

SYNTAX CONSTRAINTS IN PARAMETER SET SIGNALING OF SUBPICTURES**CROSS REFERENCE TO RELATED APPLICATIONS**

[001] Under the applicable patent law and/or rules pursuant to the Paris Convention, this application is made to timely claim the priority to and benefits of International Patent Application No. PCT/CN2019/111807, filed on October 18, 2019. For all purposes under the law, the entire disclosure of the aforementioned application is incorporated by reference as part of the disclosure of this application.

TECHNICAL FIELD

[002] This document is related to video and image coding and decoding technologies.

BACKGROUND

[003] Digital video accounts for the largest bandwidth use on the internet and other digital communication networks. As the number of connected user devices capable of receiving and displaying video increases, it is expected that the bandwidth demand for digital video usage will continue to grow.

SUMMARY

[004] The disclosed techniques may be used by video or image decoder or encoder embodiments for in which sub-picture based coding or decoding is performed.

[005] In one example aspect a method of video processing is disclosed. The method includes performing a conversion between a picture of a video and a bitstream representation of the video. The picture comprises one or more sub-pictures, and the bitstream representation conforms to a format rule that specifies that a length of a syntax element is equal to $\text{Ceil}(\text{Log}_2(\text{SS}))$ bits. SS is greater than 0, and the syntax element indicating a horizontal or a vertical position of a top-left corner of a coding tree unit of a sub-picture of the picture.

[006] In another example aspect a method of video processing is disclosed. The method includes performing a conversion between a picture of a video and a bitstream representation of the video, wherein the picture comprises one or more sub-pictures. The bitstream representation conforms to a format rule that specifies that different sub-pictures have different identifiers.

[007] In another example aspect a method of video processing is disclosed. The method includes determining, for a video block in a first video region of a video, whether a position at which a temporal motion vector predictor is determined for a conversion between the video block and a bitstream representation of the current video block using an affine mode is within a second video region; and performing the conversion based on the determining.

[008] In another example aspect, another method of video processing is disclosed. The method includes determining, for a video block in a first video region of a video, whether a position at which an integer sample in a reference picture is fetched for a conversion between the video block and a bitstream representation of the current video block is within a second video region, wherein the reference picture is not used in an interpolation process during the conversion; and performing the conversion based on the determining.

[009] In another example aspect, another method of video processing is disclosed. The method includes determining, for a video block in a first video region of a video, whether a position at which a reconstructed luma sample value is fetched for a conversion between the video block and a bitstream representation of the current video block is within a second video region; and performing the conversion based on the determining.

[0010] In another example aspect, another method of video processing is disclosed. The method includes determining, for a video block in a first video region of a video, whether a position at which a check regarding splitting, depth derivation or split flag signaling for the video block is performed during a conversion between the video block and a bitstream representation of the current video block is within a second video region; and performing the conversion based on the determining.

[0011] In another example aspect, another method of video processing is disclosed. The method includes performing a conversion between a video comprising one or more video pictures comprising one or more video blocks, and a coded representation of the video, wherein the coded representation complies with a coding syntax requirement that the conversion is not to use sub-picture coding/decoding and a dynamic resolution conversion coding/decoding tool or a reference picture resampling tool within a video unit.

[0012] In another example aspect, another method of video processing is disclosed. The method includes performing a conversion between a video comprising one or more video pictures comprising one or more video blocks, and a coded representation of the video, wherein the coded

representation complies with a coding syntax requirement that a first syntax element `subpic_grid_idx[i][j]` is not larger than a second syntax element `max_subpics_minus1`.

[0013] In yet another example aspect, the above-described method may be implemented by a video encoder apparatus that comprises a processor.

[0014] In yet another example aspect, the above-described method may be implemented by a video decoder apparatus that comprises a processor.

[0015] In yet another example aspect, these methods may be embodied in the form of processor-executable instructions and stored on a computer-readable program medium.

[0016] These, and other, aspects are further described in the present document.

BRIEF DESCRIPTION OF THE DRAWINGS

[0017] FIG. 1 shows an example of region constraint in temporal motion vector prediction (TMVP) and sub-block TMVP.

[0018] FIG. 2 shows an example of a hierarchical motion estimation scheme.

[0019] FIG. 3 is a block diagram of an example of a hardware platform used for implementing techniques described in the present document.

[0020] FIG. 4 is a flowchart for an example method of video processing.

[0021] FIG. 5 shows an example of a picture with 18 by 12 luma CTUs that is partitioned into 12 tiles and 3 raster-scan slices (informative).

[0022] FIG. 6 shows an example of picture with 18 by 12 luma CTUs that is partitioned into 24 tiles and 9 rectangular slices (informative).

[0023] FIG. 7 shows an example of a picture that is partitioned into 4 tiles, 11 bricks, and 4 rectangular slices (informative).

[0024] FIG. 8 is a block diagram showing an example video processing system in which various techniques disclosed herein may be implemented.

[0025] FIG. 9 is a block diagram that illustrates an example video coding system.

[0026] FIG. 10 is a block diagram that illustrates an encoder in accordance with some embodiments of the present disclosure.

[0027] FIG. 11 is a block diagram that illustrates a decoder in accordance with some embodiments of the present disclosure.

[0028] FIG. 12 is a flowchart representation of a method for video processing in accordance with the present technology.

[0029] FIG. 13 is a flowchart representation of another method for video processing in accordance with the present technology.

[0030] FIG. 14 is a flowchart representation of another method for video processing in accordance with the present technology.

[0031] FIG. 15 is a flowchart representation of another method for video processing in accordance with the present technology.

[0032] FIG. 16 is a flowchart representation of another method for video processing in accordance with the present technology.

[0033] FIG. 17 is a flowchart representation of another method for video processing in accordance with the present technology.

[0034] FIG. 18 is a flowchart representation of another method for video processing in accordance with the present technology.

[0035] FIG. 19 is a flowchart representation of yet another method for video processing in accordance with the present technology.

DETAILED DESCRIPTION

[0036] The present document provides various techniques that can be used by a decoder of image or video bitstreams to improve the quality of decompressed or decoded digital video or images. For brevity, the term “video” is used herein to include both a sequence of pictures (traditionally called video) and individual images. Furthermore, a video encoder may also implement these techniques during the process of encoding in order to reconstruct decoded frames used for further encoding.

[0037] Section headings are used in the present document for ease of understanding and do not limit the embodiments and techniques to the corresponding sections. As such, embodiments from one section can be combined with embodiments from other sections.

[0038] 1. Summary

[0039] This document is related to video coding technologies. Specifically, it is related to palette coding with employing base colors based representation in video coding. It may be applied to the

existing video coding standard like HEVC, or the standard (Versatile Video Coding) to be finalized. It may be also applicable to future video coding standards or video codec.

[0040] 2. Initial Discussion

[0041] Video coding standards have evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 and H.263, ISO/IEC produced MPEG-1 and MPEG-4 Visual, and the two organizations jointly produced the H.262/MPEG-2 Video and H.264/MPEG-4 Advanced Video Coding (AVC) and H.265/HEVC standards [1,2]. Since H.262, the video coding standards are based on the hybrid video coding structure wherein temporal prediction plus transform coding are utilized. To explore the future video coding technologies beyond HEVC, Joint Video Exploration Team (JVET) was founded by VCEG and MPEG jointly in 2015. Since then, many new methods have been adopted by JVET and put into the reference software named Joint Exploration Model (JEM). In April 2018, the Joint Video Expert Team (JVET) between VCEG (Q6/16) and ISO/IEC JTC1 SC29/WG11 (MPEG) was created to work on the VVC standard targeting at 50% bitrate reduction compared to HEVC.

[0042] 2.1 The region constraint in TMVP and sub-block TMVP in VVC

[0043] FIG. 1 illustrates example region constraint in TMVP and sub-block TMVP. In TMVP and sub-block TMVP, it is constrained that a temporal MV can only be fetched from the collocated CTU plus a column of 4×4 blocks as shown in FIG. 1.

[0044] 2.2 Example Sub-picture

[0045] In some embodiments, sub-picture-based coding techniques based on flexible tiling approach can be implemented. Summary of the sub-picture-based coding techniques includes the following:

[0046] (1) Pictures can be divided into sub-pictures.

[0047] (2) The indication of existence of sub-pictures is indicated in the SPS, along with other sequence-level information of sub-pictures.

[0048] (3) Whether a sub-picture is treated as a picture in the decoding process (excluding in-loop filtering operations) can be controlled by the bitstream.

[0049] (4) Whether in-loop filtering across sub-picture boundaries is disabled can be controlled by the bitstream for each sub-picture. The DBF, SAO, and ALF processes are updated for controlling of in-loop filtering operations across sub-picture boundaries.

[0050] (5) For simplicity, as a starting point, the sub-picture width, height, horizontal offset, and vertical offset are signalled in units of luma samples in SPS. Sub-picture boundaries are constrained to be slice boundaries.

[0051] (6) Treating a sub-picture as a picture in the decoding process (excluding in-loop filtering operations) is specified by slightly updating the `coding_tree_unit()` syntax, and updates to the following decoding processes:

[0052] - The derivation process for (advanced) temporal luma motion vector prediction

[0053] - The luma sample bilinear interpolation process

[0054] - The luma sample 8-tap interpolation filtering process

[0055] - The chroma sample interpolation process

[0056] (7) Sub-picture IDs are explicitly specified in the SPS and included in the tile group headers to enable extraction of sub-picture sequences without the need of changing VCL NAL units.

[0057] (8) Output sub-picture sets (OSPS) are proposed to specify normative extraction and conformance points for sub-pictures and sets thereof.

[0058] 2.3 Example Sub-pictures in Versatile Video Coding

Sequence parameter set RBSP syntax

seq_parameter_set_rbsp() {	Descriptor
sps_decoding_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
...	
pic_width_max_in_luma_samples	ue(v)
pic_height_max_in_luma_samples	ue(v)
subpics_present_flag	u(1)
if(subpics_present_flag) {	
max_subpics_minus1	u(8)
subpic_grid_col_width_minus1	u(v)
subpic_grid_row_height_minus1	u(v)
for(i = 0; i < NumSubPicGridRows; i++)	
for(j = 0; j < NumSubPicGridCols; j++)	
subpic_grid_idx[i][j]	u(v)
for(i = 0; i <= NumSubPics; i++) {	
subpic_treated_as_pic_flag[i]	u(1)
loop_filter_across_subpic_enabled_flag[i]	u(1)
}	

}	
...	
}	

subpics_present_flag equal to 1 indicates that sub-picture parameters are present in the present in the SPS RBSP syntax. **subpics_present_flag** equal to 0 indicates that sub-picture parameters are not present in the present in the SPS RBSP syntax.

NOTE 2 – When a bitstream is the result of a sub-bitstream extraction process and contains only a subset of the sub-pictures of the input bitstream to the sub-bitstream extraction process, it might be required to set the value of **subpics_present_flag** equal to 1 in the RBSP of the SPSs.

max_subpics_minus1 plus 1 specifies the maximum number of sub-pictures that may be present in the CVS.

max_subpics_minus1 shall be in the range of 0 to 254. The value of 255 is reserved for future use by ITU-T | ISO/IEC.

subpic_grid_col_width_minus1 plus 1 specifies the width of each element of the sub-picture identifier grid in units of 4 samples. The length of the syntax element is $\text{Ceil}(\text{Log2}(\text{pic_width_max_in_luma_samples} / 4))$ bits.

The variable NumSubPicGridCols is derived as follows:

$$\text{NumSubPicGridCols} = (\text{pic_width_max_in_luma_samples} + \text{subpic_grid_col_width_minus1} * 4 + 3) / (\text{subpic_grid_col_width_minus1} * 4 + 4) \quad (7-5)$$

subpic_grid_row_height_minus1 plus 1 specifies the height of each element of the sub-picture identifier grid in units of 4 samples. The length of the syntax element is $\text{Ceil}(\text{Log2}(\text{pic_height_max_in_luma_samples} / 4))$ bits.

The variable NumSubPicGridRows is derived as follows:

$$\text{NumSubPicGridRows} = (\text{pic_height_max_in_luma_samples} + \text{subpic_grid_row_height_minus1} * 4 + 3) / (\text{subpic_grid_row_height_minus1} * 4 + 4) \quad (7-6)$$

subpic_grid_idx[i][j] specifies the sub-picture index of the grid position (i, j). The length of the syntax element is $\text{Ceil}(\text{Log2}(\text{max_subpics_minus1} + 1))$ bits.

The variables SubPicTop[subpic_grid_idx[i][j]], SubPicLeft[subpic_grid_idx[i][j]],

SubPicWidth[subpic_grid_idx[i][j]], SubPicHeight[subpic_grid_idx[i][j]], and NumSubPics are derived as follows:

NumSubPics = 0

for(i = 0; i < NumSubPicGridRows; i++) {

 for(j = 0; j < NumSubPicGridCols; j++) {

 if (i == 0)

 SubPicTop[subpic_grid_idx[i][j]] = 0

 else if(subpic_grid_idx[i][j] != subpic_grid_idx[i - 1][j]) {

 SubPicTop[subpic_grid_idx[i][j]] = i

```

        SubPicHeight[ subpic_grid_idx[ i - 1 ][ j ] ]      =      i - SubPicTop[ subpic_grid_idx[ i - 1 ][ j ] ]
    }
    if ( j == 0 )
        SubPicLeft[ subpic_grid_idx[ i ][ j ] ] = 0
    else if ( subpic_grid_idx[ i ][ j ] != subpic_grid_idx[ i ][ j - 1 ] ) {
        SubPicLeft[ subpic_grid_idx[ i ][ j ] ] = j
        SubPicWidth[ subpic_grid_idx[ i ][ j ] ] = j - SubPicLeft[ subpic_grid_idx[ i ][ j - 1 ] ]
    }
    if ( i == NumSubPicGridRows - 1 )
        SubPicHeight[ subpic_grid_idx[ i ][ j ] ] = i - SubPicTop[ subpic_grid_idx[ i - 1 ][ j ] ] + 1
    if ( j == NumSubPicGridRows - 1 )
        SubPicWidth[ subpic_grid_idx[ i ][ j ] ]      =      j - SubPicLeft[ subpic_grid_idx[ i ][ j - 1 ] ] + 1
    if ( subpic_grid_idx[ i ][ j ] > NumSubPics )
        NumSubPics = subpic_grid_idx[ i ][ j ]
    }
}

```

subpic_treated_as_pic_flag[i] equal to 1 specifies that the i-th sub-picture of each coded picture in the CVS is treated as a picture in the decoding process excluding in-loop filtering operations. **subpic_treated_as_pic_flag[i]** equal to 0 specifies that the i-th sub-picture of each coded picture in the CVS is not treated as a picture in the decoding process excluding in-loop filtering operations. When not present, the value of **subpic_treated_as_pic_flag[i]** is inferred to be equal to 0.

loop_filter_across_subpic_enabled_flag[i] equal to 1 specifies that in-loop filtering operations may be performed across the boundaries of the i-th sub-picture in each coded picture in the CVS.

loop_filter_across_subpic_enabled_flag[i] equal to 0 specifies that in-loop filtering operations are not performed across the boundaries of the i-th sub-picture in each coded picture in the CVS. When not present, the value of **loop_filter_across_subpic_enabled_flag[i]** is inferred to be equal to 1.

It is a requirement of bitstream conformance that the following constraints apply:

- For any two sub-pictures subpicA and subpicB, when the index of subpicA is less than the index of subpicB, any coded NAL unit of subpicA shall succeed any coded NAL unit of subpicB in decoding order.
- The shapes of the sub-pictures shall be such that each sub-picture, when decoded, shall have its entire left boundary and entire top boundary consisting of picture boundaries or consisting of boundaries of previously decoded sub-pictures.

The list **CtbToSubPicIdx[ctbAddrRs]** for **ctbAddrRs** ranging from 0 to **PicSizeInCtbsY - 1**, inclusive, specifying the conversion from a CTB address in picture raster scan to a sub-picture index, is derived as follows:

```

for( ctbAddrRs = 0; ctbAddrRs < PicSizeInCtbsY; ctbAddrRs++ ) {
    posX = ctbAddrRs % PicWidthInCtbsY * CtbSizeY

```

posY = ctbAddrRs / PicWidthInCtbsY * CtbSizeY

CtbToSubPicIdx[ctbAddrRs] = -1

```
for( i = 0; CtbToSubPicIdx[ ctbAddrRs ] < 0 && i < NumSubPics; i++ ) {
    if( ( posX >= SubPicLeft[ i ] * ( subpic_grid_col_width_minus1 + 1 ) * 4 ) &&
        ( posX < ( SubPicLeft[ i ] + SubPicWidth[ i ] ) *
            ( subpic_grid_col_width_minus1 + 1 ) * 4 ) &&
        ( posY >= SubPicTop[ i ] *
            ( subpic_grid_row_height_minus1 + 1 ) * 4 ) &&
        ( posY < ( SubPicTop[ i ] + SubPicHeight[ i ] ) *
            ( subpic_grid_row_height_minus1 + 1 ) * 4 ) )
        CtbToSubPicIdx[ ctbAddrRs ] = i
    }
}
```

num_bricks_in_slice_minus1, when present, specifies the number of bricks in the slice minus 1. The value of **num_bricks_in_slice_minus1** shall be in the range of 0 to **NumBricksInPic** - 1, inclusive. When **rect_slice_flag** is equal to 0 and **single_brick_per_slice_flag** is equal to 1, the value of **num_bricks_in_slice_minus1** is inferred to be equal to 0. When **single_brick_per_slice_flag** is equal to 1, the value of **num_bricks_in_slice_minus1** is inferred to be equal to 0.

The variable **NumBricksInCurrSlice**, which specifies the number of bricks in the current slice, and

SliceBrickIdx[i], which specifies the brick index of the *i*-th brick in the current slice, are derived as follows:

```
if( rect_slice_flag ) {
    sliceIdx = 0
    while( slice_address != slice_id[ sliceIdx ] )
        sliceIdx++
    NumBricksInCurrSlice = NumBricksInSlice[ sliceIdx ]
    brickIdx = TopLeftBrickIdx[ sliceIdx ]
    for( bIdx = 0; brickIdx <= BottomRightBrickIdx[ sliceIdx ]; brickIdx++ ) (7-92)
        if( BricksToSliceMap[ brickIdx ] == sliceIdx )
            SliceBrickIdx[ bIdx++ ] = brickIdx
    }
    else {
        NumBricksInCurrSlice = num_bricks_in_slice_minus1 + 1
        SliceBrickIdx[ 0 ] = slice_address
        for( i = 1; i < NumBricksInCurrSlice; i++ )
            SliceBrickIdx[ i ] = SliceBrickIdx[ i - 1 ] + 1
    }
}
```

The variables SubPicIdx, SubPicLeftBoundaryPos, SubPicTopBoundaryPos, SubPicRightBoundaryPos, and SubPicBotBoundaryPos are derived as follows:

```

SubPicIdx = CtbToSubPicIdx[ CtbAddrBsToRs[ FirstCtbAddrBs[ SliceBrickIdx[ 0 ] ] ] ]
if( subpic_treated_as_pic_flag[ SubPicIdx ] ) {
    SubPicLeftBoundaryPos = SubPicLeft[ SubPicIdx ] * ( subpic_grid_col_width_minus1 + 1 ) * 4
    SubPicRightBoundaryPos = ( SubPicLeft[ SubPicIdx ] + SubPicWidth[ SubPicIdx ] ) *
        ( subpic_grid_col_width_minus1 + 1 ) * 4
    SubPicTopBoundaryPos = SubPicTop[ SubPicIdx ] * ( subpic_grid_row_height_minus1 + 1 ) * 4
    SubPicBotBoundaryPos = ( SubPicTop[ SubPicIdx ] + SubPicHeight[ SubPicIdx ] ) *
        ( subpic_grid_row_height_minus1 + 1 ) * 4
}
...

```

Derivation process for temporal luma motion vector prediction

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 cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples,
- a reference index refIdxLX, with X being 0 or 1.

Outputs of this process are:

- the motion vector prediction mvLXCol in 1/16 fractional-sample accuracy,
- the availability flag availableFlagLXCol.

The variable currCb specifies the current luma coding block at luma location (xCb, yCb).

The variables mvLXCol and availableFlagLXCol are derived as follows:

- If slice_temporal_mvp_enabled_flag is equal to 0 or (cbWidth * cbHeight) is less than or equal to 32, both components of mvLXCol are set equal to 0 and availableFlagLXCol is set equal to 0.
- Otherwise (slice_temporal_mvp_enabled_flag is equal to 1), the following ordered steps apply:
 1. The bottom right collocated motion vector and the bottom and right boundary sample locations are derived as follows:

$$xColBr = xCb + cbWidth \quad (8-421)$$

$$yColBr = yCb + cbHeight \quad (8-422)$$

$$rightBoundaryPos = subpic_treated_as_pic_flag[SubPicIdx] ?$$

$$SubPicRightBoundaryPos : pic_width_in_luma_samples - 1 \quad (8-423)$$

$$botBoundaryPos = subpic_treated_as_pic_flag[SubPicIdx] ?$$

$$SubPicBotBoundaryPos : pic_height_in_luma_samples - 1 \quad (8-424)$$

- If yCb >> CtbLog2SizeY is equal to yColBr >> CtbLog2SizeY, yColBr is less than or equal to botBoundaryPos and xColBr is less than or equal to rightBoundaryPos, the following applies:

- The variable colCb specifies the luma coding block covering the modified location given by $((xColBr \gg 3) \ll 3, (yColBr \gg 3) \ll 3)$ inside the collocated picture specified by ColPic.
- The luma location $(xColCb, yColCb)$ is set equal to the top-left sample of the collocated luma coding block specified by colCb relative to the top-left luma sample of the collocated picture specified by ColPic.
- The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb, $(xColCb, yColCb)$, refIdxLX and sbFlag set equal to 0 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.

...

Luma sample bilinear interpolation process

Inputs to this process are:

- a luma location in full-sample units $(xInt_L, yInt_L)$,
- a luma location in fractional-sample units $(xFrac_L, yFrac_L)$,
- the luma reference sample array refPicLX_L.

Output of this process is a predicted luma sample value predSampleLX_L.

The variables shift1, shift2, shift3, shift4, offset1, offset2 and offset3 are derived as follows:

$$\text{shift1} = \text{BitDepth}_Y - 6 \quad (8-453)$$

$$\text{offset1} = 1 \ll (\text{shift1} - 1) \quad (8-454)$$

$$\text{shift2} = 4 \quad (8-455)$$

$$\text{offset2} = 1 \ll (\text{shift2} - 1) \quad (8-456)$$

$$\text{shift3} = 10 - \text{BitDepth}_Y \quad (8-457)$$

$$\text{shift4} = \text{BitDepth}_Y - 10 \quad (8-458)$$

$$\text{offset4} = 1 \ll (\text{shift4} - 1) \quad (8-459)$$

The variable picW is set equal to pic_width_in_luma_samples and the variable picH is set equal to pic_height_in_luma_samples.

The luma interpolation filter coefficients $fb_L[p]$ for each 1/16 fractional sample position p equal to $xFrac_L$ or $yFrac_L$ are specified in Table 8-10.

The luma locations in full-sample units $(xInt_i, yInt_i)$ are derived as follows for $i = 0..1$:

- If subpic_treated_as_pic_flag[SubPicIdx] is equal to 1, the following applies:

$$xInt_i = \text{Clip3}(\text{SubPicLeftBoundaryPos}, \text{SubPicRightBoundaryPos}, xInt_L + i) \quad (8-460)$$

$$yInt_i = \text{Clip3}(\text{SubPicTopBoundaryPos}, \text{SubPicBotBoundaryPos}, yInt_L + i) \quad (8-461)$$

- Otherwise (subpic_treated_as_pic_flag[SubPicIdx] is equal to 0), the following applies:

$$\begin{aligned} xInt_i &= \text{Clip3}(0, \text{picW} - 1, \text{sps_ref_wraparound_enabled_flag} \text{ ? } \\ &\quad \text{ClipH}((\text{sps_ref_wraparound_offset_minus1} + 1) * \text{MinCbSizeY}, \text{picW}, (xInt_L + i)) : (8-462) \\ &\quad xInt_L + i) \\ yInt_i &= \text{Clip3}(0, \text{picH} - 1, yInt_L + i) \end{aligned} \quad (8-463)$$

...

Derivation process for subblock-based temporal merging candidates

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 cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples.
- the availability flag availableFlagA₁ of the neighbouring coding unit,
- the reference index refIdxLXA₁ of the neighbouring coding unit with X being 0 or 1,
- the prediction list utilization flag predFlagLXA₁ of the neighbouring coding unit with X being 0 or 1,
- the motion vector in 1/16 fractional-sample accuracy mvLXA₁ of the neighbouring coding unit with X being 0 or 1.

Outputs of this process are:

- the availability flag availableFlagSbCol,
- the number of luma coding subblocks in horizontal direction numSbX and in vertical direction numSbY,
- the reference indices refIdxL0SbCol and refIdxL1SbCol,
- the luma motion vectors in 1/16 fractional-sample accuracy mvL0SbCol[xSbIdx][ySbIdx] and mvL1SbCol[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX - 1, ySbIdx = 0 .. numSbY - 1,
- the prediction list utilization flags predFlagL0SbCol[xSbIdx][ySbIdx] and predFlagL1SbCol[xSbIdx][ySbIdx] with xSbIdx = 0..numSbX - 1, ySbIdx = 0 .. numSbY - 1.

The availability flag availableFlagSbCol is derived as follows.

- If one or more of the following conditions is true, availableFlagSbCol is set equal to 0.
 - slice_temporal_mvp_enabled_flag is equal to 0.
 - sps_sbtmvp_enabled_flag is equal to 0.
 - cbWidth is less than 8.
 - cbHeight is less than 8.
- Otherwise, the following ordered steps apply:
 1. The location (xCtb, yCtb) of the top-left sample of the luma coding tree block that contains the current coding block and the location (xCtr, yCtr) of the below-right center sample of the current luma coding block are derived as follows:

$$xCtb = (xCb \gg \text{CtuLog2Size}) \ll \text{CtuLog2Size} \quad (8-542)$$

$$yCtb = (yCb \gg \text{CtuLog2Size}) \ll \text{CtuLog2Size} \quad (8-543)$$

$$xCtr = xCb + (cbWidth / 2) \quad (8-544)$$

$$yCtr = yCb + (cbHeight / 2) \quad (8-545)$$

2. The luma location (xColCtrCb, yColCtrCb) is set equal to the top-left sample of the collocated luma coding block covering the location given by (xCtr, yCtr) inside ColPic relative to the top-left luma sample of the collocated picture specified by ColPic.
3. The derivation process for subblock-based temporal merging base motion data as specified in clause 8.5.5.4 is invoked with the location (xCtb, yCtb), the location (xColCtrCb, yColCtrCb), the availability flag availableFlagA₁, and the prediction list utilization flag predFlagLXA₁, and the reference index refIdxLXA₁, and the motion vector mvLXA₁, with X being 0 and 1 as inputs and the motion vectors ctrMvLX, and the prediction list utilization flags ctrPredFlagLX of the collocated block, with X being 0 and 1, and the temporal motion vector tempMv as outputs.
4. The variable availableFlagSbCol is derived as follows:
 - If both ctrPredFlagL0 and ctrPredFlagL1 are equal to 0, availableFlagSbCol is set equal to 0.
 - Otherwise, availableFlagSbCol is set equal to 1.

When availableFlagSbCol is equal to 1, the following applies:

- The variables numSbX, numSbY, sbWidth, sbHeight and refIdxLXSbCol are derived as follows:

$$\text{numSbX} = \text{cbWidth} \gg 3 \quad (8-546)$$

$$\text{numSbY} = \text{cbHeight} \gg 3 \quad (8-547)$$

$$\text{sbWidth} = \text{cbWidth} / \text{numSbX} \quad (8-548)$$

$$\text{sbHeight} = \text{cbHeight} / \text{numSbY} \quad (8-549)$$

$$\text{refIdxLXSbCol} = 0 \quad (8-550)$$
- For xSbIdx = 0..numSbX – 1 and ySbIdx = 0..numSbY – 1, the motion vectors mvLXSbCol[xSbIdx][ySbIdx] and prediction list utilization flags predFlagLXSbCol[xSbIdx][ySbIdx] are derived as follows:
 - The luma location (xSb, ySb) specifying the top-left sample of the current coding subblock relative to the top-left luma sample of the current picture is derived as follows:

$$xSb = xCb + xSbIdx * sbWidth + sbWidth / 2 \quad (8-551)$$

$$ySb = yCb + ySbIdx * sbHeight + sbHeight / 2 \quad (8-552)$$
 - The location (xColSb, yColSb) of the collocated subblock inside ColPic is derived as follows.
 - The following applies:

$$\begin{aligned} yColSb &= \text{Clip3}(yCtb, \\ &\quad \text{Min}(\text{CurPicHeightInSamplesY} - 1, yCtb + (1 \ll \text{CtbLog2SizeY}) - 1), \\ &\quad ySb + (\text{tempMv}[1] \gg 4)) \end{aligned} \quad (8-553)$$
 - If subpic_treated_as_pic_flag[SubPicIdx] is equal to 1, the following applies:

$$\begin{aligned} xColSb &= \text{Clip3}(xCtb, \\ &\quad \text{Min}(\text{SubPicRightBoundaryPos}, xCtb + (1 \ll \text{CtbLog2SizeY}) + 3), \\ &\quad xSb + (\text{tempMv}[0] \gg 4)) \end{aligned} \quad (8-554)$$

- Otherwise (subpic_treated_as_pic_flag[SubPicIdx] is equal to 0), the following applies:

$$\begin{aligned} \text{xColSb} &= \text{Clip3}(\text{xCtb}, \\ &\quad \text{Min}(\text{CurPicWidthInSamplesY} - 1, \text{xCtb} + (1 \ll \text{CtbLog2SizeY}) + 3), (8-555) \\ &\quad \text{xSb} + (\text{tempMv}[0] \gg 4)) \end{aligned}$$

...

Derivation process for subblock-based temporal merging base motion data

Inputs to this process are:

- the location (xCtb, yCtb) of the top-left sample of the luma coding tree block that contains the current coding block,
- the location (xColCtCb, yColCtCb) of the top-left sample of the collocated luma coding block that covers the below-right center sample.
- the availability flag availableFlagA₁ of the neighbouring coding unit,
- the reference index refIdxLXA₁ of the neighbouring coding unit,
- the prediction list utilization flag predFlagLXA₁ of the neighbouring coding unit,
- the motion vector in 1/16 fractional-sample accuracy mvLXA₁ of the neighbouring coding unit.

Outputs of this process are:

- the motion vectors ctrMvL0 and ctrMvL1,
- the prediction list utilization flags ctrPredFlagL0 and ctrPredFlagL1,
- the temporal motion vector tempMv.

The variable tempMv is set as follows:

$$\text{tempMv}[0] = 0 \quad (8-558)$$

$$\text{tempMv}[1] = 0 \quad (8-559)$$

The variable currPic specifies the current picture.

When availableFlagA₁ is equal to TRUE, the following applies:

- If all of the following conditions are true, tempMv is set equal to mvL0A₁:
 - predFlagL0A₁ is equal to 1,
 - DiffPicOrderCnt(ColPic, RefPicList[0][refIdxL0A₁]) is equal to 0,
- Otherwise, if all of the following conditions are true, tempMv is set equal to mvL1A₁:
 - slice_type is equal to B,
 - predFlagL1A₁ is equal to 1,
 - DiffPicOrderCnt(ColPic, RefPicList[1][refIdxL1A₁]) is equal to 0.

The location (xColCb, yColCb) of the collocated block inside ColPic is derived as follows.

- The following applies:

$$\begin{aligned} \text{yColCb} &= \text{Clip3}(\text{yCtb}, \\ &\quad \text{Min}(\text{CurPicHeightInSamplesY} - 1, \text{yCtb} + (1 \ll \text{CtbLog2SizeY}) - 1), (8-560) \\ &\quad \text{yColCtCb} + (\text{tempMv}[1] \gg 4)) \end{aligned}$$

- If subpic_treated_as_pic_flag[SubPicIdx] is equal to 1, the following applies:

$$\begin{aligned}
 xColCb &= Clip3(xCtb, \\
 &\quad Min(SubPicRightBoundaryPos, xCtb + (1 \ll CtbLog2SizeY) + 3), (8-561) \\
 &\quad xColCtrCb + (tempMv[0] \gg 4)) \\
 - \quad & \text{Otherwise (subpic_treated_as_pic_flag[SubPicIdx] is equal to 0, the following applies:} \\
 xColCb &= Clip3(xCtb, \\
 &\quad Min(CurPicWidthInSamplesY - 1, xCtb + (1 \ll CtbLog2SizeY) + 3), (8-562) \\
 &\quad xColCtrCb + (tempMv[0] \gg 4))
 \end{aligned}$$

...

Luma sample interpolation filtering process

Inputs to this process are:

- a luma location in full-sample units ($xInt_L, yInt_L$),
- a luma location in fractional-sample units ($xFrac_L, yFrac_L$),
- a luma location in full-sample units ($xSbInt_L, ySbInt_L$) specifying the top-left sample of the bounding block for reference sample padding relative to the top-left luma sample of the reference picture,
- the luma reference sample array $refPicXL$,
- the half sample interpolation filter index $hpelIdx$,
- a variable $sbWidth$ specifying the width of the current subblock,
- a variable $sbHeight$ specifying the height of the current subblock,
- a luma location (xSb, ySb) specifying the top-left sample of the current subblock relative to the top-left luma sample of the current picture,

Output of this process is a predicted luma sample value $predSampleLXL$.

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

- The variable $shift1$ is set equal to $Min(4, BitDepth_Y - 8)$, the variable $shift2$ is set equal to 6 and the variable $shift3$ is set equal to $Max(2, 14 - BitDepth_Y)$.
- The variable $picW$ is set equal to $pic_width_in_luma_samples$ and the variable $picH$ is set equal to $pic_height_in_luma_samples$.

The luma interpolation filter coefficients $f_L[p]$ for each 1/16 fractional sample position p equal to $xFrac_L$ or $yFrac_L$ are derived as follows:

- If $MotionModelIdx[xSb][ySb]$ is greater than 0, and $sbWidth$ and $sbHeight$ are both equal to 4, the luma interpolation filter coefficients $f_L[p]$ are specified in Table 8-12.
- Otherwise, the luma interpolation filter coefficients $f_L[p]$ are specified in Table 8-11 depending on $hpelIdx$.

The luma locations in full-sample units ($xInt_i, yInt_i$) are derived as follows for $i = 0..7$:

- If $subpic_treated_as_pic_flag[SubPicIdx]$ is equal to 1, the following applies:

$$xInt_i = Clip3(SubPicLeftBoundaryPos, SubPicRightBoundaryPos, xInt_L + i - 3) \quad (8-771)$$

$$yInt_i = Clip3(SubPicTopBoundaryPos, SubPicBotBoundaryPos, yInt_L + i - 3) \quad (8-772)$$

- Otherwise ($subpic_treated_as_pic_flag[SubPicIdx]$ is equal to 0), the following applies:

$$\begin{aligned}
 xInt_i &= \text{Clip3}(0, \text{picW} - 1, \text{sps_ref_wraparound_enabled_flag} ? \\
 &\quad \text{ClipH}((\text{sps_ref_wraparound_offset_minus1} + 1) * \text{MinCbSizeY}, \text{picW}, xInt_L + i - 3) : (8-773) \\
 &\quad xInt_L + i - 3) \\
 yInt_i &= \text{Clip3}(0, \text{picH} - 1, yInt_L + i - 3) \quad (8-774)
 \end{aligned}$$

...

Chroma sample interpolation process

Inputs to this process are:

- a chroma location in full-sample units ($xInt_C$, $yInt_C$),
- a chroma location in 1/32 fractional-sample units ($xFrac_C$, $yFrac_C$),
- a chroma location in full-sample units ($xSbInt_C$, $ySbInt_C$) specifying the top-left sample of the bounding block for reference sample padding relative to the top-left chroma sample of the reference picture,
- a variable $sbWidth$ specifying the width of the current subblock,
- a variable $sbHeight$ specifying the height of the current subblock,
- the chroma reference sample array $refPicLXC$.

Output of this process is a predicted chroma sample value $predSampleLXC$ The variables $shift1$, $shift2$ and $shift3$ are derived as follows:

- The variable $shift1$ is set equal to $\text{Min}(4, \text{BitDepth}_C - 8)$, the variable $shift2$ is set equal to 6 and the variable $shift3$ is set equal to $\text{Max}(2, 14 - \text{BitDepth}_C)$.
- The variable $picW_C$ is set equal to $\text{pic_width_in_luma_samples} / \text{SubWidthC}$ and the variable $picH_C$ is set equal to $\text{pic_height_in_luma_samples} / \text{SubHeightC}$.

The chroma interpolation filter coefficients $f_C[p]$ for each 1/32 fractional sample position p equal to $xFrac_C$ or $yFrac_C$ are specified in Table 8-13.

The variable $xOffset$ is set equal to $(\text{sps_ref_wraparound_offset_minus1} + 1) * \text{MinCbSizeY} / \text{SubWidthC}$.

The chroma locations in full-sample units ($xInt_i$, $yInt_i$) are derived as follows for $i = 0..3$:

- If $\text{subpic_treated_as_pic_flag}[\text{SubPicIdx}]$ is equal to 1, the following applies:

$$\begin{aligned}
 xInt_i &= \text{Clip3}(\text{SubPicLeftBoundaryPos} / \text{SubWidthC}, \text{SubPicRightBoundaryPos} / \text{SubWidthC}, xInt_L + i) \quad (8-785) \\
 yInt_i &= \text{Clip3}(\text{SubPicTopBoundaryPos} / \text{SubHeightC}, \text{SubPicBotBoundaryPos} / \text{SubHeightC}, yInt_L + i) \quad (8-786)
 \end{aligned}$$
- Otherwise ($\text{subpic_treated_as_pic_flag}[\text{SubPicIdx}]$ is equal to 0), the following applies:

$$\begin{aligned}
 xInt_i &= \text{Clip3}(0, \text{picW}_C - 1, \text{sps_ref_wraparound_enabled_flag} ? \text{ClipH}(xOffset, \text{picW}_C, xInt_C + i - 1) : (8-787) \\
 &\quad xInt_C + i - 1) \\
 yInt_i &= \text{Clip3}(0, \text{picH}_C - 1, yInt_C + i - 1) \quad (8-788)
 \end{aligned}$$

[0059] 2.4 Example Encoder-only GOP-based temporal filter

[0060] In some embodiments, an encoder-only temporal filter can be implemented. The filtering is done at the encoder side as a pre-processing step. Source pictures before and after the selected

picture to encode are read and a block based motion compensation method relative to the selected picture is applied on those source pictures. Samples in the selected picture are temporally filtered using sample values after motion compensation.

[0061] The overall filter strength is set depending on the temporal sub layer of the selected picture as well as the QP. Only pictures at temporal sub layers 0 and 1 are filtered and pictures of layer 0 are filter by a stronger filter than pictures of layer 1. The per sample filter strength is adjusted depending on the difference between the sample value in the selected picture and the co-located samples in motion compensated pictures so that small differences between a motion compensated picture and the selected picture are filtered more strongly than larger differences.

[0062] GOP Based Temporal Filter

[0063] A temporal filter is introduced directly after reading picture and before encoding. Below are the steps described in more detail.

[0064] Operation 1: Pictures are read by the encoder

[0065] Operation 2: If a picture is low enough in the coding hierarchy, it is filtered before encoding. Otherwise the picture is encoded without filtering. RA pictures with $\text{POC \% } 8 = 0$ are filtered as well as LD pictures with $\text{POC \% } 4 = 0$. AI pictures are never filtered.

[0066] The overall filter strength, s_o , is set according to the equation below for RA.

$$s_o(n) = \begin{cases} 1.5, & n \bmod 16 = 0 \\ 0.95, & n \bmod 16 \neq 0 \end{cases}$$

[0067] where n is the number of pictures read.

[0068] For the LD case, $s_o(n) = 0.95$ is used.

[0069] Operation 3: Two pictures before and/or after the selected picture (referred to as original picture further down) are read. In the edge cases e.g. if is the first picture or close to the last picture, only the available pictures are read.

[0070] Operation 4: Motion of the read pictures before and after, relative to the original picture is estimated per 8x8 picture block.

[0071] A hierarchical motion estimation scheme is used and the layers L0, L1 and L2, are illustrated in FIG. 2. Subsampled pictures are generated by averaging each 2x2 block for all read pictures and the original picture, e.g. L1 in FIG. 1. L2 is derived from L1 using the same subsampling method.

[0072] FIG. 2 shows examples of different layers of the hierarchical motion estimation. L0 is the original resolution. L1 is a subsampled version of L0. L2 is a subsampled version of L1.

[0073] First, motion estimation is done for each 16x16 block in L2. The squared difference is calculated for each selected motion vector and the motion vector corresponding to the smallest difference is selected. The selected motion vector is then used as initial value when estimating the motion in L1. Then the same is done for estimating motion in L0. As a final step, subpixel motion is estimated for each 8x8 block by using an interpolation filter on L0.

[0074] The VTM 6-tap interpolation filter can be used:

0:	0,	0,	64,	0,	0,	0
1:	1,	-3,	64,	4,	-2,	0
2:	1,	-6,	62,	9,	-3,	1
3:	2,	-8,	60,	14,	-5,	1
4:	2,	-9,	57,	19,	-7,	2
5:	3,	-10,	53,	24,	-8,	2
6:	3,	-11,	50,	29,	-9,	2
7:	3,	-11,	44,	35,	-10,	3
8:	1,	-7,	38,	38,	-7,	1
9:	3,	-10,	35,	44,	-11,	3
10:	2,	-9,	29,	50,	-11,	3
11:	2,	-8,	24,	53,	-10,	3
12:	2,	-7,	19,	57,	-9,	2
13:	1,	-5,	14,	60,	-8,	2
14:	1,	-3,	9,	62,	-6,	1
15:	0,	-2,	4,	64,	-3,	1

[0075] Operation 5: Motion compensation is applied on the pictures before and after the original picture according to the best matching motion for each block, e.g., so that the sample coordinates of the original picture in each block have the best matching coordinates in the referenced pictures.

[0076] Operation 6: The samples of the processed one by one for the luma and chroma channels as described in the following steps.

[0077] Operation 7: The new sample value, I_n , is calculated using the following formula.

$$I_n = \frac{I_o + \sum_{i=0}^3 w_r(i, a) I_r(i)}{1 + \sum_{i=0}^3 w_r(i, a)}$$

[0078] Where I_o is the sample value of the original sample, $I_r(i)$ is the intensity of the corresponding sample of motion compensated picture i and $w_r(i, a)$ is the weight of motion compensated picture i when the number of available motion compensated pictures is a .

[0079] In the luma channel, the weights, $w_r(i, a)$, is defined as follows:

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

[0080] Where

$$\begin{aligned} s_l &= 0.4 \\ s_r(i, 2) &= \begin{cases} 1.2, & i = 0 \\ 1.0, & i = 1 \end{cases} \\ s_r(i, 4) &= \begin{cases} 0.60, & i = 0 \\ 0.85, & i = 1 \\ 0.85, & i = 2 \\ 0.60, & i = 3 \end{cases} \end{aligned}$$

[0081] For all other cases of i , and a : $s_r(i, a) = 0.3$

$$\sigma_l(QP) = 3 * (QP - 10)$$

$$\Delta I(i) = I_r(i) - I_o$$

[0082] For the chroma channels, the weights, $w_r(i, a)$, is defined as follows:

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

[0083] Where $s_c = 0.55$ and $\sigma_c = 30$

[0084] Operation 8: The filter is applied for the current sample. The resulting sample value is stored separately.

[0085] Operation 9: The filtered picture is encoded.

[0086] 2.5 Example Picture partitions (tiles, bricks, slices)

[0087] In some embodiments, a picture is divided into one or more tile rows and one or more tile columns. A tile is a sequence of CTUs that covers a rectangular region of a picture.

[0088] A tile is divided into one or more bricks, each of which consisting of a number of CTU rows within the tile.

[0089] A tile that is not partitioned into multiple bricks is also referred to as a brick. However, a brick that is a true subset of a tile is not referred to as a tile.

[0090] A slice either contains a number of tiles of a picture or a number of bricks of a tile.

[0091] A sub-picture contains one or more slices that collectively cover a rectangular region of a picture.

[0092] Two modes of slices are supported, namely the raster-scan slice mode and the rectangular slice mode. In the raster-scan slice mode, a slice contains a sequence of tiles in a tile raster scan of a picture. In the rectangular slice mode, a slice contains a number of bricks of a picture that collectively form a rectangular region of the picture. The bricks within a rectangular slice are in the order of brick raster scan of the slice.

[0093] FIG. 5 shows an example of raster-scan slice partitioning of a picture, where the picture is divided into 12 tiles and 3 raster-scan slices.

[0094] FIG. 6 shows an example of rectangular slice partitioning of a picture, where the picture is divided into 24 tiles (6 tile columns and 4 tile rows) and 9 rectangular slices.

[0095] FIG. 7 shows an example of a picture partitioned into tiles, bricks, and rectangular slices, where the picture is divided into 4 tiles (2 tile columns and 2 tile rows), 11 bricks (the top-left tile contains 1 brick, the top-right tile contains 5 bricks, the bottom-left tile contains 2 bricks, and the bottom-right tile contain 3 bricks), and 4 rectangular slices.

Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	Descriptor
...	
single_tile_in_pic_flag	u(1)
if(!single_tile_in_pic_flag) {	
uniform_tile_spacing_flag	u(1)
if(uniform_tile_spacing_flag) {	
tile_cols_width_minus1	ue(v)
tile_rows_height_minus1	ue(v)
} else {	
num_tile_columns_minus1	ue(v)
num_tile_rows_minus1	ue(v)
for(i = 0; i < num_tile_columns_minus1; i++)	

tile_column_width_minus1[i]	ue(v)
for(i = 0; i < num_tile_rows_minus1; i++)	
tile_row_height_minus1[i]	ue(v)
}	
brick_splitting_present_flag	u(1)
if(uniform_tile_spacing_flag && brick_splitting_present_flag)	
num_tiles_in_pic_minus1	ue(v)
for(i = 0; brick_splitting_present_flag && i <= num_tiles_in_pic_minus1 + 1; i++) {	
if(RowHeight[i] > 1)	
brick_split_flag[i]	u(1)
if(brick_split_flag[i]) {	
if(RowHeight[i] > 2)	
uniform_brick_spacing_flag[i]	u(1)
if(uniform_brick_spacing_flag[i])	
brick_height_minus1[i]	ue(v)
else {	
num_brick_rows_minus2[i]	ue(v)
for(j = 0; j <= num_brick_rows_minus2[i]; j++)	
brick_row_height_minus1[i][j]	ue(v)
}	
}	
}	
single_brick_per_slice_flag	u(1)
if(!single_brick_per_slice_flag)	
rect_slice_flag	u(1)
if(rect_slice_flag && !single_brick_per_slice_flag) {	
num_slices_in_pic_minus1	ue(v)
bottom_right_brick_idx_length_minus1	ue(v)
for(i = 0; i < num_slices_in_pic_minus1; i++) {	
bottom_right_brick_idx_delta[i]	u(v)
brick_idx_delta_sign_flag[i]	u(1)
}	
}	
loop_filter_across_bricks_enabled_flag	u(1)
if(loop_filter_across_bricks_enabled_flag)	
loop_filter_across_slices_enabled_flag	u(1)

}	
if(rect_slice_flag) {	
signalled_slice_id_flag	u(1)
if(signalled_slice_id_flag) {	
signalled_slice_id_length_minus1	ue(v)
for(i = 0; i <= num_slices_in_pic_minus1; i++)	
slice_id[i]	u(v)
}	
}	
...	

slice_header() {	Descriptor
slice_pic_parameter_set_id	ue(v)
if(rect_slice_flag NumBricksInPic > 1)	
slice_address	u(v)
if(!rect_slice_flag && !single_brick_per_slice_flag)	
num_bricks_in_slice_minus1	ue(v)
non_reference_picture_flag	u(1)
slice_type	ue(v)
...	

single_tile_in_pic_flag equal to 1 specifies that there is only one tile in each picture referring to the PPS.

single_tile_in_pic_flag equal to 0 specifies that there is more than one tile in each picture referring to the PPS.

NOTE – In absence of further brick splitting within a tile, the whole tile is referred to as a brick. When a picture contains only a single tile without further brick splitting, it is referred to as a single brick.

It is a requirement of bitstream conformance that the value of **single_tile_in_pic_flag** shall be the same for all PPSs that are referred to by coded pictures within a CVS.

uniform_tile_spacing_flag equal to 1 specifies that tile column boundaries and likewise tile row boundaries are distributed uniformly across the picture and signalled using the syntax elements **tile_cols_width_minus1** and **tile_rows_height_minus1**. **uniform_tile_spacing_flag** equal to 0 specifies that tile column boundaries and likewise tile row boundaries may or may not be distributed uniformly across the picture and signalled using the syntax elements **num_tile_columns_minus1** and **num_tile_rows_minus1** and a list of syntax element pairs **tile_column_width_minus1[i]** and **tile_row_height_minus1[i]**. When not present, the value of **uniform_tile_spacing_flag** is inferred to be equal to 1.

tile_cols_width_minus1 plus 1 specifies the width of the tile columns excluding the right-most tile column of the picture in units of CTBs when **uniform_tile_spacing_flag** is equal to 1. The value of **tile_cols_width_minus1** shall be

in the range of 0 to $\text{PicWidthInCtbsY} - 1$, inclusive. When not present, the value of $\text{tile_cols_width_minus1}$ is inferred to be equal to $\text{PicWidthInCtbsY} - 1$.

tile_rows_height_minus1 plus 1 specifies the height of the tile rows excluding the bottom tile row of the picture in units of CTBs when $\text{uniform_tile_spacing_flag}$ is equal to 1. The value of $\text{tile_rows_height_minus1}$ shall be in the range of 0 to $\text{PicHeightInCtbsY} - 1$, inclusive. When not present, the value of $\text{tile_rows_height_minus1}$ is inferred to be equal to $\text{PicHeightInCtbsY} - 1$.

num_tile_columns_minus1 plus 1 specifies the number of tile columns partitioning the picture when $\text{uniform_tile_spacing_flag}$ is equal to 0. The value of $\text{num_tile_columns_minus1}$ shall be in the range of 0 to $\text{PicWidthInCtbsY} - 1$, inclusive. If $\text{single_tile_in_pic_flag}$ is equal to 1, the value of $\text{num_tile_columns_minus1}$ is inferred to be equal to 0. Otherwise, when $\text{uniform_tile_spacing_flag}$ is equal to 1, the value of $\text{num_tile_columns_minus1}$ is inferred as specified in clause 6.5.1.

num_tile_rows_minus1 plus 1 specifies the number of tile rows partitioning the picture when $\text{uniform_tile_spacing_flag}$ is equal to 0. The value of $\text{num_tile_rows_minus1}$ shall be in the range of 0 to $\text{PicHeightInCtbsY} - 1$, inclusive. If $\text{single_tile_in_pic_flag}$ is equal to 1, the value of $\text{num_tile_rows_minus1}$ is inferred to be equal to 0. Otherwise, when $\text{uniform_tile_spacing_flag}$ is equal to 1, the value of $\text{num_tile_rows_minus1}$ is inferred as specified in clause 6.5.1.

The variable NumTilesInPic is set equal to $(\text{num_tile_columns_minus1} + 1) * (\text{num_tile_rows_minus1} + 1)$. When $\text{single_tile_in_pic_flag}$ is equal to 0, NumTilesInPic shall be greater than 1.

tile_column_width_minus1[i] plus 1 specifies the width of the i -th tile column in units of CTBs.

tile_row_height_minus1[i] plus 1 specifies the height of the i -th tile row in units of CTBs.

brick_splitting_present_flag equal to 1 specifies that one or more tiles of pictures referring to the PPS may be divided into two or more bricks. **brick_splitting_present_flag** equal to 0 specifies that no tiles of pictures referring to the PPS are divided into two or more bricks.

num_tiles_in_pic_minus1 plus 1 specifies the number of tiles in each picture referring to the PPS. The value of $\text{num_tiles_in_pic_minus1}$ shall be equal to $\text{NumTilesInPic} - 1$. When not present, the value of $\text{num_tiles_in_pic_minus1}$ is inferred to be equal to $\text{NumTilesInPic} - 1$.

brick_split_flag[i] equal to 1 specifies that the i -th tile is divided into two or more bricks. **brick_split_flag**[i] equal to 0 specifies that the i -th tile is not divided into two or more bricks. When not present, the value of **brick_split_flag**[i] is inferred to be equal to 0. In some embodiments, PPS parsing dependency on SPS is introduced by adding the syntax condition "if($\text{RowHeight}[i] > 1$)" (e.g., similarly for $\text{uniform_brick_spacing_flag}[i]$).

uniform_brick_spacing_flag[i] equal to 1 specifies that horizontal brick boundaries are distributed uniformly across the i -th tile and signalled using the syntax element **brick_height_minus1**[i]. **uniform_brick_spacing_flag**[i] equal to 0 specifies that horizontal brick boundaries may or may not be distributed uniformly across i -th tile and signalled using the syntax element **num_brick_rows_minus2**[i] and a list of syntax elements **brick_row_height_minus1**[i][j]. When not present, the value of **uniform_brick_spacing_flag**[i] is inferred to be equal to 1.

brick_height_minus1[*i*] plus 1 specifies the height of the brick rows excluding the bottom brick in the *i*-th tile in units of CTBs when **uniform_brick_spacing_flag**[*i*] is equal to 1. When present, the value of **brick_height_minus1** shall be in the range of 0 to **RowHeight**[*i*] - 2, inclusive. When not present, the value of **brick_height_minus1**[*i*] is inferred to be equal to **RowHeight**[*i*] - 1.

num_brick_rows_minus2[*i*] plus 2 specifies the number of bricks partitioning the *i*-th tile when **uniform_brick_spacing_flag**[*i*] is equal to 0. When present, the value of **num_brick_rows_minus2**[*i*] shall be in the range of 0 to **RowHeight**[*i*] - 2, inclusive. If **brick_split_flag**[*i*] is equal to 0, the value of **num_brick_rows_minus2**[*i*] is inferred to be equal to -1. Otherwise, when **uniform_brick_spacing_flag**[*i*] is equal to 1, the value of **num_brick_rows_minus2**[*i*] is inferred as specified in 6.5.1.

brick_row_height_minus1[*i*][*j*] plus 1 specifies the height of the *j*-th brick in the *i*-th tile in units of CTBs when **uniform_tile_spacing_flag** is equal to 0.

The following variables are derived, and, when **uniform_tile_spacing_flag** is equal to 1, the values of **num_tile_columns_minus1** and **num_tile_rows_minus1** are inferred, and, for each *i* ranging from 0 to **NumTilesInPic** - 1, inclusive, when **uniform_brick_spacing_flag**[*i*] is equal to 1, the value of **num_brick_rows_minus2**[*i*] is inferred, by invoking the CTB raster and brick scanning conversion process as specified in clause 6.5.1:

- the list **RowHeight**[*j*] for *j* ranging from 0 to **num_tile_rows_minus1**, inclusive, specifying the height of the *j*-th tile row in units of CTBs,
- the list **CtbAddrRsToBs**[*ctbAddrRs*] for *ctbAddrRs* ranging from 0 to **PicSizeInCtbsY** - 1, inclusive, specifying the conversion from a CTB address in the CTB raster scan of a picture to a CTB address in the brick scan,
- the list **CtbAddrBsToRs**[*ctbAddrBs*] for *ctbAddrBs* ranging from 0 to **PicSizeInCtbsY** - 1, inclusive, specifying the conversion from a CTB address in the brick scan to a CTB address in the CTB raster scan of a picture,
- the list **BrickId**[*ctbAddrBs*] for *ctbAddrBs* ranging from 0 to **PicSizeInCtbsY** - 1, inclusive, specifying the conversion from a CTB address in brick scan to a brick ID,
- the list **NumCtusInBrick**[*brickIdx*] for *brickIdx* ranging from 0 to **NumBricksInPic** - 1, inclusive, specifying the conversion from a brick index to the number of CTUs in the brick,
- the list **FirstCtbAddrBs**[*brickIdx*] for *brickIdx* ranging from 0 to **NumBricksInPic** - 1, inclusive, specifying the conversion from a brick ID to the CTB address in brick scan of the first CTB in the brick.

single_brick_per_slice_flag equal to 1 specifies that each slice that refers to this PPS includes one brick.

single_brick_per_slice_flag equal to 0 specifies that a slice that refers to this PPS may include more than one brick.

When not present, the value of **single_brick_per_slice_flag** is inferred to be equal to 1.

rect_slice_flag equal to 0 specifies that bricks within each slice are in raster scan order and the slice information is not signalled in PPS. **rect_slice_flag** equal to 1 specifies that bricks within each slice cover a rectangular region of the picture and the slice information is signalled in the PPS. When **brick_splitting_present_flag** is equal to 1, the value of **rect_slice_flag** shall be equal to 1. When not present, **rect_slice_flag** is inferred to be equal to 1.

num_slices_in_pic_minus1 plus 1 specifies the number of slices in each picture referring to the PPS. The value of **num_slices_in_pic_minus1** shall be in the range of 0 to **NumBricksInPic** - 1, inclusive. When not present and

single_brick_per_slice_flag is equal to 1, the value of num_slices_in_pic_minus1 is inferred to be equal to NumBricksInPic - 1.

bottom_right_brick_idx_length_minus1 plus 1 specifies the number of bits used to represent the syntax element bottom_right_brick_idx_delta[i]. The value of bottom_right_brick_idx_length_minus1 shall be in the range of 0 to Ceil(Log2(NumBricksInPic)) - 1, inclusive.

bottom_right_brick_idx_delta[i] when i is greater than 0 specifies the difference between the brick index of the brick located at the bottom-right corner of the i-th slice and the brick index of the bottom-right corner of the (i - 1)-th slice. bottom_right_brick_idx_delta[0] specifies the brick index of the bottom right corner of the 0-th slice. When single_brick_per_slice_flag is equal to 1, the value of bottom_right_brick_idx_delta[i] is inferred to be equal to 1. The value of the BottomRightBrickIdx[num_slices_in_pic_minus1] is inferred to be equal to NumBricksInPic - 1. The length of the bottom_right_brick_idx_delta[i] syntax element is bottom_right_brick_idx_length_minus1 + 1 bits.

brick_idx_delta_sign_flag[i] equal to 1 indicates a positive sign for bottom_right_brick_idx_delta[i]. sign_bottom_right_brick_idx_delta[i] equal to 0 indicates a negative sign for bottom_right_brick_idx_delta[i]. It is a requirement of bitstream conformance that a slice shall include either a number of complete tiles or only a consecutive sequence of complete bricks of one tile.

The variable TopLeftBrickIdx[i], BottomRightBrickIdx[i], NumBricksInSlice[i] and BricksToSliceMap[j], which specify the brick index of the brick located at the top left corner of the i-th slice, the brick index of the brick located at the bottom right corner of the i-th slice, the number of bricks in the i-th slice and the mapping of bricks to slices, are derived as follows:

```

for( j = 0; i == 0 && j < NumBricksInPic; j++ )
    BricksToSliceMap[ j ] = -1
NumBricksInSlice[ i ] = 0
BottomRightBrickIdx[ i ] = bottom_right_brick_idx_delta[ i ] + ( ( i == 0 ) ? 0 :
    ( brick_idx_delta_sign_flag[ i ] ? BottomRightBrickIdx[ i - 1 ] :
    -BottomRightBrickIdx[ i - 1 ] ) )
for( j = BottomRightBrickIdx[ i ]; j >= 0; j-- ) {
    if( BrickColBd[ j ] <= BrickColBd[ BottomRightBrickIdx[ i ] ] &&
        BrickRowBd[ j ] <= BrickRowBd[ BottomRightBrickIdx[ i ] ] &&
        BricksToSliceMap[ j ] == -1 ) {
        TopLeftBrickIdx[ i ] = j
        NumBricksInSlice[ i ]++
        BricksToSliceMap[ j ] = i
    }
}

```

General slice header semantics

When present, the value of each of the slice header syntax elements `slice_pic_parameter_set_id`, `non_reference_picture_flag`, `colour_plane_id`, `slice_pic_order_cnt_lsb`, `recovery_poc_cnt`, `no_output_of_prior_pics_flag`, `pic_output_flag`, and `slice_temporal_mvp_enabled_flag` shall be the same in all slice headers of a coded picture.

The variable `CuQpDeltaVal`, specifying the difference between a luma quantization parameter for the coding unit containing `cu_qp_delta_abs` and its prediction, is set equal to 0. The variables `CuQpOffsetCb`, `CuQpOffsetCr`, and `CuQpOffsetCbCr`, specifying values to be used when determining the respective values of the Qp'_{Cb} , Qp'_{Cr} , and Qp'_{CbCr} quantization parameters for the coding unit containing `cu_chroma_qp_offset_flag`, are all set equal to 0.

`slice_pic_parameter_set_id` specifies the value of `pps_pic_parameter_set_id` for the PPS in use. The value of `slice_pic_parameter_set_id` shall be in the range of 0 to 63, inclusive.

It is a requirement of bitstream conformance that the value of `TemporalId` of the current picture shall be greater than or equal to the value of `TemporalId` of the PPS that has `pps_pic_parameter_set_id` equal to `slice_pic_parameter_set_id`.

`slice_address` specifies the slice address of the slice. When not present, the value of `slice_address` is inferred to be equal to 0.

If `rect_slice_flag` is equal to 0, the following applies:

- The slice address is the brick ID as specified by Equation (7-59).
- The length of `slice_address` is $\text{Ceil}(\text{Log}_2(\text{NumBricksInPic}))$ bits.
- The value of `slice_address` shall be in the range of 0 to $\text{NumBricksInPic} - 1$, inclusive.

Otherwise (`rect_slice_flag` is equal to 1), the following applies:

- The slice address is the slice ID of the slice.
- The length of `slice_address` is `signalled_slice_id_length_minus1 + 1` bits.
- If `signalled_slice_id_flag` is equal to 0, the value of `slice_address` shall be in the range of 0 to `num_slices_in_pic_minus1`, inclusive. Otherwise, the value of `slice_address` shall be in the range of 0 to $2^{(\text{signalled_slice_id_length_minus1} + 1)} - 1$, inclusive.

It is a requirement of bitstream conformance that the following constraints apply:

- The value of `slice_address` shall not be equal to the value of `slice_address` of any other coded slice NAL unit of the same coded picture.
- When `rect_slice_flag` is equal to 0, the slices of a picture shall be in increasing order of their `slice_address` values.
- The shapes of the slices of a picture shall be such that each brick, when decoded, shall have its entire left boundary and entire top boundary consisting of a picture boundary or consisting of boundaries of previously decoded brick(s).

`num_bricks_in_slice_minus1`, when present, specifies the number of bricks in the slice minus 1. The value of `num_bricks_in_slice_minus1` shall be in the range of 0 to $\text{NumBricksInPic} - 1$, inclusive. When `rect_slice_flag` is equal to 0 and `single_brick_per_slice_flag` is equal to 1, the value of `num_bricks_in_slice_minus1` is inferred to be

equal to 0. When `single_brick_per_slice_flag` is equal to 1, the value of `num_bricks_in_slice_minus1` is inferred to be equal to 0.

The variable `NumBricksInCurrSlice`, which specifies the number of bricks in the current slice, and `SliceBrickIdx[ i ]`, which specifies the brick index of the *i*-th brick in the current slice, are derived as follows:

```

if( rect_slice_flag ) {
    sliceIdx = 0
    while( slice_address != slice_id[ sliceIdx ] )
        sliceIdx++
    NumBricksInCurrSlice = NumBricksInSlice[ sliceIdx ]
    brickIdx = TopLeftBrickIdx[ sliceIdx ]
    for( bIdx = 0; brickIdx <= BottomRightBrickIdx[ sliceIdx ]; brickIdx++ )
        if( BricksToSliceMap[ brickIdx ] == sliceIdx )
            SliceBrickIdx[ bIdx++ ] = brickIdx
} else {
    NumBricksInCurrSlice = num_bricks_in_slice_minus1 + 1
    SliceBrickIdx[ 0 ] = slice_address
    for( i = 1; i < NumBricksInCurrSlice; i++ )
        SliceBrickIdx[ i ] = SliceBrickIdx[ i - 1 ] + 1
}

```

(7-92)

The variables `SubPicIdx`, `SubPicLeftBoundaryPos`, `SubPicTopBoundaryPos`, `SubPicRightBoundaryPos`, and `SubPicBotBoundaryPos` are derived as follows:

```

SubPicIdx = CtbToSubPicIdx[ CtbAddrBsToRs[ FirstCtbAddrBs[ SliceBrickIdx[ 0 ] ] ] ]
if( subpic_treated_as_pic_flag[ SubPicIdx ] ) {
    SubPicLeftBoundaryPos = SubPicLeft[ SubPicIdx ] * ( subpic_grid_col_width_minus1 + 1 ) * 4
    SubPicRightBoundaryPos = ( SubPicLeft[ SubPicIdx ] + SubPicWidth[ SubPicIdx ] ) *
        ( subpic_grid_col_width_minus1 + 1 ) * 4
    SubPicTopBoundaryPos = SubPicTop[ SubPicIdx ] * ( subpic_grid_row_height_minus1 + 1 ) * 4
    SubPicBotBoundaryPos = ( SubPicTop[ SubPicIdx ] + SubPicHeight[ SubPicIdx ] ) *
        ( subpic_grid_row_height_minus1 + 1 ) * 4
}

```

(7-93)

[0096] 2.6 Example Syntax and Semantics

Sequence parameter set RBSP syntax

<code>seq_parameter_set_rbsp() {</code>	Descriptor
<code>sps_decoding_parameter_set_id</code>	u(4)
<code>sps_video_parameter_set_id</code>	u(4)
<code>sps_max_sub_layers_minus1</code>	u(3)
<code>sps_reserved_zero_5bits</code>	u(5)

profile_tier_level(sps_max_sub_layers_minus1)	
gdr_enabled_flag	u(1)
sps_seq_parameter_set_id	u(4)
chroma_format_idc	u(2)
if(chroma_format_idc == 3)	
separate_colour_plane_flag	u(1)
ref_pic_resampling_enabled_flag	u(1)
sps_seq_parameter_set_id	ue(v)
chroma_format_idc	ue(v)
if(chroma_format_idc == 3)	
separate_colour_plane_flag	u(1)
pic_width_max_in_luma_samples	ue(v)
pic_height_max_in_luma_samples	ue(v)
sps_log2_ctu_size_minus5	u(2)
<u>subpics_present_flag</u>	<u>u(1)</u>
<u>sps_num_subpics_minus1</u>	<u>u(8)</u>
<u>for(i = 0; i <= sps_num_subpics_minus1; i++) {</u>	
<u> subpic_ctu_top_left_x[i]</u>	<u>u(v)</u>
<u> subpic_ctu_top_left_y[i]</u>	<u>u(v)</u>
<u> subpic_width_minus1[i]</u>	<u>u(v)</u>
<u> subpic_height_minus1[i]</u>	<u>u(v)</u>
<u> subpic_treated_as_pic_flag[i]</u>	<u>u(1)</u>
<u> loop_filter_across_subpic_enabled_flag[i]</u>	<u>u(1)</u>
<u> }</u>	
<u> }</u>	
<u>sps_subpic_id_present_flag</u>	<u>u(1)</u>
<u>if(sps_subpics_id_present_flag) {</u>	
<u> sps_subpic_id_signalling_present_flag</u>	<u>u(1)</u>
<u> if(sps_subpics_id_signalling_present_flag) {</u>	
<u> sps_subpic_id_len_minus1</u>	<u>ue(v)</u>
<u> for(i = 0; i <= sps_num_subpics_minus1; i++)</u>	
<u> sps_subpic_id[i]</u>	<u>u(v)</u>
<u> }</u>	
<u> }</u>	
bit_depth_minus8	ue(v)
min_qp_prime_ts_minus4	ue(v)

sps_weighted_pred_flag	u(1)
sps_weighted_bipred_flag	u(1)
log2_max_pic_order_cnt_lsb_minus4	u(4)
if(sps_max_sub_layers_minus1 > 0)	
sps_sub_layer_ordering_info_present_flag	u(1)
for(i = (sps_sub_layer_ordering_info_present_flag ? 0 : sps_max_sub_layers_minus1); i <= sps_max_sub_layers_minus1; i++) {	
sps_max_dec_pic_buffering_minus1[i]	ue(v)
sps_max_num_reorder_pics[i]	ue(v)
sps_max_latency_increase_plus1[i]	ue(v)
}	
long_term_ref_pics_flag	u(1)
inter_layer_ref_pics_present_flag	u(1)
sps_idr_rpl_present_flag	u(1)
rpl1_same_as_rpl0_flag	u(1)
for(i = 0; i < !rpl1_same_as_rpl0_flag ? 2 : 1; i++) {	
num_ref_pic_lists_in_sps[i]	ue(v)
for(j = 0; j < num_ref_pic_lists_in_sps[i]; j++)	
ref_pic_list_struct(i, j)	
}	
if(ChromaArrayType != 0)	
qtbtt_dual_tree_intra_flag	u(1)
log2_min_luma_coding_block_size_minus2	ue(v)
partition_constraints_override_enabled_flag	u(1)
sps_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
sps_log2_diff_min_qt_min_cb_inter_slice	ue(v)
sps_max_mtt_hierarchy_depth_inter_slice	ue(v)
sps_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)
if(sps_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
sps_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)
sps_log2_diff_max_tt_min_qt_intra_slice_luma	ue(v)
}	
if(sps_max_mtt_hierarchy_depth_inter_slice != 0) {	
sps_log2_diff_max_bt_min_qt_inter_slice	ue(v)
sps_log2_diff_max_tt_min_qt_inter_slice	ue(v)
}	

if(qtbtt_dual_tree_intra_flag) {	
sps_log2_diff_min_qt_min_cb_intra_slice_chroma	ue(v)
sps_max_mtt_hierarchy_depth_intra_slice_chroma	ue(v)
if(sps_max_mtt_hierarchy_depth_intra_slice_chroma != 0) {	
sps_log2_diff_max_bt_min_qt_intra_slice_chroma	ue(v)
sps_log2_diff_max_tt_min_qt_intra_slice_chroma	ue(v)
}	
}	
sps_max_luma_transform_size_64_flag	u(1)
sps_joint_cbr_enabled_flag	u(1)
if(ChromaArrayType != 0) {	
same_qp_table_for_chroma	u(1)
numQpTables = same_qp_table_for_chroma ? 1 : (sps_joint_cbr_enabled_flag ? 3 : 2)	
for(i = 0; i < numQpTables; i++) {	
qp_table_start_minus26[i]	se(v)
num_points_in_qp_table_minus1[i]	ue(v)
for(j = 0; j <= num_points_in_qp_table_minus1[i]; j++) {	
delta_qp_in_val_minus1[i][j]	ue(v)
delta_qp_diff_val[i][j]	ue(v)
}	
}	
}	
sps_sao_enabled_flag	u(1)
sps_alf_enabled_flag	u(1)
sps_transform_skip_enabled_flag	u(1)
if(sps_transform_skip_enabled_flag)	
sps_bdpcm_enabled_flag	u(1)
if(sps_bdpcm_enabled_flag && chroma_format_idc == 3)	
sps_bdpcm_chroma_enabled_flag	u(1)
sps_ref_wraparound_enabled_flag	u(1)
if(sps_ref_wraparound_enabled_flag)	
sps_ref_wraparound_offset_minus1	ue(v)
sps_temporal_mvp_enabled_flag	u(1)
if(sps_temporal_mvp_enabled_flag)	
sps_sbtmvp_enabled_flag	u(1)
sps_amvr_enabled_flag	u(1)

sps_bdof_enabled_flag	u(1)
if(sps_bdof_enabled_flag)	
sps_bdof_pic_present_flag	u(1)
sps_smvd_enabled_flag	u(1)
sps_dmv_r_enabled_flag	u(1)
if(sps_dmv_r_enabled_flag)	
sps_dmv_r_pic_present_flag	u(1)
sps_mmvd_enabled_flag	u(1)
sps_isp_enabled_flag	u(1)
sps_mrl_enabled_flag	u(1)
sps_mip_enabled_flag	u(1)
if(ChromaArrayType != 0)	
sps_cclm_enabled_flag	u(1)
if(sps_cclm_enabled_flag && chroma_format_idc == 1)	
sps_cclm_collocated_chroma_flag	u(1)
sps_mts_enabled_flag	u(1)
if(sps_mts_enabled_flag) {	
sps_explicit_mts_intra_enabled_flag	u(1)
sps_explicit_mts_inter_enabled_flag	u(1)
}	
sps_sbt_enabled_flag	u(1)
sps_affine_enabled_flag	u(1)
if(sps_affine_enabled_flag) {	
sps_affine_type_flag	u(1)
sps_affine_amvr_enabled_flag	u(1)
sps_affine_prof_enabled_flag	u(1)
if(sps_affine_prof_enabled_flag)	
sps_prof_pic_present_flag	u(1)
}	
if(chroma_format_idc == 3) {	
sps_palette_enabled_flag	u(1)
sps_act_enabled_flag	u(1)
}	
sps_bcw_enabled_flag	u(1)
sps_ibc_enabled_flag	u(1)
sps_ciip_enabled_flag	u(1)

if(sps_mmvd_enabled_flag)	
sps_fpel_mmvd_enabled_flag	u(1)
sps_triangle_enabled_flag	u(1)
sps_lmcs_enabled_flag	u(1)
sps_lfnst_enabled_flag	u(1)
sps_ladf_enabled_flag	u(1)
if(sps_ladf_enabled_flag) {	
sps_num_ladf_intervals_minus2	u(2)
sps_ladf_lowest_interval_qp_offset	se(v)
for(i = 0; i < sps_num_ladf_intervals_minus2 + 1; i++) {	
sps_ladf_qp_offset[i]	se(v)
sps_ladf_delta_threshold_minus1[i]	ue(v)
}	
}	
sps_scaling_list_enabled_flag	u(1)
sps_loop_filter_across_virtual_boundaries_disabled_present_flag	u(1)
if(sps_loop_filter_across_virtual_boundaries_disabled_present_flag) {	
sps_num_ver_virtual_boundaries	u(2)
for(i = 0; i < sps_num_ver_virtual_boundaries; i++)	
sps_virtual_boundaries_pos_x[i]	u(13)
sps_num_hor_virtual_boundaries	u(2)
for(i = 0; i < sps_num_hor_virtual_boundaries; i++)	
sps_virtual_boundaries_pos_y[i]	u(13)
}	
general_hrd_parameters_present_flag	u(1)
if(general_hrd_parameters_present_flag) {	
num_units_in_tick	u(32)
time_scale	u(32)
sub_layer_cpb_parameters_present_flag	u(1)
if(sub_layer_cpb_parameters_present_flag)	
general_hrd_parameters(0, sps_max_sub_layers_minus1)	
else	
general_hrd_parameters(sps_max_sub_layers_minus1, sps_max_sub_layers_minus1)	
}	
vui_parameters_present_flag	u(1)
if(vui_parameters_present_flag)	

vui_parameters()	
sps_extension_flag	u(1)
if(sps_extension_flag)	
while(more_rbsp_data())	
sps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

Picture parameter set Rbsp syntax

pic_parameter_set_rbsp() {	Descriptor
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	u(4)
pps_seq_parameter_set_id	ue(v)
pic_width_in_luma_samples	ue(v)
pic_height_in_luma_samples	ue(v)
conformance_window_flag	u(1)
if(conformance_window_flag) {	
conf_win_left_offset	ue(v)
conf_win_right_offset	ue(v)
conf_win_top_offset	ue(v)
conf_win_bottom_offset	ue(v)
}	
scaling_window_flag	u(1)
if(scaling_window_flag) {	
scaling_win_left_offset	ue(v)
scaling_win_right_offset	ue(v)
scaling_win_top_offset	ue(v)
scaling_win_bottom_offset	ue(v)
}	
output_flag_present_flag	u(1)
mixed_nalu_types_in_pic_flag	u(1)
<u>pps_subpic_id_signalling_present_flag</u>	<u>u(1)</u>
<u>if(pps_subpics_id_signalling_present_flag) {</u>	
<u>pps_num_subpics_minus1</u>	<u>ue(v)</u>
<u>pps_subpic_id_len_minus1</u>	<u>ue(v)</u>
<u>for(i = 0; i <= pps_num_subpics_minus1; i++)</u>	

<u>pps_subpic_id[i]</u>	<u>u(v)</u>
<u>i</u>	
no_pic_partition_flag	u(1)
if(!no_pic_partition_flag) {	
pps_log2_ctu_size_minus5	u(2)
num_exp_tile_columns_minus1	ue(v)
num_exp_tile_rows_minus1	ue(v)
for(i = 0; i <= num_exp_tile_columns_minus1; i++)	
tile_column_width_minus1[i]	ue(v)
for(i = 0; i <= num_exp_tile_rows_minus1; i++)	
tile_row_height_minus1[i]	ue(v)
rect_slice_flag	u(1)
if(rect_slice_flag)	
single_slice_per_subpic_flag	u(1)
if(rect_slice_flag && !single_slice_per_subpic_flag) {	
num_slices_in_pic_minus1	ue(v)
tile_idx_delta_present_flag	u(1)
for(i = 0; i < num_slices_in_pic_minus1; i++) {	
slice_width_in_tiles_minus1[i]	ue(v)
slice_height_in_tiles_minus1[i]	ue(v)
if(slice_width_in_tiles_minus1[i] == 0 && slice_height_in_tiles_minus1[i] == 0) {	
num_slices_in_tile_minus1[i]	ue(v)
numSlicesInTileMinus1 = num_slices_in_tile_minus1[i]	
for(j = 0; j < numSlicesInTileMinus1; j++)	
slice_height_in_ctu_minus1[i++]	ue(v)
}	
if(tile_idx_delta_present_flag && i < num_slices_in_pic_minus1)	
tile_idx_delta[i]	se(v)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
loop_filter_across_slices_enabled_flag	u(1)
}	
entropy_coding_sync_enabled_flag	u(1)
if(!no_pic_partition_flag entropy_coding_sync_enabled_flag)	

entry_point_offsets_present_flag	u(1)
cabac_init_present_flag	u(1)
for(i = 0; i < 2; i++)	
num_ref_idx_default_active_minus1[i]	ue(v)
rpl1_idx_present_flag	u(1)
init_qp_minus26	se(v)
<u><i>if(sps_transform_skip_enabled_flag)</i></u>	
<u><i>log2_transform_skip_max_size_minus2</i></u>	<u><i>ue(v)</i></u>
cu_qp_delta_enabled_flag	u(1)
pps_cb_qp_offset	se(v)
pps_cr_qp_offset	se(v)
pps_joint_cbr_qp_offset_present_flag	u(1)
if(pps_joint_cbr_qp_offset_present_flag)	
pps_joint_cbr_qp_offset_value	se(v)
pps_slice_chroma_qp_offsets_present_flag	u(1)
cu_chroma_qp_offset_enabled_flag	u(1)
if(cu_chroma_qp_offset_enabled_flag) {	
chroma_qp_offset_list_len_minus1	ue(v)
for(i = 0; i <= chroma_qp_offset_list_len_minus1; i++) {	
cb_qp_offset_list[i]	se(v)
cr_qp_offset_list[i]	se(v)
if(pps_joint_cbr_qp_offset_present_flag)	
joint_cbr_qp_offset_list[i]	se(v)
}	
}	
pps_weighted_pred_flag	u(1)
pps_weighted_bipred_flag	u(1)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
}	
}	

constant_slice_header_params_enabled_flag	u(1)
if(constant_slice_header_params_enabled_flag) {	
pps_dep_quant_enabled_idc	u(2)
for(i = 0; i < 2; i++)	
pps_ref_pic_list_sps_idc[i]	u(2)
pps_mvd_l1_zero_idc	u(2)
pps_collocated_from_l0_idc	u(2)
pps_six_minus_max_num_merge_cand_plus1	ue(v)
pps_max_num_merge_cand_minus_max_num_triangle_cand_plus1	ue(v)
}	
picture_header_extension_present_flag	u(1)
slice_header_extension_present_flag	u(1)
pps_extension_flag	u(1)
if(pps_extension_flag)	
while(more_rbsp_data())	
pps_extension_data_flag	u(1)
rbsp_trailing_bits()	
}	

Picture header RBSP syntax

picture_header_rbsp() {	Descriptor
non_reference_picture_flag	u(1)
gdr_pic_flag	u(1)
no_output_of_prior_pics_flag	u(1)
if(gdr_pic_flag)	
recovery_poc_cnt	ue(v)
ph_pic_parameter_set_id	ue(v)
<u>if(sps_subpic_id_present_flag && !sps_subpic_id_signalling_flag) {</u>	
<u>ph_subpic_id_signalling_present_flag</u>	<u>u(1)</u>
<u>if(ph_subpics_id_signalling_present_flag) {</u>	
<u>ph_subpic_id_len_minus1</u>	<u>ue(v)</u>
<u>for(i = 0; i <= sps_num_subpics_minus1; i++)</u>	
<u>ph_subpic_id[i]</u>	<u>u(v)</u>
<u>}</u>	
<u>}</u>	
if(!sps_loop_filter_across_virtual_boundaries_disabled_present_flag) {	

ph_loop_filter_across_virtual_boundaries_disabled_present_flag	u(1)
if(ph_loop_filter_across_virtual_boundaries_disabled_present_flag) {	
ph_num_ver_virtual_boundaries	u(2)
for(i = 0; i < ph_num_ver_virtual_boundaries; i++)	
ph_virtual_boundaries_pos_x[i]	u(13)
ph_num_hor_virtual_boundaries	u(2)
for(i = 0; i < ph_num_hor_virtual_boundaries; i++)	
ph_virtual_boundaries_pos_y[i]	u(13)
}	
}	
if(separate_colour_plane_flag == 1)	
colour_plane_id	u(2)
if(output_flag_present_flag)	
pic_output_flag	u(1)
pic_rpl_present_flag	u(1)
if(pic_rpl_present_flag) {	
for(i = 0; i < 2; i++) {	
if(num_ref_pic_lists_in_sps[i] > 0 && !pps_ref_pic_list_sps_idc[i] && (i == 0 (i == 1 && rpl1_idx_present_flag)))	
pic_rpl_sps_flag[i]	u(1)
if(pic_rpl_sps_flag[i]) {	
if(num_ref_pic_lists_in_sps[i] > 1 && (i == 0 (i == 1 && rpl1_idx_present_flag)))	
pic_rpl_idx[i]	u(v)
} else	
ref_pic_list_struct(i, num_ref_pic_lists_in_sps[i])	
for(j = 0; j < NumLtrpEntries[i][RplIdx[i]]; j++) {	
if(ltrp_in_slice_header_flag[i][RplIdx[i]])	
pic_poc_lsb_lt[i][j]	u(v)
pic_delta_poc_msb_present_flag[i][j]	u(1)
if(pic_delta_poc_msb_present_flag[i][j])	
pic_delta_poc_msb_cycle_lt[i][j]	ue(v)
}	
}	
}	
if(partition_constraints_override_enabled_flag) {	

partition_constraints_override_flag	ue(v)
if(partition_constraints_override_flag) {	
pic_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
pic_log2_diff_min_qt_min_cb_inter_slice	ue(v)
pic_max_mtt_hierarchy_depth_inter_slice	ue(v)
pic_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)
if(pic_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
pic_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)
pic_log2_diff_max_tt_min_qt_intra_slice_luma	ue(v)
}	
if(pic_max_mtt_hierarchy_depth_inter_slice != 0) {	
pic_log2_diff_max_bt_min_qt_inter_slice	ue(v)
pic_log2_diff_max_tt_min_qt_inter_slice	ue(v)
}	
if(qtbtt_dual_tree_intra_flag) {	
pic_log2_diff_min_qt_min_cb_intra_slice_chroma	ue(v)
pic_max_mtt_hierarchy_depth_intra_slice_chroma	ue(v)
if(pic_max_mtt_hierarchy_depth_intra_slice_chroma != 0) {	
pic_log2_diff_max_bt_min_qt_intra_slice_chroma	ue(v)
pic_log2_diff_max_tt_min_qt_intra_slice_chroma	ue(v)
}	
}	
}	
if(cu_qp_delta_enabled_flag) {	
pic_cu_qp_delta_subdiv_intra_slice	ue(v)
pic_cu_qp_delta_subdiv_inter_slice	ue(v)
}	
if(cu_chroma_qp_offset_enabled_flag) {	
pic_cu_chroma_qp_offset_subdiv_intra_slice	ue(v)
pic_cu_chroma_qp_offset_subdiv_inter_slice	ue(v)
}	
if(sps_temporal_mvp_enabled_flag)	
pic_temporal_mvp_enabled_flag	u(1)
if(!pps_mvd_l1_zero_idc)	
mvd_l1_zero_flag	u(1)

if(!pps_six_minus_max_num_merge_cand_plus1)	
pic_six_minus_max_num_merge_cand	ue(v)
if(sps_affine_enabled_flag)	
pic_five_minus_max_num_subblock_merge_cand	ue(v)
if(sps_fpel_mmvd_enabled_flag)	
pic_fpel_mmvd_enabled_flag	u(1)
if(sps_bdof_pic_present_flag)	
pic_disable_bdof_flag	u(1)
if(sps_dmvr_pic_present_flag)	
pic_disable_dmvr_flag	u(1)
if(sps_prof_pic_present_flag)	
pic_disable_prof_flag	u(1)
if(sps_triangle_enabled_flag && MaxNumMergeCand >= 2 && !pps_max_num_merge_cand_minus_max_num_triangle_cand_minus1)	
pic_max_num_merge_cand_minus_max_num_triangle_cand	ue(v)
if (sps_ibc_enabled_flag)	
pic_six_minus_max_num_ibc_merge_cand	ue(v)
if(sps_joint_cbr_enabled_flag)	
pic_joint_cbr_sign_flag	u(1)
if(sps_sao_enabled_flag) {	
pic_sao_enabled_present_flag	u(1)
if(pic_sao_enabled_present_flag) {	
pic_sao_luma_enabled_flag	u(1)
if(ChromaArrayType != 0)	
pic_sao_chroma_enabled_flag	u(1)
}	
}	
if(sps_alf_enabled_flag) {	
pic_alf_enabled_present_flag	u(1)
if(pic_alf_enabled_present_flag) {	
pic_alf_enabled_flag	u(1)
if(pic_alf_enabled_flag) {	
pic_num_alf_aps_ids_luma	u(3)
for(i = 0; i < pic_num_alf_aps_ids_luma; i++)	
pic_alf_aps_id_luma[i]	u(3)
if(ChromaArrayType != 0)	

pic_alf_chroma_idc	u(2)
if(pic_alf_chroma_idc)	
pic_alf_aps_id_chroma	u(3)
}	
}	
}	
if (!pps_dep_quant_enabled_flag)	
pic_dep_quant_enabled_flag	u(1)
if(!pic_dep_quant_enabled_flag)	
sign_data_hiding_enabled_flag	u(1)
if(deblocking_filter_override_enabled_flag) {	
pic_deblocking_filter_override_present_flag	u(1)
if(pic_deblocking_filter_override_present_flag) {	
pic_deblocking_filter_override_flag	u(1)
if(pic_deblocking_filter_override_flag) {	
pic_deblocking_filter_disabled_flag	u(1)
if(!pic_deblocking_filter_disabled_flag) {	
pic_beta_offset_div2	se(v)
pic_tc_offset_div2	se(v)
}	
}	
}	
}	
if(sps_lmcs_enabled_flag) {	
pic_lmcs_enabled_flag	u(1)
if(pic_lmcs_enabled_flag) {	
pic_lmcs_aps_id	u(2)
if(ChromaArrayType != 0)	
pic_chroma_residual_scale_flag	u(1)
}	
}	
if(sps_scaling_list_enabled_flag) {	
pic_scaling_list_present_flag	u(1)
if(pic_scaling_list_present_flag)	
pic_scaling_list_aps_id	u(3)
}	

if(picture_header_extension_present_flag) {	
ph_extension_length	ue(v)
for(i = 0; i < ph_extension_length; i++)	
ph_extension_data_byte[i]	u(8)
}	
rbsp_trailing_bits()	
}	

subpics_present_flag equal to 1 indicates that subpicture parameters are present in the present in the SPS RBSP syntax. **subpics_present_flag** equal to 0 indicates that subpicture parameters are not present in the present in the SPS RBSP syntax.

NOTE 2 – When a bitstream is the result of a sub-bitstream extraction process and contains only a subset of the subpictures of the input bitstream to the sub-bitstream extraction process, it might be required to set the value of **subpics_present_flag** equal to 1 in the RBSP of the SPSs.

sps_num_subpics_minus1 plus 1 specifies the number of subpictures. **sps_num_subpics_minus1** shall be in the range of 0 to 254. When not present, the value of **sps_num_subpics_minus1** is inferred to be equal to 0.

subpic_ctu_top_left_x[i] specifies horizontal position of top left CTU of i-th subpicture in unit of CtbSizeY. The length of the syntax element is $\text{Ceil}(\text{Log}_2(\text{pic_width_max_in_luma_samples} / \text{CtbSizeY}))$ bits. When not present, the value of **subpic_ctu_top_left_x[i]** is inferred to be equal to 0.

subpic_ctu_top_left_y[i] specifies vertical position of top left CTU of i-th subpicture in unit of CtbSizeY. The length of the syntax element is $\text{Ceil}(\text{Log}_2(\text{pic_height_max_in_luma_samples} / \text{CtbSizeY}))$ bits. When not present, the value of **subpic_ctu_top_left_y[i]** is inferred to be equal to 0.

subpic_width_minus1[i] plus 1 specifies the width of the i-th subpicture in units of CtbSizeY. The length of the syntax element is $\text{Ceil}(\text{Log}_2(\text{pic_width_max_in_luma_samples} / \text{CtbSizeY}))$ bits. When not present, the value of **subpic_width_minus1[i]** is inferred to be equal to $\text{Ceil}(\text{pic_width_max_in_luma_samples} / \text{CtbSizeY}) - 1$.

subpic_height_minus1[i] plus 1 specifies the height of the i-th subpicture in units of CtbSizeY. The length of the syntax element is $\text{Ceil}(\text{Log}_2(\text{pic_height_max_in_luma_samples} / \text{CtbSizeY}))$ bits. When not present, the value of **subpic_height_minus1[i]** is inferred to be equal to $\text{Ceil}(\text{pic_height_max_in_luma_samples} / \text{CtbSizeY}) - 1$.

subpic_treated_as_pic_flag[i] equal to 1 specifies that the i-th subpicture of each coded picture in the CVS is treated as a picture in the decoding process excluding in-loop filtering operations. **subpic_treated_as_pic_flag[i]** equal to 0 specifies that the i-th subpicture of each coded picture in the CVS is not treated as a picture in the decoding process excluding in-loop filtering operations. When not present, the value of **subpic_treated_as_pic_flag[i]** is inferred to be equal to 0.

loop_filter_across_subpic_enabled_flag[i] equal to 1 specifies that in-loop filtering operations may be performed across the boundaries of the i-th subpicture in each coded picture in the CVS.

loop_filter_across_subpic_enabled_flag[i] equal to 0 specifies that in-loop filtering operations are not performed

across the boundaries of the i -th subpicture in each coded picture in the CVS. When not present, the value of `loop_filter_across_subpic_enabled_pic_flag[i]` is inferred to be equal to 1.

It is a requirement of bitstream conformance that the following constraints apply:

- For any two subpictures `subpicA` and `subpicB`, when the index of `subpicA` is less than the index of `subpicB`, any coded NAL unit of `subpicA` shall succeed any coded NAL unit of `subpicB` in decoding order.
- The shapes of the subpictures shall be such that each subpicture, when decoded, shall have its entire left boundary and entire top boundary consisting of picture boundaries or consisting of boundaries of previously decoded subpictures.

`sps_subpic_id_present_flag` equal to 1 specifies that subpicture Id mapping is present in the SPS.

`sps_subpic_id_present_flag` equal to 0 specifies that subpicture Id mapping is not present in the SPS.

`sps_subpic_id_signalling_present_flag` equal to 1 specifies that subpicture Id mapping is signalled in the SPS.

`sps_subpic_id_signalling_present_flag` equal to 0 specifies that subpicture Id mapping is not signalled in the SPS.

When not present, the value of `sps_subpic_id_signalling_present_flag` is inferred to be equal to 0.

`sps_subpic_id_len_minus1` plus 1 specifies the number of bits used to represent the syntax element

`sps_subpic_id[i]`. The value of `sps_subpic_id_len_minus1` shall be in the range of 0 to 15, inclusive.

`sps_subpic_id[i]` specifies that subpicture Id of the i -th subpicture. The length of the `sps_subpic_id[i]` syntax element is `sps_subpic_id_len_minus1 + 1` bits. When not present, and when `sps_subpic_id_present_flag` equal to 0, the value of `sps_subpic_id[i]` is inferred to be equal to i , for each i in the range of 0 to `sps_num_subpics_minus1`, inclusive

`ph_pic_parameter_set_id` specifies the value of `pps_pic_parameter_set_id` for the PPS in use. The value of `ph_pic_parameter_set_id` shall be in the range of 0 to 63, inclusive.

It is a requirement of bitstream conformance that the value of `TemporalId` of the picture header shall be greater than or equal to the value of `TemporalId` of the PPS that has `pps_pic_parameter_set_id` equal to `ph_pic_parameter_set_id`.

`ph_subpic_id_signalling_present_flag` equal to 1 specifies that subpicture Id mapping is signalled in the picture header. `ph_subpic_id_signalling_present_flag` equal to 0 specifies that subpicture Id mapping is not signalled in the picture header.

`ph_subpic_id_len_minus1` plus 1 specifies the number of bits used to represent the syntax element

`ph_subpic_id[i]`. The value of `ph_subpic_id_len_minus1` shall be in the range of 0 to 15, inclusive.

It is a requirement of bitstream conformance that the value of `ph_subpic_id_len_minus1` shall be the same for all picture headers that are referred to by coded pictures in a CVS.

`ph_subpic_id[i]` specifies that subpicture Id of the i -th subpicture. The length of the `ph_subpic_id[i]` syntax element is `ph_subpic_id_len_minus1 + 1` bits.

The list `SubpicIdList[i]` is derived as follows:

```
for( i = 0; i <= sps_num_subpics_minus1; i++ )
    SubpicIdList[i] = sps_subpic_id_present_flag ?
```

(7-39)

$$\begin{aligned} & (\text{sps_subpic_id_signalling_present_flag} \quad ? \quad \text{sps_subpic_id}[i] \quad : \\ & (\text{ph_subpic_id_signalling_present_flag} \quad ? \quad \text{ph_subpic_id}[i] : \text{pps_subpic_id}[i])) : i \end{aligned}$$

Deblocking filter process

General

Inputs to this process are the reconstructed picture prior to deblocking, i.e., the array recPicture_L and, when ChromaArrayType is not equal to 0, the arrays recPicture_{Cb} and recPicture_{Cr} .

Outputs of this process are the modified reconstructed picture after deblocking, i.e., the array recPicture_L and, when ChromaArrayType is not equal to 0, the arrays recPicture_{Cb} and recPicture_{Cr} .

The vertical edges in a picture are filtered first. Then the horizontal edges in a picture are filtered with samples modified by the vertical edge filtering process as input. The vertical and horizontal edges in the CTBs of each CTU are processed separately on a coding unit basis. The vertical edges of the coding blocks in a coding unit are filtered starting with the edge on the left-hand side of the coding blocks proceeding through the edges towards the right-hand side of the coding blocks in their geometrical order. The horizontal edges of the coding blocks in a coding unit are filtered starting with the edge on the top of the coding blocks proceeding through the edges towards the bottom of the coding blocks in their geometrical order.

NOTE – Although the filtering process is specified on a picture basis in this Specification, the filtering process can be implemented on a coding unit basis with an equivalent result, provided the decoder properly accounts for the processing dependency order so as to produce the same output values.

The deblocking filter process is applied to all coding subblock edges and transform block edges of a picture, except the following types of edges:

- Edges that are at the boundary of the picture,
- Edges that coincide with the boundaries of a subpicture for which loop filter across subpic enabled flag[SubPicIdx] is equal to 0,
- Edges that coincide with the virtual boundaries of the picture when $\text{pps_loop_filter_across_virtual_boundaries_disabled_flag}$ is equal to 1,
- Edges that coincide with tile boundaries when $\text{loop_filter_across_tiles_enabled_flag}$ is equal to 0,
- Edges that coincide with slice boundaries when $\text{loop_filter_across_slices_enabled_flag}$ is equal to 0,
- Edges that coincide with upper or left boundaries of slices with $\text{slice_deblocking_filter_disabled_flag}$ equal to 1,
- Edges within slices with $\text{slice_deblocking_filter_disabled_flag}$ equal to 1,
- Edges that do not correspond to 4x4 sample grid boundaries of the luma component,
- Edges that do not correspond to 8x8 sample grid boundaries of the chroma component,
- Edges within the luma component for which both sides of the edge have $\text{intra_bdpcm_luma_flag}$ equal to 1,
- Edges within the chroma components for which both sides of the edge have $\text{intra_bdpcm_chroma_flag}$ equal to 1,
- Edges of chroma subblocks that are not edges of the associated transform unit.

...

Deblocking filter process for one direction

Inputs to this process are:

- the variable treeType specifying whether the luma (DUAL_TREE_LUMA) or chroma components (DUAL_TREE_CHROMA) are currently processed,
- when treeType is equal to DUAL_TREE_LUMA, the reconstructed picture prior to deblocking, i.e., the array recPicture_L,
- when ChromaArrayType is not equal to 0 and treeType is equal to DUAL_TREE_CHROMA, the arrays recPicture_{Cb} and recPicture_{Cr},
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered.

Outputs of this process are the modified reconstructed picture after deblocking, i.e:

- when treeType is equal to DUAL_TREE_LUMA, the array recPicture_L,
- when ChromaArrayType is not equal to 0 and treeType is equal to DUAL_TREE_CHROMA, the arrays recPicture_{Cb} and recPicture_{Cr}.

The variables firstCompIdx and lastCompIdx are derived as follows:

$$\text{firstCompIdx} = (\text{treeType} == \text{DUAL_TREE_CHROMA}) ? 1 : 0 \quad (8-1010)$$

$$\text{lastCompIdx} = (\text{treeType} == \text{DUAL_TREE_LUMA} \mid \mid \text{ChromaArrayType} == 0) ? 0 : 2 \quad (8-1011)$$

For each coding unit and each coding block per colour component of a coding unit indicated by the colour component index cIdx ranging from firstCompIdx to lastCompIdx, inclusive, with coding block width nCbW, coding block height nCbH and location of top-left sample of the coding block (xCb, yCb), when cIdx is equal to 0, or when cIdx is not equal to 0 and edgeType is equal to EDGE_VER and xCb % 8 is equal to 0, or when cIdx is not equal to 0 and edgeType is equal to EDGE_HOR and yCb % 8 is equal to 0, the edges are filtered by the following ordered steps:

1. The variable filterEdgeFlag is derived as follows:
 - If edgeType is equal to EDGE_VER and one or more of the following conditions are true, filterEdgeFlag is set equal to 0:
 - The left boundary of the current coding block is the left boundary of the picture.
 - *The left boundary of the current coding block is the left or right boundary of the subpicture and loop filter across subpic enabled flag/ SubPicIdx is equal to 0.*
 - The left boundary of the current coding block is the left boundary of the tile and loop_filter_across_tiles_enabled_flag is equal to 0.
 - The left boundary of the current coding block is the left boundary of the slice and loop_filter_across_slices_enabled_flag is equal to 0.
 - The left boundary of the current coding block is one of the vertical virtual boundaries of the picture and VirtualBoundariesDisabledFlag is equal to 1.
 - Otherwise, if edgeType is equal to EDGE_HOR and one or more of the following conditions are true, the variable filterEdgeFlag is set equal to 0:
 - The top boundary of the current luma coding block is the top boundary of the picture.

- The top boundary of the current coding block is the top or bottom boundary of the subpicture and loop filter across subpic enabled flag SubPicIdx is equal to 0.
- The top boundary of the current coding block is the top boundary of the tile and loop_filter_across_tiles_enabled_flag is equal to 0.
- The top boundary of the current coding block is the top boundary of the slice and loop_filter_across_slices_enabled_flag is equal to 0.
- The top boundary of the current coding block is one of the horizontal virtual boundaries of the picture and VirtualBoundariesDisabledFlag is equal to 1.
- Otherwise, filterEdgeFlag is set equal to 1.

[0097] 2.7 Example TPM, HMVP and GEO

[0098] TPM (triangular Prediction Mode) in VVC divides a block into two triangles with different motion information.

[0099] HMVP (History-based Motion vector Prediction) in VVC maintains a table of motion information to be used for motion vector prediction. The table is updated after decoding an inter-coded block, but it is not updated if the inter-coded block is TPM-coded.

[00100] GEO (geometry partition mode) is an extension of TPM. With GEO, a block can be divided by a straight-line into two partitions, which may be or may not be triangles.

[00101] 2.8 ALF, CC-ALF and virtual boundary

[00102] ALF (Adaptive Loop-Filter) in VVC is applied after a picture has been decoded, to enhance the picture quality.

[00103] Virtual Boundary (VB) is adopted in VVC to make ALF friendly to hardware design. With VB, ALF is conducted in an ALF processing unit bounded by two ALF virtual boundaries.

[00104] CC-ALF (cross-component ALF) as filters the chroma samples by referring to the information of luma samples.

[00105] 3. Examples of technical problems solved by disclosed embodiments

[00106] (1) There are some designs that can violate the sub-picture constrain.

[00107] A. TMVP in the affine constructed candidates may fetch a MV in the collocated picture out of the range of the current sub-picture.

[00108] B. When deriving gradients in Bi-Directional Optical Flow (BDOF) and Prediction Refinement Optical Flow (PROF), two extended rows and two extended columns of integer reference samples are required to be fetched. These reference samples may be out of the range of the current sub-picture.

[00109] C. When deriving the chroma residual scaling factor in luma mapping chroma scaling (LMCS), the accessed reconstructed luma samples may be out of the range of the range of the current sub-picture.

[00110] D. The neighboring block may be out of the range of the current sub-picture, when deriving the luma intra prediction mode, reference samples for intra prediction, reference samples for CCLM, neighboring block availability for spatial neighboring candidates for merge/AMVP/CIIP/IBC/LMCS, quantization parameters, CABAC initialization process, ctxInc derivation using left and above syntax elements, and ctxInc for the syntax element `mtt_split_cu_vertical_flag`. The representation of sub-picture may lead to sub-picture with incomplete CTUs. The CTU partitions and CU splitting process may need to consider incomplete CTUs.

[00111] (2) The signaled syntax elements related to sub-picture may be arbitrarily large, which may cause an overflow problem.

[00112] (3) The representation of sub-pictures may lead to non-rectangular sub-pictures.

[00113] (4) Currently the sub-picture and sub-picture grid is defined in units of 4 samples. And the length of syntax element is dependent on the picture height divided by 4. However, since the current `pic_width_in_luma_samples` and `pic_height_in_luma_samples` shall be an integer multiple of $\text{Max}(8, \text{MinCbSizeY})$, the sub-picture grid may need to be defined in units of 8 samples.

[00114] (5) The SPS syntax, `pic_width_max_in_luma_samples` and `pic_height_max_in_luma_samples` may need to be restricted to be no smaller than 8.

[00115] (6) Interaction between reference picture resampling/scalability and sub-picture is not considered in the current design.

[00116] (7) In temporal filtering, samples across different sub-pictures may be required.

[00117] (8) When signaling the slices, the information could be inferred without signaling in some cases.

[00118] (9) It is possible that all the defined slices cannot cover the whole picture or sub-picture.

[00119] (1) The IDs of two sub-pictures may be identical.

[00120] (11) $\text{pic_width_max_in_luma_samples} / \text{CtbSizeY}$ may be equal to 0, resulting in a meaningless $\text{Log2}()$ operation.

[00121] (12) ID in PH is more preferable than in PPS, but less preferable than in SPS, which is inconsistent.

[00122] (13) `log2_transform_skip_max_size_minus2` in PPS is parsed depending on `sps_transform_skip_enabled_flag` in SPS, resulting in a parsing dependency.

[00123] (14) `loop_filter_across_subpic_enabled_flag` for deblocking only consider the current sub-picture, without considering the neighbouring sub-picture.

[00124] 4. Example techniques and embodiments

[00125] The detailed listing below should be considered as examples to explain general concepts. These items should not be interpreted in a narrow way. Furthermore, these items can be combined in any manner. Hereinafter, temporal filter is used to represent filters that require samples in other pictures. $\text{Max}(x, y)$ returns the larger one of x and y . $\text{Min}(x, y)$ returns the smaller one of x and y .

1. The position (named position RB) at which a temporal MV predictor is fetched in a picture to generate affine motion candidates (e.g. a constructed affine merge candidate) must be in a required sub-picture, assuming the top-left corner coordinate of the required sub-picture is (x_{TL}, y_{TL}) and bottom-right coordinate of the required sub-picture is (x_{BR}, y_{BR}) .
 - a. In one example, the required sub-picture is the sub-picture covering the current block.
 - b. In one example, if position RB with a coordinate (x, y) is out of the required sub-picture, the temporal MV predictor is treated as unavailable.
 - i. In one example, position RB is out of the required sub-picture if $x > x_{BR}$.
 - ii. In one example, position RB is out of the required sub-picture if $y > y_{BR}$.
 - iii. In one example, position RB is out of the required sub-picture if $x < x_{TL}$.
 - iv. In one example, position RB is out of the required sub-picture if $y < y_{TL}$.
 - c. In one example, position RB, if outside of the required sub-picture, a replacement of RB is utilized.
 - i. Alternatively, furthermore, the replacement position shall be in the required sub-picture.
 - d. In one example, position RB is clipped to be in the required sub-picture.
 - i. In one example, x is clipped as $x = \text{Min}(x, x_{BR})$.
 - ii. In one example, y is clipped as $y = \text{Min}(y, y_{BR})$.
 - iii. In one example, x is clipped as $x = \text{Max}(x, x_{TL})$.
 - iv. In one example, y is clipped as $y = \text{Max}(y, y_{TL})$.

- e. In one example, the position RB may be the bottom right position inside the corresponding block of current block in the collocated picture.
 - f. The proposed method may be utilized in other coding tools which require to access motion information from a picture different than the current picture.
 - g. In one example, whether the above methods are applied (e.g., position RB must be in a required sub-picture (e.g. to do as claimed in 1.a and/or 1.b)) may depend on one or more syntax elements signaled in VPS/DPS/SPS/PPS/APS/slice header/tile group header. For example, the syntax element may be `subpic_treated_as_pic_flag[ SubPicIdx ]`, where `SubPicIdx` is the sub-picture index of sub-picture covering the current block.
2. The position (named position S) at which an integer sample is fetched in a reference not used in the interpolation process must be in a required sub-picture, assuming the top-left corner coordinate of the required sub-picture is (xTL, yTL) and the bottom-right coordinate of the required sub-picture is (xBR, yBR).
- a. In one example, the required sub-picture is the sub-picture covering the current block.
 - b. In one example, if position S with a coordinate (x, y) is out of the required sub-picture, the reference sample is treated as unavailable.
 - i. In one example, position S is out of the required sub-picture if $x > xBR$.
 - ii. In one example, position S is out of the required sub-picture if $y > yBR$.
 - iii. In one example, position S is out of the required sub-picture if $x < xTL$.
 - iv. In one example, position S is out of the required sub-picture if $y < yTL$.
 - c. In one example, position S is clipped to be in the required sub-picture.
 - i. In one example, x is clipped as $x = \text{Min}(x, xBR)$.
 - ii. In one example, y is clipped as $y = \text{Min}(y, yBR)$.
 - iii. In one example, x is clipped as $x = \text{Max}(x, xTL)$.
 - iv. In one example, y is clipped as $y = \text{Max}(y, yTL)$.
 - d. In one example, whether position S must be in a required sub-picture (e.g. to do as claimed in 2.a and/or 2.b) may depend on one or more syntax elements signaled in VPS/DPS/SPS/PPS/APS/slice header/tile group header. For example,

the syntax element may be `subpic_treated_as_pic_flag[ SubPicIdx ]`, where `SubPicIdx` is the sub-picture index of sub-picture covering the current block.

- e. In one example, the fetched integer sample is used to generate gradients in BDOF and/or PORF.
3. The position (named position R) at which the reconstructed luma sample value is fetched may be in a required sub-picture, assuming the top-left corner coordinate of the required sub-picture is (xTL, yTL) and the bottom-right coordinate of the required sub-picture is (xBR, yBR).
 - a. In one example, the required sub-picture is the sub-picture covering the current block.
 - b. In one example, if position R with a coordinate (x, y) is out of the required sub-picture, the reference sample is treated as unavailable.
 - i. In one example, position R is out of the required sub-picture if $x > xBR$.
 - ii. In one example, position R is out of the required sub-picture if $y > yBR$.
 - iii. In one example, position R is out of the required sub-picture if $x < xTL$.
 - iv. In one example, position R is out of the required sub-picture if $y < yTL$.
 - c. In one example, position R is clipped to be in the required sub-picture.
 - i. In one example, x is clipped as $x = \text{Min}(x, xBR)$.
 - ii. In one example, y is clipped as $y = \text{Min}(y, yBR)$.
 - iii. In one example, x is clipped as $x = \text{Max}(x, xTL)$.
 - iv. In one example, y is clipped as $y = \text{Max}(y, yTL)$.
 - d. In one example, whether position R must be in a required sub-picture (e.g. to do as claimed in 4.a and/or 4.b) may depend on one or more syntax elements signaled in VPS/DPS/SPS/PPS/APS/slice header/tile group header. For example, the syntax element may be `subpic_treated_as_pic_flag[ SubPicIdx ]`, where `SubPicIdx` is the sub-picture index of sub-picture covering the current block.
 - e. In one example, the fetched luma sample is used to derive the scaling factor for the chroma component(s) in LMCS.
 4. The position (named position N) at which the picture boundary check for BT/TT/QT splitting, BT/TT/QT depth derivation, and/or the signaling of CU split flag must be in a

required sub-picture, assuming the top-left corner coordinate of the required sub-picture is (xTL, yTL) and the bottom-right coordinate of the required sub-picture is (xBR, yBR).

- a. In one example, the required sub-picture is the sub-picture covering the current block.
 - b. In one example, if position N with a coordinate (x, y) is out of the required sub-picture, the reference sample is treated as unavailable.
 - i. In one example, position N is out of the required sub-picture if $x > xBR$.
 - ii. In one example, position N is out of the required sub-picture if $y > yBR$.
 - iii. In one example, position N is out of the required sub-picture if $x < xTL$.
 - iv. In one example, position N is out of the required sub-picture if $y < yTL$.
 - c. In one example, position N is clipped to be in the required sub-picture.
 - i. In one example, x is clipped as $x = \text{Min}(x, xBR)$.
 - ii. In one example, y is clipped as $y = \text{Min}(y, yBR)$.
 - iii. In one example, x is clipped as $x = \text{Max}(x, xTL)$.
 - d. In one example, y is clipped as $y = \text{Max}(y, yTL)$. In one example, whether position N must be in a required sub-picture (e.g. to do as claimed in 5.a and/or 5.b) may depend on one or more syntax elements signaled in VPS/DPS/SPS/PPS/APS/slice header/tile group header. For example, the syntax element may be `subpic_treated_as_pic_flag[ SubPicIdx ]`, where SubPicIdx is the sub-picture index of sub-picture covering the current block.
5. History -based Motion Vector Prediction (HMVP) table may be reset before decoding a new sub-picture in one picture.
- a. In one example, the HMVP table used for IBC coding may be reset
 - b. In one example, the HMVP table used for inter coding may be reset
 - c. In one example, the HMVP table used for intra coding may be reset
6. The sub-picture syntax elements may be defined in units of N (such as N = 8, 32, and etc) samples.
- a. In one example, the width of each element of the sub-picture identifier grid in units of N samples.
 - b. In one example, the height of each element of the sub-picture identifier grid in units of N samples.

- c. In one example, N is set to the width and/or height of CTU.
- 7. The syntax element of picture width and picture height may be restricted to be no smaller than K ($K \geq 8$).
 - a. In one example, the picture width may need to be restricted to be no smaller than 8.
 - b. In one example, the picture height may need to be restricted to be no smaller than 8.
- 8. A conformance bitstream shall satisfy that sub-picture coding and Adaptive resolution conversion (ARC)/ Dynamic resolution conversion (DRC)/ Reference picture resampling (RPR) are disallowed to be enabled for one video unit (e.g., sequence).
 - a. In one example, signaling of enabling sub-picture coding may be under the conditions of disallowing ARC/DRC/RPR.
 - i. In one example, when sub-picture is enabled, such as **subpics_present_flag** equal to 1, **pic_width_in_luma_samples** for all pictures for which this SPS is active is equal to **max_width_in_luma_samples**.
 - b. Alternatively, sub-picture coding and ARC/DRC/RPR may be both enabled for one video unit (e.g., sequence).
 - i. In one example, a conformance bitstream shall satisfy that the downsampled sub-picture due to ARC/DRC/RPR shall still be in the form of K CTUs in width and M CTUs in height wherein K and M are both integers.
 - ii. In one example, a conformance bitstream shall satisfy that for sub-pictures not located at picture boundaries (e.g., right boundary and/or bottom boundary), the downsampled sub-picture due to ARC/DRC/RPR shall still be in the form of K CTUs in width and M CTUs in height wherein K and M are both integers.
 - iii. In one example, CTU sizes may be adaptively changed based on the picture resolution.

- 1) In one example, a max CTU size may be signaled in SPS. For each picture with less resolution, the CTU size may be changed accordingly based on the reduced resolution.
 - 2) In one example, CTU size may be signaled in SPS and PPS, and/or sub-picture level.
9. The syntax element `subpic_grid_col_width_minus1` and `subpic_grid_row_height_minus1` may be constrained.
- a. In one example, `subpic_grid_col_width_minus1` must be no larger (or must be smaller) than T1.
 - b. In one example, `subpic_grid_row_height_minus1` must be no larger (or must be smaller) than T2.
 - c. In one example, in a conformance bit-stream, `subpic_grid_col_width_minus1` and/or `subpic_grid_row_height_minus1` must follow the constraint such as bullet 3.a or 3.b.
 - d. In one example, T1 in 3.a and/or T2 in 3.b may depend on profiles/levels/tiers of a video coding standard.
 - e. In one example, T1 in 3.a may depend on the picture width.
 - i. For example, T1 is equal to $\text{pic_width_max_in_luma_samples}/4$ or $\text{pic_width_max_in_luma_samples}/4+\text{Off}$. Off may be 1, 2, -1, -2, etc.
 - f. In one example, T2 in 3.b may depend on the picture width.
 - i. For example, T2 is equal to $\text{pic_height_max_in_luma_samples}/4$ or $\text{pic_height_max_in_luma_samples}/4-1+\text{Off}$. Off may be 1, 2, -1, -2, etc.
10. It is constrained that a boundary between two sub-pictures must be a boundary between two CTUs.
- a. In other words, a CTU cannot be covered by more than one sub-picture.
 - b. In one example, the unit of `subpic_grid_col_width_minus1` may be the CTU width (such as 32, 64, 128), instead of 4 as in VVC. The sub-picture grid width should be $(\text{subpic_grid_col_width_minus1}+1) * \text{CTU width}$.
 - c. In one example, the unit of `subpic_grid_col_height_minus1` may be the CTU height (such as 32, 64, 128), instead of 4 as in VVC. The sub-picture grid height should be $(\text{subpic_grid_col_height_minus1}+1) * \text{CTU height}$.

- d. In one example, in a conformance bit-stream, the constraint must be satisfied if the sub-picture approach is applied.
11. It is constrained that the shape of a sub-picture must be rectangular.
- a. In one example, in a conformance bit-stream, the constraint must be satisfied if the sub-picture approach is applied.
 - b. Sub-picture may only contain rectangular slices. For example, in a conformance bit-stream, the constraint must be satisfied if the sub-picture approach is applied.
12. It is constrained that two sub-pictures cannot be overlapped.
- a. In one example, in a conformance bit-stream, the constraint must be satisfied if the sub-picture approach is applied.
 - b. Alternatively, two sub-pictures may be overlapped with each other.
13. It is constrained that any position in the picture must be covered by one and only one sub-picture.
- a. In one example, in a conformance bit-stream, the constraint must be satisfied if the sub-picture approach is applied.
 - b. Alternatively, one sample may not belong to any sub-picture.
 - c. Alternatively, one sample may belong to more than one sub-pictures.
14. It may be constrained that sub-pictures defined in a SPS mapped to every resolution presented in the same sequence should obey the location and/or size constrained mentioned above.
- a. In one example, the width and height of a sub-picture defined in the SPS mapped to a resolution presented in the same sequence, should be integer multiple times of N (such as 8, 16, 32) luma samples.
 - b. In one example, sub-pictures may be defined for certain layer and may be mapped to other layers.
 - i. For example, sub-pictures may be defined for the layer with the highest resolution in the sequence.
 - ii. For example, sub-pictures may be defined for the layer with the lowest resolution in the sequence.
 - iii. Which layer the sub-pictures are defined for may be signaled in SPS/VPS/PPS/slice header.

- c. In one example, when sub-pictures and different resolutions are both applied, all resolutions (e.g., width or/and height) may be integer multiple of a given resolution.
 - d. In one example, the width and/or height of a sub-picture defined in the SPS may be integer multiple times (e.g., M) of the CTU size.
 - e. Alternatively, sub-pictures and different resolutions in a sequence may not be allowed simultaneously.
15. Sub-pictures may only apply to a certain layer(s)
- a. In one example, sub-pictures defined in a SPS may only apply to the layer with the highest resolution in a sequence.
 - b. In one example, sub-pictures defined in a SPS may only apply to the layer with the lowest temporal id in a sequence.
 - c. Which layer(s) that sub-pictures may be applied to may be indicated by one or multiple syntax elements in SPS/VPS/PPS.
 - d. Which layer(s) that sub-picture cannot be applied to may be indicated by one or multiple syntax elements in SPS/VPS/PPS.
16. In one example, the position and/or dimensions of a sub-picture may be signaled without using `subpic_grid_idx`.
- a. In one example, the top-left position of a sub-picture may be signaled.
 - b. In one example, the bottom-right position of a sub-picture may be signaled.
 - c. In one example, the width of sub-picture may be signaled.
 - d. In one example, the height of a sub-picture may be signaled.
17. For temporal filter, when performing the temporal filtering of a sample, only samples within the same sub-picture that the current sample belongs to may be used. The required samples may be in the same picture that the current sample belongs to or in other pictures.
18. In one example, whether to and/or how to apply a partitioning method (such as QT, horizontal BT, vertical BT, horizontal TT, vertical TT, or not split, etc.) may depend on whether the current block (or partition) crosses one or multiple boundary of a sub-picture.
- a. In one example, the picture boundary handling method for partitioning in VVC may also be applied when a picture boundary is replaced by a sub-picture boundary.

- b. In one example, whether to parse a syntax element (e.g. a flag) which represents a partitioning method (such as QT, horizontal BT, vertical BT, horizontal TT, vertical TT, or not split, etc.) may depend on whether the current block (or partition) crosses one or multiple boundary of a sub-picture.
19. Instead of splitting one picture into multiple sub-pictures with independent coding of each sub-picture, it is proposed to split a picture into at least two sets of sub-regions, with the first set including multiple sub-pictures and the second set including all the remaining samples.
- a. In one example, a sample in the second set is not in any sub-pictures.
 - b. Alternatively, furthermore, the second set may be encoded/decoded based on the information of the first set.
 - c. In one example, a default value may be utilized to mark whether a sample/MxK sub-region belonging to the second set.
 - i. In one example, the default value may be set equal to $(\text{max_subpics_minus1} + K)$ wherein K is an integer greater than 1.
 - ii. The default value may be assigned to `subpic_grid_idx[i][j]` to indicate that grid belongs to the second set.
20. It is proposed that the syntax element `subpic_grid_idx[i][j]` cannot be larger than `max_subpics_minus1`.
- a. For example, it is constrained that in a conformance bit-stream, `subpic_grid_idx[i][j]` cannot be larger than `max_subpics_minus1`.
 - b. For example, the codeword to code `subpic_grid_idx[i][j]` cannot be larger than `max_subpics_minus1`.
21. It is proposed that, any integer number from 0 to `max_subpics_minus1` must be equal to at least one `subpic_grid_idx[i][j]`.
22. IBC virtual buffer may be reset before decoding a new sub-picture in one picture.
- a. In one example, all the samples in the IBC virtual buffer may be reset to -1.
23. Palette entry list may be reset before decoding a new sub-picture in one picture.
- a. In one example, `PredictorPaletteSize` may be set equal to 0 before decoding a new sub-picture in one picture.

24. Whether to signal the information of slices (e.g. number of slices and/or ranges of slices) may depend on the number of tiles and/or the number of bricks.

- a. In one example, if the number of bricks in a picture is one, num_slices_in_pic_minus1 is not signaled and inferred to be 0.
- b. In one example, if the number of bricks in a picture is one, the information of slices (e.g. number of slices and/or ranges of slices) may not be signaled.
- c. In one example, if the number of bricks in a picture is one, the number of slices may be inferred to be one. And the slice covers the whole picture. In one example, if the number of bricks in a picture is one, single_brick_per_slice_flag is not signaled and inferred to be one.
 - i. Alternatively, if the number of bricks in a picture is one, single_brick_per_slice_flag must be one.
- d. An exemplary syntax design is as below:

pic_parameter_set_rbsp() {	Descriptor
...	
<u>if(NumBricksInPic > 1){</u>	
single_brick_per_slice_flag	u(1)
if(!single_brick_per_slice_flag)	
rect_slice_flag	u(1)
if(rect_slice_flag && !single_brick_per_slice_flag) {	
num_slices_in_pic_minus1	ue(v)
bottom_right_brick_idx_length_minus1	ue(v)
for(i = 0; i < num_slices_in_pic_minus1; i++) {	
bottom_right_brick_idx_delta[i]	u(v)
brick_idx_delta_sign_flag[i]	u(1)
}	
}	
<u>}</u>	
loop_filter_across_bricks_enabled_flag	u(1)
if(loop_filter_across_bricks_enabled_flag)	
loop_filter_across_slices_enabled_flag	u(1)
}	
...	

25. Whether to signal slice_address may be decoupled from whether slices are signaled to be rectangles (e.g. whether rect_slice_flag is equal to 0 or 1).

a. An exemplary syntax design is as below:

if([[rect_slice_flag]] NumBricksInPic > 1)	
slice_address	u(v)

26. Whether to signal slice_address may depend on the number of slices when slices are signaled to be rectangles.

<u>if((rect_slice_flag && num_slices_in_pic_minus1 > 0) (!rect_slice_flag && NumBricksInPic > 1))</u>	
slice_address	u(v)

27. Whether to signal num_bricks_in_slice_minus1 may depend on the slice_address and/or the number of bricks in the picture.

a. An exemplary syntax design is as below:

if(!rect_slice_flag && !single_brick_per_slice_flag && slice_address < NumBricksInPic - 1)	
num_bricks_in_slice_minus1	ue(v)

28. Whether to signal loop_filter_across_bricks_enabled_flag may depend on the number of tiles and/or the number of bricks.

a. In one example, loop_filter_across_bricks_enabled_flag is not signaled if the number of bricks is less than 2.

b. An exemplary syntax design is as below:

pic_parameter_set_rbsp() {	Descriptor
...	
<u>if(NumBricksInPic > 1)</u>	
loop_filter_across_bricks_enabled_flag	u(1)
if(loop_filter_across_bricks_enabled_flag)	
loop_filter_across_slices_enabled_flag	u(1)
...	

29. It is a requirement of bitstream conformance that all the slices of a picture must cover the whole picture.

a. The requirement must be satisfied when slices are signaled to be rectangles (e.g. rect_slice_flag is equal to 1).

30. It is a requirement of bitstream conformance that all the slices of a sub-picture must cover the whole sub-picture.

- a. The requirement must be satisfied when slices are signaled to be rectangles (e.g. `rect_slice_flag` is equal to 1).

- 31. It is a requirement of bitstream conformance that a slice cannot be overlapped with more than one sub-picture.
- 32. It is a requirement of bitstream conformance that a tile cannot be overlapped with more than one sub-picture.
- 33. It is a requirement of bitstream conformance that a brick cannot be overlapped with more than one sub-picture.

In the following discussion, a basic unit block (BUB) with dimensions $CW \times CH$ is a rectangle region. For example, a BUB may be a Coding Tree Block (CTB).

- 34. In one example, the number of sub-pictures (denoted as N) may be signaled.
 - a. It may be required on a conformance bit-stream that there are at least two sub-pictures in a picture if sub-pictures are used (e.g. `subpics_present_flag` is equal to 1).
 - b. Alternatively, N minus d (i.e., $N-d$) may be signaled, where d is an integer such as 0, 1, or 2.
 - c. For example, $N-d$ may be coded with fixed length coding e.g. $u(x)$.
 - i. In one example, x may be a fixed number such as 8.
 - ii. In one example, x or $x-dx$ may be signaled before $N-d$ is signaled, where dx is an integer such as 0, 1 or 2. The signaled x may not be larger than a maximum value in a conformance bitstream.
 - iii. In one example, x may be derived on-the-fly.
 - 1) For example, x may be derived as a function of the total number (denoted as M) of BUBs in the picture. E.g., $x = \text{Ceil}(\log_2(M+d_0))+d_1$, where d_0 and d_1 are two integers, such as -2, -1, 0, 1, 2, etc. Here, $\text{Ceil}()$ function returns the smallest integer value that is bigger than or equal to the input value.
 - 2) M may be derived as $M = \text{Ceiling}(W/CW) \times \text{Ceiling}(H/CH)$, where W and H represent the width and height of the picture, and CW and CH represent the width and height of a BUB.
 - d. For example, $N-d$ may be coded with a unary code or a truncated unary code.

- e. In one example, the allowed maximum value of N-d may be a fixed number.
 - i. Alternatively, the allowed maximum value of N-d may be derived as a function of the total number (denoted as M) of BUBs in the picture. E.g., $x = \text{Ceil}(\log_2(M+d_0))+d_1$, where d0 and d1 are two integers, such as -2, -1, 0, 1, 2, etc. Here, Ceil() function returns the smallest integer value that is bigger than or equal to the input value.
35. In one example, a sub-picture may be signaled by indications of one or multiple its selected positions (e.g., top-left/top-right/bottom-left/bottom-right position) and/or its width and/or its height.
- a. In one example, the top-left position of a sub-picture may be signaled in the granularity of a basic unit block (BUB) with dimensions $CW \times CH$.
 - i. For example, the column index (denoted as Col) in terms of BUBs of the top-left BUB of the sub-picture may be signaled.
 - 1) For example, Col - d may be signaled, where d is an integer such as 0, 1, or 2.
 - a) Alternatively, d may be equal to Col of a sub-picture previously coded, added by d1, where d1 is an integer such as -1, 0, or 1.
 - b) The sign of Col -d may be signaled.
 - ii. For example, the row index (denoted as Row) in term of BUBs of the top-left BUB of the sub-picture may be signaled.
 - 1) For example, Row - d may be signaled, where d is an integer such as 0, 1, or 2.
 - a) Alternatively, d may be equal to Row of a sub-picture previously coded, added by d1, where d1 is an integer such as -1, 0, or 1.
 - b) The sign of Row -d may be signaled.
 - iii. The row/column index (denoted as Row) mentioned above may be represented in the Coding Tree Block (CTB) unit, e.g., the x or y coordinate relative to the top-left position of a picture may be divided by CTB size and signaled.

- iv. In one example, whether to signal the position of a sub-picture may depend on the sub-picture index.
 - 1) In one example, for the first sub-picture within a picture, the top-left position may be not signaled.
 - a) Alternatively, furthermore, the top-left position may be inferred, e.g., to be (0, 0).
 - 2) In one example, for the last sub-picture within a picture, the top-left position may be not signaled.
 - a) The top-left position may be inferred depending on information of sub-pictures previously signaled.
- b. In one example, indications of the width/height/a selected position of a sub-picture may be signaled with truncated unary/truncated binary/unary/fixed length/K-th EG coding (e.g., $K=0, 1, 2, 3$).
- c. In one example, the width of a sub-picture may be signaled in the granularity of a BUB with dimensions $CW \times CH$.
 - i. For example, the number of columns of BUBs in the sub-picture (denoted as W) may be signaled.
 - ii. For example, $W-d$ may be signaled, where d is an integer such as 0, 1, or 2.
 - 1) Alternatively, d may be equal to W of a sub-picture previously coded, added by $d1$, where $d1$ is an integer such as -1, 0, or 1.
 - 2) The sign of $W-d$ may be signaled.
- d. In one example, the height of a sub-picture may be signaled in the granularity of a BUB with dimensions $CW \times CH$.
 - i. For example, the number of rows of BUBs in the sub-picture (denoted as H) may be signaled.
 - ii. For example, $H-d$ may be signaled, where d is an integer such as 0, 1, or 2.
 - 1) Alternatively, d may be equal to H of a sub-picture previously coded, added by $d1$, where $d1$ is an integer such as -1, 0, or 1.
 - 2) The sign of $H-d$ may be signaled.

- e. In one example, Col-d may be coded with fixed length coding e.g. $u(x)$.
 - i. In one example, x may be a fixed number such as 8.
 - ii. In one example, x or $x-dx$ may be signaled before Col-d is signaled, where dx is an integer such as 0, 1 or 2. The signaled x may not be larger than a maximum value in a conformance bitstream.
 - iii. In one example, x may be derived on-the-fly.
 - 1) For example, x may be derived as a function of the total number (denoted as M) of BUB columns in the picture. E.g., $x = \text{Ceil}(\log_2(M+d_0))+d_1$, where d_0 and d_1 are two integers, such as -2, -1, 0, 1, 2, etc. Here, $\text{Ceil}()$ function returns the smallest integer value that is bigger than or equal to the input value.
 - 2) M may be derived as $M = \text{Ceiling}(W/CW)$, where W represents the width of the picture, and CW represents the width of a BUB.
- f. In one example, Row-d may be coded with fixed length coding e.g. $u(x)$.
 - i. In one example, x may be a fixed number such as 8.
 - ii. In one example, x or $x-dx$ may be signaled before Row-d is signaled, where dx is an integer such as 0, 1 or 2. The signaled x may not be larger than a maximum value in a conformance bitstream.
 - iii. In one example, x may be derived on-the-fly.
 - 1) For example, x may be derived as a function of the total number (denoted as M) of BUB rows in the picture. E.g., $x = \text{Ceil}(\log_2(M+d_0))+d_1$, where d_0 and d_1 are two integers, such as -2, -1, 0, 1, 2, etc. Here, $\text{Ceil}()$ function returns the smallest integer value that is bigger than or equal to the input value.
 - 2) M may be derived as $M = \text{Ceiling}(H/CH)$, where H represents the height of the picture, and CH represents the height of a BUB.
- g. In one example, W-d may be coded with fixed length coding e.g. $u(x)$.
 - i. In one example, x may be a fixed number such as 8.
 - ii. In one example, x or $x-dx$ may be signaled before W-d is signaled, where dx is an integer such as 0, 1 or 2. The signaled x may not be larger than a maximum value in a conformance bitstream.

- iii. In one example, x may be derived on-the-fly.
 - 1) For example, x may be derived as a function of the total number (denoted as M) of BUB columns in the picture. E.g., $x = \text{Ceil}(\log_2(M+d_0))+d_1$, where d_0 and d_1 are two integers, such as -2, -1, 0, 1, 2, etc. Here, $\text{Ceil}()$ function returns the smallest integer value that is bigger than or equal to the input value.
 - 2) M may be derived as $M = \text{Ceiling}(W/CW)$, where W represents the width of the picture, and CW represents the width of a BUB.
- h. In one example, $H-d$ may be coded with fixed length coding e.g. $u(x)$.
 - i. In one example, x may be a fixed number such as 8.
 - ii. In one example, x or $x-dx$ may be signaled before $H-d$ is signaled, where dx is an integer such as 0, 1 or 2. The signaled x may not be larger than a maximum value in a conformance bitstream.
 - iii. In one example, x may be derived on-the-fly.
 - 1) For example, x may be derived as a function of the total number (denoted as M) of BUB rows in the picture. E.g., $x = \text{Ceil}(\log_2(M+d_0))+d_1$, where d_0 and d_1 are two integers, such as -2, -1, 0, 1, 2, etc. Here, $\text{Ceil}()$ function returns the smallest integer value that is bigger than or equal to the input value.
 - 2) M may be derived as $M = \text{Ceiling}(H/CH)$, where H represents the height of the picture, and CH represents the height of a BUB.
- i. Col- d and/or Row- d may be signaled for all sub-pictures.
 - i. Alternatively, Col- d and/or Row- d may not be signaled for all sub-pictures.
 - 1) Col- d and/or Row- d may not be signaled if the number of sub-pictures are less than 2. (equal to 1).
 - 2) For example, Col- d and/or Row- d may not be signaled for the first sub-picture (e.g. with the sub-picture index (or sub-picture ID) equal to 0).
 - a) When they are not signaled, they may be inferred to be 0.

- 3) For example, Col-d and/or Row-d may not be signaled for the last sub-picture (e.g. with the sub-picture index (or sub-picture ID) equal to NumSubPics-1).
 - a) When they are not signaled, they may be inferred depending on the positions and dimensions of sub-pictures already signaled.
- j. W-d and/or H-d may be signaled for all sub-pictures.
 - i. Alternatively, W-d and/or H-d may not be signaled for all sub-pictures.
 - 1) W-d and/or H-d may not be signaled if the number of sub-pictures are less than 2. (equal to 1).
 - 2) For example, W-d and/or H-d may not be signaled for the last sub-picture (e.g. with the sub-picture index (or sub-picture ID) equal to NumSubPics-1).
 - a) When they are not signaled, they may be inferred depending on the positions and dimensions of sub-pictures already signaled.
 - k. In the above bullets, a BUB may be a Coding Tree Block (CTB).
- 36. In one example, the information of sub-pictures should be signaled after information of the CTB size (e.g. log2_ctu_size_minus5) has already been signaled.
- 37. subpic_treated_as_pic_flag[i] may not be signaled for each sub-pictures. Instead, one subpic_treated_as_pic_flag is signaled to control whether a sub-picture is treated as a picture for all sub-pictures.
- 38. loop_filter_across_subpic_enabled_flag [i] may not be signaled for each sub-pictures. Instead, one loop_filter_across_subpic_enabled_flag is signaled to control whether loop filters can be applied across sub-pictures for all sub-pictures.
- 39. subpic_treated_as_pic_flag[i] (subpic_treated_as_pic_flag) and/or loop_filter_across_subpic_enabled_flag[i] (loop_filter_across_subpic_enabled_flag) may be signaled conditionally.
 - a. In one example, subpic_treated_as_pic_flag[i] and/or loop_filter_across_subpic_enabled_flag[i] may not be signaled if the number of sub-pictures are less than 2. (equal to 1).

40. RPR may be applied when sub-pictures are used.
- In one example, the scaling ratio in RPR may be constrained to be a limited set when sub-pictures are used, such as { 1:1, 1:2 and/or 2:1 }, or { 1:1, 1:2 and/or 2:1, 1:4 and/or 4:1 }, { 1:1, 1:2 and/or 2:1, 1:4 and/or 4:1, 1:8 and/or 8:1 }.
 - In one example, the CTB size of a picture A and the CTB size of a picture B may be different if the resolution of picture A and picture B are different.
 - In one example, suppose a sub-picture SA with dimensions $SAW \times SAH$ is in picture A and a sub-picture SB with dimensions $SBW \times SBH$ is in picture B, SA corresponds to SB, and the scaling ratios between picture A and picture B are R_w and R_h along the horizontal and vertical directions, then
 - SAW/SBW or SBW/SAW should be equal to R_w .
 - SAH/SBH or SBH/SAH should be equal to R_h .
41. When sub-pictures are used (e.g. `sub_pics_present_flag` is true), a subpicutre index (or sub-picture ID) may be signaled in the slice header, and the slice address is interrupted as the address in a sub-picture instead of the whole picture.
42. It is required that the subpicture ID of a first subpicture must be different to the subpicture ID of a second subpicture, if the first subpicture and the second subpicture are not the same sub-picture.
- In one example, it is a requirement in a conformance bitstream that `sps_subpic_id[ i ]` must be different from `sps_subpic_id[ j ]`, if i is not equal to j .
 - In one example, it is a requirement in a conformance bitstream that `pps_subpic_id[ i ]` must be different from `pps_subpic_id[ j ]`, if i is not equal to j .
 - In one example, it is a requirement in a conformance bitstream that `ph_subpic_id[ i ]` must be different from `ph_subpic_id[ j ]`, if i is not equal to j .
 - In one example, it is a requirement in a conformance bitstream that `SubpicIdList[ i ]` must be different from `SubpicIdList[ j ]`, if i is not equal to j .
 - In one example, a difference denoted as $D[i]$ equal to $X_subpic_id[i] - X_subpic_id[i - P]$ may be signaled.
 - For example, X may be `sps`, `pps` or `ph`.
 - For example, P is equal to 1.
 - For example, $i > P$.

- iv. For example, $D[i]$ must be larger than 0.
 - v. For example, $D[i]-1$ may be signaled.
43. It is proposed that the length of a syntax element specifying the horizontal or vertical position of top left CTU (e.g. **subpic_ctu_top_left_x** or **subpic_ctu_top_left_y**) may be derived to be $\text{Ceil}(\text{Log}_2(SS))$ bits, wherein SS must be larger than 0. Here, $\text{Ceil}()$ function returns the smallest integer value that is bigger than or equal to the input value.
- a. In one example, $SS = (\text{pic_width_max_in_luma_samples} + RR) / \text{CtbSizeY}$ when the syntax element specifies the horizontal position of top left CTU (e.g. **subpic_ctu_top_left_x**).
 - b. In one example, $SS = (\text{pic_height_max_in_luma_samples} + RR) / \text{CtbSizeY}$ when the syntax element specifies the vertical position of top left CTU (e.g. **subpic_ctu_top_left_y**).
 - c. In one example, RR is a non-zero integer such as $\text{CtbSizeY}-1$.
44. It is proposed that the length of a syntax element specifying the horizontal or vertical position of top left CTU of a subpicture (e.g. **subpic_ctu_top_left_x** or **subpic_ctu_top_left_y**) may be derived to be $\text{Ceil}(\text{Log}_2(SS))$ bits, wherein SS must be larger than 0. Here, $\text{Ceil}()$ function returns the smallest integer value that is bigger than or equal to the input value.
- a. In one example, $SS = (\text{pic_width_max_in_luma_samples} + RR) / \text{CtbSizeY}$ when the syntax element specifies the horizontal position of top left CTU of a subpicture (e.g. **subpic_ctu_top_left_x**).
 - b. In one example, $SS = (\text{pic_height_max_in_luma_samples} + RR) / \text{CtbSizeY}$ when the syntax element specifies the vertical position of top left CTU of a subpicture (e.g. **subpic_ctu_top_left_y**).
 - c. In one example, RR is a non-zero integer such as $\text{CtbSizeY}-1$.
45. It is proposed that the default value of the length of a syntax element (which may plus an offset P such as 1) specifying the width or height of a subpicture (e.g. **subpic_width_minus1** or **subpic_height_minus1**) may be derived to be $\text{Ceil}(\text{Log}_2(SS)) - P$, wherein SS must be larger than 0. Here, $\text{Ceil}()$ function returns the smallest integer value that is bigger than or equal to the input value.

- a. In one example, $SS = (\text{pic_width_max_in_luma_samples} + RR) / \text{CtbSizeY}$ when the syntax element specifies the default width (which may plus an offset P) of a subpicture (e.g. **subpic_width_minus1**).
 - b. In one example, $SS = (\text{pic_height_max_in_luma_samples} + RR) / \text{CtbSizeY}$ when the syntax element specifies the default height (which may plus an offset P) of a subpicture (e.g. **subpic_height_minus1**).
 - c. In one example, RR is a non-zero integer such as $\text{CtbSizeY}-1$.
46. It is proposed that, the information of IDs of sub-pictures should be signaled at least in one of SPS, PPS, and the picture header if it is determined that the information should be signaled.
- a. In one example, it is a requirement in a conformance bitstream that at least one of `sps_subpic_id_signalling_present_flag`, `pps_subpic_id_signalling_present_flag` and `ph_subpic_id_signalling_present_flag` should be equal to 1 if `sps_subpic_id_present_flag` is equal to 1.
47. It is proposed that, if the information of IDs of sub-pictures is not signaled in any one of SPS, PPS, and the picture header, but it is determined that the information should be signaled. default IDs should be assigned.
- a. In one example, if `sps_subpic_id_signalling_present_flag`, `pps_subpic_id_signalling_present_flag` and `ph_subpic_id_signalling_present_flag` are all equal to 0 and `sps_subpic_id_present_flag` is equal to 1, `SubpicIdList[ i ]` should be set equal to $i+P$, where P is an offset such as 0. An exemplary description is as below:
- ```
for(i = 0; i <= sps_num_subpics_minus1; i++)
 SubpicIdList[i] = sps_subpic_id_present_flag ?
 (sps_subpic_id_signalling_present_flag ? sps_subpic_id[i] :
 (ph_subpic_id_signalling_present_flag ? ph_subpic_id[i] :
 (pps_subpic_id_signalling_present_flag ? pps_subpic_id[i] : i))) : i
```
48. It is proposed that the information of sub-picture IDs are not signaled in a picture header if they are signaled in the corresponding PPS.

a. An exemplary syntax design is as below,

| picture_header_rbsp( ) {                                                                                   | Descriptor |
|------------------------------------------------------------------------------------------------------------|------------|
| <b>non_reference_picture_flag</b>                                                                          | u(1)       |
| <b>gdr_pic_flag</b>                                                                                        | u(1)       |
| <b>no_output_of_prior_pics_flag</b>                                                                        | u(1)       |
| if( gdr_pic_flag )                                                                                         |            |
| <b>recovery_poc_cnt</b>                                                                                    | ue(v)      |
| <b>ph_pic_parameter_set_id</b>                                                                             | ue(v)      |
| if( sps_subpic_id_present_flag && !sps_subpic_id_signalling_flag && !<br>pps_subpic_id_signalling_flag ) { |            |
| <b>ph_subpic_id_signalling_present_flag</b>                                                                | u(1)       |
| if( ph_subpics_id_signalling_present_flag ) {                                                              |            |
| <b>ph_subpic_id_len_minus1</b>                                                                             | ue(v)      |
| for( i = 0; i <= sps_num_subpics_minus1; i++ )                                                             |            |
| <b>ph_subpic_id[ i ]</b>                                                                                   | u(v)       |
| }                                                                                                          |            |
| }                                                                                                          |            |
| ...                                                                                                        |            |

b. In one example, the sub-picture IDs are set according to the information of sub-picture IDs signaled in SPS if they are signaled in SPS; Otherwise, the sub-picture IDs are set according to the information of sub-picture IDs signaled in PPS if they are signaled in PPS, Otherwise, the sub-picture IDs are set according to the information of sub-picture IDs signaled in the the picture header if they are signaled in the picture header. An exemplary description is as below,

```

for(i = 0; i <= sps_num_subpics_minus1; i++)
 SubpicIdList[i] = sps_subpic_id_present_flag ?
 (sps_subpic_id_signalling_present_flag ? sps_subpic_id[i] :
 (pps_subpic_id_signalling_present_flag ? pps_subpic_id[i] :
 (ph_subpic_id_signalling_present_flag ? ph_subpic_id[i] : i))) : i

```

c. In one example, the sub-picture IDs are set according to the information of sub-picture IDs signaled in the picture header if they are signaled in the picture header; Otherwise, the sub-picture IDs are set according to the information of sub-picture IDs signaled in PPS if they are signaled in PPS, Otherwise, the sub-

picture IDs are set according to the information of sub-picture IDs signaled in the SPS if they are signaled in SPS. An exemplary description is as below,

```

for(i = 0; i <= sps_num_subpics_minus1; i++)
 SubpicIdList[i] = sps_subpic_id_present_flag ?
 (ph_subpic_id_signalling_present_flag ? ph_subpic_id[i] :
 (pps_subpic_id_signalling_present_flag ? pps_subpic_id[i] :
 (sps_subpic_id_signalling_present_flag ? sps_subpic_id[i] : i))) : i

```

49. It is proposed that the deblocking process on an edge E should depend on the determination of whether loop-filtering is allowed across the subpicture boundaries (e.g. determined by `loop_filter_across_subpic_enabled_flag`) on both sides (denoted as P-side and Q-side) of the edge. P-side represents the side in the current block, and Q-side represents the side in the neighbouring block, which may belong to a different sub-picture. In the following discussion, it is assumed that P-side and Q-side belongs two different sub-pictures. `loop_filter_across_subpic_enabled_flag[P] = 0/1` means that loop-filtering is disallowed/allowed across the subpicture boundaries of the subpicture containing P-side. `loop_filter_across_subpic_enabled_flag[Q] = 0/1` means that loop-filtering is disallowed/allowed across the subpicture boundaries of the subpicture containing Q-side.

- a. In one example, E is not filtered if `loop_filter_across_subpic_enabled_flag[P]` is equal to 0 or `loop_filter_across_subpic_enabled_flag[Q]` is equal to 0.
- b. In one example, E is not filtered if `loop_filter_across_subpic_enabled_flag[P]` is equal to 0 and `loop_filter_across_subpic_enabled_flag[Q]` is equal to 0.
- c. In one example, whether to filter the two sides of E are controlled separately.
  - i. For example, P-side of E is filtered if and only if `loop_filter_across_subpic_enabled_flag[P]` is equal 1.
  - ii. For example, Q-side of E is filtered if and only if `loop_filter_across_subpic_enabled_flag[Q]` is equal 1.

50. It is proposed that, the signaling/parsing of a syntax element SE in PPS specifying the maximum block size used for transform skip (such as `log2_transform_skip_max_size_minus2`) should be decoupled from any syntax element in SPS (such as `sps_transform_skip_enabled_flag`).

a. An exemplary syntax change is as below:

|                                            |       |
|--------------------------------------------|-------|
| pic_parameter_set_rbsp( ) {                |       |
| ...                                        |       |
| [[if( sps_transform_skip_enabled_flag )]]  |       |
| <b>log2_transform_skip_max_size_minus2</b> | ue(v) |
| ...                                        |       |

b. Alternatively, SE may be signaled in SPS, such as:

|                                            |       |
|--------------------------------------------|-------|
| seq_parameter_set_rbsp( ) {                |       |
| ...                                        |       |
| if( sps_transform_skip_enabled_flag )      |       |
| <b>log2_transform_skip_max_size_minus2</b> | ue(v) |
| ...                                        |       |

c. Alternatively, SE may be signaled in the picture header, such as:

|                                            |       |
|--------------------------------------------|-------|
| picture_header_rbsp( ) {                   |       |
| ...                                        |       |
| if( sps_transform_skip_enabled_flag )      |       |
| <b>log2_transform_skip_max_size_minus2</b> | ue(v) |
| ...                                        |       |

51. Whether to and/or how to update the HMVP table (or named as list/storage/map etc.) after decoding a first block may depend on whether the first block is coded with GEO.

- a. In one example, the HMVP table may not be updated after decoding the first block if the first block is coded with GEO.
- b. In one example, the HMVP table may be updated after decoding the first block if the first block is coded with GEO.
  - i. In one example, the HMVP table may be updated with the motion information of one partition divided by GEO.
  - ii. In one example, the HMVP table may be updated with the motion information of multiple partitions divided by GEO

52. In CC-ALF, luma samples out of the current processing unit (e.g., ALF processing unit bounded by two ALF virtual boundaries) is excluded from filtering on chroma samples in the corresponding processing unit.

- a. Padded luma samples out of the current processing unit may be used to filter the chroma samples in the corresponding processing unit.
  - i. Any padding method disclosed in this document may be used to pad the luma samples.
- b. Alternatively, luma samples out of the current processing unit may be used to filter chroma samples in the corresponding processing unit.

## [00126] 5. Embodiments

[00127] In the following embodiments, the newly added texts are bold italicized and the deleted texts are marked by “[ ]”.

### [00128] 5.1 Embodiment 1: sub-picture constraint on affine constructed merge candidates

#### 8.5.5.6 Derivation process for constructed affine control point motion vector merging candidates

Inputs to this process are:

- a luma location (  $x_{Cb}$ ,  $y_{Cb}$  ) specifying the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- two variables  $cbWidth$  and  $cbHeight$  specifying the width and the height of the current luma coding block,
- the availability flags  $availableA_0$ ,  $availableA_1$ ,  $availableA_2$ ,  $availableB_0$ ,  $availableB_1$ ,  $availableB_2$ ,  $availableB_3$ ,
- the sample locations (  $x_{NbA_0}$ ,  $y_{NbA_0}$  ), (  $x_{NbA_1}$ ,  $y_{NbA_1}$  ), (  $x_{NbA_2}$ ,  $y_{NbA_2}$  ), (  $x_{NbB_0}$ ,  $y_{NbB_0}$  ), (  $x_{NbB_1}$ ,  $y_{NbB_1}$  ), (  $x_{NbB_2}$ ,  $y_{NbB_2}$  ) and (  $x_{NbB_3}$ ,  $y_{NbB_3}$  ).

Output of this process are:

- the availability flag of the constructed affine control point motion vector merging candidates  $availableFlagConstK$ , with  $K = 1..6$ ,
- the reference indices  $refIdxLXConstK$ , with  $K = 1..6$ ,  $X$  being 0 or 1,
- the prediction list utilization flags  $predFlagLXConstK$ , with  $K = 1..6$ ,  $X$  being 0 or 1,
- the affine motion model indices  $motionModelIdxConstK$ , with  $K = 1..6$ ,
- the bi-prediction weight indices  $bcwIdxConstK$ , with  $K = 1..6$ ,
- the constructed affine control point motion vectors  $cpMvLXConstK[cpIdx]$  with  $cpIdx = 0..2$ ,  $K = 1..6$  and  $X$  being 0 or 1.

...

The fourth (collocated bottom-right) control point motion vector  $cpMvLXCorner[3]$ , reference index  $refIdxLXCorner[3]$ , prediction list utilization flag  $predFlagLXCorner[3]$  and the availability flag  $availableFlagCorner[3]$  with  $X$  being 0 and 1 are derived as follows:

- The reference indices for the temporal merging candidate,  $refIdxLXCorner[3]$ , with  $X$  being 0 or 1, are set equal to 0.
- The variables  $mvLXCol$  and  $availableFlagLXCol$ , with  $X$  being 0 or 1, 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 (slice\_temporal\_mvp\_enabled\_flag is equal to 1), the following applies:
 
$$xColBr = xCb + cbWidth \quad (8-601)$$

$$yColBr = yCb + cbHeight \quad (8-602)$$

$$rightBoundaryPos = subpic\_treated\_as\_pic\_flag[SubPicIdx] ?$$

$$SubPicRightBoundaryPos : pic\_width\_in\_luma\_samples - 1$$

$$botBoundaryPos = subpic\_treated\_as\_pic\_flag[SubPicIdx] ?$$

$$SubPicBotBoundaryPos : pic\_height\_in\_luma\_samples - 1$$
- If  $yCb \gg CtbLog2SizeY$  is equal to  $yColBr \gg CtbLog2SizeY$ ,  $yColBr$  is less than or equal to  $botBoundaryPos$  and  $xColBr$  is less than or equal to  $rightBoundaryPos$ , the following applies:
  - The variable colCb specifies the luma coding block covering the modified location given by  $((xColBr \gg 3) \ll 3, (yColBr \gg 3) \ll 3)$  inside the collocated picture specified by ColPic.
  - The luma location  $(xColCb, yColCb)$  is set equal to the top-left sample of the collocated luma coding block specified by colCb relative to the top-left luma sample of the collocated picture specified by ColPic.
  - The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb,  $(xColCb, yColCb)$ , refIdxLXCorner[3] and sbFlag set equal to 0 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.

...

## [00129] 5.2 Embodiment 2: sub-picture constraint on affine constructed merge candidates

### 8.5.5.6 Derivation process for constructed affine control point motion vector merging candidates

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,
- two variables cbWidth and cbHeight specifying the width and the height of the current luma coding block,
- the availability flags availableA<sub>0</sub>, availableA<sub>1</sub>, availableA<sub>2</sub>, availableB<sub>0</sub>, availableB<sub>1</sub>, availableB<sub>2</sub>, availableB<sub>3</sub>,
- the sample locations  $(xNbA_0, yNbA_0)$ ,  $(xNbA_1, yNbA_1)$ ,  $(xNbA_2, yNbA_2)$ ,  $(xNbB_0, yNbB_0)$ ,  $(xNbB_1, yNbB_1)$ ,  $(xNbB_2, yNbB_2)$  and  $(xNbB_3, yNbB_3)$ .

Output of this process are:

- the availability flag of the constructed affine control point motion vector merging candidates availableFlagConstK, with  $K = 1..6$ ,
- the reference indices refIdxLXConstK, with  $K = 1..6$ , X being 0 or 1,
- the prediction list utilization flags predFlagLXConstK, with  $K = 1..6$ , X being 0 or 1,
- the affine motion model indices motionModelIdxConstK, with  $K = 1..6$ ,

- the bi-prediction weight indices bcwIdxConstK, with  $K = 1..6$ ,
- the constructed affine control point motion vectors cpMvLXConstK[ cpldx ] with cpldx = 0..2,  $K = 1..6$  and X being 0 or 1.

...

The fourth (collocated bottom-right) control point motion vector cpMvLXCorner[ 3 ], reference index refIdxLXCorner[ 3 ], prediction list utilization flag predFlagLXCorner[ 3 ] and the availability flag availableFlagCorner[ 3 ] with X being 0 and 1 are derived as follows:

- The reference indices for the temporal merging candidate, refIdxLXCorner[ 3 ], with X being 0 or 1, are set equal to 0.
- The variables mvLXCol and availableFlagLXCol, with X being 0 or 1, 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 (slice\_temporal\_mvp\_enabled\_flag is equal to 1), the following applies:

$$xColBr = xCb + cbWidth \quad (8-601)$$

$$yColBr = yCb + cbHeight \quad (8-602)$$

*rightBoundaryPos = subpic\_treated\_as\_pic\_flag[SubPicIdx] ?*

*SubPicRightBoundaryPos : pic\_width\_in\_luma\_samples - 1*

*botBoundaryPos = subpic\_treated\_as\_pic\_flag[SubPicIdx] ?*

*SubPicBotBoundaryPos : pic\_height\_in\_luma\_samples - 1*

*xColBr = Min( rightBoundaryPos, xColBr )*

*yColBr = Min( botBoundaryPos, yColBr )*

- If  $yCb \gg CtbLog2SizeY$  is equal to  $yColBr \gg CtbLog2SizeY$ , [[yColBr is less than pic\_height\_in\_luma\_samples and xColBr is less than pic\_width\_in\_luma\_samples, the following applies]]:
  - The variable colCb specifies the luma coding block covering the modified location given by  $((xColBr \gg 3) \ll 3, (yColBr \gg 3) \ll 3)$  inside the collocated picture specified by ColPic.
  - The luma location  $(xColCb, yColCb)$  is set equal to the top-left sample of the collocated luma coding block specified by colCb relative to the top-left luma sample of the collocated picture specified by ColPic.
  - The derivation process for collocated motion vectors as specified in clause 8.5.2.12 is invoked with currCb, colCb,  $(xColCb, yColCb)$ , refIdxLXCorner[ 3 ] and sbFlag set equal to 0 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.

...

### [00130] 5.3 Embodiment 3: Fetching integer samples under the sub-picture constraint

#### 8.5.6.3.3 Luma integer sample fetching process

Inputs to this process are:

- a luma location in full-sample units (  $x_{IntL}$ ,  $y_{IntL}$  ),
- the luma reference sample array  $refPicLX_L$ .

Output of this process is a predicted luma sample value  $predSampleLX_L$ .

The variable  $shift$  is set equal to  $\text{Max}(2, 14 - \text{BitDepth}_Y)$ .

The variable  $picW$  is set equal to  $pic\_width\_in\_luma\_samples$  and the variable  $picH$  is set equal to  $pic\_height\_in\_luma\_samples$ .

The luma locations in full-sample units (  $x_{Int}$ ,  $y_{Int}$  ) are derived as follows:

– If subpic treated as pic flag[ SubPicIdx ] is equal to 1, the following applies:

$$x_{Int} = \text{Clip3}( \text{SubPicLeftBoundaryPos}, \text{SubPicRightBoundaryPos}, x_{Int} )$$

$$y_{Int} = \text{Clip3}( \text{SubPicTopBoundaryPos}, \text{SubPicBotBoundaryPos}, y_{Int} )$$

– Otherwise:

$$x_{Int} = \text{Clip3}( 0, picW - 1, sps\_ref\_wraparound\_enabled\_flag ? \quad (8-782)$$

$$\text{ClipH}( ( sps\_ref\_wraparound\_offset\_minus1 + 1 ) * \text{MinCbSizeY}, picW, x_{IntL} ) : x_{IntL} )$$

$$y_{Int} = \text{Clip3}( 0, picH - 1, y_{IntL} ) \quad (8-783)$$

The predicted luma sample value  $predSampleLX_L$  is derived as follows:

$$predSampleLX_L = refPicLX_L[ x_{Int} ][ y_{Int} ] \ll shift3 \quad (8-784)$$

#### **[00131] 5.4 Embodiment 4: deriving the variable $invAvgLuma$ in chroma residual scaling of LMCS**

##### **8.7.5.3 Picture reconstruction with luma dependent chroma residual scaling process for chroma samples**

Inputs to this process are:

- a chroma location (  $x_{Curr}$ ,  $y_{Curr}$  ) of the top-left chroma sample of the current chroma transform block relative to the top-left chroma sample of the current picture,
- a variable  $nCurrSw$  specifying the chroma transform block width,
- a variable  $nCurrSh$  specifying the chroma transform block height,
- a variable  $tuCbfChroma$  specifying the coded block flag of the current chroma transform block,
- an  $(nCurrSw) \times (nCurrSh)$  array  $predSamples$  specifying the chroma prediction samples of the current block,
- an  $(nCurrSw) \times (nCurrSh)$  array  $resSamples$  specifying the chroma residual samples of the current block,

Output of this process is a reconstructed chroma picture sample array  $recSamples$ .

The variable  $sizeY$  is set equal to  $\text{Min}( CtbSizeY, 64 )$ .

The reconstructed chroma picture sample  $recSamples$  is derived as follows for  $i = 0..nCurrSw - 1$ ,

$j = 0..nCurrSh - 1$ :

- ...
- Otherwise, the following applies:
  - ...
  - The variable  $currPic$  specifies the array of reconstructed luma samples in the current picture.
  - For the derivation of the variable  $varScale$  the following ordered steps apply:

1. The variable `invAvgLuma` is derived as follows:

– The array `recLuma[ i ]` with  $i=0..(2 * \text{sizeY} - 1)$  and the variable `cnt` are derived as follows:

– The variable `cnt` is set equal to 0.

– *The variables `rightBoundaryPos` and `botBoundaryPos` are derived as follows:*

*`rightBoundaryPos = subpic_treated_as_pic_flag[ SubPicIdx ] ?`*

*`SubPicRightBoundaryPos : pic_width_in_luma_samples - 1`*

*`botBoundaryPos = subpic_treated_as_pic_flag[ SubPicIdx ] ?`*

*`SubPicBotBoundaryPos : pic_height_in_luma_samples - 1`*

– When `availL` is equal to TRUE, the array `recLuma[ i ]` with  $i = 0..\text{sizeY} - 1$  is set equal to `currPic[ xCuCb - 1 ][ Min( yCuCb + i, [[pic_height_in_luma_samples - 1]]`

*`botBoundaryPos` ) ]* with  $i = 0..\text{sizeY} - 1$ , and `cnt` is set equal to `sizeY`

– When `availT` is equal to TRUE, the array `recLuma[ cnt + i ]` with  $i = 0..\text{sizeY} - 1$  is set equal to

`currPic[ Min( xCuCb + i, [[pic_width_in_luma_samples - 1]] rightBoundaryPos ) ][ yCuCb - 1 ]` with  $i = 0..\text{sizeY} - 1$ , and `cnt` is set equal to  $(\text{cnt} + \text{sizeY})$

– The variable `invAvgLuma` is derived as follows:

– If `cnt` is greater than 0, the following applies:

$$\text{invAvgLuma} = \text{Clip1}_Y \left( \left( \sum_{k=0}^{\text{cnt}-1} \text{recLuma}[k] + (\text{cnt} >> 1) \right) \gg \text{Log2}(\text{cnt}) \right) \quad (8-1013)$$

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

$$\text{invAvgLuma} = 1 << (\text{BitDepth}_Y - 1) \quad (8-1014)$$

## [00132] 5.5 Embodiment 5: an example of defining the sub-picture element in unit of N (such as N=8 or 32) other than 4 samples

### 7.4.3.3 Sequence parameter set RBSP semantics

`subpic_grid_col_width_minus1` plus 1 specifies the width of each element of the sub-picture identifier grid in units of  $\frac{N}{4}$  samples. The length of the syntax element is  $\text{Ceil}(\text{Log2}(\text{pic\_width\_max\_in\_luma\_samples} / \frac{N}{4}))$  bits.

The variable `NumSubPicGridCols` is derived as follows:

$$\begin{aligned} \text{NumSubPicGridCols} = & (\text{pic\_width\_max\_in\_luma\_samples} + \text{subpic\_grid\_col\_width\_minus1} * [[4+3]] \frac{N}{4} + N - 1) / \\ & (\text{subpic\_grid\_col\_width\_minus1} * [[4+3]] \frac{N}{4} + N - 1) \end{aligned} \quad (7-5)$$

`subpic_grid_row_height_minus1` plus 1 specifies the height of each element of the sub-picture identifier grid in units of 4 samples. The length of the syntax element is  $\text{Ceil}(\text{Log2}(\text{pic\_height\_max\_in\_luma\_samples} / \frac{N}{4}))$  bits.

The variable `NumSubPicGridRows` is derived as follows:

$$\begin{aligned} \text{NumSubPicGridRows} = & (\text{pic\_height\_max\_in\_luma\_samples} + \text{subpic\_grid\_row\_height\_minus1} * \frac{N}{4} + N - 1) / \\ & (\text{subpic\_grid\_row\_height\_minus1} * [[4+3]] \frac{N}{4} + N - 1) \end{aligned}$$

### 7.4.7.1 General slice header semantics

The variables `SubPicIdx`, `SubPicLeftBoundaryPos`, `SubPicTopBoundaryPos`, `SubPicRightBoundaryPos`, and `SubPicBotBoundaryPos` are derived as follows:

```

SubPicIdx = CtbToSubPicIdx[CtbAddrBsToRs[FirstCtbAddrBs[SliceBrickIdx[0]]]]
if(
 subpic_treated_as_pic_flag[SubPicIdx]
)
{
 SubPicLeftBoundaryPos = SubPicLeft[SubPicIdx] * (subpic_grid_col_width_minus1 + 1) * $\frac{4}{N}$
 SubPicRightBoundaryPos = (SubPicLeft[SubPicIdx] + SubPicWidth[SubPicIdx]) *
 (subpic_grid_col_width_minus1 + 1) * $\frac{4}{N}$ (7-93)
 SubPicTopBoundaryPos = SubPicTop[SubPicIdx] * (subpic_grid_row_height_minus1 + 1) * $\frac{4}{N}$
 SubPicBotBoundaryPos = (SubPicTop[SubPicIdx] + SubPicHeight[SubPicIdx]) *
 (subpic_grid_row_height_minus1 + 1) * $\frac{4}{N}$
}

```

**[00133] 5.6 Embodiment 6: restrict the picture width and the picture height to be equal or larger than 8**

**7.4.3.3 Sequence parameter set RBSP semantics**

**pic\_width\_max\_in\_luma\_samples** specifies the maximum width, in units of luma samples, of each decoded picture referring to the SPS. **pic\_width\_max\_in\_luma\_samples** shall not be equal to 0 and shall be an integer multiple of  $[[\text{MinCbSizeY}]] \text{ } \underline{\text{Max}(8, \text{MinCbSizeY})}$ .

**pic\_height\_max\_in\_luma\_samples** specifies the maximum height, in units of luma samples, of each decoded picture referring to the SPS. **pic\_height\_max\_in\_luma\_samples** shall not be equal to 0 and shall be an integer multiple of  $[[\text{MinCbSizeY}]] \text{ } \underline{\text{Max}(8, \text{MinCbSizeY})}$ .

**[00134] 5.7 Embodiment 7: sub-picture boundary check for BT/TT/QT splitting, BT/TT/QT depth derivation, and/or the signaling of CU split flag**

**6.4.2 Allowed binary split process**

The variable **allowBtSplit** is derived as follows:

- ...
- Otherwise, if all of the following conditions are true, **allowBtSplit** is set equal to FALSE
  - **btSplit** is equal to **SPLIT\_BT\_VER**
  - $y_0 + \text{cbHeight}$  is greater than  $[[\text{pic\_height\_in\_luma\_samples}]] \text{ } \underline{\text{subpic\_treated\_as\_pic\_flag}[ \text{SubPicIdx} ] ? \text{SubPicBotBoundaryPos} + 1 : \text{pic\_height\_in\_luma\_samples}}$ .
- Otherwise, if all of the following conditions are true, **allowBtSplit** is set equal to FALSE
  - **btSplit** is equal to **SPLIT\_BT\_VER**
  - **cbHeight** is greater than **MaxTbSizeY**
  - $x_0 + \text{cbWidth}$  is greater than  $[[\text{pic\_width\_in\_luma\_samples}]] \text{ } \underline{\text{subpic\_treated\_as\_pic\_flag}[ \text{SubPicIdx} ] ? \text{SubPicRightBoundaryPos} + 1 : \text{pic\_width\_in\_luma\_samples}}$
- Otherwise, if all of the following conditions are true, **allowBtSplit** is set equal to FALSE
  - **btSplit** is equal to **SPLIT\_BT\_HOR**
  - **cbWidth** is greater than **MaxTbSizeY**
  - $y_0 + \text{cbHeight}$  is greater than  $[[\text{pic\_height\_in\_luma\_samples}]] \text{ } \underline{\text{subpic\_treated\_as\_pic\_flag}[ \text{SubPicIdx} ] ? \text{SubPicBotBoundaryPos} + 1 : \text{pic\_height\_in\_luma\_samples}}$ .

- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
  - $x0 + cbWidth$  is greater than  $[[pic\_width\_in\_luma\_samples]]$  *subpic treated as pic flag[ SubPicIdx ] ? SubPicRightBoundaryPos + 1 : pic width in luma samples*
  - $y0 + cbHeight$  is greater than  $[[pic\_height\_in\_luma\_samples]]$  *subpic treated as pic flag[ SubPicIdx ] ? SubPicBotBoundaryPos + 1 : pic height in luma samples*.
  - $cbWidth$  is greater than  $minQtSize$
- Otherwise, if all of the following conditions are true, allowBtSplit is set equal to FALSE
  - $btSplit$  is equal to  $SPLIT\_BT\_HOR$
  - $x0 + cbWidth$  is greater than  $[[pic\_width\_in\_luma\_samples]]$  *subpic treated as pic flag[ SubPicIdx ] ? SubPicRightBoundaryPos + 1 : pic width in luma samples*
  - $y0 + cbHeight$  is less than or equal to  $[[pic\_height\_in\_luma\_samples]]$  *subpic treated as pic flag[ SubPicIdx ] ? SubPicBotBoundaryPos + 1 : pic height in luma samples*.

#### 6.4.2 Allowed ternary split process

The variable allowTtSplit is derived as follows:

- If one or more of the following conditions are true, allowTtSplit is set equal to FALSE:
  - $cbSize$  is less than or equal to  $2 * MinTtSizeY$
  - $cbWidth$  is greater than  $Min( MaxTbSizeY, maxTtSize )$
  - $cbHeight$  is greater than  $Min( MaxTbSizeY, maxTtSize )$
  - $mttDepth$  is greater than or equal to  $maxMttDepth$
  - $x0 + cbWidth$  is greater than  $[[pic\_width\_in\_luma\_samples]]$  *subpic treated as pic flag[ SubPicIdx ] ? SubPicRightBoundaryPos + 1 : pic width in luma samples*
  - $y0 + cbHeight$  is greater than  $[[pic\_height\_in\_luma\_samples]]$  *subpic treated as pic flag[ SubPicIdx ] ? SubPicBotBoundaryPos + 1 : pic height in luma samples*.
  - $treeType$  is equal to  $DUAL\_TREE\_CHROMA$  and  $( cbWidth / SubWidthC ) * ( cbHeight / SubHeightC )$  is less than or equal to 32
  - $treeType$  is equal to  $DUAL\_TREE\_CHROMA$  and  $modeType$  is equal to  $MODE\_TYPE\_INTRA$
- Otherwise, allowTtSplit is set equal to TRUE.

#### 7.3.8.2 Coding tree unit syntax

| dual_tree_implicit_qt_split( x0, y0, cbSize, cqtDepth ) {                                                                                                                                                                                                                                      | Descriptor |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| ...                                                                                                                                                                                                                                                                                            |            |
| if( x1 < [[pic_width_in_luma_samples]] <u>(subpic treated as pic flag[SubPicIdx] ? SubPicRightBoundaryPos + 1 : pic width in luma samples)</u> )                                                                                                                                               |            |
| dual_tree_implicit_qt_split( x1, y0, cbSize / 2, cqtDepth + 1 )                                                                                                                                                                                                                                |            |
| if( y1 < [[pic_height_in_luma_samples]] <u>(subpic treated as pic flag[SubPicIdx] ? SubPicBotBoundaryPos + 1 : pic height in luma samples)</u> )                                                                                                                                               |            |
| dual_tree_implicit_qt_split( x0, y1, cbSize / 2, cqtDepth + 1 )                                                                                                                                                                                                                                |            |
| if( x1 < [[pic_width_in_luma_samples]] <u>(subpic treated as pic flag[SubPicIdx] ? SubPicRightBoundaryPos + 1 : pic width in luma samples)</u> && y1 < [[pic_height_in_luma_samples]] <u>(subpic treated as pic flag[SubPicIdx] ? SubPicBotBoundaryPos + 1 : pic height in luma samples)</u> ) |            |
| dual_tree_implicit_qt_split( x1, y1, cbSize / 2, cqtDepth + 1 )                                                                                                                                                                                                                                |            |
| } else {                                                                                                                                                                                                                                                                                       |            |
| ...                                                                                                                                                                                                                                                                                            |            |
| }                                                                                                                                                                                                                                                                                              |            |
| }                                                                                                                                                                                                                                                                                              |            |

#### 7.3.8.4 Coding tree syntax

|                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| coding_tree( x0, y0, cbWidth, cbHeight, qgOnY, qgOnC, cbSubdiv, cqtDepth, mttDepth, depthOffset, partIdx, treeTypeCurr, modeTypeCurr ) {                                                                                                                                                                                                                                                                                                                   | <b>Descriptor</b> |
| if( ( allowSplitBtVer    allowSplitBtHor    allowSplitTtVer    allowSplitTtHor    allowSplitQT )<br>&&( x0 + cbWidth <= [[pic_width_in_luma_samples]]<br><u>(subpic treated as pic flag[SubPicIdx] ? SubPicRightBoundaryPos + 1 :</u><br><u>pic_width_in_luma_samples)</u> )<br>&&( y0 + cbHeight <= [[pic_height_in_luma_samples]]<br><u>(subpic treated as pic flag[SubPicIdx] ? SubPicBotBoundaryPos +</u><br><u>1: pic_height_in_luma_samples)</u> ) ) |                   |
| split_cu_flag                                                                                                                                                                                                                                                                                                                                                                                                                                              | ae(v)             |
| if( cu_qp_delta_enabled_flag && qgOnY && cbSubdiv <= cu_qp_delta_subdiv ) {                                                                                                                                                                                                                                                                                                                                                                                |                   |
| ...                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                   |
| depthOffset += ( y0 + cbHeight > [[pic_height_in_luma_samples]]<br><u>(subpic treated as pic flag[SubPicIdx] ? SubPicBotBoundaryPos +</u><br><u>1: pic_height_in_luma_samples)</u> ) ? 1 : 0                                                                                                                                                                                                                                                               |                   |
| y1 = y0 + ( cbHeight / 2 )                                                                                                                                                                                                                                                                                                                                                                                                                                 |                   |
| coding_tree( x0, y0, cbWidth, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 1, cqtDepth, mttDepth + 1, depthOffset, 0, treeType, modeType )                                                                                                                                                                                                                                                                                                                       |                   |
| if( y1 < [[pic_height_in_luma_samples]] <u>(subpic treated as pic flag[SubPicIdx] ?</u><br><u>SubPicBotBoundaryPos + 1: pic_height_in_luma_samples)</u> )                                                                                                                                                                                                                                                                                                  |                   |
| coding_tree( x0, y1, cbWidth, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 1, cqtDepth, mttDepth + 1, depthOffset, 1, treeType, modeType )                                                                                                                                                                                                                                                                                                                       |                   |
| ...                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                   |
| if( x1 < [[pic_width_in_luma_samples]] <u>(subpic treated as pic flag[SubPicIdx] ?</u><br><u>SubPicRightBoundaryPos + 1: pic_width_in_luma_samples)</u> )                                                                                                                                                                                                                                                                                                  |                   |
| coding_tree( x1, y0, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2, cqtDepth + 1, 0, 0, 1, treeType, modeType )                                                                                                                                                                                                                                                                                                                                    |                   |
| if( y1 < [[pic_height_in_luma_samples]] <u>(subpic treated as pic flag[SubPicIdx] ?</u><br><u>SubPicBotBoundaryPos + 1: pic_height_in_luma_samples)</u> )                                                                                                                                                                                                                                                                                                  |                   |
| coding_tree( x0, y1, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2, cqtDepth + 1, 0, 0, 2, treeType, modeType )                                                                                                                                                                                                                                                                                                                                    |                   |
| if( y1 < [[pic_height_in_luma_samples]] <u>(subpic treated as pic flag[SubPicIdx] ?</u><br><u>SubPicBotBoundaryPos + 1: pic_height_in_luma_samples)</u> && x1 < [[pic_width_in_luma_samples]]<br><u>(subpic treated as pic flag[SubPicIdx] ?</u><br><u>SubPicRightBoundaryPos + 1: pic_width_in_luma_samples)</u> )                                                                                                                                        |                   |
| coding_tree( x1, y1, cbWidth / 2, cbHeight / 2, qgOnY, qgOnC, cbSubdiv + 2, cqtDepth + 1, 0, 0, 3, treeType, modeType )                                                                                                                                                                                                                                                                                                                                    |                   |

## [00135] 5.8 Embodiment 8: an example of defining the sub-pictures

|                                                        | Descriptor  |
|--------------------------------------------------------|-------------|
| seq_parameter_set_rbsp( ) {                            |             |
| sps_decoding_parameter_set_id                          | u(4)        |
| ...                                                    |             |
| pic_width_max_in_luma_samples                          | ue(v)       |
| pic_height_max_in_luma_samples                         | ue(v)       |
| [[subpics_present_flag                                 | u(1)        |
| if( subpics_present_flag ) {                           |             |
| max_subpics_minus1                                     | u(8)        |
| subpic_grid_col_width_minus1                           | u(v)        |
| subpic_grid_row_height_minus1                          | u(v)        |
| for( i = 0; i < NumSubPicGridRows; i++ )               |             |
| for( j = 0; j < NumSubPicGridCols; j++ )               |             |
| subpic_grid_idx[ i ][ j ]                              | u(v)        |
| for( i = 0; i <= NumSubPics; i++ ) {                   |             |
| subpic_treated_as_pic_flag[ i ]                        | u(1)        |
| loop_filter_across_subpic_enabled_flag[ i ]            | u(1)        |
| }                                                      |             |
| }]]                                                    |             |
| bit_depth_luma_minus8                                  | ue(v)       |
| ...                                                    |             |
| log2_ctu_size_minus5                                   | u(2)        |
| ...                                                    |             |
| <u>subpics_present_flag</u>                            | <u>u(1)</u> |
| <u>if( subpics_present_flag ) {</u>                    |             |
| <u>num_subpics_minus1</u>                              | <u>u(8)</u> |
| <u>for( i = 0; i &lt;= num_subpics_minus1; i++ ) {</u> |             |
| <u>subpic_ctb_addr x[i]</u>                            | <u>u(8)</u> |
| <u>subpic_ctb_addr y[i]</u>                            | <u>u(8)</u> |
| <u>subpic_ctb_width_minus1[i]</u>                      | <u>u(8)</u> |
| <u>subpic_ctb_height_minus1[i]</u>                     | <u>u(8)</u> |
| <u>subpic_treated_as_pic_flag[ i ]</u>                 | <u>u(1)</u> |
| <u>loop_filter_across_subpic_enabled_flag[ i ]</u>     | <u>u(1)</u> |
| }                                                      |             |
| ...                                                    |             |

## [00136] 5.9 Embodiment 9: an example of defining the sub-pictures

|                                                        | Descriptor   |
|--------------------------------------------------------|--------------|
| seq_parameter_set_rbsp( ) {                            |              |
| sps_decoding_parameter_set_id                          | u(4)         |
| ...                                                    |              |
| pic_width_max_in_luma_samples                          | ue(v)        |
| pic_height_max_in_luma_samples                         | ue(v)        |
| [[subpics_present_flag                                 | u(1)         |
| if( subpics_present_flag ) {                           |              |
| max_subpics_minus1                                     | u(8)         |
| subpic_grid_col_width_minus1                           | u(v)         |
| subpic_grid_row_height_minus1                          | u(v)         |
| for( i = 0; i < NumSubPicGridRows; i++ )               |              |
| for( j = 0; j < NumSubPicGridCols; j++ )               |              |
| subpic_grid_idx[ i ][ j ]                              | u(v)         |
| for( i = 0; i <= NumSubPics; i++ ) {                   |              |
| subpic_treated_as_pic_flag[ i ]                        | u(1)         |
| loop_filter_across_subpic_enabled_flag[ i ]            | u(1)         |
| }                                                      |              |
| }]]                                                    |              |
| bit_depth_luma_minus8                                  | ue(v)        |
| ...                                                    |              |
| log2_ctu_size_minus5                                   | u(2)         |
| ...                                                    |              |
| <u>subpics_present_flag</u>                            | <u>u(1)</u>  |
| <u>if( subpics_present_flag ) {</u>                    |              |
| <u>num_subpics_minus1</u>                              | <u>ue(v)</u> |
| <u>for( i = 0; i &lt;= num_subpics_minus1; i++ ) {</u> |              |
| <u>subpic_ctb_addr x[i]</u>                            | <u>u(8)</u>  |
| <u>subpic_ctb_addr y[i]</u>                            | <u>u(8)</u>  |
| <u>subpic_ctb_width_minus1[i]</u>                      | <u>u(8)</u>  |
| <u>subpic_ctb_height_minus1[i]</u>                     | <u>u(8)</u>  |
| <u>subpic_treated_as_pic_flag[i]</u>                   | <u>u(1)</u>  |
| <u>loop_filter_across_subpic_enabled_flag[i]</u>       | <u>u(1)</u>  |
| }                                                      |              |
| ...                                                    |              |

## [00137] 5.10 Embodiment 10: an example of defining the sub-pictures

|                                                    | Descriptor   |
|----------------------------------------------------|--------------|
| seq_parameter_set_rbsp() {                         |              |
| sps_decoding_parameter_set_id                      | u(4)         |
| ...                                                |              |
| pic_width_max_in_luma_samples                      | ue(v)        |
| pic_height_max_in_luma_samples                     | ue(v)        |
| [[subpics_present_flag                             | u(1)         |
| if( subpics_present_flag ) {                       |              |
| max_subpics_minus1                                 | u(8)         |
| subpic_grid_col_width_minus1                       | u(v)         |
| subpic_grid_row_height_minus1                      | u(v)         |
| for( i = 0; i < NumSubPicGridRows; i++ )           |              |
| for( j = 0; j < NumSubPicGridCols; j++ )           |              |
| subpic_grid_idx[ i ][ j ]                          | u(v)         |
| for( i = 0; i <= NumSubPics; i++ ) {               |              |
| subpic_treated_as_pic_flag[ i ]                    | u(1)         |
| loop_filter_across_subpic_enabled_flag[ i ]        | u(1)         |
| }                                                  |              |
| }]]                                                |              |
| ...                                                |              |
| log2_ctu_size_minus5                               | u(2)         |
| ...                                                |              |
| <u>subpics_present_flag</u>                        | <u>u(1)</u>  |
| <u>if( subpics_present_flag ) {</u>                |              |
| <u>num_subpics_minus2</u>                          | <u>u(v)</u>  |
| subpic_addr_x_length_minus1                        | <u>ue(v)</u> |
| subpic_addr_y_length_minus1                        | <u>ue(v)</u> |
| <u>for( i = 0; i &lt; NumSubPics; i++ ) {</u>      |              |
| <u>subpic_ctb_addr x[i]</u>                        | <u>u(v)</u>  |
| <u>subpic_ctb_addr y[i]</u>                        | <u>u(v)</u>  |
| <u>subpic_ctb_width_minus1[i]</u>                  | <u>u(v)</u>  |
| <u>subpic_ctb_height_minus1[i]</u>                 | <u>u(v)</u>  |
| <u>subpic_treated_as_pic_flag[ i ]</u>             | <u>u(1)</u>  |
| <u>loop_filter_across_subpic_enabled_flag[ i ]</u> | <u>u(1)</u>  |
| }                                                  |              |
| ...                                                |              |

## [00138] 5.11 Embodiment 11: an example of defining the sub-pictures

|                                                    | Descriptor   |
|----------------------------------------------------|--------------|
| seq_parameter_set_rbsp() {                         |              |
| sps_decoding_parameter_set_id                      | u(4)         |
| ...                                                |              |
| pic_width_max_in_luma_samples                      | ue(v)        |
| pic_height_max_in_luma_samples                     | ue(v)        |
| [[subpics_present_flag                             | u(1)         |
| if( subpics_present_flag ) {                       |              |
| max_subpics_minus1                                 | u(8)         |
| subpic_grid_col_width_minus1                       | u(v)         |
| subpic_grid_row_height_minus1                      | u(v)         |
| for( i = 0; i < NumSubPicGridRows; i++ )           |              |
| for( j = 0; j < NumSubPicGridCols; j++ )           |              |
| subpic_grid_idx[ i ][ j ]                          | u(v)         |
| for( i = 0; i <= NumSubPics; i++ ) {               |              |
| subpic_treated_as_pic_flag[ i ]                    | u(1)         |
| loop_filter_across_subpic_enabled_flag[ i ]        | u(1)         |
| }                                                  |              |
| }]]                                                |              |
| ...                                                |              |
| log2_ctu_size_minus5                               | u(2)         |
| ...                                                |              |
| <u>subpics_present_flag</u>                        | <u>u(1)</u>  |
| <u>if( subpics_present_flag ) {</u>                |              |
| <u>num_subpics_minus2</u>                          | <u>u(v)</u>  |
| subpic_addr_x_length_minus1                        | <u>ue(v)</u> |
| subpic_addr_y_length_minus1                        | <u>ue(v)</u> |
| <u>for( i = 0; i &lt; NumSubPics; i++ ) {</u>      |              |
| <u>if( i &lt; NumSubPics - 1 ) {</u>               |              |
| <u>subpic_ctb_addr_x[i]</u>                        | <u>u(v)</u>  |
| <u>subpic_ctb_addr_y[i]</u>                        | <u>u(v)</u>  |
| <u>subpic_ctb_width_minus1[i]</u>                  | <u>u(v)</u>  |
| <u>subpic_ctb_height_minus1[i]</u>                 | <u>u(v)</u>  |
| }                                                  |              |
| <u>subpic_treated_as_pic_flag[ i ]</u>             | <u>u(1)</u>  |
| <u>loop_filter_across_subpic_enabled_flag[ i ]</u> | <u>u(1)</u>  |

|     |  |
|-----|--|
| }   |  |
| ... |  |

$NumSubPics = num\_subpics\_minus2 + 2$ .

## [00139] 5.12 Embodiment: deblocking considering sub-pictures

### Deblocking filter process

#### General

Inputs to this process are the reconstructed picture prior to deblocking, i.e., the array  $recPicture_L$  and, when ChromaArrayType is not equal to 0, the arrays  $recPicture_{Cb}$  and  $recPicture_{Cr}$ .

Outputs of this process are the modified reconstructed picture after deblocking, i.e., the array  $recPicture_L$  and, when ChromaArrayType is not equal to 0, the arrays  $recPicture_{Cb}$  and  $recPicture_{Cr}$ .

The vertical edges in a picture are filtered first. Then the horizontal edges in a picture are filtered with samples modified by the vertical edge filtering process as input. The vertical and horizontal edges in the CTBs of each CTU are processed separately on a coding unit basis. The vertical edges of the coding blocks in a coding unit are filtered starting with the edge on the left-hand side of the coding blocks proceeding through the edges towards the right-hand side of the coding blocks in their geometrical order. The horizontal edges of the coding blocks in a coding unit are filtered starting with the edge on the top of the coding blocks proceeding through the edges towards the bottom of the coding blocks in their geometrical order.

NOTE – Although the filtering process is specified on a picture basis in this Specification, the filtering process can be implemented on a coding unit basis with an equivalent result, provided the decoder properly accounts for the processing dependency order so as to produce the same output values.

The deblocking filter process is applied to all coding subblock edges and transform block edges of a picture, except the following types of edges:

- Edges that are at the boundary of the picture,
- [[Edges that coincide with the boundaries of a subpicture for which  $loop\_filter\_across\_subpic\_enabled\_flag[SubPicIdx]$  is equal to 0,]]
- Edges that coincide with the virtual boundaries of the picture when  $pps\_loop\_filter\_across\_virtual\_boundaries\_disabled\_flag$  is equal to 1,
- Edges that coincide with tile boundaries when  $loop\_filter\_across\_tiles\_enabled\_flag$  is equal to 0,
- Edges that coincide with slice boundaries when  $loop\_filter\_across\_slices\_enabled\_flag$  is equal to 0,
- Edges that coincide with upper or left boundaries of slices with  $slice\_deblocking\_filter\_disabled\_flag$  equal to 1,
- Edges within slices with  $slice\_deblocking\_filter\_disabled\_flag$  equal to 1,
- Edges that do not correspond to 4x4 sample grid boundaries of the luma component,
- Edges that do not correspond to 8x8 sample grid boundaries of the chroma component,
- Edges within the luma component for which both sides of the edge have  $intra\_bdpcm\_luma\_flag$  equal to 1,
- Edges within the chroma components for which both sides of the edge have  $intra\_bdpcm\_chroma\_flag$  equal to 1,

- Edges of chroma subblocks that are not edges of the associated transform unit.

...

#### Deblocking filter process for one direction

Inputs to this process are:

- the variable `treeType` specifying whether the luma (`DUAL_TREE_LUMA`) or chroma components (`DUAL_TREE_CHROMA`) are currently processed,
- when `treeType` is equal to `DUAL_TREE_LUMA`, the reconstructed picture prior to deblocking, i.e., the array `recPictureL`,
- when `ChromaArrayType` is not equal to 0 and `treeType` is equal to `DUAL_TREE_CHROMA`, the arrays `recPictureCb` and `recPictureCr`,
- a variable `edgeType` specifying whether a vertical (`EDGE_VER`) or a horizontal (`EDGE_HOR`) edge is filtered.

Outputs of this process are the modified reconstructed picture after deblocking, i.e:

- when `treeType` is equal to `DUAL_TREE_LUMA`, the array `recPictureL`,
- when `ChromaArrayType` is not equal to 0 and `treeType` is equal to `DUAL_TREE_CHROMA`, the arrays `recPictureCb` and `recPictureCr`.

The variables `firstCompIdx` and `lastCompIdx` are derived as follows:

$$\text{firstCompIdx} = (\text{treeType} == \text{DUAL\_TREE\_CHROMA}) ? 1 : 0 \quad (8-1010)$$

$$\text{lastCompIdx} = (\text{treeType} == \text{DUAL\_TREE\_LUMA} \mid \mid \text{ChromaArrayType} == 0) ? 0 : 2 \quad (8-1011)$$

For each coding unit and each coding block per colour component of a coding unit indicated by the colour component index `cIdx` ranging from `firstCompIdx` to `lastCompIdx`, inclusive, with coding block width `nCbW`, coding block height `nCbH` and location of top-left sample of the coding block ( `xCb`, `yCb` ), when `cIdx` is equal to 0, or when `cIdx` is not equal to 0 and `edgeType` is equal to `EDGE_VER` and `xCb % 8` is equal 0, or when `cIdx` is not equal to 0 and `edgeType` is equal to `EDGE_HOR` and `yCb % 8` is equal to 0, the edges are filtered by the following ordered steps:

2. The variable `filterEdgeFlag` is derived as follows:
  - If `edgeType` is equal to `EDGE_VER` and one or more of the following conditions are true, `filterEdgeFlag` is set equal to 0:
    - The left boundary of the current coding block is the left boundary of the picture.
    - [[The left boundary of the current coding block is the left or right boundary of the subpicture and `loop_filter_across_subpic_enabled_flag[ SubPicIdx ]` is equal to 0.]]
    - The left boundary of the current coding block is the left boundary of the tile and `loop_filter_across_tiles_enabled_flag` is equal to 0.
    - The left boundary of the current coding block is the left boundary of the slice and `loop_filter_across_slices_enabled_flag` is equal to 0.
    - The left boundary of the current coding block is one of the vertical virtual boundaries of the picture and `VirtualBoundariesDisabledFlag` is equal to 1.

- Otherwise, if edgeType is equal to EDGE\_HOR and one or more of the following conditions are true, the variable filterEdgeFlag is set equal to 0:
  - The top boundary of the current luma coding block is the top boundary of the picture.
  - [[The top boundary of the current coding block is the top or bottom boundary of the subpicture and loop\_filter\_across\_subpic\_enabled\_flag[ SubPicIdx ] is equal to 0.]]
  - The top boundary of the current coding block is the top boundary of the tile and loop\_filter\_across\_tiles\_enabled\_flag is equal to 0.
  - The top boundary of the current coding block is the top boundary of the slice and loop\_filter\_across\_slices\_enabled\_flag is equal to 0.
  - The top boundary of the current coding block is one of the horizontal virtual boundaries of the picture and VirtualBoundariesDisabledFlag is equal to 1.
- Otherwise, filterEdgeFlag is set equal to 1.

...

#### ***Filtering process for a luma sample using short filters***

Inputs to this process are:

- the sample values  $p_i$  and  $q_i$  with  $i = 0..3$ ,
- the locations of  $p_i$  and  $q_i$ ,  $(xP_i, yP_i)$  and  $(xQ_i, yQ_i)$  with  $i = 0..2$ ,
- a variable dE,
- the variables dEp and dEq containing decisions to filter samples  $p_1$  and  $q_1$ , respectively,
- a variable  $t_c$ .

Outputs of this process are:

- the number of filtered samples nDp and nDq,
- the filtered sample values  $p'_i$  and  $q'_j$  with  $i = 0..nDp - 1, j = 0..nDq - 1$ .

Depending on the value of dE, the following applies:

- If the variable dE is equal to 2, nDp and nDq are both set equal to 3 and the following strong filtering applies:

$$p_0' = \text{Clip3}(p_0 - 3 * t_c, p_0 + 3 * t_c, (p_2 + 2 * p_1 + 2 * p_0 + 2 * q_0 + q_1 + 4) >> 3) \quad (8-1150)$$

$$p_1' = \text{Clip3}(p_1 - 2 * t_c, p_1 + 2 * t_c, (p_2 + p_1 + p_0 + q_0 + 2) >> 2) \quad (8-1151)$$

$$p_2' = \text{Clip3}(p_2 - 1 * t_c, p_2 + 1 * t_c, (2 * p_3 + 3 * p_2 + p_1 + p_0 + q_0 + 4) >> 3) \quad (8-1152)$$

$$q_0' = \text{Clip3}(q_0 - 3 * t_c, q_0 + 3 * t_c, (p_1 + 2 * p_0 + 2 * q_0 + 2 * q_1 + q_2 + 4) >> 3) \quad (8-1153)$$

$$q_1' = \text{Clip3}(q_1 - 2 * t_c, q_1 + 2 * t_c, (p_0 + q_0 + q_1 + q_2 + 2) >> 2) \quad (8-1154)$$

$$q_2' = \text{Clip3}(q_2 - 1 * t_c, q_2 + 1 * t_c, (p_0 + q_0 + q_1 + 3 * q_2 + 2 * q_3 + 4) >> 3) \quad (8-1155)$$

- Otherwise, nDp and nDq are set both equal to 0 and the following weak filtering applies:

- The following applies:

$$\Delta = (9 * (q_0 - p_0) - 3 * (q_1 - p_1) + 8) >> 4 \quad (8-1156)$$

- When Abs( $\Delta$ ) is less than  $t_c * 10$ , the following ordered steps apply:

- The filtered sample values  $p_0'$  and  $q_0'$  are specified as follows:

$$\Delta = \text{Clip3}(-t_c, t_c, \Delta) \quad (8-1157)$$

$$p_0' = \text{Clip1}(p_0 + \Delta) \quad (8-1158)$$

$$q_0' = \text{Clip1}(q_0 - \Delta) \quad (8-1159)$$

- When dEp is equal to 1, the filtered sample value  $p_1'$  is specified as follows:

$$\Delta p = \text{Clip3}(-(t_c \gg 1), t_c \gg 1, ((p_2 + p_0 + 1) \gg 1) - p_1 + \Delta) \gg 1 \quad (8-1160)$$

$$p_1' = \text{Clip1}(p_1 + \Delta p) \quad (8-1161)$$

- When dEq is equal to 1, the filtered sample value  $q_1'$  is specified as follows:

$$\Delta q = \text{Clip3}(-(t_c \gg 1), t_c \gg 1, ((q_2 + q_0 + 1) \gg 1) - q_1 - \Delta) \gg 1 \quad (8-1162)$$

$$q_1' = \text{Clip1}(q_1 + \Delta q) \quad (8-1163)$$

- nDp is set equal to dEp + 1 and nDq is set equal to dEq + 1.

When nDp is greater than 0 and pred\_mode\_plt\_flag of the coding unit that includes the coding block containing the sample  $p_0$  is equal to 1, nDp is set equal to 0

When nDq is greater than 0 and pred\_mode\_plt\_flag of the coding unit that includes the coding block containing the sample  $q_0$  is equal to 1, nDq is set equal to 0

When nDp is greater than 0 and loop\_filter\_across\_subpic\_enabled\_flag[subPicIdxP] is equal to 0, nDp is set equal to 0, wherein subPicIdxP is the subpicture index of the subpicture containing the sample  $p_0$ .

When nDq is greater than 0 and loop\_filter\_across\_subpic\_enabled\_flag[subPicIdxQ] is equal to 0, nDq is set equal to 0, wherein subPicIdxQ is the subpicture index of the subpicture containing the sample  $q_0$ .

#### Filtering process for a luma sample using long filters

Inputs to this process are:

- the variables maxFilterLengthP and maxFilterLengthQ,
- the sample values  $p_i$  and  $q_j$  with  $i = 0..maxFilterLengthP$  and  $j = 0..maxFilterLengthQ$ ,
- the locations of  $p_i$  and  $q_j$ ,  $(x_{P_i}, y_{P_i})$  and  $(x_{Q_j}, y_{Q_j})$  with  $i = 0..maxFilterLengthP - 1$  and  $j = 0..maxFilterLengthQ - 1$ ,
- a variable  $t_c$ .

Outputs of this process are:

- the filtered sample values  $p_i'$  and  $q_j'$  with  $i = 0..maxFilterLengthP - 1$ ,  $j = 0..maxFilterLengthQ - 1$ .

The variable refMiddle is derived as follows:

- If maxFilterLengthP is equal to maxFilterLengthQ and maxFilterLengthP is equal to 5, the following applies:

$$\text{refMiddle} = (p_4 + p_3 + 2 * (p_2 + p_1 + p_0 + q_0 + q_1 + q_2) + q_3 + q_4 + 8) \gg 4 \quad (8-1164)$$

- Otherwise, if maxFilterLengthP is equal to maxFilterLengthQ and maxFilterLengthP is not equal to 5, the following applies:

$$\text{refMiddle} = (p_6 + p_5 + p_4 + p_3 + p_2 + p_1 + 2 * (p_0 + q_0) + q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + 8) \gg 4 \quad (8-1165)$$

- Otherwise, if one of the following conditions are true,

- maxFilterLengthQ is equal to 7 and maxFilterLengthP is equal to 5,
- maxFilterLengthQ is equal to 5 and maxFilterLengthP is equal to 7,

the following applies:

$$\text{refMiddle} = (p_5 + p_4 + p_3 + p_2 + 2 * (p_1 + p_0 + q_0 + q_1) + q_2 + q_3 + q_4 + q_5 + 8) \gg 4 \quad (8-1166)$$

- Otherwise, if one of the following conditions are true,
  - maxFilterLengthQ is equal to 5 and maxFilterLengthP is equal to 3,
  - maxFilterLengthQ is equal to 3 and maxFilterLengthP is equal to 5,
 the following applies:

$$\text{refMiddle} = (p_3 + p_2 + p_1 + p_0 + q_0 + q_1 + q_2 + q_3 + 4) \gg 3 \quad (8-1167)$$

- Otherwise, if maxFilterLengthQ is equal to 7 and maxFilterLengthP is equal to 3, the following applies:

$$\text{refMiddle} = (2 * (p_2 + p_1 + p_0 + q_0) + p_0 + p_1 + q_1 + q_2 + q_3 + q_4 + q_5 + q_6 + 8) \gg 4 \quad (8-1168)$$

- Otherwise, the following applies:

$$\text{refMiddle} = (p_6 + p_5 + p_4 + p_3 + p_2 + p_1 + 2 * (q_2 + q_1 + q_0 + p_0) + q_0 + q_1 + 8) \gg 4 \quad (8-1169)$$

The variables refP and refQ are derived as follows:

$$\text{refP} = (p_{\text{maxFilterLengthP}} + p_{\text{maxFilterLengthP}-1} + 1) \gg 1 \quad (8-1179)$$

$$\text{refQ} = (q_{\text{maxFilterLengthQ}} + q_{\text{maxFilterLengthQ}-1} + 1) \gg 1 \quad (8-1171)$$

The variables  $f_i$  and  $t_{cPD_i}$  are defined as follows:

- If maxFilterLengthP is equal to 7, the following applies:

$$f_{0..6} = \{ 59, 50, 41, 32, 23, 14, 5 \} \quad (8-1172)$$

$$t_{cPD_{0..6}} = \{ 6, 5, 4, 3, 2, 1, 1 \} \quad (8-1173)$$

- Otherwise, if maxFilterLengthP is equal to 5, the following applies:

$$f_{0..4} = \{ 58, 45, 32, 19, 6 \} \quad (8-1174)$$

$$t_{cPD_{0..4}} = \{ 6, 5, 4, 3, 2 \} \quad (8-1175)$$

- Otherwise, the following applies:

$$f_{0..2} = \{ 53, 32, 11 \} \quad (8-1176)$$

$$t_{cPD_{0..2}} = \{ 6, 4, 2 \} \quad (8-1177)$$

The variables  $g_j$  and  $t_{cQD_j}$  are defined as follows:

- If maxFilterLengthQ is equal to 7, the following applies:

$$g_{0..6} = \{ 59, 50, 41, 32, 23, 14, 5 \} \quad (8-1178)$$

$$t_{cQD_{0..6}} = \{ 6, 5, 4, 3, 2, 1, 1 \} \quad (8-1179)$$

- Otherwise, if maxFilterLengthQ is equal to 5, the following applies:

$$g_{0..4} = \{ 58, 45, 32, 19, 6 \} \quad (8-1180)$$

$$t_{cQD_{0..4}} = \{ 6, 5, 4, 3, 2 \} \quad (8-1181)$$

- Otherwise, the following applies:

$$g_{0..2} = \{ 53, 32, 11 \} \quad (8-1182)$$

$$t_{cQD_{0..2}} = \{ 6, 4, 2 \} \quad (8-1183)$$

The filtered sample values  $p_i'$  and  $q_j'$  with  $i = 0..\text{maxFilterLengthP} - 1$  and  $j = 0..\text{maxFilterLengthQ} - 1$  are derived as follows:

$$p_i' = \text{Clip3}(p_i - (t_{cPD_i} * t_{cPD_i}) \gg 1, p_i + (t_{cPD_i} * t_{cPD_i}) \gg 1, (\text{refMiddle} * f_i + \text{refP} * (64 - f_i) + 32) \gg 6) \quad (8-1184)$$

$$q_j' = \text{Clip3}(q_j - (t_{cQD_j} * t_{cQD_j}) \gg 1, q_j + (t_{cQD_j} * t_{cQD_j}) \gg 1, (\text{refMiddle} * g_j + \text{refQ} * (64 - g_j) + 32) \gg 6) \quad (8-1185)$$

When `pred_mode_plt_flag` of the coding unit that includes the coding block containing the sample  $p_i$  is equal to 1, the filtered sample value,  $p_i'$  is substituted by the corresponding input sample value  $p_i$  with  $i = 0..maxFilterLengthP - 1$ .

When `pred_mode_plt_flag` of the coding unit that includes the coding block containing the sample  $q_i$  is equal to 1, the filtered sample value,  $q_i'$  is substituted by the corresponding input sample value  $q_i$  with  $j = 0..maxFilterLengthQ - 1$ .

When `loop_filter_across_subpic_enabled_flag[subPicIdxP]` is equal to 0, wherein `subPicIdxP` is the subpicture index of the subpicture containing the sample  $p_0$ , the filtered sample value,  $p_i'$  is substituted by the corresponding input sample value  $p_i$  with  $i = 0..maxFilterLengthP - 1$ .

When `loop_filter_across_subpic_enabled_flag[subPicIdxO]` is equal to 0, wherein `subPicIdxO` is the subpicture index of the subpicture containing the sample  $q_0$ , the filtered sample value,  $q_i'$  is substituted by the corresponding input sample value  $q_i$  with  $j = 0..maxFilterLengthO - 1$ .

#### *Filtering process for a chroma sample*

This process is only invoked when `ChromaArrayType` is not equal to 0.

Inputs to this process are:

- the variable `maxFilterLength`,
- the chroma sample values  $p_i$  and  $q_i$  with  $i = 0..maxFilterLengthCbCr$ ,
- the chroma locations of  $p_i$  and  $q_i$ ,  $(xP_i, yP_i)$  and  $(xQ_i, yQ_i)$  with  $i = 0..maxFilterLengthCbCr - 1$ ,
- a variable  $t_c$ .

Outputs of this process are the filtered sample values  $p_i'$  and  $q_i'$  with  $i = 0..maxFilterLengthCbCr - 1$ .

The filtered sample values  $p_i'$  and  $q_i'$  with  $i = 0..maxFilterLengthCbCr - 1$  are derived as follows:

- If `maxFilterLengthCbCr` is equal to 3, the following strong filtering applies:

$$p_0' = \text{Clip3}(p_0 - t_c, p_0 + t_c, (p_3 + p_2 + p_1 + 2 * p_0 + q_0 + q_1 + q_2 + 4) >> 3) \quad (8-1186)$$

$$p_1' = \text{Clip3}(p_1 - t_c, p_1 + t_c, (2 * p_3 + p_2 + 2 * p_1 + p_0 + q_0 + q_1 + 4) >> 3) \quad (8-1187)$$

$$p_2' = \text{Clip3}(p_2 - t_c, p_2 + t_c, (3 * p_3 + 2 * p_2 + p_1 + p_0 + q_0 + 4) >> 3) \quad (8-1188)$$

$$q_0' = \text{Clip3}(q_0 - t_c, q_0 + t_c, (p_2 + p_1 + p_0 + 2 * q_0 + q_1 + q_2 + q_3 + 4) >> 3) \quad (8-1189)$$

$$q_1' = \text{Clip3}(q_1 - t_c, q_1 + t_c, (p_1 + p_0 + q_0 + 2 * q_1 + q_2 + 2 * q_3 + 4) >> 3) \quad (8-1190)$$

$$q_2' = \text{Clip3}(q_2 - t_c, q_2 + t_c, (p_0 + q_0 + q_1 + 2 * q_2 + 3 * q_3 + 4) >> 3) \quad (8-1191)$$

- Otherwise, the following weak filtering applies:

$$\Delta = \text{Clip3}(-t_c, t_c, (((q_0 - p_0) << 2) + p_1 - q_1 + 4) >> 3) \quad (8-1192)$$

$$p_0' = \text{Clip1}(p_0 + \Delta) \quad (8-1193)$$

$$q_0' = \text{Clip1}(q_0 - \Delta) \quad (8-1194)$$

When `pred_mode_plt_flag` of the coding unit that includes the coding block containing the sample  $p_i$  is equal to 1, the filtered sample value,  $p_i'$  is substituted by the corresponding input sample value  $p_i$  with  $i = 0..maxFilterLengthCbCr - 1$ .

When `pred_mode_plt_flag` of the coding unit that includes the coding block containing the sample  $q_i$  is equal to 1, the filtered sample value,  $q_i'$  is substituted by the corresponding input sample value  $q_i$  with  $i = 0..maxFilterLengthCbCr - 1$ :

When `loop_filter_across_subpic_enabled_flag[subPicIdxP]` is equal to 0, wherein `subPicIdxP` is the subpicture index of the subpicture containing the sample  $p_0$ , the filtered sample value,  $p_i'$  is substituted by the corresponding input sample value  $p_i$  with  $i = 0..maxFilterLengthCbCr - 1$ .

When `loop_filter_across_subpic_enabled_flag[subPicIdxO]` is equal to 0, wherein `subPicIdxO` is the subpicture index of the subpicture containing the sample  $q_0$ , the filtered sample value,  $q_i'$  is substituted by the corresponding input sample value  $q_i$  with  $i = 0..maxFilterLengthCbCr - 1$ .

[00140] FIG. 3 is a block diagram of a video processing apparatus 300. The apparatus 300 may be used to implement one or more of the methods described herein. The apparatus 300 may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus 300 may include one or more processors 312, one or more memories 314 and video processing hardware 316. The processor(s) 312 may be configured to implement one or more methods described in the present document. The memory (memories) 314 may be used for storing data and code used for implementing the methods and techniques described herein. The video processing hardware 316 may be used to implement, in hardware circuitry, some techniques described in the present document.

[00141] FIG. 4 is a flowchart for a method 400 of processing a video. The method 400 includes determining (402), for a video block in a first video region of a video, whether a position at which a temporal motion vector predictor is determined for a conversion between the video block and a bitstream representation of the current video block using an affine mode is within a second video region, and performing (404) the conversion based on the determining.

[00142] The following solutions may be implemented as preferred solutions in some embodiments.

[00143] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 1).

[00144] 1. A method of video processing, comprising: determining, for a video block in a first video region of a video, whether a position at which a temporal motion vector predictor is determined for a conversion between the video block and a bitstream representation of the current video block using an affine mode is within a second video region; and performing the conversion based on the determining.

[00145] 2. The method of solution 1, wherein the video block is covered by the first region and the second region.

[00146] 3. The method of any of solutions 1-2, wherein, in case that the position of the temporal motion vector predictor is outside of the second video region, then the temporal motion vector predictor is marked as unavailable and is unused in the conversion.

[00147] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 2).

[00148] 4. A method of video processing, comprising: determining, for a video block in a first video region of a video, whether a position at which an integer sample in a reference picture is fetched for a conversion between the video block and a bitstream representation of the current video block is within a second video region, wherein the reference picture is not used in an interpolation process during the conversion; and performing the conversion based on the determining.

[00149] 5. The method of solution 4, wherein the video block is covered by the first region and the second region.

[00150] 6. The method of any of solutions 4-5, wherein, in case that the position of the sample is outside of the second video region, then the sample is marked as unavailable and is unused in the conversion.

[00151] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 3).

[00152] 7. A method of video processing, comprising: determining, for a video block in a first video region of a video, whether a position at which a reconstructed luma sample value is fetched for a conversion between the video block and a bitstream representation of the current video block is within a second video region; and performing the conversion based on the determining.

[00153] 8. The method of solution 7, wherein the luma sample is covered by the first region and the second region.

[00154] 9. The method of any of solutions 7-8, wherein, in case that the position of the luma sample is outside of the second video region, then the luma sample is marked as unavailable and is unused in the conversion.

[00155] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 4).

[00156] 10. A method of video processing, comprising: determining, for a video block in a first video region of a video, whether a position at which a check regarding splitting, depth derivation or split flag signaling for the video block is performed during a conversion between the video block and a bitstream representation of the current video block is within a second video region; and performing the conversion based on the determining.

[00157] 11. The method of solution 10, wherein the position is covered by the first region and the second region.

[00158] 12. The method of any of solutions 10-11, wherein, in case that the position is outside of the second video region, then the luma sample is marked as unavailable and is unused in the conversion.

[00159] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 8).

[00160] 13. A method of video processing, comprising: performing a conversion between a video comprising one or more video pictures comprising one or more video blocks, and a coded representation of the video, wherein the coded representation complies with a coding syntax requirement that the conversion is not to use sub-picture coding/decoding and a dynamic resolution conversion coding/decoding tool or a reference picture resampling tool within a video unit.

[00161] 14. The method of solution 13, wherein the video unit corresponds to a sequence of the one or more video pictures.

[00162] 15. The method of any of solutions 13-14, wherein the dynamic resolution conversion coding/decoding tool comprises an adaptive resolution conversion coding/decoding tool.

[00163] 16. The method of any of solutions 13-14, wherein the dynamic resolution conversion coding/decoding tool comprises a dynamic resolution conversion coding/decoding tool.

[00164] 17. The method of any of solutions 13-16, wherein the coded representation indicates that the video unit complies with the coding syntax requirement.

[00165] 18. The method of solution 17, wherein the coded representation indicates that the video unit uses sub-picture coding.

[00166] 19. The method of solution 17, wherein the coded representation indicates that the video unit uses the dynamic resolution conversion coding/decoding tool or the reference picture resampling tool.

[00167] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 10).

[00168] 20. The method of any of solutions 1-19, wherein the second video region comprises a video sub-picture and wherein boundaries of the second video region and another video region is also a boundary between two coding tree units.

[00169] 21. The method of any of solutions 1-19, wherein the second video region comprises a video sub-picture and wherein boundaries of the second video region and another video region is also a boundary between two coding tree units.

[00170] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 11).

[00171] 22. The method of any of solutions 1-21, wherein the first video region and the second video region have rectangular shapes.

[00172] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 12).

[00173] 23. The method of any of solutions 1-22, wherein the first video region and the second video region are non-overlapping.

[00174] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 13).

[00175] 24. The method of any of solutions 1-23, wherein the video picture is divided into video regions such that a pixel in the video picture is covered by one and only one video region.

[00176] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 15).

[00177] 25. The method of any of solutions 1-24, wherein the video picture is split into the first video region and the second video region due to the video picture being in a specific layer of the video sequence.

[00178] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 10).

[00179] 26. A method of video processing, comprising: performing a conversion between a video comprising one or more video pictures comprising one or more video blocks, and a coded representation of the video, wherein the coded representation complies with a coding syntax

requirement that a first syntax element `subpic_grid_idx[i][j]` is not larger than a second syntax element `max_subpics_minus1`.

[00180] 27. The method of solution 26, wherein a codeword representing the first syntax element is not larger than a codeword representing the second syntax element.

[00181] 28. The method of any of solutions 1-27, wherein the first video region comprises a video sub-picture.

[00182] 29. The method of any of solutions 1-28, wherein the second video region comprises a video sub-picture.

[00183] 30. The method of any of solutions 1 to 29, wherein the conversion comprises encoding the video into the coded representation.

[00184] 31. The method of any of solutions 1 to 29, wherein the conversion comprises decoding the coded representation to generate pixel values of the video.

[00185] 32. A video decoding apparatus comprising a processor configured to implement a method recited in one or more of solutions 1 to 31.

[00186] 33. A video encoding apparatus comprising a processor configured to implement a method recited in one or more of solutions 1 to 31.

[00187] 34. A computer program product having computer code stored thereon, the code, when executed by a processor, causes the processor to implement a method recited in any of solutions 1 to 31.

[00188] 35. A method, apparatus or system described in the present document.

[00189] FIG. 8 is a block diagram showing an example video processing system 800 in which various techniques disclosed herein may be implemented. Various implementations may include some or all of the components of the system 800. The system 800 may include input 802 for receiving video content. The video content may be received in a raw or uncompressed format, e.g., 8 or 10 bit multi-component pixel values, or may be in a compressed or encoded format. The input 802 may represent a network interface, a peripheral bus interface, or a storage interface. Examples of network interface include wired interfaces such as Ethernet, passive optical network (PON), etc. and wireless interfaces such as Wi-Fi or cellular interfaces.

[00190] The system 800 may include a coding component 804 that may implement the various coding or encoding methods described in the present document. The coding component 804 may reduce the average bitrate of video from the input 802 to the output of the coding component 804

to produce a coded representation of the video. The coding techniques are therefore sometimes called video compression or video transcoding techniques. The output of the coding component 804 may be either stored, or transmitted via a communication connected, as represented by the component 806. The stored or communicated bitstream (or coded) representation of the video received at the input 802 may be used by the component 808 for generating pixel values or displayable video that is sent to a display interface 810. The process of generating user-viewable video from the bitstream representation is sometimes called video decompression. Furthermore, while certain video processing operations are referred to as “coding” operations or tools, it will be appreciated that the coding tools or operations are used at an encoder and corresponding decoding tools or operations that reverse the results of the coding will be performed by a decoder.

**[00191]** Examples of a peripheral bus interface or a display interface may include universal serial bus (USB) or high definition multimedia interface (HDMI) or Displayport, and so on. Examples of storage interfaces include SATA (serial advanced technology attachment), PCI, IDE interface, and the like. The techniques described in the present document may be embodied in various electronic devices such as mobile phones, laptops, smartphones or other devices that are capable of performing digital data processing and/or video display.

**[00192]** FIG. 9 is a block diagram that illustrates an example video coding system 100 that may utilize the techniques of this disclosure.

**[00193]** As shown in FIG. 9, video coding system 100 may include a source device 110 and a destination device 120. Source device 110 generates encoded video data which may be referred to as a video encoding device. Destination device 120 may decode the encoded video data generated by source device 110 which may be referred to as a video decoding device.

**[00194]** Source device 110 may include a video source 112, a video encoder 114, and an input/output (I/O) interface 116.

**[00195]** Video source 112 may include a source such as a video capture device, an interface to receive video data from a video content provider, and/or a computer graphics system for generating video data, or a combination of such sources. The video data may comprise one or more pictures. Video encoder 114 encodes the video data from video source 112 to generate a bitstream. The bitstream may include a sequence of bits that form a coded representation of the video data. The bitstream may include coded pictures and associated data. The coded picture is a coded

representation of a picture. The associated data may include sequence parameter sets, picture parameter sets, and other syntax structures. I/O interface 116 may include a modulator/demodulator (modem) and/or a transmitter. The encoded video data may be transmitted directly to destination device 120 via I/O interface 116 through network 130a. The encoded video data may also be stored onto a storage medium/server 130b for access by destination device 120.

[00196] Destination device 120 may include an I/O interface 126, a video decoder 124, and a display device 122.

[00197] I/O interface 126 may include a receiver and/or a modem. I/O interface 126 may acquire encoded video data from the source device 110 or the storage medium/ server 130b. Video decoder 124 may decode the encoded video data. Display device 122 may display the decoded video data to a user. Display device 122 may be integrated with the destination device 120, or may be external to destination device 120 which be configured to interface with an external display device.

[00198] Video encoder 114 and video decoder 124 may operate according to a video compression standard, such as the High Efficiency Video Coding (HEVC) standard, Versatile Video Coding (VVC) standard and other current and/or further standards.

[00199] FIG. 10 is a block diagram illustrating an example of video encoder 200, which may be video encoder 114 in the system 100 illustrated in FIG. 9.

[00200] Video encoder 200 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 10, video encoder 200 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of video encoder 200. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[00201] The functional components of video encoder 200 may include a partition unit 201, a predication unit 202 which may include a mode select unit 203, a motion estimation unit 204, a motion compensation unit 205 and an intra prediction unit 206, a residual generation unit 207, a transform unit 208, a quantization unit 209, an inverse quantization unit 210, an inverse transform unit 211, a reconstruction unit 212, a buffer 213, and an entropy encoding unit 214.

[00202] In other examples, video encoder 200 may include more, fewer, or different functional components. In an example, predication unit 202 may include an intra block copy (IBC) unit. The IBC unit may perform predication in an IBC mode in which at least one reference picture is a picture where the current video block is located.

[00203] Furthermore, some components, such as motion estimation unit 204 and motion compensation unit 205 may be highly integrated, but are represented in the example of FIG. 5 separately for purposes of explanation.

[00204] Partition unit 201 may partition a picture into one or more video blocks. Video encoder 200 and video decoder 300 may support various video block sizes.

[00205] Mode select unit 203 may select one of the coding modes, intra or inter, e.g., based on error results, and provide the resulting intra- or inter-coded block to a residual generation unit 207 to generate residual block data and to a reconstruction unit 212 to reconstruct the encoded block for use as a reference picture. In some example, Mode select unit 203 may select a combination of intra and inter predication (CIIP) mode in which the predication is based on an inter predication signal and an intra predication signal. Mode select unit 203 may also select a resolution for a motion vector (e.g., a sub-pixel or integer pixel precision) for the block in the case of inter-predication.

[00206] To perform inter prediction on a current video block, motion estimation unit 204 may generate motion information for the current video block by comparing one or more reference frames from buffer 213 to the current video block. Motion compensation unit 205 may determine a predicted video block for the current video block based on the motion information and decoded samples of pictures from buffer 213 other than the picture associated with the current video block.

[00207] Motion estimation unit 204 and motion compensation unit 205 may perform different operations for a current video block, for example, depending on whether the current video block is in an I slice, a P slice, or a B slice.

[00208] In some examples, motion estimation unit 204 may perform uni-directional prediction for the current video block, and motion estimation unit 204 may search reference pictures of list 0 or list 1 for a reference video block for the current video block. Motion estimation unit 204 may then generate a reference index that indicates the reference picture in list 0 or list 1 that contains the reference video block and a motion vector that indicates a spatial displacement between the current video block and the reference video block. Motion estimation unit 204 may output the reference index, a prediction direction indicator, and the motion vector as the motion information of the current video block. Motion compensation unit 205 may generate the predicted video block of the current block based on the reference video block indicated by the motion information of the current video block.

[00209] In other examples, motion estimation unit 204 may perform bi-directional prediction for the current video block, motion estimation unit 204 may search the reference pictures in list 0 for a reference video block for the current video block and may also search the reference pictures in list 1 for another reference video block for the current video block. Motion estimation unit 204 may then generate reference indexes that indicate the reference pictures in list 0 and list 1 containing the reference video blocks and motion vectors that indicate spatial displacements between the reference video blocks and the current video block. Motion estimation unit 204 may output the reference indexes and the motion vectors of the current video block as the motion information of the current video block. Motion compensation unit 205 may generate the predicted video block of the current video block based on the reference video blocks indicated by the motion information of the current video block.

[00210] In some examples, motion estimation unit 204 may output a full set of motion information for decoding processing of a decoder.

[00211] In some examples, motion estimation unit 204 may do not output a full set of motion information for the current video. Rather, motion estimation unit 204 may signal the motion information of the current video block with reference to the motion information of another video block. For example, motion estimation unit 204 may determine that the motion information of the current video block is sufficiently similar to the motion information of a neighboring video block.

[00212] In one example, motion estimation unit 204 may indicate, in a syntax structure associated with the current video block, a value that indicates to the video decoder 300 that the current video block has the same motion information as the another video block.

[00213] In another example, motion estimation unit 204 may identify, in a syntax structure associated with the current video block, another video block and a motion vector difference (MVD). The motion vector difference indicates a difference between the motion vector of the current video block and the motion vector of the indicated video block. The video decoder 300 may use the motion vector of the indicated video block and the motion vector difference to determine the motion vector of the current video block.

[00214] As discussed above, video encoder 200 may predictively signal the motion vector. Two examples of predictive signaling techniques that may be implemented by video encoder 200 include advanced motion vector predication (AMVP) and merge mode signaling.

[00215] Intra prediction unit 206 may perform intra prediction on the current video block. When

intra prediction unit 206 performs intra prediction on the current video block, intra prediction unit 206 may generate prediction data for the current video block based on decoded samples of other video blocks in the same picture. The prediction data for the current video block may include a predicted video block and various syntax elements.

[00216] Residual generation unit 207 may generate residual data for the current video block by subtracting (e.g., indicated by the minus sign) the predicted video block(s) of the current video block from the current video block. The residual data of the current video block may include residual video blocks that correspond to different sample components of the samples in the current video block.

[00217] In other examples, there may be no residual data for the current video block for the current video block, for example in a skip mode, and residual generation unit 207 may not perform the subtracting operation.

[00218] Transform processing unit 208 may generate one or more transform coefficient video blocks for the current video block by applying one or more transforms to a residual video block associated with the current video block.

[00219] After transform processing unit 208 generates a transform coefficient video block associated with the current video block, quantization unit 209 may quantize the transform coefficient video block associated with the current video block based on one or more quantization parameter (QP) values associated with the current video block.

[00220] Inverse quantization unit 210 and inverse transform unit 211 may apply inverse quantization and inverse transforms to the transform coefficient video block, respectively, to reconstruct a residual video block from the transform coefficient video block. Reconstruction unit 212 may add the reconstructed residual video block to corresponding samples from one or more predicted video blocks generated by the predication unit 202 to produce a reconstructed video block associated with the current block for storage in the buffer 213.

[00221] After reconstruction unit 212 reconstructs the video block, loop filtering operation may be performed reduce video blocking artifacts in the video block.

[00222] Entropy encoding unit 214 may receive data from other functional components of the video encoder 200. When entropy encoding unit 214 receives the data, entropy encoding unit 214 may perform one or more entropy encoding operations to generate entropy encoded data and output a bitstream that includes the entropy encoded data.

[00223] FIG. 11 is a block diagram illustrating an example of video decoder 300 which may be video decoder 114 in the system 100 illustrated in FIG. 9.

[00224] The video decoder 300 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 11, the video decoder 300 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of the video decoder 300. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[00225] In the example of FIG. 11, video decoder 300 includes an entropy decoding unit 301, a motion compensation unit 302, an intra prediction unit 303, an inverse quantization unit 304, an inverse transformation unit 305, and a reconstruction unit 306 and a buffer 307. Video decoder 300 may, in some examples, perform a decoding pass generally reciprocal to the encoding pass described with respect to video encoder 200 (e.g., FIG. 10).

[00226] Entropy decoding unit 301 may retrieve an encoded bitstream. The encoded bitstream may include entropy coded video data (e.g., encoded blocks of video data). Entropy decoding unit 301 may decode the entropy coded video data, and from the entropy decoded video data, motion compensation unit 302 may determine motion information including motion vectors, motion vector precision, reference picture list indexes, and other motion information. Motion compensation unit 302 may, for example, determine such information by performing the AMVP and merge mode.

[00227] Motion compensation unit 302 may produce motion compensated blocks, possibly performing interpolation based on interpolation filters. Identifiers for interpolation filters to be used with sub-pixel precision may be included in the syntax elements.

[00228] Motion compensation unit 302 may use interpolation filters as used by video encoder 200 during encoding of the video block to calculate interpolated values for sub-integer pixels of a reference block. Motion compensation unit 302 may determine the interpolation filters used by video encoder 200 according to received syntax information and use the interpolation filters to produce predictive blocks.

[00229] Motion compensation unit 302 may use some of the syntax information to determine sizes of blocks used to encode frame(s) and/or slice(s) of the encoded video sequence, partition information that describes how each macroblock of a picture of the encoded video sequence is partitioned, modes indicating how each partition is encoded, one or more reference frames (and

reference frame lists) for each inter-encoded block, and other information to decode the encoded video sequence.

**[00230]** Intra prediction unit 303 may use intra prediction modes for example received in the bitstream to form a prediction block from spatially adjacent blocks. Inverse quantization unit 303 inverse quantizes, i.e., de-quantizes, the quantized video block coefficients provided in the bitstream and decoded by entropy decoding unit 301. Inverse transform unit 303 applies an inverse transform.

**[00231]** Reconstruction unit 306 may sum the residual blocks with the corresponding prediction blocks generated by motion compensation unit 202 or intra-prediction unit 303 to form decoded blocks. If desired, a deblocking filter may also be applied to filter the decoded blocks in order to remove blockiness artifacts. The decoded video blocks are then stored in buffer 307, which provides reference blocks for subsequent motion compensation/intra prediction and also produces decoded video for presentation on a display device.

**[00232]** FIG. 12 is a flowchart representation of a method for video processing in accordance with the present technology. The method 1200 includes, at operation 1210, performing a conversion between a picture of a video and a bitstream representation of the video. The picture comprises one or more sub-pictures, and the bitstream representation conforms to a format rule that specifies that a length of a syntax element is equal to  $\text{Ceil}(\text{Log}_2(\text{SS}))$  bits. SS is greater than 0, and the syntax element indicating a horizontal or a vertical position of a top-left corner of a coding tree unit of a sub-picture of the picture.

**[00233]** In some embodiments, the format rule further specifies that a default value of a length of a second syntax element is equal to  $\text{Ceil}(\text{Log}_2(\text{SS})) - P$ , where SS is greater than 0. The second syntax element indicating a default width or a default height of a sub-picture of the picture. In some embodiments, a maximum picture width in luma samples is represented as `pic_width_max_in_luma_samples` and a dimension of a coding tree block is represented as `CtbSizeY`. SS is equal to  $(\text{pic\_width\_max\_in\_luma\_samples} + \text{RR}) / \text{CtbSizeY}$  in case the syntax element specifies the horizontal position of the top-left corner of the coding tree unit or the default width of the sub-picture, RR being a non-zero integer. In some embodiments, a maximum picture height in luma samples is represented as `pic_height_max_in_luma_samples` and a dimension of a coding tree block is represented as `CtbSizeY`. SS is equal to  $(\text{pic\_height\_max\_in\_luma\_samples} + \text{RR}) / \text{CtbSizeY}$  in case the syntax element specifies the vertical position of the top-left corner of

the coding tree unit or the default height of the sub-picture,  $RR$  being a non-zero integer. In some embodiments,  $RR = CtbSizeY - 1$ .

**[00234]** FIG. 13 is a flowchart representation of a method for video processing in accordance with the present technology. The method 1300 includes, at operation 1310, performing a conversion between a picture of a video and a bitstream representation of the video, wherein the picture comprises one or more sub-pictures. The bitstream representation conforms to a format rule that specifies that different sub-pictures have different identifiers.

**[00235]** In some embodiments, the format rule specifies that identifiers of one or more sub-pictures are included at a video unit level, the video unit comprising a sequence parameter set, a picture parameter set, or a picture header. In some embodiments, a first syntax flag at the sequence parameter set indicates whether signaling of the identifiers of the one or more sub-pictures is present at the sequence parameter set level, a second syntax flag at the picture parameter set indicates whether signaling of the identifiers of the one or more sub-pictures is present at the picture parameter set level, and a third syntax flag at the picture header indicates whether signaling of the identifiers of the one or more sub-pictures is present at the picture header level. At least one of the first syntax flag, the second syntax flag, or the third syntax flag is equal to 1. In some embodiments, signaling of the identifiers of the one or more sub-pictures is omitted at the picture header level in case the first syntax flag indicates that signaling of the identifiers of the one or more sub-pictures is present at the sequence parameter set level.

**[00236]** In some embodiments, the format rule specifies that identifiers of one or more sub-pictures are included in a list of sub-picture identifiers. In some embodiments, the identifier of a first sub-picture in the list is denoted as  $SubpicIdList[i]$  and the identifier of a second sub-picture is denoted as  $SubpicIdList[j]$ , where  $j = i - P$ . The format rule specifies that a difference  $D[i]$  between  $SubpicIdList[i]$  and  $SubpicIdList[j]$  is indicated in the bitstream representation. In some embodiments,  $P$  is equal to 1. In some embodiments,  $i > P$ . In some embodiments,  $D[i]$  is greater than 0. In some embodiments,  $D[i] - 1$  is included in the bitstream representation.

**[00237]** In some embodiments, the list of sub-picture identifiers is determined based on an order that the sequence parameter set is considered first, the picture parameter set is considered next, and the picture header is considered last. In some embodiments, the list of sub-picture identifiers is determined based on an order that the picture head is considered first, the picture parameter set is considered next, and the sequence parameter set is considered last.

[00238] In some embodiments, the format rule specifies that, in case identifiers of one or more sub-pictures are omitted at one or more video unit levels, default values are assigned to the identifiers in a list of sub-picture identifiers. The one or more video units comprise at least a sequence parameter set, a picture parameter set, or a picture header. In some embodiments, an identifier of a sub-picture in the list is denoted as  $\text{SubpicIdList}[i]$ , and a default value for  $\text{SubpicIdList}[i]$  is  $i + P$ ,  $P$  being an offset value.

[00239] FIG. 14 is a flowchart representation of a method for video processing in accordance with the present technology. The method 1400 includes, at operation 1410, determining, for a conversion between a block of a first sub-picture of a video and a bitstream representation of the video, whether to apply a deblocking process across an edge between the block and a neighboring block of a second sub-picture based on whether a loop filtering process is allowed across subpicture boundaries. The method 1400 also includes, at operation 1420, performing the conversion based on the determining.

[00240] In some embodiments, a first syntax flag indicates whether the loop filtering process is allowed to access samples across boundaries for the first side of the edge and a second syntax flag indicates whether the loop filtering process is allowed to access samples across boundaries for the second side of the edge. The edge is not filtered in case at least one of the first syntax flag or the second syntax flag indicates that the loop filtering process is disallowed. In some embodiments, the first syntax flag indicates that the loop filtering process is allowed to access samples across boundaries for the first side of the edge, and the edge is not filtered due to the second syntax flag indicating that the loop filtering process is disallowed to access samples across boundaries for the second side of the edge.

[00241] In some embodiments, the first side of the edge is filtered and whether the second side of the edge is filtered are determined separately from each other. In some embodiments, samples on the first side of the edge are filtered due to the first syntax flag being 1. In some embodiments, samples on the second side of the edge are filtered due to the second syntax flag being 1. In some embodiments, the first side of the edge is in the first sub-picture, and the second side of the edge is in the second sub-picture.

[00242] FIG. 15 is a flowchart representation of a method for video processing in accordance with the present technology. The method 1500 includes, at operation 1510, performing a conversion between a video and a bitstream representation of the video. The bitstream representation conforms

to a format rule that specifies a syntax element indicating a maximum block size used for a transform skip coding tool in a picture parameter set is signaled independently from any syntax flags in a sequence parameter set.

[00243] FIG. 16 is a flowchart representation of a method for video processing in accordance with the present technology. The method 1600 includes, at operation 1610, performing a conversion between a video and a bitstream representation of the video. The bitstream representation conforms to a format rule that specifies a syntax element indicating a maximum block size used for a transform skip coding tool is signaled based on a syntax flag in a sequence parameter set.

[00244] In some embodiments, the syntax element is included in a sequence parameter set or a picture header. In some embodiments, the syntax element comprises

`log2_transform_skip_max_size_minus2`. In some embodiments, the syntax flag in the sequence parameter set comprises `sps_transform_skip_enabled_flag`.

[00245] FIG. 17 is a flowchart representation of a method for video processing in accordance with the present technology. The method 1700 includes, at operation 1710, performing a conversion between a current block of a video and a bitstream representation of the video. The method 1700 also includes, at operation 1720, determining, after the conversion, a manner of updating a history-based motion vector prediction table based on whether the current block was coded using a geometry partition mode in which a block is predicted by a weighted sum of at least two predictions. Weights for the weighted sum are generated for at least two partitions. At least some partitions have an angular edge, and wherein the history-based motion vector prediction table includes motion candidates based on previously coded blocks of the video.

[00246] In some embodiments, the history-based motion vector prediction table is not updated in case the current block is coded using the geometry partition mode. In some embodiments, the history-based motion vector prediction table is updated in case the current block is coded using the geometry partition mode. In some embodiments, the history-based motion vector prediction table is updated using motion information of one prediction of the current block. In some embodiments, the history-based motion vector prediction table is updated using motion information of multiple predictions of the current block.

[00247] FIG. 18 is a flowchart representation of a method for video processing in accordance with the present technology. The method 1800 includes, at operation 1810, performing a conversion between a current processing unit of a video and a bitstream representation of the video using a

cross-component adaptive loop filtering process in which chroma samples are filtered based on corresponding luma samples. The current processing unit comprises an adaptive loop filtering processing unit determined by two virtual boundaries. Luma samples that are located outside of the current processing unit are excluded for filtering chroma samples of the current processing unit.

[00248] In some embodiments, the luma samples that are located outside of the current processing unit are padded, and the padded luma samples are used for filtering the chroma samples of the current processing unit.

[00249] FIG. 19 is a flowchart representation of a method for video processing in accordance with the present technology. The method 1900 includes, at operation 1910, performing a conversion between a current processing unit of a video and a bitstream representation of the video using a cross-component adaptive loop filtering process in which chroma samples are filtered based on corresponding luma samples. The current processing unit comprises an adaptive loop filtering process unit bounded by two virtual boundaries. Luma samples that are located outside of the current processing unit are used for filtering chroma samples of the current processing unit.

[00250] In some embodiments, the conversion generates the video from the bitstream representation. In some embodiments, the conversion generates the bitstream representation from the video.

[00251] Some embodiments of the disclosed technology include making a decision or determination to enable a video processing tool or mode. In an example, when the video processing tool or mode is enabled, the encoder will use or implement the tool or mode in the processing of a block of video, but may not necessarily modify the resulting bitstream based on the usage of the tool or mode. That is, a conversion from the block of video to the bitstream representation of the video will use the video processing tool or mode when it is enabled based on the decision or determination. In another example, when the video processing tool or mode is enabled, the decoder will process the bitstream with the knowledge that the bitstream has been modified based on the video processing tool or mode. That is, a conversion from the bitstream representation of the video to the block of video will be performed using the video processing tool or mode that was enabled based on the decision or determination.

[00252] Some embodiments of the disclosed technology include making a decision or determination to disable a video processing tool or mode. In an example, when the video processing tool or mode is disabled, the encoder will not use the tool or mode in the conversion of

the block of video to the bitstream representation of the video. In another example, when the video processing tool or mode is disabled, the decoder will process the bitstream with the knowledge that the bitstream has not been modified using the video processing tool or mode that was enabled based on the decision or determination.

**[00253]** The disclosed and other solutions, examples, embodiments, modules and the functional operations described in this document can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this document and their structural equivalents, or in combinations of one or more of them. The disclosed and other embodiments can be implemented as one or more computer program products, e.g., one or more modules of computer program instructions encoded on a computer readable medium for execution by, or to control the operation of, data processing apparatus. The computer readable medium can be a machine-readable storage device, a machine-readable storage substrate, a memory device, a composition of matter effecting a machine-readable propagated signal, or a combination of one or more them. The term “data processing apparatus” encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them. A propagated signal is an artificially generated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal, that is generated to encode information for transmission to suitable receiver apparatus.

**[00254]** A computer program (also known as a program, software, software application, script, or code) can be written in any form of programming language, including compiled or interpreted languages, and it can be deployed in any form, including as a stand-alone program or as a module, component, subroutine, or other unit suitable for use in a computing environment. A computer program does not necessarily correspond to a file in a file system. A program can be stored in a portion of a file that holds other programs or data (e.g., one or more scripts stored in a markup language document), in a single file dedicated to the program in question, or in multiple coordinated files (e.g., files that store one or more modules, sub programs, or portions of code). A computer program can be deployed to be executed on one computer or on multiple computers

that are located at one site or distributed across multiple sites and interconnected by a communication network.

[00255] The processes and logic flows described in this document can be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

[00256] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read only memory or a random-access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[00257] While this patent document contains many specifics, these should not be construed as limitations on the scope of any subject matter or of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular techniques. Certain features that are described in this patent document in the context of separate embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or

variation of a subcombination.

**[00258]** Similarly, while operations are depicted in the drawings in a particular order, this should not be understood as requiring that such operations be performed in the particular order shown or in sequential order, or that all illustrated operations be performed, to achieve desirable results. Moreover, the separation of various system components in the embodiments described in this patent document should not be understood as requiring such separation in all embodiments.

**[00259]** Only a few implementations and examples are described and other implementations, enhancements and variations can be made based on what is described and illustrated in this patent document.