

INTERPLAY BETWEEN IN-LOOP FILTERING AND VIDEO SLICES

CROSS REFERENCE TO RELATED APPLICATIONS

[001] Under the applicable patent law and/or rules pursuant to the Paris Convention, this application is made to timely claim the priority to and benefits of International Patent Application No. PCT/CN2020/075216, filed on February 14, 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

[002] This patent document relates to image and video coding and decoding.

BACKGROUND

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

SUMMARY

[004] The present document discloses techniques that can be used by video encoders and decoders for processing coded representation of video using control information useful for decoding of the coded representation.

[005] In an example aspect, a video processing method is disclosed. The method includes performing a conversion between a video comprising a video unit and a bitstream of the video, wherein the bitstream conforms to a format rule, and wherein the format rule specifies that a first syntax element indicating whether an in-loop filtering operation is performed across tile boundaries is selectively included in the bitstream responsive to whether or how the video unit is partitioned into tiles.

[006] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video unit of a video and a bitstream of the video, wherein

the bitstream conforms to a format rule, and wherein the format rule specifies that a syntax element indicating whether an in-loop filtering operation is performed across slice boundaries is selectively included in the bitstream responsive to whether or how the video unit is partitioned into slices

[007] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a bitstream of the video, wherein the bitstream conforms to a format rule, wherein the format rule specifies that information about applicability of a tool to the conversion is indicated in the bitstream at a video slice level and/or at a video picture level, and wherein the tool maps luma samples to particular values and selectively applies a scaling operation to the values of chroma samples.

[008] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a bitstream of the video, wherein the conversion conforms to a size rule, and wherein, the size rule specifies a maximum size of the video region coded using a transform-skip (TS) coding scheme or a block-based delta pulse code modulation (BDPCM) coding scheme, or a maximum size of a transform block for the video region according to a coding characteristic of the video region.

[009] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a bitstream of the video, wherein the bitstream conforms to a format rule that specifies that a minimum allowed coding block size used during the conversion controls whether and/or how a maximum allowed transform block size is included in the bitstream.

[0010] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a bitstream of the video, wherein the bitstream conforms to a format rule that specifies that a minimum allowed coding block size used during the conversion controls whether and/or how an indication of a maximum size of a video region for coding or decoding using a transform-skip (TS) scheme or a block-based delta pulse code modulation (BDPCM) scheme is included in the bitstream.

[0011] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video unit of a video and a bitstream of the video according to a format rule, wherein the bitstream conforms to the format rule that specifies that a minimum allowed coding block size used during the conversion controls whether and/or how a field

indicative of whether a transform-skip (TS) scheme or a block-based delta pulse code modulation (BDPCM) scheme is enabled or disabled is included in the bitstream.

[0012] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video unit of a video and a bitstream of the video according to a format rule, wherein the bitstream conforms to the format rule that specifies that a minimum allowed coding block size used during the conversion controls whether and/or how a field carrying information regarding applicability of a coding tool in the conversion is included at a video region level in the bitstream.

[0013] In another example aspect, a video processing method is disclosed. The method includes determining, for a conversion between a video region of a video and a bitstream of the video, that due a dual-tree coding scheme being enabled for the video region, that splitting schemes used for luma component and a chroma component of the video have different minimum allowed block sizes, and performing the conversion based on the determining.

[0014] In another example aspect, a video processing method is disclosed. The method includes determining, for a conversion between a video region of a video and a bitstream of the video, a maximum number of sub-block based merge candidates allowed for the video region based on a rule; and performing the conversion based on the determining, wherein the rule specifies that maximum number of sub-block based merge candidates used during the conversion is derivable as a sum of a first variable and a second variable, wherein the first variable is equal to zero responsive to an affine prediction being disabled, and wherein the second variable is based on whether a sub-block based temporal motion vector prediction (sbTMVP) is enabled.

[0015] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a bitstream of the video region by conforming a processing rule that is applicable to the conversion due to the video being a 4:2:2 video or a 4:4:4 video, wherein the processing rule defines that chroma and luma are aligned for one or more of the following: (a) a number of pixel lines between a virtual boundary of an adaptive loop filtering (ALF) operation and a bottom boundary of a coding tree block (CTB); or (b) a filter strength of a filter applied to a row between the virtual boundary of the ALF operation and the bottom boundary of the CTB; or (c) a padding method used for padding luma and chroma samples in a same row.

[0016] In another example aspect, a video processing method is disclosed. The method includes determining, for a conversion between a video unit of a video and a coded representation of the video, whether an indication of applicability of in-loop filtering across video regions of the video unit is included in the coded representation; and performing the conversion based on the determining.

[0017] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a coded representation of the video, wherein the coded representation conforms to a format rule, wherein the format rule specifies that information about applicability a luma mapping with chroma scaling (LMCS) tool to the conversion is indicated in the coded representation at a video slice level; wherein, the LMCS tool includes, during the conversion, to construct the current video block based on a first domain and a second domain and/or chroma residue is scaled in a luma-dependent manner.

[0018] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video and a coded representation of the video, wherein the conversion conforms to a size rule that specifies that: enforcing the size rule, during encoding, a maximum size of a video region coded using a transform-skip coding scheme or a delta pulse coded modulation coding scheme, or parsing and decoding the coded representation, during decoding, by enforcing the size rule for the maximum size of the video region decoded using a transform-skip decoding scheme or a delta pulse coded modulation decoding scheme.

[0019] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a coded representation of the video, wherein the coded representation conforms to a format rule that specifies that a minimum allowed transform block size used during the conversion controls whether or how an indication of maximum allowed transform block size is included in the coded representation.

[0020] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video unit of a video and a coded representation of the video according to a format rule, wherein the coded representation conforms to the format rule that specifies that a minimum allowed coding block size used during the conversion controls whether a field carrying information regarding applicability of a coding tool in the conversion is included at a video region level.

[0021] In another example aspect, a video processing method is disclosed. The method includes determining, for a conversion between a video region of a video and a coded representation of the video, that due to using a dual-tree coding is used for the conversion, that splitting schemes used for luma and chroma components of the video have different allowed minimum block sizes for the luma and chroma components, and performing the conversion based on the determining.

[0022] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a coded representation of the video region by conforming a format rule for the coded representation; wherein the format rule specifies that a maximum number of sub-block based merge candidates used during the conversion is derivable as a sum of a first variable and a second variable, wherein use of affine prediction controls a value of the first variable, and wherein use of sub-block based temporal motion vector prediction controls a value of the second variable.

[0023] In another example aspect, a video processing method is disclosed. The method includes performing a conversion between a video region of a video and a coded representation of the video region by conforming a processing rule that is applicable to the conversion due to the video being a 4:2:2 or a 4:4:4 video, wherein the processing rule defines that chroma and luma are aligned for one or more of the following: (a) a number of pixel lines between a virtual boundary of an adaptive loop filtering operation and a bottom boundary of a coding tree block; or (b) a filter strength of a filter applied to a row between the virtual boundary of the adaptive loop filtering operation and the bottom boundary of the coding tree block; or (c) a padding method used for padding video samples in a row.

[0024] In yet another example aspect, a video encoder apparatus is disclosed. The video encoder comprises a processor configured to implement above-described methods.

[0025] In yet another example aspect, a video decoder apparatus is disclosed. The video decoder comprises a processor configured to implement above-described methods.

[0026] In yet another example aspect, a computer readable medium having code stored thereon is disclosed. The code embodies one of the methods described herein in the form of processor-executable code.

[0027] These, and other, features are described throughout the present document.

BRIEF DESCRIPTION OF DRAWINGS

[0028] FIG. 1 shows an example of raster-scan slice partitioning of a picture, where the picture is divided into 12 tiles and 3 raster-scan slices.

[0029] FIG. 2 shows an example of rectangular slice partitioning of a picture, where the picture is divided into 24 tiles (6 tile columns and 4 tile rows) and 9 rectangular slices.

[0030] FIG. 3 shows an example of a picture partitioned into tiles and rectangular slices, where the picture is divided into 4 tiles (2 tile columns and 2 tile rows) and 4 rectangular slices.

[0031] FIG. 4 shows a picture that is partitioned into 15 tiles, 24 slices and 24 subpictures.

[0032] FIG. 5 shows a nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture.

[0033] FIG. 6 shows example of ALF filter shapes.

[0034] FIGS. 7A-7D show subsampled Laplacian calculation. FIG. 7A shows Subsampled positions for vertical gradient. FIG. 7B shows Subsampled positions for horizontal gradient. FIG. 7C shows Subsampled positions for diagonal gradient. FIG. 7D shows Subsampled positions for diagonal gradient.

[0035] FIG. 8 shows an example of a loop filter line buffer requirement in VTM-4.0 for Luma component.

[0036] FIG. 9 shows an example of a loop filter line buffer requirement in VTM-4.0 for Chroma component.

[0037] FIG. 10 shows a modified block classification at virtual boundaries.

[0038] FIG. 11 shows an example of modified ALF filtering for Luma component at virtual boundaries.

[0039] FIG. 12A-12C show Modified luma ALF filtering at virtual boundary

[0040] FIG. 13 shows examples of repetitive padding for luma ALF filtering at picture or subpicture or slice or tile boundary.

[0041] FIGS. 14A-14D show an example for mirrored padding for ALF.

[0042] FIG. 15 is a block diagram of an example video processing system.

[0043] FIG. 16 is a block diagram of a video processing apparatus.

[0044] FIG. 17 is a flowchart for an example method of video processing.

[0045] FIG. 18 is a block diagram that illustrates a video coding system in accordance with some embodiments of the present disclosure.

[0046] FIG. 19 is a block diagram that illustrates an encoder in accordance with some embodiments of the present disclosure.

[0047] FIG. 20 is a block diagram that illustrates a decoder in accordance with some embodiments of the present disclosure.

[0048] FIGS. 21 to 31 are flowcharts for example methods of video processing.

DETAILED DESCRIPTION

[0049] Section headings are used in the present document for ease of understanding and do not limit the applicability of techniques and embodiments disclosed in each section only to that section. Furthermore, H.266 terminology is used in some description only for ease of understanding and not for limiting scope of the disclosed techniques. As such, the techniques described herein are applicable to other video codec protocols and designs also.

1. Summary

[0050] This document is related to video coding technologies. Specifically, it is about signalling of subpictures, tiles, and slices. The ideas may be applied individually or in various combination, to any video coding standard or non-standard video codec that supports multi-layer video coding, e.g., the being-developed Versatile Video Coding (VVC).

2. Abbreviations

APS	Adaptation Parameter Set
AU	Access Unit
AUD	Access Unit Delimiter
AVC	Advanced Video Coding
CLVS	Coded Layer Video Sequence
CPB	Coded Picture Buffer
CRA	Clean Random Access
CTU	Coding Tree Unit
CVS	Coded Video Sequence
DPB	Decoded Picture Buffer

DPS	Decoding Parameter Set
EOB	End Of Bitstream
EOS	End Of Sequence
GDR	Gradual Decoding Refresh
HEVC	High Efficiency Video Coding
HRD	Hypothetical Reference Decoder
IDR	Instantaneous Decoding Refresh
JEM	Joint Exploration Model
MCTS	Motion-Constrained Tile Sets
NAL	Network Abstraction Layer
OLS	Output Layer Set
PH	Picture Header
PPS	Picture Parameter Set
PTL	Profile, Tier and Level
PU	Picture Unit
RBSP	Raw Byte Sequence Payload
SEI	Supplemental Enhancement Information
SPS	Sequence Parameter Set
SVC	Scalable Video Coding
VCL	Video Coding Layer
VPS	Video Parameter Set
VTM	VVC Test Model
VUI	Video Usability Information
VVC	Versatile Video Coding

3. Initial Discussion

[0051] 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, the 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). The JVET meeting is concurrently held once every quarter, and the new coding standard is targeting at 50% bitrate reduction as compared to HEVC. The new video coding standard was officially named as Versatile Video Coding (VVC) in the April 2018 JVET meeting, and the first version of VVC test model (VTM) was released at that time. As there are continuous effort contributing to VVC standardization, new coding techniques are being adopted to the VVC standard in every JVET meeting. The VVC working draft and test model VTM are then updated after every meeting. The VVC project is now aiming for technical completion (FDIS) at the July 2020 meeting.

3.1. Picture partitioning schemes in HEVC

[0052] HEVC includes four different picture partitioning schemes, namely regular slices, dependent slices, tiles, and Wavefront Parallel Processing (WPP), which may be applied for Maximum Transfer Unit (MTU) size matching, parallel processing, and reduced end-to-end delay.

[0053] Regular slices are similar as in H.264/AVC. Each regular slice is encapsulated in its own NAL unit, and in-picture prediction (intra sample prediction, motion information prediction, coding mode prediction) and entropy coding dependency across slice boundaries are disabled. Thus a regular slice can be reconstructed independently from other regular slices within the same picture (though there may still have interdependencies due to loop filtering operations).

[0054] The regular slice is the only tool that can be used for parallelization that is also available, in virtually identical form, in H.264/AVC. Regular slices based parallelization does not require

much inter-processor or inter-core communication (except for inter-processor or inter-core data sharing for motion compensation when decoding a predictively coded picture, which is typically much heavier than inter-processor or inter-core data sharing due to in-picture prediction).

However, for the same reason, the use of regular slices can incur substantial coding overhead due to the bit cost of the slice header and due to the lack of prediction across the slice boundaries.

Further, regular slices (in contrast to the other tools mentioned below) also serve as the key mechanism for bitstream partitioning to match MTU size requirements, due to the in-picture independence of regular slices and that each regular slice is encapsulated in its own NAL unit. In many cases, the goal of parallelization and the goal of MTU size matching place contradicting demands to the slice layout in a picture. The realization of this situation led to the development of the parallelization tools mentioned below.

[0055] Dependent slices have short slice headers and allow partitioning of the bitstream at treeblock boundaries without breaking any in-picture prediction. Basically, dependent slices provide fragmentation of regular slices into multiple NAL units, to provide reduced end-to-end delay by allowing a part of a regular slice to be sent out before the encoding of the entire regular slice is finished.

[0056] In WPP, the picture is partitioned into single rows of coding tree blocks (CTBs). Entropy decoding and prediction are allowed to use data from CTBs in other partitions. Parallel processing is possible through parallel decoding of CTB rows, where the start of the decoding of a CTB row is delayed by two CTBs, so to ensure that data related to a CTB above and to the right of the subject CTB is available before the subject CTB is being decoded. Using this staggered start (which appears like a wavefront when represented graphically), parallelization is possible with up to as many processors/cores as the picture contains CTB rows. Because in-picture prediction between neighboring treeblock rows within a picture is permitted, the required inter-processor/inter-core communication to enable in-picture prediction can be substantial. The WPP partitioning does not result in the production of additional NAL units compared to when it is not applied, thus WPP is not a tool for MTU size matching. However, if MTU size matching is required, regular slices can be used with WPP, with certain coding overhead.

[0057] Tiles define horizontal and vertical boundaries that partition a picture into tile columns and rows. Tile column runs from the top of a picture to the bottom of the picture. Likewise, tile

row runs from the left of the picture to the right of the picture. The number of tiles in a picture can be derived simply as number of tile columns multiply by number of tile rows.

[0058] The scan order of CTBs is changed to be local within a tile (in the order of a CTB raster scan of a tile), before decoding the top-left CTB of the next tile in the order of tile raster scan of a picture. Similar to regular slices, tiles break in-picture prediction dependencies as well as entropy decoding dependencies. However, they do not need to be included into individual NAL units (same as WPP in this regard); hence tiles cannot be used for MTU size matching. Each tile can be processed by one processor/core, and the inter-processor/inter-core communication required for in-picture prediction between processing units decoding neighboring tiles is limited to conveying the shared slice header in cases a slice is spanning more than one tile, and loop filtering related sharing of reconstructed samples and metadata. When more than one tile or WPP segment is included in a slice, the entry point byte offset for each tile or WPP segment other than the first one in the slice is signaled in the slice header.

[0059] For simplicity, restrictions on the application of the four different picture partitioning schemes have been specified in HEVC. A given coded video sequence cannot include both tiles and wavefronts for most of the profiles specified in HEVC. For each slice and tile, either or both of the following conditions must be fulfilled: 1) all coded treeblocks in a slice belong to the same tile; 2) all coded treeblocks in a tile belong to the same slice. Finally, a wavefront segment contains exactly one CTB row, and when WPP is in use, if a slice starts within a CTB row, it must end in the same CTB row.

[0060] A recent amendment to HEVC is specified in the JCT-VC output document JCTVC-AC1005, J. Boyce, A. Ramasubramonian, R. Skupin, G. J. Sullivan, A. Tourapis, Y.-K. Wang (editors), "HEVC Additional Supplemental Enhancement Information (Draft 4)," Oct. 24, 2017, publicly available herein: http://phenix.int-evry.fr/jct/doc_end_user/documents/29_Macau/wg11/JCTVC-AC1005-v2.zip. With this amendment included, HEVC specifies three MCTS-related SEI messages, namely temporal MCTSs SEI message, MCTSs extraction information set SEI message, and MCTSs extraction information nesting SEI message.

[0061] The temporal MCTSs SEI message indicates existence of MCTSs in the bitstream and signals the MCTSs. For each MCTS, motion vectors are restricted to point to full-sample locations inside the MCTS and to fractional-sample locations that require only full-sample

locations inside the MCTS for interpolation, and the usage of motion vector candidates for temporal motion vector prediction derived from blocks outside the MCTS is disallowed. This way, each MCTS may be independently decoded without the existence of tiles not included in the MCTS.

[0062] The MCTSs extraction information sets SEI message provides supplemental information that can be used in the MCTS sub-bitstream extraction (specified as part of the semantics of the SEI message) to generate a conforming bitstream for an MCTS set. The information consists of a number of extraction information sets, each defining a number of MCTS sets and containing RBSP bytes of the replacement VPSs, SPSs, and PPSs to be used during the MCTS sub-bitstream extraction process. When extracting a sub-bitstream according to the MCTS sub-bitstream extraction process, parameter sets (VPSs, SPSs, and PPSs) need to be rewritten or replaced, slice headers need to be slightly updated because one or all of the slice address related syntax elements (including `first_slice_segment_in_pic_flag` and `slice_segment_address`) typically would need to have different values.

3.2. Partitioning of pictures in VVC

[0063] In VVC, A picture is divided into one or more tile rows and one or more tile columns. A tile is a sequence of CTUs that covers a rectangular region of a picture. The CTUs in a tile are scanned in raster scan order within that tile.

[0064] A slice consists of an integer number of complete tiles or an integer number of consecutive complete CTU rows within a tile of a picture.

[0065] Two modes of slices are supported, namely the raster-scan slice mode and the rectangular slice mode. In the raster-scan slice mode, a slice contains a sequence of complete tiles in a tile raster scan of a picture. In the rectangular slice mode, a slice contains either a number of complete tiles that collectively form a rectangular region of the picture or a number of consecutive complete CTU rows of one tile that collectively form a rectangular region of the picture. Tiles within a rectangular slice are scanned in tile raster scan order within the rectangular region corresponding to that slice.

[0066] A subpicture contains one or more slices that collectively cover a rectangular region of a picture.

[0067] FIG. 1 shows an example of raster-scan slice partitioning of a picture, where the picture is divided into 12 tiles and 3 raster-scan slices.

[0068] FIG. 2 shows an example of rectangular slice partitioning of a picture, where the picture is divided into 24 tiles (6 tile columns and 4 tile rows) and 9 rectangular slices.

[0069] FIG. 3 shows an example of a picture partitioned into tiles and rectangular slices, where the picture is divided into 4 tiles (2 tile columns and 2 tile rows) and 4 rectangular slices.

[0070] FIG. 4 shows an example of subpicture partitioning of a picture, where a picture is partitioned into 18 tiles, 12 on the left-hand side each covering one slice of 4 by 4 CTUs and 6 tiles on the right-hand side each covering 2 vertically-stacked slices of 2 by 2 CTUs, altogether resulting in 24 slices and 24 subpictures of varying dimensions (each slice is a subpicture).

3.3. Signalling of SPS/PPS/Picture header/Slice header in VVC (as JVET-Q2001-vB)

7.3.2.3 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp() {	Descriptor
sps_seq_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)	
gdr_enabled_flag	u(1)
chroma_format_idc	u(2)
if(chroma_format_idc == 3)	
separate_colour_plane_flag	u(1)
res_change_in_clvs_allowed_flag	u(1)
pic_width_max_in_luma_samples	ue(v)

pic_height_max_in_luma_samples	ue(v)
sps_conformance_window_flag	u(1)
if(sps_conformance_window_flag) {	
sps_conf_win_left_offset	ue(v)
sps_conf_win_right_offset	ue(v)
sps_conf_win_top_offset	ue(v)
sps_conf_win_bottom_offset	ue(v)
}	
sps_log2_ctu_size_minus5	u(2)
subpic_info_present_flag	u(1)
if(subpic_info_present_flag) {	
sps_num_subpics_minus1	ue(v)
sps_independent_subpics_flag	u(1)
for(i = 0; sps_num_subpics_minus1 > 0 && i <= sps_num_subpics_minus1; i++) {	
if(i > 0 && pic_width_max_in_luma_samples > CtbSizeY)	
subpic_ctu_top_left_x[i]	u(v)
if(i > 0 && pic_height_max_in_luma_samples > CtbSizeY) {	
subpic_ctu_top_left_y[i]	u(v)
if(i < sps_num_subpics_minus1 && pic_width_max_in_luma_samples > CtbSizeY)	
subpic_width_minus1[i]	u(v)
if(i < sps_num_subpics_minus1 && pic_height_max_in_luma_samples > CtbSizeY)	
subpic_height_minus1[i]	u(v)
if(!sps_independent_subpics_flag) {	
subpic_treated_as_pic_flag[i]	u(1)

loop_filter_across_subpic_enabled_flag[i]	u(1)
}	
}	
sps_subpic_id_len_minus1	ue(v)
subpic_id_mapping_explicitly_signalled_flag	u(1)
if(subpic_id_mapping_explicitly_signalled_flag) {	
subpic_id_mapping_in_sps_flag	u(1)
if(subpic_id_mapping_in_sps_flag)	
for(i = 0; i <= sps_num_subpics_minus1; i++)	
sps_subpic_id[i]	u(v)
}	
}	
bit_depth_minus8	ue(v)
sps_entropy_coding_sync_enabled_flag	u(1)
if(sps_entropy_coding_sync_enabled_flag)	
sps_wpp_entry_point_offsets_present_flag	u(1)
sps_weighted_pred_flag	u(1)
sps_weighted_bipred_flag	u(1)
log2_max_pic_order_cnt_lsb_minus4	u(4)
sps_poc_msb_flag	u(1)
if(sps_poc_msb_flag)	
poc_msb_len_minus1	ue(v)
num_extra_ph_bits_bytes	u(2)
extra_ph_bits_struct(num_extra_ph_bits_bytes)	
num_extra_sh_bits_bytes	u(2)
extra_sh_bits_struct(num_extra_sh_bits_bytes)	

if(sps_max_sublayers_minus1 > 0)	
sps_sublayer_dpb_params_flag	u(1)
if(sps_ptl_dpb_hrd_params_present_flag)	
dpb_parameters(sps_max_sublayers_minus1, sps_sublayer_dpb_params_flag)	
long_term_ref_pics_flag	u(1)
inter_layer_ref_pics_present_flag	u(1)
sps_idr_rpl_present_flag	u(1)
rpl1_same_as_rpl0_flag	u(1)
for(i = 0; i < !rpl1_same_as_rpl0_flag ? 2 : 1; i++) {	
num_ref_pic_lists_in_sps[i]	ue(v)
for(j = 0; j < num_ref_pic_lists_in_sps[i]; j++)	
ref_pic_list_struct(i, j)	
}	
if(ChromaArrayType != 0)	
qtbtt_dual_tree_intra_flag	u(1)
log2_min_luma_coding_block_size_minus2	ue(v)
partition_constraints_override_enabled_flag	u(1)
sps_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
sps_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)
if(sps_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
sps_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)
sps_log2_diff_max_tt_min_qt_intra_slice_luma	ue(v)
}	
sps_log2_diff_min_qt_min_cb_inter_slice	ue(v)
sps_max_mtt_hierarchy_depth_inter_slice	ue(v)

if(sps_max_mtt_hierarchy_depth_inter_slice != 0) {	
sps_log2_diff_max_bt_min_qt_inter_slice	ue(v)
sps_log2_diff_max_tt_min_qt_inter_slice	ue(v)
}	
if(qtbtt_dual_tree_intra_flag) {	
sps_log2_diff_min_qt_min_cb_intra_slice_chroma	ue(v)
sps_max_mtt_hierarchy_depth_intra_slice_chroma	ue(v)
if(sps_max_mtt_hierarchy_depth_intra_slice_chroma != 0) {	
sps_log2_diff_max_bt_min_qt_intra_slice_chroma	ue(v)
sps_log2_diff_max_tt_min_qt_intra_slice_chroma	ue(v)
}	
}	
sps_max_luma_transform_size_64_flag	u(1)
if(ChromaArrayType != 0) {	
sps_joint_cbr_enabled_flag	u(1)
same_qp_table_for_chroma	u(1)
numQpTables = same_qp_table_for_chroma ? 1 : (sps_joint_cbr_enabled_flag ? 3 : 2)	
for(i = 0; i < numQpTables; i++) {	
qp_table_start_minus26[i]	se(v)
num_points_in_qp_table_minus1[i]	ue(v)
for(j = 0; j <= num_points_in_qp_table_minus1[i]; j++) {	
delta_qp_in_val_minus1[i][j]	ue(v)
delta_qp_diff_val[i][j]	ue(v)
}	
}	

}	
sps_sao_enabled_flag	u(1)
sps_alf_enabled_flag	u(1)
if(sps_alf_enabled_flag && ChromaArrayType != 0)	
sps_ccalf_enabled_flag	u(1)
sps_transform_skip_enabled_flag	u(1)
if(sps_transform_skip_enabled_flag) {	
log2_transform_skip_max_size_minus2	ue(v)
sps_bdpcm_enabled_flag	u(1)
}	
sps_ref_wraparound_enabled_flag	u(1)
sps_temporal_mvp_enabled_flag	u(1)
if(sps_temporal_mvp_enabled_flag)	
sps_sbtmvp_enabled_flag	u(1)
sps_amvr_enabled_flag	u(1)
sps_bdof_enabled_flag	u(1)
if(sps_bdof_enabled_flag)	
sps_bdof_pic_present_flag	u(1)
sps_smvd_enabled_flag	u(1)
sps_dmvr_enabled_flag	u(1)
if(sps_dmvr_enabled_flag)	
sps_dmvr_pic_present_flag	u(1)
sps_mmvd_enabled_flag	u(1)
sps_isp_enabled_flag	u(1)
sps_mrl_enabled_flag	u(1)
sps_mip_enabled_flag	u(1)

if(ChromaArrayType != 0)	
sps_cclm_enabled_flag	u(1)
if(chroma_format_idc == 1) {	
sps_chroma_horizontal_collocated_flag	u(1)
sps_chroma_vertical_collocated_flag	u(1)
}	
sps_mts_enabled_flag	u(1)
if(sps_mts_enabled_flag) {	
sps_explicit_mts_intra_enabled_flag	u(1)
sps_explicit_mts_inter_enabled_flag	u(1)
}	
six_minus_max_num_merge_cand	ue(v)
sps_sbt_enabled_flag	u(1)
sps_affine_enabled_flag	u(1)
if(sps_affine_enabled_flag) {	
five_minus_max_num_subblock_merge_cand	ue(v)
sps_affine_type_flag	u(1)
if(sps_amvr_enabled_flag)	
sps_affine_amvr_enabled_flag	u(1)
sps_affine_prof_enabled_flag	u(1)
if(sps_affine_prof_enabled_flag)	
sps_prof_pic_present_flag	u(1)
}	
sps_palette_enabled_flag	u(1)
if(ChromaArrayType == 3 && !sps_max_luma_transform_size_64_flag)	

sps_act_enabled_flag	u(1)
if(sps_transform_skip_enabled_flag sps_palette_enabled_flag)	
min_qp_prime_ts_minus4	ue(v)
sps_bcw_enabled_flag	u(1)
sps_ibc_enabled_flag	u(1)
if(sps_ibc_enabled_flag)	
six_minus_max_num_ibc_merge_cand	ue(v)
sps_ciip_enabled_flag	u(1)
if(sps_mmvd_enabled_flag)	
sps_fpel_mmvd_enabled_flag	u(1)
if(MaxNumMergeCand >= 2) {	
sps_gpm_enabled_flag	u(1)
if(sps_gpm_enabled_flag && MaxNumMergeCand >= 3)	
max_num_merge_cand_minus_max_num_gpm_cand	ue(v)
}	
sps_lmcs_enabled_flag	u(1)
sps_lfnst_enabled_flag	u(1)
sps_ladf_enabled_flag	u(1)
if(sps_ladf_enabled_flag) {	
sps_num_ladf_intervals_minus2	u(2)
sps_ladf_lowest_interval_qp_offset	se(v)
for(i = 0; i < sps_num_ladf_intervals_minus2 + 1; i++) {	
sps_ladf_qp_offset[i]	se(v)
sps_ladf_delta_threshold_minus1[i]	ue(v)
}	
}	

log2_parallel_merge_level_minus2	ue(v)
sps_scaling_list_enabled_flag	u(1)
sps_dep_quant_enabled_flag	u(1)
if(!sps_dep_quant_enabled_flag)	
sps_sign_data_hiding_enabled_flag	u(1)
sps_virtual_boundaries_enabled_flag	u(1)
if(sps_virtual_boundaries_enabled_flag) {	
sps_virtual_boundaries_present_flag	u(1)
if(sps_virtual_boundaries_present_flag) {	
sps_num_ver_virtual_boundaries	u(2)
for(i = 0; i < sps_num_ver_virtual_boundaries; i++)	
sps_virtual_boundaries_pos_x[i]	u(13)
sps_num_hor_virtual_boundaries	u(2)
for(i = 0; i < sps_num_hor_virtual_boundaries; i++)	
sps_virtual_boundaries_pos_y[i]	u(13)
}	
}	
if(sps_ptl_dpb_hrd_params_present_flag) {	
sps_general_hrd_params_present_flag	u(1)
if(sps_general_hrd_params_present_flag) {	
general_hrd_parameters()	
if(sps_max_sublayers_minus1 > 0)	
sps_sublayer_cpb_params_present_flag	u(1)
firstSubLayer = sps_sublayer_cpb_params_present_flag ? 0 : sps_max_sublayers_minus1	
ols_hrd_parameters(firstSubLayer, sps_max_sublayers_minus1)	

}	
}	
field_seq_flag	u(1)
vui_parameters_present_flag	u(1)
if(vui_parameters_present_flag)	
vui_parameters() /* Specified in ITU-T H.264 ISO/IEC 23002-7 */	
sps_extension_flag	u(1)
if(sps_extension_flag)	
while(more_rbsp_data())	
sps_extension_data_flag	u(1)
rbp_trailing_bits()	
}	

7.3.2.4 Picture parameter set Rbsp syntax

pic_parameter_set_rbsp() {	Descript or
pps_pic_parameter_set_id	ue(v)
pps_seq_parameter_set_id	u(4)
mixed_nalu_types_in_pic_flag	u(1)
pic_width_in_luma_samples	ue(v)
pic_height_in_luma_samples	ue(v)
pps_conformance_window_flag	u(1)
if(pps_conformance_window_flag) {	
pps_conf_win_left_offset	ue(v)
pps_conf_win_right_offset	ue(v)
pps_conf_win_top_offset	ue(v)

pps_conf_win_bottom_offset	ue(v)
}	
scaling_window_explicit_signalling_flag	u(1)
if(scaling_window_explicit_signalling_flag) {	
scaling_win_left_offset	ue(v)
scaling_win_right_offset	ue(v)
scaling_win_top_offset	ue(v)
scaling_win_bottom_offset	ue(v)
}	
output_flag_present_flag	u(1)
subpic_id_mapping_in_pps_flag	u(1)
if(subpic_id_mapping_in_pps_flag) {	
pps_num_subpics_minus1	ue(v)
pps_subpic_id_len_minus1	ue(v)
for(i = 0; i <= pps_num_subpic_minus1; i++)	
pps_subpic_id[i]	u(v)
}	
no_pic_partition_flag	u(1)
if(!no_pic_partition_flag) {	
pps_log2_ctu_size_minus5	u(2)
num_exp_tile_columns_minus1	ue(v)
num_exp_tile_rows_minus1	ue(v)
for(i = 0; i <= num_exp_tile_columns_minus1; i++)	
tile_column_width_minus1[i]	ue(v)
for(i = 0; i <= num_exp_tile_rows_minus1; i++)	
tile_row_height_minus1[i]	ue(v)

if(NumTilesInPic > 1)	
rect_slice_flag	u(1)
if(rect_slice_flag)	
single_slice_per_subpic_flag	u(1)
if(rect_slice_flag && !single_slice_per_subpic_flag) {	
num_slices_in_pic_minus1	ue(v)
if(num_slices_in_pic_minus1 > 0)	
tile_idx_delta_present_flag	u(1)
for(i = 0; i < num_slices_in_pic_minus1; i++) {	
if(NumTileColumns > 1)	
slice_width_in_tiles_minus1[i]	ue(v)
if(NumTileRows > 1 && (tile_idx_delta_present_flag tileIdx % NumTileColumns == 0))	
slice_height_in_tiles_minus1[i]	ue(v)
if(slice_width_in_tiles_minus1[i] == 0 && slice_height_in_tiles_minus1[i] == 0 && RowHeight[SliceTopLeftTileIdx[i] / NumTileColumns] > 1) {	
num_exp_slices_in_tile[i]	ue(v)
numExpSlicesInTile = num_exp_slices_in_tile[i]	
for(j = 0; j < numExpSlicesInTile; j++)	
exp_slice_height_in_ctus_minus1[j]	ue(v)
i += NumSlicesInTile[i]	
}	
if(tile_idx_delta_present_flag && i < num_slices_in_pic_minus1)	
tile_idx_delta[i]	se(v)
}	

}	
loop_filter_across_tiles_enabled_flag	u(1)
loop_filter_across_slices_enabled_flag	u(1)
}	
cabac_init_present_flag	u(1)
for(i = 0; i < 2; i++)	
num_ref_idx_default_active_minus1[i]	ue(v)
rpl1_idx_present_flag	u(1)
init_qp_minus26	se(v)
cu_qp_delta_enabled_flag	u(1)
pps_chroma_tool_offsets_present_flag	u(1)
if(pps_chroma_tool_offsets_present_flag) {	
pps_cb_qp_offset	se(v)
pps_cr_qp_offset	se(v)
pps_joint_cbr_qp_offset_present_flag	u(1)
if(pps_joint_cbr_qp_offset_present_flag)	
pps_joint_cbr_qp_offset_value	se(v)
pps_slice_chroma_qp_offsets_present_flag	u(1)
pps_cu_chroma_qp_offset_list_enabled_flag	u(1)
}	
if(pps_cu_chroma_qp_offset_list_enabled_flag) {	
chroma_qp_offset_list_len_minus1	ue(v)
for(i = 0; i <= chroma_qp_offset_list_len_minus1; i++) {	
cb_qp_offset_list[i]	se(v)
cr_qp_offset_list[i]	se(v)
if(pps_joint_cbr_qp_offset_present_flag)	

joint_cbr_qp_offset_list[i]	se(v)
}	
}	
pps_weighted_pred_flag	u(1)
pps_weighted_bipred_flag	u(1)
deblocking_filter_control_present_flag	u(1)
if(deblocking_filter_control_present_flag) {	
deblocking_filter_override_enabled_flag	u(1)
pps_deblocking_filter_disabled_flag	u(1)
if(!pps_deblocking_filter_disabled_flag) {	
pps_beta_offset_div2	se(v)
pps_tc_offset_div2	se(v)
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)
}	
}	
rpl_info_in_ph_flag	u(1)
if(deblocking_filter_override_enabled_flag)	
dbf_info_in_ph_flag	u(1)
sao_info_in_ph_flag	u(1)
alf_info_in_ph_flag	u(1)
if((pps_weighted_pred_flag pps_weighted_bipred_flag) && rpl_present_in_ph_flag)	
wp_info_in_ph_flag	u(1)

qp_delta_info_in_ph_flag	u(1)
pps_ref_wraparound_enabled_flag	u(1)
if(pps_ref_wraparound_enabled_flag)	
pps_ref_wraparound_offset	ue(v)
picture_header_extension_present_flag	u(1)
slice_header_extension_present_flag	u(1)
pps_extension_flag	u(1)
if(pps_extension_flag)	
while(more_rbsp_data())	
pps_extension_data_flag	u(1)
rbp_trailing_bits()	
}	

7.3.2.7 Picture header structure syntax

picture_header_structure() {	Descript or
gdr_or_irap_pic_flag	u(1)
if(gdr_or_irap_pic_flag)	
gdr_pic_flag	u(1)
ph_inter_slice_allowed_flag	u(1)
if(ph_inter_slice_allowed_flag)	
ph_intra_slice_allowed_flag	u(1)
non_reference_picture_flag	u(1)
ph_pic_parameter_set_id	ue(v)
ph_pic_order_cnt_lsb	u(v)
if(gdr_or_irap_pic_flag)	

no_output_of_prior_pics_flag	u(1)
if(gdr_pic_flag)	
recovery_poc_cnt	ue(v)
for(i = 0; i < NumExtraPhBits; i++)	
ph_extra_bit[i]	u(1)
if(sps_poc_msb_flag) {	
ph_poc_msb_present_flag	u(1)
if(ph_poc_msb_present_flag)	
poc_msb_val	u(v)
}	
if(sps_alf_enabled_flag && alf_info_in_ph_flag) {	
ph_alf_enabled_flag	u(1)
if(ph_alf_enabled_flag) {	
ph_num_alf_aps_ids_luma	u(3)
for(i = 0; i < ph_num_alf_aps_ids_luma; i++)	
ph_alf_aps_id_luma[i]	u(3)
if(ChromaArrayType != 0)	
ph_alf_chroma_idc	u(2)
if(ph_alf_chroma_idc)	
ph_alf_aps_id_chroma	u(3)
if(sps_ccalf_enabled_flag) {	
ph_cc_alf_cb_enabled_flag	u(1)
if(ph_cc_alf_cb_enabled_flag)	
ph_cc_alf_cb_aps_id	u(3)
ph_cc_alf_cr_enabled_flag	u(1)
if(ph_cc_alf_cr_enabled_flag)	

ph_cc_alf_cr_aps_id	u(3)
}	
}	
}	
if(sps_lmcs_enabled_flag) {	
ph_lmcs_enabled_flag	u(1)
if(ph_lmcs_enabled_flag) {	
ph_lmcs_aps_id	u(2)
if(ChromaArrayType != 0)	
ph_chroma_residual_scale_flag	u(1)
}	
}	
if(sps_scaling_list_enabled_flag) {	
ph_scaling_list_present_flag	u(1)
if(ph_scaling_list_present_flag)	
ph_scaling_list_aps_id	u(3)
}	
if(sps_virtual_boundaries_enabled_flag && !sps_virtual_boundaries_present_flag) {	
ph_virtual_boundaries_present_flag	u(1)
if(ph_virtual_boundaries_present_flag) {	
ph_num_ver_virtual_boundaries	u(2)
for(i = 0; i < ph_num_ver_virtual_boundaries; i++)	
ph_virtual_boundaries_pos_x[i]	u(13)
ph_num_hor_virtual_boundaries	u(2)
for(i = 0; i < ph_num_hor_virtual_boundaries; i++)	

ph_virtual_boundaries_pos_y[i]	u(13)
}	
}	
if(output_flag_present_flag)	
pic_output_flag	u(1)
if(rpl_info_in_ph_flag)	
ref_pic_lists()	
if(partition_constraints_override_enabled_flag)	
partition_constraints_override_flag	u(1)
if(ph_intra_slice_allowed_flag) {	
if(partition_constraints_override_flag) {	
ph_log2_diff_min_qt_min_cb_intra_slice_luma	ue(v)
ph_max_mtt_hierarchy_depth_intra_slice_luma	ue(v)
if(ph_max_mtt_hierarchy_depth_intra_slice_luma != 0) {	
ph_log2_diff_max_bt_min_qt_intra_slice_luma	ue(v)
ph_log2_diff_max_tt_min_qt_intra_slice_luma	ue(v)
}	
if(qtbtt_dual_tree_intra_flag) {	
ph_log2_diff_min_qt_min_cb_intra_slice_chroma	ue(v)
ph_max_mtt_hierarchy_depth_intra_slice_chroma	ue(v)
if(ph_max_mtt_hierarchy_depth_intra_slice_chroma != 0) {	
ph_log2_diff_max_bt_min_qt_intra_slice_chroma	ue(v)
ph_log2_diff_max_tt_min_qt_intra_slice_chroma	ue(v)
}	
}	
}	

if(cu_qp_delta_enabled_flag)	
ph_cu_qp_delta_subdiv_intra_slice	ue(v)
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
ph_cu_chroma_qp_offset_subdiv_intra_slice	ue(v)
}	
if(ph_inter_slice_allowed_flag) {	
if(partition_constraints_override_flag) {	
ph_log2_diff_min_qt_min_cb_inter_slice	ue(v)
ph_max_mtt_hierarchy_depth_inter_slice	ue(v)
if(ph_max_mtt_hierarchy_depth_inter_slice != 0) {	
ph_log2_diff_max_bt_min_qt_inter_slice	ue(v)
ph_log2_diff_max_tt_min_qt_inter_slice	ue(v)
}	
}	
if(cu_qp_delta_enabled_flag)	
ph_cu_qp_delta_subdiv_inter_slice	ue(v)
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
ph_cu_chroma_qp_offset_subdiv_inter_slice	ue(v)
if(sps_temporal_mvp_enabled_flag) {	
ph_temporal_mvp_enabled_flag	u(1)
if(ph_temporal_mvp_enabled_flag && rpl_info_in_ph_flag) {	
ph_collocated_from_l0_flag	u(1)
if((ph_collocated_from_l0_flag && num_ref_entries[0][PicRplsIdx[0]] > 1) (!ph_collocated_from_l0_flag && num_ref_entries[1][PicRplsIdx[1]] > 1))	
ph_collocated_ref_idx	ue(v)

}	
}	
mvd_l1_zero_flag	u(1)
if(sps_fpel_mmvd_enabled_flag)	
ph_fpel_mmvd_enabled_flag	u(1)
if(sps_bdof_pic_present_flag)	
ph_disable_bdof_flag	u(1)
if(sps_dmv_r_pic_present_flag)	
ph_disable_dmv_r_flag	u(1)
if(sps_prof_pic_present_flag)	
ph_disable_prof_flag	u(1)
if((pps_weighted_pred_flag pps_weighted_bipred_flag) && wp_info_in_ph_flag)	
pred_weight_table()	
}	
if(qp_delta_info_in_ph_flag)	
ph_qp_delta	se(v)
if(sps_joint_cbr_enabled_flag)	
ph_joint_cbr_sign_flag	u(1)
if(sps_sao_enabled_flag && sao_info_in_ph_flag) {	
ph_sao_luma_enabled_flag	u(1)
if(ChromaArrayType != 0)	
ph_sao_chroma_enabled_flag	u(1)
}	
if(sps_dep_quant_enabled_flag)	
ph_dep_quant_enabled_flag	u(1)

if(sps_sign_data_hiding_enabled_flag && !ph_dep_quant_enabled_flag)	
pic_sign_data_hiding_enabled_flag	u(1)
if(deblocking_filter_override_enabled_flag && dbf_info_in_ph_flag) {	
ph_deblocking_filter_override_flag	u(1)
if(ph_deblocking_filter_override_flag) {	
ph_deblocking_filter_disabled_flag	u(1)
if(!ph_deblocking_filter_disabled_flag) {	
ph_beta_offset_div2	se(v)
ph_tc_offset_div2	se(v)
ph_cb_beta_offset_div2	se(v)
ph_cb_tc_offset_div2	se(v)
ph_cr_beta_offset_div2	se(v)
ph_cr_tc_offset_div2	se(v)
}	
}	
}	
if(picture_header_extension_present_flag) {	
ph_extension_length	ue(v)
for(i = 0; i < ph_extension_length; i++)	
ph_extension_data_byte[i]	u(8)
}	
}	

7.3.7.1 General slice header syntax

slice_header() {	Descriptor
picture_header_in_slice_header_flag	u(1)
if(picture_header_in_slice_header_flag)	
picture_header_structure()	
if(subpic_info_present_flag)	
slice_subpic_id	u(v)
if((rect_slice_flag && NumSlicesInSubpic[CurrSubpicIdx] > 1) (!rect_slice_flag && NumTilesInPic > 1))	
slice_address	u(v)
for(i = 0; i < NumExtraPhBits; i++)	
sh_extra_bit[i]	u(1)
if(!rect_slice_flag && NumTilesInPic > 1)	
num_tiles_in_slice_minus1	ue(v)
if(ph_inter_slice_allowed_flag)	
slice_type	ue(v)
if(sps_alf_enabled_flag && !alf_info_in_ph_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(sps_ccalf_enabled_flag) {	
slice_cc_alf_cb_enabled_flag	u(1)
if(slice_cc_alf_cb_enabled_flag)	
slice_cc_alf_cb_aps_id	u(3)
slice_cc_alf_cr_enabled_flag	u(1)
if(slice_cc_alf_cr_enabled_flag)	
slice_cc_alf_cr_aps_id	u(3)
}	
}	
}	
if(separate_colour_plane_flag == 1)	
colour_plane_id	u(2)
if(!rpl_info_in_ph_flag && ((nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP) sps_idr_rpl_present_flag))	
ref_pic_lists()	
if((rpl_info_in_ph_flag ((nal_unit_type != IDR_W_RADL && nal_unit_type != IDR_N_LP) sps_idr_rpl_present_flag)) && (slice_type != I && num_ref_entries[0][RplsIdx[0]] > 1) (slice_type == B && num_ref_entries[1][RplsIdx[1]] > 1)) {	
num_ref_idx_active_override_flag	u(1)
if(num_ref_idx_active_override_flag)	
for(i = 0; i < (slice_type == B ? 2 : 1); i++)	
if(num_ref_entries[i][RplsIdx[i]] > 1)	
num_ref_idx_active_minus1[i]	ue(v)
}	

if(slice_type != I) {	
if(cabac_init_present_flag)	
cabac_init_flag	u(1)
if(ph_temporal_mvp_enabled_flag && !rpl_info_in_ph_flag) {	
if(slice_type == B)	
slice_collocated_from_l0_flag	u(1)
if((slice_collocated_from_l0_flag && NumRefIdxActive[0] > 1) (! slice_collocated_from_l0_flag && NumRefIdxActive[1] > 1))	
slice_collocated_ref_idx	ue(v)
}	
if(!wp_info_in_ph_flag && ((pps_weighted_pred_flag && slice_type == P) (pps_weighted_bipred_flag && slice_type == B)))	
pred_weight_table()	
}	
if(!qp_delta_info_in_ph_flag)	
slice_qp_delta	se(v)
if(pps_slice_chroma_qp_offsets_present_flag) {	
slice_cb_qp_offset	se(v)
slice_cr_qp_offset	se(v)
if(sps_joint_cbr_enabled_flag)	
slice_joint_cbr_qp_offset	se(v)
}	
if(pps_cu_chroma_qp_offset_list_enabled_flag)	
cu_chroma_qp_offset_enabled_flag	u(1)
if(sps_sao_enabled_flag && !sao_info_in_ph_flag) {	

slice_sao_luma_flag	u(1)
if(ChromaArrayType != 0)	
slice_sao_chroma_flag	u(1)
}	
if(deblocking_filter_override_enabled_flag && !dbf_info_in_ph_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)
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)
}	
}	
slice_ts_residual_coding_disabled_flag	u(1)
if(ph_lmcs_enabled_flag)	
slice_lmcs_enabled_flag	u(1)
if(pic_scaling_list_enabled_flag)	

slice_scaling_list_present_flag	u(1)
if(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()	
}	

3.4. Color space and chroma subsampling

[0071] Color space, also known as the color model (or color system), is an abstract mathematical model which simply describes the range of colors as tuples of numbers, typically as 3 or 4 values or color components (e.g. RGB). Basically speaking, color space is an elaboration of the coordinate system and sub-space.

[0072] For video compression, the most frequently used color spaces are YCbCr and RGB.

[0073] YCbCr, Y'CbCr, or Y Pb/Cb Pr/Cr, also written as YCBCR or Y'CBCR, is a family of color spaces used as a part of the color image pipeline in video and digital photography systems. Y' is the luma component and CB and CR are the blue-difference and red-difference chroma components. Y' (with prime) is distinguished from Y, which is luminance, meaning that light intensity is nonlinearly encoded based on gamma corrected RGB primaries.

[0074] Chroma subsampling is the practice of encoding images by implementing less resolution for chroma information than for luma information, taking advantage of the human visual system's lower acuity for color differences than for luminance.

3.4.1. 4:4:4

[0075] Each of the three Y'CbCr components have the same sample rate, thus there is no chroma subsampling. This scheme is sometimes used in high-end film scanners and cinematic post production.

3.4.2. 4:2:2

[0076] The two chroma components are sampled at half the sample rate of luma: the horizontal chroma resolution is halved while the vertical chroma resolution is unchanged. This reduces the bandwidth of an uncompressed video signal by one-third with little to no visual difference. An example of nominal vertical and horizontal locations of 4:2:2 color format is depicted in FIG. 5 in VVC working draft.

[0077] FIG. 5 shows a nominal vertical and horizontal locations of 4:2:2 luma and chroma samples in a picture.

3.4.3. 4:2:0

[0078] In 4:2:0, the horizontal sampling is doubled compared to 4:1:1, but as the Cb and Cr channels are only sampled on each alternate line in this scheme, the vertical resolution is halved. The data rate is thus the same. Cb and Cr are each subsampled at a factor of 2 both horizontally and vertically. There are three variants of 4:2:0 schemes, having different horizontal and vertical siting.

- In MPEG-2, Cb and Cr are cosited horizontally. Cb and Cr are sited between pixels in the vertical direction (sited interstitially).
- In JPEG/JFIF, H.261, and MPEG-1, Cb and Cr are sited interstitially, halfway between alternate luma samples.
- In 4:2:0 DV, Cb and Cr are co-sited in the horizontal direction. In the vertical direction, they are co-sited on alternating lines.

Table 3-1. SubWidthC and SubHeightC values derived from chroma_format_idc and separate_colour_plane_flag

•

chroma_format_i dc	separate_colour_plane_flag	Chroma format	SubWidth C	SubHeight C
0	0	Monochrome	1	1
1	0	4:2:0	2	2
2	0	4:2:2	2	1
3	0	4:4:4	1	1
3	1	4:4:4	1	1

3.5. Adaptive Loop Filter (ALF)

[0079] In VVC, an Adaptive Loop Filter (ALF) with block-based filter adaption is applied. For the luma component, one among 25 filters is selected for each 4×4 block, based on the direction and activity of local gradients.

3.5.1. Filter shape

[0080] Two diamond filter shapes (as shown in FIG. 6) are used. The 7×7 diamond shape is applied for luma component and the 5×5 diamond shape is applied for chroma components.

[0081] FIG. 6 shows example of ALF filter shapes (chroma: 5×5 diamond, luma: 7×7 diamond).

3.5.2. Block classification

[0082] For luma component, each 4 × 4 block is categorized into one out of 25 classes. The classification index C is derived based on its directionality D and a quantized value of activity $\hat{A}$, as follows:

$$C = 5D + \hat{A} \quad (3-1)$$

[0083] To calculate D and $\hat{A}$, gradients of the horizontal, vertical and two diagonal direction are first calculated using 1-D Laplacian:

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

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

$$g_{d1} = \sum_{k=i-2}^{i+3} \sum_{l=j-3}^{j+3} D1_{k,l}, D1_{k,l} = |2R(k, l) - R(k-1, l-1) - R(k+1, l+1)| \quad (3-4)$$

$$g_{d2} = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} D2_{k,l}, D2_{k,l} = |2R(k, l) - R(k-1, l+1) - R(k+1, l-1)| \quad (3-5)$$

[0084] Where indices i and j refer to the coordinates of the upper left sample within the 4×4 block and $R(i, j)$ indicates a reconstructed sample at coordinate (i, j) .

[0085] To reduce the complexity of block classification, the subsampled 1-D Laplacian calculation is applied. As shown in FIG. 7A-7D, same subsampled positions are used for gradient calculation of all directions.

[0086] FIGS. 7A-7D show subsampled Laplacian calculation

[0087] Then D maximum and minimum values of the gradients of horizontal and vertical directions are set as:

$$g_{h,v}^{max} = \max(g_h, g_v), g_{h,v}^{min} = \min(g_h, g_v) \quad (3-6)$$

[0088] The maximum and minimum values of the gradient of two diagonal directions are set as:

$$g_{d0,d1}^{max} = \max(g_{d0}, g_{d1}), g_{d0,d1}^{min} = \min(g_{d0}, g_{d1}) \quad (3-7)$$

[0089] To derive the value of the directionality D , these values are compared against each other and with two thresholds t_1 and t_2 :

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

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

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

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

[0090] The activity value A is calculated as:

$$A = \sum_{k=i-2}^{i+3} \sum_{l=j-2}^{j+3} (V_{k,l} + H_{k,l}) \quad (2-8)$$

[0091] A is further quantized to the range of 0 to 4, inclusively, and the quantized value is denoted as $\hat{A}$.

[0092] For chroma components in a picture, no classification method is applied, i.e. a single set of ALF coefficients is applied for each chroma component.

3.5.3. Geometric transformations of filter coefficients and clipping values

[0093] Before filtering each 4×4 luma block, geometric transformations such as rotation or diagonal and vertical flipping are applied to the filter coefficients $f(k, l)$ and to the corresponding filter clipping values $c(k, l)$ depending on gradient values calculated for that block. This is equivalent to applying these transformations to the samples in the filter support region. The idea is to make different blocks to which ALF is applied more similar by aligning their directionality.

[0094] Three geometric transformations, including diagonal, vertical flip and rotation are introduced:

$$\text{Diagonal: } f_D(k, l) = f(l, k), c_D(k, l) = c(l, k), \quad (3-9)$$

$$\text{Vertical flip: } f_V(k, l) = f(k, K - l - 1), c_V(k, l) = c(k, K - l - 1) \quad (3-10)$$

$$\text{Rotation: } f_R(k, l) = f(K - l - 1, k), c_R(k, l) = c(K - l - 1, k) \quad (3-11)$$

[0095] where K is the size of the filter and $0 \leq k, l \leq K - 1$ are coefficients coordinates, such that location $(0, 0)$ is at the upper left corner and location $(K - 1, K - 1)$ is at the lower right corner. The transformations are applied to the filter coefficients $f(k, l)$ and to the clipping values $c(k, l)$ depending on gradient values calculated for that block. The relationship between the transformation and the four gradients of the four directions are summarized in the following table.

Table 3-2. Mapping of the gradient calculated for one block and the transformations

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

3.5.4. Filter parameters signalling

[0096] ALF filter parameters are signalled in Adaptation Parameter Set (APS). In one APS, up to 25 sets of luma filter coefficients and clipping value indexes, and up to eight sets of chroma filter coefficients and clipping value indexes could be signalled. To reduce bits overhead, filter

coefficients of different classification for luma component can be merged. In slice header, the indices of the APSs used for the current slice are signaled.

[0097] Clipping value indexes, which are decoded from the APS, allow determining clipping values using a table of clipping values for both luma and Chroma components. These clipping values are dependent of the internal bitdepth. More precisely, the clipping values are obtained by the following formula:

$$\text{AlfClip} = \{\text{round}(2^{B-\alpha \cdot n}) \text{ for } n \in [0..N-1]\} \quad (3-12)$$

[0098] with B equal to the internal bitdepth, α is a pre-defined constant value equal to 2.35, and N equal to 4 which is the number of allowed clipping values in VVC.

[0099] In slice header, up to 7 APS indices can be signaled to specify the luma filter sets that are used for the current slice. The filtering process can be further controlled at CTB level. A flag is always signalled to indicate whether ALF is applied to a luma CTB. A luma CTB can choose a filter set among 16 fixed filter sets and the filter sets from APSs. A filter set index is signaled for a luma CTB to indicate which filter set is applied. The 16 fixed filter sets are pre-defined and hard-coded in both the encoder and the decoder.

[00100] For chroma component, an APS index is signaled in slice header to indicate the chroma filter sets being used for the current slice. At CTB level, a filter index is signaled for each chroma CTB if there is more than one chroma filter set in the APS.

[00101] The filter coefficients are quantized with norm equal to 128. In order to restrict the multiplication complexity, a bitstream conformance is applied so that the coefficient value of the non-central position shall be in the range of -2^7 to $2^7 - 1$, inclusive. The central position coefficient is not signalled in the bitstream and is considered as equal to 128.

3.5.5. Filtering process

[00102] At decoder side, when ALF is enabled for a CTB, each sample $R(i, j)$ within the CU is filtered, resulting in sample value $R'(i, j)$ as shown below,

$$R'(i, j) = R(i, j) + ((\sum_{k \neq 0} \sum_{l \neq 0} f(k, l) \times K(R(i+k, j+l) - R(i, j), c(k, l)) + 64) \gg 7) \quad (3-13)$$

[00103] where $f(k, l)$ denotes the decoded filter coefficients, $K(x, y)$ is the clipping function and $c(k, l)$ denotes the decoded clipping parameters. The variable k and l varies between $-\frac{L}{2}$ and

$\frac{L}{2}$ where L denotes the filter length. The clipping function $K(x, y) = \min(y, \max(-y, x))$ which corresponds to the function $Clip3(-y, y, x)$.

3.5.6. Virtual boundary filtering process for line buffer reduction

[00104] In hardware and embedded software, picture-based processing is practically unacceptable due to its high picture buffer requirement. Using on-chip picture buffers is very expensive and using off-chip picture buffers significantly increases external memory access, power consumption, and data access latency. Therefore, DF, SAO, and ALF will be changed from picture-based to LCU-based decoding in real products. When LCU-based processing is used for DF, SAO, and ALF, the entire decoding process can be done LCU by LCU in a raster scan with an LCU-pipelining fashion for parallel processing of multiple LCUs. In this case, line buffers are required for DF, SAO, and ALF because processing one LCU row requires pixels from the above LCU row. If off-chip line buffers (e.g. DRAM) are used, the external memory bandwidth and power consumption will be increased; if on-chip line buffers (e.g. SRAM) are used, the chip area will be increased. Therefore, although line buffers are already much smaller than picture buffers, it is still desirable to reduce line buffers.

[00105] In VTM-4.0, as shown in FIG. 8, the total number of line buffers required is 11.25 lines for the Luma component. The explanation of the line buffer requirement is as follows: The deblocking of horizontal edge overlapping with CTU edge cannot be performed as the decisions and filtering require lines K, L, M, M from the first CTU and Lines O, P from the bottom CTU. Therefore, the deblocking of the horizontal edges overlapping with the CTU boundary is postponed until the lower CTU comes. Therefore, for the lines K, L, M, N reconstructed luma samples have to be stored in the line buffer (4 lines). Then the SAO filtering can be performed for lines A till J. The line J can be SAO filtered as deblocking does not change the samples in line K. For SAO filtering of line K, the edge offset classification decision is only stored in the line buffer (which is 0.25 Luma lines). The ALF filtering can only be performed for lines A-F. As shown in FIG. 8, the ALF classification is performed for each 4×4 block. Each 4×4 block classification needs an activity window of size 8×8 which in turn needs a 9×9 window to compute the 1d Laplacian to determine the gradient.

[00106] Therefore, for the block classification of the 4×4 block overlapping with lines G, H, I, J needs, SAO filtered samples below the Virtual boundary. In addition, the SAO filtered samples

of lines D, E, F are required for ALF classification. Moreover, the ALF filtering of Line G needs three SAO filtered lines D, E, F from above lines. Therefore, the total line buffer requirement is as follows:

- Lines K-N (Horizontal DF pixels): 4 lines
- Lines D-J (SAO filtered pixels): 7 lines
- SAO Edge offset classifier values between line J and line K: 0.25 line

[00107] Therefore, the total number of luma lines required is $7 + 4 + 0.25 = 11.25$.

[00108] Similarly, the line buffer requirement of the Chroma component is illustrated in FIG. 9. The line buffer requirement for Chroma component is evaluated to be 6.25 lines.

[00109] FIG. 8 shows an example of a loop filter line buffer requirement in VTM-4.0 for Luma component.

[00110] FIG. 9 shows an example of a loop filter line buffer requirement in VTM-4.0 for Chroma component.

[00111] In order to eliminate the line buffer requirements of SAO and ALF, the concept of virtual boundary (VB) is introduced to reduce the line buffer requirement of ALF in the latest VVC. Modified block classification and filtering are employed for the samples near horizontal CTU boundaries. As shown in FIG. 8, VBs are upward shifted horizontal LCU boundaries by N pixels. For each LCU, SAO and ALF can process pixels above the VB before the lower LCU comes but cannot process pixels below the VB until the lower LCU comes, which is caused by DF. With consideration of the hardware implementation cost, the space between the proposed VB and the horizontal LCU boundary is set as four pixels for luma component (i.e. $N=4$ in FIG. 8 or FIG. 10) and two pixels for chroma component (i.e. $N=2$).

[00112] FIG. 10 shows a modified block classification at virtual boundaries.

[00113] Modified block classification is applied for the Luma component as depicted in FIG. 11. For the 1D Laplacian gradient calculation of the 4×4 block above the virtual boundary, only the samples above the virtual boundary are used. Similarly, for the 1D Laplacian gradient calculation of the 4×4 block below the virtual boundary, only the samples below the virtual boundary are used. The quantization of activity value A is accordingly scaled by taking into account the reduced number of samples used in 1D Laplacian gradient calculation.

[00114] For filtering processing, mirrored (symmetric) padding operation at the virtual boundaries are used for both Luma and Chroma components. As shown in FIG. 11, when the sample being filtered is located below the virtual boundary, the neighboring samples that are located above the virtual boundary are padded. Meanwhile, the corresponding samples at the other sides are also padded, symmetrically.

[00115] For another example, if one sample located at (i, j) (e.g., the P0A with dash line in FIG. 12B) is padded, then the corresponding sample located at (m, n) (e.g., the P3B with dash line in FIG. 12B) which share the same filter coefficient is also padded even the sample is available, as depicted in FIGS. 12A-12C.

[00116] FIG. 12A shows one required line above/below VB need to be padded (per side).

[00117] FIG. 12B shows 2 required lines above/below VB need to be padded (per side)

[00118] FIG. 12C shows 3 required lines above/below VB need to be padded (per side)

[00119] Different to the mirrored (symmetric) padding method used at horizontal CTU boundaries, repetitive (one-side) padding process is applied for slice, tile and subpicture boundaries when filter across the boundaries is disabled. The repetitive (one-side) padding process is also applied at picture boundary. The padded samples are used for both classification and filtering process. FIG. 13 depicts an example of repetitive padding method for luma ALF filtering at picture/subpicture/slice/tile boundary.

3.5.7. Adaptive loop filter process in spec

8.8.5.2 Coding tree block filtering process for luma samples

[00120] Inputs of this process are:

- a reconstructed luma picture sample array `recPicture` prior to the adaptive loop filtering process,
- a filtered reconstructed luma picture sample array `alfPictureL`,
- a luma location (`xCtb`, `yCtb`) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture.

[00121] Output of this process is the modified filtered reconstructed luma picture sample array `alfPictureL`.

[00122] The derivation process for filter index clause 8.8.5.3 is invoked with the location (`xCtb`, `yCtb`) and the reconstructed luma picture sample array `recPicture` as inputs, and `filterIdx[ x ][ y ]` and `transposeIdx[ x ][ y ]` with $x, y = 0..CtbSizeY - 1$ as outputs.

[00123] For the derivation of the filtered reconstructed luma samples $\text{alfPictureL}[x][y]$, each reconstructed luma sample inside the current luma coding tree block $\text{recPicture}[x][y]$ is filtered as follows with $x, y = 0..CtbSizeY - 1$:

- The array of luma filter coefficients $f[j]$ and the array of luma clipping values $c[j]$ corresponding to the filter specified by $\text{filtIdx}[x][y]$ is derived as follows with $j = 0..11$:
- If $\text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY]$ is less than 16, the following applies:

$$i = \text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY] \quad (1453)$$

$$f[j] = \text{AlfFixFiltCoeff}[\text{AlfClassToFiltMap}[i][\text{filtIdx}[x][y]]][j] \quad (1454)$$

$$c[j] = 2^{\text{BitDepth}} \quad (1455)$$

- Otherwise $(\text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY])$ is greater than or equal to 16, the following applies:

$$i = \text{slice_alf_aps_id_luma}[\text{AlfCtbFiltSetIdxY}[xCtb \gg CtbLog2SizeY][yCtb \gg CtbLog2SizeY] - 16] \quad (1456)$$

$$f[j] = \text{AlfCoeffL}[i][\text{filtIdx}[x][y]][j] \quad (1457)$$

$$c[j] = \text{AlfClipL}[i][\text{filtIdx}[x][y]][j] \quad (1458)$$

- The luma filter coefficients and clipping values index idx are derived depending on $\text{transposeIdx}[x][y]$ as follows:

- If $\text{transposeIndex}[x][y]$ is equal to 1, the following applies:

$$\text{idx}[] = \{ 9, 4, 10, 8, 1, 5, 11, 7, 3, 0, 2, 6 \} \quad (1459)$$

- Otherwise, if $\text{transposeIndex}[x][y]$ is equal to 2, the following applies:

$$\text{idx}[] = \{ 0, 3, 2, 1, 8, 7, 6, 5, 4, 9, 10, 11 \}$$

(1460)

- Otherwise, if $\text{transposeIndex}[x][y]$ is equal to 3, the following applies:

$$\text{idx}[] = \{ 9, 8, 10, 4, 3, 7, 11, 5, 1, 0, 2, 6 \}$$

(1461)

- Otherwise, the following applies:

$$\text{idx}[] = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 \}$$

(1462)

- The locations (h_{x+i}, v_{y+j}) for each of the corresponding luma samples (x, y) inside the given array recPicture of luma samples with $i, j = -3..3$ are derived as follows:

$$h_{x+i} = \text{Clip3}(0, \text{pic_width_in_luma_samples} - 1, x_{\text{Ctb}} + x + i) \quad (1463)$$

$$v_{y+j} = \text{Clip3}(0, \text{pic_height_in_luma_samples} - 1, y_{\text{Ctb}} + y + j) \quad (1464)$$

- The variables clipLeftPos , clipRightPos , clipTopPos , clipBottomPos , clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with $(x_{\text{Ctb}}, y_{\text{Ctb}})$ and (x, y) as inputs.
- The variables h_{x+i} and v_{y+j} are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with $(x_{\text{Ctb}}, y_{\text{Ctb}})$, (h_{x+i}, v_{y+j}) , 0, clipLeftPos , clipRightPos , clipTopPos , clipBottomPos , clipTopLeftFlag and clipBotRightFlag as input.
- The variable $\text{applyAlfLineBufBoundary}$ is derived as follows:
 - If the bottom boundary of the current coding tree block is the bottom boundary of current picture and $\text{pic_height_in_luma_samples} - y_{\text{Ctb}} \leq \text{CtbSizeY} - 4$, $\text{applyAlfLineBufBoundary}$ is set equal to 0;
 - Otherwise, $\text{applyAlfLineBufBoundary}$ is set equal to 1.
- The vertical sample position offsets y_1 , y_2 , y_3 and the variable alfShiftY are specified in Table 45 according to the vertical luma sample position y and $\text{applyAlfLineBufBoundary}$.

- The variable curr is derived as follows:

$$\text{curr} = \text{recPicture}[h_x][v_y] \quad (1465)$$

- The variable sum is derived as follows:

$$\begin{aligned} \text{sum} = & f[\text{idx}[0]] * (\text{Clip3}(-c[\text{idx}[0]], c[\text{idx}[0]], \text{recPicture}[h_x][v_y+y_3] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[0]], c[\text{idx}[0]], \text{recPicture}[h_x][v_y-y_3] - \text{curr})) + \\ & f[\text{idx}[1]] * (\text{Clip3}(-c[\text{idx}[1]], c[\text{idx}[1]], \text{recPicture}[h_{x+1}][v_y+y_2] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[1]], c[\text{idx}[1]], \text{recPicture}[h_{x-1}][v_y-y_2] - \text{curr})) + \\ & f[\text{idx}[2]] * (\text{Clip3}(-c[\text{idx}[2]], c[\text{idx}[2]], \text{recPicture}[h_x][v_y+y_2] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[2]], c[\text{idx}[2]], \text{recPicture}[h_x][v_y-y_2] - \text{curr})) + \\ & f[\text{idx}[3]] * (\text{Clip3}(-c[\text{idx}[3]], c[\text{idx}[3]], \text{recPicture}[h_{x-1}][v_y+y_2] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[3]], c[\text{idx}[3]], \text{recPicture}[h_{x+1}][v_y-y_2] - \text{curr})) + \\ & f[\text{idx}[4]] * (\text{Clip3}(-c[\text{idx}[4]], c[\text{idx}[4]], \text{recPicture}[h_{x+2}][v_y+y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[4]], c[\text{idx}[4]], \text{recPicture}[h_{x-2}][v_y-y_1] - \text{curr})) + \\ & f[\text{idx}[5]] * (\text{Clip3}(-c[\text{idx}[5]], c[\text{idx}[5]], \text{recPicture}[h_{x+1}][v_y+y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[5]], c[\text{idx}[5]], \text{recPicture}[h_{x-1}][v_y-y_1] - \text{curr})) + \\ & f[\text{idx}[6]] * (\text{Clip3}(-c[\text{idx}[6]], c[\text{idx}[6]], \text{recPicture}[h_x][v_y+y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[6]], c[\text{idx}[6]], \text{recPicture}[h_x][v_y-y_1] - \text{curr})) + (1466) \\ & f[\text{idx}[7]] * (\text{Clip3}(-c[\text{idx}[7]], c[\text{idx}[7]], \text{recPicture}[h_{x-1}][v_y+y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[7]], c[\text{idx}[7]], \text{recPicture}[h_{x+1}][v_y-y_1] - \text{curr})) + \\ & f[\text{idx}[8]] * (\text{Clip3}(-c[\text{idx}[8]], c[\text{idx}[8]], \text{recPicture}[h_{x-2}][v_y+y_1] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[8]], c[\text{idx}[8]], \text{recPicture}[h_{x+2}][v_y-y_1] - \text{curr})) + \\ & f[\text{idx}[9]] * (\text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[h_{x+3}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[9]], c[\text{idx}[9]], \text{recPicture}[h_{x-3}][v_y] - \text{curr})) + \\ & f[\text{idx}[10]] * (\text{Clip3}(-c[\text{idx}[10]], c[\text{idx}[10]], \text{recPicture}[h_{x+2}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[10]], c[\text{idx}[10]], \text{recPicture}[h_{x-2}][v_y] - \text{curr})) + \\ & f[\text{idx}[11]] * (\text{Clip3}(-c[\text{idx}[11]], c[\text{idx}[11]], \text{recPicture}[h_{x+1}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[\text{idx}[11]], c[\text{idx}[11]], \text{recPicture}[h_{x-1}][v_y] - \text{curr})) \end{aligned}$$

$$\text{sum} = \text{curr} + ((\text{sum} + 64) >> \text{alfShiftY})$$

(1467)

- The modified filtered reconstructed luma picture sample $\text{alfPicture}_L[x\text{Ctb} + x][y\text{Ctb} + y]$ is derived as follows:

$$\text{alfPicture}_L[x\text{Ctb} + x][y\text{Ctb} + y] = \text{Clip3}(0, (1 << \text{BitDepth}) - 1, \text{sum}) \quad (1468)$$

Table 45 – Specification of $y1$, $y2$, $y3$ and $alfShiftY$ according to the vertical luma sample position y and $applyAlfLineBufBoundary$

Condition	$alfShiftY$	$y1$	$y2$	$y3$
$(y == CtbSizeY - 5 \parallel y == CtbSizeY - 4) \&\&$ $(applyAlfLineBufBoundary == 1)$	10	0	0	0
$(y == CtbSizeY - 6 \parallel y == CtbSizeY - 3) \&\&$ $(applyAlfLineBufBoundary == 1)$	7	1	1	1
$(y == CtbSizeY - 7 \parallel y == CtbSizeY - 2) \&\&$ $(applyAlfLineBufBoundary == 1)$	7	1	2	2
otherwise	7	1	2	3

8.8.5.4 Coding tree block filtering process for chroma samples

Inputs of this process are:

- a reconstructed chroma picture sample array $recPicture$ prior to the adaptive loop filtering process,
- a filtered reconstructed chroma picture sample array $alfPicture$,
- a chroma location $(xCtbC, yCtbC)$ specifying the top-left sample of the current chroma coding tree block relative to the top left sample of the current picture,
- an alternative chroma filter index $altIdx$.

Output of this process is the modified filtered reconstructed chroma picture sample array $alfPicture$.

The width and height of the current chroma coding tree block $ctbWidthC$ and $ctbHeightC$ is derived as follows:

$$ctbWidthC = CtbSizeY / SubWidthC \quad (1500)$$

$$\text{ctbHeightC} = \text{CtbSizeY} / \text{SubHeightC} \quad (1501)$$

For the derivation of the filtered reconstructed chroma samples $\text{alfPicture}[x][y]$, each reconstructed chroma sample inside the current chroma coding tree block $\text{recPicture}[x][y]$ is filtered as follows with $x = 0..\text{ctbWidthC} - 1$, $y = 0..\text{ctbHeightC} - 1$:

- The locations (h_{x+i}, v_{y+j}) for each of the corresponding chroma samples (x, y) inside the given array recPicture of chroma samples with $i, j = -2..2$ are derived as follows:

$$h_{x+i} = \text{Clip3}(0, \text{pic_width_in_luma_samples} / \text{SubWidthC} - 1, x\text{CtbC} + x + i) \quad (1502)$$

$$v_{y+j} = \text{Clip3}(0, \text{pic_height_in_luma_samples} / \text{SubHeightC} - 1, y\text{CtbC} + y + j) \quad (1503)$$

- The variables clipLeftPos , clipRightPos , clipTopPos , clipBottomPos , clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with $(x\text{CtbC} * \text{SubWidthC}, y\text{CtbC} * \text{SubHeightC})$ and $(x * \text{SubWidthC}, y * \text{SubHeightC})$ as inputs.
- The variables h_{x+i} and v_{y+j} are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with $(x\text{CtbC}, y\text{CtbC})$, (h_{x+i}, v_{y+j}) , the variable isChroma set equal to 1, clipLeftPos , clipRightPos , clipTopPos , clipBottomPos , clipTopLeftFlag and clipBotRightFlag as input.
- The variable $\text{applyAlfLineBufBoundary}$ is derived as follows:
 - If the bottom boundary of the current coding tree block is the bottom boundary of the picture and $\text{pic_height_in_luma_samples} - (y\text{CtbC} * \text{SubHeightC}) < \text{CtbSizeY} - 4$, $\text{applyAlfLineBufBoundary}$ is set equal to 0.
 - Otherwise, $\text{applyAlfLineBufBoundary}$ is set equal to 1.
- The vertical sample position offsets $y1, y2$ and the variable alfShiftC are specified in Table 45 according to the vertical chroma sample position y and $\text{applyAlfLineBufBoundary}$.
- The variable curr is derived as follows:

$$\text{curr} = \text{recPicture}[h_x][v_y] \quad (1504)$$

- The array of chroma filter coefficients $f[j]$ and the array of chroma clipping values $c[j]$ is derived as follows with $j = 0..5$:

$$f[j] = \text{AlfCoeffc}[\text{slice_alf_aps_id_chroma}][[\text{altIdx}][j]] \quad (1505)$$

$$c[j] = \text{AlfClipc}[\text{slice_alf_aps_id_chroma}][[\text{altIdx}][j]] \quad (1506)$$

- The variable sum is derived as follows:

$$\begin{aligned} \text{sum} = & f[0] * (\text{Clip3}(-c[0], c[0], \text{recPicture}[h_x][v_y + y_2] - \text{curr}) + \\ & \text{Clip3}(-c[0], c[0], \text{recPicture}[h_x][v_y - y_2] - \text{curr})) + \\ & f[1] * (\text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x+1}][v_y + y_1] - \text{curr}) + \\ & \text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x-1}][v_y - y_1] - \text{curr})) + \\ & f[2] * (\text{Clip3}(-c[2], c[2], \text{recPicture}[h_x][v_y + y_1] - \text{curr}) + \\ & \text{Clip3}(-c[2], c[2], \text{recPicture}[h_x][v_y - y_1] - \text{curr})) + \\ & f[3] * (\text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x-1}][v_y + y_1] - \text{curr}) + \\ & \text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x+1}][v_y - y_1] - \text{curr})) + \\ & f[4] * (\text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x+2}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x-2}][v_y] - \text{curr})) + \\ & f[5] * (\text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x+1}][v_y] - \text{curr}) + \\ & \text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x-1}][v_y] - \text{curr})) \end{aligned} \quad (1507)$$

$$\text{sum} = \text{curr} + ((\text{sum} + 64) \gg \text{alfShiftC}) \quad (1508)$$

- The modified filtered reconstructed chroma picture sample $\text{alfPicture}[x\text{CtbC} + x][y\text{CtbC} + y]$ is derived as follows:

$$\text{alfPicture}[x\text{CtbC} + x][y\text{CtbC} + y] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum}) \quad (1509)$$

Table 46 – Specification of y_1 , y_2 and alfShiftC according to the vertical chroma sample position y and $\text{applyAlfLineBufBoundary}$

Condition	alfShiftC	y1	y2
(y == ctbHeightC - 2 y == ctbHeightC - 3) && (applyAlfLineBufBoundary == 1)	10	0	0
(y == ctbHeightC - 1 y == ctbHeightC - 4) && (applyAlfLineBufBoundary == 1)	7	1	1
Otherwise	7	1	2

4. Examples of technical problems addressed by disclosed solutions

[00124] The existing designs for signalling of SPS/PPS/Picture header/Slice header in VVC have the following problems:

- 1) loop_filter_across_tiles_enabled_flag is signaled even if there is only one tile.
- 2) loop_filter_across_slices_enabled_flag is signaled even if there is only one slice.
- 3) LMCS information is signaled in picture header, not in slice header.
- 4) The maximum allowed TS block size may be larger than the maximum CU size.
- 5) MaxNumSubblockMergeCand is derived in different ways depending on affine is enabled or not.
- 6) Mirrored padding is applied at ALF virtual boundary to get the unavailable samples and their corresponding samples for ALF (e.g., luma ALF and chroma ALF), wherein the position of ALF virtual boundary is utilized to determine which samples are unavailable and need to be padded. However, in the current design, the position of ALF virtual boundary in chroma ALF is not aligned with that in luma ALF for 4:2:2/4:4:4 chroma format video.
- 7) In the current design, the filter strength is reduced for ALF filtering of the rows adjacent to the virtual horizontal CTU boundary, which is dependent on the position of virtual boundary. However, the incorrect position of ALF virtual boundary makes filter strength reduced at unintentional sample rows.

5. Examples of techniques and embodiments

[00125] To solve the above problems, and others, methods as summarized below are disclosed. The listed items should be considered as examples to explain the general concepts and should not be interpreted in a narrow way. Furthermore, these items can be applied individually or combined in any manner.

[00126] In the disclosure, a neighbouring (adjacent or non-adjacent) sample (or line, or row) is “unavailable” if it is located in a different video processing unit (e.g., out of the current picture, or current sub-picture, or current tile, or current slice, or current brick, or current CTU, or current processing unit (such as ALF processing unit or narrow ALF processing unit), or any other current video unit) or not reconstructed or cross-filtering video processing unit is disallowed.

[00127] The padding method used for ALF virtual boundaries may be denoted as ‘Mirrored Padding’ wherein for a first unavailable sample located at (i, j) (or a first unavailable line j, or a first unavailable row i), it needs to be padded, and a second sample, defined by ‘corresponding sample of the first sample’ (or ‘corresponding line of the first line’, or corresponding row of the first row’) in the filter support (e.g., the corresponding sample located at (m, n) (or the corresponding line n, or the corresponding row m) which share the same distance from the current sample (or current line, or current row)) in ALF is also padded even the second sample is available.

1. Signaling of indication of whether in-loop filtering operations is performed across tile boundaries (e.g., **loop_filter_across_tiles_enabled_flag**) may be conditioned on whether and/or how a video unit (e.g., a picture) is partitioned into tiles.
 - a. In one example, **loop_filter_across_tiles_enabled_flag** is signaled only if the video unit is partitioned into more than one tile.
 - b. Alternatively, signaling of **loop_filter_across_tiles_enabled_flag** is skipped when there is only one tile for the video unit.
 - c. In one example, **loop_filter_across_tiles_enabled_flag** is signaled only if the general constraint flag **one_tile_per_pic_constraint_flag** is equal to 0.
 - i. Alternatively, if the general constraint flag **one_tile_per_pic_constraint_flag** is equal to 1, **loop_filter_across_tiles_enabled_flag** is referred (or, required) to be equal to 0.

2. Signaling of indication of whether in-loop filtering operations is performed across slice boundaries (e.g., **loop_filter_across_slices_enabled_flag**) may be conditioned on whether and/or how a video unit (e.g., a picture and/or a subpicture) is partitioned into slices.
 - a. In one example, **loop_filter_across_slices_enabled_flag** is not signaled if the video unit is partitioned into only one slice.
 - b. In one example, **loop_filter_across_slices_enabled_flag** is not signaled if each subpicture is partitioned into only one slice (e.g. **single_slice_per_subpic_flag** is equal to 1).
 - c. In one example, **loop_filter_across_slices_enabled_flag** is signaled if each subpicture is partitioned into only one slice (e.g. **single_slice_per_subpic_flag** is equal to 1).
 - d. In one example, **loop_filter_across_slices_enabled_flag** is signaled if a picture is partitioned into slices by a non-rectangular way (e.g. **rect_slice_flag** is equal to 0).
 - e. In one example, **loop_filter_across_slices_enabled_flag** is not signaled if a picture is partitioned into slices by a rectangular way (e.g. **rect_slice_flag** is equal to 0) and the number of slices is equal to 1 (e.g. **num_slices_in_pic_minus1** is equal to 0).
 - f. In one example, **loop_filter_across_slices_enabled_flag** is signaled only if the general constraint flag **one_slice_per_pic_constraint_flag** is equal to 0.
 - i. Alternatively, if the general constraint flag **one_slice_per_pic_constraint_flag** is equal to 1, **loop_filter_across_slices_enabled_flag** is referred (or, required) to be equal to 0.
3. LMCS information (e.g., indications of usage of LMCS, and/or usage of luma reshaping, and/or the adaptation_parameter_set_id of the LMCS APS to be used and/or usage of chroma residual scaling) may be signaled in the slice header and/or the picture header.
 - a. LMCS information may comprise a first indication of whether to enable LMCS or not, such as **ph_lmcs_enabled_flag** or **slice_lmcs_enabled_flag**.
 - b. LMCS information may comprise a second indication of LMCS parameters, such as **ph_lmcs_aps_id** and/or **ph_chroma_residual_scale_flag**.
 - c. LMCS information may be signaled in both the slice header and the picture header.

- d. LMCS information may be signalled in both the picture header and slice header, when it is present in the slice header, the picture header's, if there is, may be overwritten.
 - e. LMCS information may be signaled in either the slice header or the picture header but not in both.
 - f. In one example, a syntax element may be signaled in slice header to indicate which LMCS APS is used for the current slice.
 - g. In one example, a syntax element (such as `lmcs_info_in_ph_flag`) is signaled in a higher-level video unit such as in SPS or PPS, to indicate whether LMCS information is signaled in the picture header or the slice header.
 - i. The syntax element may only control the first indication.
 - ii. The syntax element may only control the second indication.
 - iii. The syntax element may control both the first and second indications.
 - h. A syntax element of LMCS information in the picture header may be set to default value if it is not present.
 - i. A syntax element of LMCS information in the slice header may be set to a default value if it is not present.
 - i. A syntax element of LMCS information in the slice header may be set to the value of the corresponding syntax element of LMCS information in the picture header, if it is not present.
 - j. In one example, more than one `adaptation_parameter_set_id` of the LMCS APS that the slices associated with the PH refers to (such as a list of `ph_lmcs_aps_id[]`) may be signalled in the picture header.
 - i. In one example, the length of the list of `ph_lmcs_aps_id[]` may be dependent on the number of slices in the picture.
 - ii. In one example, a syntax element may be signaled in slice header to indicate which LMCS APS in the list `ph_lmcs_aps_id[]` is used for the current slice.
4. The maximum allowed size for Transform Skip (TS) and/or BDPCM shall be smaller than or equal to the Coding Tree Block (CTB) size.
- a. For example, it is required that maximum allowed width and height for TS and/or BDPCM for a luma block shall be smaller than or equal to `CtbSizeY`.

- b. For example, it is required that maximum allowed width and height for TS and/or BDPCM for a chroma block shall be smaller than or equal to $CtbSizeY/subWidthC$ and/or $CtbSizeY/subHeightC$.
 - i. Alternatively, it is required that maximum allowed width for TS and/or BDPCM for a chroma block shall be smaller than or equal to $CtbSizeY/subWidthC$ and/or $CtbSizeY/subHeightC$.
 - ii. Alternatively, it is required that maximum allowed height for TS and/or BDPCM for a chroma block shall be smaller than or equal to $CtbSizeY/subWidthC$ and/or $CtbSizeY/subHeightC$.
- c. For example, it is required that $\log_2_transform_skip_max_size_minus2$ plus 2 shall be smaller than or equal to $CtbLog2SizeY$.
- d. For example, the maximum value of $\log_2_transform_skip_max_size_minus2$ is equal to $CtbLog2SizeY - 2$.
- e. For example, $MaxTsSize$ is derived as

$$MaxTsSize = \text{Min}(CtbSizeY, 1 \ll (\log_2_transform_skip_max_size_minus2 + 2)).$$
- f. For example, $MaxTsSize$ is derived as

$$MaxTsSize = 1 \ll \text{Min}(\log_2_transform_skip_max_size_minus2 + 2, CtbLog2SizeY).$$
- g. The maximum allowed block size for TS and/or BDPCM for a chroma block shall be smaller than or equal to the maximum transform block size for a chroma block.
 - i. In one example, suppose $MaxTbSizeY$ denotes the maximum transform size for a luma block, then maximum allowed width and height for TS and/or BDPCM for a chroma block may be smaller than or equal to $MaxTbSizeY / SubWidthC$.
 - ii. In one example, suppose $MaxTbSizeY$ denotes the maximum transform size for a luma block, then maximum allowed width for TS and/or BDPCM for a chroma block may be smaller than or equal to $MaxTbSizeY / SubWidthC$.
 - iii. In one example, suppose $MaxTbSizeY$ denotes the maximum transform size for a luma block, then maximum allowed height for TS and/or BDPCM

for a chroma block may be smaller than or equal to $\text{MaxTbSizeY} / \text{SubHeightC}$

- h. The maximum allowed Transform Skip (TS) block size may be signalled with a binary syntax element (such as, “0” represents equal to 16 and “1” represents equal to 32).
5. The maximum allowed Transform Skip (TS) block size and/or maximum allowed transform block size shall be no smaller than the minimum coding block size.
6. Whether to and/or how to signal or interpret or restrict the maximum allowed transform block size (e.g. denoted as MaxTbSizeY in JVET-Q2001-vB) may depend on the minimum allowed coding block size (e.g. denoted as MinCbSizeY in JVET-Q2001-vB).
 - a. In one example, it is required that MaxTbSizeY must be larger than or equal to MinCbSizeY .
 - i. In one example, when MinCbSizeY is equal to 64, the value of $\text{sps_max_luma_transform_size_64_flag}$ shall be equal to 1.
 - ii. In one example, when MinCbSizeY is equal to 64, $\text{sps_max_luma_transform_size_64_flag}$ is not signaled and inferred to be 1.
7. Whether to and/or how to signal or interpret or restrict the maximum allowed size of TS and/or BDPCM coding (e.g. denoted as MaxTsSize in JVET-Q2001-vB) may depend on the minimum allowed coding block size (e.g. denoted as MinCbSizeY in JVET-Q2001-vB).
 - a. In one example, it is required that MaxTsSize must be larger than or equal to MinCbSizeY .
 - b. In one example, it is required that MaxTsSize must be smaller than or equal to W , wherein W is an integer such as 32.
 - i. For example, it is required that MaxTsSize must satisfy $\text{MinCbSizeY} \leq \text{MaxTsSize} \leq W$.
 - ii. For example, MinCbSizeY shall be smaller than or equal to X when TS and/or BDPCM coding is enabled.
 - c. In one example, it is required that MaxTsSize must be larger than or equal to MinCbSizeY when TS and/or BDPCM is enabled (e.g. $\text{sps_transform_skip_enabled_flag}$ is equal to 1).

- i. For example, **log2_transform_skip_max_size_minus2** shall be larger than or equal to **log2_min_luma_coding_block_size_minus2** when **sps_transform_skip_enabled_flag** is equal to 1.
- d. In one example, $\text{MaxTsSize} = \max(\text{MaxTsSize}, \text{MinCbSizeY})$.
 - i. In another example, $\text{MaxTsSize} = \min(W, \max(\text{MaxTsSize}, \text{MinCbSizeY}))$ wherein W is an integer such as 32.
- e. In one example, $\text{MaxTsSize} = \max(\text{MaxTsSize}, \text{MinCbSizeY})$ when TS and/or BDPCM is enabled (e.g. **sps_transform_skip_enabled_flag** is equal to 1).
 - i. In one example, $\text{MaxTsSize} = \min(W, \max(\text{MaxTsSize}, \text{MinCbSizeY}))$ wherein W is an integer such as 32, when TS and/or BDPCM is enabled (e.g. **sps_transform_skip_enabled_flag** is equal to 1).
- f. In one example, the signaling of **MaxTsSize** (e.g. **log2_transform_skip_max_size_minus2** in JVET-Q2001-vB) may depend on **MinCbSizeY**.
 - i. In one example, the difference between **log2_transform_skip_max_size_minus2** and **log2_min_luma_coding_block_size_minus2** (denoted as **log2_diff_max_transform_skip_min_coding_block**) may be signaled to indicate **MaxTsSize**.
 - 1) For example, $\text{MaxTsSize} = 1 \ll (\text{MinCbLog2SizeY} + \text{log2_diff_max_transform_skip_min_coding_block})$.
 - 2) For example, $\text{MaxTsSize} = \min(W, 1 \ll (\text{MinCbLog2SizeY} + \text{log2_diff_max_transform_skip_min_coding_block}))$ wherein W is an integer such as 32.
- 8. Whether to and/or how to signal or interpret or restrict the indication of TS and/or BDPCM coding (e.g. denoted as **sps_transform_skip_enabled_flag** in JVET-Q2001-vB) may depend on the minimum allowed coding block size (e.g. denoted as **MinCbSizeY** in JVET-Q2001-vB).
 - a. In one example, when **MinCbSizeY** is equal to 64, **sps_transform_skip_enabled_flag** is not signaled and inferred to be 0.

- b. In one example, when MinCbSizeY is larger than the maximum allowed size for TS and/or BDPCM (e.g. the maximum allowed size for TS and/or BDPCM is 32), **sps_transform_skip_enabled_flag** is not signaled and inferred to be 0.
9. Whether to and/or how to signal or interpret the indication of coding tool X in SPS/PPS/Picture header/slice header may depend on the minimum allowed coding block size (e.g. MinCbSizeY in JVET-Q2001-vB).
- a. The indication of coding tool X in SPS/PPS/Picture header/slice header may not be signaled and inferred to be not used if the minimum allowed coding block size is larger than T, wherein T is an integer such as 32.
 - b. It is required that the coding tool X in SPS/PPS/Picture header/slice header must be indicated to be not used if the minimum allowed coding block size is larger than T, wherein T is an integer such as 32.
 - c. The coding tool X may be Combined Inter-Intra Prediction (CIIP).
 - d. The coding tool X may be Multiple Transform Selection (MTS).
 - e. The coding tool X may be Segment Block Transform (SBT).
 - f. The coding tool X may be Symmetric Motion Vector Difference (SMVD).
 - g. The coding tool X may be BDOF.
 - h. The coding tool X may be affine prediction.
 - i. The coding tool X may be Prediction Refine with Optical Flow (PROF).
 - j. The coding tool X may be Decoder-side Motion Vector Refinement (DMVR).
 - k. The coding tool X may be Bi-prediction with CU-level Weights (BCW).
 - l. The coding tool X may be Merge with Motion Vector Difference (MMVD).
 - m. The coding tool X may be Geometric Partitioning Mode (GPM).
 - n. The coding tool X may be Intra Block Copy (IBC).
 - o. The coding tool X may be palette coding.
 - p. The coding tool X may be Adaptive Color Transform (ACT).
 - q. The coding tool X may be Joint Cb-Cr Residue coding (JCCR).
 - r. The coding tool X may be Cross-Component Linear Model prediction (CCLM).
 - s. The coding tool X may be Multiple Reference Line (MRL).
 - t. The coding tool X may be Matrix-based Intra-Prediction (MIP).
 - u. The coding tool X may be Intra Subpartitions Prediction (ISP).

10. The minimum allowed block size for Binary-Tree split (e.g. MinBtSizeY in JVET-Q2001-vB) may be different for luma component and chroma components when dual-tree coding is applied.
- a. In one example, $\text{MinBtSizeY} = 1 \ll \text{MinBtLog2SizeY}$ is the minimum allowed block size for Binary-Tree split for the luma component and $\text{MinBtSizeC} = 1 \ll \text{MinBtLog2SizeC}$ is the minimum allowed block size for Binary-Tree split for the chroma components, wherein MinBtLog2SizeY may be not equal to MinBtLog2SizeC .
 - i. MinBtLog2SizeY may be signaled predicted by MinCbLog2SizeY . For example, the difference between MinBtLog2SizeY and MinCbLog2SizeY may be signaled.
 - ii. MinBtLog2SizeC may be signaled predicted by MinCbLog2SizeY . For example, the difference between MinBtLog2SizeC and MinCbLog2SizeY may be signaled.
11. The minimum allowed block size for Ternary-Tree split (e.g. MinTtSizeY in JVET-Q2001-vB) may be different for luma component and chroma components when dual-tree coding is applied. $\text{MinCbSizeY} = 1 \ll \text{MinCbLog2SizeY}$ denotes the minimum allowed coding block size.
- a. In one example, $\text{MinTtSizeY} = 1 \ll \text{MinTtLog2SizeY}$ is the minimum allowed block size for Ternary-Tree split for the luma component and $\text{MinTtSizeC} = 1 \ll \text{MinTtLog2SizeC}$ is the minimum allowed block size for Ternary-Tree split for the chroma components, wherein MinTtLog2SizeY may be not equal to MinTtLog2SizeC .
 - i. MinTtLog2SizeY may be signaled predicted by MinCbLog2SizeY . For example, the difference between MinTtLog2SizeY and MinCbLog2SizeY may be signaled.
 - ii. MinTtLog2SizeC may be signaled predicted by MinCbLog2SizeY . For example, the difference between MinTtLog2SizeC and MinCbLog2SizeY may be signaled.
12. The maximum number of subblock based merge candidates (e.g. $\text{MaxNumSubblockMergeCand}$) is derived as a sum of a first variable and a second variable,

wherein the first variable is equal to 0 if affine prediction is disabled (e.g. `sps_affine_enabled_flag` is equal to 0), and the second variable depends whether subblock based TMVP (sbTMVP) is enabled or not.

- a. In one example, the first variable represents a number of allowed affine merge candidates.
 - b. In one example, the second variable may be set to $(\text{sps_sbtmvp_enabled_flag} \ \&\& \ \text{ph_temporal_mvp_enable_flag})$.
 - c. In one example, the first variable is derived as $K-S$, wherein S is a value signalled by a syntax element (e.g. `five_minus_num_affine_merge_cand`) and K is a fixed number such as 4 or 5.
 - d. In one example, $\text{MaxNumSubblockMergeCand} = 5 - \text{five_minus_max_num_affine_merge_cand} + (\text{sps_sbtmvp_enabled_flag} \ \&\& \ \text{ph_temporal_mvp_enable_flag})$.
 - e. In one example, $\text{MaxNumSubblockMergeCand} = 4 - \text{four_minus_max_num_affine_merge_cand} + (\text{sps_sbtmvp_enabled_flag} \ \&\& \ \text{ph_temporal_mvp_enable_flag})$.
 - f. In one example, $\text{MaxNumSubblockMergeCand} = \text{Min}(W, \text{MaxNumSubblockMergeCand})$, wherein W is a fixed number such as 5.
 - g. In one example, indication of the first variable (e.g., `five_minus_max_num_affine_merge_cand` or `four_minus_max_num_affine_merge_cand`) may be conditionally signalled.
 - i. In one example, it may be signalled only if `sps_affine_enabled_flag` is equal to 1.
 - ii. When not present, `five_minus_max_num_affine_merge_cand` is inferred to be K (e.g., $K = 4$ or 5).
13. Number of rows (lines) between ALF virtual boundary and CTB bottom boundary for luma component and chroma components; and/or filter strength for a row between ALF virtual boundary and CTB bottom boundary for luma component and that for a corresponding row of chroma components; and/or padding method for luma and chroma samples with the same row; are aligned for 4:2:2 and 4:4:4 cases.

- a. In one example, the vertical (and/or horizontal) position of ALF virtual boundary (VB) in chroma ALF shall be aligned with the vertical (and/or horizontal) position of ALF virtual boundary in luma ALF for 4:2:2/4:4:4 chroma format video. Denote the vertical (and/or horizontal) position of ALF VB in luma ALF by $vbPosY$, and the vertical (and/or horizontal) position of ALF VB in chroma ALF by $vbPosC$.
 - i. In one example, when the vertical position of ALF VB in luma ALF, $vbPosY$, is equal to $CtbSizeY - S$, the vertical position of ALF VB for chroma component, $vbPosC$, may be set equal to $(CtbSizeY - S) / SubHeightC$.
 - ii. In one example, when the horizontal position of ALF VB in luma ALF, $vbPosY$ is equal to $CtbSizeY - S$, the horizontal position of ALF VB for chroma component, $vbPosC = (CtbSizeY - S) / SubWidthC$.
 - iii. In above examples, $CtbSizeY$ specifies the luma coding tree block size of each CTU and $SubHeightC$ and $SubWidthC$ are defined in Table 3-1. S is an integer such as 4.
- b. In one example, the padding method in chroma ALF for K lines (and/or H columns) near the vertical (and/or horizontal) position of ALF VB shall be aligned with the padding method in luma ALF. Denote the vertical (or horizontal) chroma sample position by Yc .
 - i. In on example, when Yc is equal to $vbPosC$ (e.g., in FIG. 14B) the following may apply:
 - 1) The above (or left) K (e.g., $K = 2$) unavailable lines may be padded. Alternatively, furthermore, the corresponding K lines of these unavailable lines below (or right) current line may also be padded even the corresponding K lines are available.
 - a. In one example, the above K unavailable lines may be padded using current line.
 - i. In one example, in FIG. 14B $C1$, $C2$, $C3$ above current line may be set equal to $C5$, $C6$, $C5$. And $C0$ above current line may be set equal to $C6$.

- b. In one example, the corresponding K lines may be padded using current line.
 - i. In one example, in FIG. 14B, C3, C2, C1 below current line may be set equal to C5, C6, C5. And C0 below current line may be set equal to C6.
- 2) The below or (right) K unavailable lines may be padded. Alternatively, furthermore, the corresponding K lines of these unavailable lines above (or left) current line may also be padded even the corresponding K lines are available.
- ii. In one example, when Y_c is equal to $vbPosC - M$ (e.g., $M = 1, 2$) (e.g., in FIG. 14A and FIG. 14D), the following may apply:
 - 1) the below (or right) K (e.g., $K = 1, 2$) unavailable lines may be padded. And the corresponding K lines of these unavailable lines above (or left) current line may also be padded even the corresponding K lines are available.
 - a. In one example, when M is equal to 1, the below K unavailable lines may be padded using current line.
 - i. In one example, in FIG. 14A, C3, C2, C1 below current line may be set equal to C5, C6, C5. And C0 below current line may be set equal to C6.
 - b. In one example, when M is equal to 1, the corresponding K lines may be padded using current line.
 - i. In one example, in FIG. 14A, C1, C2, C3 above current line may be set equal to C5, C6, C5. And C0 above current line may be set equal to C6.
 - c. In one example, when M is larger than or equal to 2, the below K unavailable lines may be padded using the downmost line above the ALF virtual boundary (e.g., $vbPosC - 1$).
 - i. In one example, in FIG. 14D, C0 below current line may be set equal to C6.

- d. In one example, when M is larger than or equal to 2, the corresponding K lines may be padded using the corresponding line of the downmost line above the ALF virtual boundary (e.g., $vbPos - 2 * M + 1$).
 - i. In one example, in FIG. 14D, $C0$ above current line may be set equal to $C6$.
- iii. In one example, when Yc is equal to $vbPosC + N$ (e.g., $N = 1$) (e.g., in FIG. 14C), the following may apply:
 - 1) the above (or left) K (e.g., $K = 1$) unavailable lines may be padded. And the corresponding K lines of these unavailable lines below (or right) current line may also be padded even the corresponding K lines are available.
 - a. In one example, the above K unavailable lines may be padded using current line.
 - i. In one example, in FIG. 14C, $C0$ above current line may be set equal to $C6$.
 - b. In one example, the corresponding K lines may be padded using current line.
 - i. In one example, in FIG. 14C, $C0$ below current line may be set equal to $C6$.
- c. In one example, the ALF filter strength of M lines (or N columns) near the vertical (or horizontal) position of ALF VB in chroma ALF shall be aligned with that in luma ALF for 4:2:2/4:4:4 chroma format video. The ALF filter strength of luma ALF and chroma ALF is controlled by $alfShiftY$ (e.g., in Table 45) and $alfShiftC$ (e.g., in Table 46).
 - i. In one example, when $Yc == vbPosC - M$ (e.g., $M = 0, 1$), $alfShiftC = T1$; when $Yc != vbPosC - M$, $alfShiftC = T2$.
 - 1) In one example, $T1 = 10$, $T2 = 7$.
 - ii. In one example, when $Yc == vbPosC + M$ (e.g., $M = 0, 1$), $alfShiftC = T1$; when $Yc != vbPosC + M$, $alfShiftC = T2$.
 - 1) In one example, $T1 = 10$, $T2 = 7$.

6. Embodiments

6.1. Embodiment of signalling of loop_filter_across_tiles_enabled_flag

7.3.2.4 Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	Descriptor
...	
if(NumTilesInPic > 1)	
loop_filter_across_tiles_enabled_flag	u(1)
...	
}	

6.2. Embodiment #1 of signalling of loop_filter_across_slices_enabled_flag

7.3.2.4 Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	Descriptor
...	
if(!(rect_slice_flag && !single_slice_per_subpic_flag && num_slices_in_pic_minus1 == 0))	
loop_filter_across_slices_enabled_flag	u(1)
...	
}	

6.3. Embodiment #2 of signalling of loop_filter_across_slices_enabled_flag

7.3.2.4 Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	Descriptor
...	

if(!rect_slice_flag single_slice_per_subpic_flag num_slices_in_pic_minus1 > 0))	
loop_filter_across_slices_enabled_flag	u(1)
...	
}	

6.4. Embodiment #3 of signalling of loop_filter_across_slices_enabled_flag

7.3.2.4 Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	Descriptor
...	
if(!(rect_slice_flag && num_slices_in_pic_minus1 == 0) && !single_slice_per_subpic_flag)	
loop_filter_across_slices_enabled_flag	u(1)
...	
}	

6.5. Embodiment #4 of signalling of loop_filter_across_slices_enabled_flag

7.3.2.4 Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	Descriptor
...	
if(!single_slice_per_subpic_flag && (!rect_slice_flag num_slices_in_pic_minus1 > 0))	
loop_filter_across_slices_enabled_flag	u(1)
...	
}	

6.6. Embodiment #1 of signalling of LMCS information

7.3.2.4 Picture parameter set RBSP syntax

pic_parameter_set_rbsp() {	Descriptor
...	
alf_info_in_ph_flag	u(1)
lmcs_info_in_ph_flag	u(1)
...	
}	

6.7. Embodiment #2 of signalling of LMCS information

7.3.2.7 Picture parameter set RBSP syntax

picture_header_structure() {	Descriptor
...	
if(sps_lmcs_enabled_flag && lmcs_info_in_ph_flag) {	
ph_lmcs_enabled_flag	u(1)
if(ph_lmcs_enabled_flag) {	
ph_lmcs_aps_id	u(2)
if(ChromaArrayType != 0)	
ph_chroma_residual_scale_flag	u(1)
}	
}	
...	
}	

6.8. Embodiment #3 of signalling of LMCS information

7.3.7.1 General slice header syntax

slice_header() {	Descriptor
...	
if(sps_lmcs_enabled_flag && !lmcs_info_in_ph_flag) {	

slice_lmcs_enabled_flag	u(1)
if(slice_lmcs_enabled_flag) {	
slice_lmcs_aps_id	u(2)
if(ChromaArrayType != 0)	
slice_chroma_residual_scale_flag	u(1)
}	
}	
...	
}	

6.9. Embodiment #4 of signalling of LMCS information

7.3.2.7 Picture parameter set RBSP syntax

picture_header_structure() {	Descriptor
...	
if(sps_lmcs_enabled_flag) {	
ph_lmcs_enabled_flag	u(1)
if(ph_lmcs_enabled_flag && lmcs_info_in_ph_flag) {	
ph_lmcs_aps_id	u(2)
if(ChromaArrayType != 0)	
ph_chroma_residual_scale_flag	u(1)
}	
}	
...	
}	

6.10. Embodiment #5 of signalling of LMCS information

7.3.7.1 General slice header syntax

slice_header () {	Descriptor
...	
if(ph_lmcs_enabled_flag)	
slice_lmcs_enabled_flag	u(1)
if(slice_lmcs_enabled_flag && !lmcs_info_in_ph_flag) {	
slice_lmcs_aps_id	u(2)
if(ChromaArrayType != 0)	
slice_chroma_residual_scale_flag	u(1)
}	
...	
}	

6.11. Embodiment #6 of signalling of LMCS information

Eg 1: When **slice_lmcs_enabled_flag** is not present, it is inferred to be equal to **ph_lmcs_enabled_flag**.

Eg 2: When **slice_lmcs_aps_id** is not present, it is inferred to be equal to **ph_lmcs_aps_id**.

Eg 3: When **slice_chroma_residual_scale_flag** is not present, it is inferred to be equal to **ph_chroma_residual_scale_flag**.

6.12. Embodiment #1 of signalling of slice information

7.3.2.4 Picture parameter set RBSP syntax

picture_header_structure() {	Descriptor
...	
no_pic_partition_flag	u(1)
if(!no_pic_partition_flag) {	
pps_log2_ctu_size_minus5	u(2)
num_exp_tile_columns_minus1	ue(v)

num_exp_tile_rows_minus1	ue(v)
for(i = 0; i <= num_exp_tile_columns_minus1; i++)	
tile_column_width_minus1[i]	ue(v)
for(i = 0; i <= num_exp_tile_rows_minus1; i++)	
tile_row_height_minus1[i]	ue(v)
if(NumTilesInPic > 1)	
rect_slice_flag	u(1)
if(rect_slice_flag)	
single_slice_per_subpic_flag	u(1)
if(rect_slice_flag && !single_slice_per_subpic_flag) {	
num_slices_in_pic_minus1	ue(v)
if(num_slices_in_pic_minus1 > 0)	
tile_idx_delta_present_flag	u(1)
for(i = 0; i < num_slices_in_pic_minus1; i++) {	
if(NumTileColumns > 1)	
slice_width_in_tiles_minus1[i]	ue(v)
if(NumTileRows > 1 && (tile_idx_delta_present_flag tileIdx % NumTileColumns == 0))	
slice_height_in_tiles_minus1[i]	ue(v)
if(slice_width_in_tiles_minus1[i] == 0 && slice_height_in_tiles_minus1[i] == 0 && RowHeight[SliceTopLeftTileIdx[i] / NumTileColumns] > 1) {	
num_exp_slices_in_tile[i]	ue(v)
numExpSlicesInTile = num_exp_slices_in_tile[i]	
for(j = 0; j < numExpSlicesInTile; j++)	

exp_slice_height_in_ctus_minus1[j]	ue(v)
i += NumSlicesInTile[i] – 1	
}	
if(tile_idx_delta_present_flag && i < num_slices_in_pic_minus1)	
tile_idx_delta[i]	se(v)
}	
}	
loop_filter_across_tiles_enabled_flag	u(1)
loop_filter_across_slices_enabled_flag	u(1)
}	
...	
}	

6.13. Embodiment of handling ALF virtual boundary

8.8.5.4 Coding tree block filtering process for chroma samples

...

- The vertical sample position offsets y1, y2 and the variable alfShiftC are specified in Table 46 ~~Table 45~~ according to the vertical chroma sample position y and applyAlfLineBufBoundary.
- The variable curr is derived as follows:

$$\text{curr} = \text{recPicture}[h_x][v_y] \quad (1504)$$

- The array of chroma filter coefficients f[j] and the array of chroma clipping values c[j] is derived as follows with j = 0..5:

$$f[j] = \text{AlfCoeffc}[\text{slice_alf_aps_id_chroma}][\text{altIdx}][j] \quad (1505)$$

$$c[j] = \text{AlfClipc}[\text{slice_alf_aps_id_chroma}][\text{altIdx}][j] \quad (1506)$$

- The variable sum is derived as follows:

$$\begin{aligned}
 \text{sum} = & f[0] * (\text{Clip3}(-c[0], c[0], \text{recPicture}[h_x][v_y+y_2] - \text{curr}) + \\
 & \text{Clip3}(-c[0], c[0], \text{recPicture}[h_x][v_y-y_2] - \text{curr})) + \\
 & f[1] * (\text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x+1}][v_y+y_1] - \text{curr}) + \\
 & \text{Clip3}(-c[1], c[1], \text{recPicture}[h_{x-1}][v_y-y_1] - \text{curr})) + \\
 & f[2] * (\text{Clip3}(-c[2], c[2], \text{recPicture}[h_x][v_y+y_1] - \text{curr}) + \\
 & \text{Clip3}(-c[2], c[2], \text{recPicture}[h_x][v_y-y_1] - \text{curr})) + \quad (1507) \\
 & f[3] * (\text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x-1}][v_y+y_1] - \text{curr}) + \\
 & \text{Clip3}(-c[3], c[3], \text{recPicture}[h_{x+1}][v_y-y_1] - \text{curr})) + \\
 & f[4] * (\text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x+2}][v_y] - \text{curr}) + \\
 & \text{Clip3}(-c[4], c[4], \text{recPicture}[h_{x-2}][v_y] - \text{curr})) + \\
 & f[5] * (\text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x+1}][v_y] - \text{curr}) + \\
 & \text{Clip3}(-c[5], c[5], \text{recPicture}[h_{x-1}][v_y] - \text{curr})) \\
 \\
 \text{sum} = & \text{curr} + ((\text{sum} + 64) >> \text{alfShiftC}) \\
 & (1508)
 \end{aligned}$$

- The modified filtered reconstructed chroma picture sample $\text{alfPicture}[x_{\text{CtbC}} + x][y_{\text{CtbC}} + y]$ is derived as follows:

$$\text{alfPicture}[x_{\text{CtbC}} + x][y_{\text{CtbC}} + y] = \text{Clip3}(0, (1 \ll \text{BitDepth}) - 1, \text{sum}) \quad (1509)$$

Table 46 – Specification of y_1 , y_2 and alfShiftC according to the vertical chroma sample position y and $\text{applyAlfLineBufBoundary}$

Condition	alfShiftC	y1	y2
(y == ctbHeightC - 2 y == ctbHeightC - 3) (y == (ctbSizeY - 4) / SubHeightC - 1 y == (ctbSizeY - 4) / SubHeightC) && (applyAlfLineBufBoundary == 1)	10	0	0
(y == ctbHeightC - 1 y == ctbHeightC - 4) (y == (ctbSizeY - 4) / SubHeightC - 2 y == (ctbSizeY - 4) / SubHeightC + 1) && (applyAlfLineBufBoundary == 1)	7	1	1
Otherwise	7	1	2

[00128] FIG. 15 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.

[00129] 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 coded 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 representation is sometimes called video decompression. Furthermore, while certain video processing operations are referred to as “coding” operations or tools, it will be appreciated that the coding tools or operations are used at an encoder and corresponding decoding tools or operations that reverse the results of the coding will be performed by a decoder.

[00130] 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.

[00131] FIG. 16 is a block diagram of a video processing apparatus 3600. The apparatus 3600 may be used to implement one or more of the methods described herein. The apparatus 3600 may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus 3600 may include one or more processors 3602, one or more memories 3604 and video processing hardware 3606. The processor(s) 3602 may be configured to implement one or more methods described in the present document. The memory (memories) 3604 may be used for storing data and code used for implementing the methods and techniques described herein. The video processing hardware 3606 may be used to implement, in hardware circuitry, some techniques described in the present document.

[00132] FIG. 18 is a block diagram that illustrates an example video coding system 100 that may utilize the techniques of this disclosure.

[00133] 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.

[00134] Source device 110 may include a video source 112, a video encoder 114, and an input/output (I/O) interface 116.

[00135] 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.

[00136] Destination device 120 may include an I/O interface 126, a video decoder 124, and a display device 122.

[00137] 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.

[00138] 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 (VVM) standard and other current and/or further standards.

[00139] 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.

[00140] 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.

[00141] 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.

[00142] 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.

[00143] 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.

[00144] 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.

[00145] 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.

[00146] 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.

[00147] 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.

[00148] 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.

[00149] 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.

[00150] In some examples, motion estimation unit 204 may output a full set of motion information for decoding processing of a decoder.

[00151] 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.

[00152] 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.

[00153] 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.

[00154] As discussed above, video encoder 200 may predictively signal the motion vector. Two examples of predictive signaling techniques that may be implemented by video encoder 200 include advanced motion vector predication (AMVP) and merge mode signaling.

[00155] 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.

[00156] 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.

[00157] 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.

[00158] 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.

[00159] 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.

[00160] 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.

[00161] After reconstruction unit 212 reconstructs the video block, loop filtering operation may be performed reduce video blocking artifacts in the video block.

[00162] 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.

[00163] Some embodiments of the disclosed technology include making a decision or determination to enable a video processing tool or mode. In an example, when the video processing tool or mode is enabled, the encoder will use or implement the tool or mode in the processing of a block of video, but may not necessarily modify the resulting bitstream based on the usage of the tool or mode. That is, a conversion from the block of video to the bitstream representation of the video will use the video processing tool or mode when it is enabled based on the decision or determination. In another example, when the video processing tool or mode is enabled, the decoder will process the bitstream with the knowledge that the bitstream has been modified based on the video processing tool or mode. That is, a conversion from the bitstream representation of the video to the block of video will be performed using the video processing tool or mode that was enabled based on the decision or determination.

[00164] 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.

[00165] 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.

[00166] 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).

[00167] 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.

[00168] 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.

[00169] 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.

[00170] 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.

[00171] 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.

[00172] 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.

[00173] A listing of solutions preferred by some embodiments is provided next.

[00174] The following solutions show example embodiments of techniques discussed in the previous section (e.g., items 1 and 2).

[00175] 1. A video processing method (e.g., method 1700 shown in FIG. 17), comprising: determining (1702), for a conversion between a video unit of a video and a coded representation of the video, whether an indication of applicability of in-loop filtering across video regions of the video unit is included in the coded representation; and performing (1704) the conversion based on the determining.

[00176] 2. The method of solution 1, wherein the video unit comprises a picture.

[00177] 3. The method of any of solutions 1-2, wherein the video region comprises a tile.

[00178] 4. The method of any of solutions 1-2, wherein the video region comprises a slice.

[00179] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 3).

[00180] 5. A method of video processing, comprising: performing a conversion between a video region of a video and a coded representation of the video, wherein the coded representation conforms to a format rule, wherein the format rule specifies that information about applicability a luma mapping with chroma scaling (LMCS) tool to the conversion is indicated in the coded representation at a video slice level; wherein, the LMCS tool includes, during the conversion, to construct the current video block based on a first domain and a second domain and/or chroma residue is scaled in a luma-dependent manner.

[00181] The following solutions show example embodiments of techniques discussed in the previous section (e.g., items 4, 5, 7, 8).

[00182] 6. A method of video processing, comprising: performing a conversion between a video and a coded representation of the video, wherein the conversion confirms to a size rule that specifies that: enforcing the size rule, during encoding, a maximum size of a video region coded using a transform-skip coding scheme or a delta pulse coded modulation coding scheme, or parsing and decoding the coded representation, during decoding, by enforcing the size rule for the maximum size of the video region decoded using a transform-skip decoding scheme or a delta pulse coded modulation decoding scheme.

[00183] 7. The method of solution 6, wherein the size rule specifies that the maximum size of the video region that is a transform skip block is less than or equal to a coding tree block size.

[00184] 8. The method of solution 6, wherein the size rule specifies that the maximum size of the video region that is processed using block based delta pulse mode coding is less than or equal to a coding tree block size.

[00185] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 5).

[00186] 9. The method of solution 6, wherein the size rule specifies that the maximum size of the video region that is a transform skip block is less than or equal to a minimum coding block size.

[00187] 10. The method of solution 6, wherein the size rule specifies that the maximum size of the video region that is processed using block based delta pulse mode coding is less than or equal to a minimum coding block size.

[00188] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 7).

[00189] 11. The method of any of solutions 6 to 10, wherein the size is indicated in a field in the coded representation and wherein a minimum allowed coding block size for the conversion controls where the field occurs in the coded representation and/or how the size is interpreted from the field.

[00190] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 8).

[00191] 12. The method of any of solutions 6 to 10, wherein a minimum allowed coding block size for the conversion controls whether a field indicative of the size rule occurs in the coded representation and/or how the size is interpreted from the field.

[00192] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 6).

[00193] 13. A method of video processing, comprising: performing a conversion between a video region of a video and a coded representation of the video, wherein the coded representation conforms to a format rule that specifies that a minimum allowed transform block size used during the conversion controls whether or how an indication of maximum allowed transform block size is included in the coded representation.

[00194] 14. The method of solution 13, wherein the format rule specifies that the minimum allowed transform block size is greater than or equal to the minimum allowed coding block size.

[00195] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 9).

[00196] 15. A method of video processing, comprising: performing a conversion between a video unit of a video and a coded representation of the video according to a format rule, wherein the coded representation conforms to the format rule that specifies that a minimum allowed coding block size used during the conversion controls whether a field carrying information regarding applicability of a coding tool in the conversion is included at a video region level.

[00197] 16. The method of solution 15, wherein the video region corresponds to a sequence parameter set or a picture parameter set or a picture header or a slice header.

[00198] 17. The method of any of solutions 15-16, wherein the coding tool includes a combined inter-intra prediction tool.

[00199] 18. The method of any of solutions 15-16, wherein the coding tool includes a multiple transform selection coding tool.

[00200] The following solutions show example embodiments of techniques discussed in the previous section (e.g., items 10, 11).

[00201] 19. A method of video processing, comprising: determining, for a conversion between a video region of a video and a coded representation of the video, that due to using a dual-tree coding is used for the conversion, that splitting schemes used for luma and chroma components of the video have different allowed minimum block sizes for the luma and chroma components, and performing the conversion based on the determining.

[00202] 20. The method of 19, wherein the splitting schemes include binary-tree splitting.

[00203] 21. The method of solution 19, wherein the splitting schemes include ternary-tree splitting.

[00204] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 12).

[00205] 22. A method of video processing, comprising: performing a conversion between a video region of a video and a coded representation of the video region by conforming a format rule for the coded representation; wherein the format rule specifies that a maximum number of sub-block based merge candidates used during the conversion is derivable as a sum of a first

variable and a second variable, wherein use of affine prediction controls a value of the first variable, and wherein use of sub-block based temporal motion vector prediction controls a value of the second variable.

[00206] 23. The method of solution 22, wherein the first variable represents a number of allowed affine merge candidates.

[00207] The following solutions show example embodiments of techniques discussed in the previous section (e.g., item 13).

[00208] 24. A method of video processing, comprising: performing a conversion between a video region of a video and a coded representation of the video region by conforming a processing rule that is applicable to the conversion due to the video being a 4:2:2 or a 4:4:4 video, wherein the processing rule defines that chroma and luma are aligned for one or more of the following: (a) a number of pixel lines between a virtual boundary of an adaptive loop filtering operation and a bottom boundary of a coding tree block; or (b) a filter strength of a filter applied to a row between the virtual boundary of the adaptive loop filtering operation and the bottom boundary of the coding tree block; or (c) a padding method used for padding video samples in a row.

[00209] 25. The method of solution 24, wherein the processing rule further defines a vertical alignment of virtual boundaries between chroma and luma components.

[00210] 26. The method of solution 24, wherein the processing rule defines that a padding method used for padding K lines and/or H columns of chroma is aligned with a padding method used for luma.

[00211] 27. The method of any of above solutions, wherein the video region comprises a video coding unit.

[00212] 28. The method of any of above solutions, wherein the video region comprises a video picture.

[00213] 29. The method of any of solutions 1 to 28, wherein the conversion comprises encoding the video into the coded representation.

[00214] 30. The method of any of solutions 1 to 28, wherein the conversion comprises decoding the coded representation to generate pixel values of the video.

[00215] 31. A video decoding apparatus comprising a processor configured to implement a method recited in one or more of solutions 1 to 30.

[00216] 32. A video encoding apparatus comprising a processor configured to implement a method recited in one or more of solutions 1 to 30.

[00217] 33. A computer program product having computer code stored thereon, the code, when executed by a processor, causes the processor to implement a method recited in any of solutions 1 to 30.

[00218] 34. A method, apparatus or system described in the present document.

[00219] FIG. 21 is a flowchart for an example method (2100) of video processing. Operation 2102 includes performing a conversion between a video comprising a video unit and a bitstream of the video, wherein the bitstream conforms to a format rule, and wherein the format rule specifies that a first syntax element indicating whether an in-loop filtering operation is performed across tile boundaries is selectively included in the bitstream responsive to whether or how the video unit is partitioned into tiles.

[00220] In some embodiments for method 2100, the video unit comprises a picture. In some embodiments for method 2100, the first syntax element is in a picture parameter set. In some embodiments for method 2100, the format rule specifies that the first syntax element is included in the bitstream in response to the video unit being partitioned into more than one tile. In some embodiments for method 2100, the format rule specifies that the first syntax element is not included in the bitstream in response to the video unit comprising only one tile. In some embodiments for method 2100, the format rule specifies that the first syntax element is included in the bitstream in response to the bitstream including a value of zero for a second syntax element that indicates whether there is one tile per picture as a constraint.

[00221] In some embodiments for method 2100, the format rule specifies that the first syntax element is required to be equal to zero and is included in the bitstream in response to the bitstream including a value of one for a second syntax element that indicates whether there is one tile per picture as a constraint. In some embodiments for method 2100, the second syntax element is `one_tile_per_pic_constraint_flag`. In some embodiments for method 2100, the in-loop filtering operation includes at least one of a deblocking filter operation, a sample adaptive offset operation, or an adaptive loop filter operation. In some embodiments for method 2100, the first syntax element is `loop_filter_across_tiles_enabled_flag`. In some embodiments for method 2100, the performing the conversion comprising encoding the video into the bitstream.

[00222] In some embodiments for method 2100, the performing the conversion comprises generating the bitstream from the video, and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium. In some embodiments for method 2100, the performing the conversion comprises decoding the video from the bitstream. In some embodiments, a video decoding apparatus comprising a processor configured to implement method 2100 and operations of the related embodiments. In some embodiments, a video encoding apparatus comprising a processor configured to implement method 2100 and operations of the related embodiments.

[00223] In some embodiments, a computer program product having computer instructions stored thereon, the instructions, when executed by a processor, causes the processor to implement method 2100 and operations related to related embodiments. In some embodiments, a non-transitory computer-readable recording medium storing a bitstream of a video which is generated by a method performed by a video processing apparatus, wherein the method comprises: generating the bitstream from the video comprising a video unit; wherein the bitstream conforms to a format rule, and wherein the format rule specifies that a first syntax element indicating whether an in-loop filtering operation is performed across tile boundaries is selectively included in the bitstream responsive to whether or how the video unit is partitioned into tiles. In some embodiments, a non-transitory computer-readable storage medium storing instructions that cause a processor to implement method 2100 and operations of the related embodiments. In some embodiments, a method of bitstream generation, comprising: generating a bitstream of a video according to method 2100 and operations of the related embodiments, and storing the bitstream on a computer-readable program medium.

[00224] FIG. 22 is a flowchart for an 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, wherein the bitstream conforms to a format rule, and wherein the format rule specifies that a syntax element indicating whether an in-loop filtering operation is performed across slice boundaries is selectively included in the bitstream responsive to whether or how the video unit is partitioned into slices.

[00225] In some embodiments of method 2200, the video unit comprises a picture. In some embodiments of method 2200, the video unit comprises a sub-picture. In some embodiments of method 2200, the syntax element is in a picture parameter set. In some embodiments of method

2200, the rule specifies that the syntax element is not included in the bitstream in response to the video unit being partitioned into only one slice. In some embodiments of method 2200, the rule specifies that the syntax element is not included in the bitstream in response to each sub-picture of the video unit being partitioned into only one slice. In some embodiments of method 2200, the rule specifies that the syntax element is included in the bitstream in response to each sub-picture of the video unit being partitioned into only one slice.

[00226] In some embodiments of method 2200, the rule specifies that the syntax element is included in the bitstream in response to a first flag being equal to 1 which specifies that each sub-picture consists of one and only one rectangular slice. In some embodiments of method 2200, the first flag is `single_slice_per_subpic_flag`. In some embodiments of method 2200, the first flag is in a picture parameter set.

[00227] In some embodiments of method 2200, the rule specifies that the syntax element is included in the bitstream in response to the video unit comprising a picture that is partitioned in a non-rectangular shape. In some embodiments of method 2200, the rule specifies that the syntax element is included in the bitstream in response to a second flag being equal to 0 which specifies that a raster-san slice mode is in use for each picture. In some embodiments of method 2200, the raster-san slice mode is a non-rectangular slice mode. In some embodiments of method 2200, the second flag is `rect_slice_flag`. In some embodiments of method 2200, the second flag is in a picture parameter set. In some embodiments of method 2200, the rule specifies that the syntax element is not included in the bitstream in response to the video unit comprising a picture that is partitioned in a rectangular shape and a number of slices of the video unit is equal to one.

[00228] In some embodiments of method 2200, the rule specifies that the syntax element is included in the bitstream in response to the bitstream including a value of zero for a syntax element that indicates whether there is one slice per picture as a constraint. In some embodiments of method 2200, the rule specifies that the syntax element is required to be equal to zero and is included in the bitstream in response to the bitstream including a value of one for a syntax element that indicates whether there is one slice per picture as a constraint. In some embodiments of method 2200, the in-loop filtering operation includes at least one of a deblocking filter operation, a sample adaptive offset operation, or an adaptive loop filter operation. In some embodiments of method 2200, the performing the conversion comprising encoding the video into the bitstream. In some embodiments of method 2200, the performing the

conversion comprises generating the bitstream from the video, and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.

[00229] In some embodiments of method 2200, the performing the conversion comprises decoding the video from the bitstream. In some embodiments, a video decoding apparatus comprising a processor configured to implement method 2200 and operations of the related embodiments. In some embodiments, a video encoding apparatus comprising a processor configured to implement method 2200 and operations of the related embodiments. In some embodiments, a computer program product having computer instructions stored thereon, the instructions, when executed by a processor, causes the processor to implement method 2200 and operations of the related embodiments. In some embodiments, a non-transitory computer-readable recording medium storing a bitstream of a video which is generated by a method performed by a video processing apparatus, wherein the method comprises: generating the bitstream from a video unit of the video; wherein the bitstream conforms to a format rule, and wherein the format rule specifies that a syntax element indicating whether an in-loop filtering operation is performed across slice boundaries is selectively included in the bitstream responsive to whether or how the video unit is partitioned into slices.

[00230] In some embodiments, a non-transitory computer-readable storage medium storing instructions that cause a processor to implement method 2200 and operations of the related embodiment. In some embodiments, a method of bitstream generation, comprising: generating a bitstream of a video according to method 2200 and operations of the related embodiment, and storing the bitstream on a computer-readable program medium.

[00231] FIG. 23 is a flowchart for an example method (2300) of video processing. Operation 2302 includes performing a conversion between a video region of a video and a bitstream of the video, wherein the bitstream conforms to a format rule, wherein the format rule specifies that information about applicability of a tool to the conversion is indicated in the bitstream at a video slice level and/or at a video picture level, and wherein the tool maps luma samples to particular values and selectively applies a scaling operation to the values of chroma samples.

[00232] In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool includes a first indication of whether to enable the tool. In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool includes a second indication of parameters, and the parameters include an identifier of

an Adaptation Parameter Set (APS) for the tool at the video picture level, and/or a value that indicates whether a chroma residual scale is enabled at the video picture level. In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool is indicated at both the video slice level and the video picture level.

[00233] In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool is indicated at both the video slice level and the video picture level, and a previous information about applicability of the tool when included at the video picture level is overwritten with the information about applicability of the tool when the information about applicability of the tool is included at the video slice level. In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool is indicated at either the video slice level or the video picture level. In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool is indicated at either the video slice level to indicate which Adaptation Parameter Set (APS) is used for a current slice of the video region. In some embodiments of method 2300, the format rule specifies that the bitstream include a syntax element at a Sequence Parameter Set (SPS) or at a Picture Parameter Set (PPS) to indicate whether the information about applicability of the tool is indicated at the video slice level or at the video picture level.

[00234] In some embodiments of method 2300, the syntax element controls only the first indication. In some embodiments of method 2300, the syntax element controls only the second indication. In some embodiments of method 2300, the syntax element controls both the first indication and the second indication. In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool is indicated in a syntax element in a picture header with a default value in response to the information not being present in the picture header. In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool is indicated in a syntax element in a slice header with a default value in response to the information not being present in the slice header. In some embodiments of method 2300, the format rule specifies that the information about applicability of the tool is indicated in a syntax element in a slice header, the format rule specifies that the syntax element has a value of a corresponding syntax element that indicates the information about applicability of the tool in a picture header, and the format rule specifies that the syntax element has the value

of the corresponding syntax element in response to the information not being present in the slice header.

[00235] In some embodiments of method 2300, the format rule specifies that a plurality of Adaptation Parameter Set (APS) identifiers of the tool enabled APS are indicated in a picture header in response to slices associated with the picture header referring to the plurality of APS identifiers. In some embodiments of method 2300, a length of a list of the plurality of APS identifiers is dependent on a number of the slices in a picture. In some embodiments of method 2300, the format rule specifies that a syntax element is included in a slice header, and wherein the syntax element indicates which tool enabled APS from the plurality of APS is to be used for a current slice. In some embodiments of method 2300, the tool, when enabled, performs a switching between samples of a reshaped domain and of an original domain for the video region in case that the video region is from a luma component, or wherein the tool, when enabled, performs a scaling of a chroma residual of the video region in case that the video region is from a chroma component.

[00236] FIG. 24 is a flowchart for an example method (2400) of video processing. Operation 2402 includes performing a conversion between a video region of a video and a bitstream of the video, wherein the conversion conforms to a size rule, and wherein, the size rule specifies a maximum size of the video region coded using a transform-skip (TS) coding scheme or a block-based delta pulse code modulation (BDPCM) coding scheme, or a maximum size of a transform block for the video region according to a coding characteristic of the video region.

[00237] In some embodiments of method 2400, the size rule specifies that the maximum size of the video region is less than or equal to a coding tree block (CTB) size. In some embodiments of method 2400, the size rule specifies that a maximum allowed width and height for the TS coding or decoding scheme and/or for the BDPCM coding or decoding scheme for a luma block is less than or equal to the CTB size. In some embodiments of method 2400, the size rule specifies that a maximum allowed width and height for the TS coding or decoding scheme and/or for the BDPCM coding or decoding scheme for a chroma block is less than or equal to the CTB size divided by subWidthC and/or the CTB size divided by subHeightC, wherein the subWidthC and the subHeightC are dependent on chroma format of the video. In some embodiments of method 2400, the size rule specifies that a maximum allowed width for the TS coding or decoding

scheme and/or for the BDPCM coding or decoding scheme for a chroma block is less than or equal to the CTB size divided by subWidthC and/or the CTB size divided by subHeightC.

[00238] In some embodiments of method 2400, the size rule specifies that a maximum allowed height for the TS coding or decoding scheme and/or for the BDPCM coding or decoding scheme for a chroma block is less than or equal to the CTB size divided by subWidthC and/or the CTB size divided by subHeightC. In some embodiments of method 2400, a first value is equal to $\log_2_transform_skip_max_size_minus2$ plus 2, wherein the first value is less than or equal to a second value for CtbLog2SizeY, and wherein $\log_2_transform_skip_max_size_minus2$ plus 2 is equal to $\log_2$ of a maximum block size used for the TS coding scheme or for the TS decoding scheme.

[00239] In some embodiments of method 2400, a first value that describes a maximum value for $\log_2_transform_skip_max_size_minus2$ is equal to a second value for CtbLog2SizeY minus 2 wherein $\log_2_transform_skip_max_size_minus2$ plus 2 is equal to $\log_2$ of a maximum block size used for the TS coding scheme or for the TS decoding scheme. In some embodiments of method 2400, the size rule specifies that the maximum size of the video region that is a transform skip block is a minimum of (CtbSizeY, $1 \ll (\log_2_transform_skip_max_size_minus2 + 2)$), wherein $\ll$ indicates a left shift operation, wherein CtbSizeY is the CTB size, and wherein $\log_2_transform_skip_max_size_minus2$ plus 2 is equal to $\log_2$ of a maximum block size used for the TS coding scheme or for the TS decoding scheme. In some embodiments of method 2400, the size rule specifies that the maximum size of the video region that is a transform skip block is $1 \ll \text{Min}(\log_2_transform_skip_max_size_minus2 + 2, \text{CtbLog2SizeY})$, wherein $\ll$ indicates a left shift operation, wherein CtbSizeY is the CTB size, and wherein $\log_2_transform_skip_max_size_minus2$ plus 2 is equal to $\log_2$ of a maximum block size used for the TS coding scheme or for the TS decoding scheme. In some embodiments of method 2400, the size rule specifies that the maximum size for the TS coding or decoding scheme and/or for the BDPCM coding or decoding scheme for a chroma block is less than or equal to a maximum transform block size for the chroma block.

[00240] In some embodiments of method 2400, a maximum allowed width and height for the TS coding or decoding scheme and/or for the BDPCM coding or decoding scheme for the chroma block is less than or equal to a maximum transform block size for a luma block divided by SubWidthC. In some embodiments of method 2400, a maximum allowed width for the TS

coding or decoding scheme and/or for the BDPCM coding or decoding scheme for the chroma block is less than or equal to a maximum transform block size for a luma block divided by SubWidthC. In some embodiments of method 2400, a maximum allowed height for the TS coding or decoding scheme and/or for the BDPCM coding or decoding scheme for the chroma block is less than or equal to a maximum transform block size for a luma block divided by subHeightC. In some embodiments of method 2400, the size rule specifies that the maximum size for the video region that is a transform skip block is indicated in the bitstream with a binary syntax element. In some embodiments of method 2400, the size rule specifies that the maximum size for the video region that is a transform skip block and/or a maximum allowed transform block size is greater than or equal to a minimum coding block size.

[00241] FIG. 25 is a flowchart for an example method (2500) of video processing. Operation 2502 includes performing a conversion between a video region of a video and a bitstream of the video, wherein the bitstream conforms to a format rule that specifies that a minimum allowed coding block size used during the conversion controls whether and/or how a maximum allowed transform block size is included in the bitstream.

[00242] In some embodiments of method 2500, the format rule specifies that the minimum allowed transform block size is greater than or equal to the minimum allowed coding block size. In some embodiments of method 2500, a value of `sps_max_luma_transform_size_64_flag` included in the bitstream is equal to 1 in response to the minimum allowed coding block size being equal to 64. In some embodiments of method 2500, a value of `sps_max_luma_transform_size_64_flag` is not included in the bitstream and is inferred to be 1 in response to the minimum allowed coding block size being equal to 64.

[00243] FIG. 26 is a flowchart for an example method (2600) of video processing. Operation 2602 includes performing a conversion between a video region of a video and a bitstream of the video, wherein the bitstream conforms to a format rule that specifies that a minimum allowed coding block size used during the conversion controls whether and/or how an indication of a maximum size of a video region for coding or decoding using a transform-skip (TS) scheme or a block-based delta pulse code modulation (BDPCM) scheme is included in the bitstream.

[00244] In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is greater than or equal to the minimum allowed coding block size. In some embodiments of method 2600, the format rule specifies that the maximum size of the video

region is less than or equal to W , wherein W is an integer. In some embodiments of method 2600, W is 32. In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is greater than or equal to the minimum allowed coding block size, and wherein the format rule specifies that the maximum size of the video region is less than or equal to W . In some embodiments of method 2600, the format rule specifies that the minimum allowed coding block size is less than or equal to X when the TS scheme and/or the BDPCM scheme is enabled. In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is greater than or equal to the minimum allowed coding block size when the TS scheme and/or the BDPCM scheme is enabled.

[00245] In some embodiments of method 2600, the format rule specifies that $\log_2$ of the maximum size is greater than or equal to $\log_2$ of a minimum luma coding block size when the TS scheme is enabled. In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is a maximum of either the maximum size or the minimum allowed coding block size. In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is a minimum of a first value and a second value, the first value is an integer W , and the second value is the maximum of either the maximum size or the minimum allowed coding block size. In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is a maximum of either the maximum size or the minimum allowed coding block size when the TS scheme and/or the BDPCM scheme is enabled. In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is a minimum of a first value and a second value when the TS scheme and/or the BDPCM scheme is enabled, the first value is an integer W , and the second value is the maximum of either the maximum size or the minimum allowed coding block size.

[00246] In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is included in the bitstream based on the minimum allowed coding block size. In some embodiments of method 2600, the format rule specifies that the maximum size of the video region is indicated by including a difference between $\log_2$ of the maximum size of the video region and $\log_2$ of the minimum allowed coding block size in the bitstream. In some embodiments of method 2600, the format rule specifies the following: $\text{MaxTsSize} = 1 \ll (\text{MinCbLog2SizeY} + \log_2_diff_max_trasform_skip_min_coding_block)$, $\ll$ indicates a left shift operation, wherein MaxTsSize is the maximum size of the video region, wherein

MinCbLog2SizeY is log2 of a minimum coding unit size, and wherein

log2_diff_max_transform_skip_min_coding_block is log2 of the difference between the maximum size of the video region and the minimum allowed coding block size in the bitstream.

[00247] In some embodiments of method 2600, the format rule specifies the following:

[00248] $\text{MaxTsSize} = \min(W, 1 \ll (\text{MinCbLog2SizeY} + \text{log2_diff_max_transform_skip_min_coding_block}))$, wherein $\ll$ indicates a left shift operation, wherein W is an integer, wherein MaxTsSize is the maximum size of the video region, wherein MinCbLog2SizeY is log2 of a minimum coding unit size, and wherein log2_diff_max_transform_skip_min_coding_block is log2 of the difference between the maximum size of the video region and the minimum allowed coding block size in the bitstream.

[00249] FIG. 27 is a flowchart for an example method (2700) of video processing. Operation 2702 includes performing a conversion between a video unit of a video and a bitstream of the video according to a format rule, wherein the bitstream conforms to the format rule that specifies that a minimum allowed coding block size used during the conversion controls whether and/or how a field indicative of whether a transform-skip (TS) scheme or a block-based delta pulse code modulation (BDPCM) scheme is enabled or disabled is included in the bitstream.

[00250] In some embodiments of method 2700, the format rule specifies that the field is inferred to be zero and is not included in the bitstream in response to the minimum allowed coding block size being equal to 64. In some embodiments of method 2700, the format rule specifies that the field is inferred to be zero and is not included in the bitstream in response to the minimum allowed coding block size being greater than a maximum allowed size for the TS coding scheme and/or the BDPCM coding scheme.

[00251] FIG. 28 is a flowchart for an example method (2800) of video processing. Operation 2802 includes performing a conversion between a video unit of a video and a bitstream of the video according to a format rule, wherein the bitstream conforms to the format rule that specifies that a minimum allowed coding block size used during the conversion controls whether and/or how a field carrying information regarding applicability of a coding tool in the conversion is included at a video region level in the bitstream.

[00252] In some embodiments of method 2800, the video region level corresponds to a sequence parameter set or a picture parameter set or a picture header or a slice header. In some embodiments of method 2800, the format rule specifies that the field for the coding tool is

inferred to be not used and is not included in the bitstream in response to the minimum allowed coding block size being greater than an integer T. In some embodiments of method 2800, the format rule specifies that the field for the coding tool indicates that the coding tool is not used and is included in the bitstream in response to the minimum allowed coding block size being greater than an integer T. In some embodiments of method 2800, the coding tool includes a combined inter-intra prediction (CIIP) tool. In some embodiments of method 2800, the coding tool includes a multiple transform selection (MTS) coding tool. In some embodiments of method 2800, the coding tool includes a segment block transform (SBT) coding tool. In some embodiments of method 2800, the coding tool includes a Symmetric Motion Vector Difference (SMVD) coding tool. In some embodiments of method 2800, the coding tool includes a bi-directional optical flow (BDOF) coding tool. In some embodiments of method 2800, the coding tool includes an affine prediction coding tool.

[00253] In some embodiments of method 2800, the coding tool includes a Prediction Refine with Optical Flow (PROF) coding tool. In some embodiments of method 2800, the coding tool includes a Decoder-side Motion Vector Refinement (DMVR) coding tool. In some embodiments of method 2800, the coding tool includes a Bi-prediction with CU-level Weights (BCW) coding tool. In some embodiments of method 2800, the coding tool includes a Merge with Motion Vector Difference (MMVD) coding tool. In some embodiments of method 2800, the coding tool includes a Geometric Partitioning Mode (GPM) coding tool. In some embodiments of method 2800, the coding tool includes an Intra Block Copy (IBC) coding tool. In some embodiments of method 2800, the coding tool includes a palette coding tool. In some embodiments of method 2800, the coding tool includes an Adaptive Color Transform (ACT) coding tool. In some embodiments of method 2800, the coding tool includes a Joint Cb-Cr Residue coding (JCCR) coding tool. In some embodiments of method 2800, the coding tool includes a Cross-Component Linear Model prediction (CCLM) coding tool. In some embodiments of method 2800, the coding tool includes a Multiple Reference Line (MRL) coding tool. In some embodiments of method 2800, the coding tool includes a Matrix-based Intra-Prediction (MIP) coding tool. In some embodiments of method 2800, the coding tool includes an Intra Subpartitions Prediction (ISP) coding tool.

[00254] FIG. 29 is a flowchart for an example method (2900) of video processing. Operation 2902 includes determining, for a conversion between a video region of a video and a bitstream of

the video, that due a dual-tree coding scheme being enabled for the video region, that splitting schemes used for luma component and a chroma component of the video have different minimum allowed block sizes. Operation 2904 includes performing the conversion based on the determining.

[00255] In some embodiments of method 2900, the splitting schemes include binary-tree splitting. In some embodiments of method 2900, a first minimum allowed block size for the binary-tree splitting for the luma component is $\text{MinBtSizeY} = 1 \ll \text{MinBtLog2SizeY}$, wherein a second minimum allowed block size for the binary-tree splitting for the chroma component is $\text{MinBtSizeC} = 1 \ll \text{MinBtLog2SizeC}$, wherein $\ll$ indicates a left shift operation, wherein MinBtSizeY is the first minimum allowed block size, wherein MinBtLog2SizeY is $\log_2$ of MinBtSizeY , wherein MinBtSizeC is the first minimum allowed block size, wherein MinBtLog2SizeC is $\log_2$ of MinBtSizeC , and wherein MinBtLog2SizeY is not equal to MinBtLog2SizeC . In some embodiments of method 2900, MinBtLog2SizeY is signaled by being predicted based on a minimum coding unit size. In some embodiments of method 2900, a difference between MinBtLog2SizeY and $\log_2$ of the minimum coding unit size is included in the bitstream. In some embodiments of method 2900, MinBtLog2SizeC is signaled by being predicted based on a minimum coding unit size.

[00256] In some embodiments of method 2900, a difference between MinBtLog2SizeC and $\log_2$ of the minimum coding unit size is included in the bitstream. In some embodiments of method 2900, the splitting schemes include ternary-tree splitting. In some embodiments of method 2900, a minimum allowed coding block size MinCbSizeY is equal to $1 \ll \text{MinCbLog2SizeY}$, wherein $\ll$ indicates a left shift operation, and wherein MinCbLog2SizeY is $\log_2$ of a minimum coding unit size. In some embodiments of method 2900, a first minimum allowed block size for the ternary-tree splitting for the luma component is $\text{MinTtSizeY} = 1 \ll \text{MinTtLog2SizeY}$, wherein a second minimum allowed block size for the ternary-tree splitting for the chroma component is $\text{MinTtSizeC} = 1 \ll \text{MinTtLog2SizeC}$, wherein $\ll$ indicates a left shift operation, wherein MinTtSizeY is the first minimum allowed block size, wherein MinTtLog2SizeY is $\log_2$ of MinTtSizeY , wherein MinTtSizeC is the first minimum allowed block size, wherein MinTtLog2SizeC is $\log_2$ of MinTtSizeC , and wherein MinTtLog2SizeY is not equal to MinTtLog2SizeC . In some embodiments of method 2900, MinTtLog2SizeY is signaled by being predicted based on a minimum coding unit size. In some embodiments of method 2900, a

difference between MinTtLog2SizeY and $\log_2$ of the minimum coding unit size is included in the bitstream. In some embodiments of method 2900, MinTtLog2SizeC is signaled by being predicted based on a minimum coding unit size. In some embodiments of method 2900, a difference between MinTtLog2SizeC and $\log_2$ of the minimum coding unit size is included in the bitstream.

[00257] FIG. 30 is a flowchart for an example method (3000) of video processing. Operation 3002 includes determining, for a conversion between a video region of a video and a bitstream of the video, a maximum number of sub-block based merge candidates allowed for the video region based on a rule. Operation 3004 includes performing the conversion based on the determining, wherein the rule specifies that maximum number of sub-block based merge candidates used during the conversion is derivable as a sum of a first variable and a second variable, wherein the first variable is equal to zero responsive to an affine prediction being disabled, and wherein the second variable is based on whether a sub-block based temporal motion vector prediction (sbTMVP) is enabled.

[00258] In some embodiments of method 3000, the first variable represents a number of allowed affine merge candidates. In some embodiments of method 3000, the second variable is set to $(\text{sps_sbtmvp_enabled_flag} \ \&\& \ \text{ph_temporal_mvp_enable_flag})$. In some embodiments of method 3000, the first variable is derived as $K-S$, wherein S is a value signaled by a syntax element and K is a fixed number. In some embodiments of method 3000, maximum number of sub-block based merge candidates is $\text{MaxNumSubblockMergeCand}$, wherein the first variable is $\text{five_minus_max_num_affine_merge_cand}$, and wherein $\text{MaxNumSubblockMergeCand} = 5 - \text{five_minus_max_num_affine_merge_cand} + (\text{sps_sbtmvp_enabled_flag} \ \&\& \ \text{ph_temporal_mvp_enable_flag})$. In some embodiments of method 3000, maximum number of sub-block based merge candidates is $\text{MaxNumSubblockMergeCand}$, wherein the first variable is $\text{four_minus_max_num_affine_merge_cand}$, and wherein $\text{MaxNumSubblockMergeCand} = 4 - \text{four_minus_max_num_affine_merge_cand} + (\text{sps_sbtmvp_enabled_flag} \ \&\& \ \text{ph_temporal_mvp_enable_flag})$. In some embodiments of method 3000, maximum number of sub-block based merge candidates is $\text{MaxNumSubblockMergeCand}$, and wherein $\text{MaxNumSubblockMergeCand} = \text{Min}(W, \text{MaxNumSubblockMergeCand})$, wherein W is a fixed number. In some embodiments of method 3000, the first variable is conditionally signaled in the bitstream. In some embodiments of method 3000, the first variable is signaled in the bitstream in

response to `sps_affine_enabled_flag` being equal to 1. In some embodiments of method 3000, `five_minus_max_num_affine_merge_cand` is inferred to be K in response to the first variable being absent from the bitstream.

[00259] FIG. 31 is a flowchart for an example method (3100) of video processing. Operation 3102 includes performing a conversion between a video region of a video and a bitstream of the video region by conforming a processing rule that is applicable to the conversion due to the video being a 4:2:2 video or a 4:4:4 video, wherein the processing rule defines that chroma and luma are aligned for one or more of the following: (a) a number of pixel lines between a virtual boundary of an adaptive loop filtering (ALF) operation and a bottom boundary of a coding tree block (CTB); or (b) a filter strength of a filter applied to a row between the virtual boundary of the ALF operation and the bottom boundary of the CTB; or (c) a padding method used for padding luma and chroma samples in a same row.

[00260] In some embodiments of method 3100, the processing rule specifies that a vertical position and/or a horizontal position are aligned for the virtual boundary between chroma and luma components for the ALF operation. In some embodiments of method 3100, the processing rule specifies that a vertical position ($vbPosC$) of the virtual boundary for the chroma component is equal to $(CtbSizeY - S) / SubHeightC$ in response to a vertical position ($vbPosY$) of the virtual boundary for the luma component being equal to $(CtbSizeY - S)$. In some embodiments of method 3100, the processing rule specifies that a horizontal position ($vbPosC$) of the virtual boundary for the chroma component is equal to $(CtbSizeY - S) / SubWidthC$ in response to a horizontal position ($vbPosY$) of the virtual boundary for the luma component being equal to $(CtbSizeY - S)$. In some embodiments of method 3100, $CtbSizeY$ is a luma CTB size of each coding tree unit (CTU), wherein S is an integer, wherein $SubHeightC$ and/or $SubWidthC$ have a value of 1 or 2. In some embodiments of method 3100, the processing rule specifies that the padding method used for padding K lines and/or H columns of a chroma component near a vertical position and/or a horizontal position of the virtual boundary is aligned with the padding method used for the luma component, and wherein Yc is a vertical or horizontal chroma sample position. In some embodiments of method 3100, Yc is equal to a vertical or horizontal position of the virtual boundary for the chroma component.

[00261] In some embodiments of method 3100, an above or left K unavailable lines are padded, or wherein K lines of the corresponding K unavailable lines below or right of the virtual

boundary are padded, or wherein K lines of the corresponding K available lines below or right of the virtual boundary are padded. In some embodiments of method 3100, the above K unavailable lines are padded using the virtual boundary. In some embodiments of method 3100, a first row of samples is located immediately above the virtual boundary, wherein samples of a second row immediately above the first row is set equal to samples of the first row, and wherein a sample of a third row that is two rows above the first row is set equal to a sample of the first row. In some embodiments of method 3100, the K lines are padded using the virtual boundary. In some embodiments of method 3100, a first row of samples is located immediately below the virtual boundary, wherein samples of a second row located immediately below the first row are set equal to samples of the first row, and wherein a sample of a third row that is two rows below the first row is set equal to a sample of the first row.

[00262] In some embodiments of method 3100, a below or right K unavailable lines are padded, or wherein K lines of the corresponding K unavailable lines above or left of the virtual boundary are padded, or wherein K lines of the corresponding K available lines above or left of the virtual boundary are padded. In some embodiments of method 3100, Y_c is equal to a vertical or horizontal position of the virtual boundary for the chroma component minus M, wherein M is an integer. In some embodiments of method 3100, a below or right K unavailable lines are padded, and wherein K lines of the corresponding K unavailable lines above or left of the virtual boundary are padded, or wherein K lines of the corresponding K available lines above or left of the virtual boundary are padded. In some embodiments of method 3100, the below K unavailable lines are padded using the virtual boundary in response to M being equal to 1. In some embodiments of method 3100, a first row of samples is located immediately below the virtual boundary, wherein samples of a second row located immediately below the first row are set equal to samples of the first row, and wherein a sample of a third row that is two rows below the first row is set equal to a sample of the first row. In some embodiments of method 3100, the corresponding K unavailable lines or the corresponding K available lines are padded using the virtual boundary in response to M being equal to 1. In some embodiments of method 3100, a first row of samples is located immediately above the virtual boundary, wherein samples of a second row immediately above the first row is set equal to samples of the first row, and wherein a sample of a third row that is two rows above the first row is set equal to a sample of the first row.

[00263] In some embodiments of method 3100, the below K unavailable lines are padded using a downmost line above the virtual boundary in response to M being greater than or equal to 2. In some embodiments of method 3100, a first row of samples is located two rows above the virtual boundary, wherein a sample of a second row located immediately below the virtual boundary is equal to a sample from the first row. In some embodiments of method 3100, the corresponding K unavailable lines or the corresponding K available lines are padded using a corresponding line of a downmost line above the virtual boundary in response to M being greater than or equal to 2. In some embodiments of method 3100, a first row of samples is located two rows above the virtual boundary, wherein a sample of a second row located two rows above the first row is equal to a sample from the first row. In some embodiments of method 3100, Y_c is equal to a vertical or horizontal position of the virtual boundary for the chroma component plus N, wherein N is an integer. In some embodiments of method 3100, an above or left K unavailable lines are padded, and wherein K lines of the corresponding K unavailable lines below or right of the virtual boundary are padded, or wherein K lines of the corresponding K available lines below or right of the virtual boundary are padded. In some embodiments of method 3100, the above K unavailable lines are padded using the virtual boundary.

[00264] In some embodiments of method 3100, a first row of samples is located two rows below the virtual boundary, and wherein a sample of a second row located immediately above the virtual boundary is equal to a sample from the first row. In some embodiments of method 3100, the corresponding K unavailable lines or the corresponding K available lines are padded using the virtual boundary. In some embodiments of method 3100, a first row of samples is located two rows below the virtual boundary, and wherein a sample of a second row located two rows below the first row is equal to a sample from the first row. In some embodiments of method 3100, the processing rule specifies that a first filter strength of M lines or N columns of a chroma component near a vertical position or a horizontal position of the virtual boundary is aligned with a second filter strength used for the luma component, wherein the second filter strength for the luma component and the first filter strength of the chroma component are controlled by alfShiftY and alfShiftC , respectively, and wherein Y_c is a vertical or horizontal chroma sample position. In some embodiments of method 3100, when $Y_c = \text{vbPosC} - M$, $\text{alfShiftC} = T1$, and when $Y_c \neq \text{vbPosC} - M$, $\text{alfShiftC} = T2$, wherein vbPosC is a horizontal position of the virtual boundary for the chroma component. In some embodiments of method 3100, $T1 = 10$, and $T2 =$

7. In some embodiments of method 3100, when $Y_c = vbPosC + M$ (e.g., $M = 0, 1$), $alfShiftC = T1$, and when $Y_c \neq vbPosC + M$, $alfShiftC = T2$, wherein $vbPosC$ is a horizontal position of the virtual boundary for the chroma component. In some embodiments of method 3100, $T1 = 10$, and $T2 = 7$.

[00265] In some embodiments of methods 2300-3100, the performing the conversion comprising encoding the video into the bitstream. In some embodiments of methods 2300-3100, the performing the conversion comprises generating the bitstream from the video, and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium. In some embodiments of methods 2300-3100, the performing the conversion comprises decoding the video from the bitstream. In some embodiments, a video decoding apparatus comprising a processor configured to implement methods 2300-3100 and operations of the related embodiments. In some embodiments, a video encoding apparatus comprising a processor configured to implement methods 2300-3100 and operations of the related embodiments. In some embodiments, a computer program product having computer instructions stored thereon, the instructions, when executed by a processor, causes the processor to implement methods 2300-3100 and operations of the related embodiments. In some embodiments, a non-transitory computer-readable recording medium storing a bitstream of a video which is generated by a method performed by a video processing apparatus, wherein the method comprises: generating the bitstream from the video region of the video, wherein the bitstream conforms to a format rule, wherein the format rule specifies that information about applicability of a tool to the conversion is indicated in the bitstream at a video slice level and/or at a video picture level, and wherein the tool maps luma samples to particular values and selectively applies a scaling operation to the values of chroma samples. In some embodiments, a non-transitory computer-readable storage medium storing instructions that cause a processor to implement methods 2300-3100 and operations of the related embodiments. In some embodiments, a method of bitstream generation, comprising: generating a bitstream of a video according to methods 2300-3100 and operations of the related embodiments, and storing the bitstream on a computer-readable program medium. In some embodiments, a method, an apparatus, a bitstream generated according to a disclosed method or a system described in the present document.

[00266] 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 representation 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. Furthermore, during conversion, a decoder may parse a bitstream with the knowledge that some fields may be present, or absent, based on the determination, as is described in the above solutions. Similarly, an encoder may determine that certain syntax fields are or are not to be included and generate the coded representation accordingly by including or excluding the syntax fields from the coded representation.

[00267] 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.

[00268] 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.

[00269] 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).

[00270] 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.

[00271] 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.

[00272] 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.

[00273] 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

1. A method of video processing, comprising:
performing a conversion between a video unit of a video and a bitstream of the video,
wherein the bitstream conforms to a format rule, and
wherein the format rule specifies that a syntax element indicating whether an in-loop filtering operation is performed across slice boundaries is selectively included in the bitstream responsive to whether or how the video unit is partitioned into slices.
2. The method of claim 1, wherein the video unit comprises a picture.
3. The method of claim 1, wherein the video unit comprises a sub-picture.
4. The method of any of claims 1 to 3, wherein the syntax element is in a picture parameter set.
5. The method of claim 1, wherein the rule specifies that the syntax element is not included in the bitstream in response to the video unit being partitioned into only one slice.
6. The method of claim 1, wherein the rule specifies that the syntax element is not included in the bitstream in response to each sub-picture of the video unit being partitioned into only one slice.
7. The method of claim 1, wherein the rule specifies that the syntax element is included in the bitstream in response to each sub-picture of the video unit being partitioned into only one slice.
8. The method of claim 7, wherein the rule specifies that the syntax element is included in the bitstream in response to a first flag being equal to 1 which specifies that each sub-picture consists of one and only one rectangular slice.
9. The method of claim 8, wherein the first flag is `single_slice_per_subpic_flag`.
10. The method of claim 8, wherein the first flag is in a picture parameter set.

11. The method of claim 1, wherein the rule specifies that the syntax element is included in the bitstream in response to the video unit comprising a picture that is partitioned in a non-rectangular shape.
12. The method of claim 11, wherein the rule specifies that the syntax element is included in the bitstream in response to a second flag being equal to 0 which specifies that a raster-san slice mode is in use for each picture.
13. The method of claim 12, wherein the raster-san slice mode is a non-rectangular slice mode.
14. The method of claim 12, wherein the second flag is `rect_slice_flag`.
15. The method of claim 12, wherein the second flag is in a picture parameter set.
16. The method of claim 1, wherein the rule specifies that the syntax element is not included in the bitstream in response to the video unit comprising a picture that is partitioned in a rectangular shape and a number of slices of the video unit is equal to one.
17. The method of claim 1, wherein the rule specifies that the syntax element is included in the bitstream in response to the bitstream including a value of zero for a syntax element that indicates whether there is one slice per picture as a constraint.
18. The method of claim 1, wherein the rule specifies that the syntax element is required to be equal to zero and is included in the bitstream in response to the bitstream including a value of one for a syntax element that indicates whether there is one slice per picture as a constraint.
19. The method of any of claims 1 to 18, wherein the in-loop filtering operation includes at least one of a deblocking filter operation, a sample adaptive offset operation, or an adaptive loop filter operation.

20. The method of any of claims 1 to 19, wherein the performing the conversion comprising encoding the video into the bitstream.
21. The method of any of claims 1 to 19, wherein the performing the conversion comprises generating the bitstream from the video, and the method further comprises storing the bitstream in a non-transitory computer-readable recording medium.
22. The method of any of claims 1 to 19, wherein the performing the conversion comprises decoding the video from the bitstream.
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 recording medium storing a bitstream of a video which is generated by a method performed by a video processing apparatus, wherein the method comprises:
generating the bitstream from a video unit of the video;
wherein the bitstream conforms to a format rule, and
wherein the format rule specifies that a syntax element indicating whether an in-loop filtering operation is performed across slice boundaries is selectively included in the bitstream responsive to whether or how the video unit is partitioned into slices.
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.

28. A method of bitstream generation, comprising:
generating a bitstream of a video according to a method recited in any of claims 1 to 22, and
storing the bitstream on a computer-readable program medium.
29. A method, an apparatus, a bitstream generated according to a disclosed method or a system
described in the present document.