the values of the cp_scale[j], cp_off[j], cp_inv_scale_plus_scale[j], and cp_inv_off_plus_off[j] syntax elements in slice headers of layers with ViewIdx equal to i. When both a texture layer and a depth layer with ViewIdx equal to i are present, the conversion parameters are associated with the texture layer with ViewIdx equal to i.

I.7.4.3.2 Sequence parameter set RBSP semantics

I.7.4.3.2.1 General sequence parameter set RBSP semantics

The specifications in clause F.7.4.3.2.1 apply.

I.7.4.3.2.2 Sequence parameter set range extension semantics

The specifications in clause F.7.4.3.2.2 apply.

I.7.4.3.2.3 Sequence parameter set multilayer extension semantic

The specifications in clause F.7.4.3.2.3 apply.

I.7.4.3.2.4 Sequence parameter set 3D extension semantics

iv_di_mc_enabled_flag[d] equal to 1 specifies that the derivation process for inter-view predicted merging candidates and the derivation process for disparity information merging candidates may be used in the decoding process of layers with DepthFlag equal to d. iv_di_mc_enabled_flag[d] equal to 0 specifies that derivation process for inter-view predicted merging candidates and the derivation process for disparity information merging candidates is not used in the decoding process of layers with DepthFlag equal to d. When not present, the value of iv_di_mc_enabled_flag[d] is inferred to be equal to 0.

iv_mv_scal_enabled_flag[d] equal to 1 specifies that motion vectors used for inter-view prediction may be scaled based on view_id_val values in the decoding process of layers with DepthFlag equal to d. iv_mv_scal_enabled_flag[d] equal to 0 specifies that motion vectors used for inter-view prediction are not scaled based on view_id_val values in the decoding process of layers with DepthFlag equal to d. When not present, the value of iv_mv_scal_enabled_flag[d] is inferred to be equal to 0.

log2_ivmc_sub_pb_size_minus3[d], when iv_di_mc_enabled_flag[d] is equal to 1 and d is equal to 0, is used to derive the minimum size of sub-block partitions used in the derivation process for sub-block partition motion vectors for an inter-layer predicted merging candidate in the decoding process of layers with DepthFlag equal to d. When not present, the value of log2_ivmc_sub_pb_size_minus3[d] is inferred to be equal to (CtbLog2SizeY − 3). The value of log2_ivmc_sub_pb_size_minus3[d] shall be in the range of (MinCbLog2SizeY − 3) to (CtbLog2SizeY − 3), inclusive.

iv_res_pred_enabled_flag[d] equal to 1 specifies that the iv_res_pred_weight_idx syntax element may be present in coding units of layers with DepthFlag equal to d. iv_res_pred_enabled_flag[d] equal to 0 specifies that the iv_res_pred_weight_idx syntax element is not present coding units of layers with DepthFlag equal to d. When not present, the value of iv_res_pred_enabled_flag[d] is inferred to be equal to 0.

vsp_mc_enabled_flag[d] equal to 1 specifies that the derivation process for a view synthesis prediction merging candidate may be used in the decoding process of layers with DepthFlag equal to d. vsp_mc_enabled_flag[d] equal to 0 specifies that the derivation process for a view synthesis prediction merging candidate is not used the decoding process of layers with DepthFlag equal to d. When not present, the value of vsp_mc_enabled_flag[d] is inferred to be equal to 0.

dbbp_enabled_flag[d] equal to 1 specifies that the dbbp_flag syntax element may be present in coding units of layers with DepthFlag equal to d. dbbp_enabled_flag[d] equal to 0 specifies that the dbbp_flag syntax element is not present coding units of layers with DepthFlag equal to d. When not present, the value of dbbp_enabled_flag[d] is inferred to be equal to 0.

depth_ref_enabled_flag[d] equal to 1 specifies that the derivation process for a depth or disparity sample array from a depth picture may be used in the derivation process for a disparity vector for texture layers in the decoding process of layers with DepthFlag equal to d. depth_ref_enabled_flag[d] equal to 0 specifies that derivation process for a depth or disparity sample array from a depth picture is not used in the derivation process for a disparity vector for texture layers in the decoding process of layers with DepthFlag equal to d. When not present, the value of depth_ref_enabled_flag[d] is inferred to be equal to 0.

tex_mc_enabled_flag[d] equal to 1 specifies that the derivation process for motion vectors for the texture merge candidate may be used in the decoding process of layers with DepthFlag equal to d. tex_mc_enabled_flag[d] equal to 0 specifies that the derivation process for motion vectors for the texture merge candidate is not used in the decoding process of layers with DepthFlag equal to d. When not present, the value of tex_mc_enabled_flag[d] is inferred to be equal to 0.

log2_texmc_sub_pb_size_minus3[d], when tex_mc_enabled_flag[d] is equal to 1, is used to derive the minimum size of sub-block partitions used in the derivation process for sub-block partition motion vectors for an inter-layer predicted merging candidate in the decoding process of layers with DepthFlag equal to d. The value of log2_texmc_sub_pb_size_minus3[layerId] shall be in the range of (MinCbLog2SizeY − 3) to (CtbLog2SizeY − 3), inclusive.

intra_contour_enabled_flag[d] equal to 1 specifies that the intra prediction mode INTRA_CONTOUR using depth intra contour prediction may be used in the decoding process of layers with DepthFlag equal to d. intra_contour_enabled_flag[d] equal to 0 specifies that the intra prediction mode INTRA_CONTOUR using depth intra contour prediction is not used in the decoding process of layers with DepthFlag equal to d. When not present, intra_contour_enabled_flag[d] is inferred to be equal to 0.

intra_dc_only_wedge_enabled_flag[d] equal to 1 specifies that the dc_only_flag syntax element may be present in coding units coded in an intra prediction mode of layers with DepthFlag equal to d, and that the intra prediction mode INTRA_WEDGE may be used in the decoding process of layers with DepthFlag equal to d. intra_dc_only_wedge_enabled_flag[d] equal to 0 specifies that the dc_only_flag syntax element is not present in coding units coded in an intra prediction mode of layers with DepthFlag equal to d and that the intra prediction mode INTRA_WEDGE is not used in the decoding process of layers with DepthFlag equal to d. When not present, the value of intra_dc_only_wedge_enabled_flag[d] is inferred to be equal to 0.

cqt_cu_part_pred_enabled_flag[d] equal to 1 specifies that coding quadtree and coding unit partitioning information may be inter-component predicted in the decoding process of layers with DepthFlag equal to d. cqt_cu_part_pred_enabled_flag[d] equal to 0 specifies that coding quadtree and coding unit partitioning information are not inter-component predicted in the decoding process of layers with DepthFlag equal to d. When not present, the value of cqt_cu_part_pred_enabled_flag[d] is inferred to be equal to 0.

inter_dc_only_enabled_flag[d] equal to 1 specifies that the dc_only_flag syntax element may be present in coding units coded an in inter prediction mode of layers with DepthFlag equal to d. inter_dc_only_enabled_flag[d] equal to 0 specifies that the dc_only_flag syntax element is not present in coding units coded in an inter prediction mode of layers with DepthFlag equal to d. When not present, the value of inter_dc_only_enabled_flag[layerId] is inferred to be equal to 0.

skip_intra_enabled_flag[d] equal to 1 specifies that the skip_intra_flag syntax element may be present in coding units of layers with DepthFlag equal to d. skip_intra_enabled_flag[d] equal to 0 specifies that the skip_intra_flag syntax element is not present in coding units of layers with DepthFlag equal to d. When not present, the value of skip_intra_enabled_flag[layerId] is inferred to be equal to 0.

I.7.4.3.3 Picture parameter set RBSP semantics

I.7.4.3.3.1 General picture parameter set RBSP semantics

The specifications in clause F.7.4.3.3.1 apply.

I.7.4.3.3.2 Picture parameter set range extension semantics

The specifications in clause F.7.4.3.3.2 apply.

I.7.4.3.3.3 Picture parameter set multilayer extension semantics

The specifications in clause F.7.4.3.3.3 apply.

I.7.4.3.3.4 General colour mapping table semantics

The specifications in clause F.7.4.3.3.4 apply.

I.7.4.3.3.5 Colour mapping octants semantics

The specifications in clause F.7.4.3.3.5 apply.

I.7.4.3.3.6 Picture parameter set 3D extension semantics

dlts_present_flag equal to 1 specifies that syntax elements for the derivation of depth look-up tables are present in the PPS. dlts_present_flag equal to 0 specifies that syntax elements for the derivation of depth look-up tables are not present in the PPS.

The variables NumDepthLayers and DepIdxToLId[j] are derived as follows:

```
j = 0
for( i = 0; i  <=  MaxNumLayersMinus1; i++ ) {
    layerId = layer_id_in_nuh[ i ]
    if( DepthLayerFlag[ layerId ] )                                  (I−13)
        DepIdxToLId[ 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_layers_minus8 plus 8 specifies the bit depth of the samples in depth layers. It is a requirement of bitstream conformance that pps_bit_depth_for_depth_layers_minus8 shall be equal to bit_depth_luma_minus8 of the SPS the current PPS refers to.

The variable depthMaxValue is set equal to (1 << (pps_bit_depth_for_depth_layers_minus8 + 8)) − 1.

dlt_flag[i] equal to 1 specifies that the a depth look-up table for the layer with nuh_layer_id equal to DepIdxToLId[i] is present in the PPS and used for the decoding of the layer with nuh_layer_id equal to DepIdxToLId[i]. dlt_flag[i] equal to 0 specifies that a depth look-up table is not present for the layer with nuh_layer_id equal to DepIdxToLId[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, inclusive, the variable DltFlag[DepIdxToLId[i]] is set equal to dlt_flag[i].

dlt_pred_flag[i] equal to 1 indicates that the depth look-up table of the layer with nuh_layer_id equal to DepIdxToLId[i] is predicted from the depth look-up table of the layer with nuh_layer_id equal to DepIdxToLId[0]. dlt_pred_flag[i] equal to 0 indicates that the depth look-up table of the layer with nuh_layer_id equal to DepIdxToLId[i] is not predicted from any other depth look-up table. The value of dlt_pred_flag[0] shall be equal to 0. It is a requirement of bitstream conformance that, when dlt_flag[0] is equal to 0, dlt_pred_flag[i] shall be equal to 0.

dlt_val_flags_present_flag[i] equal to 1 specifies the depth look-up table of the layer with nuh_layer_id equal to DepIdxToLId[i] is derived from dlt_value_flag[i][j] syntax elements. dlt_val_flags_present_flag[i] equal to 0 specifies the depth look-up table of the layer with nuh_layer_id equal to DepIdxToLId[i] is derived from the delta_dlt( ) syntax structure. When not present, the value of dlt_val_flags_present_flag[i] is inferred to be equal to 0.

dlt_value_flag[i][j] equal to 1 specifies that j is an entry in the depth look-up table of the layer with nuh_layer_id equal to DepIdxToLId[i]. dlt_value_flag[i][j] equal to 0 specifies that j is not an entry in the depth look-up table of the layer with nuh_layer_id equal to DepIdxToLId[i].

When dlt_val_flags_present_flag[i] is equal to 1, the following applies:

– The variable layerId is set equal to DepIdxToLId[i].

– The variables DltVal[layerId][n] and NumValDlt[layerId] of the depth look-up table of the layer with nuh_layer_id equal to layerId are derived as follows:

```
for( n = 0, j = 0; j <= depthMaxValue; j++ )
    if( dlt_value_flag[ i ][ j ] )                                  (I−14)
        DltVal[ layerId ][ n++ ] = j
NumValDlt[ layerId ] = n
```

I.7.4.3.3.7 Delta depth look-up table semantics

num_val_delta_dlt specifies the number of elements in the list deltaList. The length of num_val_delta_dlt syntax element is pps_bit_depth_for_depth_layers_minus8 + 8 bits.

max_diff specifies the maximum difference between two consecutive elements in the list deltaList. The length of max_diff syntax element is pps_bit_depth_for_depth_layers_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 elements in the list deltaList. min_diff_minus1 shall be 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).

delta_dlt_val0 specifies the 0-th element in the list deltaList. The length of the delta_dlt_val0 syntax element is pps_bit_depth_for_depth_layers_minus8 + 8 bits.

delta_val_diff_minus_min[k] plus minDiff specifies the difference between the k-th element and the (k − 1)-th element in the list deltaList. The length of delta_val_diff_minus_min[k] syntax element is Ceil(Log2(max_diff − minDiff + 1)) bits. When not present, the value of delta_val_diff_minus_min[k] is inferred to be equal to 0.

The list deltaList is derived as follows:

```
deltaList[ 0 ] = delta_dlt_val0
for( k = 1; k < num_val_delta_dlt; k++ )                            (I−15)
```

deltaList[k] = deltaList[k − 1] + delta_val_diff_minus_min[k] + minDiff

The variables layerId and refLayerId are set equal to DepIdxToLId[i] and DepIdxToLId[0], respectively.

The variables DltVal[layerId][n] and NumValDlt[layerId] of the depth look-up table of the layer with nuh_layer_id equal to layerId are derived as follows:

```
for( n = 0, j = 0; j  <=  depthMaxValue; j++ ) {
    inRefDltFlag = 0
    if( dlt_pred_flag[ i ] )
        for( k = 0; k < NumValDlt[ refLayerId ]; k++ )
            inRefDltFlag = inRefDltFlag  ||  ( DltVal[ refLayerId ][ k ]  = =  j )
    inUpdateDltFlag = 0                                                            (I−16)
    for( k = 0; k < num_val_delta_dlt; k++ )
        inUpdateDltFlag = inUpdateDltFlag  ||  ( deltaList[ k ]  = =  j )
    if( inRefDltFlag  !=  inUpdateDltFlag )
        DltVal[ layerId ][ n++ ] = j
}
NumValDlt[ layerId ] = n
```

I.7.4.3.4 Supplemental enhancement information RBSP semantics

The specifications in clause F.7.4.3.4 apply.

I.7.4.3.5 Access unit delimiter RBSP semantics

The specifications in clause F.7.4.3.5 apply.

I.7.4.3.6 End of sequence RBSP semantics

The specifications in clause F.7.4.3.6 apply.

I.7.4.3.7 End of bitstream RBSP semantics

The specifications in clause F.7.4.3.7 apply.

I.7.4.3.8 Filler data RBSP semantics

The specifications in clause F.7.4.3.8 apply.

I.7.4.3.9 Slice segment layer RBSP semantics

The specifications in clause F.7.4.3.9 apply.

I.7.4.3.10 RBSP slice segment trailing bits semantics

The specifications in clause F.7.4.3.10 apply.

I.7.4.3.11 RBSP trailing bits semantics

The specifications in clause F.7.4.3.11 apply.

I.7.4.3.12 Byte alignment semantics

The specifications in clause F.7.4.3.12 apply.

I.7.4.4 Profile, tier and level semantics

The specifications in clause F.7.4.4 apply.

I.7.4.5 Scaling list data semantics

The specifications in clause F.7.4.5 apply.

I.7.4.6 Supplemental enhancement information message semantics

The specifications in clause F.7.4.6 apply.

I.7.4.7 Slice segment header semantics

I.7.4.7.1 General slice segment header semantics

The specifications in clause F.7.4.7.1 apply with the following modifications and additions.

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

The list curCmpLIds and the variable numCurCmpLIds are derived as follows:

curCmpLIds = DepthFlag ? { nuh_layer_id } : RefPicLayerId

numCurCmpLIds = DepthFlag ? 1 : NumActiveRefLayerPics

The list inCmpRefViewIdcs[i], the variable cpAvailableFlag, and the variable allRefCmpLayersAvailFlag are derived as follows:

– The variables cpAvailableFlag and allRefCmpLayersAvailFlag are set equal to 1.
– For i in the range of 0 to numCurCmpLIds − 1, inclusive, the following applies:
  – The variable inCmpRefViewIdcs[i] is set equal to ViewOrderIdx[curCmpLIds[i]].
  – When CpPresentFlag[ViewIdx][inCmpRefViewIdcs[i]] is equal to 0, cpAvailableFlag is set equal to 0.
  – The variable refCmpCurLIdAvailFlag is set equal to 0.
  – When ViewCompLayerPresentFlag[inCmpRefViewIdcs[i]][!DepthFlag] is equal to 1, the following applies:
    – The variable j is set equal to LayerIdxInVps[ViewCompLayerId[inCmpRefViewIdcs[i]][!DepthFlag]].
    – When all of the following conditions are true, refCmpCurLIdAvailFlag is set equal to 1:
      – direct_dependency_flag[LayerIdxInVps[nuh_layer_id]][j] is equal to 1.
      – sub_layers_vps_max_minus1[j] is greater than or equal to TemporalId.
      – TemporalId is equal to 0 or max_tid_il_ref_pics_plus1[j][LayerIdxInVps[nuh_layer_id]] is greater than TemporalId.
  – When refCmpCurLIdAvailFlag is equal to 0, allRefCmpLayersAvailFlag is set equal to 0.

The variable inCmpPredAvailFlag is derived as follows:

– If allRefCmpLayersAvailFlag is equal to 0, inCmpPredAvailFlag is set equal to 0.
– Otherwise (allRefCmpLayersAvailFlag) is equal to 1, the following applies:
  – If DepthFlag is equal to 0, the following applies:

inCmpPredAvailFlag = vsp_mc_enabled_flag[DepthFlag] | |
dbbp_enabled_flag[DepthFlag] | | depth_ref_enabled_flag[DepthFlag] (I−17)

  – Otherwise (DepthFlag is equal to 1), the following applies:

inCmpPredAvailFlag = intra_contour_enabled_flag[DepthFlag] | |
cqt_cu_part_pred_enabled_flag[DepthFlag] | | tex_mc_enabled_flag[DepthFlag] (I−18)

in_comp_pred_flag equal to 0 specifies that reference pictures required for inter-component prediction of the current picture may not be present and that inter-component prediction of the current picture is disabled. in_comp_pred_flag equal to 1 specifies all reference pictures required for inter-component prediction of the current picture are present and that inter-component prediction of the current picture is enabled. When not present, the value of in_comp_pred_flag is inferred to be equal to 0.

When in_comp_pred_flag is equal to 1, the following applies for i in the range of 0 to numCurCmpLIds − 1, inclusive:

– It is a requirement of bitstream conformance that there is a picture in the DPB with PicOrderCntVal equal to the PicOrderCntVal of the current picture, and a nuh_layer_id value equal to ViewCompLayerId[inCmpRefViewIdcs[i]][!DepthFlag].

The variables IvDiMcEnabledFlag, IvMvScalEnabledFlag, IvResPredEnabledFlag, VspMcEnabledFlag, DbbpEnabledFlag, DepthRefEnabledFlag, TexMcEnabledFlag, IntraContourEnabledFlag, IntraDcOnlyWedgeEnabledFlag, CqtCuPartPredEnabledFlag, InterDcOnlyEnabledFlag, SkipIntraEnabledFlag and DisparityDerivationFlag are derived as follows:

IvDiMcEnabledFlag = NumRefListLayers[nuh_layer_id] > 0 && iv_di_mc_enabled_flag[DepthFlag] (I−19)

IvMvScalEnabledFlag = iv_mv_scal_enabled_flag[DepthFlag] && ViewIdx != 0 (I−20)

IvResPredEnabledFlag = NumRefListLayers[nuh_layer_id] > 0

&& iv_res_pred_enabled_flag[DepthFlag] (I−21)

VspMcEnabledFlag = NumRefListLayers[nuh_layer_id] > 0 &&
vsp_mc_enabled_flag[DepthFlag] && in_comp_pred_flag && cpAvailableFlag (I−22)

DbbpEnabledFlag = dbbp_enabled_flag[DepthFlag] && in_comp_pred_flag (I−23)

DepthRefEnabledFlag = depth_ref_enabled_flag[DepthFlag] && in_comp_pred_flag
&& cpAvailableFlag (I−24)

TexMcEnabledFlag = tex_mc_enabled_flag[DepthFlag] && in_comp_pred_flag (I−25)

IntraContourEnabledFlag = intra_contour_enabled_flag[DepthFlag] && in_comp_pred_flag (I−26)

IntraDcOnlyWedgeEnabledFlag = intra_dc_only_wedge_enabled_flag[DepthFlag] (I−27)

CqtCuPartPredEnabledFlag = cqt_cu_part_pred_enabled_flag[DepthFlag] && in_comp_pred_flag &&
slice_type != I && !(nal_unit_type >= BLA_W_LP && nal_unit_type <= RSV_IRAP_VCL23) (I−28)

InterDcOnlyEnabledFlag = inter_dc_only_enabled_flag[DepthFlag] (I−29)

SkipIntraEnabledFlag = skip_intra_enabled_flag[DepthFlag] (I−30)

DisparityDerivationFlag = IvDiMcEnabledFlag | | IvResPredEnabledFlag | |
VspMcEnabledFlag | | DbbpEnabledFlag (I−31)

When TexMcEnabledFlag is equal to 1, or CqtCuPartPredEnabledFlag is equal to 1, or IntraContourEnabledFlag is equal to 1, let TexturePic be the picture in the current access unit with nuh_layer_id equal to ViewCompLayerId[ViewIdx][0].

num_inter_layer_ref_pics_minus1 plus 1 specifies the number of pictures that may be used in decoding of the current picture for inter-layer prediction. The length of the num_inter_layer_ref_pics_minus1 syntax element is Ceil(Log2(NumRefListLayers[nuh_layer_id])) bits. The value of num_inter_layer_ref_pics_minus1 shall be in the range of 0 to NumRefListLayers[nuh_layer_id] − 1, inclusive.

The variables numRefLayerPics and refLayerPicIdc[j] are derived as follows:

```
for( i = 0, j = 0; i < NumRefListLayers[ nuh_layer_id ]; i++ ) {
    refLayerIdx = LayerIdxInVps[ IdRefListLayer[ nuh_layer_id ][ i ] ]
    if( sub_layers_vps_max_minus1[ refLayerIdx ] >= TemporalId && ( TemporalId = = 0 | |          (I−32)
            max_tid_il_ref_pics_plus1[ refLayerIdx ][ LayerIdxInVps[ nuh_layer_id ] ] > TemporalId ) )
        refLayerPicIdc[ j++ ] = i
}
numRefLayerPics = j
```

The variable NumActiveRefLayerPics is derived as follows:

```
if( nuh_layer_id = = 0 | | numRefLayerPics = = 0 )
    NumActiveRefLayerPics = 0
else if( default_ref_layers_active_flag )
    NumActiveRefLayerPics = numRefLayerPics
else if( !inter_layer_pred_enabled_flag )                                                    (I−33)
    NumActiveRefLayerPics = 0
else if( max_one_active_ref_layer_flag | | NumRefListLayers[ nuh_layer_id ] = = 1 )
    NumActiveRefLayerPics = 1
else
    NumActiveRefLayerPics = num_inter_layer_ref_pics_minus1 + 1
```

All slices of a coded picture shall have the same value of NumActiveRefLayerPics.

inter_layer_pred_layer_idc[i] specifies the variable, RefPicLayerId[i], representing the nuh_layer_id of the i-th picture that may be used by the current picture for inter-layer prediction. The length of the inter_layer_pred_layer_idc[i] syntax element is Ceil(Log2(NumRefListLayers[nuh_layer_id])) bits. The value of inter_layer_pred_layer_idc[i] shall be in the range of 0 to NumRefListLayers[nuh_layer_id] − 1, inclusive. When i is greater than 0, inter_layer_pred_layer_idc[i] shall be greater than inter_layer_pred_layer_idc[i − 1]. When not present, the value of inter_layer_pred_layer_idc[i] is inferred to be equal to refLayerPicIdc[i].

The variables RefPicLayerId[i] for all values of i in the range of 0 to NumActiveRefLayerPics − 1, inclusive, are derived as follows:

```
for( i = 0, j = 0; i < NumActiveRefLayerPics; i++ )                                          (I−34)
    RefPicLayerId[ i ] = IdRefListLayer[ nuh_layer_id ][ inter_layer_pred_layer_idc[ i ] ]
```

The variable NumExtraMergeCand is derived as follows:

$$\text{NumExtraMergeCand} = \text{IvDiMcEnabledFlag} \;||\; \text{TexMcEnabledFlag} \;||\; \text{VspMcEnabledFlag} \qquad \text{(I−35)}$$

five_minus_max_num_merge_cand specifies the maximum number of merging motion vector prediction (MVP) candidates supported in the slice subtracted from (5 + NumExtraMergeCand).

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

$$\text{MaxNumMergeCand} = 5 + \text{NumExtraMergeCand} - \text{five\_minus\_max\_num\_merge\_cand} \qquad \text{(I−36)}$$

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

slice_ic_enabled_flag equal to 1 specifies that the illu_comp_flag[x0][y0] syntax element may be present in coding units of the current slice. slice_ic_enabled_flag equal to 0 specifies that the illu_comp_flag[x0][y0] syntax element is not present in coding units of the current slice. When not present, the value of slice_ic_enabled_flag is inferred to be equal to 0.

slice_ic_disabled_merge_zero_idx_flag equal to 1 specifies that the illu_comp_flag[x0][y0] syntax element is not present in coding units of the current slice when merge_flag[x0][y0] is equal to 1 and merge_idx[x0][y0] is equal to 0. slice_ic_disabled_merge_zero_idx_flag equal to 0 specifies that illu_comp_flag[x0][y0] syntax element may be present in coding units of the current slice when merge_flag[x0][y0] is equal to 1 and merge_idx[x0][y0] is equal to 0. When not present, the value of slice_ic_disabled_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 parameters for the derivation of a horizontal component of a disparity vector from a depth value. When not present and CpPresentFlag[ViewIdx][j] is equal to 1, 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 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 header having a ViewIdx equal to viewIdxB shall be the same, when viewIdxA is equal to viewIdxB.

The variable DepthToDisparityB[j][d] specifying the horizontal component of a disparity vector 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 variable DepthToDisparityF[j][d] specifying the horizontal component of a disparity vector between the view with ViewIdx equal j and the current view corresponding to the depth value d in the current view are derived as follows:

– The variable log2Div is set equal to (BitDepth_Y − 1 + cp_precision).

– For d in range of 0 to ((1 << BitDepth_Y) − 1), inclusive, the following applies:

  – For m in the range of 0 to (num_cp[ViewIdx] − 1), inclusive, the following applies:

$$j = \text{cp\_ref\_voi[ViewIdx][m]} \qquad \text{(I−37)}$$

$$\text{offset} = (\text{cp\_off[j]} \ll \text{BitDepth}_Y) + ((1 \ll \text{log2Div}) \gg 1) \qquad \text{(I−38)}$$

$$\text{scale} = \text{cp\_scale[j]} \qquad \text{(I−39)}$$

$$\text{DepthToDisparityB[j][d]} = (\text{scale} * d + \text{offset}) \gg \text{log2Div} \qquad \text{(I−40)}$$

$$\text{invOffset} = ((\text{cp\_inv\_off\_plus\_off[j]} - \text{cp\_off[j]}) \ll \text{BitDepth}_Y) + ((1 \ll \text{log2Div}) \gg 1) \qquad \text{(I−41)}$$

$$\text{invScale} = \text{cp\_inv\_scale\_plus\_scale[j]} - \text{cp\_scale[j]} \qquad \text{(I−42)}$$

$$\text{DepthToDisparityF[j][d]} = (\text{invScale} * d + \text{invOffset}) \gg \text{log2Div} \qquad \text{(I−43)}$$

I.7.4.7.2 Reference picture list modification semantics

The specifications in clause F.7.4.7.2 apply.

I.7.4.7.3 Weighted prediction parameters semantics

The specifications in clause F.7.4.7.3 apply.

I.7.4.8 Short-term reference picture set semantics

The specifications in clause F.7.4.8 apply.

I.7.4.9 Slice segment data semantics

I.7.4.9.1 General slice segment data semantics

The specifications in clause F.7.4.9.1 apply.

I.7.4.9.2 Coding tree unit semantics

The specifications in clause F.7.4.9.2 apply.

I.7.4.9.3 Sample adaptive offset semantics

The specifications in clause F.7.4.9.3 apply.

I.7.4.9.4 Coding quadtree semantics

The specifications in clause F.7.4.9.4 apply with the following modifications:

The variable predSplitCuFlag specifying whether the split_cu_flag[x0][y0] syntax element is inter-component predicted is derived as follows:

– If CqtCuPartPredEnabledFlag is equal to 1, the following applies:
  – Let colTextCu be the coding unit containing the luma coding block covering the luma location (x0, y0) in the picture TexturePic.
  – The variable log2TextCbSize is set equal to log2CbSize of the coding unit colTextCu.
  – The variable predSplitCuFlag is set equal to (log2CbSize <= log2TextCbSize).
– Otherwise (CqtCuPartPredEnabledFlag 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.

I.7.4.9.5 Coding unit semantics

The specifications in clause F.7.4.9.5 apply with the following modifications and additions:

cu_skip_flag[x0][y0] equal to 1 specifies that for the current coding unit, when decoding a P or B slice, no more syntax elements except the merging candidate index merge_idx[x0][y0], the iv_res_pred_weight_idx[x0][y0] syntax element, and the illu_comp_flag[x0][y0] syntax element may be parsed after cu_skip_flag[x0][y0]. cu_skip_flag[x0][y0] equal to 0 specifies that the coding unit is not skipped. 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 cu_skip_flag[x0][y0] is not present, it is inferred to be equal to 0.

skip_intra_flag[x0][y0] equal to 1 specifies that for the current coding unit no more syntax elements except skip_intra_mode_idx[x0][y0] are parsed after skip_intra_flag[x0][y0]. skip_intra_flag[x0][y0] equal to 0 specifies that more syntax elements may be parsed after skip_intra_flag[x0][y0]. When not present, the value of skip_intra_flag[x0][y0] is inferred to be equal to 0.

pred_mode_flag equal to 0 specifies that the current coding unit is coded in inter prediction mode. pred_mode_flag equal to 1 specifies that the current coding unit is coded in intra prediction mode. The variable CuPredMode[x][y] is derived as follows for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1:

– If pred_mode_flag is equal to 0, CuPredMode[x][y] is set equal to MODE_INTER.
– Otherwise (pred_mode_flag is equal to 1), CuPredMode[x][y] is set equal to MODE_INTRA.

When pred_mode_flag is not present, the variable CuPredMode[x][y] is derived as follows for x = x0..x0 + nCbS − 1 and y = y0..y0 + nCbS − 1:

– If slice_type is equal to I or skip_intra_flag[x0][y0] is equal to 1, CuPredMode[x][y] is inferred to be equal to

MODE_INTRA.

– Otherwise (slice_type is equal to P or B and skip_intra_flag[x0][y0] is equal to 0), when cu_skip_flag[x0][y0] is equal to 1, CuPredMode[x][y] is inferred to be equal to MODE_SKIP.

The variables predPartModeFlag and partPredIdc are derived as follows:

– If CqtCuPartPredEnabledFlag is equal to 1, the following applies:

  – Let colTextCu be the coding unit containing the luma coding block covering the luma location (x0, y0) in the picture TexturePic.

  – The variables log2TextCbSize and partTextMode are set equal to log2CbSize and PartMode, respectively, of the coding unit colTextCu.

  – The variable predPartModeFlag is derived as follows:

    predPartModeFlag = log2TextCbSize = = log2CbSize && partTextMode = = PART_2Nx2N (I−44)

  – The variable partPredIdc is derived as follows:

    – If one or more of the following conditions are true, partPredIdc is set equal to 0:

      – log2TextCbSize is not equal to log2CbSize.

      – partTextMode is equal to PART_2Nx2N or PART_NxN.

    – Otherwise, if partTextMode is equal to PART_2NxN, PART_2NxnU, or PART_2NxnL, partPredIdc is set equal to 1.

    – Otherwise, partPredIdc is set equal to 2.

– Otherwise (CqtCuPartPredEnabledFlag is equal to 0), predPartModeFlag and partPredIdc are set equal to 0.

part_mode specifies partitioning mode of the current coding unit. The semantics of part_mode depend on CuPredMode[x0][y0]. The variables PartMode and IntraSplitFlag are derived from the value of part_mode and partPredIdc as defined in Table I.1

Table I.1 – Name association to prediction mode and partitioning type

CuPredMode[x0][y0]	part_mode	partPredIdc	IntraSplitFlag	PartMode
MODE_INTRA	0	0	0	PART_2Nx2N
	1	0	1	PART_NxN
MODE_INTER	0	0	0	PART_2Nx2N
	1	0	0	PART_2NxN
	2	0	0	PART_Nx2N
	3	0	0	PART_NxN
	4	0	0	PART_2NxnU
	5	0	0	PART_2NxnD
	6	0	0	PART_nLx2N
	7	0	0	PART_nRx2N
	0	1	0	PART_2Nx2N
	1	1	0	PART_2NxN
	2	1	0	PART_2NxnU
	3	1	0	PART_2NxnD
	0	2	0	PART_2Nx2N
	1	2	0	PART_Nx2N
	2	2	0	PART_nLx2N
	3	2	0	PART_nRx2N

The value of part_mode is restricted as follows:

– If CuPredMode[x0][y0] is equal to MODE_INTRA, part_mode shall be equal to 0 or 1.

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

  – If partPredIdc is equal to 0, the following applies:

    – If log2CbSize is greater than MinCbLog2SizeY and amp_enabled_flag is equal to 1, part_mode shall be in the range of 0 to 2, inclusive, or in the range of 4 to 7, inclusive.

    – Otherwise, if log2CbSize is greater than MinCbLog2SizeY and amp_enabled_flag is equal to 0, or log2CbSize is equal to 3, part_mode shall be in the range of 0 to 2, inclusive.

    – Otherwise (log2CbSize is greater than 3 and less than or equal to MinCbLog2SizeY), the value of part_mode shall be in the range of 0 to 3, inclusive.

  – Otherwise (partPredIdc is not equal to 0), the following applies:

    – If log2CbSize is greater than MinCbLog2SizeY and amp_enabled_flag is equal to 1, part_mode shall be in the range of 0 to 3, inclusive.

    – Otherwise (log2CbSize is equal to MinCbLog2SizeY or amp_enabled_flag is equal to 0), part_mode shall be in the range of 0 to 1, inclusive.

When part_mode is not present, the variables PartMode and IntraSplitFlag are derived as follows:

– PartMode is set equal to PART_2Nx2N.

– IntraSplitFlag is set equal to 0.

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 !DcOnlyFlag[x0][y0].

I.7.4.9.5.1 Intra mode extension semantics

no_dim_flag[x0][y0] equal to 1 specifies that the intra modes INTRA_WEDGE or INTRA_CONTOUR are not used for the current prediction unit. no_dim_flag[x0][y0] equal to 0 specifies that the intra mode INTRA_WEDGE or INTRA_CONTOUR is used for the current prediction unit. When not present, the value of no_dim_flag[x0][y0] is inferred to be equal to 1. When log2CbSize is greater than Log2MaxTrafoSize and DcOnlyFlag[x0][y0] is equal to 0, the value of no_dim_flag[x0][y0] shall be equal to 1.

For x = x0..x0 + (1 << log2PbSize) − 1, y = y0..y0 + (1 << log2PbSize) − 1, the variable DimFlag[x][y] is derived as follows:

DimFlag[x][y] = !no_dim_flag[x0][y0] (I−45)

depth_intra_mode_idx_flag[x0][y0] equal to 0, when DimFlag[x0][y0] is equal to 1, specifies that the intra mode INTRA_WEDGE is used for the current prediction unit. depth_intra_mode_idx_flag[x0][y0] equal to 1, when DimFlag[x0][y0] is equal to 1, specifies that the intra mode INTRA_CONTOUR is used for the current prediction unit. When not present, the value of depth_intra_mode_idx_flag[x0][y0] is inferred to be equal to (!IntraDcOnlyWedgeEnabledFlag || IntraContourEnabledFlag). When DimFlag[x0][y0] is equal to 1 and nal_unit_type is equal to BLA_W_LP, BLA_W_RADL, BLA_N_LP, IDR_W_RADL or IDR_N_LP, it is a requirement of bitstream conformance that depth_intra_mode_idx_flag[x0][y0] is equal to 0.

wedge_full_tab_idx[x0][y0] specifies the index of the partition pattern in the list WedgePatternTable[log2PbSize] when the intra mode INTRA_WEDGE is used for the current prediction unit.

I.7.4.9.5.2 Coding unit extension semantics

skip_intra_mode_idx[x0][y0] equal to 0, when skip_intra_flag[x0][y0] is equal to 1, specifies that the intra prediction mode INTRA_ANGULAR26 is used for the current prediction unit. skip_intra_mode_idx[x0][y0] equal to 1, when skip_intra_flag[x0][y0] is equal to 1, specifies that the intra prediction mode INTRA_ANGULAR10 is used for the current prediction unit. skip_intra_mode_idx[x0][y0] equal to 2 or 3, when skip_intra_flag[x0][y0] is equal to 1, specifies that the intra prediction mode INTRA_SINGLE is used for the current prediction unit.

dbbp_flag[x0][y0] equal to 1 specifies that the decoding process for inter sample prediction for depth predicted sub-block partitions is used for the current coding unit. dbbp_flag[x0][y0] equal to 0 specifies that the decoding process for inter sample prediction for depth predicted sub-block partitions is not used for the current coding unit. When not present, the value of dbbp_flag[x0][y0] is inferred to be equal to 0.

For x = x0..x0 + (1 << log2CbSize) − 1, y = y0..y0 + (1 << log2CbSize) − 1, the variable DbbpFlag[x][y] is derived as follows:

DbbpFlag[x][y] = dbbp_flag[x0][y0] (I−46)

dc_only_flag[x0][y0] equal to 1 specifies that the transform_tree( ) syntax structure is not present for the current coding unit and that the depth_dcs( ) syntax structure is present for the current coding unit. dc_only_flag[x0][y0] equal to 0 specifies the transform_tree( ) syntax structure may be present for the current coding unit and that the depth_dcs( ) syntax structure may be present for the current coding unit. When not present, the value of dc_only_flag[x0][y0] is inferred to be equal to 0. It is a requirement of bitstream conformance, that when pcm_flag[x0][y0] is equal to 1, the value of dc_only_flag[x0][y0] shall be equal to 0.

For x = x0..x0 + (1 << log2CbSize) − 1, y = y0..y0 + (1 << log2CbSize) − 1, the variable DcOnlyFlag[x][y] is derived as follows:

DcOnlyFlag[x][y] = dc_only_flag[x0][y0] (I−47)

iv_res_pred_weight_idx[x0][y0] not equal to 0 specifies that the bilinear sample interpolation and residual prediction process is used for the current coding unit and the index of the weighting factor used in the bilinear sample interpolation and residual prediction process. iv_res_pred_weight_idx[x0][y0] equal to 0 specifies that the bilinear sample interpolation and residual prediction process is not used for the current coding unit. When not present, the value of iv_res_pred_weight_idx[x0][y0] is inferred to be equal to 0.

When CuPredMode[x0][y0] is not equal to MODE_INTRA, the variable icCuEnableFlag is derived as follows:

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

  icCuEnableFlag = (merge_idx[x0][y0] != 0) || !slice_ic_disabled_merge_zero_idx_flag (I−48)

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

  – For X in the range of 0 to 1, inclusive, the variable refViewIdxLX is set equal to ViewIdx(RefPicListX[ref_idx_lX[x0][y0]]).

– The flag icCuEnableFlag is derived as follows:

$$\text{icCuEnableFlag} = (\text{inter\_pred\_idc}[\,x0\,][\,y0\,] \;!=\; \text{Pred\_L0} \;\&\&\; \text{refViewIdxL1} \;!=\; \text{ViewIdx}) \;||\; (\text{inter\_pred\_idc}[\,x0\,][\,y0\,] \;!=\; \text{Pred\_L1} \;\&\&\; \text{refViewIdxL0} \;!=\; \text{ViewIdx}) \quad \text{(I−49)}$$

illu_comp_flag[x0][y0] equal to 1 specifies that the illumination compensated sample prediction process is used for the current coding unit. illu_comp_flag[x0][y0] equal to 0 specifies that the illumination compensated sample prediction process is not used for the current coding unit. When not present, the value of illu_comp_flag[x0][y0] is inferred to be equal to 0.

For x = x0..x0 + (1 << log2CbSize) − 1, y = y0..y0 + (1 << log2CbSize) − 1, the variable IlluCompFlag[x][y] is derived as follows:

$$\text{IlluCompFlag}[\,x\,][\,y\,] = \text{illu\_comp\_flag}[\,x0\,][\,y0\,] \quad \text{(I−50)}$$

I.7.4.9.5.3 Depth DCs semantics

depth_dc_present_flag[x0 + k][y0 + j] equal to 1 specifies that the depth_dc_abs[x0 + k][y0 + j][i] syntax element is present and that the depth_dc_sign_flag[x0 + k][y0 + j][i] syntax element may be present. depth_dc_present_flag[x0 + k][y0 + j] equal to 0 specifies that the depth_dc_abs[x0 + k][y0 + j][i] and depth_dc_sign_flag[x0 + k][y0 + j][i] syntax elements are not present. When not present, the value of depth_dc_present_flag[x0 + k][y0 + j] is inferred to be equal to 1.

depth_dc_abs[x0 + k][y0 + j][i] and **depth_dc_sign_flag**[x0 + k][y0 + j][i] are used to derive DcOffset[x0 + k][y0 + j][i]. When not present, the values of depth_dc_abs[x0 + k][y0 + j][i] and depth_dc_sign_flag[x0 + k][y0 + j][i] are inferred to be equal to 0. The variable DcOffset[x0 + k][y0 + j][i] is derived as follows:

$$\text{DcOffset}[\,x0 + k\,][\,y0 + j\,][\,i\,] = (1 - 2 * \text{depth\_dc\_sign\_flag}[\,x0 + k\,][\,y0 + j\,][\,i\,]) * (\text{depth\_dc\_abs}[\,x0 + k\,][\,y0 + j\,][\,i\,] - \text{dcNumSeg} + 2) \quad \text{(I−51)}$$

I.7.4.9.6 Prediction unit semantics

The specifications in clause F.7.4.9.6 apply with the following addition at the end of the specification of semantics of inter_pred_idc[x0][y0]:

It is a requirement of bitstream conformance that, when DbbpFlag[x0][y0] is equal to 1, inter_pred_idc[x0][y0] shall not be equal to PRED_BI.

I.7.4.9.7 PCM sample semantics

The specifications in clause F.7.4.9.7 apply.

I.7.4.9.8 Transform tree semantics

The specifications in clause F.7.4.9.8 apply with the following additions at the end of the specification of split_transform_flag[x0][y0][trafoDepth]:

When DimFlag[x0][y0] is equal to 1 and PartMode is equal to PART_2Nx2N, the value of split_transform_flag[x0][y0][0] shall be equal to 0.

When DimFlag[x0][y0] is equal to 1 and PartMode is equal to PART_NxN, the value of split_transform_flag[x0][y0][1] shall be equal to 0.

I.7.4.9.9 Motion vector difference coding semantics

The specifications in clause F.7.4.9.9 apply.

I.7.4.9.10 Transform unit semantics

The specifications in clause F.7.4.9.10 apply.

I.7.4.9.11 Residual coding semantics

The specifications in clause F.7.4.9.11 apply.

I.7.4.9.12 Cross-component prediction semantics

The specifications in clause F.7.4.9.12 apply.

I.8 Decoding process

I.8.1 General decoding process

I.8.1.1 General

The specifications in clause F.8.1.1 apply.

I.8.1.2 Decoding process for a coded picture with nuh_layer_id greater than 0

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

1. The decoding of NAL units is specified in clause I.8.2.
2. The processes in clauses G.8.1.3, F.8.3.4 and I.8.3.1 to I.8.3.5 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, clause G.8.1.3 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 clause F.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 DisparityDerivationFlag is equal to 1 and DepthFlag is equal to 0, the derivation process for the candidate picture list for disparity vector derivation in clause I.8.3.1 is invoked at the beginning of the decoding process for each P or B slice.
   - When DisparityDerivationFlag is equal to 1, the derivation process for the default reference view order index for disparity derivation as specified in clause I.8.3.2 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 derivation process for a depth look-up table in clause I.8.3.3 is invoked at the beginning of the decoding process of the first slice segment of a coded picture.
   - At the beginning of the decoding process for each P or B slice, the derivation process for the alternative target reference index for temporal motion vector prediction in merge mode as specified in clause I.8.3.4 is invoked.
   - When IvResPredEnabledFlag is equal to 1, the derivation process for the target reference index for residual prediction as specified in clause I.8.3.5 is invoked at the beginning of the decoding process for each P or B slice.
3. The processes in clauses I.8.4, I.8.5, I.8.6, and I.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.

I.8.2 NAL unit decoding process

The specifications in clause G.8.2 apply.

I.8.3 Slice decoding process

I.8.3.1 Derivation process for the candidate picture list for disparity vector derivation

The function tempId(picA) is specified as follows:

tempId(picA) = TemporalId of picA (I−52)

The variable NumDdvCandPics is set equal to 0 and when slice_temporal_mvp_enabled_flag is equal to 1, the list DdvCandPicList is constructed as follows:

1. The variable X is set equal to (1 − collocated_from_l0_flag), the variable DdvCandPicList[0] is set equal to RefPicListX[collocated_ref_idx], and NumDdvCandPics is set equal to 1.
2. The variables NumDdvCandPics, DdvCandPicList[1], and minTempIdRefs are derived as follows:

```
minTempIdRefs = 7
for( k = 0; k  <=  ( slice_type  = =  B ? 1 : 0 ); k++ ) {
    X = k ? collocated_from_l0_flag : ( 1 − collocated_from_l0_flag )
    for( i = 0; i  <=  num_ref_idx_lX_active_minus1; i++ )
```

```
            if( ViewIdx  = =  ViewIdx( RefPicListX[ i ] )
                    &&  NumDdvCandPics  !=  2
                    &&  ( X  = =  collocated_from_l0_flag  | |  i  !=  collocated_ref_idx ) )        (I−53)
                if( RefPicListX[ i ] is an IRAP picture )
                    DdvCandPicList[ NumDdvCandPics++ ] = RefPicListX[ i ]
                else
                    minTempIdRefs = Min( minTempIdRefs, tempId( RefPicListX[ i ] ) )
    }
```

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

```
    minPocDiff = 255
    for( k = 0; k  <=  ( slice_type  = =  B ? 1 : 0 ); k++ ) {
        X = k ? collocated_from_l0_flag : ( 1 − collocated_from_l0_flag )
        for( i = 0; i  <=  num_ref_idx_lX_active_minus1; i++ )
            if( ViewIdx  = =  ViewIdx( RefPicListX[ i ] )
                    &&  ( X  = =  collocated_from_l0_flag  | |  i  !=  collocated_ref_idx )
                    &&  tempId( RefPicListX[ i ] )  = =  minTempIdRefs
                    &&  Abs( DiffPicOrderCnt( CurrPic,  RefPicListX[ i ] ) ) < minPocDiff ) {        (I−54)
                minPocDiff = Abs( DiffPicOrderCnt( CurrPic, RefPicListX[ i ] ) )
                Z = X
                candIdx = i
            }
    }
    if( minPocDiff < 255 )
        DdvCandPicList[ NumDdvCandPics++ ] = RefPicListZ[ candIdx ]
```

I.8.3.2 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 DefaultRefViewIdx is set equal to −1, the variable DispAvailFlag is set equal to 0, and the following applies for candViewIdx in the range of 0 to (ViewIdx − 1), inclusive:

- The following applies for X in the range of 0 to (slice_type = = B) ? 1 : 0, inclusive:
  - The following applies for i in the range of 0 to num_ref_idx_lX_active_minus1, inclusive:
    - When all of the following conditions are true, DefaultRefViewIdx is set equal to candViewIdx and DispAvailFlag is set equal to 1.
      - DispAvailFlag is equal to 0.
      - ViewIdx(RefPicListX[i]) is equal to candViewIdx.
      - DiffPicOrderCnt(CurrPic, RefPicListX[i]) is equal to 0.

I.8.3.3 Derivation process for a depth look-up table

For i in the range of 0 to (1 << BitDepth_Y) − 1, inclusive, the list DltIdxToVal[i] specifying the depth value corresponding to the i-th index in the depth look-up table is derived as follows:

```
    DltIdxToVal[ i ] = ( i < NumValDlt[ nuh_layer_id ] ) ? DltVal[ nuh_layer_id ][ i ] : 0        (I−55)
```

The list DltValToIdx[d] specifying the index in the depth look-up table corresponding to the depth value d is derived as follows:

```
for( d = 0; d < ( 1  <<  BitDepthY ); d++ ) {
    idxLower = 0
    foundFlag = 0
    for( iL = 1; iL < NumValDlt[ nuh_layer_id ]; iL++ )
        if( !foundFlag  &&  DltIdxToVal[ iL ] > d ) {
            idxLower = iL − 1
            foundFlag = 1
        }                                                                                        (I−56)
    idxUpper = foundFlag ? ( idxLower + 1 ) : ( NumValDlt[ nuh_layer_id ] − 1 )
    if( Abs( d − DltIdxToVal[ idxLower ] ) < Abs ( d − DltIdxToVal[ idxUpper ] ) )
        DltValToIdx[ d ] = idxLower
    else
        DltValToIdx[ d ] = idxUpper
```

}

I.8.3.4 Derivation process for the alternative target reference index for temporal motion vector prediction 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 (slice_type = = B) ? 1 : 0, inclusive:

```
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 ) | |        (I−57)
            ( !zeroIdxLtFlag && RefPicListX[ i ] is a long-term reference picture ) )
        AltRefIdxLX = i
```

I.8.3.5 Derivation process for the target reference index for residual prediction

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

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

– The variable RpRefIdxLX is set equal to −1 and the variable RpRefPicAvailFlagLX is set equal to 0.

– For i in the range of 0 to NumViews − 1, inclusive, the following applies:

  – The variable RefRpRefAvailFlagLX[ViewOIdxList[i]] is set equal to 0.

For X in the range of 0 to (slice_type = = B) ? 1 : 0, inclusive, the following applies:

– The variable minPocDiff is set equal to $2^{15} - 1$.

– For i in the range of 0 to num_ref_idx_lX_active_minus1, inclusive, the following applies:

  – The variable pocDiff is set equal to Abs(DiffPicOrderCnt(CurrPic, RefPicListX[i])).

  – When pocDiff is not equal to 0 and pocDiff is less than minPocDiff, the following applies:

minPocDiff = pocDiff (I−58)

RpRefIdxLX = i (I−59)

RpRefPicAvailFlagLX = 1 (I−60)

– When DispAvailFlag is equal to 1 and RpRefPicAvailFlagLX is equal to 1, the following applies for i in the range of 0 to NumActiveRefLayerPics − 1, inclusive:

  – Let picV the picture in the current AU with nuh_layer_id equal to RefPicLayerId[i].

  – When there is a picture picA in one of the reference picture sets RefPicSetLtCurr, RefPicSetStCurrBefore, and RefPicSetStCurrAfter of picV with DiffPicOrderCnt(picA, RefPicListX[RpRefIdxLX]) equal to 0, RefRpRefAvailFlagLX[ViewIdx(RefPicLayerId[i])] is set equal to 1.

The variable RpRefPicAvailFlag is derived as follows:

RpRefPicAvailFlag = (RpRefPicAvailFlagL0 | | RpRefPicAvailFlagL1) && DispAvailFlag (I−61)

When RpRefPicAvailFlag is equal to 1 and RefRpRefAvailFlagL0[ViewOIdxList[i]] is equal to 1 for any i in the range of 0 to NumViews − 1, inclusive, it is a requirement of bitstream conformance that PicOrderCnt(RefPicList0[RpRefIdxL0]) shall be the same for all slices of a coded picture.

When RpRefPicAvailFlag is equal to 1 and RefRpRefAvailFlagL1[ViewOIdxList[i]] is equal to 1 for any i in the range of 0 to NumViews − 1, inclusive, it is a requirement of bitstream conformance that PicOrderCnt(RefPicList1[RpRefIdxL1]) shall be the same for all slices of a coded picture.

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

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

The specifications in clause 8.4.1 apply with the following modification:

– All invocations of the process specified in clause 8.4.2 are replaced with invocations of the process specified in clause I.8.4.2.

– All invocations of the process specified in clause 8.4.4.1 are replaced with invocations of the process specified in clause I.8.4.4.1.

I.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 I.2 specifies the value for the intra prediction mode and the associated names.

Table I.2 – 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_WEDGE
36	INTRA_CONTOUR
37	INTRA_SINGLE

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

– If skip_intra_flag[xPb][yPb] is equal to 1, the following applies:
  - – If skip_intra_mode_idx[xPb][yPb] is equal to 0, IntraPredModeY[xPb][yPb] is set equal to INTRA_ANGULAR26.
  - – Otherwise, if skip_intra_mode_idx[xPb][yPb] is equal to 1, IntraPredModeY[xPb][yPb] is set equal to INTRA_ANGULAR10.
  - – Otherwise (skip_intra_mode_idx[xPb][yPb] is equal to 2 or 3), IntraPredModeY[xPb][yPb] is set equal to INTRA_SINGLE.

– Otherwise, if DimFlag[xPb][yPb] is equal to 1, the following applies:
  - – If depth_intra_mode_idx_flag[xPb][yPb] is equal to 0, IntraPredModeY[xPb][yPb] is set equal to INTRA_WEDGE.
  - – Otherwise (depth_intra_mode_idx_flag[xPb][yPb] is equal to 1), IntraPredModeY[xPb][yPb] is set equal to INTRA_CONTOUR.

– Otherwise (skip_intra_flag[xPb][yPb] and DimFlag[xPb][yPb] are both equal to 0), IntraPredModeY[xPb][yPb] is derived by 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 clause 6.4.1 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 IntraPredModeY[xNbX][yNbX] is greater 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 (I−62)

       candModeList[1] = INTRA_DC (I−63)

       candModeList[2] = INTRA_ANGULAR26 (I−64)

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

       candModeList[0] = candIntraPredModeA (I−65)

       candModeList[1] = 2 + ((candIntraPredModeA + 29) % 32) (I−66)

       candModeList[2] = 2 + ((candIntraPredModeA − 2 + 1) % 32) (I−67)

   – Otherwise (candIntraPredModeB is not equal to candIntraPredModeA), the following applies:
     – candModeList[0] and candModeList[1] are derived as follows:

       candModeList[0] = candIntraPredModeA (I−68)

       candModeList[1] = candIntraPredModeB (I−69)

     – 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]) (I−70)

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

           (candModeList[0], candModeList[2]) = Swap(candModeList[0], candModeList[2]) (I−71)

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

           (candModeList[1], candModeList[2]) = Swap(candModeList[1], candModeList[2]) (I−72)

     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.

I.8.4.3 Derivation process for chroma intra prediction mode

The specifications in clause 8.4.3 apply.

I.8.4.4 Decoding process for intra blocks

I.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 * SubWidthC, yTb0 * SubHeightC) (I−73)

The variable splitFlag is derived as follows:

– If DcOnlyFlag[xTbY][yTbY] is equal to 1, the following applies:

splitFlag = !DimFlag[xTbY][yTbY] && (log2TrafoSize > Log2MaxTrafoSize) (I−74)

– Otherwise, if skip_intra_flag[xTbY][yTbY] is equal to 1, splitFlag is set equal to 0.

– Otherwise, 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:

   – If cIdx is equal to 0 or ChromaArrayType is not equal to 2, the following applies:

     – The variable xTb1 is set equal to xTb0 + (1 << (log2TrafoSize − 1)).

     – The variable yTb1 is set equal to yTb0 + (1 << (log2TrafoSize − 1)).

   – Otherwise (ChromaArrayType is equal to 2 and cIdx is greater than 0), the following applies:

     – The variable xTb1 is set equal to xTb0 + (1 << (log2TrafoSize − 1)).

     – The variable yTb1 is set equal to yTb0 + (2 << (log2TrafoSize − 1)).

2. The general decoding process for intra blocks as specified in this clause 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 clause 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 clause 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 clause 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), for the variable blkIdx proceeding over the values 0..(cIdx > 0 && ChromaArrayType = = 2 ? 1 : 0), the following ordered steps apply:

1. The variable nTbS is set equal to 1 << log2TrafoSize.
2. The variable yTbOffset is set equal to blkIdx * nTbS.
3. The variable yTbOffsetY is set equal to yTbOffset * SubHeightC.
4. The variable residualDpcm is derived as follows:
   – If all of the following conditions are true, residualDpcm is set equal to 1.
     – implicit_rdpcm_enabled_flag is equal to 1.
     – either transform_skip_flag[xTbY][yTbY + yTbOffsetY][cIdx] is equal to 1, or cu_transquant_bypass_flag is equal to 1.
     – either predModeIntra is equal to 10, or predModeIntra is equal to 26.
   – Otherwise, residualDpcm is set equal to 0.
5. The general intra sample prediction process as specified in clause I.8.4.4.2.1 is invoked with the transform block location (xTb0, yTb0 + yTbOffset), 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.
6. The variable residualFlag is set equal to !(skip_intra_flag[xTb0][xTb0] || DcOnlyFlag[xTb0][xTb0]) and depending on the value of residualFlag, the following applies:
   – If residualFlag is equal to 1, the following applies:
     – The scaling and transformation process as specified in clause 8.6.2 is invoked with the luma location (xTbY, yTbY + yTbOffsetY), 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.
     – When residualDpcm is equal to 1, the directional residual modification process for blocks using a transform bypass as specified in clause 8.6.5 is invoked with the variable mDir set equal to predModeIntra / 26, the variable nTbS, and the (nTbS)x(nTbS) array r set equal to the array resSamples as inputs, and the output is a modified (nTbS)x(nTbS) array resSamples.
     – When cross_component_prediction_enabled_flag is equal to 1, ChromaArrayType is equal to 3, and cIdx is not equal to 0, the residual modification process for transform blocks using cross-component prediction as specified in clause 8.6.6 is invoked with the current luma transform block location (xTbY, yTbY), the variable nTbS, the variable cIdx, the (nTbS)x(nTbS) array r_Y set equal to the corresponding luma residual sample array resSamples of the current transform block, and the (nTbS)x(nTbS) array r set equal to the array resSamples as inputs, and the output is a modified (nTbS)x(nTbS) array resSamples.
   – Otherwise (residualFlag is equal to 0), for x, y = 0..nTbS − 1, resSamples[x][y] is set equal to 0.
7. The picture construction process prior to in-loop filtering for a colour component as specified in clause 8.6.7 is invoked with the transform block location (xTb0, yTb0 + yTbOffset), the variables nCurrSw and nCurrSh both set equal to nTbS, the variable cIdx, the (nTbS)x(nTbS) array predSamples, and the (nTbS)x(nTbS) array resSamples as inputs.

When trafoDepth is equal to 0, DcOnlyFlag[xTb0][yTb0] is equal to 1, and DimFlag[xTb0][yTb0] is equal to 0, the depth DC offset assignment process as specified in clause I.8.4.4.3 is invoked with the location (xTb0, yTb0), and the transform size trafoSize set equal to nTbS as inputs.

I.8.4.4.2 Intra sample prediction

I.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.

Outputs of this process are 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:

$$(xNbCmp, yNbCmp) = (xTbCmp + x, yTbCmp + y) \quad (I−75)$$

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

$$(xTbY, yTbY) = (cIdx == 0) ? (xTbCmp, yTbCmp) : (xTbCmp * SubWidthC, yTbCmp * SubHeightC) \quad (I−76)$$

$$(xNbY, yNbY) = (cIdx == 0) ? (xNbCmp, yNbCmp) : (xNbCmp * SubWidthC, yNbCmp * SubHeightC) \quad (I−77)$$

– The availability derivation process for a block in z-scan order as specified in clause 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" and skip_intra_flag[xTbCmp][yTbCmp] is not equal to 2 or 3, the reference sample substitution process for intra sample prediction in clause 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 predModeIntra is in the range of 0 to 34, inclusive, intra_smoothing_disabled_flag is equal to 0, and either cIdx is equal to 0 or ChromaArrayType is equal to 3, the filtering process of neighbouring samples specified in clause 8.4.4.2.3 is invoked with the sample array p, the transform block size nTbS and the colour component index cIdx 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 clause 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 clause 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 clause 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_WEDGE, the corresponding intra prediction mode specified in clause I.8.4.4.2.2 is invoked with the location (xTbY, yTbY), 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_CONTOUR, the corresponding intra prediction mode specified in clause I.8.4.4.2.3 is invoked with the location (xTbY, yTbY), 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_SINGLE, the corresponding intra prediction mode specified in clause I.8.4.4.2.4 is invoked with the location (xTbY, yTbY), the sample array p, and the transform block size nTbS as inputs, and the output is the predicted sample array predSamples.

I.8.4.4.2.2 Specification of intra prediction mode INTRA_WEDGE

Inputs to this process are:

– a luma location (xTb, yTb) specifying the top-left sample of the current transform block relative to the top-left luma 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 are the predicted samples predSamples[x][y], with x, y = 0..nTbS − 1.

The list WedgePatternTable and the variable NumWedgePattern are specified by the derivation process for a wedgelet partition pattern table in clause I.6.6.

The partition pattern wedgePattern[x][y] with x, y = 0..nTbS − 1 is derived as follows:

wedgePattern = WedgePatternTable[Log2(nTbS)][wedge_full_tab_idx[xTb][yTb]] (I−78)

The depth sub-block partition DC value derivation and assignment process as specified in clause I.8.4.4.2.5 is invoked with the neighbouring samples p[x][y], the partition pattern wedgePattern[x][y], the luma location (xTb, yTb), and the transform size nTbS as inputs, and the output is assigned to predSamples[x][y].

I.8.4.4.2.3 Specification of intra prediction mode INTRA_CONTOUR

Inputs to this process are:

– a luma location (xTb, yTb) specifying the top-left sample of the current transform block relative to the top-left luma 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 are the predicted samples predSamples[x][y], with x, y = 0..nTbS − 1.

The partition pattern contourPattern[x][y], with x, y = 0..nTbS − 1 is derived as follows:

– The variable refSamples is set equal to the reconstructed picture sample array S_L of the picture TexturePic.

– The variable threshVal is derived as follows:

threshVal = (refSamples[xTb][yTb] + refSamples[xTb][yTb + nTbS − 1]
+ refSamples[xTb + nTbS − 1][yTb] + refSamples[xTb + nTbS − 1][yTb + nTbS − 1]) >> 2 (I−79)

– For x = 0..nTbS − 1 and y = 0..nTbS − 1, the following applies:

contourPattern[x][y] = (refSamples[xTb + x][yTb + y] > threshVal) (I−80)

The depth sub-block partition DC value derivation and assignment process as specified in clause I.8.4.4.2.5 is invoked with the neighbouring samples p[x][y], the partition pattern contourPattern[x][y], the luma location (xTb, yTb), and the transform size nTbS as inputs, and the output is assigned to predSamples[x][y].

I.8.4.4.2.4 Specification of intra prediction mode INTRA_SINGLE

Inputs to this process are:

– a luma location (xTb, yTb) specifying the top-left sample of the current transform block relative to the top-left luma 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 are the predicted samples predSamples[x][y], with x, y = 0..nTbS − 1.

Depending on the value of skip_intra_mode_idx[xTb][yTb], the luma location (xN, yN) is derived as follows:

– If skip_intra_mode_idx[xTb][yTb] is equal to 2, (xN, yN) is set equal to (−1, nTbS >> 1).

– Otherwise (skip_intra_mode_idx[xTb][yTb] is equal to 3), (xN, yN) is set equal to (nTbS >> 1, −1).

The variable singleSampleVal is derived as follows:

– If p[xN][yN] is marked as "available for intra prediction", singleSampleVal is set equal to p[xN][yN].

– Otherwise (p[xN][yN] is marked as "not available for intra prediction"), singleSampleVal is set equal to (1 << (BitDepth_Y − 1))

The values of the prediction samples predSamples[x][y], with x, y = 0..nTbS − 1, are derived as follows:

predSamples[x][y] = singleSampleVal (I−81)

I.8.4.4.2.5 Depth sub-block partition DC 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 partition pattern partitionPattern[x][y], with x, y = 0..nTbS − 1, specifying the sub-block partitions of the current prediction block,

– a luma location (xTb, yTb) specifying the top-left sample of the current transform block relative to the top-left luma sample of the current picture,

– a variable nTbS specifying the transform block size.

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

The variables vertEdgeFlag and horEdgeFlag are derived as follows:

vertEdgeFlag = (partitionPattern[0][0] != partitionPattern[nTbS − 1][0]) (I−82)

horEdgeFlag = (partitionPattern[0][0] != partitionPattern[0][nTbS − 1]) (I−83)

The variables dcValBR and dcValLT are derived as follows:

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

  – The variable dcValBR is derived as follows:

    – If horEdgeFlag is equal to 1, the following applies:

      dcValBR = ((p[−1][nTbS − 1] + p[nTbS − 1][−1]) >> 1) (I−84)

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

      vertAbsDiff = Abs(p[−1][0] − p[−1][nTbS * 2 − 1]) (I−85)

      horAbsDiff = Abs(p[0][−1] − p[nTbS * 2 − 1][−1]) (I−86)

      dcValBR = (horAbsDiff > vertAbsDiff) ? p[nTbS * 2 − 1][−1] : p[−1][nTbS * 2 − 1] (I−87)

  – The variable dcValLT is derived as follows:

    dcValLT = (p[−1][0] + p[0][−1]) >> 1 (I−88)

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

  dcValBR = horEdgeFlag ? p[−1][nTbS − 1] : p[nTbS − 1][−1] (I−89)

  dcValLT = horEdgeFlag ? p[(nTbS − 1) >> 1][−1] : p[−1][(nTbS − 1) >> 1] (I−90)

The predicted sample values predSamples[x][y] are derived as follows:

– 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 follows:

      predDcVal = (partitionPattern[x][y] = = partitionPattern[0][0]) ? dcValLT : dcValBR (I−91)

      dcOffset = DcOffset[xTb][yTb][partitionPattern[x][y]] (I−92)

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

      predSamples[x][y] = predDcVal + dcOffset (I−93)

    – Otherwise (DltFlag[nuh_layer_id] is equal to 1), the following applies:

$$\text{predSamples[x][y] = DltIdxToVal[Clip1}_Y\text{(DltValToIdx[predDcVal] + dcOffset)]} \quad \text{(I−94)}$$

I.8.4.4.3 Depth DC offset assignment process

Inputs to this process are:

– a luma location (xTb, yTb) specifying the top-left luma sample of the current transform block relative to the top-left luma sample of the current picture,

– a variable nTbS specifying the transform block size.

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

Depending on the value of DltFlag[nuh_layer_id], the variable dcVal is derived as follows:

– If DltFlag[nuh_layer_id] is equal to 0, dcVal is set equal to DcOffset[xTb][yTb][0].

– Otherwise (DltFlag[nuh_layer_id] is equal to 1), dcVal is derived as follows:

$$\text{dcPred} = (S_L[\text{ xTb }][\text{ yTb }] + S_L[\text{ xTb }][\text{ yTb + nTbS} - 1] + S_L[\text{ xTb + nTbS} - 1][\text{ yTb }] + S_L[\text{ xTb + nTbS} - 1][\text{ yTb + nTbS} - 1] + 2) \gg 2 \quad \text{(I−95)}$$

$$\text{dcVal = DltIdxToVal[Clip1}_Y\text{(DltValToIdx[dcPred] + DcOffset[xTb][yTb][0])]} - \text{dcPred} \quad \text{(I−96)}$$

For x, y = 0..nTbS − 1, the variable S_L[x][y] is modified as follows:

$$S_L[\text{ xTb + x }][\text{ yTb + y }] = \text{Clip1}_Y(S_L[\text{ xTb + x }][\text{ yTb + y }] + \text{dcVal}) \quad \text{(I−97)}$$

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

I.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 clause 8.6.1 is invoked with the luma location (xCb, yCb) as input.

The variable $nCbS_L$ is set equal to 1 << log2CbSize. When ChromaArrayType is not equal to 0, the variable $nCbSw_C$ is set equal to (1 << log2CbSize) / SubWidthC and the variable $nCbSh_C$ is set equal to (1 << log2CbSize) / SubHeightC.

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

1. When DisparityDerivationFlag is equal to 1, the following applies:
   – If DepthFlag is equal to 0, the derivation process for a disparity vector for texture layers as specified in clause I.8.5.5 is invoked with the luma location (xCb, yCb) and the coding block size $nCbS_L$ as inputs.
   – Otherwise (DepthFlag is equal to 1), the derivation process for a disparity vector for depth layers as specified in clause I.8.5.6 is invoked with the luma location (xCb, yCb) and the coding block size $nCbS_L$ as inputs.
2. The inter prediction process as specified in clause I.8.5.2 is invoked with the luma location (xCb, yCb) and the luma coding block size log2CbSize as inputs, and the output is the array $predSamples_L$ and, when ChromaArrayType is not equal to 0, the arrays $predSamples_{Cb}$, and $predSamples_{Cr}$.
3. The decoding process for the residual signal of coding units coded in inter prediction mode specified in clause I.8.5.4 is invoked with the luma location (xCb, yCb) and the luma coding block size log2CbSize as inputs, and the outputs are the array $resSamples_L$ and, when ChromaArrayType is not equal to 0, the arrays $resSamples_{Cb}$, and $resSamples_{Cr}$.
4. The reconstructed samples of the current coding unit are derived as follows:
   – The picture construction process prior to in-loop filtering for a colour component as specified in clause 8.6.7 is invoked with the luma coding block location (xCb, yCb), the variable nCurrSw set equal to $nCbS_L$, the variable nCurrSh set equal to $nCbS_L$, the variable cIdx set equal to 0, the $(nCbS_L)$x$(nCbS_L)$ array predSamples set equal to $predSamples_L$, and the $(nCbS_L)$x$(nCbS_L)$ array resSamples set equal to

$resSamples_L$ as inputs.

- When ChromaArrayType is not equal to 0, the picture construction process prior to in-loop filtering for a colour component as specified in clause 8.6.7 is invoked with the chroma coding block location (xCb / SubWidthC, yCb / SubHeightC), the variable nCurrSw set equal to $nCbSw_C$, the variable nCurrSh set equal to $nCbSh_C$, the variable cIdx set equal to 1, the $(nCbSw_C)x(nCbSh_C)$ array predSamples set equal to $predSamples_{Cb}$, and the $(nCbSw_C)x(nCbSh_C)$ array resSamples set equal to $resSamples_{Cb}$ as inputs.
- When ChromaArrayType is not equal to 0, the picture construction process prior to in-loop filtering for a colour component as specified in clause 8.6.7 is invoked with the chroma coding block location (xCb / SubWidthC, yCb / SubHeightC), the variable nCurrSw set equal to $nCbSw_C$, the variable nCurrSh set equal to $nCbSh_C$, the variable cIdx set equal to 2, the $(nCbSw_C)x(nCbSh_C)$ array predSamples set equal to $predSamples_{Cr}$, and the $(nCbSw_C)x(nCbSh_C)$ array resSamples set equal to $resSamples_{Cr}$ as inputs.

I.8.5.2 Inter prediction process

The specifications in clause 8.5.2 apply with the following modification:

- All invocations of the process specified in clause 8.5.3 are replaced with invocations of the process specified in clause I.8.5.3.

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

I.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 height 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 $(nCbS_L)x(nCbS_L)$ array $predSamples_L$ of luma prediction samples, where nCS_L is derived as specified below,
- when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below,
- when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below.

The variable $nCbS_L$ is set equal to nCbS. When ChromaArrayType is not equal to 0, the variable $nCbSw_C$ is set equal to nCbS / SubWidthC and the variable $nCbSh_C$ is set equal to nCbS / SubHeightC.

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 clause I.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, when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0 and predFlagL1, and the flag subPbMotionFlag as outputs.
2. Depending on the value of subPbMotionFlag and DbbpFlag[xCb][yCb], the following applies:
   - If both subPbMotionFlag and DbbpFlag[xCb][yCb] are equal to 0, the decoding process for inter sample prediction as specified in clause I.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, when ChromaArrayType is not equal to 0, 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 $(nCbS_L)x(nCbS_L)$ array $predSamples_L$ of prediction luma samples and, when ChromaArrayType is not equal to 0, two $(nCbSw_C)x(nCbSh_C)$ arrays $predSamples_{Cr}$ and $predSamples_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr, as outputs.

– Otherwise, if subPbMotionFlag is equal to 1, the decoding process for inter sample prediction for rectangular sub-block partitions as specified in clause I.8.5.3.4 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, and the luma prediction block height nPbH as inputs, and the inter prediction samples that are an $(nCbS_L)x(nCbS_L)$ array $predSamples_L$ of prediction luma samples and, when ChromaArrayType is not equal to 0, two $(nCbSw_C)x(nCbSh_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$ of chroma prediction samples, one for each of the chroma components Cb and Cr, as outputs.

– Otherwise (DbbpFlag[xCb][yCb] is equal to 1), the decoding process for inter sample prediction for depth predicted sub-block partitions as specified in clause I.8.5.3.5 is invoked with the luma coding block location (xCb, yCb), the luma coding block size nCbS, the luma motion vectors mvL0 and mvL1, when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1, the reference indices refIdxL0 and refIdxL1, the prediction list utilization flags predFlagL0 and predFlagL1, and the variable partIdx as inputs, and the inter prediction samples that are an $(nCbS_L)x(nCbS_L)$ array $predSamples_L$ of prediction luma samples and, when ChromaArrayType is not equal to 0, two $(nCbSw_C)x(nCbSh_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$ of chroma prediction 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 ? SubPbArrayMvL0[xCb + x][yCb + y] : mvL0 (I−98)

MvL1[xCb + x][yCb + y] = subPbMotionFlag ? SubPbArrayMvL1[xCb + x][yCb + y] : mvL1 (I−99)

RefIdxL0[xCb + x][yCb + y] = subPbMotionFlag ?
SubPbArrayRefIdxL0[xCb + x][yCb + y] : refIdxL0 (I−100)

RefIdxL1[xCb + x][yCb + y] = subPbMotionFlag ?
SubPbArrayRefIdxL1[xCb + x][yCb + y] : refIdxL1 (I−101)

PredFlagL0[xCb + x][yCb + y] = subPbMotionFlag ?
SubPbArrayPredFlagL0[xCb + x][yCb + y] : predFlagL0 (I−102)

PredFlagL1[xCb + x][yCb + y] = subPbMotionFlag ?
SubPbArrayPredFlagL1[xCb + x][yCb + y] : predFlagL1 (I−103)

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

I.8.5.3.2.1 General

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,

– when ChromaArrayType is not equal to 0, 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 specifying whether the current PU is coded using sub-block partitions.

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 variables currPic and LX 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 vspMcFlag, ivMcFlag, 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 clause I.8.5.3.2.7 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, the prediction list utilization flags predFlagL0 and predFlagL1, the flag ivMcFlag, the flag vspMcFlag, 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,

       refIdxLX = ref_idx_lX[xPb][yPb] (I−104)

       predFlagLX = 1 (I−105)

     – Otherwise, the variables refIdxLX and predFlagLX are specified by:

       refIdxLX = −1 (I−106)

       predFlagLX = 0 (I−107)

  2. The variable mvdLX is derived as follows:

     mvdLX[0] = MvdLX[xPb][yPb][0] (I−108)

     mvdLX[1] = MvdLX[xPb][yPb][1] (I−109)

  3. When predFlagLX is equal to 1, the derivation process for luma motion vector prediction in clause I.8.5.3.2.3 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:

$$uLX[0] = (mvpLX[0] + mvdLX[0] + 2^{16}) \% 2^{16} \quad (I−110)$$

$$mvLX[0] = (uLX[0] >= 2^{15}) ? (uLX[0] − 2^{16}) : uLX[0] \quad (I−111)$$

$$uLX[1] = (mvpLX[1] + mvdLX[1] + 2^{16}) \% 2^{16} \quad (I−112)$$

$$mvLX[1] = (uLX[1] >= 2^{15}) ? (uLX[1] − 2^{16}) : uLX[1] \quad (I−113)$$

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 clause 8.5.3.2.10 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):

IvMcFlag[x][y] = ivMcFlag (I−114)
VspMcFlag[x][y] = vspMcFlag (I−115)

I.8.5.3.2.2 Derivation process for an initial merge candidate list

The specifications in clause 8.5.3.2.2 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 with "When slice_type is equal to B and numCurrMergeCand is less than 5, the derivation process for combined bi-predictive merging candidates".

– "derivation process for merging candidates from neighbouring prediction unit partitions in clause 8.5.3.2.3" is replaced with "derivation process for merging candidates from neighbouring prediction unit partitions in clause I.8.5.3.2.3".

– "temporal luma motion vector prediction in clause 8.5.3.2.8 is invoked" is replaced with "temporal luma motion vector prediction in clause I.8.5.3.2.5 is invoked".

– The outputs of the process are replaced with:

  – the 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,

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

  – the modified variable partIdx specifying the modified index of the current prediction unit within the current coding unit,

  – the 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,

  – the variables nOrigPbW and nOrigPbH specifying the original width and the height of the luma prediction block,

  – the merging candidate list, mergeCandList,

  – the luma motion vectors mvL0N and mvL1N, with N being replaced by all elements of mergeCandList,

  – the reference indices refIdxL0N and refIdxL1N, with N being replaced by all elements of mergeCandList,

  – the prediction list utilization flags predFlagL0N and predFlagL1N, with N being replaced by all elements of mergeCandList.

I.8.5.3.2.3 Derivation process for spatial merging candidates

The specifications in clause 8.5.3.2.3 apply with the following modifications and additions:

– The function differentMotionLoc(xN, yN, xM ,yM) is specified as follows:

  – If one or more of the following conditions are true, for X in the range of 0 to 1, inclusive, differentMotionLoc(xN, yN ,xM ,yM) is set equal to 1:

    – PredFlagLX[xN][yN] is not equal to PredFlagLX[xM][yM].

    – MvLX[xN][yN] is not equal to MvLX[xM][yM].

    – RefIdxLX[xN][yN] is not equal to RefIdxLX[xM][yM].

  – Otherwise, differentMotionLoc(xN, yN, xM ,yM) is set equal to 0.

– "the prediction units covering the luma locations ($xNbA_1$, $yNbA_1$) and ($xNbB_1$, $yNbB_1$) have the same motion vectors and the same reference indices" is replaced with "differentMotionLoc($xNbA_1$, $yNbA_1$, $xNbB_1$, $yNbB_1$) is equal to 0".

– "the prediction units covering the luma locations ($xNbB_1$, $yNbB_1$) and ($xNbB_0$, $yNbB_0$) have the same motion vectors and the same reference indices" is replaced with "differentMotionLoc($xNbB_1$, $yNbB_1$, $xNbB_0$, $yNbB_0$) is equal to 0".

– "the prediction units covering the luma locations ($xNbA_1$, $yNbA_1$) and ($xNbA_0$, $yNbA_0$) have the same motion vectors and the same reference indices" is replaced with "differentMotionLoc($xNbA_1$, $yNbA_1$, $xNbA_0$, $yNbA_0$) is equal to 0".

– "prediction units covering the luma locations ($xNbA_1$, $yNbA_1$) and ($xNbB_2$, $yNbB_2$) have the same motion vectors and the same reference indices" is replaced with "differentMotionLoc($xNbA_1$, $yNbA_1$, $xNbB_2$, $yNbB_2$) is equal to 0".

– "the prediction units covering the luma locations ($xNbB_1$, $yNbB_1$) and ($xNbB_2$, $yNbB_2$) have the same motion vectors and the same reference indices" is replaced with "differentMotionLoc($xNbB_1$, $yNbB_1$, $xNbB_2$, $yNbB_2$) is equal to 0".

I.8.5.3.2.4 Derivation process for luma motion vector prediction

The specifications in clause 8.5.3.2.6 apply with the following modifications:

– "temporal luma motion vector prediction in clause 8.5.3.2.8 is invoked" is replaced by "temporal luma motion vector prediction in clause I.8.5.3.2.5 is invoked".

I.8.5.3.2.5 Derivation process for temporal luma motion vector prediction

Inputs to this process are:

– 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,

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

– 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 currPb specifies the current luma prediction block at luma location (xPb, yPb).

The variables mvLXCol and availableFlagLXCol are derived as follows:

– If slice_temporal_mvp_enabled_flag is equal to 0, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

– Otherwise, the following ordered steps apply:

1. The bottom right collocated motion vector is derived as follows:

xColBr = xPb + nPbW (I−116)

yColBr = yPb + nPbH (I−117)

– If yPb >> CtbLog2SizeY is equal to yColBr >> CtbLog2SizeY, yColBr is less than pic_height_in_luma_samples and xColBr is less than pic_width_in_luma_samples, the following applies:

  – The luma location (xColPb, yColPb) is set equal to ((xColBr >> 4) << 4, (yColBr >> 4) << 4).

  – The variable colPb specifies the luma prediction block covering the modified location given by (xColPb, yColPb) inside the collocated picture specified by ColPic.

  – Depending on merge_flag[xPb][yPb], the following applies:

    – If merge_flag[xPb][yPb] is equal to 0, the derivation process for collocated motion vectors as specified in clause 8.5.3.2.9 is invoked with currPb, colPb, (xColPb, yColPb) and refIdxLX as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.

    – Otherwise (merge_flag[xPb][yPb] is equal to 1), the derivation process for collocated motion vectors for merge mode as specified in clause I.8.5.3.2.6 is invoked with currPb, colPb, (xColPb, yColPb) and refIdxLX as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.

– Otherwise, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.

2. When availableFlagLXCol is equal to 0, the central collocated motion vector is derived as follows:

xColCtr = xPb + (nPbW >> 1) (I−118)

yColCtr = yPb + (nPbH >> 1) (I−119)

– The luma location (xColPb, yColPb) is set equal to ((xColCtr >> 4) << 4, (yColCtr >> 4) << 4).

– The variable colPb specifies the luma prediction block covering the modified location given by (xColPb, yColPb) inside the collocated picture specified by ColPic.

– Depending on merge_flag[xPb][yPb], the following applies:

  – If merge_flag[xPb][yPb] is equal to 0, the derivation process for collocated motion vectors as specified in clause 8.5.3.2.9 is invoked with currPb, colPb, (xColPb, yColPb) and refIdxLX as inputs, and the output is assigned to mvLXCol and availableFlagLXCol..

  – Otherwise (merge_flag[xPb][yPb] is equal to 1), the derivation process for collocated motion vectors for merge mode as specified in clause I.8.5.3.2.6 is invoked with currPb, colPb, (xColPb, yColPb) and refIdxLX as inputs, and the output is assigned to mvLXCol and availableFlagLXCol.

I.8.5.3.2.6 Derivation process for collocated motion vectors for merge mode

Inputs to this process are:

– a variable currPb specifying the current prediction block,

– a variable colPb specifying the collocated prediction block inside the collocated picture specified by ColPic,

– a luma location (xColPb, yColPb) specifying a sample inside 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 arrays predFlagL0Col[x][y], mvL0Col[x][y], and refIdxL0Col[x][y] are set equal to PredFlagL0[x][y], MvL0[x][y], and RefIdxL0[x][y], respectively, of the collocated picture specified by ColPic, and the arrays predFlagL1Col[x][y], mvL1Col[x][y], and refIdxL1Col[x][y] are set equal to PredFlagL1[x][y], MvL1[x][y], and RefIdxL1[x][y], respectively, of the collocated picture specified by ColPic.

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 following applies:

  – 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 NoBackwardPredFlag is equal to 1, 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.

  – The motion vector mvLXCol and availableFlagLXCol are derived as follows:

    – The variables curLtFlag and colLtFlag are derived as follows:

      curLtFlag = LongTermRefPic(CurrPic, currPb, refIdxLX, LX) (I−120)

      colLtFlag = LongTermRefPic(ColPic, colPb, refIdxCol, listCol) (I−121)

    – When curLtFlag is not equal to colLtFlag and AltRefIdxLX is not equal to −1, the variables AltRefFlagLX, refIdxLX, and curLtFlag are modified as follows:

      AltRefFlagLX = 1 (I−122)

refIdxLX = AltRefIdxLX (I−123)

curLtFlag = LongTermRefPic(CurrPic, currPb, refIdxLX, LX) (I−124)

– The motion vector mvLXCol is modified as follows:

  – If curLtFlag is not equal to colLtFlag, 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 colPb in the collocated picture specified by ColPic, and the following applies:

    – The variables colDiff and currDiff are set equal to 0 and modified as follows:

      – If curLtFlag is equal to 0, the following applies:

        colDiff = DiffPicOrderCnt(ColPic, refPicListCol[refIdxCol]) (I−125)

        currDiff = DiffPicOrderCnt(CurrPic, RefPicListX[refIdxLX]) (I−126)

      – Otherwise (curLtFlag is equal to 1), when IvMvScalEnabledFlag is equal to 1, the following applies:

        colDiff = ViewIdVal(ColPic) − ViewIdVal(refPicListCol[refIdxCol]) (I−127)

        currDiff =ViewIdVal(CurrPic) − ViewIdVal(RefPicListX[refIdxLX]) (I−128)

    – Depending on the values of colDiff and currDiff, the following applies:

      – If colDiff is equal to currDiff, or colDiff is equal to 0, or currDiff is equal to 0, mvLXCol is derived as follows:

        mvLXCol = mvCol (I−129)

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

        tx = (16384 + (Abs(td) >> 1)) / td (I−130)

        distScaleFactor = Clip3(−4096, 4095, (tb * tx + 32) >> 6) (I−131)

        mvLXCol = Clip3(−32768, 32767, Sign(distScaleFactor * mvCol) *
            ((Abs(distScaleFactor * mvCol) + 127) >> 8)) (I−132)

      where td and tb are derived as follows:

        td = Clip3(−128, 127, colDiff) (I−133)

        tb = Clip3(−128, 127, currDiff) (I−134)

I.8.5.3.2.7 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) specifies 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 flag ivMcFlag specifying whether the current PU is coded using an inter-view predicted merge candidate,

– the flag vspMcFlag specifying whether the current PU is coded using the view synthesis prediction merge candidate,

– the flag subPbMotionFlag specifying whether the current PU is coded using sub-block partitions.

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

– If one or more of the following conditions are true, for X in the range of 0 to 1, inclusive, differentMotion(N, M) is set equal to 1:

  – predFlagLXN is not equal to predFlagLXM.

  – mvLXN is not equal to mvLXM.

  – refIdxLXN is not equal to refIdxLXM.

– Otherwise, differentMotion(N, M) is set equal to 0.

The variables AltRefFlagL0 and AltRefFlagL1 are set 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 an initial merge candidate list as specified in clause I.8.5.3.2.2 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 outputs 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 merging candidate list initMergeCandList, 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 initMergeCandList.

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 initMergeCandList, availableFlagN is set equal to 1.

   – Otherwise (N is not an element in initMergeCandList), availableFlagN is set equal to 0.

3. Depending on the values of IvDiMcEnabledFlag, DispAvailFlag, and IlluCompFlag[xCb][yCb], the following applies:

   – If IvDiMcEnabledFlag is equal to 0, DispAvailFlag is equal to 0, or IlluCompFlag[xCb][yCb] is equal to 1, the flags availableFlagIV and availableFlagIVShift are set equal to 0.

   – Otherwise (IvDiMcEnabledFlag is equal to 1, DispAvailFlag is equal to 1, and IlluCompFlag[xCb][yCb] is equal to 0), the derivation process for inter-view predicted merging candidates as specified in clause I.8.5.3.2.8 is invoked with the luma location (xPb, yPb), and the variables nPbW and nPbH as inputs, and the availability flags availableFlagIV and availableFlagIVShift, the prediction list utilization flags predFlagLXIV and predFlagLXIVShift, the reference indices refIdxLXIV and refIdxLXIVShift, and the motion vectors mvLXIV and mvLXIVShift (with X in the range of 0 to 1, inclusive) as outputs.

4. Depending on the value of TexMcEnabledFlag, the texture merging candidate is derived as follows:

   – If TexMcEnabledFlag is equal to 0, the variable availableFlagT is set equal to 0.

   – Otherwise (TexMcEnabledFlag is equal to 1), the following applies:

     – The derivation process for sub-block partition motion vectors for an inter-layer predicted merging candidate as specified in clause I.8.5.3.2.9 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, and the merging candidate indicator N equal to T as inputs, and the prediction utilization flag predFlagLXT, the reference index refIdxLXT, and the motion vector mvLXT (with X in the range of 0 to 1, inclusive) as outputs.

     – The flag availableFlagT is set equal to (predFlagL0T || predFlagL1T).

5. Depending on the values of IvDiMcEnabledFlag, DispAvailFlag, and DepthFlag, the following applies:

   – If IvDiMcEnabledFlag is equal to 0, DispAvailFlag is equal to 0, or DepthFlag is equal to 1, the flags availableFlagDI and availableFlagDIShift are set equal to 0.

   – Otherwise (IvDiMcEnabledFlag is equal to 1, DispAvailFlag is equal to 1, and DepthFlag is equal to 0), the derivation process for disparity information merging candidates as specified in clause I.8.5.3.2.12 is invoked with the luma location (xPb, yPb), and the variables nPbW and nPbH as inputs, and the availability flags availableFlagDI and availableFlagDIShift, the prediction list utilization flags predFlagLXDI and predFlagLXDIShift, the reference indices refIdxLXDI and refIdxLXDIShift, and the motion vectors mvLXDI and mvLXDIShift (with X in the range of 0 to 1, inclusive) as outputs.

6. Depending on the values of VspMcEnabledFlag, DispAvailFlag, IlluCompFlag[xCb][yCb], iv_res_pred_weight_idx[xCb][yCb], and DbbpFlag[xCb][yCb], the following applies:

   – If one or more of the following conditions are true, the flag availableFlagVSP is set equal to 0:

     – VspMcEnabledFlag is equal to 0.

     – DispAvailFlag is equal to 0.

     – IlluCompFlag[xCb][yCb] is equal to 1.

     – iv_res_pred_weight_idx[xCb][yCb] is not equal to 0.

     – DbbpFlag[xCb][yCb] is equal to 1.

   – Otherwise, the derivation process for a view synthesis prediction merging candidate as specified in clause I.8.5.3.2.13 is invoked with the luma locations (xPb, yPb) and the variables nPbW and nPbH as inputs, and the availability flag availableFlagVSP, the prediction list utilization flag predFlagLXVSP, the reference index refIdxLXVSP, and the motion vector mvLXVSP (with X in the range of 0 to 1, inclusive) as outputs.

7. The merging candidate list extMergeCandList is constructed as follows:

```
i = 0
if( availableFlagT )
    extMergeCandList[ i++ ] = T
if( availableFlagIV  &&  ( !availableFlagT  ||  differentMotion( T, IV ) ) )
    extMergeCandList[ i++ ] = IV
N = DepthFlag ? T : IV
if( availableFlagA1  &&  ( !availableFlagN  ||  differentMotion( N, A1 ) ) )
    extMergeCandList[ i++ ] = A1
if( availableFlagB1  &&  ( !availableFlagN  ||  differentMotion( N, B1 ) ) )
    extMergeCandList[ i++ ] = B1
if( availableFlagVSP  &&  ( !availableFlagA1  ||  !VspMcFlag[ xPb − 1 ][ yPb + nPbH − 1 ] )  &&
    i < MaxNumMergeCand )
    extMergeCandList[ i++ ] = VSP
if( availableFlagB0 )
    extMergeCandList[ i++ ] = B0                                                    (I−135)
if( availableFlagDI  &&  ( !availableFlagA1  ||  differentMotion( A1, DI ) )  &&
    ( !availableFlagB1  ||  differentMotion( B1, DI ) )  &&  ( i < MaxNumMergeCand ) )
    extMergeCandList[ i++ ] = DI
if( availableFlagA0  &&  i < MaxNumMergeCand )
    extMergeCandList[ i++ ] = A0
if( availableFlagB2  &&  i < MaxNumMergeCand )
    extMergeCandList[ i++ ] = B2
if( availableFlagIVShift  &&  i < MaxNumMergeCand  &&
                ( !availableFlagIV  ||  differentMotion( IV, IVShift ) ) )
    extMergeCandList[ i++ ] = IVShift
if( availableFlagDIShift  &&  !availableFlagIVShift  &&  i < MaxNumMergeCand )
    extMergeCandList[ i++ ] = DIShift
j = 0
while( i < MaxNumMergeCand ) {                                                      (I−136)
    N = initMergeCandList[ j++ ]
    if( N  !=  A1  &&  N  !=  B1  &&  N  !=  B0  &&  N  !=  A0  &&  N  !=  B2 )
        extMergeCandList[ i++ ] = N
}
```

8. The variable N is derived as follows:

   – If (nOrigPbW + nOrigPbH) is equal to 12, the following applies:

     N = initMergeCandList[merge_idx[xOrigP][yOrigP]]   (I−137)

   – Otherwise, ((nOrigPbW + nOrigPbH) is not equal to 12), the following applies:

N = extMergeCandList[merge_idx[xOrigP][yOrigP]] (I−138)

9. When N is equal to Col, the following applies for X in the range of 0 to 1, inclusive:

   – When AltRefFlagLX is equal to 1, refIdxLXCol is set equal to AltRefIdxLX.

10. When availableFlagVSP is equal to 1, N is equal to A_1, and VspMcFlag[xPb − 1][yPb + nPbH − 1] is equal to 1, N is set equal to VSP.

11. The variable subPbMotionFlag is derived as follows:

    subPbMotionFlag = (!DepthFlag && !DbbpFlag[xCb][yCb] && N = = IV)
    || N = = VSP || N = = T (I−139)

12. Depending on the value of subPbMotionFlag, the following applies:

   – If subPbMotionFlag is equal to 0, the following applies, for X in the range of 0 to 1, inclusive:

     mvLX = mvLXN (I−140)

     refIdxLX = refIdxLXN (I−141)

     predFlagLX = predFlagLXN (I−142)

   – Otherwise (subPbMotionFlag is equal to 1), SubPbArrayPartSize is set equal to SubPbArrayPartSizeN and the following applies for X in the range of 0 to 1, inclusive:

     mvLX = (0, 0) (I−143)

     SubPbArrayMvLX = SubPbArrayMvLXN (I−144)

     SubPbArrayMvCLX = SubPbArrayMvCLXN (I−145)

     refIdxLX = −1 (I−146)

     SubPbArrayRefIdxLX = SubPbArrayRefIdxLXN (I−147)

     predFlagLX = 0 (I−148)

     SubPbArrayPredFlagLX = SubPbArrayPredFlagLXN (I−149)

   NOTE – When subPbMotionFlag is equal to 1, luma motion vectors, chroma motion vectors, reference indices, and prediction utilization flags are given in sub-block partition granularity in the arrays SubPbArrayPredFlagLX, SubPbArrayMvLX, SubPbArrayMvCLX, SubPbArrayRefIdxLX (with X in the range of 0 to 1, inclusive). Moreover, the width and height of the sub-block partitions is stored in the array SubPbArrayPartSize.

13. When all of the following conditions are true, refIdxL1 is set equal to −1 and predFlagL1 is set equal to 0:

   – predFlagL0 and predFlagL1 are equal to 1.

   – (nOrigPbW + nOrigPbH) is equal to 12 or DbbpFlag[xCb][yCb] is equal to 1.

14. The flag ivMcFlag is set equal to (N = = IV || N = = IVShift).

15. The flag vspMcFlag is set equal to (N = = VSP).

I.8.5.3.2.8 Derivation process for inter-view predicted merging candidates

Inputs to this process are:

– 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,

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

Outputs of this process are (with X in the range of 0 to 1, inclusive):

– the availability flags availableFlagIV and availableFlagIVShift specifying whether the inter-view predicted and shifted inter-view predicted merging candidates are available,

– the prediction list utilization flags predFlagLXIV and predFlagLXIVShift,

– the reference indices refIdxLXIV and refIdxLXIVShift,

– the motion vectors mvLXIV and mvLXIVShift.

The availability flags availableFlagIV and availableFlagIVShift are set equal to 0, and for X in the range of 0 to 1, inclusive, the following applies:

– The variables predFlagLXIV and predFlagLXIVShift are set equal to 0, the variables refIdxLXIV and refIdxLXIVShift are set equal to −1, and the motion vectors mvLXIV and mvLXIVShift are set equal to (0, 0).

The inter-view predicted merging candidate is derived by the following ordered steps:

1. Depending on the values of DbbpFlag[xPb][yPb] and DepthFlag, the following applies:
   - If DbbpFlag[xPb][yPb] is equal to 0 and DepthFlag is equal to 0, the derivation process for sub-block partition motion vectors for an inter-layer predicted merging candidate as specified in clause I.8.5.3.2.9 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, and the merging candidate indicator N equal to IV as inputs, and the outputs are the flag predFlagLXIV, the reference index refIdxLXIV, and the motion vector mvLXIV (with X in the range of 0 to 1, inclusive).
   - Otherwise (DbbpFlag[xPb][yPb] is not equal to 0 or DepthFlag is equal to 1), the derivation process for motion vectors for an inter-view predicted merging candidate as specified in clause I.8.5.3.2.10 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, and the disparity vector offset (xOff, yOff) equal to (0, 0) as inputs, and the outputs are the flag predFlagLXIV, the reference index refIdxLXIV, and the motion vector mvLXIV (with X in the range of 0 to 1, inclusive).
2. The availability flag availableFlagIV is set equal to (predFlagL0IV || predFlagL1IV).

When DepthFlag is equal to 0, the shifted inter-view predicted merging candidate is derived by the following ordered steps:

1. The derivation process for motion vectors for an inter-view predicted merging candidate as specified in clause I.8.5.3.2.10 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, and the disparity vector offset (xOff, yOff) equal to (nPbW * 2, nPbH * 2) as inputs, and the outputs are the flag predFlagLXIVShift, the reference index refIdxLXIVShift, and the motion vector mvLXIVShift (with X in the range of 0 to 1, inclusive).
2. The availability flag availableFlagIVShift is set equal to (predFlagL0IVShift || predFlagL1IVShift).

I.8.5.3.2.9 Derivation process for sub-block partition motion vectors for an inter-layer predicted merging candidate

Inputs to this process are:

– a luma location (xPb, yPb) of the top-left sample of the current prediction luma block relative to the top-left luma sample of the current picture,

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

– a merging candidate indicator N.

Outputs of this process are (with X in the range of 0 to 1, inclusive):

– the prediction utilization flag predFlagLXN,

– the reference index refIdxLXN,

– the motion vector mvLXN.

For X in the range of 0 to 1, inclusive, the variable predFlagLXN is set equal to 0, the variable refIdxLXN is set equal to −1, the motion vector mvLXN is set equal to (0, 0).

The variables nSbW and nSbH specifying the width and height of the sub-block partitions and the luma location (xSbDef, ySbDef) of the default sub-block partition are derived as follows:

subPbTmcSize = 1 << (log2_texmc_sub_pb_size_minus3[DepthFlag] + 3) (I−150)

subPbIvMcSize = 1 << (log2_ivmc_sub_pb_size_minus3[DepthFlag] + 3) (I−151)

minSize = (N = = T) ? subPbTmcSize : subPbIvMcSize (I−152)

nSbW = (nPbW % minSize != 0 || nPbH % minSize != 0) ? nPbW : minSize (I−153)

nSbH = (nPbW % minSize != 0 || nPbH % minSize != 0) ? nPbH : minSize (I−154)

(xSbDef, ySbDef) = (xPb + ((nPbW / nSbW) / 2) * nSbW, yPb + ((nPbH / nSbH) / 2) * nSbH) (I−155)

Depending on the value of N, the variables predFlagLXN, refIdxLXN, and mvLXN are derived as follows:

– If N is equal to IV, the derivation process for motion vectors for an inter-view predicted merging candidate as specified in clause I.8.5.3.2.10 is invoked with the luma location (xSbDef, ySbDef), the variables nSbW and nSbH, and the disparity vector offset (xOff, yOff) equal to (0, 0) as inputs, and the outputs are the flag predFlagLXN, the reference index refIdxLXN, and the motion vector mvLXN (with X in the range of 0 to 1, inclusive).

– Otherwise (N is equal to T), the derivation process for motion vectors for the texture merge candidate as specified in clause I.8.5.3.2.11 is invoked with the luma location (xSbDef, ySbDef), and the variables nSbW and nSbH as inputs, and the outputs are the flags predFlagLXN, the reference index refIdxLXN, and the motion vector mvLXN (with X in the range of 0 to 1, inclusive).

When predFlagL0N or predFlagL1N is equal to 1, the following applies:

– 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:

    – The luma location (xSb, ySb) of the current sub-block partition is derived as follows:

      (xSb, ySb) = (xPb + xBlk * nSbW, yPb + yBlk * nSbH) (I−156)

    – Depending on the value of N, the following applies:

      – If N is equal to IV, the derivation process for motion vectors for an inter-view predicted merging candidate as specified in clause I.8.5.3.2.10 is invoked with the luma location (xSb, ySb), the variables nSbW and nSbH, and the disparity vector offset (xOff, yOff) equal to (0, 0) as inputs, and the outputs are the flag spPredFlagLX[xBlk][yBlk], the reference index spRefIdxLX[xBlk][yBlk], and the motion vector spMvLX[xBlk][yBlk] (with X in the range of 0 to 1, inclusive).

      – Otherwise (N is equal to T), the derivation process for motion vectors for the texture merge candidate as specified in clause I.8.5.3.2.11 is invoked with the luma location (xSb, ySb), and the variables nSbW and nSbH as inputs, and the outputs are the flags spPredFlagLX[xBlk][yBlk], the reference index spRefIdxLX[xBlk][yBlk], and the motion vector spMvLX[xBlk][yBlk] (with X in the range of 0 to 1, inclusive).

    – When spPredFlagL0[xBlk][yBlk] and spPredFlagL1[xBlk][yBlk] are both equal to 0, the following applies for X in the range of 0 to 1, inclusive:

      spPredFlagLX[xBlk][yBlk] = predFlagLXN (I−157)

      spRefIdxLX[xBlk][yBlk] = refIdxLXN (I−158)

      spMvLX[xBlk][yBlk] = mvLXN (I−159)

– 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:

  – The variable SubPbArrayPartSizeN is derived as follows:

    SubPbArrayPartSizeN[xPb + x][yPb + y] = (nSbW, nSbH) (I−160)

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

    – The variables SubPbArrayPredFlagLX, SubPbArrayMvLX, and SubPbArrayRefIdxLX are derived as follows:

      SubPbArrayPredFlagLXN[xPb + x][yPb + y] = spPredFlagLX[x / nSbW][y / nSbH] (I−161)

      SubPbArrayRefIdxLXN[xPb + x][yPb + y] = spRefIdxLX[x / nSbW][y / nSbH] (I−162)

      SubPbArrayMvLXN[xPb + x][yPb + y] = spMvLX[x / nSbW][y / nSbH] (I−163)

    – The derivation process for chroma motion vectors as specified in clause 8.5.3.2.10 is invoked with SubPbArrayMvLXN[xPb + x][yPb + y] as input, and the output is SubPbArrayMvCLXN[xPb + x][yPb + y].

I.8.5.3.2.10 Derivation process for motion vectors for an inter-view predicted merging candidate

Inputs to this process are:

– a luma location (xBlk, yBlk) of the top-left sample of the current luma prediction block or sub-block partition relative to the top-left luma sample of the current picture,

– two variables nBlkW and nBlkH specifying the width and the height of the current luma prediction block or sub-block partition,

– a disparity vector offset (xOff, yOff).

Outputs of this process are (with X in the range of 0 to 1, inclusive):

– the prediction utilization flag predFlagLXInterView,

– the reference index refIdxLXInterView,

– the motion vector mvLXInterView.

For X in the range of 0 to 1, inclusive, the flag predFlagLXInterView is set equal to 0, the variable refIdxLXInterView is set equal to −1, and the motion vector mvLXInterView is set equal to (0, 0).

The luma location (xRef, yRef) is derived as follows:

dispVec[0] = DispRefVec[xBlk][yBlk][0] + xOff (I−164)

dispVec[1] = DispRefVec[xBlk][yBlk][1] + yOff (I−165)

xRefFull = xBlk + (nBlkW >> 1) + ((dispVec[0] + 2) >> 2) (I−166)

yRefFull = yBlk + (nBlkH >> 1) + ((dispVec[1] + 2) >> 2) (I−167)

xRef = Clip3(0, pic_width_in_luma_samples − 1, (xRefFull >> 3) << 3) (I−168)

yRef = Clip3(0, pic_height_in_luma_samples − 1, (yRefFull >> 3) << 3) (I−169)

Let ivRefPic the picture with nuh_layer_id equal to ViewCompLayerId[RefViewIdx[xBlk][yBlk]][DepthFlag] in the current access unit and let ivRefPb be the luma prediction block covering the luma location (xRef, yRef) in the picture ivRefPic.

The luma location (xIvRefPb, yIvRefPb) is set equal to the location of the top-left sample of ivRefPb relative to the top-left luma sample of the picture ivRefPic.

When ivRefPb is not coded in an intra prediction mode, the following applies for X in the range of 0 to (slice_type = = B) ? 1 : 0, inclusive:

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

  – Y is set equal to (k = = 0) ? X : (1 − X).

  – The variables predFlagLYIvRef[x][y], mvLYIvRef[x][y], and refIdxLYIvRef[x][y] are set equal to PredFlagLY[x][y], MvLY[x][y], and RefIdxLY[x][y], respectively, of picture ivRefPic.

  – The variable refPicListYIvRef is set equal to RefPicListY of the slice containing ivRefPb in picture ivRefPic.

  – When predFlagLYIvRef[xIvRefPb][yIvRefPb] is equal to 1, the following applies for i in the range of 0 to num_ref_idx_lX_active_minus1, inclusive:

    – When PicOrderCnt(refPicListYIvRef[refIdxLYIvRef[xIvRefPb][yIvRefPb]]) is equal to PicOrderCnt(RefPicListX[i]) and predFlagLXInterView is equal to 0, the following applies:

      predFlagLXInterView = 1 (I−170)

      refIdxLXInterView = i (I−171)

      mvLXInterView = mvLYIvRef[xIvRefPb][yIvRefPb] (I−172)

I.8.5.3.2.11 Derivation process for motion vectors for the texture merge candidate

Inputs to this process are:

– a luma location (xSb, ySb) of the top-left sample of the current luma sub-block partition relative to the top-left luma sample of the current picture,

– two variables nSbW and nSbH specifying the width and the height of the current sub-block partition,

Outputs of this process are (with X in the range of 0 to 1, inclusive):

– the prediction utilization flag predFlagLXT,

– the reference index refIdxLXT,

– the motion vector mvLXT.

For X in the range of 0 to 1, inclusive, the variable predFlagLXT is set equal to 0, the variable refIdxLXT is set equal to −1, and the motion vector mvLX0T is set equal to (0, 0).

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

$$xRefFull = xSb + (nSbW >> 1) \quad (I-173)$$

$$yRefFull = ySb + (nSbH >> 1) \quad (I-174)$$

$$xRef = (xRefFull >> 3) << 3 \quad (I-175)$$

$$yRef = (yRefFull >> 3) << 3 \quad (I-176)$$

Let textPb be the prediction block covering the luma sample location (xRef, yRef) in the picture TexturePic.

For X in the range of 0 to (slice_type = = B) ? 1 : 0, inclusive, the following applies:

– The arrays textPredFlagLX[x][y], textRefIdxLX[x][y], and textMvLX[x][y] are set equal to PredFlagLX[x][y], RefIdxLX[x][y], and MvLX[x][y], respectively, of the picture TexturePic.

– The list textRefPicListX is set equal to RefPicListX of the slice containing textPb in the picture TexturePic.

– When textPredFlagLX[xRef][yRef] is equal to 1, the following applies for i in the range of 0 to num_ref_idx_lX_active_minus1, inclusive:

  – When PicOrderCnt(RefPicListX[i]) is equal to PicOrderCnt(textRefPicListX[textRefIdxLX]), ViewIdx(RefPicListX[i]) is equal to ViewIdx(textRefPicListX[textRefIdxLX]), and predFlagLXT is equal to 0, the following applies:

$$predFlagLXT = 1 \quad (I-177)$$

$$refIdxLXT = i \quad (I-178)$$

$$mvLXT[0] = (textMvLX[xRef][yRef][0] + 2) >> 2 \quad (I-179)$$

$$mvLXT[1] = (textMvLX[xRef][yRef][1] + 2) >> 2 \quad (I-180)$$

I.8.5.3.2.12 Derivation process for disparity information merging candidates

Inputs to this process are:

– 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,

– two variables nPbW and nPbH specifying the width and the height of the current prediction block.

Outputs of this process are (with X in the range of 0 to 1, inclusive):

– the flags availableFlagDI and availableFlagDIShift, specifying whether the disparity information merging candidate and the shifted disparity information candidate are available,

– the prediction list utilization flags predFlagLXDI and predFlagLXDIShift,

– the reference indices refIdxLXDI and refIdxLXDIShift,

– the motion vectors mvLXDI and mvLXDIShift.

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

– The variables predFlagLXDI and predFlagLXDIShift are set equal to 0, the variables refIdxLXDI and refIdxLXDIShift are set equal to −1, and the motion vectors mvLXDI and mvLXDIShift are set equal to (0, 0).

The disparity information merging candidate is derived as follows:

– For X in the range of 0 to (slice_type = = B) ? 1 : 0, inclusive, 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 equal to the PicOrderCntVal, ViewIdx(RefPicListX[i]) is equal to RefViewIdx[xPb][yPb], and predFlagLXDI is equal to 0, the following applies:

$$availableFlagDI = 1 \quad (I-181)$$

$$predFlagLXDI = 1 \quad (I-182)$$

$$refIdxLXDI = i \quad (I-183)$$

$$mvLXDI = (DispRefVec[xPb][yPb][0], 0) \quad (I-184)$$

When availableFlagDI is equal to 1, the shifted disparity information merging candidate is derived as follows:

– The variable availableFlagDIShift is set equal to 1.

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

predFlagLXDIShift = predFlagLXDI (I−185)

refIdxLXDIShift = refIdxLXDI (I−186)

mvLXDIShift = predFlagLXDI ? (mvLXDI[0] + 4, mvLXDI[1]): (0, 0) (I−187)

I.8.5.3.2.13 Derivation process for a view synthesis prediction merging candidate

Inputs to this process are:

– 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,

– two variables nPbW and nPbH specifying the width and the height of the current prediction block.

Outputs of this process are (with X in the range of 0 to 1, inclusive):

– the availability flag availableFlagVSP specifying whether the view synthesis prediction merging candidate is available,

– the prediction list utilization flag predFlagLXVSP,

– the reference index refIdxLXVSP,

– the motion vector mvLXVSP.

The variable availableFlagVSP is set equal to 0 and for X in the range of 0 to 1, inclusive, the following applies:

– The variable predFlagLXVSP is set equal to 0, the variable refIdxLXVSP is set equal to −1, and the motion vector mvLXVSP is set equal to (0, 0).

For X in the range of 0 to (slice_type = = B ? 1 : 0), inclusive, the following applies:

– For i in the range of 0 to num_ref_idx_lX_active_minus1, inclusive, the following applies:

  – When availableFlagVSP is equal to 0 and ViewIdx(RefPicListX[i]) is equal to RefViewIdx[xPb][yPb], the following applies:

    availableFlagVSP = 1 (I−188)

    predFlagLXVSP = 1 (I−189)

    refIdxLXVSP = i (I−190)

    mvLXVSP = DispVec[xPb][yPb] (I−191)

When availableFlagVSP is equal to 1 the following applies:

– The derivation process for a depth or disparity sample array from a depth picture as specified in clause I.8.5.7 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, the disparity vector DispVec[xPb][yPb], the reference view order index RefViewIdx[xPb][yPb], and the variable partIdc equal to 2 as inputs, and the outputs are the array disparitySamples of size (nPbW)x(nPbH), and the flag horSplitFlag.

– 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:

  – The variable SubPbArrayPartSizeVSP[xPb + x][yPb + y] is set equal to (horSplitFlag ? (8, 4) : (4, 8)).

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

    – The variables SubPbArrayPredFlagLXVSP, SubPbArrayMvLXVSP, and SubPbArrayRefIdxLXVSP are derived as follows:

      SubPbArrayPredFlagLXVSP[xPb + x][yPb + y] = predFlagLXVSP (I−192)

      SubPbArrayRefIdxLXVSP[xPb + x][yPb + y] = refIdxLXVSP (I−193)

      SubPbArrayMvLXVSP[xPb + x][yPb + y] =
      predFlagLXVSP ? (disparitySamples[x][y], 0) : (0, 0) (I−194)

    – When ChromaArrayType is not equal to 0, the derivation process for chroma motion vectors as specified in clause 8.5.3.2.9 is invoked with SubPbArrayMvLXVSP[xPb + x][yPb + y] as input, and the output is SubPbArrayMvCLXVSP[xPb + x][yPb + y].

I.8.5.3.3 Decoding process for inter prediction samples

I.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,

– when ChromaArrayType is not equal to 0, 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,

– when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below,

– when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below.

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

– The variable mvLX is set equal to (mvLX << 2).

– When ChromaArrayType is not equal to 0, the variable mvCLX is set equal to (mvCLX << 2).

The variable $nCbS_L$ is set equal to nCbS. When ChromaArrayType is not equal to 0, the variable $nCbSw_C$ is set equal to nCbS / SubWidthC and the variable $nCbSh_C$ is set equal to nCbS / SubHeightC.

1. Let $predSamplesL0_L$ and $predSamplesL1_L$ be (nPbW)x(nPbH) arrays of predicted luma sample values and, when ChromaArrayType is not equal to 0, $predSamplesL0_{Cb}$, $predSamplesL1_{Cb}$, $predSamplesL0_{Cr}$, and $predSamplesL1_{Cr}$ be (nPbW / SubWidthC)x(nPbH / SubHeightC) arrays of predicted chroma sample values.

2. For X in the range of 0 to 1, inclusive, when predFlagLX is equal to 1, the following applies:

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

     – If iv_res_pred_weight_idx[xCb][yCb] is not equal to 0, the bilinear sample interpolation and residual prediction process as specified in clause I.8.5.3.3.3 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 and nPbH, the prediction list utilization flags predFlagL0 and predFlagL1, the reference indices refIdxL0 and refIdxL1, the motion vectors mvL0 and mvL1, when ChromaArrayType is not equal to 0, the motion vectors mvCL0 and mvCL1, and the prediction list indication X as inputs, and the array $predSamplesLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $predSamplesLX_{Cb}$ and $predSamplesLX_{Cr}$, as outputs.

     – Otherwise (iv_res_pred_weight_idx[xCb][yCb] is equal to 0), the following applies:

       – The reference picture consisting of an ordered two-dimensional array $refPicLX_L$ of luma samples and, when ChromaArrayType is not equal to 0, two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples is derived by invoking the process specified in clause 8.5.3.3.2 with refIdxLX as input.

       – The array $predSamplesLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $predSamplesLX_{Cb}$ and $predSamplesLX_{Cr}$ are derived by invoking the fractional sample interpolation process specified in clause 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, when ChromaArrayType is not equal to 0, mvCLX, and the reference arrays $refPicLX_L$, $refPicLX_{Cb}$, and $refPicLX_{Cr}$ as inputs.

3. Depending on the value of IlluCompFlag[xCb][yCb], the array $predSamples_L$ is derived as follows:
   - If IlluCompFlag[xCb][yCb] is equal to 0, the following applies:
     - The prediction samples inside the current luma prediction block, $predSamples_L$[x_L + xBl][y_L + yBl] with x_L = 0..nPbW − 1 and y_L = 0..nPbH − 1, are derived by invoking the weighted sample prediction process specified in clause 8.5.3.3.4 with the prediction block width nPbW, the 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.
   - Otherwise (IlluCompFlag[xCb][yCb] is equal to 1), the following applies:
     - The prediction samples inside the current luma prediction block, $predSamples_L$[x_L + xBl][y_L + yBl] with x_L = 0..nPbW − 1 and y_L = 0..nPbH − 1, are derived by invoking the illumination compensated sample prediction process specified in clause I.8.5.3.3.2, 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 and nPbH, the sample arrays $predSamplesL0_L$ and $predSamplesL1_L$, the variables predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvL0, mvL1 and cIdx equal to 0 as inputs.

4. When ChromaArrayType is not equal to 0, depending on the values of IlluCompFlag[xCb][yCb] and nPbW, the arrays $predSamples_{Cb}$, and $predSamples_{Cr}$ are derived as follows:
   - If IlluCompFlag[xCb][yCb] is equal to 0 or nPbW is less than or equal to 8, the following applies:
     - The prediction samples inside the current chroma component Cb prediction block, $predSamples_{Cb}$[x_C + xBl / SubWidthC][y_C + yBl / SubHeightC] with x_C = 0..nPbW / SubWidthC − 1 and y_C = 0..nPbH / SubHeightC − 1 , are derived by invoking the weighted sample prediction process specified in clause 8.5.3.3.4 with the prediction block width nPbW set equal to nPbW / SubWidthC, the prediction block height nPbH set equal to nPbH / SubHeightC, the sample arrays $predSamplesL0_{Cb}$ and $predSamplesL1_{Cb}$, and the variables predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 1 as inputs.
     - The prediction samples inside the current chroma component Cr prediction block, $predSamples_{Cr}$[x_C + xBl / SubWidthC][y_C + yBl / SubHeightC] with x_C = 0..nPbW / SubWidthC − 1 and y_C = 0..nPbH / SubHeightC − 1 , are derived by invoking the weighted sample prediction process specified in clause 8.5.3.3.4 with the prediction block width nPbW set equal to nPbW / SubWidthC, the prediction block height nPbH set equal to nPbH / SubHeightC, the sample arrays $predSamplesL0_{Cr}$ and $predSamplesL1_{Cr}$, and the variables predFlagL0, predFlagL1, refIdxL0, refIdxL1, and cIdx equal to 2 as inputs.
   - Otherwise (IlluCompFlag[xCb][yCb] is equal to 1 and nPbW is greater than 8), the following applies:
     - The prediction samples inside the current chroma component Cb prediction block, $predSamples_{Cb}$[x_C + xBl / SubWidthC][y_C + yBl / SubHeightC] with x_C = 0..nPbW / SubWidthC − 1 and y_C = 0..nPbH / SubHeightC − 1 , are derived by invoking the illumination compensated sample prediction process specified in clause I.8.5.3.3.2, with the luma location (xCb, yCb), the size of the current luma coding block nCbS, the chroma location (xBl / SubWidthC, yBl / SubHeightC), the width and the height of the current chroma prediction block (nPbW / SubWidthC) and (nPbH / SubHeightC), the sample arrays $predSamplesL0_{Cb}$ and $predSamplesL1_{Cb}$, the variables predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvCL0, mvCL1, and cIdx equal to 1 as inputs.
     - The prediction samples inside the current chroma component Cr prediction block, $predSamples_{Cr}$[x_C + xBl / SubWidthC][y_C + yBl / SubHeightC] with x_C = 0..nPbW / SubWidthC − 1 and y_C = 0..nPbH / SubHeightC − 1 , are derived by invoking the illumination compensated sample prediction process specified in clause I.8.5.3.3.2, with the luma location (xCb, yCb), the size of the current luma coding block nCbS, the chroma location (xBl / SubWidthC, yBl / SubHeightC), the width and the height of the current chroma prediction block (nPbW / SubWidthC) and (nPbH / SubHeightC), the sample arrays $predSamplesL0_{Cr}$ and $predSamplesL1_{Cr}$, the variables predFlagL0, predFlagL1, refIdxL0, refIdxL1, mvCL0, mvCL1, and cIdx equal to 2 as inputs.

I.8.5.3.3.2 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 luma or chroma prediction block relative to the top-left sample of the current luma coding block,

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

– two (nPbW)x(nPbH) arrays predSamplesL0 and predSamplesL1,

– two prediction list utilization flags predFlagL0 and predFlagL1,

– two reference indices refIdxL0 and refIdxL1,

– two motion vector mvL0 and mvL1,

– a colour component index cIdx.

Output of this process is an (nPbW)x(nPbH) array predSamples of prediction sample values.

The variables bitDepth, shift1, shift2, offset1, and offset2 are derived as follows:

$$\text{bitDepth} = (\text{ cIdx } == 0 \text{ }) \text{ ? BitDepth}_Y : \text{BitDepth}_C \quad \text{(I−195)}$$

$$\text{shift1} = \text{Max}(2, 14 - \text{bitDepth}) \quad \text{(I−196)}$$

$$\text{shift2} = \text{Max}(3, 15 - \text{bitDepth}) \quad \text{(I−197)}$$

$$\text{offset1} = 1 << (\text{shift1} - 1) \quad \text{(I−198)}$$

$$\text{offset2} = 1 << (\text{shift2} - 1) \quad \text{(I−199)}$$

The derivation process for illumination compensation mode availability and parameters as specified in clause I.8.5.3.3.2.1 is invoked with the luma location (xCb, yCb), the size of the current luma coding block nCbS, the prediction list utilization flags predFlagL0 and predFlagL1, the reference indices refIdxL0 and refIdxL1, the motion vectors mvL0 and mvL1, the variable bitDepth, and the variable cIdx as inputs, and the outputs are the flags puIcFlagL0 and puIcFlagL1, the variables icWeightL0 and icWeightL1, and the variables icOffsetL0 and icOffsetL1.

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:

$$\text{clipPredVal} = \text{Clip3}(0, (1 << \text{bitDepth}) - 1, (\text{predSamplesLX}[x][y] + \text{offset1}) >> \text{shift1}) \quad \text{(I−200)}$$

$$\begin{aligned}\text{predValX} = {} & \text{!puIcFlagLX ? clipPredVal :}\\ & (\text{Clip3}(0, (1 << \text{bitDepth}) - 1, (\text{clipPredVal} * \text{icWeightLX}) >> 5) + \text{icOffsetLX})\end{aligned} \quad \text{(I−201)}$$

– If predFlagL0 and predFlagL1 are equal to 1, the following applies:

$$\text{predSamples}[x][y] = \text{Clip3}(0, (1 << \text{bitDepth}) - 1, (\text{predVal0} + \text{predVal1} + \text{offset2}) >> \text{shift2}) \quad \text{(I−202)}$$

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

$$\text{predSamples}[x][y] = \text{predFlagL0 ? predVal0 : predVal1} \quad \text{(I−203)}$$

I.8.5.3.3.2.1 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 luma coding block relative to the top-left luma sample of the current picture,

– the size of the current luma coding block nCbS,

– two prediction list utilization flags predFlagL0 and predFlagL1,

– two reference indices refIdxL0 and refIdxL1,

– two motion vectors mvL0 and mvL1,

– a bit depth of samples bitDepth,

– a variable cIdx specifying colour component index.

Outputs of this process are:

– the flags puIcFlagL0 and puIcFlagL1 specifying whether illumination compensation is enabled,

– the variables icWeightL0 and icWeightL1 specifying weights for illumination compensation,

– the 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 subWidth, subHeight, and the location (xC, yC) specifying the top-left sample of the current luma or chroma coding block are derived as follows:

subWidth = (cIdx = = 0) ? 1 : SubWidthC (I−204)

subHeight = (cIdx = = 0) ? 1 : SubHeightC (I−205)

(xC, yC) = (xCb / subWidth , yCb / subHeight) (I−206)

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 clause 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 inputs, 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 clause 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 inputs, and the output is assigned to availFlagCurLeftCol.

When availFlagCurAboveRow is equal to 1 or availFlagCurLeftCol is equal to 1, the following applies:

1. Depending on the value of cIdx, the variable curRecSamples specifying the reconstructed picture samples before deblocking filter of the current picture is derived as follows:

   curRecSamples = (cIdx = = 0) ? S_L : ((cIdx = = 1) ? S_{Cb} : S_{Cr}) (I−207)

2. For X in the range of 0 to 1, inclusive, when predFlagLX is equal to 1, the following applies:

   – Let refPicLX be the picture RefPicListX[refIdxLX].

   – When ViewIdx(refPicLX) is not equal to ViewIdx, the following applies:

     – The variable puIcFlagLX is set equal to 1.

     – The luma location (xRefBlkLX, yRefBlkLX) specifying the top-left sample of the reference block in the picture refPicLX is derived as follows:

       xRefBlkLX = xC + (mvLX[0] >> (2 + (cIdx ? 1 : 0))) (I−208)

       yRefBlkLX = yC + (mvLX[1] >> (2 + (cIdx ? 1 : 0))) (I−209)

     – Depending on the value of cIdx, the variable refRecSamplesLX specifying the reconstructed picture samples of the picture refPicLX is derived as follows:

       – If cIdx is equal to 0, refRecSamplesLX is set equal to reconstructed picture sample array S_L of picture refPicLX.

       – Otherwise, if cIdx is equal to 1, refRecSamplesLX is set equal to the reconstructed chroma sample array S_{Cb} of picture refPicLX.

       – Otherwise (cIdx is equal to 2), refRecSamplesLX is set equal to the reconstructed chroma sample array S_{Cr} of picture refPicLX.

3. The lists curSampleList, refSampleList0, and refSampleList1, specifying the neighbouring samples in pictures CurrPic, refPicL0, and refPicL1, respectively, are derived as follows:

   – The variable numSamples specifying the number of elements of curSampleList, refSampleList0, and refSampleList1 is set equal to 0.

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

     – When (curNbColFlag ? availFlagCurLeftCol : availFlagCurAboveRow) is equal to 1, the following applies, for i in the range of 0 to (nCbS / (curNbColFlag ? subHeight : subWidth)) − 1, inclusive:

– The variables xOff and yOff are derived as follows:

xOff = curNbColFlag ? −1: i (I−210)

yOff = curNbColFlag ? i : −1 (I−211)

– For X in the range of 0 to 1, inclusive, when puIcFlagLX is equal to 1, the following applies:

xP = Clip3(0, (pic_width_in_luma_samples / subWidth) − 1, xRefBlkLX + xOff) (I−212)

yP = Clip3(0, (pic_height_in_luma_samples / subHeight) − 1, yRefBlkLX + yOff) (I−213)

refSampleListLX[numSamples] = refRecSamplesLX[xP][yP] (I−214)

– The variables curSampleList and numSamples are modified as follows:

curSampleList[numSamples++] = curRecSamples[xC + xOff][yC + yOff] (I−215)

4. For X in the range of 0 to 1, inclusive, when puIcFlagLX is equal to 1, icWeightLX and icOffsetLX are modified as follows:

   – The derivation process for illumination compensation parameters as specified in clause I.8.5.3.3.2.2 is invoked, with the list of neighbouring samples in the current picture curSampleList, the list of neighbouring samples in the reference picture refSampleListX, the number of neighbouring samples numSamples, the variable bitDepth, and the variable cIdx as inputs, and the variables icWeightLX and icOffsetLX as outputs.

I.8.5.3.3.2.2 Derivation process for illumination compensation parameters

Inputs to this process are:

– a list curSampleList specifying the current neighbouring samples,

– a list refSampleList specifying the reference neighbouring samples,

– a variable numSamples specifying the number of elements of curSampleList and refSampleList,

– a variable bitDepth specifying the bit depth of samples,

– a variable cIdx specifying colour component index.

Outputs of this process are:

– the variable icWeight specifying a weight for illumination compensation,

– the variable icOffset specifying an offset for illumination compensation.

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

The variables sumRef and sumCur are set equal to 0 and the following applies for i in the range of 0 to (numSamples / 2 − 1), inclusive:

sumRef += refSampleList[2 * i] (I−216)

sumCur += curSampleList[2 * i] (I−217)

The variables avgShift and avgOffset are derived as follows:

avgShift = Log2(numSamples / 2) (I−218)

avgOffset = 1 << (avgShift − 1) (I−219)

Depending on the value of cIdx, the variables icWeight and icOffset are derived as follows:

– If cIdx is equal to 0, the following applies:

  – The variables sumRefSquare and sumProdRefCur are set equal to 0, and the following applies for i in the range of 0 to (numSamples / 2 − 1), inclusive:

sumRefSquare += (refSampleList[2 * i] * refSampleList[2 * i]) >> precShift (I−220)

sumProdRefCur += (refSampleList[2 * i] * curSampleList[2 * i]) >> precShift (I−221)

  – The variables numerDiv and denomDiv are derived as follows:

denomDiv = ((sumRefSquare + (sumRefSquare >> 7)) << avgShift)
− ((sumRef * sumRef) >> precShift) (I−222)

numerDiv = Clip3(0, 2 * denomDiv, ((sumProdRefCur + (sumRefSquare >> 7)) << avgShift)
− ((sumRef * sumCur) >> precShift)) (I−223)

– The variables shiftNumer and shiftDenom are derived as follows:

shiftDenom = Max(0, Floor(Log2(Abs(denomDiv))) − 5) (I−224)

shiftNumer = Max(0, shiftDenom − 12) (I−225)

– The variables sNumerDiv and sDenomDiv are derived as follows:

sDenomDiv = denomDiv >> shiftDenom (I−226)

sNumerDiv = numerDiv >> shiftNumer (I−227)

– The value of variable divCoeff is derived from Table I.3 depending on the value of sDenomDiv, and the variables icWeight and icOffset are derived as follows:

icWeight = (sNumerDiv * divCoeff) >> (shiftDenom − shiftNumer + 10) (I−228)

icOffset = (sumCur − ((icWeight * sumRef) >> 5) + avgOffset) >> avgShift (I−229)

– Otherwise (cIdx is not equal to 0), the following applies:

icWeight = 32 (I−230)

icOffset = (sumCur − sumRef + avgOffset) >> avgShift (I−231)

Table I.3 – 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	

I.8.5.3.3.3 Bilinear sample interpolation and residual prediction process

The process is only invoked when iv_res_pred_weight_idx[xCb][yCb] is in not equal to 0.

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 current luma prediction block,

– two prediction list utilization flags predFlagL0 and predFlagL1,

– two reference indices refIdxL0 and refIdxL1,

– two motion vectors mvL0 and mvL1,

– when ChromaArrayType is not equal to 0, two motion vectors mvCL0 and mvCL1,

– a prediction list indication X.

Outputs of this process are:

– the (nPbW)x(nPbH) array $predSamplesLX_L$,

– when ChromaArrayType is not equal to 0, the (nPbW / SubWidthC)x(nPbH / SubHeightC) arrays $predSamplesLX_{Cb}$ and $predSamplesLX_{Cr}$.

The location (xPb, yPb) is derived as follows:

$$xPb = xCb + xBl \quad (I-232)$$

$$yPb = yCb + yBl \quad (I-233)$$

The prediction list indication variable Y is set equal to (1 − X) and the variable availFlag is set equal to 0.

The variable ivRefFlagLX is set equal to (DiffPicOrderCnt(CurrPic, RefPicListX[refIdxLX]) = = 0) and the variable ivRefFlagLY is set equal to predFlagLY ? (DiffPicOrderCnt(CurrPic, RefPicListY[refIdxLY]) = = 0) : 0.

Depending on the values of ivRefFlagLX, ivRefFlagLY, and RpRefIdxLX, the following applies:

– If ivRefFlagLX is equal to 0, 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 clause I.8.5.3.3.3.3 is invoked with the prediction list indication variable equal to X, the motion vector mvLX, and the picture RefPicListX[refIdxLX] as inputs, and the modified motion vector mvLX as output.

– Otherwise (when ivRefFlagLX is equal to 1), the following applies:

  – If predFlagLY is equal to 1 and ivRefFlagLY is equal to 0, the following applies:

    – The variable availFlag is set equal to 1.

    – The residual prediction motion vector scaling process as specified in clause I.8.5.3.3.3.3 is invoked with the prediction list indication variable equal to Y, the motion vector mvLY, and the picture RefPicListY[refIdxLY] as inputs, and the modified motion vector mvLY as output.

    – The motion vector mvT is set equal to mvLY and the prediction list indication variable Z is set equal to Y.

  – Otherwise (predFlagLY is equal to 0 or ivRefFlagLY is equal to 1), the following applies:

    – The variable W is set equal to (predFlagLY && ivRefFlagLY) ? 0 : X.

    – The derivation process for a motion vector from a reference block for residual prediction as specified in clause I.8.5.3.3.3.4 is invoked with the luma location (xPb, yPb), the variables nPbW and nPbH, the picture RefPicListW[refIdxLW], and the motion vector mvLW as inputs, and the flag availFlag, the motion vector mvT, and the prediction list utilization variable Z as outputs.

    – When availFlag is equal to 0 and RpRefPicAvailFlagLW is equal to 1, availFlag is set equal to 1, mvT is set equal to (0, 0), and Z is set equal to W.

When ChromaArrayType is not equal to 0, the derivation process for chroma motion vectors in clause 8.5.3.2.10 is invoked with mvLX as input, and the output being mvCLX.

The array $predSamplesLX_L$, and, when ChromaArrayType is not equal to 0, the arrays $predSamplesLX_{Cb}$ and $predSamplesLX_{Cr}$ are derived as follows:

– The reference picture consisting of an ordered two-dimensional array $refPicLX_L$ of luma samples and, when ChromaArrayType is not equal to 0, two ordered two-dimensional arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ of chroma samples, are derived by invoking the reference picture selection process as specified in clause 8.5.3.3.2 with refIdxLX as input.

– The array $predSamplesLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $predSamplesLX_{Cb}$ and $predSamplesLX_{Cr}$, are derived by invoking the bilinear sample interpolation process specified in clause I.8.5.3.3.3.1 with the luma locations (xCb, yCb) and (xBl, yBl), the luma prediction block width nPbW, the luma prediction block height nPbH, the motion vector mvLX, and, when ChromaArrayType is not equal to 0, the motion vector mvCLX, the reference array $refPicLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $refPicLX_{Cb}$ and $refPicLX_{Cr}$ as inputs.

When one of the following conditions is true, availFlag is set equal to 0:

– ivRefFlagLX is equal to 0 and RefRpRefAvailFlagLX[RefViewIdx[xPb][yPb]] is equal to 0.

– ivRefFlagLX is equal to 1 and RefRpRefAvailFlagLZ[ViewIdx(RefPicListX[refIdxLX])] is equal to 0.

When availFlag is equal to 1, the following applies:

– Depending on the value of ivRefFlagLX, the variables rpPic, rpRefPic, and mvRp are derived as follows:

  – If ivRefFlagLX is equal to 0, the following applies:

    – Let rpPic be the picture with PicOrderCnt(rpPic) equal to PicOrderCntVal and nuh_layer_id equal to ViewCompLayerId[RefViewIdx[xPb][yPb]][DepthFlag].

    – Let rpRefPic be the picture with PicOrderCnt(rpRefPic) equal to PicOrderCnt(RefPicListX[RpRefIdxLX]) and nuh_layer_id equal to ViewCompLayerId[RefViewIdx[xPb][yPb]][DepthFlag].

    – The variable mvRp is set equal to DispVec[xPb][yPb].

  – Otherwise (ivRefFlagLX is equal to 1), the following applies:

    – Let rpPic be the picture RefPicListZ[RpRefIdxLZ].

    – Let rpRefPic be the picture with PicOrderCnt(rpRefPic) equal to PicOrderCnt(rpPic) and nuh_layer_id equal to ViewCompLayerId[ViewIdx(RefPicListX[refIdxLX])][DepthFlag].

    – The variable mvRp is set equal to mvT.

– When ChromaArrayType is not equal to 0, the derivation process for chroma motion vectors in clause 8.5.3.2.10 is invoked with mvRp as input, and the output being mvRpC.

– The array $rpSamplesLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $rpSamplesLX_{Cb}$ and $rpSamplesLX_{Cr}$ are derived as follows:

  – Let the reference sample array $rpPicLX_L$ correspond to the decoded sample array S_L derived in clause 8.7 for the previously decoded picture rpPic.

  – When ChromaArrayType is not equal to 0, let the reference sample arrays $rpPicLX_{Cb}$ and $rpPicLX_{Cr}$ correspond to the decoded sample arrays S_{Cb} and S_{Cr}, respectively, derived in clause 8.7 for the previously decoded picture rpPic.

  – The array $rpSamplesLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $rpSamplesLX_{Cb}$ and $rpSamplesLX_{Cr}$ are derived by invoking the bilinear sample interpolation process specified in clause I.8.5.3.3.3.1 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, when ChromaArrayType is not equal to 0, the motion vector mvCLX equal to mvRpC, the reference array $rpPicLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $rpPicLX_{Cb}$ and $rpPicLX_{Cr}$ as inputs.

– The array $rpRefSamplesLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $rpRefSamplesLX_{Cb}$ and $rpRefSamplesLX_{Cr}$ are derived as follows:

  – Let the reference sample array $rpRefPicLX_L$ correspond to the decoded sample array S_L derived in clause 8.7 for the previously decoded picture rpRefPic.

  – When ChromaArrayType is not equal to 0, let the reference sample arrays $rpRefPicLX_{Cb}$ and $rpRefPicLX_{Cr}$ correspond to the decoded sample arrays S_{Cb} and S_{Cr}, respectively, derived in clause 8.7 for the previously decoded picture rpRefPic.

  – The array $rpRefSamplesLX_L$ and, when ChromaArrayType is not equal to 0, the arrays $rpRefSamplesLX_{Cb}$ and $rpRefSamplesLX_{Cr}$ are derived by invoking the bilinear sample interpolation process specified in clause I.8.5.3.3.3.1 with the luma locations (xCb, yCb) and (xBl, yBl), the luma prediction block width nPbW, the luma prediction block height nPbH, the motion vector mvLX equal to (mvLX + mvRp), and, when ChromaArrayType is not equal to 0, the motion vector mvCLX equal to (mvCLX + mvRpC), the reference arrays with $rpRefPicLX_L$, and, when ChromaArrayType is not equal to 0, the arrays $rpRefPicLX_{Cb}$ and $rpRefPicLX_{Cr}$ as inputs.

– The variable shiftVal is set equal to (iv_res_pred_weight_idx[xCb][yCb] − 1).

– The modified prediction samples $predSamplesLX_L$[x][y] with x = 0..(nPbW) − 1 and y = 0..(nPbH) − 1 are derived as follows:

$$predSamplesLX_L[\ x\ ][\ y\ ] = predSamplesLX_L[\ x\ ][\ y\ ] + ((rpSamplesLX_L[\ x\ ][\ y\ ] - rpRefSamplesLX_L[\ x\ ][\ y\ ]) >> shiftVal) \quad (I-234)$$

– When ChromaArrayType is not equal to 0 and nPbW is greater than 8, the following applies:

  – The modified prediction samples $predSamplesLX_{Cb}$[x][y] with x = 0..(nPbW / SubWidthC) − 1 and y = 0..(nPbH / SubHeightC) − 1 are derived as follows:

$$predSamplesLX_{Cb}[\ x\ ][\ y\ ] = predSamplesLX_{Cb}[\ x\ ][\ y\ ] + ((rpSamplesLX_{Cb}[\ x\ ][\ y\ ] - rpRefSamplesLX_{Cb}[\ x\ ][\ y\ ]) >> shiftVal) \quad (I-235)$$

– The modified prediction samples $predSamplesLX_{Cr}[\ x\ ][\ y\ ]$ with $x = 0..(nPbW / SubWidthC) - 1$ and $y = 0..(nPbH / SubHeightC) - 1$ are derived as follows:

$$predSamplesLX_{Cr}[\ x\ ][\ y\ ] = predSamplesLX_{Cr}[\ x\ ][\ y\ ] + ((rpSamplesLX_{Cr}[\ x\ ][\ y\ ] - rpRefamplesLX_{Cr}[\ x\ ][\ y\ ]) >> shiftVal) \quad (I-236)$$

I.8.5.3.3.3.1 Bilinear sample interpolation process

The specifications in clause 8.5.3.3.3.1 apply with the following modifications:

– All invocations of the process specified in clause 8.5.3.3.3.2 are replaced with invocations of the process specified in clause I.8.5.3.3.3.2 with chromaFlag equal to 0 as additional input.

– All invocations of the process specified in clause 8.5.3.3.3.3 are replaced with invocations of the process specified in clause I.8.5.3.3.3.2 with chromaFlag equal to 1 as additional input.

I.8.5.3.3.3.2 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 sample array refPicLX,

– a flag chromaFlag.

Output of this process is a predicted sample value predSampleLX.

Depending on the value of chromaFlag, the following applies:

– If chromaFlag is equal 0, the following applies:

  – The variables xFrac and yFrac are set equal to (xFrac << 1) and (yFrac << 1), respectively.

  – The variables picWidth and picHeight are set equal to pic_width_in_luma_samples and pic_height_in_luma_samples, respectively.

– Otherwise (chromaFlag is equal to 1), the variables picWidth and picHeight are set equal to PicWidthInSamplesC and PicHeightInSamplesC, respectively.

In Figure 8-5, the positions labelled with upper-case letters $B_{i,j}$ within shaded blocks represent samples at full-sample locations inside the given two-dimensional array refPicLX of samples. These samples may be used for generating the predicted sample value predSampleLX. The locations ($xB_{i,j}$, $yB_{i,j}$) for each of the corresponding samples $B_{i,j}$ inside the given array refPicLX of samples are derived as follows:

$$xB_{i,j} = Clip3(0, picWidth - 1, xInt + i) \quad (I-237)$$

$$yB_{i,j} = Clip3(0, picHeight - 1, yInt + j) \quad (I-238)$$

The positions labelled with lower-case letters within un-shaded blocks represent samples at eighth-pel sample fractional locations. The location offset in fractional-sample units (xFrac, yFrac) specifies which of the generated samples at full-sample and fractional-sample locations is assigned to the predicted sample value predSampleLX. This assignment is as specified in Table 8-9, with xFracC, yFracC, and $predSampeLX_C$ replaced by xFrac, yFrac, and predSampleLX, respectively. The output is the value of predSampleLX.

The variables shift1, shift2, and shift3 are derived as follows:

– The variable bitDepth is set equal to chromaFlag ? $BitDepth_C$: $BitDepth_Y$.

– The variable shift1 is set equal to Min(4, bitDepth − 8), the variable shift2 is set equal to 6, and the variable shift3 is set equal to Max(2, 14 − bitDepth).

Given the samples $B_{i,j}$ at full-sample locations ($xB_{i,j}$, $yB_{i,j}$), the samples $ab_{0,0}$ to $hh_{0,0}$ at fractional sample positions are derived as follows:

– The samples labelled $ab_{0,0}$, $ac_{0,0}$, $ad_{0,0}$, $ae_{0,0}$, $af_{0,0}$, $ag_{0,0}$, and $ah_{0,0}$ are derived by applying a 2-tap filter to the nearest integer position samples as follows:

$$ab_{0,0} = (56 * B_{0,0} + 8 * B_{1,0}) >> shift1 \quad (I-239)$$

$$ac_{0,0} = (48 * B_{0,0} + 16 * B_{1,0}) >> shift1 \quad (I-240)$$

$$ad_{0,0} = (40 * B_{0,0} + 24 * B_{1,0}) >> shift1 \quad (I-241)$$

$$ae_{0,0} = (32 * B_{0,0} + 32 * B_{1,0}) >> shift1 \quad (I-242)$$

$$af_{0,0} = (24 * B_{0,0} + 40 * B_{1,0}) >> shift1 \quad (I-243)$$

$$ag_{0,0} = (16 * B_{0,0} + 48 * B_{1,0}) >> shift1 \quad (I-244)$$

$$ah_{0,0} = (8 * B_{0,0} + 56 * B_{1,0}) >> shift1 \quad (I-245)$$

– The samples labelled $ba_{0,0}$, $ca_{0,0}$, $da_{0,0}$, $ea_{0,0}$, $fa_{0,0}$, $ga_{0,0}$, and $ha_{0,0}$ are derived by applying a 2-tap filter to the nearest integer position samples as follows:

$$ba_{0,0} = (56 * B_{0,0} + 8 * B_{0,1}) >> shift1 \quad (I-246)$$

$$ca_{0,0} = (48 * B_{0,0} + 16 * B_{0,1}) >> shift1 \quad (I-247)$$

$$da_{0,0} = (40 * B_{0,0} + 24 * B_{0,1}) >> shift1 \quad (I-248)$$

$$ea_{0,0} = (32 * B_{0,0} + 32 * B_{0,1}) >> shift1 \quad (I-249)$$

$$fa_{0,0} = (24 * B_{0,0} + 40 * B_{0,1}) >> shift1 \quad (I-250)$$

$$ga_{0,0} = (16 * B_{0,0} + 48 * B_{0,1}) >> shift1 \quad (I-251)$$

$$ha_{0,0} = (8 * B_{0,0} + 56 * B_{0,1}) >> shift1 \quad (I-252)$$

– The samples labelled $bX_{0,0}$, $cX_{0,0}$, $dX_{0,0}$, $eX_{0,0}$, $fX_{0,0}$, $gX_{0,0}$, and $hX_{0,0}$ for X being replaced by b, c, d, e, f, g, and h, respectively, are derived by applying a 2-tap filter to the intermediate values $aX_{0,i}$ with i = 0..1 in the vertical direction as follows:

$$bX_{0,0} = (56 * aX_{0,0} + 8 * aX_{0,1}) >> shift2 \quad (I-253)$$

$$cX_{0,0} = (48 * aX_{0,0} + 16 * aX_{0,1}) >> shift2 \quad (I-254)$$

$$dX_{0,0} = (40 * aX_{0,0} + 24 * aX_{0,1}) >> shift2 \quad (I-255)$$

$$eX_{0,0} = (32 * aX_{0,0} + 32 * aX_{0,1}) >> shift2 \quad (I-256)$$

$$fX_{0,0} = (24 * aX_{0,0} + 40 * aX_{0,1}) >> shift2 \quad (I-257)$$

$$gX_{0,0} = (16 * aX_{0,0} + 48 * aX_{0,1}) >> shift2 \quad (I-258)$$

$$hX_{0,0} = (8 * aX_{0,0} + 56 * aX_{0,1}) >> shift2 \quad (I-259)$$

I.8.5.3.3.3.3 Residual prediction motion vector scaling process

Inputs to this process are:

– a prediction list indication 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 follows:

$$tx = (16384 + (Abs(td) >> 1)) / td \quad (I-260)$$

$$distScaleFactor = Clip3(-4096, 4095, (tb * tx + 32) >> 6) \quad (I-261)$$

$$mv = Clip3(-32768, 32767, Sign(distScaleFactor * mvLX) * ((Abs(distScaleFactor * mvLX) + 127) >> 8)) \quad (I-262)$$

where td and tb are derived as:

$$td = Clip3(-128, 127, DiffPicOrderCnt(CurrPic, refPicLX)) \quad (I-263)$$

$$tb = Clip3(-128, 127, DiffPicOrderCnt(CurrPic, RefPicListX[RpRefIdxLX])) \quad (I-264)$$

I.8.5.3.3.3.4 Derivation process for a motion vector from a reference block for residual prediction

Inputs to this process are:

– 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,

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

– a reference picture refPic,

– a disparity vector dispVec.

Outputs of this process are:

– a flag availFlag,

– a motion vector mvT,

– a prediction list indication variable Y.

The flag availFlag is set equal to 0, the motion vector mvT is set equal to (0, 0), and the prediction list indication variable Y is set equal to 0.

The reference luma location (xRef, yRef) in refPic is derived as follows:

xRefFull = xPb + (nPbW >> 1) + ((dispVec[0] + 2) >> 2) (I−265)

yRefFull = yPb + (nPbH >> 1) + ((dispVec[1] + 2) >> 2) (I−266)

xRef = Clip3(0, pic_width_in_luma_samples − 1, (xRefFull >> 3) << 3) (I−267)

yRef = Clip3(0, pic_height_in_luma_samples − 1, (yRefFull >> 3) << 3) (I−268)

Let refPb be the luma prediction block covering the luma position (xRef, yRef) in the picture refPic.

The variable cuPredModeRef[x][y] is set equal to CuPredMode[x][y] of the picture refPic.

When cuPredModeRef[xRef][yRef] is equal to MODE_SKIP or MODE_INTER, the following applies for X in the range of 0 to 1, inclusive:

– The variables predFlagRef[x][y], mvRef[x][y], and refIdxRef[x][y] are set equal to PredFlagLX[x][y], MvLX[x][y], and RefIdxLX[x][y], respectively, of picture refPic.

– The variable refPicListRef is set equal to RefPicListX of the slice containing refPb in the picture refPic.

– When availFlag is equal to 0, predFlagRef[xRef][yRef] is equal to 1, DiffPicOrderCnt(refPic, refPicListRef[refIdxRef[xRef][xRef]]) is not equal to 0, and RpRefPicAvailFlagLX is equal to 1, the following applies:

  – The variable availFlag is set equal to 1.

  – The variable Y is set equal to X.

  – The residual prediction motion vector scaling process as specified in clause I.8.5.3.3.3.3 is invoked with the prediction list indication variable equal to X, the motion vector mvRef[xRef][yRef], and the reference picture refPicListRef[refIdxRef[xRef][xRef]] as inputs, and the output being the motion vector mvT.

I.8.5.3.4 Decoding process for inter sample prediction for rectangular sub-block partitions

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

– when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below,

– when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below.

When ChromaArrayType is not equal to 0, the variable $nCbSw_C$ is set equal to (nCbS / SubWidthC) and the variable $nCbSh_C$ is set equal to (nCbS / SubHeightC).

The luma location (xPb, yPb) specifying the top-left sample of the current luma prediction block relative to the top-left sample of the current picture is set equal to (xCb + xBl, yCb + yBl).

The variables nSbW and nSbH specifying the width and height of the sub-block partitions are derived as follows:

(nSbW, nSbH) = SubPbArrayPartSize[xPb][yPb] (I−269)

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-block partition relative to the top-left sample of the current luma coding block is derived as follows:

    xSb = xBl + x * nSbW (I−270)

    ySb = yBl + y * nSbH (I−271)

  – For X in the range of 0 to 1, inclusive, the variables mvLX, mvCLX, refIdxLX, and predFlagLX are derived as follows:

    mvLX = SubPbArrayMvLX[xCb + xSb][yCb + ySb] (I−272)

    mvCLX = (ChromaArrayType != 0) ? SubPbArrayMvCLX[xCb + xSb][yCb + ySb] : (0, 0) (I−273)

    refIdxLX = SubPbArrayRefIdxLX[xCb + xSb][yCb + ySb] (I−274)

    predFlagLX = SubPbArrayPredFlagLX[xCb + xSb][yCb + ySb] (I−275)

  – The decoding process for inter sample prediction as specified in clause I.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, when ChromaArrayType is not equal to 0, 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 that are an (nCbSL)x(nCbSL) array $predSamples_L$ of prediction luma samples, and, when ChromaArrayType is not equal to 0, two $(nCbSw_C)x(nCbSh_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr, as outputs.

I.8.5.3.5 Decoding process for inter sample prediction for depth predicted sub-block partitions

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,

– two luma motion vectors mvL0 and mvL1,

– when ChromaArrayType is not equal to 0, the chroma motion vectors mvCL0 and mvCL1,

– two reference indices refIdxL0 and refIdxL1,

– two prediction list utilization flags predFlagL0, and predFlagL1,

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

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,

– when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $predSamples_{Cb}$ of chroma prediction samples for the component Cb, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below,

– when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $predSamples_{Cr}$ of chroma prediction samples for the component Cr, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below.

The variable $nCbS_L$ is set equal to nCbS. When ChromaArrayType is not equal to 0, the variable $nCbSw_C$ is set equal to nCbS / SubWidthC and the variable $nCbSh_C$ is set equal to nCbS / SubHeightC.

The decoding process for inter sample prediction as specified in clause I.8.5.3.3.1 is invoked with the luma coding block location (xCb, yCb), the luma prediction block location (xBl, yBl) set to (0, 0), the luma coding block size nCbS, the luma prediction block width nPbW equal to nCbS, the luma prediction block height nPbH equal to nCbS, the luma motion vectors mvL0 and mvL1, when ChromaArrayType is not equal to 0, 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 $(nCbS_L)x(nCbS_L)$ array $predSamples_L$ of prediction luma samples and, when ChromaArrayType is not equal to 0, two $(nCbSw_C)x(nCbSh_C)$ arrays $predSamples_{Cb}$ and $predSamples_{Cr}$ of prediction chroma samples, one for each of the chroma components Cb and Cr, as outputs.

The partition pattern contourPattern is derived as follows:

– The derivation process for a depth or disparity sample array from a depth picture as specified in clause I.8.5.7 is invoked with the luma location (xBlk, yBlk) equal to (xCb, yCb), the variable nBlkW equal to nCbS, the variable nBlkH equal to nCbS, the disparity vector dispVec equal to DispRefVec[xCb][yCb], the reference view order index refViewIdx equal to RefViewIdx[xCb][yCb], and the variable partIdc equal to 1 as inputs, and the output is the array refSamples of size (nCbS)x(nCbS).

– The variable threshVal is derived as follows:

$$\begin{aligned} threshVal = (& refSamples[0][0] + refSamples[0][nCbS - 1] \\ & + refSamples[nCbS - 1][0] + refSamples[nCbS - 1][nCbS - 1]) >> 2 \end{aligned} \qquad \text{(I−276)}$$

– For x = 0..nCbS − 1 and y = 0..nCbS − 1, the following applies:

$$contourPattern[x][y] = (refSamples[x][y] > threshVal) \qquad \text{(I−277)}$$

The array $TempSamplesDbbp_L$ and, when ChromaArrayType is not equal to 0, the arrays $TempSamplesDbbp_{Cb}$ and $TempSamplesDbbp_{Cr}$ are modified as follows:

```
for( y = 0; y < nCbS_L; y++ )
    for( x = 0; x < nCbS_L; x++ )
        if( contourPattern[ x ][ y ] = = ( partIdx != contourPattern[ 0 ][ 0 ] ) ) {
            TempSamplesDbbp_L[ x ][ y ] = predSamples_L[ x ][ y ]
            if( ChromaArrayType != 0 && ( x % SubWidthC ) = = 0 && ( y % SubHeightC ) = = 0 ) {
                x_C = x / SubWidthC                                                    (I−278)
                y_C = y / SubHeightC
                TempSamplesDbbp_Cb[ x_C ][ y_C ] = predSamples_Cb[ x_C ][ y_C ]
                TempSamplesDbbp_Cr[ x_C ][ y_C ] = predSamples_Cr[ x_C ][ y_C ]
            }
        }
```

When partIdx is equal to 1, the array $predSamples_L$ and, when ChromaArrayType is not equal to 0, the arrays $predSamples_{Cb}$ and $predSamples_{Cr}$ are modified as follows:

– The derivation process for contour boundary filtered samples as specified in clause I.8.5.3.5.1.1 is invoked with, the luma coding block size $nCbS_L$, the current coding block width nCbSw equal to $nCbS_L$, the current coding block height nCbSh equal to $nCbS_L$, the partition pattern contourPattern, and the array predSamples of prediction samples equal to $TempSamplesDbbp_L$ as inputs, and the output is assigned to the array $predSamples_L$ of luma prediction samples.

– When ChromaArrayType is not equal to 0, the derivation process for contour boundary filtered samples as specified in clause I.8.5.3.5.1.1 is invoked with, the luma coding block size $nCbS_L$, the current coding block width nCbSw equal to $nCbSw_C$, the current coding block height nCbSh equal to $nCbSh_C$, the partition pattern contourPattern, and the array predSamples of prediction samples equal to $TempSamplesDbbp_{Cb}$ as inputs, and the output is assigned to the array $predSamples_{Cb}$ of chroma prediction samples.

– When ChromaArrayType is not equal to 0, the derivation process for contour boundary filtered samples as specified in clause I.8.5.3.5.1.1 is invoked with, the luma coding block size $nCbS_L$, the current coding block width nCbSw equal to nCbSw, the current coding block height nCbSh equal to $nCbSh_C$, the partition pattern contourPattern, and the array predSamples of prediction samples equal to $TempSamplesDbbp_{Cr}$ as inputs, and the output is assigned to the array $predSamples_{Cr}$ of chroma prediction samples.

I.8.5.3.5.1.1 Derivation process for contour boundary filtered samples

Inputs to this process are:

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

– a variable nCbSw specifying the width of the current luma or chroma coding block,

– a variable nCbSh specifying the height of the current luma or chroma coding block,

– an $(nCbS_L)x(nCbS_L)$ partition pattern contourPattern,

– an (nCbSw)x(nCbSh) array predSamples prediction samples.

Output of this process is a modified (nCbSw)x(nCbSh) array predSamples of prediction samples.

The (nCbSw)x(nCbSh) array p is set equal to predSamples.

The variable xOff, yOff, $nCbS_N$, and n are derived as follows:

– If PartMode is equal to PART_Nx2N, xOff is set equal to 1, yOff is set equal to 0, $nCbS_N$ is set equal to nCbSw, and n is set equal to ($nCbS_L$ / nCbSw).

– Otherwise (PartMode is not equal to PART_Nx2N), xOff is set equal to 0, yOff is set equal to 1, $nCbS_N$ is set equal to nCbSh, and n is set equal to ($nCbS_L$ / nCbSh).

The values of predSamples are derived as follows:

```
for( y = 0; y < nCbSh; y++ )
    for( x = 0; x < nCbSw; x++ ) {
        filt = p[ x ][ y ]
        prevFlag = contourPattern[ Max( 0, n * ( x − xOff ) ) ][ Max( 0, n * ( y − yOff ) ) ]            (I−279)
        nextFlag = contourPattern[ Min( n * ( x + xOff ), nCbS_L − 1 ) ][ Min( n * ( y + yOff ), nCbS_L − 1 ) ]
        if( prevFlag  !=  nextFlag )
            filt = ( p[ Max( 0, x − xOff ) ][ Max( 0, y − yOff ) ] + ( filt  <<  1 ) +
                    p[ Min( x + xOff, nCbS_N − 1 ) ][ Min( y + yOff, nCbS_N − 1 ) ] )  >>  2
        predSamples[ x ][ y ] = filt
    }
```

I.8.5.4 Decoding process for the residual signal of coding units coded in inter prediction mode

I.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 $(nCbS_L)x(nCbS_L)$ array $resSamples_L$ of luma residual samples, where $nCbS_L$ is derived as specified below,

– when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $resSamples_{Cb}$ of chroma residual samples for the component Cb, where nCbSwC and nCbShC are derived as specified below,

– when ChromaArrayType is not equal to 0, an $(nCbSw_C)x(nCbSh_C)$ array $resSamples_{Cr}$ of chroma residual samples for the component Cr, where $nCbSw_C$ and $nCbSh_C$ are derived as specified below.

The variable $nCbS_L$ is set equal to 1 << log2CbSize. When ChromaArrayType is not equal to 0, the variable $nCbSw_C$ is set equal to $nCbS_L$ / SubWidthC and the variable $nCbSh_C$ is set equal to $nCbS_L$ / SubHeightC.

Let $resSamples_L$ be an $(nCbS_L)x(nCbS_L)$ array of luma residual samples and, when ChromaArrayType is not equal to 0, let $resSamples_{Cb}$ and $resSamples_{Cr}$ be two $(nCbSw_C)x(nCbSh_C)$ arrays of chroma residual samples.

– If DcOnlyFlag[xCb][yCb] is equal to 0, the following applies, depending on the value of rqt_root_cbf:

  – If rqt_root_cbf is equal to 0 or cu_skip_flag[xCb][yCb] is equal to 1, all samples of the $(nCbS_L)x(nCbS_L)$ array $resSamples_L$ and, when ChromaArrayType is not equal to 0, all samples of the two $(nCbSw_C)x(nCbSh_C)$ 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 clause 8.5.4.2 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 a modified version of the $(nCbS_L)x(nCbS_L)$ array $resSamples_L$ as output.

    2. When ChromaArrayType is not equal to 0, the decoding process for chroma residual blocks as specified in

clause 8.5.4.3 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 nCbSw set equal to $nCbSw_C$, the variable nCbSh set equal to $nCbSh_C$, and the $(nCbSw_C)x(nCbSh_C)$ array $resSamples_{Cb}$ as inputs, and a modified version of the $(nCbSw_C)x(nCbSh_C)$ array $resSamples_{Cb}$ as output.

3. When ChromaArrayType is not equal to 0, the decoding process for chroma residual blocks as specified in clause 8.5.4.3 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 nCbSw set equal to $nCbSw_C$, the variable nCbSh set equal to $nCbSh_C$, and the $(nCbSw_C)x(nCbSh_C)$ array $resSamples_{Cr}$ as inputs, and a modified version of the $(nCbSw_C)x(nCbSh_C)$ array $resSamples_{Cr}$ as output.

– Otherwise (DcOnlyFlag[xCb][yCb] is equal to 1), for x in the range of 0 to $nCbS_L - 1$, inclusive, and y in the range of 0 to $nCbS_L - 1$, inclusive, $resSamples_L$[x][y] is set equal to DcOffset[xCb][yCb][0].

NOTE – When DcOnlyFlag[xCb][yCb] is equal to 1, ChromaArrayType is equal to 0 in this version of this Specification.

I.8.5.5 Derivation process for a disparity vector for texture layers

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 dvAvailFlag is set equal to 0 and the disparity vector dispVec is set equal to (0, 0).

The variable checkParallelMergeFlag is derived as follows:

– If one or more of the following conditions are true, checkParallelMergeFlag is set equal to 1:

  – CuPredMode[xCb][yCb] is equal to MODE_SKIP.

  – CuPredMode[xCb][yCb] is equal to MODE_INTER and merge_flag[xCb][yCb] is equal to 1.

– Otherwise, checkParallelMergeFlag is set equal to 0.

When slice_temporal_mvp_enabled_flag is equal to 1, the derivation process for a disparity vector from temporal neighbouring blocks as specified in clause I.8.5.5.1 is invoked with the luma location (xCb, yCb), and the variable nCbS as inputs, and the outputs are the flag dvAvailFlag, the disparity vector dispVec, and the reference view order index refViewIdx.

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

1. The variable N is set equal to (i = = 0) ? A_1 : B_1.

2. The variable (xN, yN) is set equal to (i = = 0) ? (xCb − 1, yCb + nCbS − 1) : (xCb + nCbS − 1, yCb − 1).

3. The derivation process for z-scan order block availability as specified in clause 6.4.1 is invoked with (xCurr, yCurr) set equal to the (xCb, yCb) and the luma location (xN, yN) as inputs, and the output is assigned to nbAvailFlag.

4. When CuPredMode[xN][yN] is equal to MODE_INTRA, nbAvailFlag is set equal to 0.

5. When all of the following conditions are true, nbAvailFlag is set equal to 0.

   – checkParallelMergeFlag is equal to 1,

   – (xCb >> Log2ParMrgLevel) is equal to (xN >> Log2ParMrgLevel),

   – (yCb >> Log2ParMrgLevel) is equal to (yN >> Log2ParMrgLevel).

6. The flag tPredNbAvailFlag is set equal to nbAvailFlag.

7. When N is equal to B_1 and ((yN >> CtbLog2SizeY) << CtbLog2SizeY) is less than ((yCb >> CtbLog2SizeY) << CtbLog2SizeY), tPredNbAvailFlag is set equal to 0.

8. The flag tPredNbDvAvailFlagN is set equal to 0.

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

   – When dvAvailFlag is equal to 0, nbAvailFlag is equal to 1, and PredFlagLX[xN][yN] is equal to 1, the following applies:

– If DiffPicOrderCnt(RefPicListX[RefIdxLX[xN][yN]], CurrPic) is equal to 0, the following applies:

refViewIdx = ViewIdx(RefPicListX[RefIdxLX[xN][yN]]) (I−280)

dispVec = MvLXN[xN][yN] (I−281)

dvAvailFlag = 1 (I−282)

– Otherwise (DiffPicOrderCnt(RefPicListX[RefIdxLX[xN][yN]] , CurrPic) is not equal to 0), when tPredNbAvailFlag is equal to 1, tPredNbDvAvailFlagN is equal to 0, CuPredMode[xN][yN] is equal to MODE_SKIP, and IvMcFlag[xN][yN] is equal to 1, the following applies:

tPredNbDispVecN = DispRefVec[xN][yN] (I−283)

tPredNbRefViewIdxN = RefViewIdx[xN][yN] (I−284)

tPredNbDvAvailFlagN = 1 (I−285)

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

– The variable N is set equal to (i = = 0) ? A_1 : B_1.

– When dvAvailFlag is equal to 0 and tPredNbDvAvailFlagN is equal to 1, the following applies:

dispVec = tPredNbDispVecN (I−286)

refViewIdx = tPredNbRefViewIdxN (I−287)

dvAvailFlag = 1 (I−288)

When dvAvailFlag is equal to 0, refViewIdx is set equal to DefaultRefViewIdx and dispVec is set equal to (0, 0).

Depending on the value of DepthRefEnabledFlag, the following applies:

– If DepthRefEnabledFlag is equal to 1, the following applies:

  – The derivation process for a depth or disparity sample array from a depth picture as specified in clause I.8.5.7 is invoked with the luma location (xBlk, yBlk) equal to (xCb, yCb), the variable nBlkW equal to nCbS, the variable nBlkH equal to nCbS, the disparity vector dispVec, the reference view order index refViewIdx, and the variable partIdc equal to 0 as inputs, and the array disparitySamples of size (nCbS)x(nCbS) as output.

  – The disparity vector dispRefVec is set equal to (disparitySamples[0][0], 0).

– Otherwise (DepthRefEnabledFlag is equal to 0), the disparity vector dispRefVec is set equal to dispVec.

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):

DispVec[x][y] = dispVec (I−289)

DispRefVec[x][y] = dispRefVec (I−290)

RefViewIdx[x][y] = refViewIdx (I−291)

I.8.5.5.1 Derivation process for a disparity vector from temporal neighbouring 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 availability flag dvAvailFlag,

– the disparity vector dispVec,

– the reference view order index refViewIdx.

The luma location (xRef, yRef) is derived as follows:

xRef = ((xCb + nCbS / 2) >> 4) << 4 (I−292)

yRef = ((yCb + nCbS / 2) >> 4) << 4 (I−293)