

CROSS COMPONENT ADAPTIVE LOOP FILTERING FOR VIDEO CODING

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] 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/CN2020/070001, filed on January 1, 2020. 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

[0001] This patent document relates to video coding techniques, devices and systems.

BACKGROUND

[0002] Currently, efforts are underway to improve the performance of current video codec technologies to provide better compression ratios or provide video coding and decoding schemes that allow for lower complexity or parallelized implementations. Industry experts have recently proposed several new video coding tools and tests are currently underway for determining their effectivity.

SUMMARY

[0002] Devices, systems and methods related to digital video coding, and specifically, to management of motion vectors are described. The described methods may be applied to existing video coding standards (e.g., High Efficiency Video Coding (HEVC) or Versatile Video Coding) and future video coding standards or video codecs.

[0003] In one representative aspect, the disclosed technology may be used to provide a method for video processing. This method includes performing a conversion between a video unit of a video and a bitstream of the video according to a rule, where the rule specifies that whether a cross-component adaptive loop filter (CC-ALF) mode and an adaptive loop filter (ALF) mode are enabled for coding the video unit are indicated in the bitstream in a mutually independent manner.

[0004] In another representative aspect, the disclosed technology may be used to provide a method for video processing. This method includes performing a conversion between a video unit of a chroma component a video and a bitstream of the video, wherein the bitstream conforms to a format rule, where the format rule specifies that the bitstream includes a syntax element indicating whether a cross-component filter for the chroma component is enabled for all slices associated with a picture header only when: a value for a chroma array type is not equal to zero, or a color format of the video is not 4:0:0.

[0005] In another representative aspect, the disclosed technology may be used to provide a method for video processing. This method includes performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that when a chroma quantization parameter (QP) table is used to derive parameters of the deblocking filter, processing by the chroma QP table is performed on individual chroma QP values.

[0006] In another representative aspect, the disclosed technology may be used to provide another method for video processing. This method includes performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein the chroma QP offsets are at picture/slice/tile/brick/subpicture level.

[0007] In another representative aspect, the disclosed technology may be used to provide another method for video processing. This method includes performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein information pertaining to a same luma coding unit is used in the deblocking filter and for deriving a chroma QP offset.

[0008] In another representative aspect, the disclosed technology may be used to provide another method for video processing. This method includes performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter,

wherein an indication of enabling usage of the chroma QP offsets is signaled in the coded representation.

[0009] In another representative aspect, the disclosed technology may be used to provide another method for video processing. This method includes performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein the chroma QP offsets used in the deblocking filter are identical of whether JCCR coding method is applied on a boundary of the video unit or a method different from the JCCR coding method is applied on the boundary of the video unit.

[0010] In another representative aspect, the disclosed technology may be used to provide another method for video processing. This method includes performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein a boundary strength (BS) of the deblocking filter is calculated without comparing reference pictures and/or a number of motion vectors (MVs) associated with the video unit at a P side boundary with reference pictures of the video unit at a Q side boundary.

[0011] In another example aspect, another method of video processing is disclosed. The method includes determining, for a conversion between a video unit of a component of a video and a coded representation of the video, a size of a quantization group for the video unit, based on a constraint rule that specifies that the size must be larger than K, where K is a positive number and performing the conversion based on the determining.

[0012] In another example aspect, another method of video processing is disclosed. The method includes performing a conversion between a video unit of a video and a bitstream of the video according to a rule, wherein the rule specifies whether the bitstream includes at least one of a control flag of a chroma block-based delta pulse code modulation (BDPCM) mode, a palette mode, or an adaptive color transform (ACT) mode is based on a value of a chroma array type of the video.

[0013] Further, in a representative aspect, an apparatus in a video system comprising a processor and a non-transitory memory with instructions thereon is disclosed. The instructions upon

execution by the processor, cause the processor to implement any one or more of the disclosed methods.

[0014] Additionally, in a representative aspect, a video decoding apparatus comprising a processor configured to implement any one or more of the disclosed methods.

[0015] In another representative aspect, a video encoding apparatus comprising a processor configured to implement any one or more of the disclosed methods.

[0016] Also, a computer program product stored on a non-transitory computer readable media, the computer program product including program code for carrying out any one or more of the disclosed methods is disclosed.

[0017] The above and other aspects and features of the disclosed technology are described in greater detail in the drawings, the description and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0018] FIG. 1 shows an example of an overall processing flow of a blocking deblocking filter process.

[0019] FIG. 2 shows an example of a flow diagram of a Bs calculation.

[0020] FIG. 3 shows an example of a referred information for Bs calculation at CTU boundary.

[0021] FIG. 4 shows an example of pixels involved in filter on/off decision and strong/weak filter selection.

[0022] FIG. 5 shows an example of an overall processing flow of deblocking filter process in VVC.

[0023] FIG. 6 shows an example of a luma deblocking filter process in VVC.

[0024] FIG. 7 shows an example of a chroma deblocking filter process in VVC

[0025] FIG. 8 shows an example of a filter length determination for sub PU boundaries.

[0026] FIGs. 9A and 9B show examples of center positions of a chroma block.

[0027] FIG. 10 shows examples of blocks at P side and Q side.

- [0028] FIG. 11 shows examples of usage of a luma block's decoded information.
- [0029] FIG. 12 is a block diagram of an example of a hardware platform for implementing a visual media decoding or a visual media encoding technique described in the present document.
- [0030] FIG. 13 shows a flowchart of an example method for video coding.
- [0031] FIG. 14A shows an example of Placement of CC-ALF with respect to other loop filters.
- [0032] FIG. 14B shows an example of Placement of CC-ALF with respect to a diamond shaped filter.
- [0033] FIG. 15 shows an example flowchart for an adaptive color transform (ACT) process.
- [0034] FIG. 16 shows an example flowchart for high-level deblocking control mechanism.
- [0035] FIG. 17 shows an example flowchart for high-level deblocking control mechanism.
- [0036] FIG. 18 is a block diagram that illustrates a video coding system in accordance with some embodiments of the present disclosure.
- [0037] FIG. 19 is a block diagram that illustrates an encoder in accordance with some embodiments of the present disclosure.
- [0038] FIG. 20 is a block diagram that illustrates a decoder in accordance with some embodiments of the present disclosure.
- [0039] FIG. 21 is a block diagram of an example video processing system in which disclosed techniques may be implemented.
- [0040] FIGS. 22-24 are flowcharts for example methods of video processing.

DETAILED DESCRIPTION

1. Video coding in HEVC/H.265

[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. 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.

2.1. Deblocking scheme in HEVC

[0042] A deblocking filter process is performed for each CU in the same order as the decoding process. First, vertical edges are filtered (horizontal filtering), then horizontal edges are filtered (vertical filtering). Filtering is applied to 8x8 block boundaries which are determined to be filtered, for both luma and chroma components. 4x4 block boundaries are not processed in order to reduce the complexity.

[0043] FIG. 1 illustrates the overall processing flow of deblocking filter process. A boundary can have three filtering status: no filtering, weak filtering and strong filtering. Each filtering decision is based on boundary strength, B_s , and threshold values, β and t_c .

[0044] Three kinds of boundaries may be involved in the filtering process: CU boundary, TU boundary and PU boundary. CU boundaries, which are outer edges of CU, are always involved in the filtering since CU boundaries are always also TU boundary or PU boundary. When PU shape is $2N \times N$ ($N > 4$) and RQT depth is equal to 1, TU boundary at 8x8 block grid and PU boundary between each PU inside CU are involved in the filtering. One exception is that when the PU boundary is inside the TU, the boundary is not filtered.

2.1.1. Boundary strength calculation

[0045] Generally speaking, boundary strength (B_s) reflects how strong filtering is needed for the boundary. If B_s is large, strong filtering should be considered.

[0046] Let P and Q be defined as blocks which are involved in the filtering, where P represents the block located in left (vertical edge case) or above (horizontal edge case) side of the boundary and

Q represents the block located in right (vertical edge case) or above (horizontal edge case) side of the boundary. FIG. 2 illustrates how the Bs value is calculated based on the intra coding mode, existence of non-zero transform coefficients and motion information, reference picture, number of motion vectors and motion vector difference.

[0047] Bs is calculated on a 4x4 block basis, but it is re-mapped to an 8x8 grid. The maximum of the two values of Bs which correspond to 8 pixels consisting of a line in the 4x4 grid is selected as the Bs for boundaries in the 8x8 grid.

[0048] In order to reduce line buffer memory requirement, only for CTU boundary, information in every second block (4x4 grid) in left or above side is re-used as depicted in FIG. 3.

2.1.2. β and t_C decision

[0049] Threshold values β and t_C which involving in filter on/off decision, strong and weak filter selection and weak filtering process are derived based on luma quantization parameter of P and Q blocks, QP_P and QP_Q , respectively. Q used to derive β and t_C is calculated as follows.

$$Q = ((QP_P + QP_Q + 1) \gg 1).$$

[0050] A variable β is derived as shown in Table 1, based on Q. If Bs is greater than 1, the variable t_C is specified as Table 1 with $\text{Clip3}(0, 55, Q + 2)$ as input. Otherwise (BS is equal or less than 1), the variable t_C is specified as Table 1 with Q as input.

Table 1 Derivation of threshold variables β and t_C from input Q

Q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
β	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	7	8
tc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Q	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37
β	9	10	11	12	13	14	15	16	17	18	20	22	24	26	28	30	32	34	36
tc	1	1	1	1	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4
Q	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	
β	38	40	42	44	46	48	50	52	54	56	58	60	62	64	64	64	64	64	
tc	5	5	6	6	7	8	9	9	10	10	11	11	12	12	13	13	14	14	

2.1.3. Filter on/off decision for 4 lines

[0051] Filter on/off decision is done for four lines as a unit. FIG. 4 illustrates the pixels involving in filter on/off decision. The 6 pixels in the two red boxes for the first four lines are used to determine filter on/off for 4 lines. The 6 pixels in two red boxes for the second 4 lines are used to determine filter on/off for the second four lines.

[0052] If $dp0 + dq0 + dp3 + dq3 < \beta$, filtering for the first four lines is turned on and strong/weak filter selection process is applied. Each variable is derived as follows.

$$dp0 = |p_{2,0} - 2*p_{1,0} + p_{0,0}|, dp3 = |p_{2,3} - 2*p_{1,3} + p_{0,3}|, dp4 = |p_{2,4} - 2*p_{1,4} + p_{0,4}|, dp7 = |p_{2,7} - 2*p_{1,7} + p_{0,7}|$$

$$dq0 = |q_{2,0} - 2*q_{1,0} + q_{0,0}|, dq3 = |q_{2,3} - 2*q_{1,3} + q_{0,3}|, dq4 = |q_{2,4} - 2*q_{1,4} + q_{0,4}|, dq7 = |q_{2,7} - 2*q_{1,7} + q_{0,7}|$$

[0053] If the condition is not met, no filtering is done for the first 4 lines. Additionally, if the condition is met, dE, dEp1 and dEp2 are derived for weak filtering process. The variable dE is set equal to 1. If $dp0 + dp3 < (\beta + (\beta \gg 1)) \gg 3$, the variable dEp1 is set equal to 1. If $dq0 + dq3 < (\beta + (\beta \gg 1)) \gg 3$, the variable dEq1 is set equal to 1.

[0054] For the second four lines, decision is made in a same fashion with above.

2.1.4. Strong/weak filter selection for 4 lines

[0055] After the first four lines are determined to filtering on in filter on/off decision, if following two conditions are met, strong filter is used for filtering of the first four lines. Otherwise, weak filter is used for filtering. Involving pixels are same with those used for filter on/off decision as depicted in Figure 4.

$$1) 2*(dp0+dq0) < (\beta \gg 2), |p3_0 - p0_0| + |q0_0 - q3_0| < (\beta \gg 3) \text{ and } |p0_0 - q0_0| < (5 * t_c + 1) \gg 1$$

$$2) 2*(dp3+dq3) < (\beta \gg 2), |p3_3 - p0_3| + |q0_3 - q3_3| < (\beta \gg 3) \text{ and } |p0_3 - q0_3| < (5 * t_c + 1) \gg 1$$

[0056] As a same fashion, if following two conditions are met, strong filter is used for filtering of the second 4 lines. Otherwise, weak filter is used for filtering.

$$1) 2*(dp4+dq4) < (\beta \gg 2), |p3_4 - p0_4| + |q0_4 - q3_4| < (\beta \gg 3) \text{ and } |p0_4 - q0_4| < (5 * t_c + 1) \gg 1$$

$$2) 2*(dp7+dq7) < (\beta \gg 2), |p3_7 - p0_7| + |q0_7 - q3_7| < (\beta \gg 3) \text{ and } |p0_7 - q0_7| < (5 * t_c + 1) \gg 1$$

2.1.4.1. Strong filtering

[0057] For strong filtering, filtered pixel values are obtained by following equations. It is worth to note that three pixels are modified using four pixels as an input for each P and Q block, respectively.

$$p0' = (p_2 + 2*p_1 + 2*p_0 + 2*q_0 + q_1 + 4) \gg 3$$

$$q0' = (p_1 + 2*p_0 + 2*q_0 + 2*q_1 + q_2 + 4) \gg 3$$

$$p1' = (p_2 + p_1 + p_0 + q_0 + 2) \gg 2$$

$$q1' = (p_0 + q_0 + q_1 + q_2 + 2) \gg 2$$

$$p2' = (2*p_3 + 3*p_2 + p_1 + p_0 + q_0 + 4) \gg 3$$

$$q2' = (p_0 + q_0 + q_1 + 3*q_2 + 2*q_3 + 4) \gg 3$$

2.1.4.2. Weak filtering

[0058] Let's define Δ as follows.

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

When $\text{abs}(\Delta)$ is less than $t_c * 10$,

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

$$p_0' = \text{Clip1}_Y(p_0 + \Delta)$$

$$q_0' = \text{Clip1}_Y(q_0 - \Delta)$$

If $dEp1$ is equal to 1,

$$\Delta p = \text{Clip3}(-(t_c >> 1), t_c >> 1, (((p_2 + p_0 + 1) >> 1) - p_1 + \Delta) >> 1)$$

$$p_1' = \text{Clip1}_Y(p_1 + \Delta p)$$

If $dEq1$ is equal to 1,

$$\Delta q = \text{Clip3}(-(t_c >> 1), t_c >> 1, (((q_2 + q_0 + 1) >> 1) - q_1 - \Delta) >> 1)$$

$$q_1' = \text{Clip1}_Y(q_1 + \Delta q)$$

[0059] It is worth to note that maximum two pixels are modified using three pixels as an input for each P and Q block, respectively.

2.1.4.3. Chroma filtering

[0060] B_s of chroma filtering is inherited from luma. If $B_s > 1$ or if coded chroma coefficient existing case, chroma filtering is performed. No other filtering decision is there. And only one filter is applied for chroma. No filter selection process for chroma is used. The filtered sample values p_0' and q_0' are derived as follows.

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

$$p_0' = \text{Clip1}_C(p_0 + \Delta)$$

$$q_0' = \text{Clip1}_C(q_0 - \Delta)$$

2.2 Deblocking scheme in VVC

[0061] In the VTM6, deblocking filtering process is mostly the same to those in HEVC. However, the following modifications are added.

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

[0062] FIG. 5 depicts a flowchart of deblocking filters process in VVC for a coding unit.

2.2.1. Filter strength dependent on reconstructed average luma

[0063] In HEVC, the filter strength of the deblocking filter is controlled by the variables β and t_C which are derived from the averaged quantization parameters qP_L . In the VTM6, deblocking filter controls the strength of the deblocking filter by adding offset to qP_L according to the luma level of the reconstructed samples if the SPS flag of this method is true. The reconstructed luma level LL is derived as follow:

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

where, the sample values $p_{i,k}$ and $q_{i,k}$ with $i = 0..3$ and $k = 0$ and 3 are derived as shown in _____. Then LL is used to decide the offset $qpOffset$ based on the threshold signaled in SPS. After that, the qP_L , which is derived as follows, is employed to derive the β and t_C .

$$qP_L = ((Qp_Q + Qp_P + 1) \gg 1) + qpOffset \quad (3-2)$$

where Qp_Q and Qp_P denote the quantization parameters of the coding units containing the sample $q_{0,0}$ and $p_{0,0}$, respectively. In the current VVC, this method is only applied on the luma deblocking process.

2.2.2. 4x4 deblocking grid for luma

[0064] HEVC uses an 8x8 deblocking grid for both luma and chroma. In VTM6, deblocking on a 4x4 grid for luma boundaries was introduced to handle blocking artifacts from rectangular transform shapes. Parallel friendly luma deblocking on a 4x4 grid is achieved by restricting the number of samples to be deblocked to 1 sample on each side of a vertical luma boundary where one side has a width of 4 or less or to 1 sample on each side of a horizontal luma boundary where one side has a height of 4 or less.

2.2.3. Boundary Strength derivation for luma

[0065] The detailed boundary strength derivation could be found in Table 2. The conditions in Table 2 are checked sequentially.

Table 2 Boundary strength derivation

Conditions	Y	U	V
P and Q are BDPCM	0	N/A	N/A
P or Q is intra	2	2	2
It is a transform block edge, and P or Q is CIIP	2	2	2
It is a transform block edge, and P or Q has non-zero transform coefficients	1	1	1
It is a transform block edge, and P or Q is JCCR	N/A	1	1
P and Q are in different coding modes	1	1	1

One or more of the following conditions are true: 1. P and Q are both IBC, and the BV distance $\geq$ half-pel in x- or y-dir 2. P and Q have different ref pictures*, or have different number of MVs 3. Both P and Q have only one mv, and the MV distance $\geq$ half-pel in x- or y-dir 4. P has two MVs pointing to two different ref pictures, and P and Q have same ref pictures in the list 0, the MV pair in the list 0 or list 1 has a distance $\geq$ half-pel in x- or y-dir 5. P has two MVs pointing to two different ref pictures, and P and Q have different ref pictures in the list 0, the MV of P in the list 0 and the MV of Q in the list 1 have the distance $\geq$ half-pel in x- or y-dir, or the MV of P in the list 1 and the MV of Q in the list 0 have the distance $\geq$ half-pel in x- or y-dir 6. Both P and Q have two MVs pointing to the same ref pictures, and both of the following two conditions are satisfied: ○ The MV of P in the list 0 and the MV of Q in the list 0 has a distance $\geq$ half-pel in x- or y-dir or the MV of P in the list 1 and the MV of Q in the list 1 has a distance $\geq$ half-pel in x- or y-dir ○ The MV of P in the list 0 and the MV of Q in the list 1 has a distance $\geq$ half-pel in x- or y-dir or the MV of P in the list 1 and the MV of Q in the list 0 has a distance $\geq$ half-pel in x- or y-dir *Note: The determination of whether the reference pictures used for the two coding subblocks are the same or different is based	1	N/A	N/A
--	---	-----	-----

only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether the index position within a reference picture list is different.			
Otherwise	0	0	0

2.2.4. Stronger deblocking filter for luma

[0066] The proposal uses a bilinear filter when samples at either one side of a boundary belong to a large block. A sample belonging to a large block is defined as when the width ≥ 32 for a vertical edge, and when height ≥ 32 for a horizontal edge.

[0067] The bilinear filter is listed below.

[0068] Block boundary samples p_i for $i=0$ to S_p-1 and q_j for $j=0$ to S_q-1 (p_i and q_j follow the definitions in HEVC deblocking described above) are then replaced by linear interpolation as follows:

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

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

where $tcPD_i$ and $tcPD_j$ term is a position dependent clipping described in Section 2.2.5 and $g_j, f_i, Middle_{s,t}, P_s$ and Q_s are given below:

Sp, Sq 7, 7 (p side: 7, q side: 7)	$f_i = 59 - i * 9$, can also be described as $\mathbf{f} = \{59, 50, 41, 32, 23, 14, 5\}$ $g_j = 59 - j * 9$, can also be described as $\mathbf{g} = \{59, 50, 41, 32, 23, 14, 5\}$ $Middle_{7,7} = (2 * (p_o + q_o) + p_1 + q_1 + p_2 + q_2 + p_3 + q_3 + p_4 + q_4 + p_5 + q_5 + p_6 + q_6 + 8) \gg 4$ $P_7 = (p_6 + p_7 + 1) \gg 1, \quad Q_7 = (q_6 + q_7 + 1) \gg 1$
--	--

7, 3 (p side: 7 q side: 3)	$f_i = 59 - i * 9$, can also be described as $\mathbf{f} = \{59, 50, 41, 32, 23, 14, 5\}$ $g_j = 53 - j * 21$, can also be described as $\mathbf{g} = \{53, 32, 11\}$ $Middle_{7,3} = (2 * (p_o + q_o) + q_0 + 2 * (q_1 + q_2) + p_1 + q_1 + p_2 + p_3 + p_4 + p_5 + p_6 + 8) \gg 4$ $P_7 = (p_6 + p_7 + 1) \gg 1$, $Q_3 = (q_2 + q_3 + 1) \gg 1$
3, 7 (p side: 3 q side: 7)	$g_j = 59 - j * 9$, can also be described as $\mathbf{g} = \{59, 50, 41, 32, 23, 14, 5\}$ $f_i = 53 - i * 21$, can also be described as $\mathbf{f} = \{53, 32, 11\}$ $Middle_{3,7} = (2 * (q_o + p_o) + p_0 + 2 * (p_1 + p_2) + q_1 + p_1 + q_2 + q_3 + q_4 + q_5 + q_6 + 8) \gg 4$ $Q_7 = (q_6 + q_7 + 1) \gg 1$, $P_3 = (p_2 + p_3 + 1) \gg 1$
7, 5 (p side: 7 q side: 5)	$g_j = 58 - j * 13$, can also be described as $\mathbf{g} = \{58, 45, 32, 19, 6\}$ $f_i = 59 - i * 9$, can also be described as $\mathbf{f} = \{59, 50, 41, 32, 23, 14, 5\}$ $Middle_{7,5} = (2 * (p_o + q_o + p_1 + q_1) + q_2 + p_2 + q_3 + p_3 + q_4 + p_4 + q_5 + p_5 + 8) \gg 4$ $Q_5 = (q_4 + q_5 + 1) \gg 1$, $P_7 = (p_6 + p_7 + 1) \gg 1$
5, 7 (p side: 5 q side: 7)	$g_j = 59 - j * 9$, can also be described as $\mathbf{g} = \{59, 50, 41, 32, 23, 14, 5\}$ $f_i = 58 - i * 13$, can also be described as $\mathbf{f} = \{58, 45, 32, 19, 6\}$ $Middle_{5,7} = (2 * (q_o + p_o + p_1 + q_1) + q_2 + p_2 + q_3 + p_3 + q_4 + p_4 + q_5 + p_5 + 8) \gg 4$ $Q_7 = (q_6 + q_7 + 1) \gg 1$, $P_5 = (p_4 + p_5 + 1) \gg 1$
5, 5 (p side: 5 q side: 5)	$g_j = 58 - j * 13$, can also be described as $\mathbf{g} = \{58, 45, 32, 19, 6\}$ $f_i = 58 - i * 13$, can also be described as $\mathbf{f} = \{58, 45, 32, 19, 6\}$ $Middle_{5,5} = (2 * (q_o + p_o + p_1 + q_1 + q_2 + p_2) + q_3 + p_3 + q_4 + p_4 + 8) \gg 4$ $Q_5 = (q_4 + q_5 + 1) \gg 1$, $P_5 = (p_4 + p_5 + 1) \gg 1$
5, 3 (p side: 5 q side: 3)	$g_j = 53 - j * 21$, can also be described as $\mathbf{g} = \{53, 32, 11\}$ $f_i = 58 - i * 13$, can also be described as $\mathbf{f} = \{58, 45, 32, 19, 6\}$ $Middle_{5,3} = (q_o + p_o + p_1 + q_1 + q_2 + p_2 + q_3 + p_3 + 4) \gg 3$

	$Q_3 = (q_2 + q_3 + 1) \gg 1, \quad P_5 = (p_4 + p_5 + 1) \gg 1$
3, 5 (p side: 3 q side: 5)	$g_j = 58 - j * 13, \text{ can also be described as } \mathbf{g} = \{58, 45, 32, 19, 6\}$ $f_i = 53 - i * 21, \text{ can also be described as } \mathbf{f} = \{53, 32, 11\}$ $Middle_{3,5} = (q_o + p_o + p_1 + q_1 + q_2 + p_2 + q_3 + p_3 + 4) \gg 3$ $Q_5 = (q_4 + q_5 + 1) \gg 1, \quad P_3 = (p_2 + p_3 + 1) \gg 1$

2.2.5. Deblocking control for luma

[0069] The deblocking decision process is described in this sub-section.

[0070] Wider-stronger luma filter is filters are used only if all of the **Condition1**, **Condition2** and **Condition 3** are TRUE.

[0071] The condition 1 is the “large block condition”. This condition detects whether the samples at P-side and Q-side belong to large blocks, which are represented by the variable bSidePisLargeBlk and bSideQisLargeBlk respectively. The bSidePisLargeBlk and bSideQisLargeBlk are defined as follows.

bSidePisLargeBlk = ((edge type is vertical and p₀ belongs to CU with width >= 32) || (edge type is horizontal and p₀ belongs to CU with height >= 32)) ? TRUE: FALSE

bSideQisLargeBlk = ((edge type is vertical and q₀ belongs to CU with width >= 32) || (edge type is horizontal and q₀ belongs to CU with height >= 32)) ? TRUE: FALSE

[0072] Based on bSidePisLargeBlk and bSideQisLargeBlk, the condition 1 is defined as follows.

Condition1 = (bSidePisLargeBlk || bSideQisLargeBlk) ? TRUE: FALSE

[0073] Next, if **Condition 1** is true, the condition 2 will be further checked. First, the following variables are derived:

dp0, dp3, dq0, dq3 are first derived as in HEVC

if (p side is greater than or equal to 32)

dp0 = (dp0 + Abs(p_{5,0} - 2 * p_{4,0} + p_{3,0}) + 1) >> 1

dp3 = (dp3 + Abs(p_{5,3} - 2 * p_{4,3} + p_{3,3}) + 1) >> 1

if (q side is greater than or equal to 32)

$$dq0 = (dq0 + \text{Abs}(q_{5,0} - 2 * q_{4,0} + q_{3,0}) + 1) \gg 1$$

$$dq3 = (dq3 + \text{Abs}(q_{5,3} - 2 * q_{4,3} + q_{3,3}) + 1) \gg 1$$
 dpq0, dpq3, dp, dq, d are then derived as in HEVC.

[0074] Then the condition 2 is defined as follows.

Condition2 = $(d < \beta)$? TRUE: FALSE

Where $d = dp0 + dq0 + dp3 + dq3$, as shown in section 2.1.4.

[0075] If Condition1 and Condition2 are valid it is checked if any of the blocks uses sub-blocks:

If(bSidePisLargeBlk)

If(mode block P == SUBBLOCKMODE)

Sp = 5

else

Sp = 7

else

Sp = 3

If(bSideQisLargeBlk)

If(mode block Q == SUBBLOCKMODE)

Sq = 5

else

Sq = 7

else

Sq = 3

[0076] Finally, if both the Condition 1 and Condition 2 are valid, the proposed deblocking method will check the condition 3 (the large block Strong filter condition), which is defined as follows.

In the Condition3 StrongFilterCondition, the following variables are derived:

dpq is derived as in HEVC.

sp3 = $\text{Abs}(p3 - p0)$, derived as in HEVC

if (p side is greater than or equal to 32)

if(Sp==5)

```

    sp3 = ( sp3 + Abs( p5 - p3 ) + 1 ) >> 1
else
    sp3 = ( sp3 + Abs( p7 - p3 ) + 1 ) >> 1
sq3 = Abs( q0 - q3 ), derived as in HEVC
if (q side is greater than or equal to 32)
    If(Sq==5)
        sq3 = ( sq3 + Abs( q5 - q3 ) + 1 ) >> 1
    else
        sq3 = ( sq3 + Abs( q7 - q3 ) + 1 ) >> 1

```

[0077] As in HEVC derive, StrongFilterCondition = (dpq is less than ($\beta >> 2$), $sp3 + sq3$ is less than ($3 * \beta >> 5$), and $Abs(p0 - q0)$ is less than ($5 * tC + 1$) >> 1) ? TRUE : FALSE

[0078] FIG. 6 depicts the flowchart of luma deblocking filter process.

2.2.6. Strong deblocking filter for chroma

[0079] The following strong deblocking filter for chroma is defined:

$$\begin{aligned}
 p_2' &= (3 * p_3 + 2 * p_2 + p_1 + p_0 + q_0 + 4) >> 3 \\
 p_1' &= (2 * p_3 + p_2 + 2 * p_1 + p_0 + q_0 + q_1 + 4) >> 3 \\
 p_0' &= (p_3 + p_2 + p_1 + 2 * p_0 + q_0 + q_1 + q_2 + 4) >> 3
 \end{aligned}$$

[0080] The proposed chroma filter performs deblocking on a 4x4 chroma sample grid.

2.2.7. Deblocking control for chroma

[0081] The above chroma filter performs deblocking on a 8x8 chroma sample grid. The chroma strong filters are used on both sides of the block boundary. Here, the chroma filter is selected when both sides of the chroma edge are greater than or equal to 8 (in unit of chroma sample), and the following decision with three conditions are satisfied. The first one is for decision of boundary strength as well as large block. The second and third one are basically the same as for HEVC luma decision, which are on/off decision and strong filter decision, respectively.

[0082] FIG. 7 depicts the flowchart of chroma deblocking filter process.

2.2.8. Position dependent clipping

[0083] The proposal also introduces a position dependent clipping $tcPD$ which is applied to the output samples of the luma filtering process involving strong and long filters that are modifying 7, 5 and 3 samples at the boundary. Assuming quantization error distribution, it is proposed to increase clipping value for samples which are expected to have higher quantization noise, thus expected to have higher deviation of the reconstructed sample value from the true sample value.

[0084] For each P or Q boundary filtered with proposed asymmetrical filter, depending on the result of decision making process described in Section 2.2, position dependent threshold table is selected from $Tc7$ and $Tc3$ tables that are provided to decoder as a side information:

$$Tc7 = \{ 6, 5, 4, 3, 2, 1, 1 \};$$

$$Tc3 = \{ 6, 4, 2 \};$$

$$tcPD = (SP == 3) ? Tc3 : Tc7;$$

$$tcQD = (SQ == 3) ? Tc3 : Tc7;$$

[0085] For the P or Q boundaries being filtered with a short symmetrical filter, position dependent threshold of lower magnitude is applied:

$$Tc3 = \{ 3, 2, 1 \};$$

[0086] Following defining the threshold, filtered p'_i and q'_i sample values are clipped according to tcP and tcQ clipping values:

$$p''_i = clip3(p'_i + tcP_i, p'_i - tcP_i, p'_i);$$

$$q''_j = clip3(q'_j + tcQ_j, q'_j - tcQ_j, q'_j);$$

where p'_i and q'_i are filtered sample values, p''_i and q''_j are output sample value after the clipping and tcP_i tcP_i are clipping thresholds that are derived from the VVC tc parameter and $tcPD$ and $tcQD$. Term $clip3$ is a clipping function as it is specified in VVC.

2.2.9. Sub-block deblocking adjustment

[0087] To enable parallel friendly deblocking using both long filters and sub-block deblocking the long filters is restricted to modify at most 5 samples on a side that uses sub-block deblocking (AFFINE or ATMVP) as shown in the luma control for long filters. Additionally, the sub-block

deblocking is adjusted such that that sub-block boundaries on an 8x8 grid that are close to a CU or an implicit TU boundary is restricted to modify at most two samples on each side.

[0088] Following applies to sub-block boundaries that not are aligned with the CU boundary.

```

If(mode block Q == SUBBLOCKMODE && edge!=0){
  if (!(implicitTU && (edge == (64 / 4))))
    if (edge == 2 || edge == (orthogonalLength - 2) || edge == (56 / 4) || edge == (72 / 4))
      Sp = Sq = 2;
    else
      Sp = Sq = 3;
  else
    Sp = Sq = bSideQisLargeBlk? 5:3
}

```

[0089] Where edge equal to 0 corresponds to CU boundary, edge equal to 2 or equal to orthogonalLength-2 corresponds to sub-block boundary 8 samples from a CU boundary etc. Where implicit TU is true if implicit split of TU is used. FIG. 8 show the flowcharts of determination process for TU boundaries and sub-PU boundaries.

[0090] Filtering of horizontal boundary is limiting Sp= 3 for luma, Sp=1 and Sq=1 for chroma, when the horizontal boundary is aligned with the CTU boundary.

2.2.10. Deblocking decision adapted to smaller difference in motion

[0091] HEVC enables deblocking of a prediction unit boundary when the difference in at least one motion vector component between blocks on respective side of the boundary is equal to or larger than a threshold of 1 sample. In VTM6, a threshold of a half luma sample is introduced to also enable removal of blocking artifacts originating from boundaries between inter prediction units that have a small difference in motion vectors.

2.3. Combined inter and intra prediction (CIIP)

[0092] In VTM6, when a CU is coded in merge mode, if the CU contains at least 64 luma samples (that is, CU width times CU height is equal to or larger than 64), and if both CU width and

CU height are less than 128 luma samples, an additional flag is signalled to indicate if the combined inter/intra prediction (CIIP) mode is applied to the current CU. As its name indicates, the CIIP prediction combines an inter prediction signal with an intra prediction signal. The inter prediction signal in the CIIP mode P_{inter} is derived using the same inter prediction process applied to regular merge mode; and the intra prediction signal P_{intra} is derived following the regular intra prediction process with the planar mode. Then, the intra and inter prediction signals are combined using weighted averaging, where the weight value is calculated depending on the coding modes of the top and left neighbouring blocks as follows:

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

[0093] The CIIP prediction is formed as follows:

$$P_{CIIP} = ((4 - wt) * P_{inter} + wt * P_{intra} + 2) \gg 2$$

2.4. Chroma QP table design in VTM-6.0

[0094] A chroma QP table design presented in JVET-O0650 was adopted into VVC. It proposes a signalling mechanism for chroma QP tables, which enables that it is flexible to provide encoders the opportunity to optimize the table for SDR and HDR content. It supports for signalling the tables separately for Cb and Cr components. The proposed mechanism signals the chroma QP table as a piece-wise linear function.

2.5. Transform Skip(TS)

[0095] As in HEVC, the residual of a block can be coded with transform skip mode. To avoid the redundancy of syntax coding, the transform skip flag is not signalled when the CU level MTS_CU_flag is not equal to zero. The block size limitation for transform skip is the same to that for

MTS in JEM4, which indicate that transform skip is applicable for a CU when both block width and height are equal to or less than 32. Note that implicit MTS transform is set to DCT2 when LFNST or MIP is activated for the current CU. Also the implicit MTS can be still enabled when MTS is enabled for inter coded blocks.

[0096] In addition, for transform skip block, minimum allowed Quantization Parameter (QP) is defined as $6 * (\text{internalBitDepth} - \text{inputBitDepth}) + 4$.

2.6. Joint coding of chroma residuals (JCCR)

[0097] VVC Draft 6 supports a mode where the chroma residuals are coded jointly. The usage (activation) of a joint chroma coding mode is indicated by a TU-level flag *tu_joint_cbr_residual_flag* and the selected mode is implicitly indicated by the chroma CBFs. The flag *tu_joint_cbr_residual_flag* is present if either or both chroma CBFs for a TU are equal to 1. In the PPS and slice header, chroma QP offset values are signalled for the joint chroma residual coding mode to differentiate from the usual chroma QP offset values signalled for regular chroma residual coding mode. These chroma QP offset values are used to derive the chroma QP values for those blocks coded using the joint chroma residual coding mode. When a corresponding joint chroma coding mode (modes 2 in Table 3) is active in a TU, this chroma QP offset is added to the applied luma-derived chroma QP during quantization and decoding of that TU. For the other modes (modes 1 and 3 in Table 3), the chroma QPs are derived in the same way as for conventional Cb or Cr blocks. The reconstruction process of the chroma residuals (resCb and resCr) from the transmitted transform blocks is depicted in Table 3. When this mode is activated, one single joint chroma residual block (resJointC[x][y] in Table 3) is signalled, and residual block for Cb (resCb) and residual block for Cr (resCr) are derived considering information such as *tu_cbf_cb*, *tu_cbf_cr*, and CSign, which is a sign value specified in the slice header.

[0098] At the encoder side, the joint chroma components are derived as explained in the following. Depending on the mode (listed in the tables above), resJointC{1,2} are generated by the encoder as follows:

- If mode is equal to 2 (single residual with reconstruction $C_b = C$, $C_r = C_{\text{Sign}} * C$), the joint residual is determined according to

$$\text{resJointC}[x][y] = (\text{resCb}[x][y] + C_{\text{Sign}} * \text{resCr}[x][y]) / 2.$$

- Otherwise, if mode is equal to 1 (single residual with reconstruction $C_b = C$, $C_r = (C_{\text{Sign}} * C) / 2$), the joint residual is determined according to

$$\text{resJointC}[x][y] = (4 * \text{resCb}[x][y] + 2 * C_{\text{Sign}} * \text{resCr}[x][y]) / 5.$$

- Otherwise (mode is equal to 3, i. e., single residual, reconstruction $C_r = C$, $C_b = (C_{\text{Sign}} * C) / 2$), the joint residual is determined according to

$$\text{resJointC}[x][y] = (4 * \text{resCr}[x][y] + 2 * C_{\text{Sign}} * \text{resCb}[x][y]) / 5.$$

Table 3 Reconstruction of chroma residuals. The value C_{Sign} is a sign value (+1 or -1), which is specified in the slice header, $\text{resJointC}[x][y]$ is the transmitted residual.

tu_cbf_cb	tu_cbf_cr	reconstruction of Cb and Cr residuals	mode
1	0	$\text{resCb}[x][y] = \text{resJointC}[x][y]$ $\text{resCr}[x][y] = (C_{\text{Sign}} * \text{resJointC}[x][y]) \gg 1$	1
1	1	$\text{resCb}[x][y] = \text{resJointC}[x][y]$ $\text{resCr}[x][y] = C_{\text{Sign}} * \text{resJointC}[x][y]$	2
0	1	$\text{resCb}[x][y] = (C_{\text{Sign}} * \text{resJointC}[x][y]) \gg 1$ $\text{resCr}[x][y] = \text{resJointC}[x][y]$	3

[0099] Different QPs are utilized are the above three modes. For mode 2, the QP offset signaled in PPS for JCCR coded block is applied, while for other two modes, it is not applied, instead, the QP offset signaled in PPS for non-JCCR coded block is applied.

[0100] The corresponding specification is as follows:

8.7.1 Derivation process for quantization parameters

The variable Q_{pY} is derived as follows:

$$Q_{pY} = ((q_{pY_PRED} + CuQpDeltaVal + 64 + 2 * QpBdOffset_Y) \% (64 + QpBdOffset_Y)) - QpBdOffset_Y \quad (8-933)$$

The luma quantization parameter $Q_{p'Y}$ is derived as follows:

$$Q_{p'Y} = Q_{pY} + QpBdOffset_Y \quad (8-934)$$

When ChromaArrayType is not equal to 0 and treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies:

- When treeType is equal to DUAL_TREE_CHROMA, the variable Q_{pY} is set equal to the luma quantization parameter Q_{pY} of the luma coding unit that covers the luma location $(xCb + cbWidth / 2, yCb + cbHeight / 2)$.
- The variables q_{pCb} , q_{pCr} and q_{pCbCr} are derived as follows:

$$q_{p_{Chroma}} = Clip3(-QpBdOffset_C, 63, Q_{pY}) \quad (8-935)$$

$$q_{p_{Cb}} = ChromaQpTable[0][q_{p_{Chroma}}] \quad (8-936)$$

$$q_{p_{Cr}} = ChromaQpTable[1][q_{p_{Chroma}}] \quad (8-937)$$

$$q_{p_{CbCr}} = ChromaQpTable[2][q_{p_{Chroma}}] \quad (8-938)$$

- The chroma quantization parameters for the Cb and Cr components, $Q_{p'Cb}$ and $Q_{p'Cr}$, and joint Cb-Cr coding $Q_{p'CbCr}$ are derived as follows:

$$Q_{p'Cb} = Clip3(-QpBdOffset_C, 63, q_{p_{Cb}} + pps_cb_qp_offset + slice_cb_qp_offset + CuQpOffset_{Cb}) + QpBdOffset_C \quad (8-939)$$

$$Qp'_{Cr} = \text{Clip3}(-QpBdOffset_C, 63, qP_{Cr} + pps_cr_qp_offset + slice_cr_qp_offset + CuQpOffset_{Cr}) + QpBdOffset_C \quad (8-940)$$

$$Qp'_{CbCr} = \text{Clip3}(-QpBdOffset_C, 63, qP_{CbCr} + pps_cbcr_qp_offset + slice_cbcr_qp_offset + CuQpOffset_{CbCr}) + QpBdOffset_C \quad (8-941)$$

8.7.3 Scaling process for transform coefficients

Inputs to this process are:

- a luma location ($xTbY, yTbY$) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture,
- a variable $nTbW$ specifying the transform block width,
- a variable $nTbH$ specifying the transform block height,
- a variable $cIdx$ specifying the colour component of the current block,
- a variable $bitDepth$ specifying the bit depth of the current colour component.

Output of this process is the $(nTbW) \times (nTbH)$ array d of scaled transform coefficients with elements $d[x][y]$.

The quantization parameter qP is derived as follows:

- If $cIdx$ is equal to 0 and $transform_skip_flag[xTbY][yTbY]$ is equal to 0, the following applies:

$$qP = Qp'_Y \quad (8-950)$$

- Otherwise, if $cIdx$ is equal to 0 (and $transform_skip_flag[xTbY][yTbY]$ is equal to 1), the following applies:

$$qP = \text{Max}(QpPrimeTsMin, Qp'_Y) \quad (8-951)$$

- Otherwise, if $TuCResMode[xTbY][yTbY]$ is equal to 2, the following applies:

$$qP = Qp'_{CbCr} \quad (8-952)$$

- Otherwise, if cIdx is equal to 1, the following applies:

$$qP = Qp'_{Cb} \quad (8-953)$$

- Otherwise (cIdx is equal to 2), the following applies:

$$qP = Qp'_{Cr} \quad (8-954)$$

2.7. Cross-Component Adaptive Loop Filter (CC-ALF)

FIG. 14A below illustrates the placement of CC-ALF with respect to the other loop filters. CC-ALF operates by applying a linear, diamond shaped filter (FIG. 14B) to the luma channel for each chroma component, which is expressed as

$$\Delta I_i(x, y) = \sum_{(x_0, y_0) \in S_i} I_0(x_c + x_0, y_c + y_0) c_i(x_0, y_0),$$

where

(x, y) is chroma component i location being refined

(x_c, y_c) is the luma location based on (x, y)

S_i is filter support in luma for chroma component i

$c_i(x_0, y_0)$ represents the filter coefficients

Key features characteristics of the CC-ALF process include:

- The luma location (x_c, y_c) , around which the support region is centered, is computed based on the spatial scaling factor between the luma and chroma planes.
- All filter coefficients are transmitted in the APS and have 8-bit dynamic range.
- An APS may be referenced in the slice header.
- CC-ALF coefficients used for each chroma component of a slice are also stored in a buffer corresponding to a temporal sublayer. Reuse of these sets of temporal sublayer filter coefficients is facilitated using slice-level flags.
- The application of the CC-ALF filters is controlled on a variable block size and signalled by a context-coded flag received for each block of samples. The block size along with an CC-ALF enabling flag is received at the slice-level for each chroma component.

- Boundary padding for the horizontal virtual boundaries makes use of repetition. For the remaining boundaries the same type of padding is used as for regular ALF.

2.8 Derivation process for quantization parameters

The QP is derived based on neighboring QPs and the decoded delta QP. The texts related to QP derivation in JVET-P2001-vE is given as follows.

Inputs to this process are:

- a luma location (x_{Cb} , y_{Cb}) specifying the top-left luma sample of the current 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 variable $treeType$ specifying whether a single tree (SINGLE_TREE) or a dual tree is used to partition the coding tree node and, when a dual tree is used, whether the luma (DUAL_TREE_LUMA) or chroma components (DUAL_TREE_CHROMA) are currently processed. In this process, the luma quantization parameter $Qp'Y$ and the chroma quantization parameters $Qp'Cb$, $Qp'Cr$ and $Qp'CbCr$ are derived.

The luma location (x_{Qg} , y_{Qg}), specifies the top-left luma sample of the current quantization group relative to the top left luma sample of the current picture. The horizontal and vertical positions x_{Qg} and y_{Qg} are set equal to $CuQgTopLeftX$ and $CuQgTopLeftY$, respectively.

NOTE – : The current quantization group is a rectangular region inside a coding tree block that shares the same qPY_PRED . Its width and height are equal to the width and height of the coding tree node of which the top-left luma sample position is assigned to the variables $CuQgTopLeftX$ and $CuQgTopLeftY$.

When $treeType$ is equal to SINGLE_TREE or DUAL_TREE_LUMA, the predicted luma quantization parameter qPY_PRED is derived by the following ordered steps:

1. The variable qPY_PREV is derived as follows:
 - If one or more of the following conditions are true, qPY_PREV is set equal to $SliceQpY$:
 - The current quantization group is the first quantization group in a slice.
 - The current quantization group is the first quantization group in a tile.

- The current quantization group is the first quantization group in a CTB row of a tile and `entropy_coding_sync_enabled_flag` is equal to 1.
 - Otherwise, `qPY_PREV` is set equal to the luma quantization parameter `QpY` of the last luma coding unit in the previous quantization group in decoding order.
2. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (x_{Curr}, y_{Curr}) set equal to (x_{Cb}, y_{Cb}) , the neighbouring location (x_{NbY}, y_{NbY}) set equal to $(x_{Qg} - 1, y_{Qg})$, `checkPredModeY` set equal to `FALSE`, and `cIdx` set equal to 0 as inputs, and the output is assigned to `availableA`. The variable `qPY_A` is derived as follows:
- If one or more of the following conditions are true, `qPY_A` is set equal to `qPY_PREV`:
 - `availableA` is equal to `FALSE`.
 - The CTB containing the luma coding block covering the luma location $(x_{Qg} - 1, y_{Qg})$ is not equal to the CTB containing the current luma coding block at (x_{Cb}, y_{Cb}) , i.e. all of the following conditions are true:
 - $(x_{Qg} - 1) \gg CtbLog2SizeY$ is not equal to $(x_{Cb}) \gg CtbLog2SizeY$
 - $(y_{Qg}) \gg CtbLog2SizeY$ is not equal to $(y_{Cb}) \gg CtbLog2SizeY$
 - Otherwise, `qPY_A` is set equal to the luma quantization parameter `QpY` of the coding unit containing the luma coding block covering $(x_{Qg} - 1, y_{Qg})$.
3. The derivation process for neighbouring block availability as specified in clause 6.4.4 is invoked with the location (x_{Curr}, y_{Curr}) set equal to (x_{Cb}, y_{Cb}) , the neighbouring location (x_{NbY}, y_{NbY}) set equal to $(x_{Qg}, y_{Qg} - 1)$, `checkPredModeY` set equal to `FALSE`, and `cIdx` set equal to 0 as inputs, and the output is assigned to `availableB`. The variable `qPY_B` is derived as follows:
- If one or more of the following conditions are true, `qPY_B` is set equal to `qPY_PREV`:
 - `availableB` is equal to `FALSE`.
 - The CTB containing the luma coding block covering the luma location $(x_{Qg}, y_{Qg} - 1)$ is not equal to the CTB containing the current luma coding block at (x_{Cb}, y_{Cb}) , i.e. all of the following conditions are true:
 - $(x_{Qg}) \gg CtbLog2SizeY$ is not equal to $(x_{Cb}) \gg CtbLog2SizeY$
 - $(y_{Qg} - 1) \gg CtbLog2SizeY$ is not equal to $(y_{Cb}) \gg CtbLog2SizeY$
 - Otherwise, `qPY_B` is set equal to the luma quantization parameter `QpY` of the coding unit containing the luma coding block covering $(x_{Qg}, y_{Qg} - 1)$.

4. The predicted luma quantization parameter qPY_PRED is derived as follows:
- If all the following conditions are true, then qPY_PRED is set equal to the luma quantization parameter QpY of the coding unit containing the luma coding block covering ($xQg, yQg - 1$):
 - availableB is equal to TRUE.
 - The current quantization group is the first quantization group in a CTB row within a tile.
 - Otherwise, qPY_PRED is derived as follows:

$$qPY_PRED = (qPY_A + qPY_B + 1) >> 1 \quad (1115)$$

The variable QpY is derived as follows:

$$QpY = ((qPY_PRED + CuQpDeltaVal + 64 + 2 * QpBdOffset) \% (64 + QpBdOffset)) - QpBdOffset \quad (1116)$$

The luma quantization parameter $Qp'Y$ is derived as follows:

$$Qp'Y = QpY + QpBdOffset \quad (1117)$$

When ChromaArrayType is not equal to 0 and treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies:

- When treeType is equal to DUAL_TREE_CHROMA, the variable QpY is set equal to the luma quantization parameter QpY of the luma coding unit that covers the luma location ($xCb + cbWidth / 2, yCb + cbHeight / 2$).

- The variables $qPCb$, $qPCr$ and $qPCbCr$ are derived as follows:

$$qPChroma = Clip3(-QpBdOffset, 63, QpY) \quad (1118)$$

$$qPCb = ChromaQpTable[0][qPChroma] \quad (1119)$$

$$qPCr = ChromaQpTable[1][qPChroma] \quad (1120)$$

$$qPCbCr = ChromaQpTable[2][qPChroma] \quad (1121)$$

- The chroma quantization parameters for the Cb and Cr components, $Qp'Cb$ and $Qp'Cr$, and joint Cb-Cr coding $Qp'CbCr$ are derived as follows:

$$Qp'Cb = Clip3(-QpBdOffset, 63, qPCb + pps_cb_qp_offset + slice_cb_qp_offset + CuQpOffsetCb) + QpBdOffset \quad (1122)$$

$$Qp'Cr = Clip3(-QpBdOffset, 63, qPCr + pps_cr_qp_offset + slice_cr_qp_offset + CuQpOffsetCr) + QpBdOffset \quad (1123)$$

$$Qp'CbCr = Clip3(-QpBdOffset, 63, qPCbCr + pps_joint_cbcr_qp_offset + slice_joint_cbcr_qp_offset + CuQpOffsetCbCr) + QpBdOffset \quad (1124)$$

2.9 Adaptive color transform (ACT)

FIG. 15 illustrates the decoding flowchart with the ACT be applied. As illustrated in FIG. 1, the color space conversion is carried out in residual domain. Specifically, one additional decoding module, namely inverse ACT, is introduced after inverse transform to convert the residuals from YCgCo domain back to the original domain.

In the VVC, unless the maximum transform size is smaller than the width or height of one coding unit (CU), one CU leaf node is also used as the unit of transform processing. Therefore, in the proposed implementation, the ACT flag is signaled for one CU to select the color space for coding its residuals. Additionally, following the HEVC ACT design, for inter and IBC CUs, the ACT is only enabled when there is at least one non-zero coefficient in the CU. For intra CUs, the ACT is only enabled when chroma components select the same intra prediction mode of luma component, i.e., DM mode.

The core transforms used for the color space conversions are kept the same as that used for the HEVC. Specifically, the following forward and inverse YCgCo color transform matrices, as described as follows, as applied.

$$\begin{bmatrix} C'_0 \\ C'_1 \\ C'_2 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 1 \\ 2 & -1 & -1 \\ 0 & -2 & 2 \end{bmatrix} \begin{bmatrix} C_0 \\ C_1 \\ C_2 \end{bmatrix} / 4 \quad \begin{bmatrix} C_0 \\ C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 1 & -1 & -1 \\ 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} C'_0 \\ C'_1 \\ C'_2 \end{bmatrix}$$

Additionally, to compensate the dynamic range change of residuals signals before and after color transform, the QP adjustments of (-5, -5, -3) are applied to the transform residuals.

On the other hand, as shown in FIG. 15), the forward and inverse color transforms need to access the residuals of all three components. Correspondingly, in the proposed implementation, the ACT is disabled in the following two scenarios where not all residuals of three components are available.

1. *Separate-tree partition*: when separate-tree is applied, luma and chroma samples inside one CTU are partitioned by different structures. This results in that the CUs in the luma-tree only contains luma component and the CUs in the chroma-tree only contains two chroma components.

Intra sub-partition prediction (ISP): the ISP sub-partition is only applied to luma while chroma signals are coded without splitting. In the current ISP design, except the last ISP sub-partitions, the other sub-partitions only contain luma component.

2.10 High-level deblocking control in the VVC draft 7

The control mechanism is shown in the following FIG. 16.

2.11 Cross-Component Adaptive Loop Filter (CC-ALF)

FIG. 14A illustrates the placement of CC-ALF with respect to the other loop filters. CC-ALF operates by applying a linear, diamond shaped filter (FIG. 14B) to the luma channel for each chroma component, which is expressed as

$$\Delta I_i(x, y) = \sum_{(x_0, y_0) \in S_i} I_0(x_c + x_0, y_c + y_0) c_i(x_0, y_0),$$

where

(x, y) is chroma component i location being refined

(x_c, y_c) is the luma location based on (x, y)

S_i is filter support in luma for chroma component i

$c_i(x_0, y_0)$ represents the filter coefficients (2-3)

The luma location (x_c, y_c) , around which the support region is centered, is computed based on the spatial scaling factor between the luma and chroma planes. All filter coefficients are transmitted in the APS and have 8-bit dynamic range. An APS may be referenced in the slice header. CC-ALF coefficients used for each chroma component of a slice are also stored in a buffer corresponding to a temporal sublayer. Reuse of these sets of temporal sublayer filter coefficients is facilitated using slice-level flags. The application of the CC-ALF filters is controlled on a variable block size (i.e. 16×16 , 32×32 , 64×64 , 128×128) and signalled by a context-coded flag received for each block of samples. The block size along with an CC-ALF enabling flag is received at the slice-level for each chroma component. Boundary padding for the horizontal virtual boundaries makes use of repetition. For the remaining boundaries the same type of padding is used as for regular ALF.

2.11.1 Syntax design of CC-ALF in JVET-Q0058

7.3.2.6 Picture header RBSP syntax

picture_header_rbsp() {	Descriptor
non_reference_picture_flag	u(1)
gdr_pic_flag	u(1)
...	
if(sps_triangle_enabled_flag && MaxNumMergeCand >= 2 && !pps_max_num_merge_cand_minus_max_num_triangle_cand_plus1)	
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)
}	
pic_cross_component_alf_cb_enabled_flag	u(1)
if(pic_cross_component_alf_cb_enabled_flag) {	
pic_cross_component_alf_cb_aps_id	u(3)
pic_cross_component_cb_filters_signalled_minus1	ue(v)
}	
pic_cross_component_alf_cr_enabled_flag	u(1)
if(pic_cross_component_alf_cr_enabled_flag) {	
pic_cross_component_alf_cr_aps_id	u(3)
pic_cross_component_cr_filters_signalled_minus1	ue(v)
}	
}	
}	
if(!pps_dep_quant_enabled_flag)	
pic_dep_quant_enabled_flag	u(1)
...	
rbsp_trailing_bits()	
}	

7.3.2.16 Adaptive loop filter data syntax

alf_data() {	Descriptor
alf_luma_filter_signal_flag	u(1)
alf_chroma_filter_signal_flag	u(1)
alf_cross_component_cb_filter_signal_flag	u(1)
alf_cross_component_cr_filter_signal_flag	u(1)
if(alf_luma_filter_signal_flag) {	
alf_luma_clip_flag	u(1)
alf_luma_num_filters_signalled_minus1	ue(v)
if(alf_luma_num_filters_signalled_minus1 > 0) {	
for(filtIdx = 0; filtIdx < NumAlfFilters; filtIdx++)	
alf_luma_coeff_delta_idx[filtIdx]	u(v)
}	
for(sfIdx = 0; sfIdx <= alf_luma_num_filters_signalled_minus1; sfIdx++)	
{	
for(j = 0; j < 12; j++) {	
alf_luma_coeff_abs[sfIdx][j]	uek(v)
if(alf_luma_coeff_abs[sfIdx][j])	
alf_luma_coeff_sign[sfIdx][j]	u(1)
}	
}	
if(alf_luma_clip_flag) {	
for(sfIdx = 0; sfIdx <= alf_luma_num_filters_signalled_minus1; sfIdx++)	
{	
for(j = 0; j < 12; j++)	
alf_luma_clip_idx[sfIdx][j]	u(2)

}	
}	
}	
if(alf_chroma_filter_signal_flag) {	
...	
}	
}	
if (alf_cross_component_cb_filter_signal_flag) {	
alf_cross_component_cb_filters_signalled_minus1	ue(v)
for(k = 0; k < (alf_cross_component_cb_filters_signalled_minus1+1); k++)	
{	
for (j = 0; j < 8; j++)	
alf_cross_component_cb_coeff_plus32[k][j]	u(6)
}	
}	
if (alf_cross_component_cr_filter_signal_flag) {	
alf_cross_component_cr_filters_signalled_minus1	ue(v)
for(k = 0; k < (alf_cross_component_cr_filters_signalled_minus1+1); k++)	
{	
for (j = 0; j < 8; j++)	
alf_cross_component_cr_coeff_plus32[k][j]	u(6)
}	
}	
}	

7.3.7 Slice header syntax

slice_header() {	Descriptor
slice_pic_order_cnt_lsb	u(v)
if(subpics_present_flag)	
slice_subpic_id	u(v)
if(rect_slice_flag NumTilesInPic > 1)	
slice_address	u(v)
if(!rect_slice_flag && NumTilesInPic > 1)	
num_tiles_in_slice_minus1	ue(v)
slice_type	ue(v)
...	
if(sps_alf_enabled_flag && !pic_alf_enabled_present_flag) {	
slice_alf_enabled_flag	u(1)
if(slice_alf_enabled_flag) {	
slice_num_alf_aps_ids_luma	u(3)
for(i = 0; i < slice_num_alf_aps_ids_luma; i++)	
slice_alf_aps_id_luma[i]	u(3)
if(ChromaArrayType != 0)	
slice_alf_chroma_idc	u(2)
if(slice_alf_chroma_idc)	
slice_alf_aps_id_chroma	u(3)
}	
if(ChromaArrayType != 0)	
slice_cross_component_alf_cb_enabled_flag	u(1)
if(slice_cross_component_alf_cb_enabled_flag) {	
slice_cross_component_alf_cb_aps_id	u(3)

slice_cross_component_cb_filters_signalled_minus1	ue(v)
}	
if(ChromaArrayType != 0)	
slice_cross_component_alf_cr_enabled_flag	u(1)
if(slice_cross_component_alf_cr_enabled_flag) {	
slice_cross_component_alf_cr_aps_id	u(3)
slice_cross_component_cr_filters_signalled_minus1	ue(v)
}	
}	
if(deblocking_filter_override_enabled_flag && !pic_deblocking_filter_override_present_flag)	
slice_deblocking_filter_override_flag	u(1)
if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
}	
}	
if(entry_point_offsets_present_flag && NumEntryPoints > 0) {	
offset_len_minus1	ue(v)
for(i = 0; i < NumEntryPoints; i++)	
entry_point_offset_minus1[i]	u(v)

}	
if(slice_header_extension_present_flag) {	
slice_header_extension_length	ue(v)
for(i = 0; i < slice_header_extension_length; i++)	
slice_header_extension_data_byte[i]	u(8)
}	
byte_alignment()	
}	

pic_cross_component_alf_cb_enabled_flag equal to 1 specifies that cross component Cb filter is enabled for all slices associated with the PH and may be applied to Cb colour component in the slices. **pic_cross_component_alf_cb_enabled_flag** equal to 0 specifies that cross component Cb filter may be disabled for one, or more, or all slices associated with the PH. When not present, **pic_cross_component_alf_cb_enabled_flag** is inferred to be equal to 0.

pic_cross_component_alf_cb_aps_id specifies the **adaptation_parameter_set_id** of the ALF APS that the Cb colour component of the slices associated with the PH refers to.

The value of **alf_cross_component_cb_filter_signal_flag** of the APS NAL unit having **aps_params_type** equal to **ALF_APS** and **adaptation_parameter_set_id** equal to **pic_cross_component_alf_cb_aps_id** shall be equal to 1.

pic_cross_component_cb_filters_signalled_minus1 plus 1 specifies the number of cross component Cb filters. The value of **pic_cross_component_cb_filters_signalled_minus1** shall be in the range 0 to 3

When **pic_cross_component_alf_cb_enabled_flag** equal to 1, it is a requirement of bitstream conformance that **pic_cross_component_cb_filters_signalled_minus1** shall be less than or equal to the value of **alf_cross_component_cb_filters_signalled_minus1** in the referenced ALF APS referred to by **pic_cross_component_alf_cb_aps_id**.

pic_cross_component_alf_cr_enabled_flag equal to 1 specifies that cross component Cr filter is enabled for all slices associated with the PH and may be applied to Cr colour component in the slices. **pic_cross_component_alf_cr_enabled_flag** equal to 0 specifies that cross component Cr filter may be

disabled for one, or more, or all slices associated with the PH. When not present, `pic_cross_component_alf_cr_enabled_flag` is inferred to be equal to 0.

`pic_cross_component_alf_cr_aps_id` specifies the `adaptation_parameter_set_id` of the ALF APS that the Cr colour component of the slices associated with the PH refers to.

The value of `alf_cross_component_cr_filter_signal_flag` of the APS NAL unit having `aps_params_type` equal to `ALF_APS` and `adaptation_parameter_set_id` equal to `pic_cross_component_alf_cr_aps_id` shall be equal to 1.

`pic_cross_component_cr_filters_signalled_minus1` plus 1 specifies the number of cross component Cr filters. The value of `pic_cross_component_cr_filters_signalled_minus1` shall be in the range 0 to 3

When `pic_cross_component_alf_cr_enabled_flag` equal to 1, it is a requirement of bitstream conformance that `pic_cross_component_cr_filters_signalled_minus1` shall be less than or equal to the value of `alf_cross_component_cr_filters_signalled_minus1` in the referenced ALF APS referred to by `pic_cross_component_alf_cr_aps_id`.

`alf_cross_component_cb_filter_signal_flag` equal to 1 specifies that a cross component Cb filter is signalled. `alf_cross_component_cb_filter_signal_flag` equal to 0 specifies that a cross component Cb filter is not signalled. When `ChromaArrayType` is equal to 0, `alf_cross_component_cb_filter_signal_flag` shall be equal to 0.

`alf_cross_component_cb_filters_signalled_minus1` plus 1 specifies the number of cross component Cb filters signalled in the current ALF APS. The value of `alf_cross_component_cb_filters_signalled_minus1` shall be in the range 0 to 3.

`alf_cross_component_cb_coeff_plus32[ k ][ j ]` minus 32 specifies the value of the j-th coefficient of the signalled k-th cross-component Cb filter set. When `alf_cross_component_cb_coeff_plus32[ k ][ j ]` is not present, it is inferred to be equal to 32.

The signalled k-th cross component Cb filter coefficients `CcAlfApsCoeffCb[ adaptation_parameter_set_id ][ k ]` with elements `CcAlfApsCoeffCb[ adaptation_parameter_set_id ][ k ][ j ]`, with $j = 0..7$ are derived as follows:

$$CcAlfApsCoeffCb[adaptation_parameter_set_id][k][j] = alf_cross_component_cb_coeff_plus32[k][j] - 32 \quad (7-51)$$

alf_cross_component_cr_filter_signal_flag equal to 1 specifies that a cross component Cr filter is signalled. **alf_cross_component_cr_filter_signal_flag** equal to 0 specifies that a cross component Cr filter is not signalled. When **ChromaArrayType** is equal to 0, **alf_cross_component_cr_filter_signal_flag** shall be equal to 0.

alf_cross_component_cr_filters_signalled_minus1 plus 1 specifies the number of cross component Cr filters signalled in the current ALF APS. The value of **alf_cross_component_cr_filters_signalled_minus1** shall be in the range 0 to 3.

alf_cross_component_cr_coeff_plus32[*k*][*j*] minus 32 specifies the value of the *j*-th coefficient of the signalled *k*-th cross-component Cr filter set. When **alf_cross_component_cr_coeff_abs**[*k*][*j*] is not present, it is inferred to be equal to 32.

The signalled *k*-th cross component Cr filter coefficients $CcAlfApsCoeff_{Cr}[adaptation_parameter_set_id][k]$ with elements $CcAlfApsCoeff_{Cr}[adaptation_parameter_set_id][k][j]$, with $j = 0..7$ are derived as follows:
 $CcAlfApsCoeff_{Cr}[adaptation_parameter_set_id][k][j] = alf_cross_component_cr_coeff_plus32[k][j] - 32$ (7-52)

3. Drawbacks of existing implementations

[0101] DMVR and BIO do not involve the original signal during refining the motion vectors, which may result in coding blocks with inaccurate motion information. Also, DMVR and BIO sometimes employ the fractional motion vectors after the motion refinements while screen videos usually have integer motion vectors, which makes the current motion information more inaccurate and make the coding performance worse.

1. The interaction between chroma QP table and chroma deblocking may have problems, e.g. chroma QP table should be applied to individual QP but not weighted sum of QPs.
2. The logic of luma deblocking filtering process is complicated for hardware design.
3. The logic of boundary strength derivation is too complicated for both software and hardware design.
4. In the BS decision process, JCCR is treated separately from those blocks coded without JCCT applied. However, JCCR is only a special way to code the residual. Therefore, such design may bring additional complexity without no clear benefits.

5. In chroma edge decision, Q_{pQ} and Q_{pP} are set equal to the Q_{pY} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively. However, in the quantization/de-quantization process, the QP for a chroma sample is derived from the QP of luma block covering the corresponding luma sample of the center position of current chroma CU. When dual tree is enabled, the different locations of luma blocks may result in different QPs. Therefore, in the chroma deblocking process, wrong QPs may be used for filter decision. Such a misalignment may result in visual artifacts. An example is shown in FIG. 9A-9B, including FIGS. 9A and 9B. In FIG. 9A-9B, the left side (FIG. 9A) is the corresponding CTB partitioning for luma block and the right side (FIG. 9B) in the chroma CTB partitioning under dual tree. When determining the QP for chroma block, denoted by CU_c1 , the center position of CU_c1 is firstly derived. Then the corresponding luma sample of the center position of CU_c1 is identified and luma QP associated with the luma CU that covers the corresponding luma sample, i.e., CU_Y3 is then utilized to derive the QP for CU_c1 . However, when making filter decisions for the depicted three samples (with solid circles), the QPs of CUs that cover the corresponding 3 samples are selected. Therefore, for the 1st, 2nd, and 3rd chroma sample (depicted in FIG. 9B), the QPs of CU_Y2 , CU_Y3 , CU_Y4 are utilized, respectively. That is, chroma samples in the same CU may use different QPs for filter decision which may result in wrong decisions.
6. A different picture level QP offset (i.e., $pps_joint_cbcr_qp_offset$) is applied to JCCR coded blocks which is different from the picture level offsets for Cb/Cr (e.g., $pps_cb_qp_offset$ and $pps_cr_qp_offset$) applied to non-JCCR coded blocks. However, in the chroma deblocking filter decision process, only those offsets for non-JCCR coded blocks are utilized. The missing of consideration of coded modes may result in wrong filter decision.
7. The TS and non-TS coded blocks employ different QPs in the de-quantization process, which may be also considered in the deblocking process.
8. Different QPs are used in the scaling process (quantization/dequantization) for JCCR coded blocks with different modes. Such a design is not consistent.
9. The chroma deblocking for Cb/Cr could be unified for parallel design.

10. The chroma QP in the deblocking is derived based on the QP used in the chroma dequantization process (e.g. qP), however, the qP should be clipped or minus 5 for TS and ACT blocks when it is used in the deblocking process.
11. The prediction process for three components may be not same when both BDPCM and ACT are enabled.
12. Enabling/disabling CC-ALF for a chroma component is signaled in picture header. However, if the color format is 4:0:0, there is no need to signal such information, but it is signaled in current VVC specification.
13. For chroma BDPCM, ACT and Palette modes, the SPS controlling flags are signaled under the condition check of chroma_format_idc being equal to 3. It means that even separate plane coding is used for 4:4:4 color format, those flags still need to be signalled even the picture is treated as monochrome pictures. The related syntax elements are defined as follows:

7.3.2.3 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp() {	Descript or
...	
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)
...	
if(chroma_format_idc == 3) {	
sps_palette_enabled_flag	u(1)
sps_act_enabled_flag	u(1)
}	
...	

rbsp_trailing_bits()	
}	

4. Example techniques and embodiments

[0102] The detailed embodiments described below should be considered as examples to explain general concepts. These embodiments should not be interpreted narrowly way. Furthermore, these embodiments can be combined in any manner.

[0103] The proposed methods described below may be applied to the deblocking filter. Alternatively, they may be applied to other kinds of in-loop filters, e.g., those rely on quantization parameters.

[0104] The methods described below may be also applicable to other decoder motion information derivation technologies in addition to the DMVR and BIO mentioned below.

[0105] In the following examples, $MVM[i].x$ and $MVM[i].y$ denote the horizontal and vertical component of the motion vector in reference picture list i (i being 0 or 1) of the block at M (M being P or Q) side. Abs denotes the operation to get the absolute value of an input, and “&&” and “||” denotes the logical operation AND and OR. Referring to FIG. 10, P may denote the samples at P side and Q may denote the samples at Q side. The blocks at P side and Q side may denote the block marked by the dash lines.

Regarding chroma QP in deblocking

1. When chroma QP table is used to derive parameters to control chroma deblocking (e.g., in the decision process for chroma block edges), chroma QP offsets may be applied after applying chroma QP table.
 - a. In one example, the chroma QP offsets may be added to the value outputted by a chroma QP table.
 - b. Alternatively, the chroma QP offsets may be not considered as input to a chroma QP table.

- c. In one example, the chroma QP offsets may be the picture-level or other video unit-level (slice/tile/brick/subpicture) chroma quantization parameter offset (e.g., **pps_cb_qp_offset**, **pps_cr_qp_offset** in the specification).
- 2. QP clipping may be not applied to the input of a chroma QP table.
- 3. It is proposed that deblocking process for chroma components may be based on the mapped chroma QP (by the chroma QP table) on each side.
 - a. In one example, it is proposed that deblocking parameters, (e.g., β and tC) for chroma may be based on QP derived from luma QP on each side.
 - b. In one example, the chroma deblocking parameter may depend on chroma QP table value with QpP as the table index, where QpP, is the luma QP value on P-side.
 - c. In one example, the chroma deblocking parameter may depend on chroma QP table value with QpQ as the table index, where QpQ, is the luma QP value on Q-side.
- 4. It is proposed that deblocking process for chroma components may be based on the QP applied to quantization/dequantization for the chroma block.
 - a. In one example, QP for deblocking process may be equal to the QP in dequantization.
 - b. In one example, the QP selection for deblocking process may depend on the indication of usage of TS and/or ACT blocks.
 - i. In one example, the QP for the deblocking process may be derived by $\text{Max}(\text{QpPrimeTsMin}, qP) - (\text{cu_act_enabled_flag}[\text{xTbY}][\text{yTbY}]? N:0)$, where QpPrimeTsMin is the minimal QP for TS blocks and cu_act_enabled_flag is the flag of usage of ACT.
 - 1. In one example, qP may be the Qp'_{Cb} or Qp'_{Cr} given in section 2.8.
 - ii. In one example, the QP for the deblocking process may be derived by $\text{Max}(\text{QpPrimeTsMin}, qP - (\text{cu_act_enabled_flag}[\text{xTbY}][\text{yTbY}]? N:0))$, where QpPrimeTsMin is the minimal QP for TS blocks and cu_act_enabled_flag is the flag of usage of ACT.
 - 1. In one example, qP may be the Qp'_{Cb} or Qp'_{Cr} given in section 2.8.
 - iii. In the above examples, N may be set to same or different values for each color component.

1. In one example, N may be set to 5 for the Cb/B/G/U/ component and/or N may be set to 3 for the Cr/R/B/V component.
- c. In one example, the QP of a block used in the deblocking process may be derived by the process described in section 2.8 with delta QP (e.g. *CuQpDeltaVal*) equal to 0.
 - i. In one example, the above derivation may be applied only when the coded block flag (cbf) of a block is equal to 0.
- d. In one example, the above examples may be applied on luma and/or chroma blocks.
- e. In one example, the QP of a first block used in the deblocking process may be set equal to the QP stored and utilized for predicting a second block.
 - i. In one example, for a block with all zero coefficients, the associated QP used in the deblocking process may be set equal to the QP stored and utilized for predicting a second block.
5. It is proposed to consider the picture/slice/tile/brick/subpicture level quantization parameter offsets used for different coding methods in the deblocking filter decision process.
 - a. In one example, selection of picture/slice/tile/brick/subpicture level quantization parameter offsets for filter decision (e.g., the chroma edge decision in the deblocking filter process) may depend on the coded methods for each side.
 - b. In one example, the filtering process (e.g., the chroma edge decision process) which requires to use the quantization parameters for chroma blocks may depend on whether the blocks use JCCR.
 - i. Alternatively, furthermore, the picture/slice-level QP offsets (e.g., **pps_joint_cbr_qp_offset**) applied to JCCR coded blocks may be further taken into consideration in the deblocking filtering process.
 - ii. In one example, the **cQpPicOffset** which is used to decide T_c and β settings may be set to **pps_joint_cbr_qp_offset** instead of **pps_cb_qp_offset** or **pps_cr_qp_offset** under certain conditions:
 1. In one example, when either block in P or Q sides uses JCCR.
 2. In one example, when both blocks in P or Q sides uses JCCR.
 - iii. Alternatively, furthermore, the filtering process may depend on the mode of JCCR (e.g., whether mode is equal to 2).

6. The chroma filtering process (e.g., the chroma edge decision process) which requires to access the decoded information of a luma block may utilize the information associated with the same luma coding block that is used to derive the chroma QP in the dequantization/quantization process.
 - a. In one example, the chroma filtering process (e.g., the chroma edge decision process) which requires to use the quantization parameters for luma blocks may utilize the luma coding unit covering the corresponding luma sample of the center position of current chroma CU.
 - b. An example is depicted in FIGS. 9A-9B wherein the decoded information of CU_{Y3} may be used for filtering decision of the three chroma samples (1st, 2nd and 3rd) in FIG. 9B.
7. The chroma filtering process (e.g., the chroma edge decision process) may depend on the quantization parameter applied to the scaling process of the chroma block (e.g., quantization/dequantization).
 - a. In one example, the QP used to derive β and Tc may depend on the QP applied to the scaling process of the chroma block.
 - b. Alternatively, furthermore, the QP used to the scaling process of the chroma block may have taken the chroma CU level QP offset into consideration.
8. Whether to invoke above bullets may depend on the sample to be filtered is in the block at P or Q side.
 - a. For example, whether to use the information of the luma coding block covering the corresponding luma sample of current chroma sample or use the information of the luma coding block covering the corresponding luma sample of center position of chroma coding block covering current chroma sample may depend on the block position.
 - i. In one example, if the current chroma sample is in the block at the Q side, QP information of the luma coding block covering the corresponding luma sample of center position of chroma coding block covering current chroma sample may be used.

- ii. In one example, if the current chroma sample is in the block at the P side, QP information of the luma coding block covering the corresponding luma sample of the chroma sample may be used.
- 9. Chroma QP used in deblocking may depend on information of the corresponding transform block.
 - a. In one example, chroma QP for deblocking at P-side may depend on the transform block's mode at P-side.
 - i. In one example, chroma QP for deblocking at P-side may depend on if the transform block at P-side is coded with JCCR applied.
 - ii. In one example, chroma QP for deblocking at P-side may depend on if the transform block at P-side is coded with joint_cb_cr mode and the mode of JCCR is equal to 2.
 - b. In one example, chroma QP for deblocking at Q-side may depend on the transform block's mode at Q-side.
 - i. In one example, chroma QP for deblocking at Q-side may depend on if the transform block at Q-side is coded with JCCR applied.
 - ii. In one example, chroma QP for deblocking at Q-side may depend on if the transform block at Q-side is coded with JCCR applied and the mode of JCCR is equal to 2.
- 10. Signaling of chroma QPs may be in coding unit.
 - a. In one example, when coding unit size is larger than the maximum transform block size, i.e., maxTB, chroma QP may be signaled in CU level. Alternatively, it may be signaled in TU level.
 - b. In one example, when coding unit size is larger than the size of VPDU, chroma QP may be signaled in CU level. Alternatively, it may be signaled in TU level.
- 11. Whether a block is of joint_cb_cr mode may be indicated at coding unit level.
 - a. In one example, whether a transform block is of joint_cb_cr mode may inherit the information of the coding unit containing the transform block.
- 12. Chroma QP used in deblocking may depend on chroma QP used in scaling process minus QP offset due to bit depth.

- a. In one example, Chroma QP used in deblocking at P-side is set to the JCCR chroma QP used in scaling process, i.e. Qp'_{CbCr} , minus $QpBdOffsetC$ when $TuCResMode[xTb][yTb]$ is equal to 2 where (xTb, yTb) denotes the transform blocking containing the first sample at P-side, i.e. $p_{0,0}$.
 - b. In one example, Chroma QP used in deblocking at P-side is set to the Cb chroma QP used in scaling process, i.e. Qp'_{Cb} , minus $QpBdOffsetC$ when $TuCResMode[xTb][yTb]$ is equal to 2 where (xTb, yTb) denotes the transform blocking containing the first sample at P-side, i.e. $p_{0,0}$.
 - c. In one example, Chroma QP used in deblocking at P-side is set to the Cr chroma QP used in scaling process, i.e. Qp'_{Cr} , minus $QpBdOffsetC$ when $TuCResMode[xTb][yTb]$ is equal to 2 where (xTb, yTb) denotes the transform blocking containing the first sample at P-side, i.e. $p_{0,0}$.
 - d. In one example, Chroma QP used in deblocking at Q-side is set to the JCCR chroma QP used in scaling process, i.e. Qp'_{CbCr} , minus $QpBdOffsetC$ when $TuCResMode[xTb][yTb]$ is equal to 2 where (xTb, yTb) denotes the transform blocking containing the last sample at Q-side, i.e. $q_{0,0}$.
 - e. In one example, Chroma QP used in deblocking at Q-side is set to the Cb chroma QP used in scaling process, i.e. Qp'_{Cb} , minus $QpBdOffsetC$ when $TuCResMode[xTb][yTb]$ is equal to 2 where (xTb, yTb) denotes the transform blocking containing the last sample at Q-side, i.e. $q_{0,0}$.
 - f. In one example, Chroma QP used in deblocking at Q-side is set to the Cr chroma QP used in scaling process, i.e. Qp'_{Cr} , minus $QpBdOffsetC$ when $TuCResMode[xTb][yTb]$ is equal to 2 where (xTb, yTb) denotes the transform blocking containing the last sample at Q-side, i.e. $q_{0,0}$.
13. Different color components may have different deblocking strength control.
- a. In one example, each component may have its **pps_beta_offset_div2**, **pps_tc_offset_div2** and/or **pic_beta_offset_div2**, **pic_tc_offset_div2** and/or **slice_beta_offset_div2**, **slice_tc_offset_div2**.
 - b. In one example, for joint_cb_cr mode, a different set of **beta_offset_div2**, **tc_offset_div2** may be applied in PPS and/or picture header and/or slice header.

14. Instead of using overriding mechanism, the deblocking control offset may be accumulated taking account of offsets at different levels.
- In one example, **pps_beta_offset_div2** and/or **pic_beta_offset_div2** and/or **slice_beta_offset_div2** may be accumulated to get the deblocking offset at slice level.
 - In one example, **pps_tc_offset_div2** and/or **pic_tc_offset_div2** and/or **slice_tc_offset_div2** may be accumulated to get the deblocking offset at slice level.

Regarding QP settings

15. It is proposed to signal the indication of enabling block-level chroma QP offset (e.g. **slice_cu_chroma_qp_offset_enabled_flag**) at the slice/tile/brick/subpicture level.
- Alternatively, the signaling of such an indication may be conditionally signaled.
 - In one example, it may be signaled under the condition of JCCR enabling flag.
 - In one example, it may be signaled under the condition of block-level chroma QP offset enabling flag in picture level.
 - Alternatively, such an indication may be derived instead.
 - In one example, the **slice_cu_chroma_qp_offset_enabled_flag** may be signaled only when the PPS flag of chroma QP offset (e.g. **slice_cu_chroma_qp_offset_enabled_flag**) is true.
 - In one example, the **slice_cu_chroma_qp_offset_enabled_flag** may be inferred to false only when the PPS flag of chroma QP offset (e.g. **slice_cu_chroma_qp_offset_enabled_flag**) is false.
 - In one example, whether to use the chroma QP offset on a block may be based on the flags of chroma QP offset at PPS level and/or slice level.
16. Same QP derivation method is used in the scaling process (quantization/dequantization) for JCCR coded blocks with different modes.
- In one example, for JCCR with mode 1 and 3, the QP is dependent on the QP offset signaled in the picture/slice level (e.g., **pps_cbcr_qp_offset**, **slice_cbcr_qp_offset**).

Filtering procedures

17. Deblocking for all color components excepts for the first color component may follow the deblocking process for the first color component.
- In one example, when the color format is 4:4:4, deblocking process for the second and third components may follow the deblocking process for the first component.
 - In one example, when the color format is 4:4:4 in RGB color space, deblocking process for the second and third components may follow the deblocking process for the first component.
 - In one example, when the color format is 4:2:2, vertical deblocking process for the second and third components may follow the vertical deblocking process for the first component.
 - In above examples, the deblocking process may refer to deblocking decision process and/or deblocking filtering process.
18. How to calculate gradient used in the deblocking filter process may depend on the coded mode information and/or quantization parameters.
- In one example, the gradient computation may only consider the gradient of a side wherein the samples at that side are not lossless coded.
 - In one example, if both sides are lossless coded or nearly lossless coded (e.g., quantization parameters equal to 4), gradient may be directly set to 0.
 - Alternatively, if both sides are lossless coded or nearly lossless coded (e.g., quantization parameters equal to 4), Boundary Strength (e.g., BS) may be set to 0.
 - In one example, if the samples at P side are lossless coded and the samples at Q side are lossy coded, the gradients used in deblocking on/off decision and/or strong filters on/off decision may only include gradients of the samples at Q side, vice versa.
 - Alternatively, furthermore, the gradient of one side may be scaled by N.
 - N is an integer number (e.g. 2) and may depend on
 - Video contents (e.g. screen contents or natural contents)
 - A message signaled in the DPS/SPS/VPS/PPS/APS/picture header/slice header/tile group header/ Largest coding unit

- (LCU)/Coding unit (CU)/LCU row/group of LCUs/TU/PU block/Video coding unit
- c. Position of CU/PU/TU/block/Video coding unit
 - d. Coded modes of blocks containing the samples along the edges
 - e. Transform matrices applied to the blocks containing the samples along the edges
 - f. Block dimension/Block shape of current block and/or its neighboring blocks
 - g. Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)
 - h. Coding tree structure (such as dual tree or single tree)
 - i. Slice/tile group type and/or picture type
 - j. Color component (e.g. may be only applied on Cb or Cr)
 - k. Temporal layer ID
 - l. Profiles/Levels/Tiers of a standard
 - m. Alternatively, N may be signalled to the decoder

Regarding boundary strength derivation

- 19. It is proposed to treat JCCR coded blocks as those non-JCCR coded blocks in the boundary strength decision process.
 - a. In one example, the determination of boundary strength (BS) may be independent from the checking of usage of JCCR for two blocks at P and Q sides.
 - a. In one example, the boundary strength (BS) for a block may be determined regardless if the block is coded with JCCR or not.
- 20. It is proposed to derive the boundary strength (BS) without comparing the reference pictures and/or number of MVs associated with the block at P side with the reference pictures of the block at Q side.
 - b. In one example, deblocking filter may be disabled even when two blocks are with different reference pictures.
 - c. In one example, deblocking filter may be disabled even when two blocks are with different number of MVs (e.g., one is uni-predicted and the other is bi-predicted).

- d. In one example, the value of BS may be set to 1 when motion vector differences for one or all reference picture lists between the blocks at P side and Q side is larger than or equal to a threshold Th.
 - i. Alternatively, furthermore, the value of BS may be set to 0 when motion vector differences for one or all reference picture lists between the blocks at P side and Q side is smaller than or equal to a threshold Th.
- e. In one example, the difference of the motion vectors of two blocks being larger than a threshold Th may be defined as $(Abs(MVP[0].x - MVQ[0].x) > Th \parallel Abs(MVP[0].y - MVQ[0].y) > Th \parallel Abs(MVP[1].x - MVQ[1].x) > Th \parallel Abs(MVP[1].y - MVQ[1].y) > Th)$
 - i. Alternatively, the difference of the motion vectors of two blocks being larger than a threshold Th may be defined as $(Abs(MVP[0].x - MVQ[0].x) > Th \&\& Abs(MVP[0].y - MVQ[0].y) > Th \&\& Abs(MVP[1].x - MVQ[1].x) > Th) \&\& Abs(MVP[1].y - MVQ[1].y) > Th)$
 - ii. Alternatively, in one example, the difference of the motion vectors of two blocks being larger than a threshold Th may be defined as $(Abs(MVP[0].x - MVQ[0].x) > Th \parallel Abs(MVP[0].y - MVQ[0].y) > Th) \&\& (Abs(MVP[1].x - MVQ[1].x) > Th \parallel Abs(MVP[1].y - MVQ[1].y) > Th)$
 - iii. Alternatively, in one example, the difference of the motion vectors of two blocks being larger than a threshold Th may be defined as $(Abs(MVP[0].x - MVQ[0].x) > Th \&\& Abs(MVP[0].y - MVQ[0].y) > Th) \parallel (Abs(MVP[1].x - MVQ[1].x) > Th \&\& Abs(MVP[1].y - MVQ[1].y) > Th)$
- f. In one example, a block which does not have a motion vector in a given list may be treated as having a zero-motion vector in that list.
- g. In the above examples, Th is an integer number (e.g. 4, 8 or 16).
- h. In the above examples, Th may depend on
 - i. Video contents (e.g. screen contents or natural contents)
 - ii. A message signaled in the DPS/SPS/VPS/PPS/APS/picture header/slice header/tile group header/ Largest coding unit (LCU)/Coding unit (CU)/LCU row/group of LCUs/TU/PU block/Video coding unit

- iii. Position of CU/PU/TU/block/Video coding unit
- iv. Coded modes of blocks containing the samples along the edges
- v. Transform matrices applied to the blocks containing the samples along the edges
- vi. Block dimension/Block shape of current block and/or its neighboring blocks
- vii. Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)
- viii. Coding tree structure (such as dual tree or single tree)
- ix. Slice/tile group type and/or picture type
- x. Color component (e.g. may be only applied on Cb or Cr)
- xi. Temporal layer ID
- xii. Profiles/Levels/Tiers of a standard
- xiii. Alternatively, Th may be signalled to the decoder.
- i. The above examples may be applied under certain conditions.
 - i. In one example, the condition is the *blkP* and *blkQ* are not coded with intra modes.
 - ii. In one example, the condition is the *blkP* and *blkQ* have zero coefficients on luma component.
 - iii. In one example, the condition is the *blkP* and *blkQ* are not coded with the CIIP mode.
 - iv. In one example, the condition is the *blkP* and *blkQ* are coded with a same prediction mode (e.g. IBC or Inter).

Regarding luma deblocking filtering process

- 21. The deblocking may use different QPs for TS coded blocks and non-TS coded blocks.
 - a. In one example, the QP for TS may be used on TS coded blocks while the QP for non-TS may be used on non-TS coded blocks.
- 22. The luma filtering process (e.g., the luma edge decision process) may depend on the quantization parameter applied to the scaling process of the luma block.
 - a. In one example, the QP used to derive beta and Tc may depend on the clipping range of transform skip, e.g. as indicated by QpPrimeTsMin.

23. It is proposed to use an identical gradient computation for large block boundaries and smaller block boundaries.
- a. In one example, the deblocking filter on/off decision described in section 2.1.4 may be also applied for large block boundary.
 - i. In one example, the threshold *beta* in the decision may be modified for large block boundary.
 1. In one example, *beta* may depend on quantization parameter.
 2. In one example, *beta* used for deblocking filter on/off decision for large block boundaries may be smaller than that for smaller block boundaries.
 - a. Alternatively, in one example, *beta* used for deblocking filter on/off decision for large block boundaries may be larger than that for smaller block boundaries.
 - b. Alternatively, in one example, *beta* used for deblocking filter on/off decision for large block boundaries may be equal to that for smaller block boundaries.
 3. In one example, *beta* is an integer number and may be based on
 - a. Video contents (e.g. screen contents or natural contents)
 - b. A message signaled in the DPS/SPS/VPS/PPS/APS/picture header/slice header/tile group header/ Largest coding unit (LCU)/Coding unit (CU)/LCU row/group of LCUs/TU/PU block/Video coding unit
 - c. Position of CU/PU/TU/block/Video coding unit
 - d. Coded modes of blocks containing the samples along the edges
 - e. Transform matrices applied to the blocks containing the samples along the edges
 - f. Block dimension of current block and/or its neighboring blocks
 - g. Block shape of current block and/or its neighboring blocks

- h. Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)
- i. Coding tree structure (such as dual tree or single tree)
- j. Slice/tile group type and/or picture type
- k. Color component (e.g. may be only applied on Cb or Cr)
- l. Temporal layer ID
- m. Profiles/Levels/Tiers of a standard
- n. Alternatively, *beta* may be signalled to the decoder.

Regarding scaling matrix (dequantization matrix)

- 24. The values for specific positions of quantization matrices may be set to constant.
 - a. In one example, the position may be the position of (x, y) wherein x and y are two integer variables (e.g., x=y=0), and (x, y) is the coordinate relative to a TU/TB/PU/PB/CU/CB.
 - i. In one example, the position may be the position of DC.
 - b. In one example, the constant value may be 16.
 - c. In one example, for those positions, signaling of the matrix values may not be utilized.
- 25. A constrain may be set that the average/weighted average of some positions of quantization matrices may be a constant.
 - a. In one example, deblocking process may depend on the constant value.
 - b. In one example, the constant value may be indicated in DPS/VPS/SPS/PPS/Slice/Picture/Tile/Brick headers.
- 26. One or multiple indications may be signaled in the picture header to inform the scaling matrix to be selected in the picture associated with the picture header.

Regarding Cross Component Adaptive Loop Filter (CCALF)

- 27. CCALF may be applied before some loop filtering process at the decoder
 - a. In one example, CCALF may be applied before deblocking process at the decoder.
 - b. In one example, CCALF may be applied before SAO at the decoder.
 - c. In one example, CCALF may be applied before ALF at the decoder.

- d. Alternatively, the order of different filters (e.g., CCALF, ALF, SAO, deblocking filter) may be NOT fixed.
 - i. In one example, the invoke of CCLAF may be before one filtering process for one video unit or after another one for another video unit.
 - ii. In one example, the video unit may be a CTU/CTB/slice/tile/brick/picture/sequence.
- e. Alternatively, indications of the order of different filters (e.g., CCALF, ALF, SAO, deblocking filter) may be signaled or derived on-the-fly.
 - i. Alternatively, indication of the invoking of CCALF may be signaled or derived on-the-fly.
- f. The explicit (e.g. signaling from the encoder to the decoder) or implicit (e.g. derived at both encoder and decoder) indications of how to control CCALF may be decoupled for different color components (such as Cb and Cr).
- g. Whether and/or how to apply CCALF may depend on color formats (such as RGB and YCbCr) and/or color sampling format (such as 4:2:0, 4:2:2 and 4:4:4), and/or color down-sampling positions or phases)

Regarding chroma QP offset lists

28. Signaling and/or selection of chroma QP offset lists may be dependent on the coded prediction modes/picture types/slice or tile or brick types.
- a. Chroma QP offset lists, e.g. `cb_qp_offset_list[ i ]`, `cr_qp_offset_list[ i ]`, and `joint_cbc_r_qp_offset_list[ i ]`, may be different for different coding modes.
 - b. In one example, whether and how to apply chroma QP offset lists may depend on whether the current block is coded in intra mode.
 - c. In one example, whether and how to apply chroma QP offset lists may depend on whether the current block is coded in inter mode.
 - d. In one example, whether and how to apply chroma QP offset lists may depend on whether the current block is coded in palette mode.
 - e. In one example, whether and how to apply chroma QP offset lists may depend on whether the current block is coded in IBC mode.

- f. In one example, whether and how to apply chroma QP offset lists may depend on whether the current block is coded in transform skip mode.
- g. In one example, whether and how to apply chroma QP offset lists may depend on whether the current block is coded in BDPCM mode.
- h. In one example, whether and how to apply chroma QP offset lists may depend on whether the current block is coded in transform_quant_skip or lossless mode.

Regarding chroma deblocking at CTU boundary

- 29. How to select the QPs (e.g., using corresponding luma or chroma dequantized QP) utilized in the deblocking filter process may be dependent on the position of samples relative to the CTU/CTB/VPDU boundaries.
- 30. How to select the QPs (e.g., using corresponding luma or chroma dequantized QP) utilized in the deblocking filter process may depend on color formats (such as RGB and YCbCr) and/or color sampling format (such as 4:2:0, 4:2:2 and 4:4:4), and/or color down-sampling positions or phases).
- 31. For edges at CTU boundary, the deblocking may be based on luma QP of the corresponding blocks.
 - a. In one example, for horizontal edges at CTU boundary, the deblocking may be based on luma QP of the corresponding blocks.
 - i. In one example, the deblocking may be based on luma QP of the corresponding blocks at P-side.
 - ii. In one example, the deblocking may be based on luma QP of the corresponding blocks at Q-side.
 - b. In one example, for vertical edges at CTU boundary, the deblocking may be based on luma QP of the corresponding blocks.
 - i. In one example, the deblocking may be based on luma QP of the corresponding blocks at P-side.
 - ii. In one example, the deblocking may be based on luma QP of the corresponding blocks at Q-side.
 - c. In one example, for edges at CTU boundary, the deblocking may be based on luma QP at P-side and chroma QP at Q-side.

- d. In one example, for edges at CTU boundary, the deblocking may be based on luma QP at Q-side and chroma QP at P-side.
 - e. In this bullet, “CTU boundary” may refer to a specific CTU boundary such as the upper CTU boundary or the lower CTU boundary.
32. For horizontal edges at CTU boundary, the deblocking may be based on a function of chroma QPs at P-side.
- a. In one example, the deblocking may be based on an averaging function of chroma QPs at P-side.
 - i. In one example, the function may be based on the average of the chroma QPs for each 8 luma samples.
 - ii. In one example, the function may be based on the average of the chroma QPs for each 16 luma samples.
 - iii. In one example, the function may be based on the average of the chroma QPs for each 32 luma samples.
 - iv. In one example, the function may be based on the average of the chroma QPs for each 64 luma samples.
 - v. In one example, the function may be based on the average of the chroma QPs for each CTU.
 - b. In one example, the deblocking may be based on a maximum function of chroma QPs at P-side.
 - i. In one example, the function may be based on the maximum of the chroma QPs for each 8 luma samples.
 - ii. In one example, the function may be based on the maximum of the chroma QPs for each 16 luma samples.
 - iii. In one example, the function may be based on the maximum of the chroma QPs for each 32 luma samples.
 - iv. In one example, the function may be based on the maximum of the chroma QPs for each 64 luma samples.
 - v. In one example, the function may be based on the maximum of the chroma QPs for each CTU.

- c. In one example, the deblocking may be based on a minimum function of chroma QPs at P-side.
 - i. In one example, the function may be based on the minimum of the chroma QPs for each 8 luma samples.
 - ii. In one example, the function may be based on the minimum of the chroma QPs for each 16 luma samples.
 - iii. In one example, the function may be based on the minimum of the chroma QPs for each 32 luma samples.
 - iv. In one example, the function may be based on the minimum of the chroma QPs for each 64 luma samples.
 - v. In one example, the function may be based on the minimum of the chroma QPs for each CTU.
- d. In one example, the deblocking may be based on a subsampling function of chroma QPs at P-side.
 - i. In one example, the function may be based on the chroma QPs of the k-th chroma sample for each 8 luma samples.
 - 1. In one example, the k-th sample may be the first sample.
 - 2. In one example, the k-th sample may be the last sample.
 - 3. In one example, the k-th sample may be the third sample.
 - 4. In one example, the k-th sample may be the fourth sample.
 - ii. In one example, the function may be based on the chroma QPs of the k-th chroma sample for each 16 luma samples.
 - 1. In one example, the k-th sample may be the first sample.
 - 2. In one example, the k-th sample may be the last sample.
 - 3. In one example, the k-th sample may be the 7-th sample.
 - 4. In one example, the k-th sample may be the 8-th sample.
 - iii. In one example, the function may be based on the chroma QPs of the k-th chroma sample for each 32 luma samples.
 - 1. In one example, the k-th sample may be the first sample.
 - 2. In one example, the k-th sample may be the last sample.

3. In one example, the k-th sample may be the 15-th sample.
 4. In one example, the k-th sample may be the 16-th sample.
 - iv. In one example, the function may be based on the chroma QPs of the k-th chroma sample for each 64 luma samples.
 1. In one example, the k-th sample may be the first sample.
 2. In one example, the k-th sample may be the last sample.
 3. In one example, the k-th sample may be the 31-th sample.
 4. In one example, the k-th sample may be the 32-th sample.
 - v. In one example, the function may be based on the chroma QPs of the k-th chroma sample for each CTU.
 - e. Alternatively, the above items may be applied to chroma QPs at Q-side for deblocking process.
33. It may be constrained that quantization group for chroma component must be larger than a certain size.
 - a. In one example, it may be constrained that the width of quantization group for chroma component must be larger than a certain value, K.
 - i. In one example, K is equal to 4.
34. It may be constrained that quantization group for luma component must be larger than a certain size.
 - a. In one example, it may be constrained that the width of quantization group for luma component must be larger than a certain value, K.
 - i. In one example, K is equal to 8.
35. It may be constrained that QP for chroma component may be the same for a chroma row segment with length $4*m$ starting from $(4*m*x, 2y)$ relative to top-left of the picture, where x and y are non-negative integers; and m is a positive integer.
 - a. In one example, m may be equal to 1.
 - b. In one example, the width of a quantization group for chroma component must be no smaller than $4*m$.

36. It may be constrained that QP for chroma component may be the same for a chroma column segment with length $4*n$ starting from $(2*x, 4*n*y)$ relative to top-left of the picture, where x and y are non-negative integers; and m is a positive integer.
- In one example, n may be equal to 1.
 - In one example, the height of a quantization group for chroma component must be no smaller than $4*n$.

Regarding chroma deblocking filtering process

37. A first syntax element controlling the usage of coding tool X may be signalled in a first video unit (such as picture header), depending on a second syntax element signalled in a second video unit (such as SPS or PPS, or VPS).
- In one example, the first syntax element is signalled only if the second syntax element indicates that the coding tool X is enabled.
 - In one example, X is Bi-Direction Optical Flow (BDOF).
 - In one example, X is Prediction Refinement Optical Flow (PROF).
 - In one example, X is Decoder-side Motion Vector Refinement (DMVR).
 - In one example, the signalling of the usage of a coding tool X may be under the condition check of slice types (e.g., P or B slices; non-I slices).

Regarding chroma deblocking filtering process

38. Deblocking filter decision processes for two chroma blocks may be unified to be only invoked once and the decision is applied to two chroma blocks.
- In one example, the decision for whether to perform deblocking filter may be same for Cb and Cr components.
 - In one example, if the deblocking filter is determined to be applied, the decision for whether to perform stronger deblocking filter may be same for Cb and Cr components.
 - In one example, the deblocking condition and strong filter on/off condition, as described in section 2.2.7, may be only checked once. However, it may be modified to check the information of both chroma components.

- i. In one example, the average of gradients of Cb and Cr components may be used in the above decisions for both Cb and Cr components.
- ii. In one example, the chroma stronger filters may be performed only when the strong filter condition is satisfied for both Cb and Cr components.
 - 1. Alternatively, in one example, the chroma weak filters may be performed only when the strong filter condition is not satisfied at least one chroma component

On ACT

- 39. Whether deblocking QP is equal to dequantization QP may depend on whether ACT is applied.
 - a. In one example, when ACT is applied to a block, deblocking QP values may depend on QP values before ACT QP adjustment.
 - b. In one example, when ACT is not applied to a block, deblocking QP values may always equal to dequantization QP values.
 - c. In one example, when both ACT and TS are not applied to a block, deblocking QP values may always equal to dequantization QP values.
- 40. ACT and BDPCM may be applied exclusively at block level.
 - a. In one example, when ACT is applied on a block, luma BDPCM shall not be applied on that block.
 - b. In one example, when ACT is applied on a block, chroma BDPCM shall not be applied on that block.
 - c. In one example, when ACT is applied on a block, both luma and chroma BDPCM shall not be applied on that block.
 - d. In one example, when luma and/or chroma BDPCM is applied on a block, ACT shall not be applied on that block.
- 41. Whether to enable BDPCM mode may be inferred based on the usage of ACT (e.g. cu_act_enabled_flag).
 - a. In one example, the inferred value of chroma BDPCM mode may be defined as (cu_act_enabled_flag && intra_bdpcm_luma_flag && sps_bdpcm_chroma_enabled_flag? true: false).

- b. In one example, if `sps_bdpcm_chroma_enabled_flag` is false, the `intra_bdpcm_luma_flag` may be inferred to false when it is not signalled and `cu_act_enabled_flag` is true.
- c. In one example, the `cu_act_enabled_flag` may be inferred to false when `intra_bdpcm_luma_flag` is true and `sps_bdpcm_chroma_enabled_flag` is false.
- d. In one example, the `intra_bdpcm_luma_flag` may be inferred to false when `cu_act_enabled_flag` is true and `sps_bdpcm_chroma_enabled_flag` is false.

CC-ALF related

42. ChromaArrayType in the specification.

- a. In one example, whether cross component Cb filter is enabled for all slices associated with the picture header and may be applied to Cb colour component in the slices (e.g., **`pic_cross_component_alf_cb_enabled_flag`**) may be conditionally signalled, such as being conditioned on “if(ChromaArrayType != 0)” or the color format is not 4:0:0 (e.g., `chroma_format_idc != 0`).
- b. In one example, whether cross component Cr filter is enabled for all slices associated with the picture header and may be applied to Cr colour component in the slices (e.g., **`pic_cross_component_alf_cr_enabled_flag`**) may be conditionally signalled, such as being conditioned on “if(ChromaArrayType != 0)” or the color format is not 4:0:0 (e.g., `chroma_format_idc != 0`).

43. ALF and CC-ALF may be controlled separately.

- a. In one example, whether CC-ALF is enabled for a video processing unit may be signalled with a first syntax element.
 - i. In one example, it may be signalled in sequence/video/picture level (e.g., in SPS) which is independent from the ALF enabling flag (e.g., **`sps_alf_enabled_flag`**).
 - 1. Alternatively, it may be signalled under the conditional check of ALF is enabled.
 - ii. Alternatively, furthermore, the first syntax element may be signalled under the condition check of the color format and/or separate plane coding and/or ChromaArrayType in the specification.

- b. Alternatively, furthermore, a second syntax element may be further signalled in picture header/PPS/slice header, e.g., `pic_ccalf_enabled_present_flag`, to specify whether CC-ALF related syntax elements (e.g., **`pic_cross_component_alf_cb_enabled_flag`**, `pic_cross_component_alf_cb_aps_id`, `pic_cross_component_cb_filters_signalled_minus1`, `pic_cross_component_alf_cr_enabled_flag`, `pic_cross_component_alf_cr_aps_id`, and `pic_cross_component_cr_filters_signalled_minus1`) are present in the picture header.
 - i. Alternatively, the second syntax element may be signaled when the ALF enabling flag (e.g., `sps_alf_enabled_flag`) is true.
- c. In one example, syntax elements in PPS or PH or slice header related to CC-ALF are signalled only if both the conditions below are true:
 - i. `ChromaArrayType != 0` or the color format is not 4:0:0.
 - ii. It is signalled in a higher level video unit (e.g. SPS) that CC-ALF is enabled.
- d. In one example, syntax elements in PPS or PH or slice header related to CC-ALF are signalled when one of the conditions below is true:
 - i. `ChromaArrayType != 0` or the color format is not 4:0:0.
 - ii. It is signalled in a higher level video unit (e.g. SPS) that CC-ALF is enabled.

Regarding high level syntax

44. The control flag of chroma BDPCM mode (e.g. `sps_bdpcm_chroma_enabled_flag`), palette mode (e.g. `sps_palette_enabled_flag`) and/or ACT mode (e.g. `sps_act_enabled_flag`) may be signalled at least being based on the value of chroma array type (e.g. `ChromaArrayType`).
- a. In one example, the `sps_bdpcm_chroma_enabled_flag` may be signalled only when `sps_bdpcm_enabled_flag` is true && `ChromaArrayType` is equal to 3.
 - b. In one example, the `sps_palette_enabled_flag` may be signalled only when `ChromaArrayType` is equal to 3.
 - c. In one example, the `sps_act_enabled_flag` may be signalled only when `ChromaArrayType` is equal to 3.
 - d. In one example, the `sps_bdpcm_chroma_enabled_flag` may NOT be signalled when `ChromaArrayType` is unequal to 3.

- e. In one example, the `sps_palette_enabled_flag` may NOT be signalled when `ChromaArrayType` is unequal to 3.
- f. In one example, the `sps_act_enabled_flag` may NOT be signalled when `ChromaArrayType` is unequal to 3.
- g. In one example, a conformace bitstream shall satisfy that the `sps_bdpcm_chroma_enabled_flag` shall be set equal to 0 when `ChromaArrayType` is not equal to 3.
- h. In one example, a conformace bitstream shall satisfy that the `sps_palette_enabled_flag` shall be set equal to 0 when `ChromaArrayType` is not equal to 3.
- i. In one example, a conformace bitstream shall satisfy that the `sps_act_enabled_flag` shall be set equal to 0 when `ChromaArrayType` is not equal to 3.

General implementation concepts

45. The above proposed methods may be applied under certain conditions.
- a. In one example, the condition is the colour format is 4:2:0 and/or 4:2:2.
 - i. Alternatively, furthermore, for 4:4:4 colour format, how to apply deblocking filter to the two colour chroma components may follow the current design.
 - b. In one example, indication of usage of the above methods may be signalled in sequence/picture/slice/tile/brick/a video region-level, such as SPS/PPS/picture header/slice header.
 - c. In one example, the usage of above methods may depend on
 - ii. Video contents (e.g. screen contents or natural contents)
 - iii. A message signaled in the DPS/SPS/VPS/PPS/APS/picture header/slice header/tile group header/ Largest coding unit (LCU)/Coding unit (CU)/LCU row/group of LCUs/TU/PU block/Video coding unit
 - iv. Position of CU/PU/TU/block/Video coding unit
 - a. In one example, for filtering samples along the CTU/CTB boundaries (e.g., the first K (e.g., K=4/8) to the top/left/right/bottom boundaries), the existing design may be applied. While for other samples, the proposed method (e.g., bullets 3/4) may be applied instead.

- v. Coded modes of blocks containing the samples along the edges
- vi. Transform matrices applied to the blocks containing the samples along the edges
- vii. Block dimension of current block and/or its neighboring blocks
- viii. Block shape of current block and/or its neighboring blocks
- ix. Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)
- x. Coding tree structure (such as dual tree or single tree)
- xi. Slice/tile group type and/or picture type
- xii. Color component (e.g. may be only applied on Cb or Cr)
- xiii. Temporal layer ID
- xiv. Profiles/Levels/Tiers of a standard
- xv. Alternatively, m and/or n may be signalled to the decoder.

5. Additional Embodiments

[0106] The embodiments are based on JVET-O2001-vE. The newly added texts are shown in underlined bold italicized font. The deleted texts are marked by underlined bold text.

5.1. Embodiment #1 on chroma QP in deblocking

8.8.3.6 Edge filtering process for one direction

...

- Otherwise ($cIdx$ is not equal to 0), the filtering process for edges in the chroma coding block of current coding unit specified by $cIdx$ consists of the following ordered steps:
 1. The variable $cQpPicOffset$ is derived as follows:

$$cQpPicOffset = cIdx == 1 ? pps_cb_qp_offset : pps_cr_qp_offset(8-1065)$$

8.8.3.6.3 Decision process for chroma block edges

...

The variables Qp_Q and Qp_P are set equal to the Qp_Y values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable Q_{pC} is derived as follows:

$$q_{Pi} = \text{Clip3}(0, 63, ((Q_{pQ} + Q_{pP} + 1) \gg 1) + cQpPicOffset) \quad (8-1132)$$

$$Q_{pC} = \text{ChromaQpTable}[cIdx - 1][q_{Pi}] \quad (8-1133)$$

$$q_{Pi} = (Q_{pQ} + Q_{pP} + 1) \gg 1 \quad (8-1132)$$

$$Q_{pC} = \text{ChromaQpTable}[cIdx - 1][q_{Pi}] + cQpPicOffset \quad (8-1133)$$

NOTE – The variable $cQpPicOffset$ provides an adjustment for the value of $pps_cb_qp_offset$ or $pps_cr_qp_offset$, according to whether the filtered chroma component is the Cb or Cr component. However, to avoid the need to vary the amount of the adjustment within the picture, the filtering process does not include an adjustment for the value of $slice_cb_qp_offset$ or $slice_cr_qp_offset$ nor (when $cu_chroma_qp_offset_enabled_flag$ is equal to 1) for the value of $CuQpOffset_{Cb}$, $CuQpOffset_{Cr}$, or $CuQpOffset_{CbCr}$.

The value of the variable β' is determined as specified in Table 8-18 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 63, Q_{pC} + (slice_beta_offset_div2 \ll 1)) \quad (8-1134)$$

where $slice_beta_offset_div2$ is the value of the syntax element $slice_beta_offset_div2$ for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (1 \ll (BitDepth_C - 8)) \quad (8-1135)$$

The value of the variable tc' is determined as specified in Table 8-18 based on the chroma quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (slice_tc_offset_div2 \ll 1)) \quad (8-1136)$$

where $slice_tc_offset_div2$ is the value of the syntax element $slice_tc_offset_div2$ for the slice that contains sample $q_{0,0}$.

The variable t_c is derived as follows:

$$t_c = (\text{BitDepth}_c < 10) ? (t_c' + 2) >> (10 - \text{BitDepth}_c) : t_c' * (1 << (\text{BitDepth}_c - 8))$$

(8-1137)

5.2. Embodiment #2 on boundary strength derivation

8.8.3.5 Derivation process of boundary filtering strength

Inputs to this process are:

- a picture sample array `recPicture`,
- a location (`xCb`, `yCb`) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a variable `nCbW` specifying the width of the current coding block,
- a variable `nCbH` specifying the height of the current coding block,
- a variable `edgeType` specifying whether a vertical (`EDGE_VER`) or a horizontal (`EDGE_HOR`) edge is filtered,
- a variable `cIdx` specifying the colour component of the current coding block,
- a two-dimensional (`nCbW`) $\times$ (`nCbH`) array `edgeFlags`.

Output of this process is a two-dimensional (`nCbW`) $\times$ (`nCbH`) array `bS` specifying the boundary filtering strength.

...

For x_{D_i} with $i = 0..xN$ and y_{D_j} with $j = 0..yN$, the following applies:

- If `edgeFlags[  $x_{D_i}$  ][  $y_{D_j}$  ]` is equal to 0, the variable `bS[  $x_{D_i}$  ][  $y_{D_j}$  ]` is set equal to 0.
- Otherwise, the following applies:

...

- The variable `bS[  $x_{D_i}$  ][  $y_{D_j}$  ]` is derived as follows:

- If $cIdx$ is equal to 0 and both samples p_0 and q_0 are in a coding block with $intra_bdpcm_flag$ equal to 1, $bS[xD_i][yD_j]$ is set equal to 0.
- Otherwise, if the sample p_0 or q_0 is in the coding block of a coding unit coded with intra prediction mode, $bS[xD_i][yD_j]$ is set equal to 2.
- Otherwise, if the block edge is also a transform block edge and the sample p_0 or q_0 is in a coding block with $ciip_flag$ equal to 1, $bS[xD_i][yD_j]$ is set equal to 2.
- Otherwise, if the block edge is also a transform block edge and the sample p_0 or q_0 is in a transform block which contains one or more non-zero transform coefficient levels, $bS[xD_i][yD_j]$ is set equal to 1.
- Otherwise, if the block edge is also a transform block edge, $cIdx$ is greater than 0, and the sample p_0 or q_0 is in a transform unit with $tu_joint_cbcr_residual_flag$ equal to 1, $bS[xD_i][yD_j]$ is set equal to 1.
- Otherwise, if the prediction mode of the coding subblock containing the sample p_0 is different from the prediction mode of the coding subblock containing the sample q_0 (i.e. one of the coding subblock is coded in IBC prediction mode and the other is coded in inter prediction mode), $bS[xD_i][yD_j]$ is set equal to 1.
- Otherwise, if $cIdx$ is equal to 0 and one or more of the following conditions are true, $bS[xD_i][yD_j]$ is set equal to 1:
 - **The absolute difference between the horizontal or vertical component of list 0 motion vectors used in the prediction of the two coding subblocks is greater than or equal to 8 in 1/16 luma samples, or the absolute difference between the horizontal or vertical component of the list 1 motion vectors used in the prediction of the two coding subblocks is greater than or equal to 8 in units of 1/16 luma samples.**
 - **The coding subblock containing the sample p_0 and the coding subblock containing the sample q_0 are both coded in IBC prediction mode, and the absolute difference between the horizontal or vertical component of the**

block vectors used in the prediction of the two coding subblocks is greater than or equal to 8 in units of 1/16 luma samples.

- For the prediction of the coding subblock containing the sample p_0 different reference pictures or a different number of motion vectors are used than for the prediction of the coding subblock containing the sample q_0 .

NOTE 1 – The determination of whether the reference pictures used for the two coding subblocks are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether the index position within a reference picture list is different.

NOTE 2 – The number of motion vectors that are used for the prediction of a coding subblock with top-left sample covering (x_{Sb}, y_{Sb}) , is equal to $\text{PredFlagL0}[x_{Sb}][y_{Sb}] + \text{PredFlagL1}[x_{Sb}][y_{Sb}]$.

- One motion vector is used to predict the coding subblock containing the sample p_0 and one motion vector is used to predict the coding subblock containing the sample q_0 , and the absolute difference between the horizontal or vertical component of the motion vectors used is greater than or equal to 8 in units of 1/16 luma samples.
- Two motion vectors and two different reference pictures are used to predict the coding subblock containing the sample p_0 , two motion vectors for the same two reference pictures are used to predict the coding subblock containing the sample q_0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two coding subblocks for the same reference picture is greater than or equal to 8 in units of 1/16 luma samples.

- Two motion vectors for the same reference picture are used to predict the coding subblock containing the sample p_0 , two motion vectors for the same reference picture are used to predict the coding subblock containing the sample q_0 and both of the following conditions are true:
 - The absolute difference between the horizontal or vertical component of list 0 motion vectors used in the prediction of the two coding subblocks is greater than or equal to 8 in 1/16 luma samples, or the absolute difference between the horizontal or vertical component of the list 1 motion vectors used in the prediction of the two coding subblocks is greater than or equal to 8 in units of 1/16 luma samples.
 - The absolute difference between the horizontal or vertical component of list 0 motion vector used in the prediction of the coding subblock containing the sample p_0 and the list 1 motion vector used in the prediction of the coding subblock containing the sample q_0 is greater than or equal to 8 in units of 1/16 luma samples, or the absolute difference between the horizontal or vertical component of the list 1 motion vector used in the prediction of the coding subblock containing the sample p_0 and list 0 motion vector used in the prediction of the coding subblock containing the sample q_0 is greater than or equal to 8 in units of 1/16 luma samples.
- Otherwise, the variable $bS[xD_i][yD_j]$ is set equal to 0.

5.3. Embodiment #3 on boundary strength derivation

8.8.3.6 Derivation process of boundary filtering strength

Inputs to this process are:

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

- a variable nCbW specifying the width of the current coding block,
- a variable nCbH specifying the height of the current coding block,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a variable cIdx specifying the colour component of the current coding block,
- a two-dimensional (nCbW)x(nCbH) array edgeFlags.

Output of this process is a two-dimensional (nCbW)x(nCbH) array bS specifying the boundary filtering strength.

...

For xD_i with $i = 0..xN$ and yD_j with $j = 0..yN$, the following applies:

- If edgeFlags[xD_i][yD_j] is equal to 0, the variable bS[xD_i][yD_j] is set equal to 0.
- Otherwise, the following applies:

...

- The variable bS[xD_i][yD_j] is derived as follows:
 - If cIdx is equal to 0 and both samples p_0 and q_0 are in a coding block with intra_bdpcm_flag equal to 1, bS[xD_i][yD_j] is set equal to 0.
 - Otherwise, if the sample p_0 or q_0 is in the coding block of a coding unit coded with intra prediction mode, bS[xD_i][yD_j] is set equal to 2.
 - Otherwise, if the block edge is also a transform block edge and the sample p_0 or q_0 is in a coding block with ciip_flag equal to 1, bS[xD_i][yD_j] is set equal to 2.
 - Otherwise, if the block edge is also a transform block edge and the sample p_0 or q_0 is in a transform block which contains one or more non-zero transform coefficient levels, bS[xD_i][yD_j] is set equal to 1.

- **Otherwise, if the block edge is also a transform block edge, cIdx is greater than 0, and the sample p_0 or q_0 is in a transform unit with tu joint cbr residual flag equal to 1, $bS[xD_i][yD_j]$ is set equal to 1.**
- Otherwise, if the prediction mode of the coding subblock containing the sample p_0 is different from the prediction mode of the coding subblock containing the sample q_0 (i.e. one of the coding subblock is coded in IBC prediction mode and the other is coded in inter prediction mode), $bS[xD_i][yD_j]$ is set equal to 1.
- Otherwise, if cIdx is equal to 0 and one or more of the following conditions are true, $bS[xD_i][yD_j]$ is set equal to 1:
 - The coding subblock containing the sample p_0 and the coding subblock containing the sample q_0 are both coded in IBC prediction mode, and the absolute difference between the horizontal or vertical component of the block vectors used in the prediction of the two coding subblocks is greater than or equal to 8 in units of 1/16 luma samples.
 - For the prediction of the coding subblock containing the sample p_0 different reference pictures or a different number of motion vectors are used than for the prediction of the coding subblock containing the sample q_0 .

NOTE 1 – The determination of whether the reference pictures used for the two coding subblocks are the same or different is based only on which pictures are referenced, without regard to whether a prediction is formed using an index into reference picture list 0 or an index into reference picture list 1, and also without regard to whether the index position within a reference picture list is different.

NOTE 2 – The number of motion vectors that are used for the prediction of a coding subblock with top-left sample covering (xSb, ySb), is equal to $PredFlagL0[xSb][ySb] + PredFlagL1[xSb][ySb]$.

- One motion vector is used to predict the coding subblock containing the sample p_0 and one motion vector is used to predict the coding subblock

containing the sample q_0 , and the absolute difference between the horizontal or vertical component of the motion vectors used is greater than or equal to 8 in units of 1/16 luma samples.

- Two motion vectors and two different reference pictures are used to predict the coding subblock containing the sample p_0 , two motion vectors for the same two reference pictures are used to predict the coding subblock containing the sample q_0 and the absolute difference between the horizontal or vertical component of the two motion vectors used in the prediction of the two coding subblocks for the same reference picture is greater than or equal to 8 in units of 1/16 luma samples.
- Two motion vectors for the same reference picture are used to predict the coding subblock containing the sample p_0 , two motion vectors for the same reference picture are used to predict the coding subblock containing the sample q_0 and both of the following conditions are true:
 - The absolute difference between the horizontal or vertical component of list 0 motion vectors used in the prediction of the two coding subblocks is greater than or equal to 8 in 1/16 luma samples, or the absolute difference between the horizontal or vertical component of the list 1 motion vectors used in the prediction of the two coding subblocks is greater than or equal to 8 in units of 1/16 luma samples.
 - The absolute difference between the horizontal or vertical component of list 0 motion vector used in the prediction of the coding subblock containing the sample p_0 and the list 1 motion vector used in the prediction of the coding subblock containing the sample q_0 is greater than or equal to 8 in units of 1/16 luma samples, or the absolute difference between the horizontal or vertical component of the list 1 motion vector used in the prediction of the coding subblock containing the sample p_0 and list 0 motion vector used in the prediction of the coding subblock containing the sample q_0 is greater than or equal to 8 in units of 1/16 luma samples.

- Otherwise, the variable $bS[xD_i][yD_j]$ is set equal to 0.

5.4. Embodiment #4 on luma deblocking filtering process

8.8.3.6.1 Decision process for luma block edges

Inputs to this process are:

- a picture sample array $recPicture$,
- a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a location (xBl, yBl) specifying the top-left sample of the current block relative to the top-left sample of the current coding block,
- a variable $edgeType$ specifying whether a vertical ($EDGE_VER$) or a horizontal ($EDGE_HOR$) edge is filtered,
- a variable bS specifying the boundary filtering strength,
- a variable $maxFilterLengthP$ specifying the max filter length,
- a variable $maxFilterLengthQ$ specifying the max filter length.

Outputs of this process are:

- the variables dE , dEp and dEq containing decisions,
- the modified filter length variables $maxFilterLengthP$ and $maxFilterLengthQ$,
- the variable tC .

...

The following ordered steps apply:

...

1. When $sidePisLargeBlk$ or $sideQisLargeBlk$ is greater than 0, the following applies:
 - a. The variables $dp0L$, $dp3L$ are derived and $maxFilterLengthP$ is modified as follows:
 - **If $sidePisLargeBlk$ is equal to 1, the following applies:**

$$\underline{dp0L = (dp0 + Abs(p_{5,0} - 2 * p_{4,0} + p_{3,0}) + 1) >> 1} \quad (8-1087)$$

$$\underline{dp3L = (dp3 + Abs(p_{5,3} - 2 * p_{4,3} + p_{3,3}) + 1) >> 1} \quad (8-1088)$$

– Otherwise, the following applies:

$$dp0L = dp0 \quad (8-1089)$$

$$dp3L = dp3 \quad (8-1090)$$

$$\underline{maxFilterLengthP = 3} \quad (8-1091)$$

$$\underline{maxFilterLengthP = sidePisLargeBlk ? maxFilterLengthP : 3}$$

b. The variables dq0L and dq3L are derived as follows:

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

$$\underline{dq0L = (dq0 + Abs(q_{5,0} - 2 * q_{4,0} + q_{3,0}) + 1) >> 1} \quad (8-1092)$$

$$\underline{dq3L = (dq3 + Abs(q_{5,3} - 2 * q_{4,3} + q_{3,3}) + 1) >> 1} \quad (8-1093)$$

– Otherwise, the following applies:

$$dq0L = dq0 \quad (8-1094)$$

$$dq3L = dq3 \quad (8-1095)$$

$$\underline{maxFilterLengthQ = sidePisLargeBlk ? maxFilterLengthQ : 3}$$

...

2. The variables dE, dEp and dEq are derived as follows:

...

5.5. Embodiment #5 on chroma deblocking filtering process

8.8.3.6.3 Decision process for chroma block edges

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

Inputs to this process are:

- a chroma picture sample array recPicture,
- a chroma location (xCb, yCb) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
- a chroma location (xBl, yBl) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a variable cIdx specifying the colour component index,
- a variable cQpPicOffset specifying the picture-level chroma quantization parameter offset,
- a variable bS specifying the boundary filtering strength,
- a variable maxFilterLengthCbCr.

Outputs of this process are

- the modified variable maxFilterLengthCbCr,
- the variable tc.

The variable maxK is derived as follows:

- If edgeType is equal to EDGE_VER, the following applies:

$$\text{maxK} = (\text{SubHeightC} = 1) ? 3 : 1 \quad (8-1124)$$

- Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$\text{maxK} = (\text{SubWidthC} = 1) ? 3 : 1 \quad (8-1125)$$

The values p_i and q_i with $i = 0.. \text{maxFilterLengthCbCr}$ and $k = 0.. \text{maxK}$ are derived as follows:

- If edgeType is equal to EDGE_VER, the following applies::

$$q_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + i][\text{yCb} + \text{yBl} + k] \quad (8-1126)$$

$$p_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} - i - 1][\text{yCb} + \text{yBl} + k] \quad (8-1127)$$

$$\text{subSampleC} = \text{SubHeightC} \quad (8-1128)$$

– Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$q_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} + i] \quad (8-1129)$$

$$p_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} - i - 1] \quad (8-1130)$$

$$\text{subSampleC} = \text{SubWidthC} \quad (8-1131)$$

When ChromaArrayType is not equal to 0 and treeType is equal to SINGLE TREE or DUAL TREE CHROMA, the following applies:

- When treeType is equal to DUAL TREE CHROMA, the variable Op_Y is set equal to the luma quantization parameter Op_Y of the luma coding unit that covers the luma location $(\text{xCb} + \text{cbWidth} / 2, \text{yCb} + \text{cbHeight} / 2)$.
- The variables qP_{Cb} , qP_{Cr} and qP_{CbCr} are derived as follows:

$$qPi_{Chroma} = \text{Clip3}(-OpBdOffset_C, 63, Op_Y) \quad (8-935)$$

$$qPi_{Cb} = \text{ChromaOpTable}[0][qPi_{Chroma}] \quad (8-936)$$

$$qPi_{Cr} = \text{ChromaOpTable}[1][qPi_{Chroma}] \quad (8-937)$$

$$qPi_{CbCr} = \text{ChromaOpTable}[2][qPi_{Chroma}] \quad (8-938)$$

- The chroma quantization parameters for the Cb and Cr components, Op'_{Cb} and Op'_{Cr} , and joint Cb-Cr coding Op'_{CbCr} are derived as follows:

$$Op'_{Cb} = \text{Clip3}(-OpBdOffset_C, 63, qP_{Cb} + \text{pps_cb_qp_offset} + \text{slice_cb_qp_offset} + \text{CuOpOffset}_{Cb}) \quad (8-939)$$

$$\underline{Op'_{Cr} = Clip3(-OpBdOffset_C, 63, qP_{Cr} + pps_cr_qp_offset + slice_cr_qp_offset + CuOpOffset_{Cr})} \quad (8-940)$$

$$\underline{Op'_{CbCr} = Clip3(-OpBdOffset_C, 63, qP_{CbCr} + pps_cbcr_qp_offset + slice_cbcr_qp_offset + CuOpOffset_{CbCr})} \quad (8-941)$$

The variables Op_O and Op_P are set equal to the corresponding Op'_{Cb} or Op'_{Cr} or Op'_{CbCr} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable Op_C is derived as follows:

$$\underline{Op_C = (Op_O + Op_P + 1) \gg 1} \quad (8-1133)$$

The value of the variable β' is determined as specified in Table 8-18 based on the quantization parameter Q derived as follows:

$$Q = Clip3(0, 63, Op_C + (slice_beta_offset_div2 \ll 1)) \quad (8-1134)$$

where $slice_beta_offset_div2$ is the value of the syntax element $slice_beta_offset_div2$ for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (1 \ll (BitDepth_C - 8)) \quad (8-1135)$$

The value of the variable tc' is determined as specified in Table 8-18 based on the chroma quantization parameter Q derived as follows:

$$Q = Clip3(0, 65, Op_C + 2 * (bS - 1) + (slice_tc_offset_div2 \ll 1)) \quad (8-1136)$$

where $slice_tc_offset_div2$ is the value of the syntax element $slice_tc_offset_div2$ for the slice that contains sample $q_{0,0}$.

The variable t_c is derived as follows:

$$t_c = (\text{BitDepth}_c < 10) ? (t_c' + 2) \gg (10 - \text{BitDepth}_c) : t_c' * (1 \ll (\text{BitDepth}_c - 8))$$

(8-1137)

When $\text{maxFilterLengthCbCr}$ is equal to 1 and bS is not equal to 2, $\text{maxFilterLengthCbCr}$ is set equal to 0.

5.6. Embodiment #6 on chroma QP in deblocking

8.8.3.6.3 Decision process for chroma block edges

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

Inputs to this process are:

- a chroma picture sample array recPicture ,
- a chroma location (x_{Cb} , y_{Cb}) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
- a chroma location (x_{Bl} , y_{Bl}) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a variable $cIdx$ specifying the colour component index,
- a variable $cQpPicOffset$ specifying the picture-level chroma quantization parameter offset,
- a variable bS specifying the boundary filtering strength,
- a variable $\text{maxFilterLengthCbCr}$.

Outputs of this process are

- the modified variable $\text{maxFilterLengthCbCr}$,
- the variable t_c .

The variable maxK is derived as follows:

- If edgeType is equal to EDGE_VER, the following applies:

$$\text{maxK} = (\text{SubHeightC} == 1) ? 3 : 1 \quad (8-1124)$$

- Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$\text{maxK} = (\text{SubWidthC} == 1) ? 3 : 1 \quad (8-1125)$$

The values p_i and q_i with $i = 0.. \text{maxFilterLengthCbCr}$ and $k = 0.. \text{maxK}$ are derived as follows:

- If edgeType is equal to EDGE_VER, the following applies::

$$q_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + i][\text{yCb} + \text{yBl} + k] \quad (8-1126)$$

$$p_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} - i - 1][\text{yCb} + \text{yBl} + k] \quad (8-1127)$$

$$\text{subSampleC} = \text{SubHeightC} \quad (8-1128)$$

- Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$q_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} + i] \quad (8-1129)$$

$$p_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} - i - 1] \quad (8-1130)$$

$$\text{subSampleC} = \text{SubWidthC} \quad (8-1131)$$

The variables Q_{pQ} and Q_{pP} are set equal to the Q_{pY} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variables $jccr_flag_P$ and $jccr_flag_O$ are set equal to the tu joint cbr residual flag values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable Q_{pC} is derived as follows:

$$\mathbf{qPi = Clip3(0, 63, ((Q_{pO} + Q_{pP} + 1) >> 1) + cQpPicOffset)} \quad (8-1132)$$

$$\underline{qPi = Clip3(0, 63, ((QpO + (jccr_flag_P ? pps_joint_cbr_qp_offset : cQpPicOffset) + QpP + (jccr_flag_O ? pps_joint_cbr_qp_offset : cQpPicOffset) + 1) >> 1))}$$

$$QpC = ChromaQpTable[cIdx - 1][qPi] \quad (8-1133)$$

NOTE – The variable cQpPicOffset provides an adjustment for the value of pps_cb_qp_offset or pps_cr_qp_offset, according to whether the filtered chroma component is the Cb or Cr component. However, to avoid the need to vary the amount of the adjustment within the picture, the filtering process does not include an adjustment for the value of slice_cb_qp_offset or slice_cr_qp_offset nor (when cu_chroma_qp_offset_enabled_flag is equal to 1) for the value of CuQpOffsetCb, CuQpOffsetCr, or CuQpOffsetCbCr.

...

5.7. Embodiment #7 on chroma QP in deblocking

8.8.3.6.3 Decision process for chroma block edges

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

Inputs to this process are:

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

Outputs of this process are

- the modified variable maxFilterLengthCbCr,
- the variable tc.

The variable maxK is derived as follows:

- If edgeType is equal to EDGE_VER, the following applies:

$$\max K = (\text{SubHeightC} == 1) ? 3 : 1 \quad (8-1124)$$

- Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$\text{maxK} = (\text{SubWidthC} = 1) ? 3 : 1 \quad (8-1125)$$

The values p_i and q_i with $i = 0.. \text{maxFilterLengthCbCr}$ and $k = 0.. \text{maxK}$ are derived as follows:

- If edgeType is equal to EDGE_VER, the following applies:

$$q_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + i][\text{yCb} + \text{yBl} + k] \quad (8-1126)$$

$$p_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} - i - 1][\text{yCb} + \text{yBl} + k] \quad (8-1127)$$

$$\text{subSampleC} = \text{SubHeightC} \quad (8-1128)$$

- Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$q_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} + i] \quad (8-1129)$$

$$p_{i,k} = \text{recPicture}[\text{xCb} + \text{xBl} + k][\text{yCb} + \text{yBl} - i - 1] \quad (8-1130)$$

$$\text{subSampleC} = \text{SubWidthC} \quad (8-1131)$$

The variables Q_{pQ} and Q_{pP} are set equal to the Q_{pY} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variables Q_{pQ} is set equal to the luma quantization parameter Q_{pY} of the luma coding unit that covers the luma location $(\text{xCb} + \text{cbWidth} / 2, \text{yCb} + \text{cbHeight} / 2)$ wherein cbWidth specifies the width of the current chroma coding block in luma samples, and cbHeight specifies the height of the current chroma coding block in luma samples.

The variables Q_{pP} is set equal to the luma quantization parameter Q_{pY} of the luma coding unit that covers the luma location $(\text{xCb}' + \text{cbWidth}' / 2, \text{yCb}' + \text{cbHeight}' / 2)$ wherein $(\text{xCb}', \text{yCb}')$ the top-left sample of the chroma coding block covering $q_{0,0}$ relative to the top-left chroma sample of the current picture, $\text{cbWidth}'$ specifies the width of the current chroma coding block in luma samples, and cbHeight specifies the height of the current chroma coding block in luma samples.

The variable Q_{pC} is derived as follows:

$$q_{pC} = \text{Clip3}(0, 63, ((Q_{pQ} + Q_{pP} + 1) \gg 1) + c_{QpPicOffset}) \quad (8-1132)$$

$$Q_{pC} = \text{ChromaQpTable}[cIdx - 1][qPi] \quad (8-1133)$$

NOTE – The variable $cQpPicOffset$ provides an adjustment for the value of $pps_cb_qp_offset$ or $pps_cr_qp_offset$, according to whether the filtered chroma component is the Cb or Cr component. However, to avoid the need to vary the amount of the adjustment within the picture, the filtering process does not include an adjustment for the value of $slice_cb_qp_offset$ or $slice_cr_qp_offset$ nor (when $cu_chroma_qp_offset_enabled_flag$ is equal to 1) for the value of $CuQpOffset_{Cb}$, $CuQpOffset_{Cr}$, or $CuQpOffset_{CbCr}$.

The value of the variable β' is determined as specified in Table 8-18 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 63, Q_{pC} + (\text{slice_beta_offset_div2} \ll 1)) \quad (8-1134)$$

where $\text{slice_beta_offset_div2}$ is the value of the syntax element $\text{slice_beta_offset_div2}$ for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (1 \ll (\text{BitDepth}_C - 8)) \quad (8-1135)$$

The value of the variable tc' is determined as specified in Table 8-18 based on the chroma quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (\text{slice_tc_offset_div2} \ll 1)) \quad (8-1136)$$

where $\text{slice_tc_offset_div2}$ is the value of the syntax element $\text{slice_tc_offset_div2}$ for the slice that contains sample $q_{0,0}$.

5.8. Embodiment #8 on chroma QP in deblocking

When making filter decisions for the depicted three samples (with solid circles), the QPs of the luma CU that covers the center position of the chroma CU including the three samples is selected. Therefore, for the 1st, 2nd, and 3rd chroma sample (depicted in FIG. 11), only the QP of CU_{Y3} is utilized, respectively.

In this way, how to select luma CU for chroma quantization/dequantization process is aligned with that for chroma filter decision process.

5.9. Embodiment #9 on QP used for JCCR coded blocks

8.7.3 Scaling process for transform coefficients

Inputs to this process are:

- a luma location (x_{TbY} , y_{TbY}) specifying the top-left sample of the current luma transform block relative to the top-left luma sample of the current picture,
- a variable $nTbW$ specifying the transform block width,
- a variable $nTbH$ specifying the transform block height,
- a variable $cIdx$ specifying the colour component of the current block,
- a variable $bitDepth$ specifying the bit depth of the current colour component.

Output of this process is the $(nTbW) \times (nTbH)$ array d of scaled transform coefficients with elements $d[x][y]$.

The quantization parameter qP is derived as follows:

- If $cIdx$ is equal to 0 and $transform_skip_flag[x_{TbY}][y_{TbY}]$ is equal to 0, the following applies:

$$qP = Qp'_Y \quad (8-950)$$

- Otherwise, if $cIdx$ is equal to 0 (and $transform_skip_flag[x_{TbY}][y_{TbY}]$ is equal to 1), the following applies:

$$qP = \text{Max}(QpPrimeTsMin, Qp'_Y) \quad (8-951)$$

- Otherwise, if $TuCResMode[x_{TbY}][y_{TbY}]$ is unequal to 0 equal to 2, the following applies:

$$qP = Qp'_{CbCr} \quad (8-952)$$

- Otherwise, if $cIdx$ is equal to 1, the following applies:

$$qP = Qp'_{Cb} \quad (8-953)$$

- Otherwise (cIdx is equal to 2), the following applies:

$$qP = Qp'_{Cr} \quad (8-954)$$

5.10. Embodiment #10 on QP used for JCCR coded blocks

8.8.3.2 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:

$$\text{firstCompIdx} = (\text{treeType} == \text{DUAL_TREE_CHROMA}) ? 1 : 0 \quad (8-1022)$$

$$\text{lastCompIdx} = (\text{treeType} == \text{DUAL_TREE_LUMA} \mid \mid \text{ChromaArrayType} == 0) ? 0 : 2 \quad (8-1023)$$

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:

...

5. The picture sample array $recPicture$ is derived as follows:

- **If $cIdx$ is equal to 0, $recPicture$ is set equal to the reconstructed luma picture sample array prior to deblocking $recPictureL$.**
- **Otherwise, if $cIdx$ is equal to 1, $recPicture$ is set equal to the reconstructed chroma picture sample array prior to deblocking $recPictureCb$.**
- **Otherwise ($cIdx$ is equal to 2), $recPicture$ is set equal to the reconstructed chroma picture sample array prior to deblocking $recPictureCr$.**

5. The picture sample array $recPicture[i]$ with $i = 0, 1, 2$ is derived as follows:

- **$recPicture[0]$ is set equal to the reconstructed picture sample array prior to deblocking $recPictureL$.**
- **$recPicture[1]$ is set equal to the reconstructed picture sample array prior to deblocking $recPictureCb$.**
- **$recPicture[2]$ is set equal to the reconstructed picture sample array prior to deblocking $recPictureCr$.**

...

if the $cIdx$ is equal to 1, the following process are applied:

The edge filtering process for one direction is invoked for a coding block as specified in clause 8.8.3.6 with the variable $edgeType$, the variable $cIdx$, the reconstructed picture prior to deblocking $recPicture$, the location (xCb, yCb) , the coding block width $nCbW$, the coding block height $nCbH$, and the arrays bS , $maxFilterLengthPs$, and $maxFilterLengthQs$, as inputs, and the modified reconstructed picture $recPicture$ as output.

8.8.3.5 Derivation process of boundary filtering strength

Inputs to this process are:

- a picture sample array `recPicture`,
- a location (`xCb`, `yCb`) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a variable `nCbW` specifying the width of the current coding block,
- a variable `nCbH` specifying the height of the current coding block,
- a variable `edgeType` specifying whether a vertical (`EDGE_VER`) or a horizontal (`EDGE_HOR`) edge is filtered,
- a variable `cIdx` specifying the colour component of the current coding block,
- a two-dimensional (`nCbW`) $\times$ (`nCbH`) array `edgeFlags`.

Output of this process is a two-dimensional (`nCbW`) $\times$ (`nCbH`) array `bS` specifying the boundary filtering strength.

The variables `xDi`, `yDj`, `xN` and `yN` are derived as follows:

...

For `xDi` with $i = 0..xN$ and `yDj` with $j = 0..yN$, the following applies:

- If `edgeFlags[ xDi ][ yDj ]` is equal to 0, the variable `bS[ xDi ][ yDj ]` is set equal to 0.
- Otherwise, the following applies:
 - The sample values `p0` and `q0` are derived as follows:
 - If `edgeType` is equal to `EDGE_VER`, `p0` is set equal to `recPicture[cIdx][ xCb + xDi - 1 ][ yCb + yDj ]` and `q0` is set equal to `recPicture[cIdx][ xCb + xDi ][ yCb + yDj ]`.
 - Otherwise (`edgeType` is equal to `EDGE_HOR`), `p0` is set equal to `recPicture[cIdx][ xCb + xDi ][ yCb + yDj - 1 ]` and `q0` is set equal to `recPicture[cIdx][ xCb + xDi ][ yCb + yDj ]`.

..

8.8.3.6 Edge filtering process for one direction

Inputs to this process are:

- a variable edgeType specifying whether vertical edges (EDGE_VER) or horizontal edges (EDGE_HOR) are currently processed,
- a variable cIdx specifying the current colour component,
- the reconstructed picture prior to deblocking recPicture,
- a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a variable nCbW specifying the width of the current coding block,
- a variable nCbH specifying the height of the current coding block,
- the array bS specifying the boundary strength,
- the arrays maxFilterLengthPs and maxFilterLengthQs.

Output of this process is the modified reconstructed picture after deblocking recPicture.

...

- Otherwise (cIdx is not equal to 0), the filtering process for edges in the chroma coding block of current coding unit specified by cIdx consists of the following ordered steps:

1. The variable cQpPicOffset is derived as follows:

$$cOpPicOffset = cIdx == 1 ? pps_cb_qp_offset : pps_cr_qp_offset \quad (8-1065)$$

$$cOpPicOffset = (pps_cb_qp_offset + pps_cr_qp_offset + 1) >> 1 \quad (8-1065)$$

2. bS[xDk][yDm] for cIdx = 1 and 2 are modified to 1 if bS[xDk][yDm] for cIdx = 1 is equal to 1 or bS[xDk][yDm] for cIdx = 2 is equal to 1.

3. The decision process for chroma block edges as specified in clause 8.8.3.6.3 is invoked with the chroma picture sample array recPicture, the location of the chroma coding block (xCb, yCb), the location of the chroma block (xBl, yBl) set equal to (xDk, yDm), the edge direction edgeType, **the variable cIdx**, the variable

cQpPicOffset, the boundary filtering strength $bS[xD_k][yD_m]$, and the variable $maxFilterLengthCbCr$ set equal to $maxFilterLengthPs[xD_k][yD_m]$ as inputs, and the modified variable $maxFilterLengthCbCr$, and the variable tC as outputs.

4. When $maxFilterLengthCbCr$ is greater than 0, the filtering process for chroma block edges as specified in clause 8.8.3.6.4 is invoked with the chroma picture sample array $recPicture$, the location of the chroma coding block (xCb, yCb) , the chroma location of the block (xBl, yBl) set equal to (xD_k, yD_m) , the edge direction $edgeType$, the variable $maxFilterLengthCbCr$, and the $cIdx$ equal to 1 as input, and the variable tC as inputs, and the modified chroma picture sample array $recPicture$ as output.

When $maxFilterLengthCbCr$ is greater than 0, the filtering process for chroma block edges as specified in clause 8.8.3.6.4 is invoked with the chroma picture sample array $recPicture$, the location of the chroma coding block (xCb, yCb) , the chroma location of the block (xBl, yBl) set equal to (xD_k, yD_m) , the edge direction $edgeType$, the variable $maxFilterLengthCbCr$, and the $cIdx$ equal to 2 as input, and the variable tC as inputs, and the modified chroma picture sample array $recPicture$ as output.

8.8.3.6.3 Decision process for chroma block edges

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

Inputs to this process are:

- a chroma picture sample array $recPicture$,
- a chroma location (xCb, yCb) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
- a chroma location (xBl, yBl) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
- a variable $edgeType$ specifying whether a vertical ($EDGE_VER$) or a horizontal ($EDGE_HOR$) edge is filtered,
- **a variable $cIdx$ specifying the colour component index,**

- a variable cQpPicOffset specifying the picture-level chroma quantization parameter offset,
- a variable bS specifying the boundary filtering strength,
- a variable maxFilterLengthCbCr.

Outputs of this process are

- the modified variable maxFilterLengthCbCr,
- the variable tc.

The variable maxK is derived as follows:

- If edgeType is equal to EDGE_VER, the following applies:

$$\text{maxK} = (\text{SubHeightC} == 1) ? 3 : 1(8-1124)$$

- Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$\text{maxK} = (\text{SubWidthC} == 1) ? 3 : 1(8-1125)$$

The values p_i and q_i with $\underline{c} = \underline{0..1}$, $i = 0.. \text{maxFilterLengthCbCr}$ and $k = 0.. \text{maxK}$ are derived as follows:

- If edgeType is equal to EDGE_VER, the following applies::

$$q_{\underline{c},i,k} = \text{recPicture}[\underline{c}][x\text{Cb} + x\text{Bl} + i][y\text{Cb} + y\text{Bl} + k] \quad (8-1126)$$

$$p_{\underline{c},i,k} = \text{recPicture}[\underline{c}][x\text{Cb} + x\text{Bl} - i - 1][y\text{Cb} + y\text{Bl} + k] \quad (8-1127)$$

$$\text{subSampleC} = \text{SubHeightC} \quad (8-1128)$$

- Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$q_{\underline{c},i,k} = \text{recPicture}[\underline{c}][x\text{Cb} + x\text{Bl} + k][y\text{Cb} + y\text{Bl} + i] \quad (8-1129)$$

$$p_{\underline{c},i,k} = \text{recPicture}[\underline{c}][x\text{Cb} + x\text{Bl} + k][y\text{Cb} + y\text{Bl} - i - 1] \quad (8-1130)$$

$$\text{subSampleC} = \text{SubWidthC} \quad (8-1131)$$

The variables Q_{pQ} and Q_{pP} are set equal to the Q_{pY} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable Q_{pC} is derived as follows:

$$qPi = \text{Clip3}(0, 63, ((QpQ + QpP + 1) \gg 1) + cOpPicOffset) \quad (8-1132)$$

$$OpC = \text{ChromaOpTable}[cIdx - 1][qPi] + cOpPicOffset \quad (8-1133)$$

$$OpC = ((\text{ChromaOpTable}[0][qPi] + \text{ChromaOpTable}[1][qPi] + 1) \gg 1) + cOpPicOffset \quad (8-1133)$$

NOTE – The variable $cOpPicOffset$ provides an adjustment for the value of $pps_cb_qp_offset$ or $pps_cr_qp_offset$, according to whether the filtered chroma component is the Cb or Cr component. However, to avoid the need to vary the amount of the adjustment within the picture, the filtering process does not include an adjustment for the value of $slice_cb_qp_offset$ or $slice_cr_qp_offset$ nor (when $cu_chroma_qp_offset_enabled_flag$ is equal to 1) for the value of $CuQpOffsetCb$, $CuQpOffsetCr$, or $CuQpOffsetCbCr$.

The value of the variable β' is determined as specified in Table 8-18 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 63, QpC + (slice_beta_offset_div2 \ll 1)) \quad (8-1134)$$

where $slice_beta_offset_div2$ is the value of the syntax element $slice_beta_offset_div2$ for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (1 \ll (\text{BitDepth}_C - 8)) \quad (8-1135)$$

The value of the variable tc' is determined as specified in Table 8-18 based on the chroma quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 65, QpC + 2 * (bS - 1) + (slice_tc_offset_div2 \ll 1)) \quad (8-1136)$$

where $slice_tc_offset_div2$ is the value of the syntax element $slice_tc_offset_div2$ for the slice that contains sample $q_{0,0}$.

The variable tc is derived as follows:

$$tc = (\text{BitDepth}_C < 10) ? (tc' + 2) \gg (10 - \text{BitDepth}_C) : tc' * (1 \ll (\text{BitDepth}_C - 8)) \quad (8-1137)$$

When $maxFilterLengthCbCr$ is equal to 1 and bS is not equal to 2, $maxFilterLengthCbCr$ is set equal to 0.

When maxFilterLengthCbCr is equal to 3, the following ordered steps apply:

1. The variables n1, dpq0_c, dpq1_c, dp_c, dq_c and d_c, with c = 0..1, are derived as follows:

$$n1 = (\text{subSampleC} == 2) ? 1 : 3 \quad (8-1138)$$

$$dp0_c = \text{Abs}(p_{c,2,0} - 2 * p_{c,1,0} + p_{c,0,0}) \quad (8-1139)$$

$$dp1_c = \text{Abs}(p_{c,2,n1} - 2 * p_{c,1,n1} + p_{c,0,n1}) \quad (8-1140)$$

$$dq0_c = \text{Abs}(q_{c,2,0} - 2 * q_{c,1,0} + q_{c,0,0}) \quad (8-1141)$$

$$dq1_c = \text{Abs}(q_{c,2,n1} - 2 * q_{c,1,n1} + q_{c,0,n1}) \quad (8-1142)$$

$$dpq0_c = dp0_c + dq0_c \quad (8-1143)$$

$$dpq1_c = dp1_c + dq1_c \quad (8-1144)$$

$$dp_c = dp0_c + dp1_c \quad (8-1145)$$

$$dq_c = dq0_c + dq1_c \quad (8-1146)$$

$$d_c = dpq0_c + dpq1_c \quad (8-1147)$$

2. The variable d is set equal to (d0 + d1 + 1) >> 1

3. The variables dSam0 and dSam1 are both set equal to 0.

4. When d is less than β, the following ordered steps apply:

- a. The variable dpq is set equal to 2 * dpq0.
- b. The variable dSam0 is derived by invoking the decision process for a chroma sample as specified in clause 8.8.3.6.8 for the sample location (xCb + xBl, yCb + yBl) with sample values p_{0,0}, p_{3,0}, q_{0,0}, and q_{3,0}, the variables dpq, β and t_c as inputs, and the output is assigned to the decision dSam0.
- c. The variable dpq is set equal to 2 * dpq1.
- d. The variable dSam1 is modified as follows:

- If `edgeType` is equal to `EDGE_VER`, for the sample location $(x_{Cb} + x_{Bl}, y_{Cb} + y_{Bl} + n1)$, the decision process for a chroma sample as specified in clause 8.8.3.6.8 is invoked with sample values $p_{0,n1}$, $p_{3,n1}$, $q_{0,n1}$, and $q_{3,n1}$, the variables dpq , β and t_C as inputs, and the output is assigned to the decision `dSam1`.
 - Otherwise (`edgeType` is equal to `EDGE_HOR`), for the sample location $(x_{Cb} + x_{Bl} + n1, y_{Cb} + y_{Bl})$, the decision process for a chroma sample as specified in clause 8.8.3.6.8 is invoked with sample values $p_{0,n1}$, $p_{3,n1}$, $q_{0,n1}$ and $q_{3,n1}$, the variables dpq , β and t_C as inputs, and the output is assigned to the decision `dSam1`.
5. The variable `maxFilterLengthCbCr` is modified as follows:
- If `dSam0` is equal to 1 and `dSam1` is equal to 1, `maxFilterLengthCbCr` is set equal to 3.
 - Otherwise, `maxFilterLengthCbCr` is set equal to 1.

8.8.3.6.4 Filtering process for chroma block edges

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

Inputs to this process are:

- a chroma picture sample array `recPicture`,
- a chroma location (x_{Cb}, y_{Cb}) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
- a chroma location (x_{Bl}, y_{Bl}) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
- a variable `edgeType` specifying whether a vertical (`EDGE_VER`) or a horizontal (`EDGE_HOR`) edge is filtered,
- a variable `maxFilterLengthCbCr` containing the maximum chroma filter length,

6. **a variable `cIdx` specifying the colour component index,**

- the variable `tC`.

Output of this process is the modified chroma picture sample array `recPicture`.

...

The values p_i and q_i with $i = 0..maxFilterLengthCbCr$ and $k = 0..maxK$ are derived as follows:

- If edgeType is equal to EDGE_VER, the following applies:

$$q_{i,k} = \text{recPicture}[\underline{cldx}][x_{Cb} + x_{Bl} + i][y_{Cb} + y_{Bl} + k] \quad (8-1150)$$

$$p_{i,k} = \text{recPicture}[\underline{cldx}][x_{Cb} + x_{Bl} - i - 1][y_{Cb} + y_{Bl} + k] \quad (8-1151)$$

- Otherwise (edgeType is equal to EDGE_HOR), the following applies:

$$q_{i,k} = \text{recPicture}[\underline{cldx}][x_{Cb} + x_{Bl} + k][y_{Cb} + y_{Bl} + i] \quad (8-1152)$$

$$p_{i,k} = \text{recPicture}[\underline{cldx}][x_{Cb} + x_{Bl} + k][y_{Cb} + y_{Bl} - i - 1] \quad (8-1153)$$

Depending on the value of edgeType, the following applies:

- If edgeType is equal to EDGE_VER, for each sample location $(x_{Cb} + x_{Bl}, y_{Cb} + y_{Bl} + k)$, $k = 0..maxK$, the following ordered steps apply:

1. The filtering process for a chroma sample as specified in clause 8.8.3.6.9 is invoked with the variable maxFilterLengthCbCr, the sample values $p_{i,k}$, $q_{i,k}$ with $i = 0..maxFilterLengthCbCr$, the locations $(x_{Cb} + x_{Bl} - i - 1, y_{Cb} + y_{Bl} + k)$ and $(x_{Cb} + x_{Bl} + i, y_{Cb} + y_{Bl} + k)$ with $i = 0..maxFilterLengthCbCr - 1$, and the variable t_c as inputs, and the filtered sample values p_i' and q_i' with $i = 0..maxFilterLengthCbCr - 1$ as outputs.

2. The filtered sample values p_i' and q_i' with $i = 0..maxFilterLengthCbCr - 1$ replace the corresponding samples inside the sample array recPicture as follows:

$$\text{recPicture}[\underline{cldx}][x_{Cb} + x_{Bl} + i][y_{Cb} + y_{Bl} + k] = q_i' \quad (8-1154)$$

$$\text{recPicture}[\underline{cldx}][x_{Cb} + x_{Bl} - i - 1][y_{Cb} + y_{Bl} + k] = p_i' \quad (8-1155)$$

- Otherwise (edgeType is equal to EDGE_HOR), for each sample location $(x_{Cb} + x_{Bl} + k, y_{Cb} + y_{Bl})$, $k = 0..maxK$, the following ordered steps apply:

1. The filtering process for a chroma sample as specified in clause 8.8.3.6.9 is invoked with the variable maxFilterLengthCbCr, the sample values $p_{i,k}$, $q_{i,k}$, with $i = 0..maxFilterLengthCbCr$,

the locations ($x_{Cb} + x_{Bl} + k$, $y_{Cb} + y_{Bl} - i - 1$) and ($x_{Cb} + x_{Bl} + k$, $y_{Cb} + y_{Bl} + i$), and the variable t_c as inputs, and the filtered sample values p_i' and q_i' as outputs.

2. The filtered sample values p_i' and q_i' replace the corresponding samples inside the sample array `recPicture` as follows:

$$\text{recPicture}[\underline{cIdx}][x_{Cb} + x_{Bl} + k][y_{Cb} + y_{Bl} + i] = q_i' \quad (8-1156)$$

$$\text{recPicture}[\underline{cIdx}][x_{Cb} + x_{Bl} + k][y_{Cb} + y_{Bl} - i - 1] = p_i'$$

5.11. Embodiment #11

8.8.3.6.3 Decision process for chroma block edges

...

The variables Q_{p0} and Q_{p1} are set equal to the Q_{py} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable Q_{pc} is derived as follows:

$$q_{Pi} = \text{Clip3}(0, 63, ((Q_{p0} + Q_{p1} + 1) \gg 1) + c_{QpPicOffset}) \quad (8-1132)$$

$$Q_{pc} = \text{ChromaQpTable}[cIdx - 1][q_{Pi}] \quad (8-1133)$$

When ChromaArrayType is not equal to 0 and treeType is equal to SINGLE TREE or DUAL TREE CHROMA, the following applies:

- When treeType is equal to DUAL TREE CHROMA, the variable Q_{pY} is set equal to the luma quantization parameter Q_{pY} of the luma coding unit that covers the luma location ($x_{Cb} + cbWidth / 2$, $y_{Cb} + cbHeight / 2$).**

- The variables q_{PCb} , q_{PCr} and q_{PCbCr} are derived as follows:**

$$q_{PiChroma} = \text{Clip3}(-Q_{pBdOffsetC}, 63, Q_{pY}) \quad (8-935)$$

$$q_{PiCb} = \text{ChromaQpTable}[0][q_{PiChroma}] \quad (8-936)$$

$$\underline{qPiCr = ChromaOpTable[1][qPiChroma]} \\ \underline{(8-937)}$$

$$\underline{qPiCbCr = ChromaOpTable[2][qPiChroma]} \\ \underline{(8-938)}$$

– The chroma quantization parameters for the Cb and Cr components, $Op'Cb$ and $Op'Cr$, and joint Cb-Cr coding $Op'CbCr$ are derived as follows:

$$\underline{Op'Cb = Clip3(-OpBdOffsetC, 63, qPCb + pps_cb_qp_offset + slice_cb_qp_offset + CuOpOffsetCb)} \\ \underline{+ OpBdOffsetC} \quad (8-939)$$

$$\underline{Op'Cr = Clip3(-OpBdOffsetC, 63, qPCr + pps_cr_qp_offset + slice_cr_qp_offset + CuOpOffsetCr)} \\ \underline{+ OpBdOffsetC} \quad (8-940)$$

$$\underline{Op'CbCr = Clip3(-OpBdOffsetC, 63, qPCbCr + pps_cbcr_qp_offset + slice_cbcr_qp_offset + CuOpOffsetCbCr)} \\ \underline{+ OpBdOffsetC} \quad (8-941)$$

The variables OpQ and OpP are set equal to the $Op'Cb$ value when $cIdx$ is equal to 1, or the $Op'Cr$ value when $cIdx$ is equal to 2, or $Op'CbCr$ when $tu_joint_cbcr_residual_flag$ is equal to 1, of the coding units which include the coding blocks containing the sample $q0,0$ and $p0,0$, respectively.

The variable OpC is derived as follows:

$$\underline{OpC = (OpQ + OpP + 1) >> 1}$$

5.12 Embodiment #12

8.8.3.6.3 Decision process for chroma block edges

...

The variables Q_{p0} and Q_{pP} are set equal to the Q_{pY} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable Q_{pC} is derived as follows:

$$q_{Pi} = \text{Clip3}(0, 63, ((Q_{p0} + Q_{pP} + 1) \gg 1) + cQpPicOffset) \quad (8-1132)$$

$$Q_{pC} = \text{ChromaOpTable}[cIdx - 1][q_{Pi}] \quad (8-1133)$$

NOTE – The variable $cQpPicOffset$ provides an adjustment for the value of $pps_cb_qp_offset$ or $pps_cr_qp_offset$, according to whether the filtered chroma component is the Cb or Cr component. However, to avoid the need to vary the amount of the adjustment within the picture, the filtering process does not include an adjustment for the value of $slice_cb_qp_offset$ or $slice_cr_qp_offset$ nor (when $cu_chroma_qp_offset$ enabled flag is equal to 1) for the value of $CuOpOffsetCb$, $CuOpOffsetCr$, or $CuOpOffsetCbCr$.

The variables Q_{p0} and Q_{pP} are set equal to $Q_{p'CbCr} - Q_{pBdOffsetC}$ when $TuCResMode[xTb][yTb]$ is equal to 2, $Q_{p'Cb} - Q_{pBdOffsetC}$ when $cIdx$ is equal to 1; $Q_{p'Cr} - Q_{pBdOffsetC}$ when $cIdx$ is equal to 2 of the transform blocks (xTb, yTb) containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variable Q_{pC} is derived as follows:

$$Q_{pC} = (Q_{p0} + Q_{pP} + 1) \gg 1$$

...

5.13 Embodiment #13

The embodiments are based on JVET-P2001-vE. The newly added texts are highlight by underlined bold italicized text. The deleted texts are marked by underlined bold text.

Decision process for chroma block edges

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

Inputs to this process are:

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

- a chroma location (x_{Bl} , y_{Bl}) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a variable cIdx specifying the colour component index,
- a variable bS specifying the boundary filtering strength,
- a variable maxFilterLengthP specifying the maximum filter length,
- a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are

- the modified filter length variables maxFilterLengthP and maxFilterLengthQ,
- the variable t_C .

...

The variable Q_{pP} is derived as follows:

- The luma location (x_{TbP} , y_{TbP}) is set as the top-left luma sample position of the transform block containing the sample $p_{0,0}$, relative to the top-left luma sample of the picture.
- If TuCResMode[x_{TbP}][y_{TbP}] is equal to 2, Q_{pP} is set equal to $Q_{p'CbCr}$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, if cIdx is equal to 1 and transform skip flag[x_{TbP}][y_{TbP}][cIdx] is equal to 1, Q_{pP} is set equal to $\text{Max}(Q_{pPrimeTsMin}, Q_{p'Cb})$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, if cIdx is equal to 1 transform skip flag[x_{TbP}][y_{TbP}][cIdx] is equal to 0, Q_{pP} is set equal to $Q_{p'Cb}$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, Q_{pP} is set equal to $Q_{p'Cr}$ of the transform block containing the sample $p_{0,0}$.
- If cu_act_enabled flag[x_{TbP}][y_{TbP}] is equal to 1, the Q_{pP} is set equal to $Q_{pP} - 5$.

The variable Q_{pQ} is derived as follows:

- The luma location (xTb_Q , yTb_Q) is set as the top-left luma sample position of the transform block containing the sample $q_{0,0}$, relative to the top-left luma sample of the picture.
- If $TuCResMode[xTb_Q][yTb_Q]$ is equal to 2, Qp_Q is set equal to Qp'_{CbCr} of the transform block containing the sample $q_{0,0}$.
- **Otherwise, if $cIdx$ is equal to 1 and transform skip flag[xTb_O][yTb_O][$cIdx$] is equal to 1, Qp_O is set equal to $Max(QpPrimeTsMin, Qp'_{Cr}$) of the transform block containing the sample $q_{0,0}$.**
- Otherwise, if $cIdx$ is equal to 1 **transform skip flag[xTb_P][yTb_P][$cIdx$] is equal to 0**, Qp_Q is set equal to Qp'_{Cb} of the transform block containing the sample $q_{0,0}$.
- Otherwise, Qp_Q is set equal to Qp'_{Cr} of the transform block containing the sample $q_{0,0}$.
- **If cu_act_enabled flag[xTb_O][yTb_O] is equal to 1, the Qp_O is set equal to $Qp_O - 5$.**
- The variable Qp_C is derived as follows:

$$Qp_C = (Qp_Q - QpBdOffset + Qp_P - QpBdOffset + 1) \gg 1 \quad (1321)$$

...

5.14 Embodiment #14

The embodiments are based on JVET-P2001-vE. The newly added texts are highlight by grey. The deleted texts are marked by underlined bold text.

Decision process for chroma block edges

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

Inputs to this process are:

- a chroma picture sample array `recPicture`,
- a chroma location (xCb , yCb) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
- a chroma location (xB_l , yB_l) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,

- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a variable cIdx specifying the colour component index,
- a variable bS specifying the boundary filtering strength,
- a variable maxFilterLengthP specifying the maximum filter length,
- a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are

- the modified filter length variables maxFilterLengthP and maxFilterLengthQ,
- the variable t_C .
- ...

The variable Q_{PP} is derived as follows:

- The luma location (xTb_P , yTb_P) is set as the top-left luma sample position of the transform block containing the sample $p_{0,0}$, relative to the top-left luma sample of the picture.
- If $TuCRsMode[xTb_P][yTb_P]$ is equal to 2, Q_{PP} is set equal to $Q_{p'_{CbCr}}$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, if cIdx is equal to 1 and transform_skip_flag[xTb_P][yTb_P][cIdx] is equal to 1, Q_{PP} is set equal to $Max(Q_{pPrimeTsMin}, Q_{p'_{Cb}}$) of the transform block containing the sample $p_{0,0}$.
- Otherwise, if cIdx is equal to 1 transform_skip_flag[xTb_P][yTb_P][cIdx] is equal to 0, Q_{PP} is set equal to $Q_{p'_{Cb}}$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, Q_{PP} is set equal to $Q_{p'_{Cr}}$ of the transform block containing the sample $p_{0,0}$.
- If cu_act_enabled_flag[xTb_P][yTb_P] is equal to 1, the Q_{PP} is set equal to $Q_{PP} - 5$.

The variable Q_{PQ} is derived as follows:

- The luma location (xTb_Q , yTb_Q) is set as the top-left luma sample position of the transform block containing the sample $q_{0,0}$, relative to the top-left luma sample of the picture.

- If TuCResMode[xTb_Q][yTb_Q] is equal to 2, Qp_Q is set equal to Qp'cbCr of the transform block containing the sample q_{0,0}.
- Otherwise, if cIdx is equal to 1 and transform skip flag[xTb_Q][yTb_Q][cIdx] is equal to 1, Qp_Q is set equal to Max(OpPrimeTsMin, Qp'cr) of the transform block containing the sample q_{0,0}.
- Otherwise, if cIdx is equal to 1 transform skip flag[xTb_P][yTb_P][cIdx] is equal to 0, Qp_Q is set equal to Qp'cb of the transform block containing the sample q_{0,0}.
- Otherwise, Qp_Q is set equal to Qp'cr of the transform block containing the sample q_{0,0}.
- If cu act enabled flag[xTb_Q][yTb_Q] is equal to 1, the Op_Q is set equal to Op_Q - 3.
- The variable Qp_C is derived as follows:

$$Qp_C = (Qp_Q - QpBdOffset + Qp_P - QpBdOffset + 1) \gg 1 \quad (1321)$$

...

5.15 Embodiment #15

The following FIG. 17 shows the proposed controlling logic.

7.3.2.6 Picture header RBSP syntax

if(!pic_dep_quant_enabled_flag)	
sign_data_hiding_enabled_flag	u(1)
<u>if(deblocking filter override enabled flag) {</u>	
<u>pic deblocking filter override present flag</u>	<u>u(1)</u>
<u>if(pic deblocking filter override present flag) {</u>	
<u>pic deblocking filter override flag</u>	<u>u(1)</u>
<u>if(pic deblocking filter override flag) {</u>	
<u>pic deblocking filter disabled flag</u>	<u>u(1)</u>
<u>if(!pic deblocking filter disabled flag) {</u>	
<u>pic beta offset div2</u>	<u>se(v)</u>

<u>pic tc_offset_div2</u>	<u>se(v)</u>
<u>}</u>	
<u>}</u>	
<u>}</u>	
<u>}</u>	
if(sps_lmcs_enabled_flag) {	
pic_lmcs_enabled_flag	u(1)

7.3.7.1 General slice header syntax

if(deblocking_filter_override_enabled_flag <u>&&</u> <u>!pic_deblocking_filter_override_present_flag</u>)	
slice_deblocking_filter_override_flag	u(1)
if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
}	
}	

5.16 Embodiment #16

7.3.2.4 Picture parameter set RBSP syntax

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)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	

7.3.2.6 Picture header RBSP syntax

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)
pic_cb_beta_offset_div2	se(v)
pic_cb_tc_offset_div2	se(v)
pic_cr_beta_offset_div2	se(v)
pic_cr_tc_offset_div2	se(v)
}	
}	
}	

7.3.7.1 General slice header syntax

if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
slice_cb_beta_offset_div2	se(v)
slice_cb_tc_offset_div2	se(v)
slice_cr_beta_offset_div2	se(v)
slice_cr_tc_offset_div2	se(v)
}	
}	

7.4.3.4 Picture parameter set RBSP semantics

pps_cb_beta_offset_div2 and **pps_cb_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the slice headers of the slices referring to the PPS. The values of **pps_beta_offset_div2** and **pps_tc_offset_div2** shall both be in the range of -6 to 6 , inclusive. When not present, the value of **pps_beta_offset_div2** and **pps_tc_offset_div2** are inferred to be equal to 0.

pps_cr_beta_offset_div2 and **pps_cr_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the slice headers of the slices referring to the PPS. The values of **pps_beta_offset_div2** and

pps_tc_offset_div2 shall both be in the range of -6 to 6 , inclusive. When not present, the value of pps_beta_offset_div2 and pps_tc_offset_div2 are inferred to be equal to 0 .

7.4.3.6 Picture header

pic_cb_beta_offset_div2 and **pic_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to Cb component for the slices associated with the PH. The values of pic_beta_offset_div2 and pic_tc_offset_div2 shall both be in the range of -6 to 6 , inclusive. When not present, the values of pic_beta_offset_div2 and pic_tc_offset_div2 are inferred to be equal to pps_beta_offset_div2 and pps_tc_offset_div2, respectively.

pic_cr_beta_offset_div2 and **pic_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to Cr component for the slices associated with the PH. The values of pic_beta_offset_div2 and pic_tc_offset_div2 shall both be in the range of -6 to 6 , inclusive. When not present, the values of pic_beta_offset_div2 and pic_tc_offset_div2 are inferred to be equal to pps_beta_offset_div2 and pps_tc_offset_div2, respectively.

7.4.8.1 General slice header semantics

slice_cb_beta_offset_div2 and **slice_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to Cb component for the current slice. The values of slice_beta_offset_div2 and slice_tc_offset_div2 shall both be in the range of -6 to 6 , inclusive. When not present, the values of slice_beta_offset_div2 and slice_tc_offset_div2 are inferred to be equal to pic_beta_offset_div2 and pic_tc_offset_div2, respectively.

slice_cr_beta_offset_div2 and **slice_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to Cr component for the current slice. The values of slice_beta_offset_div2 and slice_tc_offset_div2 shall both be in the range of -6 to 6 , inclusive. When not present, the values of slice_beta_offset_div2 and slice_tc_offset_div2 are inferred to be equal to pic_beta_offset_div2 and pic_tc_offset_div2, respectively.

8.8.3.6.3 Decision process for chroma block edges

...

The value of the variable β' is determined as specified in Table 41 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 63, Q_{pC} + (\text{slice_beta_offset_div2} \ll 1)) \quad (1322)$$

■ If cIdx is equal to 1

$$Q = \text{Clip3}(0, 63, Q_{pC} + (\text{slice_cb_beta_offset_div2} \ll 1))$$

■ Otherwise

$$Q = \text{Clip3}(0, 63, Q_{pC} + (\text{slice_cr_beta_offset_div2} \ll 1))$$

where slice_beta_offset_div2 is the value of the syntax element slice_beta_offset_div2 for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (1 \ll (\text{BitDepth} - 8)) \quad (1323)$$

The value of the variable tc' is determined as specified in Table 41 based on the chroma quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (\text{slice_tc_offset_div2} \ll 1)) \quad (1324)$$

■ If cIdx is equal to 1

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (\text{slice_cb_tc_offset_div2} \ll 1))$$

■ Otherwise

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (\text{slice_cr_tc_offset_div2} \ll 1))$$

...

5.17 Embodiment #17

This embodiment is on top of embodiment #15.

7.3.2.4 Picture parameter set RBSP syntax

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)
pps_cb_beta_offset_div2	se(v)
pps_cb_tc_offset_div2	se(v)
pps_cr_beta_offset_div2	se(v)
pps_cr_tc_offset_div2	se(v)
}	

7.3.7.1 General slice header syntax

if(slice_deblocking_filter_override_flag) {	
slice_deblocking_filter_disabled_flag	u(1)
if(!slice_deblocking_filter_disabled_flag) {	
slice_beta_offset_div2	se(v)
slice_tc_offset_div2	se(v)
slice_cb_beta_offset_div2	se(v)
slice_cb_tc_offset_div2	se(v)
slice_cr_beta_offset_div2	se(v)
slice_cr_tc_offset_div2	se(v)
}	

}	
---	--

7.4.3.4 Picture parameter set RBSP semantics

pps_cb_beta_offset_div2 and **pps_cb_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to Cb component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the slice headers of the slices referring to the PPS. The values of **pps_beta_offset_div2** and **pps_tc_offset_div2** shall both be in the range of -6 to 6 , inclusive. When not present, the value of **pps_beta_offset_div2** and **pps_tc_offset_div2** are inferred to be equal to 0.

pps_cr_beta_offset_div2 and **pps_cr_tc_offset_div2** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to Cr component for slices referring to the PPS, unless the default deblocking parameter offsets are overridden by the deblocking parameter offsets present in the slice headers of the slices referring to the PPS. The values of **pps_beta_offset_div2** and **pps_tc_offset_div2** shall both be in the range of -6 to 6 , inclusive. When not present, the value of **pps_beta_offset_div2** and **pps_tc_offset_div2** are inferred to be equal to 0.

7.4.8.1 General slice header semantics

slice_cb_beta_offset_div2 and **slice_cb_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to Cb component for the current slice. The values of **slice_beta_offset_div2** and **slice_tc_offset_div2** shall both be in the range of -6 to 6 , inclusive. When not present, the values of **slice_beta_offset_div2** and **slice_tc_offset_div2** are inferred to be equal to **pps_beta_offset_div2** and **pps_tc_offset_div2**, respectively.

slice_cr_beta_offset_div2 and **slice_cr_tc_offset_div2** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to Cb component for the current slice. The values of **slice_beta_offset_div2** and **slice_tc_offset_div2** shall both be in the range of -6 to 6 , inclusive. When not present, the values of **slice_beta_offset_div2** and **slice_tc_offset_div2** are inferred to be equal to **pps_beta_offset_div2** and **pps_tc_offset_div2**, respectively.

8.8.3.6.3 Decision process for chroma block edges

...

The value of the variable β' is determined as specified in Table 41 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 63, Q_{pC} + (\text{slice_beta_offset_div2} \ll 1)) \quad (1322)$$

■ If cIdx is equal to 1

$$Q = \text{Clip3}(0, 63, Q_{pC} + (\text{slice_cb_beta_offset_div2} \ll 1))$$

■ Otherwise

$$Q = \text{Clip3}(0, 63, Q_{pC} + (\text{slice_cr_beta_offset_div2} \ll 1))$$

where slice_beta_offset_div2 is the value of the syntax element slice_beta_offset_div2 for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (1 \ll (\text{BitDepth} - 8)) \quad (1323)$$

The value of the variable t_C' is determined as specified in Table 41 based on the chroma quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (\text{slice_tc_offset_div2} \ll 1)) \quad (1324)$$

■ If cIdx is equal to 1

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (\text{slice_cb_tc_offset_div2} \ll 1))$$

■ Otherwise

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (\text{slice_cr_tc_offset_div2} \ll 1))$$

...

5.18 Embodiment #18

This embodiment is based on embodiment #17.

7.4.3.4 Picture parameter set RBSP semantics

pps_cb_beta_offset_div2 and pps_cb_tc_offset_div2 specify the default deblocking parameter offsets for β and t_C (divided by 2) that are applied to Cb component for the current PPS. The values

of `pps_beta_offset_div2` and `pps_tc_offset_div2` shall both be in the range of -6 to 6 , inclusive. When not present, the value of `pps_beta_offset_div2` and `pps_tc_offset_div2` are inferred to be equal to 0 .

`pps_cr_beta_offset_div2` and **`pps_cr_tc_offset_div2`** specify the default deblocking parameter offsets for β and tC (divided by 2) that are applied to Cr component for the current PPS. The values of `pps_beta_offset_div2` and `pps_tc_offset_div2` shall both be in the range of -6 to 6 , inclusive. When not present, the value of `pps_beta_offset_div2` and `pps_tc_offset_div2` are inferred to be equal to 0 .

7.4.8.1 General slice header semantics

`slice_cb_beta_offset_div2` and **`slice_cb_tc_offset_div2`** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to Cb component for the current slice. The values of `slice_beta_offset_div2` and `slice_tc_offset_div2` shall both be in the range of -6 to 6 , inclusive.

`slice_cr_beta_offset_div2` and **`slice_cr_tc_offset_div2`** specify the deblocking parameter offsets for β and tC (divided by 2) that are applied to Cr component for the current slice. The values of `slice_beta_offset_div2` and `slice_tc_offset_div2` shall both be in the range of -6 to 6 , inclusive.

8.8.3.6.1 Decision process for luma block edges

...

The value of the variable β' is determined as specified in Table 41 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 63, qP + (\text{slice_beta_offset_div2} \ll 1)) \quad (1262)$$

$$Q = \text{Clip3}(0, 63, qP + ((\text{pps_beta_offset_div2} + \text{slice_beta_offset_div2}) \ll 1)) \quad (1262)$$

where `slice_beta_offset_div2` is the value of the syntax element `slice_beta_offset_div2` for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (1 \ll (\text{BitDepth} - 8)) \quad (1263)$$

The value of the variable tC' is determined as specified in Table 41 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 65, qP + 2 * (\text{bS} - 1) + (\text{slice_tc_offset_div2} \ll 1)) \quad (1264)$$

$$Q = \text{Clip3}(0, 65, qP + 2 * (bS - 1) + ((\text{pps_tc_offset_div2} + \text{slice_tc_offset_div2}) << 1)) \quad (1264)$$

...

8.8.3.6.3 Decision process for chroma block edges

...

The value of the variable β' is determined as specified in Table 41 based on the quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 63, Q_{pC} + (\text{slice_beta_offset_div2} << 1)) \quad (1322)$$

- If cIdx is equal to 1

$$Q = \text{Clip3}(0, 63, Q_{pC} + ((\text{pps_cb_beta_offset_div2} + \text{slice_cb_beta_offset_div2}) << 1))$$

- Otherwise

$$Q = \text{Clip3}(0, 63, Q_{pC} + ((\text{pps_cr_beta_offset_div2} + \text{slice_cr_beta_offset_div2}) << 1))$$

where slice_beta_offset_div2 is the value of the syntax element slice_beta_offset_div2 for the slice that contains sample $q_{0,0}$.

The variable β is derived as follows:

$$\beta = \beta' * (1 << (\text{BitDepth} - 8)) \quad (1323)$$

The value of the variable tc' is determined as specified in Table 41 based on the chroma quantization parameter Q derived as follows:

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + (\text{slice_tc_offset_div2} << 1)) \quad (1324)$$

- If cIdx is equal to 1

$$Q = \text{Clip3}(0, 65, Q_{pC} + 2 * (bS - 1) + ((\text{pps_cb_tc_offset_div2} + \text{slice_cb_tc_offset_div2}) << 1))$$

- Otherwise

$$Q = \text{Clip3}(0, 65, Op_C + 2 * (bS - 1) + ((pps_cr_tc_offset \text{ div2} + slice_cr_tc_offset \text{ div2}) \ll 1))$$

...

5.19 Embodiment #19

This embodiment is related to the ACT.

intra_bdpcm_chroma_flag equal to 1 specifies that BDPCM is applied to the current chroma coding blocks at the location (x0, y0), i.e. the transform is skipped, the intra chroma prediction mode is specified by **intra_bdpcm_chroma_dir_flag**. **intra_bdpcm_chroma_flag** equal to 0 specifies that BDPCM is not applied to the current chroma coding blocks at the location (x0, y0).

When **intra_bdpcm_chroma_flag** is not present it is inferred to be equal to 0, it is inferred to be equal to **sps_bdpcm_chroma_enabled_flag** && **cu_act_enabled_flag** && **intra_bdpcm_luma_flag**.

The variable **BdpcmFlag[x][y][cIdx]** is set equal to **intra_bdpcm_chroma_flag** for $x = x0..x0 + cbWidth - 1$, $y = y0..y0 + cbHeight - 1$ and $cIdx = 1..2$.

intra_bdpcm_chroma_dir_flag equal to 0 specifies that the BDPCM prediction direction is horizontal. **intra_bdpcm_chroma_dir_flag** equal to 1 specifies that the BDPCM prediction direction is vertical.

When **intra_bdpcm_chroma_dir_flag** is not present, it is inferred to be equal to (**cu_act_enabled_flag** ? **intra_bdpcm_luma_dir_flag** : 0).

The variable **BdpcmDir[x][y][cIdx]** is set equal to **intra_bdpcm_chroma_dir_flag** for $x = x0..x0 + cbWidth - 1$, $y = y0..y0 + cbHeight - 1$ and $cIdx = 1..2$.

5.20 Embodiment #20

This embodiment is related to the QP derivation for deblocking.

8.8.3.6.1 Decision process for luma block edges

Inputs to this process are:

- a picture sample array **recPicture**,

- a location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a location (xBl, yBl) specifying the top-left sample of the current block relative to the top-left sample of the current coding block,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a variable bS specifying the boundary filtering strength,
- a variable maxFilterLengthP specifying the maximum filter length,
- a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are:

- the variables dE, dEp and dEq containing decisions,
- the modified filter length variables maxFilterLengthP and maxFilterLengthQ,
- the variable tc.

...

The variables Q_{p0} and Q_{p1} are set equal to the Q_{pY} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

The variables Q_{p0} is derived as follows.

Q_{p0} is set to the Q_{pY} value of the coding units which include the coding blocks containing the sample $q_{0,0}$.

If the transform skip flag of the coding blocks containing the sample $q_{0,0}$ is equal to 1,

$$Q_{p0} = \text{Max}(Q_{p\text{PrimeTsMin}}, Q_{p0} + Q_{p\text{BdOffset}}) - Q_{p\text{BdOffset}}$$

Q_{p1} is set to the Q_{pY} value of the coding units which include the coding blocks containing the sample $p_{0,0}$.

If the transform skip flag of the coding blocks containing the sample $p_{0,0}$ is equal to 1,

$$Q_{p1} = \text{Max}(Q_{p\text{PrimeTsMin}}, Q_{p1} + Q_{p\text{BdOffset}}) - Q_{p\text{BdOffset}}$$

...

8.8.3.6.3 Decision process for chroma block edges

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

Inputs to this process are:

- a chroma picture sample array recPicture,
- a chroma location (xCb, yCb) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
- a chroma location (xBl, yBl) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a variable cIdx specifying the colour component index,
- a variable bS specifying the boundary filtering strength,
- a variable maxFilterLengthP specifying the maximum filter length,
- a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are

- the modified filter length variables maxFilterLengthP and maxFilterLengthQ,
- the variable tc.

...

The variable Q_{pP} is derived as follows:

- The luma location (x_{TbP} , y_{TbP}) is set as the top-left luma sample position of the transform block containing the sample $p_{0,0}$, relative to the top-left luma sample of the picture.
- If TuCResMode[x_{TbP}][y_{TbP}] is equal to 2, Q_{pP} is set equal to $Q_{p'CbCr}$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, if cIdx is equal to 1, Q_{pP} is set equal to $Q_{p'Cb}$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, Q_{pP} is set equal to $Q_{p'Cr}$ of the transform block containing the sample $p_{0,0}$.

The variable Q_{pP} is modified as follows:

$$Q_{pP} = \text{Max}(\text{transform_skip_flag}[xTb_P][yTb_P][cIdx] ? Q_{pPrimeTsMin} : 0, Q_{pP})$$

The variable Q_{pQ} is derived as follows:

- The luma location (xTb_Q , yTb_Q) is set as the top-left luma sample position of the transform block containing the sample $q_{0,0}$, relative to the top-left luma sample of the picture.
- If $TuCResMode[xTb_Q][yTb_Q]$ is equal to 2, Q_{pQ} is set equal to $Q_{p'CbCr}$ of the transform block containing the sample $q_{0,0}$.
- Otherwise, if $cIdx$ is equal to 1, Q_{pQ} is set equal to $Q_{p'Cb}$ of the transform block containing the sample $q_{0,0}$.
- Otherwise, Q_{pQ} is set equal to $Q_{p'Cr}$ of the transform block containing the sample $q_{0,0}$.

The variable Q_{pO} is modified as follows:

$$Q_{pO} = \text{Max}(\text{transform_skip_flag}[xTb_O][yTb_O][cIdx] ? Q_{pPrimeTsMin} : 0, Q_{pO})$$

- The variable Q_{pC} is derived as follows:

$$Q_{pC} = (Q_{pQ} - Q_{pBdOffset} + Q_{pP} - Q_{pBdOffset} + 1) \gg 1 \quad (1321)$$

5.21 Embodiment #21

This embodiment is related to the QP derivation for deblocking.

8.8.3.6.1 Decision process for luma block edges

Inputs to this process are:

- a picture sample array $recPicture$,
- a location (xCb , yCb) specifying the top-left sample of the current coding block relative to the top-left sample of the current picture,
- a location (xBl , yBl) specifying the top-left sample of the current block relative to the top-left sample of the current coding block,
- a variable $edgeType$ specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,
- a variable bS specifying the boundary filtering strength,

- a variable maxFilterLengthP specifying the maximum filter length,
- a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are:

- the variables dE, dEp and dEq containing decisions,
- the modified filter length variables maxFilterLengthP and maxFilterLengthQ,
- the variable tc.

...

The variables Q_{pQ} and Q_{pP} are set equal to the Q_{pY} values of the coding units which include the coding blocks containing the sample $q_{0,0}$ and $p_{0,0}$, respectively.

If the transform skip flag of the coding unit containing the sample $q_{0,0}$ is equal to 1, Q_{pO} is modified to be:

$$Q_{pO} = \text{Max}(Q_{p\text{PrimeTsMin}}, Q_{pO} + Q_{p\text{BdOffset}}) - Q_{p\text{BdOffset}}$$

If the transform skip flag of the coding unit containing the sample $p_{0,0}$ is equal to 1, Q_{pP} is modified to be:

$$Q_{pP} = \text{Max}(Q_{p\text{PrimeTsMin}}, Q_{pP} + Q_{p\text{BdOffset}}) - Q_{p\text{BdOffset}}$$

...

8.8.3.6.3 Decision process for chroma block edges

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

Inputs to this process are:

- a chroma picture sample array recPicture,
- a chroma location (xCb, yCb) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
- a chroma location (xBl, yBl) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
- a variable edgeType specifying whether a vertical (EDGE_VER) or a horizontal (EDGE_HOR) edge is filtered,

- a variable cIdx specifying the colour component index,
- a variable bS specifying the boundary filtering strength,
- a variable maxFilterLengthP specifying the maximum filter length,
- a variable maxFilterLengthQ specifying the maximum filter length.

Outputs of this process are

- the modified filter length variables maxFilterLengthP and maxFilterLengthQ,
- the variable tc.

...

The variable Q_{pP} is derived as follows:

- The luma location (xTb_P , yTb_P) is set as the top-left luma sample position of the transform block containing the sample $p_{0,0}$, relative to the top-left luma sample of the picture.
- If $TuCResMode[xTb_P][yTb_P]$ is equal to 2, Q_{pP} is set equal to $Q_{p'CbCr}$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, if cIdx is equal to 1, Q_{pP} is set equal to $Q_{p'Cb}$ of the transform block containing the sample $p_{0,0}$.
- Otherwise, Q_{pP} is set equal to $Q_{p'Cr}$ of the transform block containing the sample $p_{0,0}$.

If transform_skip_flag[xTb_P][yTb_P][cIdx] is equal to 1, the variable Q_{pP} is modified as follows:

$$Q_{pP} = \text{Max}(Q_{pPrimeTsMin}, Q_{pP})$$

The variable Q_{pQ} is derived as follows:

- The luma location (xTb_Q , yTb_Q) is set as the top-left luma sample position of the transform block containing the sample $q_{0,0}$, relative to the top-left luma sample of the picture.
- If $TuCResMode[xTb_Q][yTb_Q]$ is equal to 2, Q_{pQ} is set equal to $Q_{p'CbCr}$ of the transform block containing the sample $q_{0,0}$.
- Otherwise, if cIdx is equal to 1, Q_{pQ} is set equal to $Q_{p'Cb}$ of the transform block containing the sample $q_{0,0}$.

- Otherwise, Q_{pQ} is set equal to $Q_{p'Cr}$ of the transform block containing the sample $q_{0,0}$.

If transform_skip_flag[xTbO][yTbO][cIdx] is equal to 1, the variable Q_{pO} is modified as follows:

$$Q_{pO} = \text{Max}(Q_{pPrimeTsMin}, Q_{pO})$$

- The variable Q_{pC} is derived as follows:

$$Q_{pC} = (Q_{pQ} - Q_{pBdOffset} + Q_{pP} - Q_{pBdOffset} + 1) \gg 1 \quad (1321)$$

5.22 Embodiment #22

This embodiment is related to CC-ALF. The newly added texts on top of the draft provided by JVET-Q0058 are highlight by underlined bold italicized text. **In this embodiment, the signaling of CC-ALF related information are under the condition check of ChromaArrayType.**

7.3.2.6 Picture header RBSP syntax

picture_header_rbsp() {	Descript or
...	
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)

}	
<u>if(ChromaArrayType != 0)</u>	
pic_cross_component_alf_cb_enabled_flag	u(1)
if(pic_cross_component_alf_cb_enabled_flag) {	
pic_cross_component_alf_cb_aps_id	u(3)
pic_cross_component_cb_filters_signalled_minus1	ue(v)
}	
<u>if(ChromaArrayType != 0)</u>	
pic_cross_component_alf_cr_enabled_flag	u(1)
if(pic_cross_component_alf_cr_enabled_flag) {	
pic_cross_component_alf_cr_aps_id	u(3)
pic_cross_component_cr_filters_signalled_minus1	ue(v)
}	
}	
}	
...	
}	

Alternatively, the newly added “ChromaArrayType != 0” may be replaced by “chroma_format_idc != 0”

5.23 Embodiment #23

This embodiment is related to CC-ALF. The newly added texts on top of the draft provided by JVET-Q0058 are highlight by underlined bold italicized text.

7.3.2.3 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp() {	Descript or
sps_decoding_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
sps_max_sublayers_minus1	u(3)
sps_reserved_zero_4bits	u(4)
sps_ptl_dpb_hrd_params_present_flag	u(1)
if(sps_ptl_dpb_hrd_params_present_flag)	
profile_tier_level(1, sps_max_sublayers_minus1)	
...	
sps_alf_enabled_flag	u(1)
<u>sps_ccalf_enabled_flag</u>	<u>u(1)</u>
...	

semantics

sps_ccalf_enabled_flag equal to 0 specifies that the cross component adaptive loop filter is disabled.

sps_ccalf_enabled_flag equal to 1 specifies that the cross component adaptive loop filter is enabled.

Alternatively, the following may apply:

seq_parameter_set_rbsp() {	Descript or
sps_decoding_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
sps_max_sublayers_minus1	u(3)

sps_reserved_zero_4bits	u(4)
sps_ptl_dpb_hrd_params_present_flag	u(1)
if(sps_ptl_dpb_hrd_params_present_flag)	
profile_tier_level(1, sps_max_sublayers_minus1)	
...	
sps_alf_enabled_flag	u(1)
<u>if (sps_alf_enabled_flag)</u>	
<u>sps_ccalf_enabled_flag</u>	<u>u(1)</u>
...	

Alternatively, the following may apply:

seq_parameter_set_rbsp() {	Descriptor
sps_decoding_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
sps_max_sublayers_minus1	u(3)
sps_reserved_zero_4bits	u(4)
sps_ptl_dpb_hrd_params_present_flag	u(1)
if(sps_ptl_dpb_hrd_params_present_flag)	
profile_tier_level(1, sps_max_sublayers_minus1)	
...	
sps_alf_enabled_flag	u(1)
<u>if (sps_alf_enabled_flag && chroma_format_idc != 0)</u>	
<u>sps_ccalf_enabled_flag</u>	<u>u(1)</u>
...	

Alternatively, the following may apply:

seq_parameter_set_rbsp() {	Descriptor
sps_decoding_parameter_set_id	u(4)
sps_video_parameter_set_id	u(4)
sps_max_sublayers_minus1	u(3)
sps_reserved_zero_4bits	u(4)
sps_ptl_dpb_hrd_params_present_flag	u(1)
if(sps_ptl_dpb_hrd_params_present_flag)	
profile_tier_level(1, sps_max_sublayers_minus1)	
...	
sps_alf_enabled_flag	u(1)
<u>if(sps_alf_enabled_flag && ChromaArrayType != 0)</u>	
<u>sps_ccalf_enabled_flag</u>	<u>u(1)</u>
...	

semantics

sps_ccalf_enabled_flag equal to 0 specifies that the cross component adaptive loop filter is disabled.

sps_ccalf_enabled_flag equal to 1 specifies that the cross component adaptive loop filter is enabled.

When not present, it is inferred to be 0.

7.3.3.2 General constraint information syntax

general_constraint_info() {	Descriptor
general_progressive_source_flag	u(1)
general_interlaced_source_flag	u(1)

general_non_packed_constraint_flag	u(1)
general_frame_only_constraint_flag	u(1)
intra_only_constraint_flag	u(1)
max_bitdepth_constraint_idc	u(4)
max_chroma_format_constraint_idc	u(2)
frame_only_constraint_flag	u(1)
no_qtbt_dual_tree_intra_constraint_flag	u(1)
no_partition_constraints_override_constraint_flag	u(1)
no_sao_constraint_flag	u(1)
no_alf_constraint_flag	u(1)
<u>no_ccalf_constraint_flag</u>	u(1)
no_joint_cbr_constraint_flag	u(1)
...	
}	

no_ccalf_constraint_flag equal to 1 specifies that sps_ccalf_enabled_flag shall be equal to 0.

no_ccalf_constraint_flag equal to 0 does not impose such a constraint.

7.3.2.6 Picture header RBSP syntax

picture_header_rbsp() {	Descript or
...	
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)
}	
}	
<u>{</u>	
<u>if(<i>sps_ccalf_enabled_flag</i> && <i>ChromaArrayType</i> != 0) {</u>	
<u>pic_ccalf_enabled_present_flag</u>	<u>u(1)</u>
<u>if(<i>pic_ccalf_enabled_present_flag</i>) {</u>	
pic_cross_component_alf_cb_enabled_flag	u(1)
if(pic_cross_component_alf_cb_enabled_flag) {	
pic_cross_component_alf_cb_aps_id	u(3)
pic_cross_component_cb_filters_signalled_minus1	ue(v)
}	
pic_cross_component_alf_cr_enabled_flag	u(1)
if(pic_cross_component_alf_cr_enabled_flag) {	
pic_cross_component_alf_cr_aps_id	u(3)
pic_cross_component_cr_filters_signalled_minus1	ue(v)
<u>}</u>	
}	
}	

...	
}	

Alternatively, the following apply:

picture_header_rbsp() {	Descript or
...	
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)
}	
}	
<u>}</u>	
<u>if(sps_ccalf_enabled_flag) {</u>	
<u>pic_ccalf_enabled_present_flag</u>	<u>u(1)</u>
<u>if(pic_ccalf_enabled_present_flag) {</u>	
pic_cross_component_alf_cb_enabled_flag	u(1)

if(pic_cross_component_alf_cb_enabled_flag) {	
pic_cross_component_alf_cb_aps_id	u(3)
pic_cross_component_cb_filters_signalled_minus1	ue(v)
}	
pic_cross_component_alf_cr_enabled_flag	u(1)
if(pic_cross_component_alf_cr_enabled_flag) {	
pic_cross_component_alf_cr_aps_id	u(3)
pic_cross_component_cr_filters_signalled_minus1	ue(v)
}	
<u>}</u>	
}	
...	
}	

Alternatively, the following apply:

picture_header_rbsp() {	Descript or
...	
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)
}	
}	
<u>{</u>	
<u>pic ccalf enabled present flag</u>	<u>u(1)</u>
<u>if(pic ccalf enabled present flag) {</u>	
pic_cross_component_alf_cb_enabled_flag	u(1)
if(pic_cross_component_alf_cb_enabled_flag) {	
pic_cross_component_alf_cb_aps_id	u(3)
pic_cross_component_cb_filters_signalled_minus1	ue(v)
}	
pic_cross_component_alf_cr_enabled_flag	u(1)
if(pic_cross_component_alf_cr_enabled_flag) {	
pic_cross_component_alf_cr_aps_id	u(3)
pic_cross_component_cr_filters_signalled_minus1	ue(v)
}	
}	
...	
}	

pic ccalf enabled present flag equal to 1 specifies that pic ccalf enabled flag, pic cross component alf cb enabled flag, pic cross component alf cb aps id, pic cross component cb filters signalled minus1, pic cross component alf cr enabled flag, pic cross component alf cr aps id, and pic cross component cr filters signalled minus1 are

present in the PH. ***pic_alf_enabled_present_flag*** equal to 0 specifies that ***pic_cross_component_alf_cb_enabled_flag***, ***pic_cross_component_alf_cb_aps_id***, ***pic_cross_component_cb_filters_signalled_minus1***, ***pic_cross_component_alf_cr_enabled_flag***, ***pic_cross_component_alf_cr_aps_id***, and ***pic_cross_component_cr_filters_signalled_minus1*** are not present in the PH. When ***pic_alf_enabled_present_flag*** is not present, it is inferred to be equal to 0.

5.24 Embodiment #24

This embodiment is related to high level syntax and is based on JVET-P2001-vE. The newly added texts are highlight by underlined bold italicized text. The deleted texts are marked by underlined bold text.

7.3.2.3 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp() {	Descript or
...	
if(sps_transform_skip_enabled_flag)	
sps_bdpcm_enabled_flag	u(1)
if(sps_bdpcm_enabled_flag && <u>chroma_format_idc</u> <u>ChromaArrayType</u> == 3)	
sps_bdpcm_chroma_enabled_flag	u(1)
...	
if(<u>chroma_format_idc</u> <u>ChromaArrayType</u> == 3) {	
sps_palette_enabled_flag	u(1)
sps_act_enabled_flag	u(1)
}	
...	

rbsp_trailing_bits()	
}	

5.1 Embodiment #25

This embodiment is related to high level syntax and is based on JVET-P2001-vE. The newly added texts are highlight by underlined bold italicized text. The deleted texts are marked by underlined bold text.

sps_bdpcm_chroma_enabled_flag equal to 1 specifies that **intra_bdpcm_chroma_flag** may be present in the coding unit syntax for intra coding units. **sps_bdpcm_chroma_enabled_flag** equal to 0 specifies that **intra_bdpcm_chroma_flag** is not present in the coding unit syntax for intra coding units. When not present, the value of **sps_bdpcm_chroma_enabled_flag** is inferred to be equal to 0.

When ChromaArrayType is not equal to 3, sps_bdpcm_chroma_enabled_flag shall be equal to 0.

sps_palette_enabled_flag equal to 1 specifies that **pred_mode_plt_flag** may be present in the coding unit syntax. **sps_palette_enabled_flag** equal to 0 specifies that **pred_mode_plt_flag** is not present in the coding unit syntax. When **sps_palette_enabled_flag** is not present, it is inferred to be equal to 0.

When ChromaArrayType is not equal to 3, sps_palette_enabled_flag shall be equal to 0.

sps_act_enabled_flag equal to 1 specifies that adaptive colour transform may be used and the **cu_act_enabled_flag** may be present in the coding unit syntax. **sps_act_enabled_flag** equal to 0 specifies that adaptive colour transform is not used and **cu_act_enabled_flag** is not present in the coding unit syntax. When **sps_act_enabled_flag** is not present, it is inferred to be equal to 0.

When ChromaArrayType is not equal to 3, sps_act_enabled_flag shall be equal to 0.

6. Example implementations of the disclosed technology

[0107] FIG. 12 is a block diagram of a video processing apparatus 1200. The apparatus 1200 may be used to implement one or more of the methods described herein. The apparatus 1200 may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus 1200 may include one or more processors 1202, one or more memories 1204 and video processing hardware 1206. The processor(s) 1202 may be configured to implement one or more methods described in the present document. The memory (memories) 1204 may be used for storing data and code used for implementing the methods and techniques described herein. The video processing hardware 1206 may be used to implement, in hardware circuitry, some techniques described in the present document, and may be partly or completely be a part of the processors 1202 (e.g., graphics processor core GPU or other signal processing circuitry).

[0108] In the present document, the term “video processing” may refer to video encoding, video decoding, video compression or video decompression. For example, video compression algorithms may be applied during conversion from pixel representation of a video to a corresponding bitstream or vice versa. The bitstream representation of a current video block may, for example, correspond to bits that are either co-located or spread in different places within the bitstream, as is defined by the syntax. For example, a macroblock may be encoded in terms of transformed and coded error residual values and also using bits in headers and other fields in the bitstream.

[0109] It will be appreciated that the disclosed methods and techniques will benefit video encoder and/or decoder embodiments incorporated within video processing devices such as smartphones, laptops, desktops, and similar devices by allowing the use of the techniques disclosed in the present document.

[0110] FIG. 13 is a flowchart for an example method 1300 of video processing. The method 1300 includes, at 1310, performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that when a chroma quantization parameter (QP) table is used to derive parameters of the deblocking filter, processing by the chroma QP table is performed on individual chroma QP values.

[0111] FIG. 18 is a block diagram that illustrates an example video coding system 100 that may utilize the techniques of this disclosure.

[0112] As shown in FIG. 18, 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.

[0113] Source device 110 may include a video source 112, a video encoder 114, and an input/output (I/O) interface 116.

[0114] 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.

[0115] Destination device 120 may include an I/O interface 126, a video decoder 124, and a display device 122.

[0116] 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.

[0117] 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.

[0118] FIG. 19 is a block diagram illustrating an example of video encoder 200, which may be video encoder 114 in the system 100 illustrated in FIG. 18.

[0119] Video encoder 200 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 19, 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.

[0120] 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.

[0121] 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.

[0122] 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. 19 separately for purposes of explanation.

[0123] 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.

[0124] 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.

[0125] 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.

[0126] 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.

[0127] 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.

[0128] 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.

[0129] In some examples, motion estimation unit 204 may output a full set of motion information for decoding processing of a decoder.

[0130] 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.

[0131] 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.

[0132] 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.

[0133] As discussed above, video encoder 200 may predictively signal the motion vector. Two examples of predictive signaling techniques that may be implemented

[0134] by video encoder 200 include advanced motion vector predication (AMVP) and merge mode signaling.

[0135] 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.

[0136] 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.

[0137] 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.

[0138] 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.

[0139] 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.

[0140] 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.

[0141] After reconstruction unit 212 reconstructs the video block, loop filtering operation may be performed reduce video blocking artifacts in the video block.

[0142] 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.

[0143] 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 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 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.

[0144] FIG. 20 is a block diagram illustrating an example of video decoder 300 which may be video decoder 114 in the system 100 illustrated in FIG. 18.

[0145] The video decoder 300 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 20, 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.

[0146] In the example of FIG. 20, 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 (FIG. 19).

[0147] 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.

[0148] 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.

[0149] Motion compensation unit 302 may use interpolation filters as used by video encoder 20 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.

[0150] 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.

[0151] 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.

[0152] The 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.

[0153] FIG. 21 is a block diagram showing an example video processing system 1900 in which various techniques disclosed herein may be implemented. Various implementations may include some or all of the components of the system 1900. The system 1900 may include input 1902 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 1902 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.

[0154] The system 1900 may include a coding component 1904 that may implement the various coding or encoding methods described in the present document. The coding component 1904 may reduce the average bitrate of video from the input 1902 to the output of the coding component 1904 to produce a bitstream representation of the video. The coding techniques are therefore sometimes called video compression or video transcoding techniques. The output of the coding component 1904 may be either stored, or transmitted via a communication connected, as represented by the component 1906. The stored or communicated bitstream (or coded) representation of the video received at the input 1902 may be used by the component 1908 for generating pixel values or displayable video that is sent to a display interface 1910. The process of generating user-viewable video from the bitstream 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.

[0155] 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.

[0156] FIG. 22 is a flowchart for example method 2200 of video processing. Operation 2202 includes performing a conversion between a video unit of a video and a bitstream of the video according to a rule, where the rule specifies that whether a cross-component adaptive loop filter (CC-ALF) mode and an adaptive loop filter (ALF) mode are enabled for coding the video unit are indicated in the bitstream in a mutually independent manner.

[0157] In some embodiments of method 2200, when the ALF mode is enabled for the video unit, a color space conversion is performed on residual values of the video unit. In some embodiments of method 2200, when the CC-ALF tool is enabled for the video unit, sample values of the video unit of a video component are filtered with sample values of another video component of the video. In some embodiments of method 2200, the rule specifies that a first syntax element selectively included

in the bitstream indicates whether the CC-ALF mode is enabled for the video unit. In some embodiments of method 2200, the first syntax element is indicated in a sequence level or a video level or a picture level associated with the video unit, and the first syntax element is different from another syntax element included in the bitstream that indicates whether the ALF mode is enabled for the video unit. In some embodiments of method 2200, the first syntax element is included in the bitstream based on the ALF mode being enabled for the video unit.

[0158] In some embodiments of method 2200, the first syntax element is included in the bitstream in case that a coding condition is met, wherein the coding condition comprises: a type of color format of the video, or whether separate plane coding is enabled for the conversion, or a sampling structure of a chroma component of the video. In some embodiments of method 2200, the rule specifies that the bitstream includes a second syntax element that indicates whether one or more syntax elements related to the CC-ALF mode are present in a picture header, and where the second syntax element is included in the picture header or a picture parameter set (PPS) or a slice header. In some embodiments of method 2200, the rule specifies that the bitstream includes the second syntax element based on the ALF mode being enabled for the video unit.

[0159] In some embodiments of method 2200, the ALF is a Weiner filter with neighboring samples as input. In some embodiments of method 2200, the rule specifies that the bitstream includes syntax elements in a picture header or in a picture parameter set (PPS) related to the CC-ALF mode when: a value for a chroma array time is not equal to zero or a color format of the video is not 4:0:0, and the bitstream includes a first syntax element that indicates that the CC-ALF mode is enabled for the video unit, wherein the first syntax element is indicated for a video level of the video that is higher than that of the video unit. In some embodiments of method 2200, the rule specifies that the bitstream includes syntax elements in a picture header or in a picture parameter set (PPS) or in a slice header related to the CC-ALF mode when: a value for a chroma array time is not equal to zero or a color format of the video is not 4:0:0, or the bitstream includes a first syntax element that indicates that the CC-ALF mode is enabled for the video unit, wherein the first syntax element is indicated for a video level of the video that is higher than that of the video unit. In some embodiments of method 2200, the video level includes a sequence parameter set (SPS).

[0160] FIG. 23 is a flowchart for example method 2300 of video processing. Operation 2302 includes performing a conversion between a video unit of a chroma component a video and a bitstream of the video, where the bitstream conforms to a format rule, where the format rule specifies that the bitstream includes a syntax element indicating whether a cross-component filter for the chroma component is enabled for all slices associated with a picture header only when: a value for a chroma array type is not equal to zero, or a color format of the video is not 4:0:0.

[0161] In some embodiments of method 2300, the chroma component includes a Cb chroma component. In some embodiments of method 2300, the chroma component includes a Cr chroma component. In some embodiments of methods 2200-2300, the video unit comprises a coding unit (CU), a prediction unit (PU), or a transform unit (TU). In some embodiments of methods 2200-2300, the performing the conversion comprising encoding the video into the bitstream. In some embodiments of methods 2200-2300, the performing the conversion comprises decoding the video from the bitstream. In some embodiments, a video decoding apparatus comprising a processor configured to implement techniques for embodiments related to methods 2200-2300. In some embodiments, a video encoding apparatus comprising a processor configured to implement techniques for embodiments related to methods 2200-2300. In some embodiments, a computer program product having computer instructions stored thereon, the instructions, when executed by a processor, causes the processor to implement techniques for embodiments related to methods 2200-2300. In some embodiments, a computer readable medium that stores a bitstream generated according to techniques for embodiments related to methods 2200-2300. In some embodiments, a video processing apparatus for storing a bitstream, wherein the video processing apparatus is configured to implement techniques for embodiments related to methods 2200-2300.

[0162] FIG. 24 is a flowchart for example method 2400 of video processing. Operation 2402 includes performing a conversion between a video unit of a video and a bitstream of the video according to a rule, wherein the rule specifies whether the bitstream includes at least one of a control flag of a chroma block-based delta pulse code modulation (BDPCM) mode, a palette mode, or an adaptive color transform (ACT) mode is based on a value of a chroma array type of the video.

[0163] Some embodiments may be described using the following clause-based format. The first set of clauses show example embodiments of techniques discussed in the previous sections.

[0164] 1. A method of video processing, comprising: performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that when a chroma quantization parameter (QP) table is used to derive parameters of the deblocking filter, processing by the chroma QP table is performed on individual chroma QP values.

[0165] 2. The method of clause 1, wherein chroma QP offsets are added to the individual chroma QP values subsequent to the processing by the chroma QP table.

[0166] 3. The method of any of clauses 1-2, wherein the chroma QP offsets are added to values outputted by the chroma QP table.

[0167] 4. The method of any of clauses 1-2, wherein the chroma QP offsets are not considered as input to the chroma QP table.

[0168] 5. The method of clause 2, wherein the chroma QP offsets are at a picture-level or at a video unit-level.

[0169] 6. A method of video processing, comprising: performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein the chroma QP offsets are at picture/slice/tile/brick/subpicture level.

[0170] 7. The method of clause 6, wherein the chroma QP offsets used in the deblocking filter are associated with a coding method applied on a boundary of the video unit.

[0171] 8. The method of clause 7, wherein the coding method is a joint coding of chrominance residuals (JCCR) method.

[0172] 9. A method of video processing, comprising: performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein information pertaining to a same luma coding unit is used in the deblocking filter and for deriving a chroma QP offset.

[0173] 10. The method of clause 9, wherein the same luma coding unit covers a corresponding luma sample of a center position of the video unit, wherein the video unit is a chroma coding unit.

[0174] 11. The method of clause 9, wherein a scaling process is applied to the video unit, and wherein one or more parameters of the deblocking filter depend at least in part on quantization/dequantization parameters of the scaling process.

[0175] 12. The method of clause 11, wherein the quantization/dequantization parameters of the scaling process include the chroma QP offset.

[0176] 13. The method of any of clauses 9-12, wherein the luma sample in the video unit is in the P side or Q side.

[0177] 14. The method of clause 13, wherein the information pertaining to the same luma coding unit depends on a relative position of the coding unit with respect to the same luma coding unit.

[0178] 15. A method of video processing, comprising: performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein an indication of enabling usage of the chroma QP offsets is signaled in the coded representation.

[0179] 16. The method of clause 15, wherein the indication is signaled conditionally in response to detecting one or more flags.

[0180] 17. The method of clause 16, wherein the one or more flags are related to a JCCR enabling flag or a chroma QP offset enabling flag.

[0181] 18. The method of clause 15, wherein the indication is signaled based on a derivation.

[0182] 19. A method of video processing, comprising: performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein the chroma QP offsets used in the deblocking filter are identical of whether JCCR coding method is applied on a boundary of the video unit or a method different from the JCCR coding method is applied on the boundary of the video unit.

[0183] 20. A method of video processing, comprising: performing a conversion between a video unit and a coded representation of the video unit, wherein, during the conversion, a deblocking

filter is used on boundaries of the video unit such that chroma QP offsets are used in the deblocking filter, wherein a boundary strength (BS) of the deblocking filter is calculated without comparing reference pictures and/or a number of motion vectors (MVs) associated with the video unit at a P side boundary with reference pictures and/or a number of motion vectors (MVs) associated with the video unit at a Q side.

[0184] 21. The method of clause 20, wherein the deblocking filter is disabled under one or more conditions.

[0185] 22. The method of clause 21, wherein the one or more conditions are associated with: a magnitude of the motion vectors (MVs) or a threshold value.

[0186] 23. The method of clause 22, wherein the threshold value is associated with at least one of: i. contents of the video unit, ii. a message signaled in DPS/SPS/VPS/PPS/APS/picture header/slice header/tile group header/ Largest coding unit (LCU)/Coding unit (CU)/LCU row/group of LCUs/TU/PU block/Video coding unit, iii. a position of CU/PU/TU/block/Video coding unit, iv. a coded mode of blocks with samples along the boundaries, v. a transform matrix applied to the video units with samples along the boundaries, vi. a shape or dimension of the video unit, vii. an indication of a color format, viii. a coding tree structure, ix. a slice/tile group type and/or picture type, x. a color component, xi. a temporal layer ID, or xii. a profile/level/tier of a standard.

[0187] 24. The method of clause 20, wherein different QP offsets are used for TS coded video units and non-TS coded video units.

[0188] 25. The method of clause 20, wherein a QP used in a luma filtering step is related to a QP used in a scaling process of a luma block.

[0189] The items below are preferably implemented by some embodiments. Additional features are shown in the listing in the previous section, e.g., items 31-32.

[0190] 26. A method of video processing, comprising: determining, for a conversion between a video unit of a component of a video and a coded representation of the video, a size of a quantization group for the video unit, based on a constraint rule that specifies that the size must be larger than K, where K is a positive number; and performing the conversion based on the determining.

[0191] 27. The method of clause 26, wherein the component is a chroma component and $K = 4$.

[0192] 28. The method of clause 26, wherein the component is a luma component and $K = 8$.

[0193] 29. The method of any of clauses 1-28, wherein the conversion comprises encoding the video into the coded representation.

[0194] 30. The method of any of clauses 1-28, wherein the conversion comprises parsing and decoding the coded representation to generate the video.

[0195] 31. A video decoding apparatus comprising a processor configured to implement a method recited in one or more of clauses 1 to 30.

[0196] 32. A video encoding apparatus comprising a processor configured to implement a method recited in one or more of clauses 1 to 30.

[0197] The second set of clauses show example embodiments of techniques discussed in the previous sections (items 42-43).

[0198] 1. A method of video processing, comprising: performing a conversion between a video unit of a video and a bitstream of the video according to a rule, wherein the rule specifies that whether a cross-component adaptive loop filter (CC-ALF) mode and/or an adaptive loop filter (ALF) mode are enabled for coding the video unit are indicated in the bitstream in a mutually independent manner.

[0199] 2. The method of clause 1, wherein when the CC-ALF tool is enabled for the video unit, sample values of the video unit of a video component are filtered with sample values of another video component of the video.

[0200] 3. The method of any of clauses 1 to 2, wherein the rule specifies that a first syntax element selectively included in the bitstream indicates whether the CC-ALF mode is enabled for the video unit.

[0201] 4. The method of clause 3, wherein the first syntax element is indicated in a sequence level or a video level or a picture level associated with the video unit, and wherein the first syntax element is different from another syntax element included in the bitstream that indicates whether the ALF mode is enabled for the video unit.

[0202] 5. The method of clause 3, wherein the first syntax element is included in the bitstream based on the ALF mode being enabled for the video unit.

[0203] 6. The method of clause 3, wherein the first syntax element is included in the bitstream in case that a coding condition is met, wherein the coding condition comprises: a type of color format of the video, or whether separate plane coding is enabled for the conversion, or a sampling structure of a chroma component of the video.

[0204] 7. The method of any of clauses 1 to 2, wherein the rule specifies that the bitstream includes a second syntax element that indicates whether one or more syntax elements related to the CC-ALF mode are present in a picture header, and wherein the second syntax element is included in the picture header or a picture parameter set (PPS) or a slice header.

[0205] 8. The method of clause 7, wherein the one or more syntax elements includes at least one of: a third syntax element that indicates the CC-ALF mode for a Cb color component is enabled for the picture, a fourth syntax element that indicates an adaptation parameter set (APS) id of an ALF APS referenced by the Cb color component of slices in a picture containing the video unit, a fifth syntax element that indicates a number of cross component Cb filters, a sixth syntax element that indicates the CC-ALF mode for a Cr color component is enabled for the picture, a seventh syntax element that indicates an APS id of an ALF APS referenced by Cr color component of the slices in the picture, or an eighth syntax element that indicates a number of cross component Cr filters.

[0206] 9. The method of clause 7, wherein the rule specifies that the bitstream includes the second syntax element based on the ALF mode being enabled for the video unit.

[0207] 10. The method of any of clauses 1-9, wherein the ALF is a Weiner filter with neighboring samples as input.

[0208] 11. The method of any of clauses 1 to 3, wherein the rule specifies that the bitstream includes syntax elements in a picture header or in a picture parameter set (PPS) related to the CC-ALF mode based on the chroma format sampling structure is not monochrome and/or a ninth syntax element that indicates that the CC-ALF mode is enabled in a higher level of the video that is higher than that of the video unit.

[0209] 12. The method of any of clauses 1 to 3, wherein the rule specifies that the bitstream includes syntax elements in a picture header or in a picture parameter set (PPS) related to the CC-ALF mode when: a value for a chroma array type is not equal to zero or a color format of the video is not 4:0:0, and the bitstream includes a first syntax element that indicates that the CC-ALF mode is enabled for the video unit, wherein the first syntax element is indicated for a higher level of the video that is higher than that of the video unit.

[0210] 13. The method of any of clauses 1 to 3, wherein the rule specifies that the bitstream includes syntax elements in a picture header or in a picture parameter set (PPS) or in a slice header related to the CC-ALF mode when: a value for a chroma array type is not equal to zero or a color format of the video is not 4:0:0, or the bitstream includes a first syntax element that indicates that the CC-ALF mode is enabled for the video unit, wherein the first syntax element is indicated for a level of the video that is higher than that of the video unit.

[0211] 14. The method of any of clauses 11 to 12, wherein the video level includes a sequence parameter set (SPS).

[0212] 15. A method of video processing, comprising: performing a conversion between a video unit of a chroma component a video and a bitstream of the video; wherein the bitstream conforms to a format rule, wherein the format rule specifies that the bitstream includes a syntax element indicating whether a cross-component filter for the chroma component is enabled for all slices associated with a picture header only when: a value for a chroma array type is not equal to zero, or a color format of the video is not 4:0:0.

[0213] 16. The method of clause 15, wherein the chroma component includes a Cb chroma component.

[0214] 17. The method of clause 15, wherein the chroma component includes a Cr chroma component.

[0215] 18. The method of any of clauses 1 to 17, wherein the video unit comprises a coding unit (CU), a prediction unit (PU), or a transform unit (TU).

[0216] 19. The method of any of clauses 1 to 18, wherein the performing the conversion comprising encoding the video into the bitstream.

[0217] 20. The method of any of clauses 1 to 18, wherein the performing the conversion comprises encoding the video into the bitstream, and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.

[0218] 21. The method of any of clauses 1 to 18, wherein the performing the conversion comprises decoding the video from the bitstream.

[0219] 22. A method for storing bitstream of a video, comprising: generating a bitstream of a video from a video unit of the video according to a rule, storing the bitstream in a non-transitory computer-readable recording medium; wherein the rule specifies that whether a cross-component adaptive loop filter (CC-ALF) mode and an adaptive loop filter (ALF) mode are enabled for coding the video unit are indicated in the bitstream in a mutually independent manner.

[0220] 23. A video decoding apparatus comprising a processor configured to implement a method recited in one or more of clauses 1 to 22.

[0221] 24. A video encoding apparatus comprising a processor configured to implement a method recited in one or more of clauses 1 to 22.

[0222] 25. A computer program product having computer instructions stored thereon, the instructions, when executed by a processor, causes the processor to implement a method recited in any of clauses 1 to 22.

[0223] 26. A non-transitory computer-readable storage medium that stores a bitstream generated according to the method in any one of clauses 1 to 22.

[0224] 27. A non-transitory computer-readable storage medium storing instructions that cause a processor to implement a method recited in any of clauses 1 to 22.

[0225] The third set of clauses show example embodiments of techniques discussed in the previous sections (e.g., item 44) .

[0226] 1. A method of video processing, comprising: performing a conversion between a video unit of a video and a bitstream of the video according to a rule, wherein the rule specifies whether the bitstream includes at least one of a control flag of a chroma block-based delta pulse code modulation (BDPCM) mode, a palette mode, or an adaptive color transform (ACT) mode is based on a value of a chroma array type of the video.

- [0227] 2. The method of clause 1, wherein the value of the chroma array type describes a sampling of chroma components.
- [0228] 3. The method of clause 1, wherein a first syntax element that indicates the chroma array type is included in a sequence parameter set of the bitstream.
- [0229] 4. The method of clause 2, wherein the value of the chroma array type is set to 3 when the chroma components are not down sampled.
- [0230] 5. The method of clause 1, wherein the at least one of the control flag of the chroma BDPCM mode, the palette mode, or the ACT mode is included in a sequence parameter set of the bitstream.
- [0231] 6. The method of clause 1, wherein the rule specifies that the bitstream includes the control flag of the chroma BDPCM mode only when i) the control flag of the chroma BDPCM mode is true and ii) the chroma array type is equal to 3.
- [0232] 7. The method of clause 1, wherein the rule specifies that the bitstream includes the control flag of the palette mode only when the chroma array type is equal to 3.
- [0233] 8. The method of clause 1, wherein the rule specifies that the bitstream includes the control flag of the ACT mode only when the chroma array type is equal to 3.
- [0234] 9. The method of clause 1, wherein the rule specifies that the bitstream omits the control flag of the chroma BDPCM mode when the chroma array type is unequal to 3.
- [0235] 10. The method of clause 1, wherein the rule specifies that the bitstream omits the control flag of the palette mode when the chroma array type is unequal to 3.
- [0236] 11. The method of clause 1, wherein the rule specifies that the control flag of the ACT mode is excluded in the bitstream when the chroma array type is unequal to 3.
- [0237] 12. The method of clause 1, wherein the rule further specifies that a conformance bitstream satisfies that the control flag of the chroma BDPCM mode is set equal to 0 when the chroma array type is unequal to 3.

[0238] 13. The method of clause 1, wherein the rule further specifies that a conformance bitstream satisfies that the control flag of the palette mode is set equal to 0 when the chroma array type is unequal to 3.

[0239] 14. The method of clause 1, wherein the rule further specifies that the control flag of the ACT mode is set equal to 0 when the chroma array type is unequal to 3.

[0240] 15. The method of any of clauses 1 to 14, wherein the conversion includes encoding the video into the bitstream.

[0241] 16. The method of any of clauses 1 to 14, wherein the conversion includes decoding the video from the bitstream.

[0242] 17. A method of storing a bitstream of a video, comprising, a method recited in any one of clauses 1 to 16, and further including storing the bitstream to a non-transitory computer-readable recording medium.

[0243] 18. A video processing apparatus comprising a processor configured to implement a method recited in any one or more of clauses 1 to 16.

[0244] 19. A computer readable medium storing program code that, when executed, causes a processor to implement a method recited in any one or more of clauses 1 to 16.

[0245] 20. A computer readable medium that stores a coded representation or a bitstream generated according to any of the above described methods.

[0246] 21. A video processing apparatus for storing a bitstream representation, wherein the video processing apparatus is configured to implement a method recited in any one or more of clauses 1 to 20.

[0247] In some embodiments, a solution includes a method, an apparatus, a bitstream generated according to an above-described method or a system for video processing is implemented.

[0248] 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, i.e., 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.

[0249] 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.

[0250] 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).

[0251] 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.

[0252] 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.

[0253] 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.

[0254] 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.

CLAIMS

What is claimed is:

1. A method of video processing, comprising:
 - performing a conversion between a video unit of a video and a bitstream of the video according to a rule,
 - wherein the rule specifies that whether a cross-component adaptive loop filter (CC-ALF) mode and/or an adaptive loop filter (ALF) mode are enabled for coding the video unit are indicated in the bitstream in a mutually independent manner.
2. The method of claim 1, wherein when the CC-ALF tool is enabled for the video unit, sample values of the video unit of a video component are filtered with sample values of another video component of the video.
3. The method of any of claims 1 to 2, wherein the rule specifies that a first syntax element selectively included in the bitstream indicates whether the CC-ALF mode is enabled for the video unit.
4. The method of claim 3,
 - wherein the first syntax element is indicated in a sequence level or a video level or a picture level associated with the video unit, and
 - wherein the first syntax element is different from another syntax element included in the bitstream that indicates whether the ALF mode is enabled for the video unit.
5. The method of claim 3,
 - wherein the first syntax element is included in the bitstream based on the ALF mode being enabled for the video unit.
6. The method of claim 3, wherein the first syntax element is included in the bitstream in case that a coding condition is met, wherein the coding condition comprises:

a type of color format of the video, or
whether separate plane coding is enabled for the conversion, or
a sampling structure of a chroma component of the video.

7. The method of any of claims 1 to 2,

wherein the rule specifies that the bitstream includes a second syntax element that indicates whether one or more syntax elements related to the CC-ALF mode are present in a picture header, and

wherein the second syntax element is included in the picture header or a picture parameter set (PPS) or a slice header.

8. The method of claim 7, wherein the one or more syntax elements includes at least one of:

a third syntax element that indicates the CC-ALF mode for a Cb color component is enabled for the picture,

a fourth syntax element that indicates an adaptation parameter set (APS) id of an ALF APS referenced by the Cb color component of slices in a picture containing the video unit,

a fifth syntax element that indicates a number of cross component Cb filters,

a sixth syntax element that indicates the CC-ALF mode for a Cr color component is enabled for the picture,

a seventh syntax element that indicates an APS id of an ALF APS referenced by Cr color component of the slices in the picture, or

a eighth syntax element that indicates a number of cross component Cr filters.

9. The method of claim 7,

wherein the rule specifies that the bitstream includes the second syntax element based on the ALF mode being enabled for the video unit.

10. The method of any of claims 1-9, wherein the ALF is a Weiner filter with neighboring samples as input.

11. The method of any of claims 1 to 3,

wherein the rule specifies that the bitstream includes syntax elements in a picture header or in a picture parameter set (PPS) related to the CC-ALF mode based on the chroma format sampling structure is not monochrome and/or a ninth syntax element that indicates that the CC-ALF mode is enabled in a higher level of the video that is higher than that of the video unit.

12. The method of any of claims 1 to 3,

wherein the rule specifies that the bitstream includes syntax elements in a picture header or in a picture parameter set (PPS) related to the CC-ALF mode when:

a value for a chroma array type is not equal to zero or a color format of the video is not 4:0:0, and

the bitstream includes a first syntax element that indicates that the CC-ALF mode is enabled for the video unit, wherein the first syntax element is indicated for a higher level of the video that is higher than that of the video unit.

13. The method of any of claims 1 to 3,

wherein the rule specifies that the bitstream includes syntax elements in a picture header or in a picture parameter set (PPS) or in a slice header related to the CC-ALF mode when:

a value for a chroma array type is not equal to zero or a color format of the video is not 4:0:0, or

the bitstream includes a first syntax element that indicates that the CC-ALF mode is enabled for the video unit, wherein the first syntax element is indicated for a level of the video that is higher than that of the video unit.

14. The method of any of claims 11 to 12, wherein the video level includes a sequence parameter set (SPS).

15. A method of video processing, comprising:

performing a conversion between a video unit of a chroma component a video and a bitstream of the video;

wherein the bitstream conforms to a format rule,

wherein the format rule specifies that the bitstream includes a syntax element indicating whether a cross-component filter for the chroma component is enabled for all slices associated with a picture header only when:

a value for a chroma array type is not equal to zero, or
a color format of the video is not 4:0:0.

16. The method of claim 15, wherein the chroma component includes a Cb chroma component.

17. The method of claim 15, wherein the chroma component includes a Cr chroma component.

18. The method of any of claims 1 to 17, wherein the video unit comprises a coding unit (CU), a prediction unit (PU), or a transform unit (TU).

19. The method of any of claims 1 to 18, wherein the performing the conversion comprising encoding the video into the bitstream.

20. The method of any of claims 1 to 18, wherein the performing the conversion comprises encoding the video into the bitstream, and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.

21. The method of any of claims 1 to 18, wherein the performing the conversion comprises decoding the video from the bitstream.

22. A method for storing bitstream of a video, comprising:

generating a bitstream of a video from a video unit of the video according to a rule,
storing the bitstream in a non-transitory computer-readable recording medium;

wherein the rule specifies that whether a cross-component adaptive loop filter (CC-ALF) mode and an adaptive loop filter (ALF) mode are enabled for coding the video unit are indicated in the bitstream in a mutually independent manner.

23. A video decoding apparatus comprising a processor configured to implement a method recited in one or more of claims 1 to 22.

24. A video encoding apparatus comprising a processor configured to implement a method recited in one or more of claims 1 to 22.

25. A computer program product having computer instructions stored thereon, the instructions, when executed by a processor, causes the processor to implement a method recited in any of claims 1 to 22.

26. A non-transitory computer-readable storage medium that stores a bitstream generated according to the method in any one of claims 1 to 22.

27. A non-transitory computer-readable storage medium storing instructions that cause a processor to implement a method recited in any of claims 1 to 22.