MAPPING RESTRICTION FOR INTRA-BLOCK COPY VIRTUAL BUFFER

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] Under the applicable patent law and/or rules pursuant to the Paris Convention, this application is made to timely claim the priority to and benefits of International Patent Application No. PCT/CN2019/097742, filed on July 25, 2019; International Patent Application No. PCT/CN2019/109849, filed on October 7, 2019; and International Patent Application No. PCT/CN2019/122606, filed on December 3, 2019. For all purposes under the law, the entire disclosure of the aforementioned applications is incorporated by reference as part of the disclosure of this application.

TECHNICAL FIELD

[0002] This patent document relates to video coding techniques, devices and systems.

BACKGROUND

[0003] In spite of the advances in video compression, digital video still 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

[0004] Devices, systems and methods related to digital video coding, and specifically, to general virtual buffers for intra block copy (IBC). The described methods may be applied to both the existing video coding standards (e.g., High Efficiency Video Coding (HEVC)) and future video coding standards or video codecs.

[0005] In one representative aspect, the disclosed technology may be used to provide a method for video processing. This method includes determining, for a conversion between a current video block of a video picture of a video and a coded representation of the video, a number of reference samples in a reference region of the video picture used for predicting the current video block, based on a rule, wherein the rule specifies that the number of reference samples is limited to a certain range; and performing, based on the determining, the conversion.

[0006] In another example aspect, a method of video processing is disclosed. The method

includes determining, for a conversion between a current video block of a video picture of a video and a coded representation of the video, availability of one or more reference samples of a reference block used for predicting the current video block based on a rule, wherein the rule specifies to use a granularity at a virtual pipeline data unit (VPDU) level in the determining; and performing, based on the determining, a conversion between the current block and a coded representation of the current block.

[0007] In yet another aspect, another method of video processing is disclosed. The method includes performing a conversion between a current video block of a video picture of a video and a coded representation of the video according to a rule, wherein the current video block is represented in the coded representation using a residual of prediction from reference samples from a reference region in the video pictures, and wherein the rule specifies that sample values in the reference region are reset at beginning of processing each virtual pipeline data unit (VPDU) during the conversion based on a position of a current VPDU of the current video block.

[0008] In yet another aspect, another method of video processing is disclosed. The method includes determining, for a current video block of a video picture of a video and a coded representation of the video, that samples from a current virtual pipeline data unit (VPDU) and three additional recently processed VPDUs in the video picture are available as reference samples for generating a prediction of the current video block due to the current video block having a size larger than a size of 64x64 luma samples; and performing the conversion based on the determining.

[0009] In yet another aspect, another method of video processing is disclosed. The method includes performing a conversion between a current video block of a video picture of a video and a coded representation of the video, and wherein, the conversion comprises using, a count of a number of available current virtual pipeline data units (VPDUs) that are mapped to a reference region of the video picture from which one or more reference samples are used for predicting the current video block.

[0010] In yet another representative aspect, the above-described method is embodied in the form of processor-executable code and stored in a computer-readable program medium.

[0011] In yet another representative aspect, a device that is configured or operable to perform the above-described method is disclosed. The device may include a processor that is programmed to implement this method.

[0012] In yet another representative aspect, a video decoder apparatus may implement a method as described herein.

[0013] The above and other aspects and features of the disclosed technology are described in greater detail in the drawings, the description and the claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] FIG. 1 shows an example of current picture referencing.

[0015] FIG. 2 shows an example a dynamic reference area in JVET-M0407.

[0016] FIG. 3 shows a flowchart of a decoding flow with reshaping.

[0017] FIG. 4 shows an example, for a current CU (in blue 602), block filled in red (604) is a crossing VPDU column reference block and block filled in yellow (606) is a crossing VPDU row reference block. Each large block indicates a 64x64 VPDU and green area (608) indicated reconstrued pixels that can be used for IBC reference.

[0018] FIG. 5 is a block diagram of an example of a hardware platform for implementing a visual media decoding or a visual media encoding technique described in the present document.

[0019] FIG. 6 is a block diagram of an example video processing system in which disclosed techniques may be implemented.

[0020] FIGS. 7A, 7B and 7C show flowcharts of example methods for video processing based on some implementations of the disclosed technology.

[0021] FIGS. 8A to 8D show flowcharts of example methods for video processing based on some implementations of the disclosed technology.

DETAILED DESCRIPTION

[0022] Embodiments of the disclosed technology may be applied to existing video coding standards (e.g., HEVC, H.265) and future standards to improve compression performance. Section headings are used in the present document to improve readability of the description and do not in any way limit the discussion or the embodiments (and/or implementations) to the respective sections only.

2 Video coding introduction

[0023] Due to the increasing demand of higher resolution video, video coding methods and techniques are ubiquitous in modern technology. Video codecs typically include an electronic

circuit or software that compresses or decompresses digital video, and are continually being improved to provide higher coding efficiency. A video codec converts uncompressed video to a compressed format or vice versa. There are complex relationships between the video quality, the amount of data used to represent the video (determined by the bit rate), the complexity of the encoding and decoding algorithms, sensitivity to data losses and errors, ease of editing, random access, and end-to-end delay (latency). The compressed format usually conforms to a standard video compression specification, e.g., the High Efficiency Video Coding (HEVC) standard (also known as H.265 or MPEG-H Part 2), the Versatile Video Coding standard to be finalized, or other current and/or future video coding standards.

[0024] Video coding standards have evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 and H.263, ISO/IEC produced MPEG-1 and MPEG-4 Visual, and the two organizations jointly produced the H.262/MPEG-2 Video and H.264/MPEG-4 Advanced Video Coding (AVC) and H.265/HEVC standards. Since H.262, the video coding standards are based on the hybrid video coding structure wherein temporal prediction plus transform coding are utilized. To explore the future video coding technologies beyond HEVC, Joint Video Exploration Team (JVET) was founded by VCEG and MPEG jointly in 2015. Since then, many new methods have been adopted by JVET and put into the reference software named Joint Exploration Model (JEM). In April 2018, the Joint Video Expert Team (JVET) between VCEG (Q6/16) and ISO/IEC JTC1 SC29/WG11 (MPEG) was created to work on the VVC standard targeting at 50% bitrate reduction compared to HEVC.

2.1 Inter prediction in HEVC/H.265

[0025] Each inter-predicted PU has motion parameters for one or two reference picture lists. Motion parameters include a motion vector and a reference picture index. Usage of one of the two reference picture lists may also be signaled using inter_pred_idc. Motion vectors may be explicitly coded as deltas relative to predictors.

[0026] When a CU is coded with skip mode, one PU is associated with the CU, and there are no significant residual coefficients, no coded motion vector delta or reference picture index. A merge mode is specified whereby the motion parameters for the current PU are obtained from neighboring PUs, including spatial and temporal candidates. The merge mode can be applied to any inter-predicted PU, not only for skip mode. The alternative to merge mode is the explicit

transmission of motion parameters, where motion vector (to be more precise, motion vector differences (MVD) compared to a motion vector predictor), corresponding reference picture index for each reference picture list and reference picture list usage are signaled explicitly per each PU. Such a mode is named Advanced motion vector prediction (AMVP) in this disclosure.

[0027] When signaling indicates that one of the two reference picture lists is to be used, the PU is produced from one block of samples. This is referred to as 'uni-prediction'. Uni-prediction is available both for P-slices and B-slices.

[0028] When signaling indicates that both of the reference picture lists are to be used, the PU is produced from two blocks of samples. This is referred to as 'bi-prediction'. Bi-prediction is available for B-slices only.

[0029] The following text provides the details on the inter prediction modes specified in HEVC. The description will start with the merge mode.

2.2 Current Picture Referencing

[0030] Current Picture Referencing (CPR), or once named as Intra Block Copy (IBC) has been adopted in HEVC Screen Content Coding extensions (HEVC-SCC) and the current VVC test model (VTM-3.0). IBC extends the concept of motion compensation from inter-frame coding to intra-frame coding. As demonstrated in FIG. 1, the current block is predicted by a reference block in the same picture when CPR is applied. The samples in the reference block must have been already reconstructed before the current block is coded or decoded. Although CPR is not so efficient for most camera-captured sequences, it shows significant coding gains for screen content. The reason is that there are lots of repeating patterns, such as icons and text characters in a screen content picture. CPR can remove the redundancy between these repeating patterns effectively. In HEVC-SCC, an inter-coded coding unit (CU) can apply CPR if it chooses the current picture as its reference picture. The MV is renamed as block vector (BV) in this case, and a BV always has an integer-pixel precision. To be compatible with main profile HEVC, the current picture is marked as a "long-term" reference picture in the Decoded Picture Buffer (DPB). It should be noted that similarly, in multiple view/3D video coding standards, the inter-view reference picture is also marked as a "long-term" reference picture.

[0031] Following a BV to find its reference block, the prediction can be generated by copying the reference block. The residual can be got by subtracting the reference pixels from the original signals. Then transform and quantization can be applied as in other coding modes.

[0032] However, when a reference block is outside of the picture, or overlaps with the current block, or outside of the reconstructed area, or outside of the valid area restricted by some constrains, part or all pixel values are not defined. Basically, there are two solutions to handle such a problem. One is to disallow such a situation, e.g. in bitstream conformance. The other is to apply padding for those undefined pixel values. The following sub-sessions describe the solutions in detail.

2.3 CPR in HEVC Screen Content Coding extensions

[0033] In the screen content coding extensions of HEVC, when a block uses current picture as reference, it should guarantee that the whole reference block is within the available reconstructed area, as indicated in the following spec text:

The variables offsetX and offsetY are derived as follows:

offsetX = (ChromaArrayType = = 0) ? 0 : (mvCLX[0] & 0x7 ? 2 : 0) (8-104)

offsetY = (ChromaArrayType = = 0) ? 0 : (mvCLX[1] & 0x7 ? 2 : 0) (8-105)

It is a requirement of bitstream conformance that when the reference picture is the current picture, the luma motion vector mvLX shall obey the following constraints:

– When the derivation process for z-scan order block availability as specified in clause 6.4.1 is invoked with (xCurr, yCurr) set equal to (xCb, yCb) and the neighbouring luma location (xNbY, yNbY) set equal to (xPb + (mvLX[0] >> 2) − offsetX, yPb + (mvLX[1] >> 2) − offsetY) as inputs, the output shall be equal to TRUE.

– When the derivation process for z-scan order block availability as specified in clause 6.4.1 is invoked with (xCurr, yCurr) set equal to (xCb, yCb) and the neighbouring luma location (xNbY, yNbY) set equal to (xPb + (mvLX[0] >> 2) + nPbW − 1 + offsetX, yPb + (mvLX[1] >> 2) + nPbH − 1 + offsetY) as inputs, the output shall be equal to TRUE.

– One or both of the following conditions shall be true:

– The value of (mvLX[0] >> 2) + nPbW + xB1 + offsetX is less than or equal to 0.

– The value of (mvLX[1] >> 2) + nPbH + yB1 + offsetY is less than or equal to 0.

– The following condition shall be true:

(xPb + (mvLX[0] >> 2) + nPbSw − 1 + offsetX) / CtbSizeY − xCb / CtbSizeY <= yCb/CtbSizeY − (yPb + (mvLX[1] >> 2) + nPbSh − 1 + offsetY) / CtbSizeY (8-106)

[0034] Thus, the case that the reference block overlaps with the current block or the reference block is outside of the picture will not happen. There is no need to pad the reference or prediction block.

2.4 CPR/IBC in VVC Test Model

[0035] In the current VVC test model, i.e. VTM-3.0 design, the whole reference block should be with the current coding tree unit (CTU) and does not overlap with the current block. Thus, there is no need to pad the reference or prediction block.

[0036] When dual tree is enabled, the partition structure may be different from luma to chroma CTUs. Therefore, for the 4:2:0 colour format, one chroma block (e.g., CU) may correspond to one collocated luma region which have been split to multiple luma CUs.

[0037] The chroma block could only be coded with the CPR mode when the following conditions shall be true:

[0038] (1) each of the luma CU within the collocated luma block shall be coded with CPR mode

[0039] (2) each of the luma 4×4 block' BV is firstly converted to a chroma block's BV and the chroma block's BV is a valid BV.

[0040] If any of the two condition is false, the chroma block shall not be coded with CPR mode.

[0041] It is noted that the definition of 'valid BV' has the following constraints:

[0042] (1) all samples within the reference block identified by a BV shall be within the restricted search range (e.g., shall be within the same CTU in current VVC design).

[0043] (2) all samples within the reference block identified by a BV have been reconstructed.

2.5 CPR/IBC in JVET-L0297/JVET-M0407/JVET-M0408

[0044] In VTM3.0, the reference area for CPR/IBC is restricted to the current CTU, which is up to 128x128. JVET-L0297/JVET-M0407/JVET-M0408 present methods to dynamically change the reference area to reuse memory to store reference samples for CPR/IBC so that a CPR/IBC block can have more reference candidate while the reference buffer for CPR/IBC can be kept or reduced from one CTU.

[0045] FIG. 2 shows a method, where a block is of 64x64 and a CTU contains 4 64x64 blocks. When coding a 64x64 blocks, previous 3 64x64 blocks can be used as reference. By doing so, a decoder just needs to store 4 64x64 blocks to support CPR/IBC. The above method was adopted into VTM4.0.

[0046] Suppose that the current luma CU's position relative to the upper-left corner of the picture is (x, y) and block vector is (BVx, BVy). In the current design, if the BV is valid can be

told by that the luma position ((x+BVx)>>6<<6+(1<<7), (y+BVy)>>6<<6) has not been reconstructed and ((x+BVx)>>6<<6+(1<<7), (y+BVy)>>6<<6) is not equal to (x>>6<<6, y>>6<<6).

2.6 Virtual IBC buffer proposed in JVET-O1170

[0047] A virtual buffer concept is introduced to help describing the reference region for IBC prediction mode. For CTU size being ctbSize, we denote wIbcBuf = 128*128/ctbSize and define a virtual IBC buffer, ibcBuf, with width being wIbcBuf and height being ctbSize. Thus,

[0048] − For CTU size being 128x128, the size of ibcBuf is also 128x128.

[0049] − For CTU size being 64x64, the size of ibcBuf is 256x64.

[0050] − For CTU size being 32x32, the size of ibcBuf is 512x32.

[0051] It is noted that VPDU width and height are min(ctbSize, 64). We denote Wv = min(ctbSize, 64).

[0052] The virtual IBC buffer, ibcBuf is maintained as follows.

[0053] (1) At the beginning of decoding each CTU row, refresh the whole ibcBuf with value (-1).

[0054] (2) At the beginning of decoding a VPDU (xVPDU, yVPDU) relative to the top-left corner of the picture, set the ibcBuf[x][y] = -1, with x = xVPDU%wIbcBuf, …, xVPDU% wIbcBuf + Wv – 1; y = yVPDU%ctbSize, …, yVPDU%ctbSize + Wv – 1.

[0055] (3) After decoding a CU contains (x, y) relative to the top-left of the picture, set

[0056] ibcBuf[x % wIbcBuf][y % ctbSize] = recSample[x][y]

[0057] So a bitstream constrain can be simply described as

[0058] *It is a requirement of bitstream conformance that for a bv, ibcBuf[(x + bv[0])% wIbcBuf] [(y + bv[1]) % ctbSize] shall not be equal to -1.*

[0059] With the concept of IBC reference buffer, it also simplifies the text for the decoding process by avoid reference to the inter interpolation and motion compensation process, including subblock process.

2.7 VPDU

[0060] Virtual pipeline data units (VPDUs) are defined as non-overlapping units in a picture. In hardware decoders, successive VPDUs are processed by multiple pipeline stages at the same time. The VPDU size is roughly proportional to the buffer size in most pipeline stages, so it is

important to keep the VPDU size small. In most hardware decoders, the VPDU size can be set to maximum transform block (TB) size. However, in VVC, ternary tree (TT) and binary tree (BT) partition may lead to the increasing of VPDUs size.

[0061] In order to keep the VPDU size as 64x64 luma samples, the following normative partition restrictions (with syntax signaling modification) are applied in VTM5:

[0062] − TT split is not allowed for a CU with either width or height, or both width and height equal to 128.

[0063] − For a 128xN CU with $N \leq 64$ (i.e. width equal to 128 and height smaller than 128), horizontal BT is not allowed.

[0064] − For an Nx128 CU with $N \leq 64$ (i.e. height equal to 128 and width smaller than 128), vertical BT is not allowed.

[0065] In VVC, generally it is agreed that the width and height of a VPDU is min(64, CtbSizeY) in luma samples. So for CTB/CTU size being 64x64, 128x128 or 256x256, VPDU size is 64x64. For CTB/CTU size being 32x32, VPDU size is 32x32.

2.8 Buffer Management and Block Vector Coding for Intra Block Copy

[0066] Various IBC buffer features and details for its corresponding management are described in PCT/CN2019/093552, which is incorporated by reference.

2.9 In-loop reshaping (ILR) in JVET-M0427

[0067] The basic idea of in-loop reshaping (ILR) is to convert the original (in the first domain) signal (prediction/reconstruction signal) to a second domain (reshaped domain).

[0068] The in-loop luma reshaper is implemented as a pair of look-up tables (LUTs), but only one of the two LUTs need to be signaled as the other one can be computed from the signaled LUT. Each LUT is a one-dimensional, 10-bit, 1024-entry mapping table (1D-LUT). One LUT is a forward LUT, *FwdLUT*, that maps input luma code values Y_i to altered values Y_r: $Y_r = FwdLUT[Y_i]$. The other LUT is an inverse LUT, *InvLUT*, that maps altered code values Y_r to $\hat{Y}_i$: $\hat{Y}_i = InvLUT[Y_r]$. ($\hat{Y}_i$ represents the reconstruction values of Y_i.).

2.9.1 PWL model

[0069] Conceptually, piece-wise linear (PWL) is implemented in the following way:

[0070] Let x1, x2 be two input pivot points, and y1, y2 be their corresponding output pivot points for one piece. The output value y for any input value x between x1 and x2 can be interpolated by the following equation:

[0071] $y = ((y2-y1)/(x2-x1)) * (x-x1) + y1$

[0072] In fixed point implementation, the equation can be rewritten as:

[0073] $y = ((m * x + 2FP_PREC-1) >> FP_PREC) + c$

[0074] Herein, m is scalar, c is an offset, and FP_PREC is a constant value to specify the precision.

[0075] Note that in CE-12 software, the PWL model is used to precompute the 1024-entry FwdLUT and InvLUT mapping tables; but the PWL model also allows implementations to calculate identical mapping values on-the-fly without pre-computing the LUTs.

2.9.2 Test CE12-2

2.9.2.1 Luma reshaping

[0076] Test 2 of the in-loop luma reshaping (i.e., CE12-2 in the proposal) provides a lower complexity pipeline that also eliminates decoding latency for block-wise intra prediction in inter slice reconstruction. Intra prediction is performed in reshaped domain for both inter and intra slices.

[0077] Intra prediction is always performed in reshaped domain regardless of slice type. With such arrangement, intra prediction can start immediately after previous TU reconstruction is done. Such arrangement can also provide a unified process for intra mode instead of being slice dependent. FIG. 3 shows the block diagram of the CE12-2 decoding process based on mode.

[0078] CE12-2 also tests 16-piece piece-wise linear (PWL) models for luma and chroma residue scaling instead of the 32-piece PWL models of CE12-1.

[0079] Inter slice reconstruction with in-loop luma reshaper in CE12-2 (lighter shaded blocks indicate signal in reshaped domain: luma residue; intra luma predicted; and intra luma reconstructed)

2.9.2.2 Luma-dependent chroma residue scaling

[0080] Luma-dependent chroma residue scaling is a multiplicative process implemented with fixed-point integer operation. Chroma residue scaling compensates for luma signal interaction with the chroma signal. Chroma residue scaling is applied at the TU level. More specifically, the following applies:

[0081] − For intra, the reconstructed luma is averaged.

[0082] − For inter, the prediction luma is averaged.

[0083] The average is used to identify an index in a PWL model. The index identifies a scaling factor cScaleInv. The chroma residual is multiplied by that number.

[0084] It is noted that the chroma scaling factor is calculated from forward-mapped predicted luma values rather than reconstructed luma values.

2.9.2.3 Signaling of ILR side information

[0085] The parameters are (currently) sent in the tile group header (similar to ALF). These reportedly take 40-100 bits.

The following spec is based on version 9 of JVET-L1001.The added syntax is prefixed with "++".

In 7.3.2.1 Sequence parameter set RBSP syntax

seq_parameter_set_rbsp( ) {	**Descriptor**
sps_seq_parameter_set_id	ue(v)
…	
sps_triangle_enabled_flag	u(1)
sps_ladf_enabled_flag	u(1)
if (sps_ladf_enabled_flag) {	
sps_num_ladf_intervals_minus2	u(2)
sps_ladf_lowest_interval_qp_offset	se(v)
for(i = 0; i < sps_num_ladf_intervals_minus2 + 1; i++) {	
sps_ladf_qp_offset[i]	se(v)
sps_ladf_delta_threshold_minus1[i]	ue(v)
}	
}	
++ **sps_reshaper_enabled_flag**	u(1)
rbsp_trailing_bits( )	
}	

In 7.3.3.1 General tile group header syntax

tile_group_header( ) {	**Descriptor**
…	
if(num_tiles_in_tile_group_minus1 > 0) {	
offset_len_minus1	ue(v)
for(i = 0; i < num_tiles_in_tile_group_minus1; i++)	
entry_point_offset_minus1[i]	u(v)
}	
++ if (sps_reshaper_enabled_flag) {	
++ **tile_group_reshaper_model_present_flag**	u(1)
++ if (tile_group_reshaper_model_present_flag)	
++ tile_group_reshaper_model ( )	
++ **tile_group_reshaper_enable_flag**	u(1)

++ if (tile_group_reshaper_enable_flag && (!(qtbtt_dual_tree_intra_flag && tile_group_type == I)))	
++ **tile_group_reshaper_chr oma_residual_scale_flag**	u(1)
++ }	
byte_alignment( )	
}	

Add a new syntax table tile group reshaper model:

++tile_group_reshaper_model () {	**Descriptor**
++ **reshaper_model_min_bin_idx**	ue(v)
++ **reshaper_model_delta_max_bin_idx**	ue(v)
++ **reshaper_model_bin_delta_abs_cw_prec_minus1**	ue(v)
++ for (i = reshaper_model_min_bin_idx; i <= reshaper_model_max_bin_idx; i++) {	
++ **reshape_model_bin_delta_abs_CW [i]**	u(v)
++ if (reshaper_model_bin_delta_abs_CW[i]) > 0)	
++ **reshaper_model_bin_delta_sign_CW_flag[** i]	u(1)
++ }	
++}	

++*In General sequence parameter set RBSP semantics, add the following semantics:*

sps_reshaper_enabled**_flag** equal to 1 specifies that reshaper is used in the coded video sequence (CVS). sps_reshaper_enabled_flag equal to 0 specifies that reshaper is not used in the CVS.

++*In tile group header syntax, add the following semantics*

tile_group_reshaper_model_present_flag equal to 1 specifies tile_group_reshaper_model() is present in tile group header. tile_group_reshaper_model_present_flag equal to 0 specifies tile_group_reshaper_model() is not present in tile group header. When tile_group_reshaper_model_present_flag is not present, it is inferred to be equal to 0.

tile_group_reshaper_enabled_flag equal to 1 specifies that reshaper is enabled for the current tile group. tile_group_reshaper_enabled_flag equal to 0 specifies that reshaper is not enabled for the current tile group. When tile_group_reshaper_enable_flag is not present, it is inferred to be equal to 0.

tile_group_reshaper_chroma_residual_scale_flag equal to 1 specifies that chroma residual scaling is enabled for the current tile group. tile_group_reshaper_chroma_residual_scale_flag equal to 0 specifies that chroma residual scaling is not enabled for the current tile group. When tile_group_reshaper_chroma_residual_scale_flag is not present, it is inferred to be equal to 0.

++*Add tile_group_reshaper_model( ) syntax*

reshape_model_min_bin_idx specifies the minimum bin (or piece) index to be used in the reshaper construction process. The value of reshape_model_min_bin_idx shall be in the range of 0 to MaxBinIdx, inclusive. The value of MaxBinIdx shall be equal to 15.

reshape_model_delta_max_bin_idx specifies the maximum allowed bin (or piece) index MaxBinIdx minus the maximum bin index to be used in the reshaper construction process. The value of reshape_model_max_bin_idx is set equal to MaxBinIdx – reshape_model_delta_max_bin_idx.

reshaper_model_bin_delta_abs_cw_prec_minus1 plus 1 specifies the number of bits used for the representation of the syntax reshape_model_bin_delta_abs_CW[i].

reshape_model_bin_delta_abs_CW[i] specifies the absolute delta codeword value for the ith bin.

reshaper_model_bin_delta_sign_CW_flag[i] specifies the sign of reshape_model_bin_delta_abs_CW[i] as follows:

– If reshape_model_bin_delta_sign_CW_flag[i] is equal to 0, the corresponding variable RspDeltaCW[i] is a positive value.

– Otherwise (reshape_model_bin_delta_sign_CW_flag[i] is not equal to 0), the corresponding variable RspDeltaCW[i] is a negative value.

When reshape_model_bin_delta_sign_CW_flag[i] is not present, it is inferred to be equal to 0.

The variable RspDeltaCW[i] = (1 2*reshape_model_bin_delta_sign_CW [i]) * reshape_model_bin_delta_abs_CW [i];

The variable RspCW[i] is derived as following steps:

The variable OrgCW is set equal to (1 << $BitDepth_Y$) / (MaxBinIdx + 1).

– If reshaper_model_min_bin_idx < = i <= reshaper_model_max_bin_idx
RspCW[i] = OrgCW + RspDeltaCW[i].

– Otherwise, RspCW[i] = 0.

The value of RspCW [i] shall be in the range of 32 to 2 * OrgCW - 1 if the value of $BitDepth_Y$ is equal to 10.

The variables InputPivot[i] with i in the range of 0 to MaxBinIdx + 1, inclusive are derived as follows

InputPivot[i] = i * OrgCW

The variable ReshapePivot[i] with i in the range of 0 to MaxBinIdx + 1, inclusive, the variable ScaleCoef[i] and InvScaleCoeff[i]with i in the range of 0 to MaxBinIdx , inclusive, are derived as follows:

```
shiftY = 14
ReshapePivot[ 0 ] = 0;
for( i = 0; i <= MaxBinIdx ; i++) {
    ReshapePivot[ i + 1 ] = ReshapePivot[ i ] + RspCW[ i ]
    ScaleCoef[ i ] = ( RspCW[ i ] * (1 << shiftY) + (1 << (Log2(OrgCW) - 1))) >> (Log2(OrgCW))
    if ( RspCW[ i ] == 0 )
        InvScaleCoeff[ i ] = 0
    else
        InvScaleCoeff[ i ] = OrgCW * (1 << shiftY) / RspCW[ i ]
}
```

The variable ChromaScaleCoef[i] with i in the range of 0 to MaxBinIdx , inclusive, are derived as follows:

ChromaResidualScaleLut[64] = {16384, 16384, 16384, 16384, 16384, 16384, 16384, 8192, 8192, 8192, 8192, 5461, 5461, 5461, 5461, 4096, 4096, 4096, 4096, 3277, 3277, 3277, 3277, 2731, 2731, 2731, 2731, 2341, 2341, 2341, 2048, 2048, 2048, 1820, 1820, 1820, 1638, 1638, 1638, 1638, 1489, 1489, 1489, 1489, 1365, 1365, 1365, 1365, 1260, 1260, 1260, 1260, 1170, 1170, 1170, 1170, 1092, 1092, 1092, 1092, 1024, 1024, 1024, 1024};

shiftC = 11

- if (RspCW[i] == 0)
  ChromaScaleCoef [i] = (1 << shiftC)
- Otherwise (RspCW[i] != 0), ChromaScaleCoef[i] = ChromaResidualScaleLut[RspCW[i] >> 1]

2.9.2.4 Usage of ILR

[0086] At the encoder side, each picture (or tile group) is firstly converted to the reshaped domain. And all the coding process is performed in the reshaped domain. For intra prediction, the neighboring block is in the reshaped domain; for inter prediction, the reference blocks (generated from the original domain from decoded picture buffer) are firstly converted to the reshaped domain. Then the residuals are generated and coded to the bitstream.

[0087] After the whole picture (or tile group) finishes encoding/decoding, samples in the reshaped domain are converted to the original domain, then deblocking filter and other filters are applied.

[0088] Forward reshaping to the prediction signal is disabled for the following cases:

[0089] ○ Current block is intra-coded

[0090] ○ Current block is coded as CPR (current picture referencing, aka intra block copy, IBC)

[0091] ○ Current block is coded as combined inter-intra mode (CIIP) and the forward reshaping is disabled for the intra prediction block

3 Drawbacks of existing implementations

[0092] In the current design of IBC virtual buffer, some problems exist.

[0093] (1) How to maintain IBC virtual buffer when CTU size is larger than 128x128 is not defined.

[0094] (2) The relationship between virtual buffer size and size of reference samples is not clear.

[0095] (3) Line buffer for IBC mode and BV for CTU row may be reduced.

[0096] (4) Subpicture might be too restricted.

[0097] (5) Chroma QP table may not be designed in a right way.

4 Example methods for general virtual buffers for IBC

[0098] Denote the width and height of a VPDU by vSize, e.g., vSize = min(64, ctbSizeY), where ctbSizeY is the luma CTB/CTU width/height.

[0099] Embodiments of the presently disclosed technology overcome the drawbacks of existing implementations, thereby providing video coding with higher coding efficiencies. The methods for general virtual buffers for IBC, based on the disclosed technology, may enhance both existing and future video coding standards, is elucidated in the following examples described for various implementations. The examples of the disclosed technology provided below explain general concepts, and are not meant to be interpreted as limiting. In an example, unless explicitly indicated to the contrary, the various features described in these examples may be combined.

IBC buffer related

1. The size of IBC virtual buffer (e.g., which is used to decide whether a block vector or a mapped sample's validity) may depend on VPDU size, CTB/CTU size.
    a. In one example, the width times height of the virtual buffer may be fixed, however, the width and/height of the virtual buffer may depend on VPDU size and/or CTB/CTU size.
    b. In one example, the height of the virtual buffer may be equal to the height of CTB/CTU.
        i. Alternatively, furthermore, the the width of the virtual buffer may be set to (IBC virtual buffer size/height of CTB).
    c. In one example, the width of the virtual buffer may be equal to the width of CTB/CTU.
    d. In one example, the width of the virtual buffer may be one or multiple times of VPDU width
    e. In one example, the height of the virtual buffer may be one or multiple times of VPDU height

2. It is proposed to allocate a larger IBC virtual buffer size compared to the required memory size for IBC BV searching area.
    a. In one example, the IBC virtual buffer size may be larger than the total size of VPDU memory used for IBC.
        i. In one example, one more CTU may be allocated to IBC virtual buffer.
    b. In one example, the width of the IBC virtual buffer size may be (128*128/ctbSizeY + ctbSizeY).
    c. In one example, the width of the IBC virtual buffer size may be (128*128/ctbSizeY + ctbSizeY) and the height of the IBC virtual buffer size may be ctbSizeY.
    d. In one example, the width of the IBC virtual buffer may be (256*128/ctbSizeY). Alternatively, furthermore, the height of the IBC virtual buffer may be ctbSizeY.
    e. In one example, a larger IBC virtual buffer may be allocated for a larger CTU/CTB.
        i. In one example, when CTU size is no smaller than K (e.g., K=128), the width of the IBC virtual buffer may be (128*128/ctbSizeY + ctbSizeY). Alternatively, furthermore, the height of the IBC virtual buffer may be ctbSizeY.
        ii. In one example, when CTU size is smaller than K (e.g, K=128), the width of the IBC virtual buffer may be (128*128/ctbSizeY). Alternatively, furthermore, the height of the IBC virtual buffer may be ctbSizeY.
3. The reference block for an IBC block may be constrained to be fully within a certain VPDU row or VPDU column
    a. In one example, the reference block may be disallowed to cross different VPDU rows.
    b. In one example, the reference block may be disallowed to cross different VPDU columns.
    c. In one example, the above VPDU row or column may be relative to the picture.
    d. In one example, the above VPDU row or column may be relative to the IBC virtual buffer.

e. Alternatively, furthermore, the above methods may be invoked when the reference block pointed by a BV cross two or more CTUs/CTBs.

4. The reference block for an IBC block may cross multiple VPDUs/cross different VPDU rows/VPDU columns. However, additional operations may be required to fill in some prediction values in the reference block.
    a. In one example, some default values may be utilized to fill in some prediction values.
5. A range constrain may be applied to block vectors (BVs) and/or block vector differences (BVD) used in IBC mode.
    a. In one example, the allowed range of BV/BVDs may depend on the location of current IBC coded block, such as coordinators relative to the CTU/CTB covering current block.
    b. In one example, block vectors may be constrained in the range of $[-2^m,2^m-1]$
    c. In one example, block vector differences after precision conversion may be constrained in the range of $[-2^n,2^n-1]$
    d. In one example, block vector differences after precision conversion may be constrained in the range of $[-2^n+1,2^n-1]$
    e. In one example, block vector differences signalled in the bitstreams may be constrained in the range of $[-2^n,2^n-1]$
    f. In one example, block vector differences signalled in the bitstreams may be constrained in the range of $[-2^n+1,2^n-1]$
    g. In one example, m is set to 18 or 17 or 15.
    h. In one example, n is set to 17 or 16 or 14.
    i. In one example, m and/or may depend on the precision for BV/motion vector storage and/or the precision associated with the block vector differences.
    j. In one example, block vectors may be constrained in the same range as motion vectors used for inter prediction mode.
    k. In one example, block vector differences may be constrained in the same range as motion vector vectors used for inter prediction mode.
    l. In one example, a conformance bitstream shall satisfy that the above sub-bullet is satisfied.

i. Alternatively, a clipping process to the BV/BVD may be applied to decoded BV/BVD before BV/BVDs are utilized to encode/decode a block.

6. The number of available samples mapped to the IBC virtual buffer may be restricted.
    a. In one example, the maximum number of available samples mapped to the buffer may be smaller than the IBC virtual buffer size.
    b. In one example, the maximum number of available samples mapped to the IBC virtual buffer may be fixed when CTB/CTU size is larger than 64x64.
    c. In one example, the number of available samples mapped to the IBC virtual buffer may be restricted to be smaller or equal to one or multiple times of number of samples in a VPDU.
        i. In one example, the number of available samples mapped to the IBC virtual buffer may be restricted to be smaller or equal to three times of number of samples in a VPDU when CTU/CTB size is larger than 64x64.
7. The unavailability marking for IBC reference samples mapped to the IBC virtual buffer may be performed in the granularity of VPDU
    a. In one example, when samples are needed to be marked as unavailable, samples within the same VPDU may also be marked as unavailable.
    b. In one example, one or multiple VPDUs may be marked as unavailable simultaneously.
    c. In one example, which VPDU's samples are marked to unavailable may depend on the position of the current VPDU.
    d. In one example, which VPDU's samples are marked to unavailable may depend on the position of the previous or most recently decoded VPDU.
8. The unavailability marking for IBC reference samples mapped to the IBC virtual buffer may be performed right before encoding/decoding a video unit (e.g., a CU/PU/TU). In one example, for a WxH video unit with top-left position (X, Y) related to the top-left of the slice/tile/brick/subpicture/picture containing the video unit, the following may apply.
    a. In one example, right before encoding/decoding a video unit, the correspinding area in the virtual buffer may be marked as unavailable.

b. In the above example, the corresponding IBC virtual buffer area may be a luma area with x=(X%IBC_buf_width)..(X%IBC_buf_width)+W-1 and y=(Y%IBC_buf_height)..(Y%IBC_buf_height)+H-1.
c. In the above examples, the corresponding IBC virtual buffer area may be a chroma area with x=(X%IBC_buf_width)/SubWidthC..((X%IBC_buf_width)+W-1) /SubWidthC and y=(Y%IBC_buf_height)/SubHeightC..((Y%IBC_buf_height)+H-1)/SubHeightC.
d. In the above examples, IBC_buf_width refers to IBC virtual buffer width; IBC_buf_height refers to IBC virtual buffer height.
e. Alternatively, the unavailability marking for IBC reference samples mapped to the IBC virtual buffer may be performed right before encoding/decoding a K*L samples/pixels in a video unit.
    i. In one example, K and/or L are both equal to 1.
    ii. In one example, K and/or L are set to the sub-block size used in a CU/PU.

9. When CTU/CTB size is larger than 64x64, IBC reference may be the current VPDU and the three most recently decoded VPDUs.
    a. In one example, an index may be maintained for each VPDU mapped to the virtual IBC buffer to track the decoding order of each VPDU.
10. A counter may be maintained to track the number of available VPDUs mapped to the buffer.
    a. In one example, the counter is reset to 0 at beginning of decoding each CTU row and increased by one when one VPDU mapped to the buffer has been decoded.
    b. In one example, when the counter is larger than a certain value, e.g. 3, one VPDU's samples mapped to the buffer may be marked as unavailable and the counter may be decreased by 1.
11. When CTU/CTB size is 128x128, the corresponding IBC virtual buffer may be of size 256x128.
    a. Alternatively, the IBC virtual buffer may be of size (k*64)x128, where k>2.
12. When CTU/CTB size is 256x256, the corresponding IBC virtual buffer may be of size 64x256 to track availability of reference samples, i.e. ibcBufW = 64, ibcBufH = 256.

a. In one example, before decoding a VPDU with top-left position (x0, y0), the corresponding VPDU row (0, y0%256) in the IBC buffer will be set to -1.

13. When CTU/CTB size is 256x256, the corresponding IBC virtual buffer may be of size 128x256 to track availability of reference samples, i.e. ibcBufW = 128, ibcBufH = 256.
    a. In one example, only one VPDU may be kept (excluding the current VPDU) for each VPDU row in the buffer except for a certain VPDU row.
        i. In one example, only one VPDU may be kept (excluding the current VPDU) for each VPDU row in the buffer except for the last VPDU row.
14. IBC buffer may be not reset at the beginning of a CTU row.
    a. In one example, the IBC virtual buffer inherited from the above CTU row may be used as the initial state of the current CTU row.
    b. Alternatively, IBC virtual buffer may be partially reset at the beginning of a CTU row.
        i. In one example, a VPDU in the above CTU row may be inhertied in the current IBC buffer while other buffer region may be reset.
        ii. In one example, the most bottom-left VPDU of the above CTU row may be inhertied in the current IBC buffer while other buffer region may be reset.
15. Whether and/or how to mark samples in the buffer as unavilable may be independent of chroma block position.
    a. In one example, only when a luma block is of the 1st block in a VPDU, corresponding samples in the IBC buffer may be marked as unavailable.
    b. In one example, it may be disallow to reset or marked samples in the buffer as unavalable when decoding a chroma coding unit.
16. It is proposed at the beginning of decoding each VPDU, the corresponding IBC virtual buffer area may be reset based on the current VPDU position.
    a. In one example, the IBC virtual buffer area corresponding to (xVPDU + ctbSizeY, yVPDU) will be reset, where (xVPDU, yVPDU) denotes the current VPDU's position relative to the top-left of the picture.
17. Whether and/or how to mark samples in the IBC virtual buffer may depend on the most recently decoded VPDU's position and the VPDU size.

a. In one example, whether and/or how to mark samples in the IBC virtual buffer as unavailable may depend on the most recently decoded VPDU's position a brick/slice/tile/picture.
b. In one example, whether and/or how to mark samples in the IBC virtual buffer as unavailable may depend on the most recently decoded VPDU's position in a CTU row of a brick/slice/tile/picture.
c. In one example, what samples in the IBC virtual buffer to be marked as unavailable may be independent of the current block's position.

IBC line buffer related

18. Block vector prediction acrossing a CTU row may be disallowed
    a. In one example, when a block vector prediction is from a different CTU row compared to the current CTU row, it may be considered as unavailable.
19. Deblocking decision for the current IBC blocks acrossing a CTU row may be independent of the other block's block vector or motion vector.
    a. In one example, when two blocks are of different CTU row and one block is coded in IBC mode and the other is coded in IBC or Inter mode, the deblocking boundary strength may be always set equal to 1.
    b. In one example, when two blocks are of different CTU row and one block is coded in IBC mode and the other is coded in IBC or Inter mode, the deblocking boundary strength may be always set equal to 0.
20. Deblocking decision for the current IBC blocks acrossing CTU row may be independent of whether the other block is coded in IBC mode or Inter mode.
    a. In one example, when two blocks are of different CTU row and one block is coded in IBC mode and the other is not coded in Intra mode, the deblocking boundary strength may be always set equal to 1.

Subpicture related

21. It is proposed to allow predictions (e.g., motion/sample prediction) among two subpictures under certain conditions.

a. In one example, if a boundary of a first subpicture coincides with the picture boundary (or conformance window boundary), it may be allowed to use information from a second subpicture.
    i. Alternatively, furthermore, the subpicture boundary is the left or right subpicture boundary.
    ii. Alternatively, furthermore, the picture boundary is the left or right picture boundary.
    iii. In one example, the left (or right) boundary of the second subpicture may coincide with the left (or right) boundary of the picture.
b. The condition is true only if picture wrapping is allowed (e.g. sps_ref_wraparound_enabled_flag is equal to 1)

22. Picture wrapping excludes subpictures.
    a. In one example, picture wrapping is enabled if subpictures are used.
    b. In one example, subpictures cannot be used if picture wrapping is enabled.
    c. Alternatively, picture wrapping may be enabled when subpicture is used.
        iv. In one example, picture wrapping may be enabled for a sub-picture which has a boundary coinciding with the picture boundary.

Chroma QP table related

23. The smallest index of chroma QP table may be independent to the bit depth for chroma component.
    a. In one example, the smallest index of chroma QP table may depend on the bit depth of luma component.
    b. In one example, the smallest index of chroma QP table may be $QpBdOffset_Y$, i.e. 6*bit_depth_luma_minus8.

5 Example implementations of the disclosed technology

5.1 Embodiment #1

[00100] When CTU size is 256x256, a 64x256 IBC virtual buffer ibcBuf is maintained, i.e. ibcBufW = 64, ibcBufH = 256. VPDU size is 64x64 and beside the current VPDU, 3 additional VPDU's on-chip memory is used to store IBC reference samples.

[00101] The buffer ibcBuf is reset to -1 at the beginning of decoding a CTU row.

[00102] At the beginning decoding a new VPDU with top-left position (x0, y0) relative to the top-left corner of the picture, the following applies

[00103] 1) For x = x0..x0+63, y = y0..y0+63, ibcBuf[x%ibcBufW][y%ibcBufH] = -1

[00104] 2) After decoding a CU, for (x, y) in that CU relative to the top-left corner of the picture, set ibcBuf[x%ibcBufW][y%ibcBufH] as the reconstructed value of sample (x, y) before in-loop filtering, e.g. SAO, deblocking, ALF.

[00105] 3) Given a bv, the reference for (x, y) is ibcBuf[(x+bv[0])%ibcBufW][(y+bv[1])%ibcBufH]

[00106] It is a bitstream constrain that the following two conditions shall be true

[00107] 1) Given a WxH block with top-left position (x, y) relative the top-left of the picture, (y%ibcBufH)+H <= ibcBufH

[00108] 2) ibcBuf[(x+bv[0])%ibcBufW][(y+bv[1])%ibcBufH] shall not contain invalid pixel value, e.g. -1, for x=0..W-1, y=0..,H-1

5.2 Embodiment #2

[00109] When CTU size is 256x256, a 128x256 IBC virtual buffer ibcBuf is maintained, i.e. ibcBufW = 128, ibcBufH = 256. VPDU size is 64x64 and beside the current VPDU, 3 additional VPDU's on-chip memory is used to store IBC reference samples.

[00110] The buffer ibcBuf is reset to -1 at the beginning of decoding a CTU row. xPrevVPDU = 0 and yPrevVPDU =0.

[00111] At the beginning decoding a new VPDU with top-left position (x0, y0) relative to the top-left corner of the picture, the following applies

[00112] 1) If (yPrevVPDU+64)%ibcBufH is not equal to 0,

[00113] for x = x0..x0+63, y = y0..y0+63, ibcBuf[(x+xPrevVPDU-64)%ibcBufW][(y+yPrevVPDU)%ibcBufH] = -1

[00114] 2) Otherwise ((yPrevVPDU+64)%ibcBufH is equal to 0),

[00115] for x = x0..x0+63, y = y0..y0+63, ibcBuf[(x+xPrevVPDU)%ibcBufW][(y+yPrevVPDU)%ibcBufH] = -1

[00116] 3) xPrevVPDU = x0 and yPrevVPDU = y0

[00117] It is a bitstream constrain that the following two conditions shall be true

[00118] 1) Given a WxH block with top-left position (x, y) relative the top-left of the

picture, (y%ibcBufH)+H <= ibcBufH

[00119] 2) ibcBuf[(x+bv[0])%ibcBufW][(y+bv[1])%ibcBufH] shall not contain invalid pixel value, e.g. -1, for x=0..W-1, y=0..,H-1

5.3 Embodiment #3

This embodiment reflects bullet 2. Changes, marked in ***bold faced italics***, are based on VVC draft 6 document JVET-O2001-vE.

log2_min_luma_coding_block_size_minus2 plus 2 specifies the minimum luma coding block size.

The variables CtbLog2SizeY, CtbSizeY, MinCbLog2SizeY, MinCbSizeY, IbcBufWidthY, IbcBufWidthC and Vsize are derived as follows:

CtbLog2SizeY = log2_ctu_size_minus5 + 5 (7-15)

CtbSizeY = 1 << CtbLog2SizeY (7-16)

MinCbLog2SizeY = log2_min_luma_coding_block_size_minus2 + 2 (7-17)

MinCbSizeY = 1 << MinCbLog2SizeY (7-18)

IbcBufWidthY = 128 * 128 / CtbSizeY + ***CtbSizeY*** (7-19)

IbcBufWidthC = IbcBufWidthY / SubWidthC (7-20)

VSize = Min(64, CtbSizeY) (7-21)

5.4 Embodiment #4

This embodiment reflects bullet 3.

Denote (xCb, yCb) as the top-left position of the current block relative to top-left of the picture. Block width and height are W and H respectively. Block vector for the block is (xBv, yBv)

Constrain of VPDU row relative to the picture:

It is a bitstream constrain that (xCb+xBv)/vSize shall be equal to (xCb+xBv+W-1/vSize).

Constrain of VPDU column relative to the picture:

It is a bitstream constrain that (yCb+yBv)/vSize shall be equal to (yCb+yBv+H-1/vSize).

Constrain of VPDU row relative to the IBC buffer:

It is a bitstream constrain that ((xCb+xBv)%IbcBufWidthY)/vSize shall be equal to (((xCb+xBv+W-1)%IbcBufWidthY)/vSize).

Constrain of VPDU column relative to the IBC buffer:

It is a bitstream constrain that ((yCb+yBv)%IbcBufHeightY)/vSize shall be equal to (((yCb+yBv+H-1)% IbcBufHeightY)/vSize).

5.5 Embodiment #5

This embodiment reflects that marking samples in the IBC virtual buffer as unavailable should be independent of chroma blocks.

The changes marked in ***bold and italic*** are based on JVET-O2001-vE. Deleted texts are marked with double brackets (e.g., [[a]] denotes the deletion of the character "a").

7.4.9.5 Coding unit semantics

When ResetIbcBuf is equal to 1, the following applies:

– For x = 0..IbcBufWidthY − 1 and y = 0..CtbSizeY − 1, the following assignments are made:

IbcVirBuf[0][x][y] = −1 (7-153)

– The variable ResetIbcBuf is set equal to 0.

When x0 % VSize is equal to 0 and y0 % VSize is equal to 0 ***and cIdx is equal to 0***, the following assignments are made for x = x0..x0 + VSize − 1 and y = y0..y0 + VSize − 1:

IbcVirBuf[0][x % IbcBufWidthY][y % CtbSizeY] = −1 (7-154)

5.6 Embodiment #6

7.4.9.5 Coding unit semantics

…

When ResetIbcBuf is equal to 1, the following applies:

– For x = 0..IbcBufWidthY − 1 and y = 0..CtbSizeY − 1, the following assignments are made:

IbcVirBuf[0][x][y] = −1 (7-153)

xPrevV = 0

yPrevV = 0

– The variable ResetIbcBuf is set equal to 0.

When x0 % VSize is equal to 0 and y0 % VSize is equal to 0 ***and cIdx is equal to 0, the following applies: [[assignments are made for x = x0..x0 + VSize − 1 and y = y0..y0 + VSize − 1:]]***

- ***If yPrevV % VSize is equal to 0, xP is set equal to xPrevV – (CtbLog2SizeY = = 7) ? 128 : (–VSize), otherwise xP is set equal to xPrevV – (CtbLog2SizeY = = 7) ? 64 : (–VSize).***
- ***For x = xP..xP + max(VSize, cbWidth) –1 and y = yPrevV.. yPrevV + max(VSize, cbHeight) –1, the follwing assignment are made***

IbcVirBuf[0][x % IbcBufWidthY][y % CtbSizeY] = −1 (7-154)

- ***The variables xPrevV is set equal to (x0 + cbWidth – 1) / VSize * VSize and yPrevV is set equal to (y0 + cbHeight – 1) / VSize * VSize.***

5.7 Embodiment #7

This embodiment gives an example of allowing subpicture prediction. The changes compared to JVET-O2001-vE are highlighted in ***bold and italic***.

8.5.6.3.3 Luma integer sample fetching process

Inputs to this process are:

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

Output of this process is a predicted luma sample value predSampleLXL

The variable shift is set equal to Max(2, 14 − $BitDepth_Y$).

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

The luma locations in full-sample units (xInt, yInt) are derived as follows:

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

  xInt = Clip3(SubPicLeftBoundaryPos, SubPicRightBoundaryPos, sps_ref_wraparound_enabled_flag ? ClipH((sps_ref_wraparound_offset_minus1 + 1) * MinCbSizeY, picW, xIntL) :xInt) ***(8-782)***

  yInt = Clip3(SubPicTopBoundaryPos, SubPicBotBoundaryPos, yInt) ***(8-782)***

- ***Otherwise:***

  xInt = Clip3(0, picW − 1, sps_ref_wraparound_enabled_flag ? (8-782)
  ClipH((sps_ref_wraparound_offset_minus1 + 1) * MinCbSizeY, picW, $xInt_L$) : $xInt_L$)

  yInt = Clip3(0, picH − 1, $yInt_L$) (8-783)

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

$predSampleLX_L$ = $refPicLX_L$[xInt][yInt] << shift3 (8-784)

5.8 Embodiment #8

This embodiment shows an examplary design of chroma QP table. The changes, marked in ***bold and italic***, are based on JVET-O2001-vE. Deleted texts are marked with double brackets (e.g., [[a]] denotes the deletion of the character "a").

7.4.3.3 Sequence parameter set RBSP semantics

…

The i-th chroma QP mapping table ChromaQpTable[i] for i = 0..same_qp_table_for_chroma ? 0 : 2 is derived as follows:

qpInVal[i][0] = [[***QpBdOffset***$_C$]]***QpBdOffset***$_Y$ + delta_qp_in_val_minus1[i][0]

qpOutVal[i][0] = −QpBdOffset$_C$ + delta_qp_out_val[i][0]

for(j = 1; j <= num_points_in_qp_table_minus1[i]; j++) {

qpInVal[i][j] = qpInVal[i][j − 1] + delta_qp_in_val_minus1[i][j] + 1

qpOutVal[i][j] = qpOutVal[i][j − 1] + delta_qp_out_val[i][j]

}

ChromaQpTable[i][qpInVal[i][0]] = qpOutVal[i][0]

for(k = qpInVal[i][0] − 1; k >= [[***QpBdOffset***$_C$]] ***QpBdOffset***$_Y$; k − −)

ChromaQpTable[i][k] = Clip3(−QpBdOffset$_C$, 63, ChromaQpTable[i][k + 1] − 1) (7-31)

for(j = 0; j < num_points_in_qp_table_minus1[i]; j++) {

sh = (delta_qp_in_val_minus1[i][j + 1] + 2) >> 1

for(k = qpInVal[i][j] + 1, m = 1; k <= qpInval[i][j + 1]; k++, m++)

ChromaQpTable[i][k] = ChromaQpTable[i][qpInVal[i][j]] +

(delta_qp_out_val[i][j + 1] * m + sh) / (delta_qp_in_val_minus1[i][j + 1] + 1)

}

for(k = qpInVal[i][num_points_in_qp_table_minus1[i]] + 1; k <= 63; k++)

ChromaQpTable[i][k] = Clip3(−QpBdOffset$_C$, 63, ChromaQpTable[i][k − 1] + 1)

When same_qp_table_for_chroma is equal to 1, ChromaQpTable[1][k] and ChromaQpTable[2][k] are set equal to ChromaQpTable[0][k] for k = [[***QpBdOffset***$_C$]] ***QpBdOffset***$_Y$..63.

It is a requirement of bitstream conformance that the values of qpInVal[i][j] ***shall be in the range of −QpBdOffset***$_Y$ ***to 63***, and qpOutVal[i][j] shall be in the range of −QpBdOffset$_C$ to 63, inclusive for i = 0..same_qp_table_for_chroma ? 0 : 2 and j = 0..num_points_in_qp_table_minus1[i].

…

5.9 Embodiment #9

This embodiment shows an examplar design of an enlarged IBC virtual buffer. The changes, marked in Italic and bold, are based on JVET-O2001-vE. Deleted texts are marked with double brackets (e.g., [[a]] denotes the deletion of the character "a").

7.4.3.3 Sequence parameter set RBSP semantics

…

The variables CtbLog2SizeY, CtbSizeY, MinCbLog2SizeY, MinCbSizeY, IbcBufWidthY, IbcBufWidthC and Vsize are derived as follows:

CtbLog2SizeY = log2_ctu_size_minus5 + 5

(7-15)

CtbSizeY = 1 << CtbLog2SizeY

(7-16)

MinCbLog2SizeY = log2_min_luma_coding_block_size_minus2 + 2 (7-17)

MinCbSizeY = 1 << MinCbLog2SizeY

(7-18)

IbcBufWidthY = 128 * 128 / CtbSizeY + ***CtbSizeY*** (7-19)

IbcBufWidthC = IbcBufWidthY / SubWidthC (7-20)

VSize = Min(64, CtbSizeY)

(7-21)

…

7.4.9.5 Coding unit semantics

…

When x0 % VSize is equal to 0 and y0 % VSize is equal to 0, the following assignments are made for x = x0..x0 + VSize − 1 and y = y0..y0 + VSize − 1:

IbcVirBuf[0][(x + ***CtbSizeY***) % IbcBufWidthY][y % CtbSizeY] = −1 (7-154)

5.10 Embodiment #10

This embodiment shows an examplar design of an enlarged IBC virtual buffer. The changes, marked in Italic and bold, are based on JVET-O2001-vE. Deleted texts are marked with double brackets (e.g., [[a]] denotes the deletion of the character "a").

7.4.3.3 Sequence parameter set RBSP semantics

…

The variables CtbLog2SizeY, CtbSizeY, MinCbLog2SizeY, MinCbSizeY, IbcBufWidthY, IbcBufWidthC and Vsize are derived as follows:

CtbLog2SizeY = log2_ctu_size_minus5 + 5

(7-15)

CtbSizeY = 1 << CtbLog2SizeY

(7-16)

MinCbLog2SizeY = log2_min_luma_coding_block_size_minus2 + 2 (7-17)

MinCbSizeY = 1 << MinCbLog2SizeY

(7-18)

W = CtbSizeY==128? 256: CtbSizeY

IbcBufWidthY = ***W***[[128]] * 128 / CtbSizeY (7-19)

IbcBufWidthC = IbcBufWidthY / SubWidthC (7-20)

VSize = Min(64, CtbSizeY)

(7-21)

…

7.4.9.5 Coding unit semantics

…

When x0 % VSize is equal to 0 and y0 % VSize is equal to 0, the following assignments are made for x = x0..x0 + VSize − 1 and y = y0..y0 + VSize − 1:

IbcVirBuf[0][(x + ***CtbSizeY***) % IbcBufWidthY][y % CtbSizeY] = −1 (7-154)

5.11 Embodiment #11

The changes, marked in bold and Italic, are based on JVET-O2001-vE. Deleted texts are marked with double brackets (e.g., [[a]] denotes the deletion of the character "a").

7.4.3.3 Sequence parameter set RBSP semantics

…

The variables CtbLog2SizeY, CtbSizeY, MinCbLog2SizeY, MinCbSizeY, IbcBufWidthY, IbcBufWidthC and Vsize are derived as follows:

CtbLog2SizeY = log2_ctu_size_minus5 + 5

(7-15)

CtbSizeY = 1 << CtbLog2SizeY

(7-16)

MinCbLog2SizeY = log2_min_luma_coding_block_size_minus2 + 2 (7-17)

MinCbSizeY = 1 << MinCbLog2SizeY

(7-18)

IbcBufWidthY = [[128]] ***256*** * 128 / CtbSizeY (7-19)

IbcBufWidthC = IbcBufWidthY / SubWidthC (7-20)

VSize = Min(64, CtbSizeY)

(7-21)

…

7.4.9.5 Coding unit semantics

…

When x0 % VSize is equal to 0 and y0 % VSize is equal to 0, the following assignments are made for x = x0..x0 + VSize − 1 and y = y0..y0 + VSize − 1:

IbcVirBuf[0][(x + ***(IbcBufWidthY >> 1)***) % IbcBufWidthY][y % CtbSizeY] = −1 (7-154)

[00120] FIG. 4 shows examples of reference blocks crossing VPDU column and VPDU row. As shown in FIG. 4, for a current CU (in blue 602), block filled in red (604) is an crossing

VPDU column reference block and block filled in yellow (606) is an crossing VPDU row reference block. Each large block indicates a 64x64 VPDU and green area (608) indicated reconstrued pixels that can be used for IBC reference.

[00121] FIG. 5 is a block diagram of a video processing apparatus 500. The apparatus 500 may be used to implement one or more of the methods described herein. The apparatus 500 may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus 500 may include one or more processors 502, one or more memories 504 and video processing hardware 506. The processor(s) 502 may be configured to implement one or more methods (including, but not limited to, method 400) described in the present document. The memory (memories) 504 may be used for storing data and code used for implementing the methods and techniques described herein. The video processing hardware 506 may be used to implement, in hardware circuitry, some techniques described in the present document.

[00122] FIG. 6 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.

[00123] 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.

[00124] 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.

[00125] In some embodiments, the video coding methods may be implemented using an apparatus that is implemented on a hardware platform as described with respect to FIG. 5 or 6.

[00126] The examples described above may be incorporated in the context of the methods described below, e.g., methods 710 to [], which may be implemented at a video decoder or a video encoder.

[00127] FIG. 7A shows a flowchart of an exemplary method 710 for video processing. The method 710 includes, at step 712, making a decision, based on a size of a virtual buffer associated with a current block that is coded based on pixels in a reference block, regarding a validity of a block vector or one or more samples mapped to the virtual buffer, a current picture comprising the current block and the reference block, and the size of the virtual buffer being based on a size of a virtual pipeline data unit (VPDU), a size of a coding tree block (CTB) size or a size of a coding tree unit (CTU). The method 700 includes, at step 714, performing, based on the decision, a conversion between the current block and a bitstream representation of the current block.

[00128] FIG. 7B shows a flowchart of an exemplary method 720 for video processing. The method 720 includes, at step 722, designating, for a current block that is coded based on pixels in a reference block, one or more reference samples of the reference block as unavailable, each of the one or more reference samples being mapped to a virtual buffer and having a corresponding sample in at least a current virtual pipeline data unit (VPDU) associated with the reference block, and a current picture comprising the current block and the reference block. The method 720 includes, at step 724, performing, based on the designating, a conversion between the current block and a bitstream representation of the current block.

[00129] FIG. 7C shows a flowchart of an exemplary method 730 for video processing. The method 730 includes, at step 732, determining, for a current block that is coded based on pixels in a reference block, a size of a virtual buffer, associated with the reference block, based on a size of a coding tree block (CTB) or a size of a coding tree unit (CTU) of the current block, a current picture comprising the current block and the reference block. The method 730 includes, at step 734, performing, based on the determining, a conversion between the current block and a bitstream representation of the current block.

[00130] FIG. 8A shows a flowchart of an exemplary method 810 for video processing. The method 810 includes, at step 812, determining, for a conversion between a current video block of a video picture of a video and a coded representation of the video, a number of reference samples in a reference region of the video picture used for predicting the current video block, based on a rule, wherein the rule specifies that the number of reference samples is limited to a certain range. The method 810 includes, at step 814, performing, based on the determining, the conversion.

[00131] FIG. 8B shows a flowchart of an exemplary method 820 for video processing. The method 820 includes, at step 822, determining, for a conversion between a current video block of a video picture of a video and a coded representation of the video, availability of one or more reference samples of a reference block used for predicting the current video block based on a rule, wherein the rule specifies to use a granularity at a virtual pipeline data unit (VPDU) level in the determining. The method 820 includes, at step 824, performing, based on the determining, a conversion between the current block and a coded representation of the current block.

[00132] FIG. 8C shows a flowchart of an exemplary method 830 for video processing. The method 830 includes, at step 832, performing a conversion between a current video block of a video picture of a video and a coded representation of the video. In some implementations, the the conversion is performed according to a rule, wherein the current video block is represented in the coded representation using a residual of prediction from reference samples from a reference region in the video pictures, and wherein the rule specifies that sample values in the reference region are reset at beginning of processing each virtual pipeline data unit (VPDU) during the conversion based on a position of a current VPDU of the current video block. In some implementations, the conversion comprises using, a count of a number of available current virtual pipeline data units (VPDUs) that are mapped to a reference region of the video picture from which one or more reference samples are used for predicting the current video block.

[00133] FIG. 8D shows a flowchart of an exemplary method 840 for video processing. The method 840 includes, at step 842, determining, for a current video block of a video picture of a video and a coded representation of the video, that samples from a current virtual pipeline data unit (VPDU) and three additional recently processed VPDUs in the video picture are available as reference samples for generating a prediction of the current video block due to the current video block having a size larger than a size of 64x64 luma samples. The method 840 includes, at step 844, performing the conversion based on the determining.

[00134] 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.

[00135] Some embodiments of the disclosed technology include making a decision or determination to disable a video processing tool or mode. In an example, when the video processing tool or mode is disabled, the encoder will not use the tool or mode in the conversion of the block of video to the bitstream representation of the video. In another example, when the video processing tool or mode is disabled, the decoder will process the bitstream with the knowledge that the bitstream has not been modified using the video processing tool or mode that was disabled based on the decision or determination.

[00136] 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.

[00137] Various solutions and embodiments described in the present document are further described using a list of clauses. The first set of clauses describe certain features and aspects of the disclosed techniques in the previous section.

[00138] 1. A method for video processing, comprising: making a decision, based on a size of a virtual buffer associated with a current block that is coded based on pixels in a reference block, regarding a validity of a block vector or one or more samples mapped to the virtual buffer, wherein a current picture comprises the current block and the reference block, and wherein the size of the virtual buffer is based on a size of a virtual pipeline data unit (VPDU), a size of a coding tree block (CTB) size or a size of a coding tree unit (CTU); and performing, based on the decision, a conversion between the current block and a bitstream representation of the current block.

[00139] 2. The method of claim 1, wherein a product of a height and a width of the virtual buffer is fixed, and wherein the height or the width is based on the size of the VPDU, the size of the CTB or the size of the CTU.

[00140] 3. The method of clause 1, wherein a width of the virtual buffer is equal to a width of the CTB or a width of the CTU.

[00141] 4. The method of clause 1, wherein a width or a height of the virtual buffer is N times a width or a height of the VPDU, respectively, and wherein $N \geq 1$ is an integer.

[00142] 5. The method of clause 1, wherein a maximum number of the one or more samples is less than the size of the virtual buffer.

[00143] 6. The method of clause 1, wherein a maximum number of the one or more samples is fixed upon a determination that the size of the CTB or the size of the CTU is larger than 64×64.

[00144] 7. The method of clause 1, wherein a number of the one or more samples is less than or equal to N times a number of samples in the VPDU, and wherein $N \geq 1$ is an integer.

[00145] 8. The method of clause 7, wherein $N = 3$ upon a determination that the size of the CTB or the size of the CTU is larger than 64×64.

[00146] 9. A method for video processing, comprising: designating, for a current block that is coded based on pixels in a reference block, one or more reference samples of the reference block as unavailable, wherein each of the one or more reference samples is mapped to a virtual buffer and has a corresponding sample in at least a current virtual pipeline data unit (VPDU) associated

with the reference block, and wherein a current picture comprises the current block and the reference block; and performing, based on the designating, a conversion between the current block and a bitstream representation of the current block.

[00147] 10. The method of clause 9, further comprising: designating the corresponding sample in the current VPDU as unavailable.

[00148] 11. The method of clause 10, wherein the designating the corresponding sample is based on a position of the current VPDU.

[00149] 12. The method of clause 10, wherein the designating the corresponding sample is based on a position of a previous VPDU or a recently decoded VPDU.

[00150] 13. The method of clause 9, wherein each of the one or more reference samples has a corresponding sample in each of three most recently decoded VPDUs.

[00151] 14. A method for video processing, comprising: determining, for a current block that is coded based on pixels in a reference block, a size of a virtual buffer, associated with the reference block, based on a size of a coding tree block (CTB) or a size of a coding tree unit (CTU) of the current block, wherein a current picture comprises the current block and the reference block; and performing, based on the determining, a conversion between the current block and a bitstream representation of the current block.

[00152] 15. The method of clause 14, wherein the size of the virtual buffer is 256×128 upon a determination that the size of the CTB or the CTU is 128×128.

[00153] 16. The method of clause 14, wherein the size of the virtual buffer is 64×256 upon a determination that the size of the CTB or the CTU is 256×256.

[00154] 17. The method of clause 14, wherein the size of the virtual buffer is 128×256 upon a determination that the size of the CTB or the CTU is 256×256.

[00155] 18. The method of any of clauses 1 to 17, wherein coding the current block based on the pixels in the reference block in the current picture that comprises the current block is an intra block copy (IBC) operation, and wherein the virtual buffer is an IBC virtual buffer.

[00156] 19. A method of video processing, comprising: allocating, for a conversion between a bitstream representation of a current block of video and the current block, an intra block coding (IBC) virtual buffer size that is greater than a minimum size of a block vector search area for the conversion; and performing the conversion based on the allocating.

[00157] 20. The method of clause 19, wherein the IBC buffer size is larger than a total size of

a virtual pipeline data unit memory used for the conversion.

[00158] 21. The method of any of clauses 19-20, wherein a width of the IBC virtual buffer size is (128*128/ctbSizeY + ctbSizeY).

[00159] 22. A method of video processing, comprising: determining, for a conversion between a bitstream representation of a current block of video and the current block, a size of a reference block for the conversion based on an intra block prediction coding based on a rule; and performing the conversion based on the determining; wherein the rule constrains the reference block to be within a virtual pipeline data unit (VPDU) column or a virtual pipeline data unit row.

[00160] 23. The method of clause 22, wherein the rule disallows the reference block to be across different VPDU rows.

[00161] 24. The method of clause 22, wherein the rule disallows the reference block to be across different VPDU columns.

[00162] 25. The method of any of clauses 22-24, wherein the VPDU column or the VPDU row are relative to a picture that contains the current block.

[00163] 26. The method of any of clauses 22-24, wherein the VPDU column or the VPDU row are relative to in intra block copy virtual buffer.

[00164] 27. The method of any of clauses 1-26, wherein the conversion comprises video coding to generate the bitstream representation from the current block.

[00165] 28. The method of any of clauses 1-26, wherein the conversion comprises video decoding to generate the current block from the bitstream representation.

[00166] 29. An apparatus in a video system comprising a processor and a non-transitory memory with instructions thereon, wherein the instructions upon execution by the processor, cause the processor to implement the method in any one of clauses 1 to 28.

[00167] 30. A computer program product stored on a non-transitory computer readable media, the computer program product including program code for carrying out the method in any one of clauses 1 to 28.

[00168] 31. A method, apparatus or system described herein.

[00169] The second set of clauses describe certain features and aspects of the disclosed techniques in the previous section, for example, Example Items 2-5, 8, and 16.

[00170] 1. A method for video processing, comprising: determining, for a conversion between a current video block of a video picture of a video and a coded representation of the video,

a number of reference samples in a reference region of the video picture used for predicting the current video block, based on a rule, wherein the rule specifies that the number of reference samples is limited to a certain range; and performing, based on the determining, the conversion.

[00171] 2. The method of clause 1, wherein the rule specifies that a maximum number of the reference samples is less than a size of the reference region.

[00172] 3. The method of clause 1, wherein the rule specifies that a maximum number of the reference samples is fixed in a case that that a size of a coding tree block (CTB) or a size of a coding tree unit (CTU) is larger than 64×64.

[00173] 4. The method of clause 1, wherein the rule specifies that the number of the reference samples is less than or equal to N times a number of samples in a virtual pipeline data unit (VPDU), and wherein $N \geq 1$ is an integer.

[00174] 5. The method of clause 4, wherein $N = 3$ in a case that a determination that a size of a coding tree block (CTB) or a size of a coding tree unit (CTU) is larger than a size of 64×64 luma samples.

[00175] 6. A method for video processing, comprising: determining, for a conversion between a current video block of a video picture of a video and a coded representation of the video, availability of one or more reference samples of a reference block used for predicting the current video block based on a rule, wherein the rule specifies to use a granularity at a virtual pipeline data unit (VPDU) level in the determining; and performing, based on the determining, a conversion between the current block and a coded representation of the current block.

[00176] 7. The method of clause 6, wherein the determining determines that the one or more reference samples as unavailable and wherein the method further comprises determining one or more corresponding samples in a current VPDU as unavailable.

[00177] 8. The method of clause 6, further comprising: determining availability of one or more VPDUs as unavailable.

[00178] 9. The method of clause 7, wherein the determining of the one or more corresponding samples is based on a position of the current VPDU.

[00179] 10. The method of clause 7, wherein the determining of the one or more corresponding samples is based on a position of a previous VPDU or a recently decoded VPDU.

[00180] 11. The method of clause 6, wherein the determining is performed right before encoding or decoding a video unit including the current video block and wherein the one or more

reference samples correspond to a reference region of the video picture.

[00181] 12. The method of clause 11, wherein, before encoding or decoding of the video unit, a corresponding area in the reference region is marked as unavailable.

[00182] 13. The method of clause 11, wherein a corresponding area in the reference region corresponds to a luma area with x=(X%IBC_buf_width)..(X%IBC_buf_width)+W-1 and y=(Y%IBC_buf_height)..(Y%IBC_buf_height)+H-1, wherein the video unit has a size of WxH, IBC_buf_width and IBC_buf_height indicate a width and a height of the virtual buffer, respectively, and a top-left position of the video unit is (X, Y).

[00183] 14. The method of clause 11, wherein a corresponding area in the reference region corresponds to a chroma area with x=(X%IBC_buf_width)/SubWidthC..((X%IBC_buf_width)+W-1) /SubWidthC and y=(Y%IBC_buf_height)/SubHeightC..((Y%IBC_buf_height)+H-1)/SubHeightC, wherein the video unit has a size of WxH, IBC_buf_width and IBC_buf_height indicate a width and a height of the virtual buffer, respectively, and a top-left position of the video unit is (X, Y).

[00184] 15. The method of clause 6, wherein the determining is performed right before encoding or decoding KxL samples or KxL pixels that are included in a video unit including the current video block.

[00185] 16. The method of clause 15, wherein at least one of K and L is equal to 1.

[00186] 17. The method of clause 15, wherein at least one of K and L is set to a sub-block size used in the video unit.

[00187] 18. A method for video processing, comprising: performing a conversion between a current video block of a video picture of a video and a coded representation of the video according to a rule, wherein the current video block is represented in the coded representation using a residual of prediction from reference samples from a reference region in the video pictures, and wherein the rule specifies that sample values in the reference region are reset at beginning of processing each virtual pipeline data unit (VPDU) during the conversion based on a position of a current VPDU of the current video block.

[00188] 19. The method of clause 18, wherein the reference region is at a position of (xVPDU + ctbSizeY , yVPDU), wherein (xVPDU, yVPDU) denotes a position of the current VPDU relative to a top-left of a picture including the current video block and ctbSizeY is a width or a height of a luma coding tree block (CTB) or a luma coding tree unit (CTU).

[00189] 20. A method for processing video, comprising: determining, for a current video block of a video picture of a video and a coded representation of the video, that samples from a current virtual pipeline data unit (VPDU) and three additional recently processed VPDUs in the video picture are available as reference samples for generating a prediction of the current video block due to the current video block having a size larger than a size of 64x64 luma samples; and performing the conversion based on the determining.

[00190] 21. The method of clause 20, wherein the current video block corresponds to a coding tree block (CTB) or a coding tree unit (CTU).

[00191] 22. The method of clause 20, wherein an index is maintained for each VPDU mapped to a reference region of the video picture.

[00192] 23. A method for video processing, comprising: performing a conversion between a current video block of a video picture of a video and a coded representation of the video, and wherein, the conversion comprises using, a count of a number of available current virtual pipeline data units (VPDUs) that are mapped to a reference region of the video picture from which one or more reference samples are used for predicting the current video block.

[00193] 24. The method of clause 23, wherein the counter is reset to zero at beginning of decoding each CTU row and increased by one at decoding a VPDU mapped to the reference region.

[00194] 25. The method of clause 23, wherein the counter is larger than a certain value and wherein samples in a VPDU mapped to the reference region are marked as unavailable and the counter is decreased by one.

[00195] 26. The method of any of clauses 1 to 25, wherein the conversion includes encoding the current video block into the coded representation.

[00196] 27. The method of any of clauses 1 to 25, wherein the conversion includes decoding the coded representation from the current video block.

[00197] 28. A video processing apparatus comprising a processor configured to implement a method recited in any one or more of clauses 1 to 27.

[00198] 29. A computer readable medium storing program code that, when executed, causes a processor to implement a method recited in any one or more of clauses 1 to 27.

[00199] From the foregoing, it will be appreciated that specific embodiments of the presently disclosed technology have been described herein for purposes of illustration, but that various modifications may be made without deviating from the scope of the invention. Accordingly, the

presently disclosed technology is not limited except as by the appended claims.

[00200] Implementations of the subject matter and the functional operations described in this patent document can be implemented in various systems, digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this specification and their structural equivalents, or in combinations of one or more of them. Implementations of the subject matter described in this specification can be implemented as one or more computer program products, i.e., one or more modules of computer program instructions encoded on a tangible and non-transitory 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 of them. The term "data processing unit" or "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.

[00201] 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.

[00202] The processes and logic flows described in this specification 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).

[00203] 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 nonvolatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[00204] It is intended that the specification, together with the drawings, be considered exemplary only, where exemplary means an example. As used herein, the use of "or" is intended to include "and/or", unless the context clearly indicates otherwise.

[00205] While this patent document contains many specifics, these should not be construed as limitations on the scope of any invention or of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular inventions. 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.

[00206] 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.

[00207] 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.