(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 December 2020 (30.12.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/259677 A1

(51) International Patent Classification:
H04N 19/103 (2014.01) *H04N 19/176* (2014.01)
H04N 19/159 (2014.01)

(21) International Application Number:
PCT/CN2020/098471

(22) International Filing Date:
28 June 2020 (28.06.2020)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
PCT/CN2019/093552
28 June 2019 (28.06.2019) CN
PCT/CN2019/094957
06 July 2019 (06.07.2019) CN
PCT/CN2019/095297
09 July 2019 (09.07.2019) CN
PCT/CN2019/095504
10 July 2019 (10.07.2019) CN
PCT/CN2019/095656
11 July 2019 (11.07.2019) CN
PCT/CN2019/095913
13 July 2019 (13.07.2019) CN
PCT/CN2019/096048
15 July 2019 (15.07.2019) CN

(71) Applicants: **BEIJING BYTEDANCE NETWORK TECHNOLOGY CO., LTD.** [CN/CN]; Room B-0035, 2/F, No.3 Building, No.30, Shixing Road, Shijingshan District, Beijing 100041 (CN). **BYTEDANCE INC.** [US/US]; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US).

(72) Inventors: **XU, Jizheng**; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **ZHANG, Li**; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **ZHANG, Kai**; 12655 West Jefferson Boulevard, Sixth Floor, Suite No. 137, Los Angeles, California 90066 (US). **LIU, Hongbin**; Jinritoutiao Post Office, China Satellite Communications Tower, No.63, Zhichun Road, Haidian District, Beijing 100080 (CN). **WANG, Yue**; Jinritoutiao Post Office, China Satellite Communications Tower, No.63, Zhichun Road, Haidian District, Beijing 100080 (CN).

(74) Agent: **LIU, SHEN & ASSOCIATES**; 10th Floor, Building 1, 10 Caihefang Road, Haidian District, Beijing 100080 (CN).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM,

(54) Title: VALIDITY CHECKS FOR A BLOCK VECTORS FOR INTRA BLOCK COPY IN VIDEO CODING

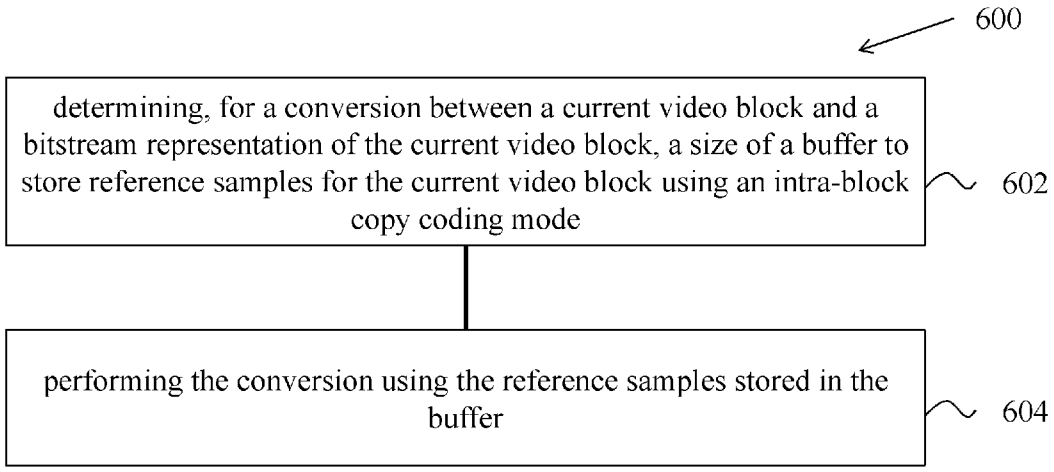

FIG. 6

(57) Abstract: A method of video processing includes determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the current video block, a block vector, wherein validity of the block vector is independent of (1) a location (P, Q) of a sample block and/or (2) whether a sample at the location (P, Q) is reconstructed, and/or (3) a location of the current video block. The block vector represents a pixel displacement between the current video block and the sample block. The method includes performing, using the block vector, the conversion in an intra block copy mode.

WO 2020/259677 A1

[Continued on next page]

AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

— *with international search report (Art. 21(3))*

VALIDITY CHECKS FOR A BLOCK VECTORS FOR INTRA BLOCK COPY IN VIDEO CODING

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] Under the applicable patent law and/or rules pursuant to the Paris Convention, this application is made to timely claim the priority to and benefits of International Patent Application No. PCT/CN2019/093552, filed on June 28, 2019, International Patent Application No. PCT/CN2019/094957, filed on July 6, 2019, International Patent Application No. PCT/CN2019/095297, filed on July 9, 2019, International Patent Application No. PCT/CN2019/095504, filed on July 10, 2019, International Patent Application No. PCT/CN2019/095656, filed on July 11, 2019, International Patent Application No. PCT/CN2019/095913, filed on July 13, 2019, and International Patent Application No. PCT/CN2019/096048, filed on July 15, 2019. For all purposes under the law, the entire disclosures of the aforementioned applications are incorporated by reference as part of the disclosure of this application.

TECHNICAL FIELD

[0002] This patent document relates to video coding and decoding 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] The present document describes various embodiments and techniques for buffer management and block vector coding for intra block copy mode for decoding or encoding video or images.

[0005] In one example aspect, a method of visual media processing is disclosed. The method includes determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the current video block, a block vector (BVx,BVy), wherein validity of the block vector (BVx, BVy) is independent of (1) a location (P, Q) of a sample block and/or (2) whether a sample at the location (P,Q) is reconstructed, and/or (3) a location of the current video block, wherein, the block vector (BVx, BVy) represents a pixel displacement between the current video block and the sample block; and performing, using the block vector, the conversion in an intra block copy mode which is based on a reconstructed block located in same video region with the current video block

comprising reference samples used for deriving a prediction block of the current video block, wherein, during the conversion, a prediction sample with a location (A, B) from reference samples in a buffer is determined at least according to a size of the buffer and/or the block vector (BVx, BVy).

[0006] In another example aspect, another method of visual media processing is disclosed. The method includes determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, whether a block vector (BVx, BVy) corresponding to the current video block is valid according to a rule, wherein the block vector (BVx, BVy) represents a pixel displacement between the current video block and a sample block; and performing, using the block vector, the conversion based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, wherein the rule specifies that the block vector (BVx, BVy) is valid in case that (1) one or more samples from the sample block are outside the current picture and/or (2) one or more samples from the sample block are outside at least one coding tree unit (CTU) associated with the current video block, and/or (3) one or more samples from the sample block fail to be reconstructed.

[0007] In yet another example aspect, another method of visual media processing is disclosed. The method includes performing a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, wherein, the conversion is based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, and wherein a virtual buffer of a defined size is used for tracking availability of the reference samples for deriving the prediction block.

[0008] In yet another example aspect, another method of visual media processing is disclosed. The method includes maintaining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, a buffer comprising reference samples from the current picture for a derivation of a prediction block of the current video block, wherein one or more reference samples in the buffer that are marked unavailable for the derivation have values outside of a pixel value range.

[0009] In another example aspect, another method of video processing is disclosed. The method includes performing a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data using a buffer comprising reference samples from the current picture for derivation of a prediction block of the current video block, wherein the conversion is based according to rule which specifies that, for the bitstream representation to conform the rule, a reference sample in the buffer is to satisfy a bitstream conformance constraint.

[0010] In yet another example aspect, a video encoder or decoder apparatus comprising a processor configured to implement an above described method is disclosed.

[0011] In another example aspect, a computer readable program medium is disclosed. The medium stores code that embodies processor executable instructions for implementing one of the disclosed methods.

[0012] These, and other, aspects are described in greater details in the present document.

BRIEF DESCRIPTION OF DRAWINGS

[0013] FIG. 1 shows an example of current picture referencing or intra block copy video or image coding technique.

[0014] FIG. 2 shows an example of dynamic reference area.

[0015] FIG. 3 shows an example of coding of a block starting from (x,y).

[0016] FIG. 4 shows examples of possible alternative way to choose the previous coded 64×64 blocks.

[0017] FIG. 5 shows an example of a possible alternative way to change the coding/decoding order of 64×64 blocks.

[0018] FIG. 6 is a flowchart of an example method of video or image processing.

[0019] FIG. 7 is a block diagram of a hardware platform for video or image coding or decoding.

[0020] FIG. 8 shows another possible alternative way to choose the previous coded 64×64 blocks, when the decoding order for 64x64 blocks is from top to bottom, left to right.

[0021] FIG. 9 shows another possible alternative way to choose the previous coded 64×64 blocks.

[0022] FIG. 10 shows an example flowchart for a decoding process with reshaping.

[0023] FIG. 11 shows another possible alternative way to choose the previous coded 64×64 blocks, when the decoding order for 64x64 blocks is from left to right, top to bottom.

[0024] FIG. 12 is an illustration of IBC reference buffer status, where a block denotes a 64x64 CTU.

[0025] FIG. 13 shows one arrangement of reference area for IBC.

[0026] FIG. 14 shows another arrangement of reference area for IBC.

[0027] FIG. 15 shows another arrangement of reference area for IBC when the current virtual pipeline data unit (VPDU) is to the right side of the picture boundary.

[0028] FIG. 16 shows an example of the status of virtual buffer when VPDUs in a CTU row are decoded sequentially.

[0029] FIG. 17 is a block diagram of an example video processing system in which disclosed techniques may be implemented.

[0030] FIG. 18 is a flowchart of an example method of visual media processing.

[0031] FIG. 19 is a flowchart of an example method of visual media processing.

[0032] FIG. 20 is a flowchart of an example method of visual media processing.

[0033] FIG. 21 is a flowchart of an example method of visual media processing.

[0034] FIG. 22 is a flowchart of an example method of visual media processing.

DETAILED DESCRIPTION

[0035] Section headings are used in the present document for ease of understanding and do not limit scope of the disclosed embodiments in each section only to that section. The present document describes various embodiments and techniques for buffer management and block vector coding for intra block copy mode for decoding or encoding video or images.

1. Summary

[0036] This patent document is related to video coding technologies. Specifically, it is related to intra block copy in video coding. It may be applied to the standard under development, e.g. Versatile Video Coding. It may be also applicable to future video coding standards or video codec.

2. Brief Discussion

[0037] 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

[0038] 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 signalled using *inter_pred_idc*. Motion vectors may be explicitly coded as deltas relative to predictors.

[0039] 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 neighbouring 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 signalled explicitly per each PU. Such a mode is named Advanced motion vector prediction (AMVP) in this disclosure.

[0040] When signalling 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.

[0041] When signalling 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.

[0042] 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

[0043] 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. 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.

[0044] 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.

[0045] Fig. 1 is an example illustration of Current Picture Referencing.

[0046] 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 constraints, 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

[0047] 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:

$$offsetX = (ChromaArrayType == 0) ? 0 : (mvCLX[0] \& 0x7 ? 2 : 0) \quad (8\text{-}104)$$

$$offsetY = (ChromaArrayType == 0) ? 0 : (mvCLX[1] \& 0x7 ? 2 : 0) \quad (8\text{-}105)$$

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:

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

[0048] 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 Examples of CPR/IBC

[0049] In a VVC test model, 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.

[0050] 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.

[0051] The chroma block could only be coded with the CPR mode when the following conditions shall be true:

1) each of the luma CU within the collocated luma block shall be coded with CPR mode
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.

[0052] If any of the two condition is false, the chroma block shall not be coded with CPR mode.

[0053] It is noted that the definition of 'valid BV' has the following constraints:

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

2.5 Examples of CPR/IBC

[0054] In some examples, the reference area for CPR/IBC is restricted to the current CTU, which is up to 128x128. The reference area is dynamically changed 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.

[0055] FIG. 2 shows a method, where a block is of 64x64 and a CTU contains 4 64x64 blocks. When coding a 64x64 block, the 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.

[0056] 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 In-loop reshaping (ILR)

[0057] 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).

[0058] 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.).

2.6.1 PWL model

[0059] Conceptually, piece-wise linear (PWL) is implemented in the following way:

[0060] 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:

y = ((y2-y1)/(x2-x1)) * (x-x1) + y1

[0061] In fixed point implementation, the equation can be rewritten as:

$y = ((m * x + 2^{FP\_PREC-1}) >> FP\_PREC) + c$

[0062] where m is scalar, c is an offset, and FP_PREC is a constant value to specify the precision.

[0063] In some examples, 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.

2.6.2.1 Luma reshaping

[0064] A method of the in-loop luma reshaping 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.

[0065] 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. 10 shows the block diagram of the CE12-2 decoding process based on mode.

[0066] 16-piece piece-wise linear (PWL) models are tested for luma and chroma residue scaling instead of the 32-piece PWL models.

[0067] Inter slice reconstruction with in-loop luma reshaper (light-green shaded blocks indicate signal in reshaped domain: luma residue; intra luma predicted; and intra luma reconstructed).

2.6.2.2 Luma-dependent chroma residue scaling

[0068] 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:

- For intra, the reconstructed luma is averaged.
- For inter, the prediction luma is averaged.

[0069] 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.

[0070] It is noted that the chroma scaling factor is calculated from forward-mapped predicted luma values rather than reconstructed luma values.

2.6.2.3 Signalling of ILR side information

[0071] The parameters are (currently) sent in the tile group header (similar to ALF). These reportedly take 40-100 bits.

[0072] In some examples, the added syntax is highlighted in italics.

[0073] 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.

[0074] When reshape_model_bin_delta_sign_CW_flag[i] is not present, it is inferred to be equal to 0.

[0075] The variable RspDeltaCW[i] = (1 2*reshape_model_bin_delta_sign_CW [i]) * reshape_model_bin_delta_abs_CW [i];

[0076] The variable RspCW[i] is derived as following steps:

[0077] 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.

[0078] 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.

[0079] The variables InputPivot[i] with i in the range of 0 to MaxBinIdx + 1, inclusive are derived as follows

InputPivot[i] = i * OrgCW

[0080] 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 ]
}
```

[0081] 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]

2.6.2.4 Usage of ILR

[0082] 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 residual are generated and coded to the bitstream.

[0083] 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.

[0084] Forward reshaping to the prediction signal is disabled for the following cases:

[0085] Current block is intra-coded

[0086] Current block is coded as CPR (current picture referencing, aka intra block copy, IBC)

[0087] Current block is coded as combined inter-intra mode (CIIP) and the forward reshaping is disabled for the intra prediction block

3. Examples of problems solved by various embodiments

[0088] In the current design of CPR/IBC, some problems exist.

1) The reference area changes dynamically, which makes encoder/decoder processing complicated.
2) Invalid block vectors are easily generated and difficult to check, which complicates both encoder and decoder.
3) Irregular reference area leads to inefficient coding of block vector.
4) How to handle CTU size smaller than 128x128 is not clear.
5) In the determination process of whether a BV is valid or invalid, for chroma blocks, the decision is based on the luma sample's availability which may result in wrong decisions due to the dual tree partition structure.

4. Example embodiments

[0089] In some embodiments, a regular buffer can be used for CPR/IBC block to get reference.

[0090] A function isRec(x,y) is defined to indicate if pixel (x,y) has been reconstructed and be referenced by IBC mode. When (x,y) is out of picture, of different slice/tile/brick, isRec(x,y) return false; when (x,y) has not been reconstructed, isRec(x,y) returns false. In another example, when sample (x,y) has been reconstructed but some other conditions are satisfied, it may also be marked as unavailable, such as out of the reference area/in a different VPDU, and isRec(x,y) returns false.

[0091] A function isRec(c, x,y) is defined to indicate if sample (x,y) for component c is available. For example, if the sample (x, y) hasn't been reconstructed yet, it is marked as unavailable. In another example, when sample (x,y) has been reconstructed but some other conditions are satisfied, it may also be marked as unavailable, such as it is out of picture/in a different slice/tile/brick/in a different VPDU,

out of allowed reference area. isRec(c, x,y) returns false when sample (x, y) is unavailable, otherwise, it returns true.

[0092] In the following discussion, the reference samples can be reconstructed samples. It is noted that 'pixel buffer' may response to 'buffer of one color component' or 'buffer of multiple color components'.

Reference buffer for CPR/IBC

1. It is proposed to use a M×N pixel buffer to store the luma reference samples for CPR/IBC.
    a. In one example, the buffer size is 64x64.
    b. In one example, the buffer size is 128x128.
    c. In one example, the buffer size is 64x128.
    d. In one example, the buffer size is 128x64.
    e. In one example, N equals to the height of a CTU.
    f. In one example, N=nH, where H is the height of a CTU, n is a positive integer.
    g. In one example, M equals to the width of a CTU.
    h. In one example, M=mW, where W is the width of a CTU, m is a positive integer.
    i. In one example, the buffer size is unequal to the CTU size, such as 96x128 or 128x96.
    j. In one example, the buffer size is equal to the CTU size
    k. In one example, M=mW and N=H, where W and H are width and height of a CTU, m is a positive integer.
    l. In one example, M=W and N=nH, where W and H are width and height of a CTU, n is a positive integer.
    m. In one example, M=mW and N=nH, where W and H are width and height of a CTU, m and n are positive integers.
    n. In above example, m and n may depend on CTU size.
        i. In one example, when CTU size is 128x128, m=1 and n=1.
        ii. In one example, when CTU size is 64x64, m=4 and n=1.
        iii. In one example, when CTU size is 32x32, m=16 and n=1.
        iv. In one example, when CTU size is 16x16, m=64 and n=1.
    o. Alternatively, the buffer size corresponds to CTU size.

p. Alternatively, the buffer size corresponds to a Virtual Pipeline Data Unit (VPDU) size.

q. M and/or N may be signaled from the encoder to the decoder, such as in VPS/SPS/PPS/picture header/slice header/tile group header.

2. M and/or N may be different in different profiles/levels/tiers defined in a standard. It is proposed to use another Mc×Nc pixel buffer to store the chroma reference samples for CPR/IBC.

a. In one example, Mc = M/2 and Nc = N/2 for 4:2:0 video

b. In one example, Mc = M and Nc = N for 4:4:4 video

c. In one example, Mc = M and Nc = N/2 for 4:2:2 video

d. Alternatively, Mc and Nc can be independent of M and N.

e. In one example, the chroma buffer includes two channels, corresponding to Cb and Cr.

f. In one example, Mc=M and Nc=N.

3. It is proposed to use a M×N sample buffer to store the RGB reference samples for CPR/IBC

a. In one example, the buffer size is 64x64.

b. In one example, the buffer size is 128x128.

c. In one example, the buffer size is 64x128.

d. In one example, the buffer size is 128x64.

e. Alternatively, the buffer size corresponds to CTU size.

f. Alternatively, the buffer size corresponds to a Virtual Pipeline Data Unit (VPDU) size.

4. It is proposed that the buffer can store reconstructed pixels before loop-filtering. Loop-filtering may refer to deblocking filter, adaptive loop filter (ALF), sample adaptive offset (SAO), a cross-component ALF, or any other filters.

a. In one example, the buffer can store samples in the current CTU.

b. In one example, the buffer can store samples outside of the current CTU.

c. In one example, the buffer can store samples from any part of the current picture.

d. In one example, the buffer can store samples from other pictures.

5. It is proposed that the buffer can store reconstructed pixels after loop-filtering. Loop-filtering may refer to deblocking filter, adaptive loop filter (ALF), sample adaptive offset (SAO), a cross-component ALF, or any other filters.

a. In one example, the buffer can store samples in the current CTU.

b. In one example, the buffer can store samples outside of the current CTU.

c. In one example, the buffer can store samples from any part of the current picture.

d. In one example, the buffer can store samples from other pictures.

6. It is proposed that the buffer can store both reconstructed samples before loop-filtering and after loop-filtering. Loop-filtering may refer to deblocking filter, adaptive loop filter (ALF), sample adaptive offset (SAO), a cross-component ALF, or any other filters.

a. In one example, the buffer can store both samples from the current picture and samples from other pictures, depending on the availability of those samples.

b. In one example, reference samples from other pictures are from reconstructed samples after loop-filtering.

c. In one example, reference samples from other pictures are from reconstructed samples before loop-filtering.

7. It is proposed that the buffer stores samples with a given bit-depth which may be different from the bit-depth for coded video data.

a. In one example, the bit-depth for the reconstruction buffer/coded video data is larger than that for IBC reference samples stored in the buffer.

b. In one example, even when the internal bit-depth is different from the input bit-depth for a video sequence, such as (10 bits vs 8 bits), the IBC reference samples are stored to be aligned with the input bit-depth.

c. In one example, the bit-depth is identical to that of the reconstruction buffer.

d. In one example, the bit-depth is identical to that of input image/video.

e. In one example, the bit-depth is identical to a predefine number.

f. In one example, the bit-depth depends on profile of a standard.

g. In one example, the bit-depth or the bit-depth difference compared to the output bit-depth/input bit-depth/internal bit-depth may be signalled in SPS/PPS/sequence header/picture header/slice header/Tile group header/Tile header or other kinds of video data units.

h. The proposed methods may be applied with the proposed buffer definitions mentioned in other bullets, alternatively, they may be also applicable to existing design of IBC.

i. The bit-depth of each color component of the buffer may be different.

Buffer initiation

8. It is proposed to initialize the buffer with a given value
    a. In one example, the buffer is initialized with a given value.
        i. In one example, the given value may depend on the input bit-depth and/or internal bit-depth.
        ii. In one example, the buffer is initialized with mid-grey value, e.g. 128 for 8-bit signal or 512 for 10-bit signal.
        iii. In one example, the buffer is initialized with forwardLUT(m) when ILR is used. E.g. m= 1<<(Bitdepth-1).
    b. Alternatively, the buffer is initialized with a value signalled in SPS/VPS/APS/PPS/sequence header/Tile group header/Picture header /tile/CTU/Coding unit/VPDU/region.
    c. In one example, the given value may be derived from samples of previously decoded pictures or slices or CTU rows or CTUs or CUs.
    d. The given value may be different for different color component.
9. Alternatively, it is proposed to initialize the buffer with decoded pixels from previously coded blocks.
    a. In one example, the decoded pixels are those before in-loop filtering.
    b. In one example, when the buffer size is a CTU, the buffer is initialized with decoded pixels of the previous decoded CTU, if available.
    c. In one example, when the buffer size is of 64x64, its buffer size is initialized with decoded pixels of the previous decoded 64x64 block, if available.
    d. Alternatively, furthermore, if no previously coded blocks are available, the methods in bullet 8 may be applied.

Reference to the buffer

10. For a block to use pixels in the buffer as reference, it can use a position (x,y), x=0,1,2,…,M-1;y=0,1,2,…,N-1, within the buffer to indicate where to get reference.
11. Alternatively, the reference position can be denoted as l = y*M+x, l=0,1,…,M*N-1.
12. Denote that the upper-left position of a block related to the current CTU as (x0,y0), a block vector (BVx,BVy)=(x-x0,y-y0) may be sent to the decoder to indicate where to get reference in the buffer.

13. Alternatively, a block vector (BVx,BVy) can be defined as (x-x0+Tx,y-y0+Ty) where Tx and Ty are predefined offsets.

14. For any pixel (x0, y0) and (BVx, BVy), its reference in the buffer can be found at (x0+BVx, y0+BVy)

    a. In one example, when (x0+BVx, y0+BVy) is outside of the buffer, it will be clipped to the boundary.

    b. Alternatively, when (x0+BVx, y0+BVy) is outside of the buffer, its reference value is predefined as a given value, e.g. mid-grey.

    c. Alternatively, the reference position is defined as ((x0+BVx) mod M, (y0+BVy) mod N) so that it is always within the buffer.

15. For any pixel (x0, y0) and (BVx, BVy), when (x0+BVx, y0+BVy) is outside of the buffer, its reference value may be derived from the values in the buffer.

    a. In one example, the value is derived from the sample ((x0+BVx) mod M, (y0+BVy) mod N) in the buffer.

    b. In one example, the value is derived from the sample ((x0+BVx) mod M, clip(y0+BVy, 0, N-1)) in the buffer.

    c. In one example, the value is derived from the sample (clip(x0+BVx, 0, M-1), (y0+BVy) mod N) in the buffer.

    d. In one example, the value is derived from the sample (clip(x0+BVx, 0, M-1), clip(y0+BVy, 0, N-1)) in the buffer.

16. It may disallow a certain coordinate outside of the buffer range

    a. In one example, for any pixel (x0, y0) relative to the upperleft corner of a CTU and block vector (BVx, BVy), it is a bitstream constraint that y0+BVy should be in the range of [0,…,N-1].

    b. In one example, for any pixel (x0, y0) relative to the upperleft corner of a CTU and block vector (BVx, BVy), it is a bitstream constraint that x0+BVx should be in the range of [0,…,M-1].

    c. In one example, for any pixel (x0, y0) relative to the upperleft corner of a CTU and block vector (BVx, BVy), it is a bitstream constraint that both y0+BVy should be in the range of [0,…,N-1] and x0+BVx should be in the range of [0,…,M-1].

17. When the signalled or derived block vector of one block points to somewhere outside the buffer, padding may be applied according to the buffer.

a. In one example, the value of any sample outside of the buffer is defined with a predefined value.

i. In one example, the value can be 1<<(Bitdepth-1), e.g. 128 for 8-bit signals and 512 for 10-bit signals.

ii. In one example, the value can be forwardLUT(m) when ILR is used. E.g. m= 1<<(Bitdepth-1).

iii. Alternatively, indication of the predefined value may be signalled or indicated at SPS/ PPS/sequence header/picture header/slice header/Tile group/Tile/CTU/CU level.

b. In one example, any sample outside of the buffer is defined as the value of the nearest sample in the buffer.

18. The methods to handle out of the buffer reference may be different horizontally and vertically or may be different according to the location of the current block (e.g., closer to picture boundary or not).

a. In one example, when y0+BVy is outside of [0, N-1], the sample value of (x0+BVx, y0+BVy) is assigned as a predefined value.

b. In one example, when x0+BVx is outside of [0, M-1], the sample value of (x0+BVx, y0+BVy) is assigned as a predefined value.

c. Alternatively, the sample value of (x0+BVx, y0+BVy) is assigned as the sample value of ((x0+BVx)mod M, y0+BVy), which may invoke other method to further derive the value if ((x0+BVx)mod M, y0+BVy) is still outside of the buffer.

d. Alternatively, the sample value of (x0+BVx, y0+BVy) is assigned as the sample value of (x0+BVx, (y0+BVy) mod N), which may invoke other method to further derive the value if (x0+BVx, (y0+BVy) mod N) is still outside of the buffer.

Block vector representation

19. Each component of a block vector (BVx, BVy) or one of the component may be normalized to a certain range.

a. In one example, BVx can be replaced by (BVx mod M).

b. Alternatively, BVx can be replaced by ((BVx+X) mod M)-X, where X is a predefined value.

i. In one example, X is 64.

ii. In one example, X is M/2;

iii. In one example, X is the horizontal coordinate of a block relative to the current CTU.

c. In one example, BVy can be replaced by (BVy mod N).

d. Alternatively, BVy can be replaced by ((BVy+Y) mod N)-Y, where Y is a predefined value.

i. In one example, Y is 64.

ii. In one example, Y is N/2;

iii. In one example, Y is the vertical coordinate of a block relative to the current CTU.

20. BVx and BVy may have different normalized ranges.

21. A block vector difference (BVDx, BVDy) can be normalized to a certain range.

a. In one example, BVDx can be replaced by (BVDx mod M) wherein the function mod returns the reminder.

b. Alternatively, BVDx can be replaced by ((BVDx+X) mod M)-X, where X is a predefined value.

i. In one example, X is 64.

ii. In one example, X is M/2;

c. In one example, BVy can be replaced by (BVDy mod N).

d. Alternatively, BVy can be replaced by ((BVDy+Y) mod N)-Y, where Y is a predefined value.

i. In one example, Y is 64.

ii. In one example, Y is N/2;

22. BVDx and BVDy may have different normalized ranges.

Validity check for a block vector

[0093] Denote the width and height of an IBC buffer as W_{buf} and H_{buf}. For a WxH block (may be a luma block, chroma block, CU, TU, 4x4, 2x2, or other subblocks) starting from (X, Y) relative to the upper-left corner of a picture, the following may apply to tell if a block vector (BVx, BVy) is valid or not. Let W_{pic} and H_{pic} be the width and height of a picture and; W_{ctu} and H_{ctu} be the width and height of

a CTU. Function floor(x) returns the largest integer no larger than x. Function isRec(x, y) returns if sample (x, y) has been reconstructed.

23. Block vector (BVx, BVy) may be set as valid even if any reference position is outside of picture boundary.
    a. In one example, the block vector may be set as valid even if $X+BVx < 0$.
    b. In one example, the block vector may be set as valid even if $X+W+BVx > W_{pic}$.
    c. In one example, the block vector may be set as valid even if $Y+BVy < 0$.
    d. In one example, the block vector may be set as valid even if $Y+H+BVy > H_{pic}$.
24. Block vector (BVx, BVy) may be set as valid even if any reference position is outside of the current CTU row.
    a. In one example, the block vector may be set as valid even if $Y+BVy<floor(Y/ H_{ctu})* H_{ctu}$.
    b. In one example, the block vector may be set as valid even if $Y+H+BVy>=floor(Y/ H_{ctu})*H_{ctu}+ H_{ctu}$.
25. Block vector (BVx, BVy) may be set as valid even if any reference position is outside of the current and left (n-1) CTUs, where n is the number of CTUs (including or excluding the current CTU) that can be used as reference area for IBC.
    a. In one example, the block vector may be set as valid even if $X+BVx<floor(X/W_{ctu})* W_{ctu} - (n-1)* W_{ctu}$.
    b. In one example, the block vector may be set as valid even if $X+W+BVx > floor(X/W_{ctu})* W_{ctu} + W_{ctu}$
26. Block vector (BVx, BVy) may be set as valid even if a certain sample has not been reconstructed.
    a. In one example, the block vector may be set as valid even if isRec(X+BVx, Y+ BVy) is false.
    b. In one example, the block vector may be set as valid even if isRec(X+BVx +W-1, Y+BVy) is false.
    c. In one example, the block vector may be set as valid even if isRec(X+BVx, Y+BVy +H-1) is false.
    d. In one example, the block vector may be set as valid even if isRec(X+BVx +W-1, Y+BVy +H-1) is false.
27. Block vector (BVx, BVy) may be always set as valid when a block is not of the 1st CTU in a CTU row.

a. Alternatively, the block vector may be always set as valid.

28. Block vector (BVx, BVy) may be always set as valid when the following 3 conditions are all satisfied

- X + BVx >= 0
- Y + BVy >= floor(Y / H_{ctu})
- isRec(X + BVx + W - 1, Y + BVy + H - 1) == true

a. Alternatively, when the three conditions are all satisfied for a block of the 1st CTU in a CTU row, the block vector may be always set as valid.

29. When a block vector (BVx, BVy) is valid, sample copying for the block may be based on the block vector.

a. In one example, prediction of sample (X, Y) may be from ((X+BVx)%W_{buf}, (Y+BVy)%H_{buf})

Buffer update

30. When coding a new picture or tile, the buffer may be reset.

a. The term "reset" may refer that the buffer is initialized.

b. The term "reset" may refer that all samples/pixels in the buffer is set to a given value (e.g., 0 or -1).

31. When finishing coding of a VPDU, the buffer may be updated with the reconstructed values of the VPDU.

32. When finishing coding of a CTU, the buffer may be updated with the reconstructed values of the CTU.

a. In one example, when the buffer is not full, the buffer may be updated CTU by CTU sequentially.

b. In one example, when the buffer is full, the buffer area corresponding to the oldest CTU will be updated.

c. In one example, when M=mW and N=H (W and H are CTU size; M and N are the buffer size) and the previous updated area started from (kW, 0), the next starting position to update will be ((k+1)W mod M, 0).

33. The buffer can be reset at the beginning of each CTU row.

a. Alternatively, the buffer may be reset at the beginning of decoding each CTU.

b. Alternatively, the buffer may be reset at the beginning of decoding one tile.

c. Alternatively, the buffer may be reset at the beginning of decoding one tile group/picture.

34. When finishing coding a block starting from (x,y), the buffer's corresponding area, starting from (x,y) will be updated with reconstruction from the block.

a. In one example, (x,y) is a position relative to the upper-left corner of a CTU.

35. When finishing coding a block relative to the picture, the buffer's corresponding area will be updated with reconstruction from the block.

a. In one example, the value at position (x mod M, y mod N) in the buffer may be updated with the reconstructed pixel value of position (x, y) relative to the upper-left corner of the picture.

b. In one example, the value at position (x mod M, y mod N) in the buffer may be updated with the reconstructed pixel value of position (x, y) relative to the upper-left corner of the current tile.

c. In one example, the value at position (x mod M, y mod N) in the buffer may be updated with the reconstructed pixel value of position (x, y) relative to the upper-left corner of the current CTU row.

d. In one example, the value in the buffer may be updated with the reconstructed pixel values after bit-depth alignment.

36. When finishing coding a block starting from (x,y), the buffer's corresponding area, starting from (xb,yb) will be updated with reconstruction from the block wherein (xb, yb) and (x, y) are two different coordinates

a. In one example, (x,y) is a position related to the upper-left corner of a CTU, and (xb, yb) is (x+update_x, y+update_y), wherein update_x and update_y point to a updatable position in the buffer.

37. For above examples, the reconstructed values of a block may indicate the reconstructed values before filters (e.g., deblocking filter) applied.

a. Alternatively, the reconstructed values of a block may indicate the reconstructed values after filters (e.g., deblocking filter) applied.

38. When the buffer is updated from reconstructed samples, the reconstructed samples may be firstly modified before being stored, such as sample bit-depth can be changed.

a. In one example, the buffer is updated with reconstructed sample value after bit-depth alignment to the bitdepth of the buffer.

b. In one example, the buffer value is updated according to the value {p+[1<<(b-1)]}>>b, where p is reconstructed sample value, b is a predefined bit-shifting value.

c. In one example, the buffer value is updated according to the value clip({p+[1<<(b-1)]}>>b, 0, (1<<bitdepth)-1), where p is reconstructed sample value, b is a predefined bit-shifting value, bitdepth is the buffer bit-depth.

d. In one example, the buffer value is updated according to the value {p+[1<<(b-1)-1]}>>b, where p is reconstructed sample value, b is a predefined bit-shifting value.

e. In one example, the buffer value is updated according to the value clip({p+[1<<(b-1)-1]}>>b, 0, (1<<bitdepth)-1), where p is reconstructed sample value, b is a predefined bit-shifting value, bitdepth is the buffer bit-depth.

f. In one example, the buffer value is updated according to the value p>>b.

g. In one example, the buffer value is updated according to the value clip(p>>b, 0, (1<<bitdepth)-1), where bitdepth is the buffer bit-depth.

h. In the above examples, b can be reconstructed bit-depth minus input sample bit-depth.

39. When use the buffer samples to form prediction, a preprocessing can be applied.

a. In one example, the prediction value is p<<b, where p is a sample value in the buffer, and b is a predefined value.

b. In one example, the prediction value is clip(p<<b, 0, 1<<bitdepth), where bitdepth is the bit-depth for reconstruction samples.

c. In one example, the prediction value is (p<<b)+(1<<(bitdepth-1)), where p is a sample value in the buffer, and b is a predefined value, bitdepth is the bit-depth for reconstruction samples.

d. In the above examples, b can be reconstructed bit-depth minus input sample bit-depth.

40. The buffer can be updated in a given order.

a. In one example, the buffer can be updated sequentially.

b. In one example, the buffer can be updated according to the order of blocks reconstructed.

41. When the buffer is full, the samples in the buffer can be replaced with latest reconstructed samples.

a. In one example, the samples can be updated in a first-in-first-out manner.

b. In one example, the oldest samples will be replaced.

c. In one example, the samples can be assigned a priority and replaced according to the priority.

d. In one example, the samples can be marked as "long-term" so that other samples will be replaced first.

e. In one example, a flag can be sent along with a block to indicate a high priority.

f. In one example, a number can be sent along with a block to indicate priority.

g. In one example, samples from a reconstructed block with a certain characteristic will be assign a higher priority so that other samples will be replace first.

   i. In one example, when the percentage of samples coded in IBC mode is larger than a threshold, all samples of the block can be assigned a high priority.

   ii. In one example, when the percentage of samples coded in Palette mode is larger than a threshold, all samples of the block can be assigned a high priority.

   iii. In one example, when the percentage of samples coded in IBC or Palette mode is larger than a threshold, all samples of the block can be assigned a high priority.

   iv. In one example, when the percentage of samples coded in transform-skip mode is larger than a threshold, all samples of the block can be assigned a high priority.

   v. The threshold can be different according to block-size, color component, CTU size.

   vi. The threshold can be signalled in SPS/ PPS/sequence header/slice header/Tile group/Tile level/a region.

h. In one example, that buffer is full may mean that the number of available samples in the buffer is equal or larger than a given threshold.

   i. In one example, when the number of available samples in the buffer is equal or larger than 64x64x3 luma samples, the buffer may be determined as full.

Alternative buffer combination

42. Instead of always using the previously coded three 64x64 blocks as a reference region, it is proposed to adaptively change it based on current block (or VPDU)'s location.

    a. In one example, when coding/decoding a 64×64 block, previous 3 64×64 blocks can be used as reference. Compared to FIG. 2, more kinds of combination of previous 64×64 blocks can be applied. Figure 2 shows an example of a different combination of previous 64×64 blocks.

43. Instead of using the z-scan order, vertical scan order may be utilized instead.

    a. In one example, when one block is split into 4 VPDUs with index 0..3 in z-scan order, the encoding/decoding order is 0, 2, 1, 3.

    b. In one example, when coding/decoding a 64×64 blocks, previous 3 64×64 blocks can be used as reference. Compared to FIG. 2, more kind of coding/decoding orders of 64×64 blocks can be applied. Figure 4 shows an example of a different coding/decoding order of 64×64 blocks.

    c. Alternatively, above methods may be applied only for screen content coding

    d. Alternatively, above methods may be applied only when CPR is enabled for one tile/tile group/picture.

    e. Alternatively, above methods may be applied only when CPR is enabled for one CTU or one CTU row.

Virtual IBC buffer

[0094] The following, the width and height of a VPDU is denoted as W_{VPDU} (e.g., 64) and H_{VPDU} (e.g., 64), respectively in luma samples. Alternatively, W_{VPDU} and/or H_{VPDU} may denote the width and/or height of other video unit (e.g., CTU).

44. A virtual buffer may be maintained to keep track of the IBC reference region status.

    a. In one example, the virtual buffer size is m W_{VPDU} x n H_{VPDU}.

        i. In one example, m is equal to 3 and n is equal to 2.

        ii. In one example, m and/or n may depend on the picture resolution, CTU sizes.

        iii. In one example, m and/or n may be signaled or pre-defined.

    b. In one example, the methods described in above bullets and sub-bullets may be applied to the virtual buffer.

    c. In one example, a sample (x, y) relative to the upper-left corner of the picture/slice/tile/brick may be mapped to (x%(mW_{VPDU}), y%(nH_{VPDU}))

45. An array may be used to track the availability of each sample associated with the virtual buffer.

    a. In one example, a flag may be associated with a sample in the virtual buffer to specify if the sample in the buffer can be used as IBC reference or not.

    b. In one example, each 4x4 block containing luma and chroma samples may share a flag to indicate if any samples associated with that block can be used as IBC reference or not.

c. In one example, an array corresponding to 3x2 VPDUs (e.g., each 4x4 block may share the same availability flag) maintained to track availability of IBC reference samples.

d. In one example, an array corresponding to 4x2 VPDUs (e.g., each 4x4 block may share the same availability flag) maintained to track availability of IBC reference samples.

46. After finishing decoding a VPDU or a video unit, certain samples associated with the virtual buffer may be marked as unavailable for IBC reference.

a. In one example, which samples may be marked as unavailable depend on the position of the most recently decoded VPDU.

b. When one sample is marked unavailable, prediction from the sample is disallowed.

i. Alternatively, other ways (e.g., using default values) may be further applied to derive a predictor to replace the unavailable sample.

47. The position of most recently decoded VPDU may be recorded to help to identify which samples associated with the virtual buffer may be marked as unavailable.

a. In one example, at the beginning of decoding a VPDU, certain samples associated with the virtual buffer may be marked as unavailable according to the position of most recently decoded VPDU.

i. In one example, denote (xPrevVPDU, yPrevVPDU) as the upper-left position relative to the upper-left corner of the picture/slice/tile/brick/other video processing unit of most recently decoded VPDU, if yPrevVPDU%(n H_{VPDU}) is equal to 0, certain positions (x, y) may be marked as unavailable.

1. In one example, x may be within a range, such as [xPrevVPDU - $2W_{VPDU}$+ $2mW_{VPDU}$)% mW_{VPDU}, ((xPrevVPDU - 2 W_{VPDU} + 2m W_{VPDU})% mW_{VPDU})-1+W_{VPDU}];

2. In one example, y may be within a range, such as [yPrevVPDU%(n H_{VPDU}), (yPrevVPDU%(nH_{VPDU}))-1+ H_{VPDU}];

3. In one example, x may be within a range, such as [xPrevVPDU - $2W_{VPDU}$+ $2mW_{VPDU}$)% mW_{VPDU}, ((xPrevVPDU - $2W_{VPDU}$+ $2mW_{VPDU}$)% mW_{VPDU})-1+W_{VPDU}] and y may be within a range, such as [yPrevVPDU%(n H_{VPDU}), (yPrevVPDU%(n H_{VPDU}))-1+ H_{VPDU}].

ii. In one example, denote (xPrevVPDU, yPrevVPDU) as the upper-left position relative to the upper-left corner of the picture/slice/tile/brick/other video

processing unit of most recently decoded VPDU, if yPrevVPDU%(n H_{VPDU}) is not equal to 0, certain positions (x, y) may be marked as unavailable.

1. In one example, x may be within a range, such as [xPrevVPDU - W_{VPDU}+ 2mW_{VPDU})% mW_{VPDU}, ((xPrevVPDU - W_{VPDU}+ 2mW_{VPDU})% mW_{VPDU})-1+W_{VPDU}];
2. In one example, y may be within a range, such as [yPrevVPDU%(n H_{VPDU}), (yPrevVPDU%(n H_{VPDU}))-1+ H_{VPDU}]
3. In one example, x may be within a range, such as [xPrevVPDU - W_{VPDU} + 2mW_{VPDU})% mW_{VPDU}, ((xPrevVPDU - W_{VPDU}+ 2mW_{VPDU})% mW_{VPDU})-1+W_{VPDU}] and y may be within a range, such as [yPrevVPDU%(n H_{VPDU}), (yPrevVPDU%(n H_{VPDU}))-1+ H_{VPDU}].

48. When a CU contains multiple VPDUs, instead of applying IBC reference availability marking process according to VPDU, the IBC reference availability marking process may be according to the CU
    a. In one example, at the beginning of decoding a CU containing multiple VPDUs, the IBC reference availability marking process may be applied for each VPDU before the VPDU within the CU is decoded.
    b. In such a case, 128x64 and 64x128 IBC blocks may be disallowed.
        i. In one example, pred_mode_ibc_flag for 128x64 and 64x128 CUs may not be sent and may be inferred to equal to 0.
49. For a reference block or sub-block, the reference availability status of the upper-right corner may not need to be checked to tell if the block vector associated with the reference block is valid or not.
    a. In one example, only the upper-left, bottom-left and bottom-right corner of a block/sub-block will be checked to tell if the block vector is valid or not.
50. The IBC buffer size may depend on VPDU size (wherein the width/height is denoted by vSize) and/or CTB/CTU size (wherein the width/height is denoted by ctbSize)
    a. In one example, the height of the buffer may be equal to ctbSize.
    b. In one example, the width of the buffer may depend on min(ctbSize, 64)
        i. In one example, the width of the buffer may be (128*128/vSize, min(ctbSize, 64))

51. An IBC buffer may contain values outside of pixel range, which indicates that the position may not be available for IBC reference, e.g., not utilized for predicting other samples.
    a. A sample value may be set to a value which indicates the sample is unavailable.
    b. In one example, the value may be -1.
    c. In one example, the value may be any value outside of [0, 1<<(internal_bit_depth) – 1] wherein internal_bit_depth is a positive integer value. For example, internal_bit_depth is the internal bitdepth used for encoding/decoding a sample for a color component.
    d. In one example, the value may be any value outside of [0, 1<<(input_bit_depth) – 1] wherein input_bit_depth is a positive integer value. For example, input_bit_depth is the input bitdepth used for encoding/decoding a sample for a color component.
52. Availability marking for samples in the IBC buffer may depend on position of the current block, size of the current block, CTU/CTB size and VPDU size. In one example, let (xCb, yCb) denotes the block's position relative to top-left of the picture; ctbSize is the size (i.e., width and/or height) of a CTU/CTB; vSize= min(ctbSize, 64); wIbcBuf and hIbcBuf are the IBC buffer width and height.
    a. In one example, if (xCb%vSize) is equal to 0 and (yCb%vSize) is equal to 0, a certain set of positions in the IBC buffer may be marked as unavailable.
    b. In one example, when the current block size is smaller than the VPDU size, i.e. min(ctbSize, 64), the region marked as unavailable may be according to the VPDU size.
    c. In one example, when the current block size is larger than the VPDU size, i.e. min(ctbSize, 64), the region marked as unavailable may be according to the CU size.
53. At the beginning of decoding a video unit (e.g., VPDU (xV, yV)) relative to the top-left position of a picture, corresponding positions in the IBC buffer may be set to a value outside of pixel range.
    a. In one example, buffer samples with position (x%wIbcBuf, y%hIbcBuf) in the buffer, with x = xV, …,xV+ctbSize-1 and y=yV,…,yV+ctbSize-1, will be set to value -1. Where wIbcBuf and hIbcBuf are the IBC buffer width and height, ctbSize is the width of a CTU/CTB.
        i. In one example, hIbcBuf may be equal to ctbSize.
54. A bitstream conformance constraint may be according to the value of a sample in the IBC buffer
    a. In one example, if a reference block associated with a block vector in IBC buffer contains value outside of pixel range, the bitstream may be illegal.

55. A bitstream conformance constraint may be set according to the availability indication in the IBC buffer.
    a. In one example, if any reference sample mapped in the IBC buffer is marked as unavailable for encoding/decoding a block, the bitstream may be illegal.
    b. In one example, when singletree is used, if any luma reference sample mapped in the IBC buffer for encoding/decoding a block is marked as unavailable, the bitstream may be illegal.
    c. A conformance bitstream may satisfy that for an IBC coded block, the associated block vector may point to a reference block mapped in the IBC buffer and each luma reference sample located in the IBC buffer for encoding/decoding a block shall be marked as available (e.g., the values of samples are within the range of [K0, K1] wherein for example, K0 is set to 0 and K1 is set to (1<<BitDepth-1) wherein BitDepth is the internal bit-depth or the input bit-depth).
56. Bitstream conformance constraints may depend on partitioning tree types and current CU's coding treeType
    a. In one example, if dualtree is allowed in high-level (e.g., slice/picture/brick/tile) and the current video block (e.g., CU/PU/CB/PB) is coded with single tree, bitstreams constraints may need to check if all components' positions mapped in the IBC buffer is marked as unavailable or not.
    b. In one example, if dualtree is allowed in high-level (e.g., slice/picture/brick/tile) and the current luma video block (e.g., CU/PU/CB/PB) is coded with dual tree, bitstreams constraints may neglect chroma components' positions mapped in the IBC buffer is marked as unavailable or not.
        i. Alternatively, in such a case, bitstreams constraints may still check all components' positions mapped in the IBC buffer is marked as unavailable or not.
    c. In one example, if single tree is used, bitstreams constraints may neglect chroma components' positions mapped in the IBC buffer is marked as unavailable or not.

Improvement to the current VTM design

57. The prediction for IBC can have a lower precision than the reconstruction.
    a. In one example, the prediction value is according to the value clip{{p+[1<<(b-1)]}>>b,0,(1<<bitdepth)-1}<<b, where p is reconstructed sample value, b is a predefined bit-shifting value, bitdepth is prediction sample bit-bitdepth.

b. In one example, the prediction value is according to the value clip{{p+[1<<(b-1)-1]}>>b,0,(1<<bitdepth)-1}<<b, where p is reconstructed sample value, b is a predefined bit-shifting value.

c. In one example, the prediction value is according to the value ((p>>b)+(1<<(bitdepth-1)))<<b, where bitdepth is prediction sample bit-bitdepth.

d. In one example, the prediction value is according to the value (clip((p>>b),0,(1<<(bitdepth-b)))+(1<<(bitdepth-1)))<<b, where bitdepth is prediction sample bit-bitdepth.

e. In one example, the prediction value is clipped in different ways depending on whether ILR is applied or not.

f. In the above examples, b can be reconstructed bit-depth minus input sample bit-depth.

g. In one example, the bit-depth or the bit-depth difference compared to the output bit-depth/input bit-depth/internal bit-depth may be signalled in SPS/PPS/sequence header/picture header/slice header/Tile group header/Tile header or other kinds of video data units.

58. Part of the prediction of IBC can have a lower precision and the other part has the same precision as the reconstruction.

a. In one example, the allowed reference area may contain samples with different precisions (e.g., bit-depth).

b. In one example, reference from other 64x64 blocks than the current 64x64 block being decoded is of low precision and reference from the current 64x64 block has the same precision as the reconstruction.

c. In one example, reference from other CTUs than the current CTU being decoded is of low precision and reference from the current CTU has the same precision as the reconstruction.

d. In one example, reference from a certain set of color components is of low precision and reference from the other color components has the same precision as the reconstruction.

59. When CTU size is MxM and reference area size is nMxnM, the reference area is the nearest available nxn CTU in a CTU row.

a. In one example, when reference area size is 128x128 and CTU size is 64x64, the nearest available 4 CTUs in a CTU row can be used for IBC reference.

b. In one example, when reference area size is 128x128 and CTU size is 32x32, the nearest available 16 CTUs in a CTU row can be used for IBC reference.

60. When CTU size is M and reference area size is nM, the reference area is the nearest available n-1 CTUs in a CTU row/tile.

a. In one example, when reference area size is 128x128 or 256x64 and CTU size is 64x64, the nearest available 3 CTUs in a CTU row can be used for IBC reference.

b. In one example, when reference area size is 128x128 or 512x32 and CTU size is 32x32, the nearest available 15 CTUs in a CTU row can be used for IBC reference.

61. When CTU size is M, VPDU size is kM and reference area size is nM, and the reference area is the nearest available n-k CTUs in a CTU row/tile.

a. In one example, CTU size is 64x64, VPDU size is also 64x64, reference are size is 128x128, the nearest 3 CTUs in a CTU row can be used for IBC reference.

b. In one example, CTU size is 32x32, VPDU size is also 64x64, reference are size is 128x128, the nearest (16-4)=12 CTUs in a CTU row can be used for IBC reference.

62. For a w x h block with upper-left corner being (x, y) using IBC, there are constraints that keep reference block from certain area for memory reuse, wherein w and h are width and height of the current block.

a. In one example, when CTU size is 128x128 and (x, y)=(m x 64,n x 64), the reference block cannot overlap with the 64x64 region starting from ((m-2)x64, n x 64).

b. In one example, when CTU size is 128x128, the reference block cannot overlap with the w x h block with upper-left corner being (x-128, y).

c. In one example, when CTU size is 128x128, (x+BVx, y+BVy) cannot be within the w*h block with upper-left corner being (x-128, y), where BVx and BVy denote the block vector for the current block.

d. In one example, when CTU size is M x M and IBC buffer size is k x M x M, reference block cannot overlap with the w x h block with upper-left corner being (x-k x M, y), where BVx and BVy denote the block vector for the current block.

e. In one example, when CTU size is M x M and IBC buffer size is k x M x M, (x+BVx, y+BVy) cannot be within the w x h block with upper-left corner being (x-k x M, y), where BVx and BVy denote the block vector for the current block.

63. When CTU size is not M x M and reference area size is nM x nM, the reference area is the nearest available nxn-1 CTU in a CTU row.

a. In one example, when reference area size is 128x128 and CTU size is 64x64, the nearest available 3 CTUs in a CTU row can be used for IBC reference.

b. In one example, when reference area size is 128x128 and CTU size is 32x32, the nearest available 15 CTUs in a CTU row can be used for IBC reference.

64. For a CU within a 64x64 block starting from (2m*64, 2n*64), i.e., a upper-left 64x64 block in a 128x128 CTU, its IBC prediction can be from reconstructed samples in the 64x64 block starting from ((2m-2)*64, 2n*64), the 64x64 block starting from ((2m-1)*64, 2n*64), the 64x64 block starting from ((2m-1)*64, (2n+1)*64) and the current 64x64 block.

65. For a CU within a 64x64 block starting from ((2m+1)*64, (2n+1)*64), i.e., a bottom-right 64x64 block in a 128x128 CTU, its IBC prediction can be from the current 128x128 CTU.

66. For a CU within a 64x64 block starting from ((2m+1)*64, 2n*64), i.e., a upper-right 64x64 block in a 128x128 CTU, its IBC prediction can be from reconstructed samples in the 64x64 block starting from ((2m-1)*64, 2n*64), the 64x64 block starting from ((2m-1)*64, (2n+1)*64), the 64x64 block starting from (2m*64, 2n*64) and the current 64x64 block.

a. Alternatively, if the 64x64 block starting from (2m*64, (2n+1)*64) has been reconstructed, the IBC prediction can be from reconstructed samples in the 64x64 block starting from ((2m-1)*64, 2n*64), the 64x64 block starting from (2m*64, 2n*64), the 64x64 block starting from (2m*64, (2n+1)*64) and the current 64x64 block.

67. For a CU within a 64x64 block starting from (2m*64, (2n+1)*64), i.e., a bottom-left 64x64 block in a 128x128 CTU, its IBC prediction can be from reconstructed samples in the 64x64 block starting from ((2m-1)*64, (2n+1)*64), the 64x64 block starting from (2m*64, 2n*64); the 64x64 block starting from ((2m+1)*64, 2n*64) and the current 64x64 block.

a. Alternatively, if the 64x64 block starting from ((2m+1)*64, 2n*64) has not been reconstructed, the IBC prediction can be from reconstructed samples in the 64x64 block starting from ((2m-1)*64, 2n*64), the 64x64 block starting from ((2m-1)*64, (2n+1)*64), the 64x64 block starting from (2m*64, 2n*64) and the current 64x64 block.

68. It is proposed to adjust the reference area based on which 64x64 blocks the current CU belongs to.

a. In one example, for a CU starting from (x,y), when (y>>6)&1 == 0, two or up to two previous 64x64 blocks, starting from ((x>>6<<6)-128, y>>6<<6) and ((x>>6<<6)-64, y>>6<<6) can be referenced by IBC mode.

b. In one example, for a CU starting from (x,y), when (y>>6)&1 == 1, one previous 64x64 block, starting from ((x>>6<<6)-64, y>>6<<6) can be referenced by IBC mode.

69. For a block starting from (x,y) and with block vector (BVx, BVy), if isRec(((x+BVx)>>6<<6)+128-(((y+BVy)>>6)&1)*64+(x%64), ((y+BVy)>>6<<6) +(y%64)) is true, the block vector is invalid.
    a. In one example, the block is a luma block.
    b. In one example, the block is a chroma block in 4:4:4 format
    c. In one example, the block contains both luma and chroma components
70. For a chroma block in 4:2:0 format starting from (x,y) and with block vector (BVx, BVy), if isRec(((x+BVx)>>5<<5)+64-(((y+BVy)>>5)&1)*32+(x%32), ((y+BVy)>>5<<5) +(y%32)) is true, the block vector is invalid.
71. The determination of whether a BV is invalid or not for a block of component c may rely on the availability of samples of component X, instead of checking the luma sample only.
    a. For a block of component c starting from (x,y) and with block vector (BVx, BVy), if isRec(c, ((x+BVx)>>6<<6)+128-(((y+BVy)>>6)&1)*64+(x%64), ((y+BVy)>>6<<6) +(y%64)) is true, the block vector may be treated as invalid.
        i. In one example, the block is a luma block (e.g., c is the luma component, or G component for RGB coding).
        ii. In one example, the block is a chroma block in 4:4:4 format (e.g., c is the cb or cr component, or B/R component for RGB coding).
        iii. In one example, availability of samples for both luma and chroma components may be checked, e.g., the block contains both luma and chroma components
    b. For a chroma block in 4:2:0 format starting from (x,y) of component c and with block vector (BVx, BVy), if isRec(c, ((x+BVx)>>5<<5)+64-(((y+BVy)>>5)&1)*32+(x%32), ((y+BVy)>>5<<5) +(y%32)) is true, the block vector may be treated as invalid.
    c. For a chroma block or sub-block starting from (x, y) of component c and with block vector (BVx, BVy), if isRec(c, x+BVx+Chroma_CTU_size, y) for a chroma component is true, the block vector may be treated as invalid, where Chroma_CTU_size is the CTU size for chroma component.
        i. In one example, for 4:2:0 format, Chroma_CTU_size may be 64.
        ii. In one example, a chroma sub-block may be a 2x2 block in 4:2:0 format.
        iii. In one example, a chroma sub-block may be a 4x4 block in 4:4:4 format.

iv. In one example, a chroma sub-block may correspond to the minimal CU size in luma component.

1. Alternatively, a chroma sub-block may correspond to the minimal CU size for the chroma component.

72. For all bullets mentioned above, it is assumed that the reference buffer contains multiple MxM blocks (M=64). However, it could be extended to other cases such as the reference buffer contains multiple NxM blocks (e.g., N=128, M=64).

73. For all bullets mentioned above, further restrictions may be applied that the reference buffer should be within the same brick/tile/tile group/slice as the current block.

a. In one example, if partial of the reference buffer is outside the current brick/tile/tile group/slice, the usage of IBC may be disabled. The signalling of IBC related syntax elements may be skipped.

b. Alternatively, if partial of the reference buffer is outside the current brick/tile/tile group/slice, IBC may be still enabled for one block, however, the block vector associated with one block may only point to the remaining reference buffer.

74. It is proposed to have K1 most recently coded VPDU, if available, in the 1st VPDU row of the CTU/CTB row and K2 most recently coded VPDU, if available, in the 2nd VPDU row of the CTU/CTB row as the reference area for IBC, excluding the current VPDU.

a. In one example, K1 is equal to 2 and K2 is equal to 1.

b. In one example, the above methods may be applied when the CTU/CTB size is 128x128 and VPDU size is 64x64.

c. In one example, the above methods may be applied when the CTU/CTB size is 64x64 and VPDU size is 64x64 and/or 32x32.

d. In one example, the above methods may be applied when the CTU/CTB size is 32x32 and VPDU size is 32x32 or smaller.

75. The above methods may be applied in different stages.

a. In one example, the module operation (e.g., a mod b) of block vectors (BVs) may be invoked in the availability check process of BVs to decide whether the BV is valid or not.

b. In one example, the module operation (e.g., a mod b) of block vectors (BVs) may be invoked to identify a reference sample's location (e.g., according to the module results

of a current sample's location and BV) in the IBC virtual buffer or reconstructed picture buffer (e.g., before in-loop filtering process).

5. Embodiments

5.1 Embodiment #1

[0095] An implementation of the buffer for IBC is described below:

[0096] The buffer size is 128x128. CTU size is also 128x128. For coding of the 1st CTU in a CTU row, the buffer is initialized with 128 (for 8-bit video signal). For coding of the k-th CTU in a CTU row, the buffer is initialized with the reconstruction before loop-filtering of the (k-1)-th CTU.

[0097] FIG. 3 shows an example of coding of a block starting from (x,y).

[0098] When coding a block starting from (x,y) related to the current CTU, a block vector (BVx, BVy) = (x-x0, y-y0) is sent to the decoder to indicate the reference block is from (x0,y0) in the IBC buffer. Suppose the width and height of the block are w and h respectively. When finishing coding of the block, a wxh area starting from (x,y) in the IBC buffer will be updated with the block's reconstruction before loop-filtering.

5.2 Embodiment #2

[0099] FIG. 4 shows examples of possible alternative way to choose the previous coded 64×64 blocks.

5.3 Embodiment #3

[0100] FIG. 5 shows an example of a possible alternative way to change the coding/decoding order of 64×64 blocks.

5.4 Embodiment #4

[0101] FIG. 8 shows another possible alternative way to choose the previous coded 64×64 blocks, when the decoding order for 64x64 blocks is from top to bottom, left to right.

5.5 Embodiment #5

[0102] FIG. 9 shows another possible alternative way to choose the previous coded 64×64 blocks.

5.6 Embodiment #6

[0103] FIG. 11 shows another possible alternative way to choose the previous coded 64×64 blocks, when the decoding order for 64x64 blocks is from left to right, top to bottom.

5.7 Embodiment #7

[0104] Suppose that CTU size is WxW, an implementation of IBC buffer with size mWxW and bitdepth being B, at the decoder is as below.

[0105] At the beginning of decoding a CTU row, initialize the buffer with value (1<<(B-1)) and set the starting point to update (xb, yb) to be (0,0).

[0106] When a CU starting from (x, y) related to a CTU upper-left corner and with size wxh is decoded, the area starting from (xb+x, yb+y) and wxh size will be updated with the reconstructed pixel values of the CU, after bit-depth aligned to B-bit.

[0107] After a CTU is decoded, the starting point to update (xb, yb) will be set as ((xb+W) mod mW, 0).

[0108] When decoding an IBC CU with block vector (BVx, BVy), for any pixel (x, y) related to a CTU upper-left corner, its prediction is extracted from the buffer at position ((x+BVx) mod mW, (y+BVy) mode W) after bit-depth alignment to the bit-depth of prediction signals.

[0109] In one example, B is set to 7, or 8 while the output/input bitdepth of the block may be equal to 10.

5.8 Embodiment #8

[0110] For a luma CU or joint luma/chroma CU starting from (x,y) related to the upper-left corner of a picture and a block vector (BVx, BVy), the block vector is invalid when isRec(((x+BVx)>>6<<6)+128-(((y+BVy)>>6)&1)*64+(x%64), ((y+BVy)>>6<<6) +(y%64)) is true.

[0111] For a chroma CU starting from (x,y) related to the upper-left corner of a picture and a block vector (BVx, BVy), the block vector is invalid when isRec(((x+BVx)>>5<<5)+64-(((y+BVy)>>5)&1)*32+(x%32), ((y+BVy)>>5<<5) +(y%32)) is true.

5.9 Embodiment #9

[0112] For a chroma block or sub-block starting from (x,y) in 4:2:0 format related to the upper-left corner of a picture and a block vector (BVx, BVy), the block vector is invalid when isRec(c, (x+BVx+64, y+BVy) is true, where c is a chroma component.

[0113] For a chroma block or sub-block starting from (x,y) in 4:4:4 format related to the upper-left corner of a picture and a block vector (BVx, BVy), the block vector is invalid when isRec(c, (x+BVx+128, y+BVy) is true, where c is a chroma component.

5.10 Embodiment #10

[0114] For a luma CU or joint luma/chroma CU starting from (x,y) related to the upper-left corner of a picture and a block vector (BVx, BVy), the block vector is invalid when isRec(((x+BVx)>>6<<6)+128-(((y+BVy)>>6)&1)*64+(x%64), ((y+BVy)>>6<<6) +(y%64)) is true.

[0115] For a chroma block or sub-block starting from (x,y) in 4:2:0 format related to the upper-left corner of a picture and a block vector (BVx, BVy), the block vector is invalid when isRec(c, ((x+BVx)>>5<<5)+64-(((y+BVy)>>5)&1)*32+(x%32), ((y+BVy)>>5<<5) +(y%32)) is true, where c is a chroma component.

5.11 Embodiment #11

[0116] This embodiment highlights an implementation of keeping two most coded VPDUs in the 1st VPDU row and one most coded VPDU in the 2nd VPDU row of a CTU/CTB row, excluding the current VPDU.

[0117] When VPDU coding order is top to bottom and left to right, the reference area is illustrated as in FIG. 13.

[0118] When VPDU coding order is left to right and top to bottom and the current VPDU is not to the right side of the picture boundary, the reference area is illustrated as in FIG. 14.

[0119] When VPDU coding order is left to right and top to bottom and the current VPDU is to the right side of the picture boundary, the reference area may be illustrated as FIG. 15.

[0120] Given a luma block (x, y) with size wxh, a block vector (BVx, BVy) is valid or not can be told by checking the following condition:

[0121] isRec(((x+BVx+128)>>6<<6) – (refy&0x40) + (x%64), ((y+BVy)>>6<<6) + (refy>>6 == y>>6)?(y%64):0), where refy = (y&0x40) ? (y+BVy) : (y+BVy+w-1).

[0122] If the above function returns true, the block vector (BVx, BVy) is invalid, otherwise the block vector might be valid.

5.12 Embodiment #12

[0123] If CTU size is 192x128, a virtual buffer with size 192x128 is maintained to track the reference samples for IBC.

[0124] A sample (x, y) relative to the upper-left corner of the picture is associated with the position (x%192, y%128) relative to the upper-left corner of the buffer. The following steps show how to mark availability of the samples associate with the virtual buffer for IBC reference.

[0125] A position (xPrevVPDU, yPrevVPDU) relative to the upper-left corner of the picture is recorded to stand for the upper-left sample of the most recently decoded VPDU.

1) At the beginning of decoding a VPDU row, all positions of the buffer are marked as unavailable. (xPrevVPDU, yPrevVPDU) is set as (0,0).

2) At the beginning of decoding the 1st CU of a VPDU, positions (x, y) with x = (xPrevVPDU - 2WVPDU+ 2mWVPDU)%(mWVPDU), .., ((xPrevVPDU - 2WVPDU+ 2mWVPDU)% (mWVPDU))-1+WVPDU; and y = yPrevVPDU%(nHVPDU), .., (yPrevVPDU%(nHVPDU))-1+HVPDU may be marked as unavailable. Then (xPrevVPDU, yPrevVPDU) is set as (xCU, yCU), i.e. the upper-left position of the CU relative to the picture.

3) After decoding a CU, positions (x, y) with x = xCU%(mWVPDU), ..., (xCU+CU_width-1)%(mWVPDU) and y = yCU%(nHVPDU),…,(yCU+CU_height-1)%(nHVPDU) are marked as available.

4) For an IBC CU with a block vector (xBV, yBV), if any position (x, y) with x = (xCU+xBV)%(mWVPDU), ..., (xCU+xBV+CU_width-1)%(mWVPDU) and y = (yCU+yBV)%(nHVPDU),…,(yCU+yBV+CU_height-1)%(nHVPDU) is marked as unavailable, the block vector is considered as invalid.

[0126] Figure 16 shows the buffer status along with the VPDU decoding status in the picture.

5.13 Embodiment #13

[0127] If CTU size is 128x128 or CTU size is greater than VPDU size (e.g., 64x64 in current design) or CTU size is greater than VPDU size (e.g., 64x64 in current design), a virtual buffer with size 192x128 is maintained to track the reference samples for IBC. In the following, when a < 0, (a % b) is defined as floor(a/b)*b, where floor (c) returns the largest integer no larger than c.

[0128] A sample (x, y) relative to the upper-left corner of the picture is associated with the position (x%192, y%128) relative to the upper-left corner of the buffer. The following steps show how to mark availability of the samples associate with the virtual buffer for IBC reference.

[0129] A position (xPrevVPDU, yPrevVPDU) relative to the upper-left corner of the picture is recorded to stand for the upper-left sample of the most recently decoded VPDU.

1) At the beginning of decoding a VPDU row, all positions of the buffer are marked as unavailable. (xPrevVPDU, yPrevVPDU) is set as (0,0).

2) At the beginning of decoding the 1st CU of a VPDU,

    a. If yPrevVPDU%64 is equal to 0, positions (x, y) with x = (xPrevVPDU – 128)%192, .., ((xPrevVPDU – 128)%192) + 63; and y = yPrevVPDU%128, .., (yPrevVPDU%128)+63,

are marked as unavailable. Then (xPrevVPDU, yPrevVPDU) is set as (xCU, yCU), i.e. the upper-left position of the CU relative to the picture.

b. Otherwise, positions (x, y) with x = (xPrevVPDU – 64)%192, .., ((xPrevVPDU – 64)%192) + 63; and y = yPrevVPDU%128, .., (yPrevVPDU%128)+63, are marked as unavailable. Then (xPrevVPDU, yPrevVPDU) is set as (xCU, yCU), i.e. the upper-left position of the CU relative to the picture.

3) After decoding a CU, positions (x, y) with x = xCU%192, ..., (xCU+CU_width-1)%192 and y = yCU%128,…,(yCU+CU_height-1)%128 are marked as available.

4) For an IBC CU with a block vector (xBV, yBV), if any position (x, y) with x = (xCU+xBV)%192, ..., (xCU+xBV+CU_width-1)%192 and y = (yCU+yBV)%128,…,(yCU+yBV+CU_height-1)%128 is marked as unavailable, the block vector is considered as invalid.

[0130] If CTU size is SxS, S is not equal to 128, let Wbuf be equal to 128*128/S. A virtual buffer with size WbufxS is maintained to track the reference samples for IBC. The VPDU size is equal to the CTU size in such a case.

[0131] A position (xPrevVPDU, yPrevVPDU) relative to the upper-left corner of the picture is recorded to stand for the upper-left sample of the most recently decoded VPDU.

1) At the beginning of decoding a VPDU row, all positions of the buffer are marked as unavailable. (xPrevVPDU, yPrevVPDU) is set as (0,0).

2) At the beginning of decoding the 1st CU of a VPDU, positions (x, y) with x = (xPrevVPDU – W_{buf}*S)%S, .., ((xPrevVPDU – W_{buf}*S)%S) + S - 1; and y = yPrevVPDU%S, .., (yPrevVPDU%S) + S -1; are marked as unavailable. Then (xPrevVPDU, yPrevVPDU) is set as (xCU, yCU), i.e. the upper-left position of the CU relative to the picture.

3) After decoding a CU, positions (x, y) with x = xCU%(W_{buf}), ..., (xCU+CU_width-1)%(W_{buf}) and y = yCU%S,…,(yCU+CU_height-1)%S are marked as available.

4) For an IBC CU with a block vector (xBV, yBV), if any position (x, y) with x = (xCU+xBV)%(Wbuf), ..., (xCU+xBV+CU_width-1)%(Wbuf) and y = (yCU+yBV)%S,…,(yCU+yBV+CU_height-1)%S is marked as unavailable, the block vector is considered as invalid.

5.14 Embodiment #14

[0132] If CTU size is 128x128 or CTU size is greater than VPDU size (e.g., 64x64 in current design) or CTU size is greater than VPDU size (e.g., 64x64 in current design), a virtual buffer with size

256x128 is maintained to track the reference samples for IBC. In the following, when a < 0, (a % b) is defined as floor(a/b)*b, where floor(c) returns the largest integer no larger than c.

[0133] A sample (x, y) relative to the upper-left corner of the picture is associated with the position (x%256, y%128) relative to the upper-left corner of the buffer. The following steps show how to mark availability of the samples associate with the virtual buffer for IBC reference.

[0134] A position (xPrevVPDU, yPrevVPDU) relative to the upper-left corner of the picture is recorded to stand for the upper-left sample of the most recently decoded VPDU.

1) At the beginning of decoding a VPDU row, all positions of the buffer are marked as unavailable. (xPrevVPDU, yPrevVPDU) is set as (0,0).
2) At the beginning of decoding the 1st CU of a VPDU,
    a. If yPrevVPDU%64 is equal to 0, positions (x, y) with x = (xPrevVPDU – 128)%256, .., ((xPrevVPDU – 128)% 256) + 63; and y = yPrevVPDU%128, .., (yPrevVPDU%128)+63, are marked as unavailable. Then (xPrevVPDU, yPrevVPDU) is set as (xCU, yCU), i.e. the upper-left position of the CU relative to the picture.
    b. Otherwise, positions (x, y) with x = (xPrevVPDU – 64)% 256, .., ((xPrevVPDU – 64)% 256) + 63; and y = yPrevVPDU%128, .., (yPrevVPDU%128)+63, are marked as unavailable. Then (xPrevVPDU, yPrevVPDU) is set as (xCU, yCU), i.e. the upper-left position of the CU relative to the picture.
3) After decoding a CU, positions (x, y) with x = xCU%256, ..., (xCU+CU_width-1)%256 and y = yCU%128,…,(yCU+CU_height-1)%128 are marked as available.
4) For an IBC CU with a block vector (xBV, yBV), if any position (x, y) with x = (xCU+xBV)%256, ..., (xCU+xBV+CU_width-1)%256 and y = (yCU+yBV)%128,…,(yCU+yBV+CU_height-1)%128 is marked as unavailable, the block vector is considered as invalid.

[0135] When CTU size is not 128x128 or less than 64x64 or less than 64x64, the same process applies as in the previous embodiment, i.e. embodiment #14.

5.15 Embodiment #15

[0136] An IBC reference availability marking process is described as follows. The changes are indicated in bolded, underlined, italicized text in this document.

7.3.7.1 General slice data syntax

slice_data( ) {	Descriptor
for(i = 0; i < NumBricksInCurrSlice; i++) {	
CtbAddrInBs = FirstCtbAddrBs[SliceBrickIdx[i]]	
for(j = 0; j < NumCtusInBrick[SliceBrickIdx[i]]; j++, CtbAddrInBs++) {	
if((j % BrickWidth[SliceBrickIdx[i]]) = = 0) {	
NumHmvpCand = 0	
NumHmvpIbcCand = 0	
xPrevVPDU = 0	
yPrevVPDU = 0	
if(CtbSizeY == 128)	
reset_ibc_isDecoded(0, 0, 256, CtbSizeY, BufWidth, BufHeight)	
else	
reset_ibc_isDecoded(0, 0, 128*128/CtbSizeY, CtbSizeY, BufWidth, BufHeight)	
}	
CtbAddrInRs = CtbAddrBsToRs[CtbAddrInBs]	
……	

reset_ibc_isDecoded(x0, y0, w, h, BufWidth, BufHeight) {	***Descriptor***
if(x0 >= 0)	
for (x = x0 % BufWidth; x < x0 + w; x+=4)	
for (y = y0 % BufHeight; y < y0 + h; y+=4)	
isDecoded[x >> 2][y >> 2] = 0	
}	

BufWidth is equal to (CtbSizeY==128)?256:(128*128/CtbSizeY) and BufHeight is equal to CtbSizeY.

7.3.7.5 Coding unit syntax

coding_unit(x0, y0, cbWidth, cbHeight, treeType) {	Descriptor
if(treeType != DUAL_TREE_CHROMA && (CtbSizeY = = 128) && (x0 % 64) = = 0 && (y0 % 64) = = 0) {	
for(x = x0; x < x0 + cbWidth; x += 64)	
for(y = y0; y < y0 + cbHeight; y += 64)	
if((yPrevVPDU % 64) = = 0)	
reset_ibc_isDecoded(xPrevVPDU – 128, yPrevVPDU, 64, 64, BufWidth, BufHeight)	
else	
reset_ibc_isDecoded(xPrevVPDU – 64, yPrevVPDU, 64, 64, BufWidth, BufHeight)	
xPrevVPDU = x0	

yPrevVPDU = y0	
}	
if(treeType != DUAL_TREE_CHROMA && (CtbSizeY < 128) && (x0 % CtbSizeY) = = 0 && (y0 % CtbSizeY) = = 0) {	
reset_ibc_isDecoded(xPrevVPDU – (128*128/CtbSizeY – CtbSizeY), yPrevVPDU, 64, 64, BufWidth, BufHeight)	
xPrevVPDU = x0	
yPrevVPDU = y0	
}	
if(slice_type != I || sps_ibc_enabled_flag) {	
if(treeType != DUAL_TREE_CHROMA && !(cbWidth = = 4 && cbHeight = = 4 && !sps_ibc_enabled_flag))	
cu_skip_flag[x0][y0]	ae(v)
if(cu_skip_flag[x0][y0] = = 0 && slice_type != I && !(cbWidth = = 4 && cbHeight = = 4))	
pred_mode_flag	ae(v)
if(((slice_type = = I && cu_skip_flag[x0][y0] = =0) || (slice_type != I && (CuPredMode[x0][y0] != MODE_INTRA || (cbWidth = = 4 && cbHeight = = 4 && cu_skip_flag[x0][y0] = = 0)))) && sps_ibc_enabled_flag && (cbWidth != 128 && cbHeight != 128))	
pred_mode_ibc_flag	ae(v)
}	
…	

8.6.2 Derivation process for motion vector components for IBC blocks

8.6.2.1 General

…

It is a requirement of bitstream comformance that when the block vector validity checking process in clause 8.6.3.2 is invoked with the block vector mvL, isBVvalid shall be true.

…

8.6.3 Decoding process for ibc blocks

8.6.3.1 General

[0137] This process is invoked when decoding a coding unit coded in ibc prediction mode.

[0138] Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,
- a variable cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples,

- variables numSbX and numSbY specifying the number of luma coding subblocks in horizontal and vertical direction,
- the motion vectors mv[xSbIdx][ySbIdx] with xSbIdx = 0 .. numSbX − 1, and ySbIdx = 0 .. numSbY − 1,
- a variable cIdx specifying the colour component index of the current block.
- ***a (nIbcBufW)x(ctbSize) array ibcBuf***

…

[0139] For each coding subblock at subblock index (xSbIdx, ySbIdx) with xSbIdx = 0 .. numSbX − 1, and ySbIdx = 0 .. numSbY − 1, the following applies:

- The luma location (xSb, ySb) specifying the top-left sample of the current coding subblock relative to the top-left luma sample of the current picture is derived as follows:

(xSb, ySb) = (xCb + xSbIdx * sbWidth, yCb + ySbIdx * sbHeight) (8-913)

If cIdx is equal to 0, nIbcBufW is set to ibcBufferWidth, otherwise nIbcBufW is set to (ibcBufferWidth / SubWidthC). The folling applies:

predSamples[xSb + x][ySb + y] = ibcBuf [(xSb + x + (mv[xSb][ySb][0] >> 4)) % nIbcRefW][ySb + y + (mv[xSb][ySb][1] >> 4)]

with x = 0..sbWidth − 1 and y = 0..sbHeight – 1.

…

8.6.3.2 Block vector validity checking process

Inputs to this process are:

- ***a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,***
- ***a variable cbWidth specifying the width of the current coding block in luma samples,***
- ***a variable cbHeight specifying the height of the current coding block in luma samples,***
- ***variables numSbX and numSbY specifying the number of luma coding subblocks in horizontal and vertical direction,***
- ***the block vectors mv[xSbIdx][ySbIdx] with xSbIdx = 0 .. numSbX − 1, and ySbIdx = 0 .. numSbY − 1,***
- ***a variable cIdx specifying the colour component index of the current block.***
- ***a (nIbcBufW)x(ctbSize) array ibcBuf***

Outputs of this process is a flag isBVvalid to indicate if the block vector is valid or not.

The following applies

1. ***isBVvalid is set eugal to true.***
2. ***If ((yCb & (ctbSize – 1)) + mv[0][0][1] + cbHeight) > ctbSize, isBVvalid is set eugal to false.***

3. *Otherwise, for each subblock index xSbIdx, ySbIdx with xSbIdx = 0 .. numSbX − 1, and ySbIdx = 0 .. numSbY – 1, its position relative to the top-left luma sample of the ibcBuf is derived:*

*xTL = (xCb + xSbIdx * sbWidth + mv[xSbIdx][ySbIdx][0]) & (nIbcBufW – 1)*

*yTL = (yCb & (ctbSize – 1)) + ySbIdx * sbHeight + mv[xSbIdx][ySbIdx][1]*

*xBR = (xCb + xSbIdx * sbWidth + sbWidth – 1 + mv[xSbIdx][ySbIdx][0]) & (nIbcBufW – 1)*

*yBR = (yCb & (ctbSize – 1)) + ySbIdx * sbHeight + sbHeight – 1 + mv[xSbIdx][ySbIdx][1]*

if (isDecoded[xTL>>2][yTL>>2] == 0) or (isDecoded[xBR>>2][yTL>>2] == 0) or (isDecoded[xBR>>2][yBR>>2] == 0), isBVvalid is set equal to false.

8.7.5 Picture reconstruction process

8.7.5.1 General

[0140] Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- the variables nCurrSw and nCurrSh specifying the width and height, respectively, of the current block,
- a variable cIdx specifying the colour component of the current block,
- an (nCurrSw) x (nCurrSh) array predSamples specifying the predicted samples of the current block,
- an (nCurrSw) x (nCurrSh) array resSamples specifying the residual samples of the current block.

[0141] Output of this process are

- a reconstructed picture sample array recSamples.
- *an IBC reference array ibcBuf.*

...

Denote nIbcBufW as the width of ibcBuf, the following applies:

ibcBuf[(xCurr + i) & (nIbcBufW – 1)][(yCurr + j) & (ctbSize – 1)] = recSamples[xCurr + i][yCurr + j]

with i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.

5.16 Embodiment #16

[0142] This is identical to the previous embodiment except for the following changes

slice_data( ) {	**Descriptor**
for(i = 0; i < NumBricksInCurrSlice; i++) {	
CtbAddrInBs = FirstCtbAddrBs[SliceBrickIdx[i]]	

for(j = 0; j < NumCtusInBrick[SliceBrickIdx[i]]; j++, CtbAddrInBs++) {	
if((j % BrickWidth[SliceBrickIdx[i]]) == 0) {	
NumHmvpCand = 0	
NumHmvpIbcCand = 0	
xPrevVPDU = 0	
yPrevVPDU = 0	
if(CtbSizeY == 128)	
reset_ibc_isDecoded(0, 0, 192, CtbSizeY, BufWidth, BufHeight)	
else	
reset_ibc_isDecoded(0, 0, 128*128/CtbSizeY, CtbSizeY, BufWidth, BufHeight)	
}	
CtbAddrInRs = CtbAddrBsToRs[CtbAddrInBs]	
……	

reset_ibc_isDecoded(x0, y0, w, h, BufWidth, BufHeight) {	***Descriptor***
if(x0 >= 0)	
for (x = x0 % BufWidth; x < x0 + w; x+=4)	
for (y = y0 % BufHeight; y < y0 + h; y+=4)	
isDecoded[x >> 2][y >> 2] = 0	
}	

BufWidth is equal to (CtbSizeY==128)?192:(128*128/CtbSizeY) and BufHeight is equal to CtbSizeY.

5.17 Embodiment #17

[0143] The changes in some examples are indicated in bolded, underlined, text in this document.

7.3.7 Slice data syntax

7.3.7.1 General slice data syntax

slice_data( ) {	**Descriptor**
for(i = 0; i < NumBricksInCurrSlice; i++) {	
CtbAddrInBs = FirstCtbAddrBs[SliceBrickIdx[i]]	
for(j = 0; j < NumCtusInBrick[SliceBrickIdx[i]]; j++, CtbAddrInBs++) {	
if((j % BrickWidth[SliceBrickIdx[i]]) == 0) {	
NumHmvpCand = 0	
NumHmvpIbcCand = 0	
resetIbcBuf = 1	
}	
CtbAddrInRs = CtbAddrBsToRs[CtbAddrInBs]	
coding_tree_unit( )	

if(entropy_coding_sync_enabled_flag && ((j + 1) % BrickWidth[SliceBrickIdx[i]] == 0)) {	
end_of_subset_one_bit /* equal to 1 */	ae(v)
if(j < NumCtusInBrick[SliceBrickIdx[i]] − 1)	
byte_alignment( )	
}	
}	
if(!entropy_coding_sync_enabled_flag) {	
end_of_brick_one_bit /* equal to 1 */	ae(v)
if(i < NumBricksInCurrSlice − 1)	
byte_alignment( )	
}	
}	
}	

7.4.8.5 Coding unit semantics

[0144] When all the following conditions are true, the history-based motion vector predictor list for the shared merging candidate list region is updated by setting NumHmvpSmrIbcCand equal to NumHmvpIbcCand, and setting HmvpSmrIbcCandList[i] equal to HmvpIbcCandList[i] for i = 0..NumHmvpIbcCand − 1:

- IsInSmr[x0][y0] is equal to TRUE.
- SmrX[x0][y0] is equal to x0.
- SmrY[x0][y0] is equal to y0.

[0145] The following assignments are made for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

CbPosX[x][y] = x0 (7-135)

CbPosY[x][y] = y0 (7-136)

CbWidth[x][y] = cbWidth (7-137)

CbHeight[x][y] = cbHeight (7-138)

Set vSize as min(ctbSize, 64) and wIbcBuf as (128*128/ctbSize). The width and height of ibcBuf is wIbcBuf and ctbSize accordingly.

If refreshIbcBuf is equal to 1, the following applies

- ***ibcBuf[x % wIbcBuf][y % ctbSize] = − 1, for x = x0..x0 + wIbcBuf − 1 and y = y0..y0 + ctbSize – 1***
- ***resetIbcBuf = 0***

When (x0 % vSize) is equal to 0 and (y0 % vSize) is equal to 0, for x = x0..x0 + vSize − 1 and y = y0..y0 + vSize – 1, the following applies

ibcBuf[x % wIbcBuf][y % ctbSize] = − 1

8.6.2 Derivation process for motion vector components for IBC blocks

8.6.2.1 General

[0146] Inputs to this process are:

- a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples.

[0147] Outputs of this process are:

- the luma motion vector in 1/16 fractional-sample accuracy mvL.

[0148] The luma motion vector mvL is derived as follows:

- The derivation process for IBC luma motion vector prediction as specified in clause 8.6.2.2 is invoked with the luma location (xCb, yCb), the variables cbWidth and cbHeight inputs, and the output being the luma motion vector mvL.
- When general_merge_flag[xCb][yCb] is equal to 0, the following applies:
  1. The variable mvd is derived as follows:

     mvd[0] = MvdL0[xCb][yCb][0] (8-883)

     mvd[1] = MvdL0[xCb][yCb][1] (8-884)
  2. The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvL, rightShift set equal to MvShift + 2, and leftShift set equal to MvShift + 2 as inputs and the rounded mvL as output.
  3. The luma motion vector mvL is modified as follows:

     $u[0] = (mvL[0] + mvd[0] + 2^{18}) \% 2^{18}$ (8-885)

     $mvL[0] = (u[0] >= 2^{17}) ? (u[0] - 2^{18}) : u[0]$ (8-886)

     $u[1] = (mvL[1] + mvd[1] + 2^{18}) \% 2^{18}$ (8-887)

     $mvL[1] = (u[1] >= 2^{17}) ? (u[1] - 2^{18}) : u[1]$ (8-888)

     NOTE 1– The resulting values of mvL[0] and mvL[1] as specified above will always be in the range of -2^{17} to $2^{17} - 1$, inclusive.

[0149] The updating process for the history-based motion vector predictor list as specified in clause 8.6.2.6 is invoked with luma motion vector mvL.

It is a requirement of bitstream conformance that the luma block vector mvL shall obey the following constraints:

- ***((yCb + (mvL[1] >> 4)) % wIbcBuf) + cbHeight is less than or equal to ctbSize***
- ***For x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight – 1, ibcBuf[(x + (mvL[0]>>4)) % wIbcBuf][(y + (mvL[1]>>4)) % ctbSize] shall not be equal to −1.***

8.7.5 Picture reconstruction process

8.7.5.1 General

[0150] Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- the variables nCurrSw and nCurrSh specifying the width and height, respectively, of the current block,
- a variable cIdx specifying the colour component of the current block,
- an (nCurrSw) x (nCurrSh) array predSamples specifying the predicted samples of the current block,
- an (nCurrSw) x (nCurrSh) array resSamples specifying the residual samples of the current block.

[0151] Output of this process ***are*** a reconstructed picture sample array recSamples ***and an IBC buffer array ibcBuf***.

[0152] Depending on the value of the colour component cIdx, the following assignments are made:

- If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array S_L and the function clipCidx1 corresponds to $Clip1_Y$.
- Otherwise, if cIdx is equal to 1, tuCbfChroma is set equal to tu_cbf_cb[xCurr][yCurr], recSamples corresponds to the reconstructed chroma sample array S_{Cb} and the function clipCidx1 corresponds to $Clip1_C$.
- Otherwise (cIdx is equal to 2), tuCbfChroma is set equal to tu_cbf_cr[xCurr][yCurr], recSamples corresponds to the reconstructed chroma sample array S_{Cr} and the function clipCidx1 corresponds to $Clip1_C$.

[0153] Depending on the value of slice_lmcs_enabled_flag, the following applies:

- If slice_lmcs_enabled_flag is equal to 0, the (nCurrSw)x(nCurrSh) block of the reconstructed samples recSamples at location (xCurr, yCurr) is derived as follows for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

  recSamples[xCurr + i][yCurr + j] = clipCidx1(predSamples[i][j] + resSamples[i][j]) (8-992)

- Otherwise (slice_lmcs_enabled_flag is equal to 1), the following applies:
  - If cIdx is equal to 0, the following applies:
    - The picture reconstruction with mapping process for luma samples as specified in clause 8.7.5.2 is invoked with the luma location (xCurr, yCurr), the block width nCurrSw and height nCurrSh, the predicted luma sample array predSamples, and the residual luma sample array resSamples as inputs, and the output is the reconstructed luma sample array recSamples.

– Otherwise (cIdx is greater than 0), the picture reconstruction with luma dependent chroma residual scaling process for chroma samples as specified in clause 8.7.5.3 is invoked with the chroma location (xCurr, yCurr), the transform block width nCurrSw and height nCurrSh, the coded block flag of the current chroma transform block tuCbfChroma, the predicted chroma sample array predSamples, and the residual chroma sample array resSamples as inputs, and the output is the reconstructed chroma sample array recSamples.

After decoding the current coding unit, the following applies:

ibcBuf[(xCurr + i) % wIbcBuf][(yCurr + j) % ctbSize] = recSamples[xCurr + i][yCurr + j] for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.

5.18 Embodiment #18

[0154] The changes in some examples are indicated in bolded, underlined, italicized text in this document.

7.3.7 Slice data syntax

7.3.7.1 General slice data syntax

slice_data( ) {	**Descriptor**
for(i = 0; i < NumBricksInCurrSlice; i++) {	
CtbAddrInBs = FirstCtbAddrBs[SliceBrickIdx[i]]	
for(j = 0; j < NumCtusInBrick[SliceBrickIdx[i]]; j++, CtbAddrInBs++) {	
if((j % BrickWidth[SliceBrickIdx[i]]) == 0) {	
NumHmvpCand = 0	
NumHmvpIbcCand = 0	
resetIbcBuf = 1	
}	
CtbAddrInRs = CtbAddrBsToRs[CtbAddrInBs]	
coding_tree_unit( )	
if(entropy_coding_sync_enabled_flag && ((j + 1) % BrickWidth[SliceBrickIdx[i]] == 0)) {	
end_of_subset_one_bit /* equal to 1 */	ae(v)
if(j < NumCtusInBrick[SliceBrickIdx[i]] − 1)	
byte_alignment( )	
}	
}	
if(!entropy_coding_sync_enabled_flag) {	
end_of_brick_one_bit /* equal to 1 */	ae(v)
if(i < NumBricksInCurrSlice − 1)	
byte_alignment( )	
}	

}	
}	

7.4.8.5 Coding unit semantics

[0155] When all the following conditions are true, the history-based motion vector predictor list for the shared merging candidate list region is updated by setting NumHmvpSmrIbcCand equal to NumHmvpIbcCand, and setting HmvpSmrIbcCandList[i] equal to HmvpIbcCandList[i] for i = 0..NumHmvpIbcCand − 1:

- IsInSmr[x0][y0] is equal to TRUE.
- SmrX[x0][y0] is equal to x0.
- SmrY[x0][y0] is equal to y0.

[0156] The following assignments are made for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

CbPosX[x][y] = x0 (7-135)

CbPosY[x][y] = y0 (7-136)

CbWidth[x][y] = cbWidth (7-137)

CbHeight[x][y] = cbHeight (7-138)

Set vSize as min(ctbSize, 64) and wIbcBufY as (128*128/CtbSizeY).

$ibcBuf_L$ is a array with width being wIbcBufY and height being CtbSizeY.

$ibcBuf_{Cb}$ and $ibcBuf_{Cr}$ are arrays with width being wIbcBufC =(wIbcBufY/SubWidthC) and height being (CtbSizeY/SubHeightC), i.e. CtbSizeC.

If resetIbcBuf is equal to 1, the following applies

- ***$ibcBuf_L$[x % wIbcBufY][y % CtbSizeY] = − 1, for x = x0..x0 + wIbcBufY − 1 and y = y0..y0 + CtbSizeY – 1***
- ***$ibcBuf_{Cb}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0..x0 + wIbcBufC − 1 and y = y0..y0 + CtbSizeC – 1***
- ***$ibcBuf_{Cr}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0..x0 + wIbcBufC − 1 and y = y0..y0 + CtbSizeC – 1***
- ***resetIbcBuf = 0***

When (x0 % vSizeY) is equal to 0 and (y0 % vSizeY) is equal to 0, the following applies

- ***$ibcBuf_L$[x % wIbcBufY][y % CtbSizeY] = − 1, for x = x0..x0 + vSize − 1 and y = y0..y0 + vSize – 1***
- ***$ibcBuf_{Cb}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0/SubWidthC..x0/SubWidthC + vSize/ SubWidthC − 1 and y = y0/SubHeightC..y0/SubHeightC + vSize/SubHeightC – 1***

– *ibcBuf$_{Cr}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0/SubWidthC..x0/SubWidthC + vSize/ SubWidthC − 1 and y = y0/SubHeightC..y0/SubHeightC + vSize/SubHeightC – 1*

8.6.2 Derivation process for motion vector components for IBC blocks

8.6.2.1 General

[0157] Inputs to this process are:

– a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
– a variable cbWidth specifying the width of the current coding block in luma samples,
– a variable cbHeight specifying the height of the current coding block in luma samples.

[0158] Outputs of this process are:

– the luma motion vector in 1/16 fractional-sample accuracy mvL.

[0159] The luma motion vector mvL is derived as follows:

– The derivation process for IBC luma motion vector prediction as specified in clause 8.6.2.2 is invoked with the luma location (xCb, yCb), the variables cbWidth and cbHeight inputs, and the output being the luma motion vector mvL.
– When general_merge_flag[xCb][yCb] is equal to 0, the following applies:
  1. The variable mvd is derived as follows:

$$mvd[0] = MvdL0[xCb][yCb][0] \quad (8\text{-}883)$$

$$mvd[1] = MvdL0[xCb][yCb][1] \quad (8\text{-}884)$$

  2. The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvL, rightShift set equal to MvShift + 2, and leftShift set equal to MvShift + 2 as inputs and the rounded mvL as output.
  3. The luma motion vector mvL is modified as follows:

$$u[0] = (mvL[0] + mvd[0] + 2^{18}) \% 2^{18} \quad (8\text{-}885)$$

$$mvL[0] = (u[0] >= 2^{17}) ? (u[0] - 2^{18}) : u[0] \quad (8\text{-}886)$$

$$u[1] = (mvL[1] + mvd[1] + 2^{18}) \% 2^{18} \quad (8\text{-}887)$$

$$mvL[1] = (u[1] >= 2^{17}) ? (u[1] - 2^{18}) : u[1] \quad (8\text{-}888)$$

NOTE 1– The resulting values of mvL[0] and mvL[1] as specified above will always be in the range of -2^{17} to $2^{17} - 1$, inclusive.

[0160] The updating process for the history-based motion vector predictor list as specified in clause 8.6.2.6 is invoked with luma motion vector mvL.

Clause 8.6.2.5 is invoked with mvL as input and mvC as output.

It is a requirement of bitstream conformance that the luma block vector mvL shall obey the following constraints:

- *((yCb + (mvL[1] >> 4)) % CtbSizeY) + cbHeight is less than or equal to CtbSizeY*
- *For x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight – 1, $ibcBuf_L$[(x + (mvL[0]>>4)) % wIbcBufY][(y +(mvL[1]>>4)) % CtbSizeY] shall not be equal to −1.*
- *If treeType is equal to SINGLE_TREE, for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight – 1, $ibcBuf_{Cb}$[(x + (mvC[0]>>5)) % wIbcBufC][(y +(mvC[1]>>5)) % CtbSizeC]] shall not be equal to −1.*

8.6.3 Decoding process for ibc blocks

8.6.3.1 General

[0161] This process is invoked when decoding a coding unit coded in ibc prediction mode.

[0162] Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,
- a variable cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples,
- colour component index of the current block.
- *the motion vector mv,*
- *an (wIbcBufY)x(CtbSizeY) array $ibcBuf_L$, an (wIbcBufC)x(CtbSizeC) array $ibcBuf_{Cb}$, an (wIbcBufC)x(CtbSizeC) array $ibcBuf_{Cr}$.*

[0163] Outputs of this process are:

- an array predSamples of prediction samples.

For x = xCb.. xCb+ Width − 1 and y = yCb..yCb + Height − 1, the following applies

If cIdx is qual to 0

predSamples[x][y] = $ibcBuf_L$[(x + mv[0] >> 4)) % wIbcBufY][(y + (mv[1] >> 4)) % CtbSizeY]

if cIdx is equal to 1

predSamples[x][y] = $ibcBuf_{Cb}$[(x + mv[0] >> 5)) % wIbcBufC][(y + (mv[1] >> 5)) % CtbSizeC]

if cIdx is equal to 2

predSamples[x][y] = $ibcBuf_{Cr}$[(x + mv[0] >> 5)) % wIbcBufC][(y + (mv[1] >> 5)) % CtbSizeC]

8.7.5 Picture reconstruction process

8.7.5.1 General

[0164] Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- the variables nCurrSw and nCurrSh specifying the width and height, respectively, of the current block,
- a variable cIdx specifying the colour component of the current block,
- an (nCurrSw) x (nCurrSh) array predSamples specifying the predicted samples of the current block,
- an (nCurrSw) x (nCurrSh) array resSamples specifying the residual samples of the current block.

[0165] Output of this process ***are*** a reconstructed picture sample array recSamples ***and IBC buffer arrays $ibcBuf_L$, $ibcBuf_{Cb}$, $ibcBuf_{Cr}$***.

[0166] Depending on the value of the colour component cIdx, the following assignments are made:

- If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array S_L and the function clipCidx1 corresponds to $Clip1_Y$.
- Otherwise, if cIdx is equal to 1, tuCbfChroma is set equal to tu_cbf_cb[xCurr][yCurr], recSamples corresponds to the reconstructed chroma sample array S_{Cb} and the function clipCidx1 corresponds to $Clip1_C$.
- Otherwise (cIdx is equal to 2), tuCbfChroma is set equal to tu_cbf_cr[xCurr][yCurr], recSamples corresponds to the reconstructed chroma sample array S_{Cr} and the function clipCidx1 corresponds to $Clip1_C$.

[0167] Depending on the value of slice_lmcs_enabled_flag, the following applies:

- If slice_lmcs_enabled_flag is equal to 0, the (nCurrSw)x(nCurrSh) block of the reconstructed samples recSamples at location (xCurr, yCurr) is derived as follows for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

  recSamples[xCurr + i][yCurr + j] = clipCidx1(predSamples[i][j] + resSamples[i][j]) (8-992)

- Otherwise (slice_lmcs_enabled_flag is equal to 1), the following applies:
  - If cIdx is equal to 0, the following applies:
    - The picture reconstruction with mapping process for luma samples as specified in clause 8.7.5.2 is invoked with the luma location (xCurr, yCurr), the block width nCurrSw and height nCurrSh, the predicted luma sample array predSamples, and the residual luma sample array resSamples as inputs, and the output is the reconstructed luma sample array recSamples.

– Otherwise (cIdx is greater than 0), the picture reconstruction with luma dependent chroma residual scaling process for chroma samples as specified in clause 8.7.5.3 is invoked with the chroma location (xCurr, yCurr), the transform block width nCurrSw and height nCurrSh, the coded block flag of the current chroma transform block tuCbfChroma, the predicted chroma sample array predSamples, and the residual chroma sample array resSamples as inputs, and the output is the reconstructed chroma sample array recSamples.

After decoding the current coding unit, the following may apply:

If cIdx is equal to 0, and if treeType is equal to SINGLE_TREE or DUAL_TREE_LUMA, the following applies

$ibcBuf_Y[(xCurr + i) \% wIbcBufY][(yCurr + j) \% CtbSizeY] = recSamples[xCurr + i][yCurr + j]$ *for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.*

If cIdx is equal to 1, and if treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies

$ibcBuf_{Cb}[(xCurr + i) \% wIbcBufC][(yCurr + j) \% CtbSizeC] = recSamples[xCurr + i][yCurr + j]$ *for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.*

If cIdx is equal to 2, and if treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies

$ibcBuf_{Cr}[(xCurr + i) \% wIbcBufC][(yCurr + j) \% CtbSizeC] = recSamples[xCurr + i][yCurr + j]$ *for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.*

5.19 Embodiment #19

[0168] The changes in some examples are indicated in bolded, underlined, text in this document.

7.3.7 Slice data syntax

7.3.7.1 General slice data syntax

slice_data( ) {	**Descriptor**
for(i = 0; i < NumBricksInCurrSlice; i++) {	
CtbAddrInBs = FirstCtbAddrBs[SliceBrickIdx[i]]	
for(j = 0; j < NumCtusInBrick[SliceBrickIdx[i]]; j++, CtbAddrInBs++) {	
if((j % BrickWidth[SliceBrickIdx[i]]) == 0) {	
NumHmvpCand = 0	
NumHmvpIbcCand = 0	
resetIbcBuf = 1	
}	
CtbAddrInRs = CtbAddrBsToRs[CtbAddrInBs]	
coding_tree_unit( )	

if(entropy_coding_sync_enabled_flag && ((j + 1) % BrickWidth[SliceBrickIdx[i]] == 0)) {	
end_of_subset_one_bit /* equal to 1 */	ae(v)
if(j < NumCtusInBrick[SliceBrickIdx[i]] − 1)	
byte_alignment( )	
}	
}	
if(!entropy_coding_sync_enabled_flag) {	
end_of_brick_one_bit /* equal to 1 */	ae(v)
if(i < NumBricksInCurrSlice − 1)	
byte_alignment( )	
}	
}	
}	

7.4.8.5 Coding unit semantics

[0169] When all the following conditions are true, the history-based motion vector predictor list for the shared merging candidate list region is updated by setting NumHmvpSmrIbcCand equal to NumHmvpIbcCand, and setting HmvpSmrIbcCandList[i] equal to HmvpIbcCandList[i] for i = 0..NumHmvpIbcCand − 1:

- IsInSmr[x0][y0] is equal to TRUE.
- SmrX[x0][y0] is equal to x0.
- SmrY[x0][y0] is equal to y0.

[0170] The following assignments are made for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

CbPosX[x][y] = x0 (7-135)

CbPosY[x][y] = y0 (7-136)

CbWidth[x][y] = cbWidth (7-137)

CbHeight[x][y] = cbHeight (7-138)

Set vSize as min(ctbSize, 64) and wIbcBufY as (128*128/CtbSizeY).

$ibcBuf_L$ is a array with width being wIbcBufY and height being CtbSizeY.

$ibcBuf_{Cb}$ and $ibcBuf_{Cr}$ are arrays with width being wIbcBufC =(wIbcBufY/SubWidthC) and height being (CtbSizeY/SubHeightC), i.e. CtbSizeC.

If resetIbcBuf is equal to 1, the following applies

- ***$ibcBuf_L$[x % wIbcBufY][y % CtbSizeY] = − 1, for x = x0..x0 + wIbcBufY − 1 and y = y0..y0 + CtbSizeY – 1***

- *ibcBuf$_{Cb}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0..x0 + wIbcBufC − 1 and y = y0..y0 + CtbSizeC – 1*
- *ibcBuf$_{Cr}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0..x0 + wIbcBufC − 1 and y = y0..y0 + CtbSizeC – 1*
- *resetIbcBuf = 0*

When (x0 % vSizeY) is equal to 0 and (y0 % vSizeY) is equal to 0, the following applies

- *ibcBuf$_{L}$[x % wIbcBufY][y % CtbSizeY] = − 1, for x = x0..x0 + min(vSize, cbWidth) − 1 and y = y0..y0 + min(vSize, cbHeight)– 1*
- *ibcBuf$_{Cb}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0/SubWidthC..x0/SubWidthC + min(vSize/ SubWidthC, cbWidth) − 1 and y = y0/SubHeightC..y0/SubHeightC + min(vSize/SubHeightC, cbHeight) – 1*
- *ibcBuf$_{Cr}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0/SubWidthC..x0/SubWidthC + min(vSize/ SubWidthC, cbWidth) − 1 and y = y0/SubHeightC..y0/SubHeightC + min(vSize/SubHeightC, cbHeight) – 1*

8.6.2 Derivation process for motion vector components for IBC blocks

8.6.2.1 General

[0171] Inputs to this process are:

- a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples.

[0172] Outputs of this process are:

- the luma motion vector in 1/16 fractional-sample accuracy mvL.

[0173] The luma motion vector mvL is derived as follows:

- The derivation process for IBC luma motion vector prediction as specified in clause 8.6.2.2 is invoked with the luma location (xCb, yCb), the variables cbWidth and cbHeight inputs, and the output being the luma motion vector mvL.
- When general_merge_flag[xCb][yCb] is equal to 0, the following applies:
  1. The variable mvd is derived as follows:

     mvd[0] = MvdL0[xCb][yCb][0] (8-883)

     mvd[1] = MvdL0[xCb][yCb][1] (8-884)
  2. The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvL, rightShift set equal to MvShift + 2, and leftShift set equal to MvShift + 2 as inputs and the rounded mvL as output.

3. The luma motion vector mvL is modified as follows:

$u[0] = (mvL[0] + mvd[0] + 2^{18}) \% 2^{18}$ (8-885)

$mvL[0] = (u[0] >= 2^{17}) ? (u[0] - 2^{18}) : u[0]$ (8-886)

$u[1] = (mvL[1] + mvd[1] + 2^{18}) \% 2^{18}$ (8-887)

$mvL[1] = (u[1] >= 2^{17}) ? (u[1] - 2^{18}) : u[1]$ (8-888)

NOTE 1– The resulting values of mvL[0] and mvL[1] as specified above will always be in the range of -2^{17} to $2^{17} - 1$, inclusive.

[0174] The updating process for the history-based motion vector predictor list as specified in clause 8.6.2.6 is invoked with luma motion vector mvL.

Clause 8.6.2.5 is invoked with mvL as input and mvC as output.

It is a requirement of bitstream conformance that the luma block vector mvL shall obey the following constraints:

- ***((yCb + (mvL[1] >> 4)) % CtbSizeY) + cbHeight is less than or equal to CtbSizeY***
- ***For x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight – 1, $ibcBuf_L$[(x + (mvL[0]>>4)) % wIbcBufY][(y +(mvL[1]>>4)) % CtbSizeY] shall not be equal to −1.***
- ***If treeType is equal to SINGLE_TREE, for x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight – 1, $ibcBuf_{Cb}$[(x + (mvC[0]>>5)) % wIbcBufC][(y +(mvC[1]>>5)) % CtbSizeC]] shall not be equal to −1.***

8.6.3 Decoding process for ibc blocks

8.6.3.1 General

[0175] This process is invoked when decoding a coding unit coded in ibc prediction mode.

[0176] Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,
- a variable cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples,
- ***a variable cIdx specifying the colour component index of the current block.***
- ***the motion vector mv,***
- ***an (wIbcBufY)x(CtbSizeY) array $ibcBuf_L$, an (wIbcBufC)x(CtbSizeC) array $ibcBuf_{Cb}$, an (wIbcBufC)x(CtbSizeC) array $ibcBuf_{Cr}$.***

Outputs of this process are:

- ***an array predSamples of prediction samples.***

For x = xCb.. xCb+ Width − 1 and y = yCb..yCb + Height − 1, the following applies

If cIdx is qual to 0

predSamples[x][y] = ibcBuf$_L$[(x + mv[0] >> 4)) % wIbcBufY][(y + (mv[1] >> 4)) % CtbSizeY]

if cIdx is equal to 1

predSamples[x][y] = ibcBuf$_{Cb}$[(x + mv[0] >> 5)) % wIbcBufC][(y + (mv[1] >> 5)) % CtbSizeC]

if cIdx is equal to 2

predSamples[x][y] = ibcBuf$_{Cr}$[(x + mv[0] >> 5)) % wIbcBufC][(y + (mv[1] >> 5)) % CtbSizeC]

8.7.5 Picture reconstruction process

8.7.5.1 General

[0177] Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- the variables nCurrSw and nCurrSh specifying the width and height, respectively, of the current block,
- a variable cIdx specifying the colour component of the current block,
- an (nCurrSw) x (nCurrSh) array predSamples specifying the predicted samples of the current block,
- an (nCurrSw) x (nCurrSh) array resSamples specifying the residual samples of the current block.

[0178] Output of this process are a reconstructed picture sample array recSamples and IBC buffer arrays ibcBuf$_L$, ibcBuf$_{Cb}$, ibcBuf$_{Cr}$.

[0179] Depending on the value of the colour component cIdx, the following assignments are made:

- If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array S_L and the function clipCidx1 corresponds to Clip1$_Y$.
- Otherwise, if cIdx is equal to 1, tuCbfChroma is set equal to tu_cbf_cb[xCurr][yCurr], recSamples corresponds to the reconstructed chroma sample array S_{Cb} and the function clipCidx1 corresponds to Clip1$_C$.
- Otherwise (cIdx is equal to 2), tuCbfChroma is set equal to tu_cbf_cr[xCurr][yCurr], recSamples corresponds to the reconstructed chroma sample array S_{Cr} and the function clipCidx1 corresponds to Clip1$_C$.

[0180] Depending on the value of slice_lmcs_enabled_flag, the following applies:

- If slice_lmcs_enabled_flag is equal to 0, the (nCurrSw)x(nCurrSh) block of the reconstructed samples recSamples at location (xCurr, yCurr) is derived as follows for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

recSamples[xCurr + i][yCurr + j] = clipCidx1(predSamples[i][j] + resSamples[i][j]) (8-992)

- Otherwise (slice_lmcs_enabled_flag is equal to 1), the following applies:
  - If cIdx is equal to 0, the following applies:
    - The picture reconstruction with mapping process for luma samples as specified in clause 8.7.5.2 is invoked with the luma location (xCurr, yCurr), the block width nCurrSw and height nCurrSh, the predicted luma sample array predSamples, and the residual luma sample array resSamples as inputs, and the output is the reconstructed luma sample array recSamples.
  - Otherwise (cIdx is greater than 0), the picture reconstruction with luma dependent chroma residual scaling process for chroma samples as specified in clause 8.7.5.3 is invoked with the chroma location (xCurr, yCurr), the transform block width nCurrSw and height nCurrSh, the coded block flag of the current chroma transform block tuCbfChroma, the predicted chroma sample array predSamples, and the residual chroma sample array resSamples as inputs, and the output is the reconstructed chroma sample array recSamples.

After decoding the current coding unit, the following may apply:

If cIdx is equal to 0, and if treeType is equal to SINGLE_TREE or DUAL_TREE_LUMA, the following applies

$ibcBuf_L[(xCurr + i) \% wIbcBufY][(yCurr + j) \% CtbSizeY] = recSamples[xCurr + i][yCurr + j]$ *for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.*

If cIdx is equal to 1, and if treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies

$ibcBuf_{Cb}[(xCurr + i) \% wIbcBufC][(yCurr + j) \% CtbSizeC] = recSamples[xCurr + i][yCurr + j]$ *for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.*

If cIdx is equal to 2, and if treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies

$ibcBuf_{Cr}[(xCurr + i) \% wIbcBufC][(yCurr + j) \% CtbSizeC] = recSamples[xCurr + i][yCurr + j]$ *for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.*

5.20 Embodiment #20

[0181] The changes in some examples are indicated in bolded, underlined, italicized text in this document.

7.3.7 Slice data syntax

7.3.7.1 General slice data syntax

slice_data( ) {	Descriptor
for(i = 0; i < NumBricksInCurrSlice; i++) {	
CtbAddrInBs = FirstCtbAddrBs[SliceBrickIdx[i]]	
for(j = 0; j < NumCtusInBrick[SliceBrickIdx[i]]; j++, CtbAddrInBs++) {	
if((j % BrickWidth[SliceBrickIdx[i]]) == 0) {	
NumHmvpCand = 0	
NumHmvpIbcCand = 0	
resetIbcBuf = 1	
}	
CtbAddrInRs = CtbAddrBsToRs[CtbAddrInBs]	
coding_tree_unit( )	
if(entropy_coding_sync_enabled_flag && ((j + 1) % BrickWidth[SliceBrickIdx[i]] == 0)) {	
end_of_subset_one_bit /* equal to 1 */	ae(v)
if(j < NumCtusInBrick[SliceBrickIdx[i]] − 1)	
byte_alignment( )	
}	
}	
if(!entropy_coding_sync_enabled_flag) {	
end_of_brick_one_bit /* equal to 1 */	ae(v)
if(i < NumBricksInCurrSlice − 1)	
byte_alignment( )	
}	
}	
}	

7.4.8.5 Coding unit semantics

[0182] When all the following conditions are true, the history-based motion vector predictor list for the shared merging candidate list region is updated by setting NumHmvpSmrIbcCand equal to NumHmvpIbcCand, and setting HmvpSmrIbcCandList[i] equal to HmvpIbcCandList[i] for i = 0..NumHmvpIbcCand − 1:

- IsInSmr[x0][y0] is equal to TRUE.
- SmrX[x0][y0] is equal to x0.
- SmrY[x0][y0] is equal to y0.

[0183] The following assignments are made for x = x0..x0 + cbWidth − 1 and y = y0..y0 + cbHeight − 1:

CbPosX[x][y] = x0 (7-135)

CbPosY[x][y] = y0 (7-136)

CbWidth[x][y] = cbWidth (7-137)

CbHeight[x][y] = cbHeight (7-138)

Set vSize as min(ctbSize, 64) and wIbcBufY as (128*128/CtbSizeY).

$ibcBuf_L$ is a array with width being wIbcBufY and height being CtbSizeY.

$ibcBuf_{Cb}$ and $ibcBuf_{Cr}$ are arrays with width being wIbcBufC =(wIbcBufY/SubWidthC) and height being (CtbSizeY/SubHeightC), i.e. CtbSizeC.

If resetIbcBuf is equal to 1, the following applies

- ***$ibcBuf_L$[x % wIbcBufY][y % CtbSizeY] = − 1, for x = x0..x0 + wIbcBufY − 1 and y = y0..y0 + CtbSizeY – 1***
- ***$ibcBuf_{Cb}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0..x0 + wIbcBufC − 1 and y = y0..y0 + CtbSizeC – 1***
- ***$ibcBuf_{Cr}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0..x0 + wIbcBufC − 1 and y = y0..y0 + CtbSizeC – 1***
- ***resetIbcBuf = 0***

When (x0 % vSizeY) is equal to 0 and (y0 % vSizeY) is equal to 0, the following applies

- ***$ibcBuf_L$[x % wIbcBufY][y % CtbSizeY] = − 1, for x = x0..x0 + max(vSize, cbWidth) − 1 and y = y0..y0 + max(vSize, cbHeight)– 1***
- ***$ibcBuf_{Cb}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0/SubWidthC..x0/SubWidthC + max(vSize/ SubWidthC, cbWidth) − 1 and y = y0/SubHeightC..y0/SubHeightC + max(vSize/SubHeightC, cbHeight) – 1***
- ***$ibcBuf_{Cr}$[x % wIbcBufC][y % CtbSizeC] = − 1, for x = x0/SubWidthC..x0/SubWidthC + max(vSize/ SubWidthC, cbWidth) − 1 and y = y0/SubHeightC..y0/SubHeightC + max(vSize/SubHeightC, cbHeight) – 1***

8.6.2 Derivation process for motion vector components for IBC blocks

8.6.2.1 General

[0184] Inputs to this process are:

- a luma location (xCb, yCb) of the top-left sample of the current luma coding block relative to the top-left luma sample of the current picture,
- a variable cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples.

[0185] Outputs of this process are:

- the luma motion vector in 1/16 fractional-sample accuracy mvL.

[0186] The luma motion vector mvL is derived as follows:

- The derivation process for IBC luma motion vector prediction as specified in clause 8.6.2.2 is invoked with the luma location (xCb, yCb), the variables cbWidth and cbHeight inputs, and the output being the luma motion vector mvL.
- When general_merge_flag[xCb][yCb] is equal to 0, the following applies:
  1. The variable mvd is derived as follows:

     mvd[0] = MvdL0[xCb][yCb][0] (8-883)

     mvd[1] = MvdL0[xCb][yCb][1] (8-884)
  2. The rounding process for motion vectors as specified in clause 8.5.2.14 is invoked with mvX set equal to mvL, rightShift set equal to MvShift + 2, and leftShift set equal to MvShift + 2 as inputs and the rounded mvL as output.
  3. The luma motion vector mvL is modified as follows:

     $u[0] = (mvL[0] + mvd[0] + 2^{18}) \% 2^{18}$ (8-885)

     $mvL[0] = (u[0] >= 2^{17}) ? (u[0] - 2^{18}) : u[0]$ (8-886)

     $u[1] = (mvL[1] + mvd[1] + 2^{18}) \% 2^{18}$ (8-887)

     $mvL[1] = (u[1] >= 2^{17}) ? (u[1] - 2^{18}) : u[1]$ (8-888)

     NOTE 1– The resulting values of mvL[0] and mvL[1] as specified above will always be in the range of -2^{17} to $2^{17} - 1$, inclusive.

[0187] The updating process for the history-based motion vector predictor list as specified in clause 8.6.2.6 is invoked with luma motion vector mvL.

Clause 8.6.2.5 is invoked with mvL as input and mvC as output.

It is a requirement of bitstream conformance that the luma block vector mvL shall obey the following constraints:

- ***((yCb + (mvL[1] >> 4)) % CtbSizeY) + cbHeight is less than or equal to CtbSizeY***
- ***For x = xCb..xCb + cbWidth − 1 and y = yCb..yCb + cbHeight – 1, $ibcBuf_L$[(x + (mvL[0]>>4)) % wIbcBufY][(y +(mvL[1]>>4)) % CtbSizeY] shall not be equal to −1.***

8.6.3 Decoding process for ibc blocks

8.6.3.1 General

[0188] This process is invoked when decoding a coding unit coded in ibc prediction mode.

[0189] Inputs to this process are:

- a luma location (xCb, yCb) specifying the top-left sample of the current coding block relative to the top-left luma sample of the current picture,
- a variable cbWidth specifying the width of the current coding block in luma samples,
- a variable cbHeight specifying the height of the current coding block in luma samples,

- a variable cIdx specifying the colour component index of the current block.
- ***the motion vector mv,***
- ***an (wIbcBufY)x(CtbSizeY) array $ibcBuf_L$, an (wIbcBufC)x(CtbSizeC) array $ibcBuf_{Cb}$, an (wIbcBufC)x(CtbSizeC) array $ibcBuf_{Cr}$.***

Outputs of this process are:

- ***an array predSamples of prediction samples.***

For x = xCb.. xCb+ Width − 1 and y = yCb..yCb + Height − 1, the following applies

If cIdx is qual to 0

predSamples[x][y] = $ibcBuf_L$[(x + mv[0] >> 4)) % wIbcBufY][(y + (mv[1] >> 4)) % CtbSizeY]

if cIdx is equal to 1

predSamples[x][y] = $ibcBuf_{Cb}$[(x + mv[0] >> 5)) % wIbcBufC][(y + (mv[1] >> 5)) % CtbSizeC]

if cIdx is equal to 2

predSamples[x][y] = $ibcBuf_{Cr}$[(x + mv[0] >> 5)) % wIbcBufC][(y + (mv[1] >> 5)) % CtbSizeC]

8.7.5 Picture reconstruction process

8.7.5.1 General

[0190] Inputs to this process are:

- a location (xCurr, yCurr) specifying the top-left sample of the current block relative to the top-left sample of the current picture component,
- the variables nCurrSw and nCurrSh specifying the width and height, respectively, of the current block,
- a variable cIdx specifying the colour component of the current block,
- an (nCurrSw) x (nCurrSh) array predSamples specifying the predicted samples of the current block,
- an (nCurrSw) x (nCurrSh) array resSamples specifying the residual samples of the current block.

[0191] Output of this process are a reconstructed picture sample array recSamples and IBC buffer arrays $ibcBuf_L$, $ibcBuf_{Cb}$, $ibcBuf_{Cr}$.

[0192] Depending on the value of the colour component cIdx, the following assignments are made:

- If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array S_L and the function clipCidx1 corresponds to $Clip1_Y$.
- Otherwise, if cIdx is equal to 1, tuCbfChroma is set equal to tu_cbf_cb[xCurr][yCurr], recSamples corresponds to the reconstructed chroma sample array S_{Cb} and the function clipCidx1 corresponds to $Clip1_C$.

– Otherwise (cIdx is equal to 2), tuCbfChroma is set equal to tu_cbf_cr[xCurr][yCurr], recSamples corresponds to the reconstructed chroma sample array S_{Cr} and the function clipCidx1 corresponds to $Clip1_C$.

[0193] Depending on the value of slice_lmcs_enabled_flag, the following applies:

– If slice_lmcs_enabled_flag is equal to 0, the (nCurrSw)x(nCurrSh) block of the reconstructed samples recSamples at location (xCurr, yCurr) is derived as follows for i = 0..nCurrSw − 1, j = 0..nCurrSh − 1:

recSamples[xCurr + i][yCurr + j] = clipCidx1(predSamples[i][j] + resSamples[i][j]) (8-992)

– Otherwise (slice_lmcs_enabled_flag is equal to 1), the following applies:
  – If cIdx is equal to 0, the following applies:
    – The picture reconstruction with mapping process for luma samples as specified in clause 8.7.5.2 is invoked with the luma location (xCurr, yCurr), the block width nCurrSw and height nCurrSh, the predicted luma sample array predSamples, and the residual luma sample array resSamples as inputs, and the output is the reconstructed luma sample array recSamples.
  – Otherwise (cIdx is greater than 0), the picture reconstruction with luma dependent chroma residual scaling process for chroma samples as specified in clause 8.7.5.3 is invoked with the chroma location (xCurr, yCurr), the transform block width nCurrSw and height nCurrSh, the coded block flag of the current chroma transform block tuCbfChroma, the predicted chroma sample array predSamples, and the residual chroma sample array resSamples as inputs, and the output is the reconstructed chroma sample array recSamples.

After decoding the current coding unit, the following may apply:

If cIdx is equal to 0, and if treeType is equal to SINGLE_TREE or DUAL_TREE_LUMA, the following applies

$ibcBuf_Y[(xCurr + i) \% wIbcBufY][(yCurr + j) \% CtbSizeY] = recSamples[xCurr + i][yCurr + j]$ ***for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.***

If cIdx is equal to 1, and if treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies

$ibcBuf_{Cb}[(xCurr + i) \% wIbcBufC][(yCurr + j) \% CtbSizeC] = recSamples[xCurr + i][yCurr + j]$ ***for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.***

If cIdx is equal to 2, and if treeType is equal to SINGLE_TREE or DUAL_TREE_CHROMA, the following applies

$ibcBuf_{Cr}[(xCurr + i) \% wIbcBufC][(yCurr + j) \% CtbSizeC] = recSamples[xCurr + i][yCurr + j]$ ***for i = 0..nCurrSw − 1, j = 0..nCurrSh – 1.***

[0194] FIG. 6 is a flowchart of an example method 600 of visual media (video or image) processing. The method 600 includes determining (602), for a conversion between a current video block and a bitstream representation of the current video block, a size of a buffer to store reference samples for the current video block using an intra-block copy coding mode, and performing (604) the conversion using the reference samples stored in the buffer.

[0195] The following clauses describe some example preferred features implemented by embodiments of method 600 and other methods. Additional examples are provided in Section 4 of the present document.

[0196] 1. A method of video processing, comprising: determining, for a conversion between a current video block and a bitstream representation of the current video block, a size of a buffer to store reference samples for the current video block using an intra-block copy coding mode; and performing the conversion using the reference samples stored in the buffer.

[0197] 2. The method of clause 1, wherein the size of the buffer is a predetermined constant.

[0198] 3. The method of any of clauses 1-2, wherein the size is MxN, where M and N are integers.

[0199] 4. The method of clause 3, wherein MxN is equal to 64x64 or 128x128 or 64x128.

[0200] 5. The method of clause 1, wherein the size of the buffer is equal to a size of a coding tree unit of the current video block.

[0201] 6. The method of clause 1, wherein the size of the buffer is equal to a size of a virtual pipeline data unit used during the conversion.

[0202] 7. The method of clause 1, wherein the size of the buffer corresponds a field in the bitstream representation.

[0203] 8. The method of clause 7, wherein the field is included in the bitstream representation at a video parameter set or sequence parameter set or picture parameter set or a picture header or a slice header or a tile group header level.

[0204] 9. The method of any of clauses 1-8, wherein the size of the buffer is different for reference samples for luma component and reference samples for chroma components.

[0205] 10. The method of any of clauses 1-8, wherein the size of the buffer is dependent on chroma subsampling format of the current video block.

[0206] 11. The method of any of clauses 1-8, wherein the reference samples are stored in RGB format.

[0207] 12. The method of any of clauses 1-11, wherein the buffer is used for storing reconstructed samples before loop filtering and after loop filtering.

[0208] 13. The method of clause 12, wherein loop filtering includes deblocking filtering or adaptive loop filtering (ALF) or sample adaptive offset (SAO) filtering.

[0209] 14. A method of video processing, comprising: initializing, for a conversion between a current video block and a bitstream representation of the current video block, a buffer to store reference samples for the current video block using an intra-block copy coding mode using initial values for the reference samples; and performing the conversion using the reference samples stored in the buffer.

[0210] 15. The method of clause 14, wherein the initial values correspond to a constant.

[0211] 16. The method of any of clauses 14-15, wherein the initial values are a function of bit-depth of the current video block.

[0212] 17. The method of clause 15, wherein the constant corresponds to a mid-grey value.

[0213] 18. The method of clause 14, wherein the initial values correspond to pixel values of a previously decoded video block.

[0214] 19. The method of clause 18, wherein the previously decoded video block corresponds to a decoded block prior to in-loop filtering.

[0215] 20. The method of any of clauses 14-19, wherein a size of the buffer is at recited in one of clauses 1-13.

[0216] 21. The method of any of clauses 1-20, wherein pixel locations within the buffer as addressed using x and y numbers.

[0217] 22. The method of any of clauses 1-20, wherein pixel locations within the buffer as addressed using a single number that extends from 0 to M*N-1, where M and N are pixel width and pixel height of the buffer.

[0218] 23. The method of any of clauses 1-20, wherein, the current bitstream representation includes a block vector for the conversion, wherein the block vector, denoted as (BVx,BVy) is equal to (x-x0,y-y0), where (x0, y0) correspond to an upper-left position of a coding tree unit of the current video block.

[0219] 24. The method of any of clauses 1-20, wherein, the current bitstream representation includes a block vector for the conversion, wherein the block vector, denoted as (BVx,BVy) is equal to (x-x0+Tx,y-y0+Ty), where (x0, y0) correspond to an upper-left position of a coding tree unit of the current video block and wherein Tx and Ty are offset values.

[0220] 25. The method of clause 24, wherein Tx and Ty are pre-defined offset values.

[0221] 26. The method of any of clauses 1-20, wherein during the conversion, for a pixel at location (x0, y0) and having a block vector (BVx, BVy), a corresponding reference in the buffer is found at a reference location (x0+BVx, y0+BVy).

[0222] 27. The method of clause 26, wherein in case that the reference location is outside the buffer, the reference in the buffer is determined by clipping at a boundary of the buffer.

[0223] 28. The method of clause 26, wherein in case that the reference location is outside the buffer, the reference in the buffer is determined to have a predetermined value.

[0224] 29. The method of any of clauses 1-20, wherein during the conversion, for a pixel at location (x0, y0) and having a block vector (BVx, BVy), a corresponding reference in the buffer is found at a reference location ((x0+BVx) mod M, (y0+BVy) mod N) where "mod" is modulo operation and M and N are integers representing x and y dimensions of the buffer.

[0225] 30. A method of video processing, comprising: resetting, during a conversion between a video and a bitstream representation of the current video block, a buffer that stores reference samples for intra block copy coding at a video boundary; and performing the conversion using the reference samples stored in the buffer.

[0226] 31. The method of clause 30, wherein the video boundary corresponds to a new picture or a new tile.

[0227] 32. The method of clause 30, wherein the conversion is performed by updating, after the resetting, the buffer with reconstructed values of a Virtual Pipeline Data Unit (VPDU).

[0228] 33. The method of clause 30, wherein the conversion is performed by updating, after the resetting, the buffer with reconstructed values of a coding tree unit.

[0229] 34. The method of clause 30, wherein the resetting is performed at beginning of each coding tree unit row.

[0230] 35. The method of clause 1, wherein the size of the buffer corresponds to L 64x64 previously decoded blocks., where L is an integer.

[0231] 36. The method of any of clauses 1-35, wherein a vertical scan order is used for reading or storing samples in the buffer during the conversion.

[0232] 37. A method of video processing, comprising: using, for a conversion between a current video block and a bitstream representation of the current video block, a buffer to store reference samples for the current video block using an intra-block copy coding mode, wherein a first bit-depth of the buffer is different than a second bit-depth of the coded data; and performing the conversion using the reference samples stored in the buffer.

[0233] 38. The method of clause 37, wherein the first bit-depth is greater than the second bit-depth.

[0234] 39. The method of any of clauses 37-38, wherein the first bit-depth is identical to a bit-depth of a reconstruction buffer used during the conversion.

[0235] 40. The method of any of clauses 37-39, wherein the first bit-depth is signaled in the bitstream representation as a value or a difference value.

[0236] 41. The method of any of clauses 37-40, wherein the conversion uses different bit-depths for chroma and luma components.

[0237] Additional embodiments and examples of clauses 37 to 41 are described in Item 7 in Section 4.

[0238] 42. A method of video processing, comprising: performing a conversion between a current video block and a bitstream representation of the current video block using an intra-block copy mode in which a first precision used for prediction calculations during the conversion is lower than a second precision used for reconstruction calculations during the conversion.

[0239] 43. The method of clause 43, wherein the prediction calculations include determining a prediction sample value from a reconstructed sample value using clip{{p+[1<<(b-1)]}>>b,0,(1<<bitdepth)-1}<<b, where p is the reconstructed sample value, b is a predefined bit-shifting value, and bitdepth is a prediction sample precision.

[0240] Additional embodiments and examples of clauses 42 to 43 are described in Item 28 to 31 and 34 in Section 4.

[0241] 44. A method of video processing, comprising: performing a conversion between a current video block and a bitstream representation of the current video block using an intra-block copy mode in which a reference area of size nM x nM is used for a coding tree unit size MxM, where n and M are integers and wherein the current video block is positioned in the coding tree unit, and wherein the reference area is a nearest available nxn coding tree unit in a coding tree unit row corresponding to the current video block.

[0242] Additional embodiments and examples of clause 4 are described in Item 35 in Section 4.

[0243] 45. A method of video processing, comprising: performing a conversion between a current video block and a bitstream representation of the current video block using an intra-block copy mode in which a reference area of size nM x nM is used for a coding tree unit size other than MxM, where n and M are integers and wherein the current video block is positioned in the coding tree unit, and wherein the reference area is a nearest available nxn-1 coding tree unit in a coding tree unit row corresponding to the current video block.

[0244] Additional embodiments and examples of clause 4 are described in Item 36 in Section 4. FIGS. 8 and 9 show additional example embodiments.

[0245] 46. The method of claim 3, wherein M=mW and N=H, where W and H are width and height of a coding tree unit (CTU) of the current video block, and m is a positive integer.

[0246] 47. The method of claim 3, wherein M=W and N=nH, where W and H are width and height of a coding tree unit (CTU), and n is a positive integer.

[0247] 48. The method of claim 3, wherein M=mW and N=nH, where W and H are width and height of a coding tree unit (CTU), m and n are positive integers.

[0248] 49. The method of any of claims 46-48, wherein n and m depend on a size of the CTU.

[0249] 50. A method of video processing, comprising: determining, for a conversion between a current video block of a video and a bitstream representation of the current video block, validity of a block vector corresponding to the current video block of a component c of the video using a component X of the video, wherein the component X is different from a luma component of the video; and performing the conversion using the block vector upon determining that the block vector is valid for the current video block. Here, the block vector, denoted as (BVx,BVy) is equal to (x-x0,y-y0), where (x0, y0) correspond to an upper-left position of a coding tree unit of the current video block.

[0250] 51. The method of clause 50, wherein the component c corresponds to the luma component of the video.

[0251] 52. The method of clause 50, wherein the current video block is a chroma block and the video is in a 4:4:4 format.

[0252] 53. The method of clause 50, wherein the video is in a 4:2:0 format, and wherein the current video block is a chroma block starting at position (x, y), and wherein the determining comprises determining the block vector to be invalid for a case in which isRec(c, ((x+BVx)>>5<<5)+64-(((y+BVy)>>5)&1)*32+(x%32), ((y+BVy)>>5<<5) +(y%32)) is true.

[0253] 54. The method of clause 50, wherein the video is in a 4:2:0 format, and wherein the current video block is a chroma block starting at position (x, y), and wherein the determining comprises determining the block vector to be invalid for a case in which if isRec(c, x+BVx+Chroma_CTU_size, y) is true.

[0254] 55. A method of video processing, comprising: determining, selectively for a conversion between a current video block of a current virtual pipeline data unit (VPDU) of a video region and a bitstream representation of the current video block, to use K1 previously processed VPDUs from a first row of the video region and K2 previously processed VPDUs from a second row of the video region; and performing the conversion, wherein the conversion excludes using remaining of the current VPDU.

[0255] 56. The method of clause 55, wherein K1 = 1 and K2 = 2.

[0256] 57. The method of any of clauses 55-56, wherein the current video block is selectively processed based on a dimension of the video region or a dimension of the current VPDU.

[0257] 58. A method of video processing, comprising: performing a validity check of a block vector for a conversion between a current video block and a bitstream representation of the current video block, wherein the block vector is used for intra block copy mode; and using a result of the validity check to selectively use the block vector during the conversion.

[0258] 59. The method of clause 58, wherein an intra block copy (IBC) buffer is used during the conversion, wherein a width and a height of the IBC buffer as Wbuf and Hbuf an dimensions of the current video block are WxH and wherein the block vector is represented as (BVx, BVy), and wherein the current video block is in a current picture having dimensions Wpic and Hpic and a coding tree unit having Wctu and Hctu as width and height, and wherein the validity check uses a pre-determined rule.

[0259] 60. The method of any of clauses 58-59, wherein the current video block is a luma block, a chroma block, a coding unit CU, a transform unit TU, a 4x4 block , a 2x2 block, or a subblock of a parent block starting from pixel coordinates (X, Y).

[0260] 61. The method of any of clauses 58-60, wherein the validity check considers the block vector that falls outside a boundary of the current picture as valid.

[0261] 62. The method of any of clauses 58-60, wherein the validity check considers the block vector that falls outside a boundary of the coding tree unit as valid.

[0262] Items 23-30 in the previous section provide additional examples and variations of the above clauses 58-62.

[0263] 63. The method of any of clauses 1-62, wherein the conversion includes generating the bitstream representation from the current video block.

[0264] 64. The method of any of clauses 1-62, wherein the conversion includes generating pixel values of the current video block from the bitstream representation.

[0265] 65. A video encoder apparatus comprising a processor configured to implement a method recited in any one or more of clauses 1-62.

[0266] 66. A video decoder apparatus comprising a processor configured to implement a method recited in any one or more of clauses 1-62.

[0267] 67. A computer readable medium having code stored thereon, the code embodying processor-executable instructions for implementing a method recited in any of or more of clauses 1-62.

[0268] FIG. 7 is a block diagram of a hardware platform of a video / image processing apparatus 700. The apparatus 700 may be used to implement one or more of the methods described herein. The apparatus 700 may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus 700 may include one or more processors 702, one or more memories 704 and video processing hardware 706. The processor(s) 702 may be configured to implement one or more methods (including, but not limited to, method 600) described in the present document. The memory (memories) 704 may be used for storing data and code used for implementing the methods and techniques described herein. The video processing hardware 706 may be used to implement, in hardware circuitry, some techniques described in the present document.

[0269] The bitstream representation corresponding to a current video block need not be a contiguous set of bits and may be distributed across headers, parameter sets, and network abstraction layer (NAL) packets.

Section A: Another additional example embodiment

[0270] In Section A, we present another example embodiment in which the current version of the VVC standard may be modified for implementing some of the techniques described in the present document.

[0271] This section analyzes several issues in the current IBC reference buffer design and presents a different design to address the issues. An independent IBC reference buffer is proposed instead of mixing with decoding memory. Compared with the current anchor, the proposed scheme shows -0.99%/-0.71%/-0.79% AI/RA/LD-B luma BD-rate for class F and -2.57%/-1.81%/-1.36% for 4:2:0 TGM, with 6.7% memory reduction; or -1.31%/-1.01%/-0.81% for class F and -3.23%/-2.33%/-1.71% for 4:2:0 TGM with 6.7% memory increase.

A1. Introduction

[0272] Intra block copy, i.e. IBC (or current picture referencing, i.e. CPR previously) coding mode, is adopted. It is realized that IBC reference samples should be stored in on-chip memory and thus a limited reference area of one CTU is defined. To restrict the extra on-chip memory for the buffer, the current design reuses the 64x64 memory for decoding the current VPDU so that only 3 additional 64x64 blocks' memory is needed to support IBC. When CTU size is 128x128, currently the reference area is shown in FIG. 2.

[0273] In the current draft (VVC draft 4), the area is defined as

- The following conditions shall be true:

  (yCb + (mvL[1] >> 4)) >> CtbLog2SizeY = yCb >> CtbLog2SizeY (8-972)

(yCb + (mvL[1] >> 4) + cbHeight – 1) >> CtbLog2SizeY = yCb >> CtbLog2SizeY (8-973)

(xCb + (mvL[0] >> 4)) >> CtbLog2SizeY >= (xCb >> CtbLog2SizeY) – 1 (8-974)

(xCb + (mvL[0] >> 4) + cbWidth – 1) >> CtbLog2SizeY <= (xCb >> CtbLog2SizeY) (8-975)

[Ed. (SL): conditions (8-218) and (8-216) might have been checked by 6.4.X.]

– When (xCb + (mvL[0] >> 4)) >> CtbLog2SizeY is equal to (xCb >> CtbLog2SizeY) – 1, the derivation process for block availability as specified in clause 6.4.X [Ed. (BB): Neighbouring blocks availability checking process tbd] is invoked with the current luma location(xCurr, yCurr) set equal to (xCb, yCb) and the neighbouring luma location (((xCb + (mvL[0] >> 4) + CtbSizeY) >> (CtbLog2SizeY – 1)) << (CtbLog2SizeY – 1), ((yCb + (mvL[1] >> 4)) >> (CtbLog2SizeY – 1)) << (CtbLog2SizeY – 1)) as inputs, and the output shall be equal to FALSE.

[0274] Thus, the total reference size is a CTU.

A2. Potential issues of the current design

[0275] The current design assumes to reuse the 64x64 memory for decoding the current VPDU and the IBC reference is aligned to VPDU memory reuse accordingly. Such a design bundles VPDU decoding memory with the IBC buffer. There might be several issues:

1. To handle smaller CTU size might be an issue. Suppose that CTU size is 32x32, it is not clear whether the current 64x64 memory for decoding the current VPDU can support 32x32 level memory reuse efficiently in different architectures.
2. The reference area varies significantly. Accordingly, too many bitstream conformance constraints are introduced. It places extra burden to encoder to exploit reference area efficiently and avoid generating legal bitstreams. It also increases the possibility to have invalid BVs in different modules, e.g. merge list. To handle those invalid BVs may introduce extra logics or extra conformance constraints. It not only introduces burdens to encoder or decoder, it may also create divergence between BV coding and MV coding.
3. The design does not scale well. Because VPDU decoding is mixed with IBC buffer, it is not easy to increase or decrease reference area relative to the current one 128x128 CTU design. It may limit the flexibility to exploit a better coding efficiency vs. on-chip memory trade-off in the later development, e.g. a lower or higher profile.
4. The bit-depth of IBC reference buffer is linked with decoding buffer. Even though screen contents usually have a lower bit-depth than internal decoding bit-depth, the buffer still needs to

spend memory to store bits mostly representing rounding or quantization noises. The issue becomes even severe when considering higher decoding bit-depth configurations.

A3. A clear IBC buffer design

[0276] To address issues listed in the above sub-section, we propose to have a dedicated IBC buffer, which is not mixed with decoding memory.

[0277] For 128x128 CTU, the buffer is defined as 128x128 with 8-bit samples, when a CU (x, y) with size wxh has been decoded, its reconstruction before loop-filtering is converted to 8-bit and written to the wxh block area starting from position (x%128, y%128). Here the modulo operator % always returns a positive number, i.e. for $x < 0,\ x\%L \triangleq -(-x\%L)$, e.g. -3%128=125.

[0278] Assume that a pixel (x,y) is coded in IBC mode with BV=(BVx, BVy), it is prediction sample in the IBC reference buffer locates at ((x+BVx)%128, (y+BVy)%128) and the pixel value will be converted to 10-bit before prediction.

[0279] When the buffer is considered as (W, H), after decoding a CTU or CU starting from (x, y), the reconstructed pixels before loop-filtering will be stored in the buffer starting from (x%W, y%H). Thus, after decoding a CTU, the corresponding IBC reference buffer will be updated accordingly. Such setting might happen when CTU size is not 128x128. For example, for 64x64 CTU, with the current buffer size, it can be considered as a 256x64 buffer. For 64x64 CTU, figure 2 shows the buffer status.

[0280] FIG. 12 is an illustration of IBC reference buffer status, where a block denotes a 64x64 CTU.

[0281] In such a design, because the IBC buffer is different from the VPDU decoding memory, all the IBC reference buffer can be used as reference.

[0282] When the bit-depth of the IBC buffer is 8-bit, compared with the current design that needs 3 additional 10-bit 64x64 buffer, the on-chip memory increase is (8*4)/(10*3)-100%=6.7%.

[0283] If we further reduce the bit-depth. The memory requirement can be further reduced. For example, for 7-bit buffer, the on-chip memory saving is 100%-(7*4)/(10*3)=6.7%.

[0284] With the design, the only bitstream conformance constraint is that the reference block shall be within the reconstructed area in the current CTU row of the current Tile.

[0285] When initialization to 512 is allowed at the beginning of each CTU row, all bitstream conformance constraints can be removed.

A4. Experimental results

[0286] In some embodiments, the disclosed methods can be implemented using VTM-4.0 software.

[0287] For a 10-bit buffer implementation and CTC, the decoder is fully compatible to the current VTM4.0 encoder, which means that the proposed decoder can exactly decode the VTM-4.0 CTC bitstreams.

[0288] For a 7-bit buffer implementation, the results shown in Table I.

[0289] For a 8-bit buffer implementation, the results shown in Table II.

Table I. Performance with a 7-bit buffer. The anchor is VTM-4.0 with IBC on for all sequences.

	All Intra				
	Over VTM-4.0 w/ IBC on				
	Y	U	V	EncT	DecT
Class A1	-0.01%	-0.09%	-0.10%	132%	101%
Class A2	0.05%	0.00%	0.06%	135%	100%
Class B	0.00%	-0.02%	0.01%	135%	100%
Class C	-0.02%	0.01%	0.03%	130%	98%
Class E	-0.13%	-0.16%	-0.04%	135%	99%
Overall	-0.02%	-0.05%	0.00%	133%	100%
Class D	0.04%	0.04%	0.12%	127%	107%
Class F	-0.99%	-1.14%	-1.18%	115%	99%
4:2:0 TGM	-2.57%	-2.73%	-2.67%	104%	102%

	Random Access				
	Over VTM-4.0 w/ IBC on				
	Y	U	V	EncT	DecT
Class A1	0.02%	-0.01%	0.01%	109%	100%
Class A2	0.00%	-0.04%	0.03%	111%	100%
Class B	-0.01%	-0.10%	-0.22%	113%	101%
Class C	-0.01%	0.17%	0.12%	115%	100%
Class E					
Overall	0.00%	0.00%	-0.04%	112%	100%
Class D	0.05%	0.16%	0.20%	117%	101%
Class F	-0.71%	-0.77%	-0.77%	109%	99%
4:2:0 TGM	-1.81%	-1.65%	-1.64%	107%	101%

	Low delay B				
	Over VTM-4.0 w/ IBC on				
	Y	U	V	EncT	DecT
Class A1					
Class A2					
Class B	0.01%	0.36%	0.30%	114%	95%
Class C	-0.01%	-0.12%	-0.10%	120%	98%
Class E	0.10%	0.20%	0.18%	107%	99%
Overall	0.03%	0.16%	0.13%	114%	97%
Class D	-0.01%	1.07%	0.18%	123%	104%
Class F	-0.79%	-0.89%	-1.01%	110%	100%
4:2:0 TGM	-1.36%	-1.30%	-1.26%	109%	102%

Table II. Performance with a 8-bit buffer. The anchor is VTM-4.0 with IBC on for all sequences.

	All Intra				
	Over VTM-4.0 w/ IBC on				
	Y	U	V	EncT	DecT
Class A1	-0.01%	0.02%	-0.10%	129%	102%
Class A2	0.02%	-0.06%	-0.02%	134%	102%
Class B	-0.04%	-0.02%	-0.07%	135%	101%
Class C	-0.03%	0.04%	0.00%	130%	98%
Class E	-0.16%	-0.14%	-0.08%	134%	100%
Overall	-0.04%	-0.03%	-0.05%	133%	100%
Class D	0.00%	0.04%	0.02%	126%	101%
Class F	-1.31%	-1.27%	-1.29%	114%	98%
4:2:0 TGM	-3.23%	-3.27%	-3.24%	101%	100%

	Random Access				
	Over VTM-4.0 w/ IBC on				
	Y	U	V	EncT	DecT
Class A1	-0.01%	-0.08%	0.04%	107%	99%
Class A2	-0.03%	-0.16%	0.06%	110%	99%
Class B	-0.01%	-0.14%	-0.22%	111%	99%
Class C	-0.01%	0.15%	0.09%	115%	100%
Class E					

Overall	-0.01%	-0.05%	-0.03%	111%	99%
Class D	0.01%	0.19%	0.22%	116%	101%
Class F	-1.01%	-0.99%	-1.01%	108%	99%
4:2:0 TGM	-2.33%	-2.14%	-2.19%	105%	100%

	Low delay B				
	Over VTM-4.0 w/ IBC on				
	Y	U	V	EncT	DecT
Class A1					
Class A2					
Class B	0.00%	0.04%	-0.14%	113%	#NUM!
Class C	-0.05%	-0.28%	-0.15%	119%	98%
Class E	0.04%	-0.16%	0.43%	107%	#NUM!
Overall	0.00%	-0.11%	0.00%	113%	#NUM!
Class D	-0.07%	1.14%	0.13%	122%	99%
Class F	-0.81%	-0.92%	-0.96%	111%	99%
4:2:0 TGM	-1.71%	-1.67%	-1.71%	106%	95%

[0290] FIG. 17 is a block diagram showing an example video processing system 1700 in which various techniques disclosed herein may be implemented. Various implementations may include some or all of the components of the system 1700. The system 1700 may include input 1702 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 1702 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.

[0291] The system 1700 may include a coding component 1704 that may implement the various coding or encoding methods described in the present document. The coding component 1704 may reduce the average bitrate of video from the input 1702 to the output of the coding component 1704 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 1704 may be either stored, or transmitted via a communication connected, as represented by the component 1706. The stored or communicated bitstream (or coded) representation of the video received at the input 1702 may be used by the component 1708 for generating pixel values or displayable video that is sent to a display interface 1710. 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.

[0292] 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.

[0293] FIG. 18 is a flowchart of an example method of visual data processing. Steps of this flowchart are discussed in connection with example 23 discussed in Section 4 of this document. At step 1802, the process determines, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the current video block, a block vector (BVx,BVy), wherein validity of the block vector (BVx, BVy) is independent of (1) a location (P, Q) of a sample block and/or (2) whether a sample at the location (P,Q) is reconstructed, and/or (3) a location of the current video block, wherein, the block vector (BVx, BVy) represents a pixel displacement between the current video block and the sample block. At step 1804, the process performs, using the block vector, the conversion in an intra block copy mode which is based on a reconstructed block located in same video region with the current video block comprising reference samples used for deriving a prediction block of the current video block, wherein, during the conversion, a prediction sample with a location (A, B) from reference samples in a buffer is determined at least according to a size of the buffer and/or the block vector (BVx, BVy).

[0294] FIG. 19 is a flowchart of an example method of visual data processing. Steps of this flowchart are discussed in connection with example 23 discussed in Section 4 of this document. At step 1902, the process determines, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, whether a block vector (BVx, BVy) corresponding to the current video block is valid according to a rule, wherein the block vector (BVx, BVy) represents a pixel displacement between the current video block and a sample block. At step 1904, the process performs, using the block vector, the conversion based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, wherein the rule specifies that the block vector (BVx, BVy) is valid in case that (1) one or more samples from the sample block are outside the current picture and/or (2) one or more samples from the sample block are outside at least one coding tree unit (CTU) associated with the current video block, and/or (3) one or more samples from the sample block fail to be reconstructed.

[0295] FIG. 20 is a flowchart of an example method of visual data processing. Steps of this flowchart are discussed in connection with example 44 discussed in Section 4 of this document. At step 2002, the process performs a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, wherein, the conversion is based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, and wherein a virtual buffer of a defined size is used for tracking availability of the reference samples for deriving the prediction block.

[0296] FIG. 21 is a flowchart of an example method of visual data processing. Steps of this flowchart are discussed in connection with example 51 discussed in Section 4 of this document. At step 2102, the process maintains, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, a buffer comprising reference samples from the current picture for a derivation of a prediction block of the current video block, wherein one or more reference samples in the buffer that are marked unavailable for the derivation have values outside of a pixel value range.

[0297] FIG. 22 is a flowchart of an example method of visual data processing. Steps of this flowchart are discussed in connection with example 54 discussed in Section 4 of this document. At step 2202, the process performs a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data using a buffer comprising reference samples from the current picture for derivation of a prediction block of the current video block, wherein the conversion is based according to rule which specifies that, for the bitstream representation to conform the rule, a reference sample in the buffer is to satisfy a bitstream conformance constraint.

[0298] Some embodiments of the present document are now presented in clause-based format.

[0299] L1. A visual media processing method, comprising:

[0300] determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the current video block, a block vector (BVx,BVy), wherein validity of the block vector (BVx, BVy) is independent of (1) a location (P, Q) of a sample block and/or (2) whether a sample at the location (P,Q) is reconstructed, and/or (3) a location of the current video block, wherein, the block vector (BVx, BVy) represents a pixel displacement between the current video block and the sample block; and

[0301] performing, using the block vector, the conversion in an intra block copy mode which is based on a reconstructed block located in same video region with the current video block comprising reference samples used for deriving a prediction block of the current video block, wherein, during the conversion, a prediction sample with a location (A, B) from reference samples in a buffer is determined at least according to a size of the buffer and/or the block vector (BVx, BVy).

[0302] L2. A visual media processing method, comprising:

[0303] determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, whether a block vector (BVx, BVy) corresponding to the current video block is valid according to a rule, wherein the block vector (BVx, BVy) represents a pixel displacement between the current video block and a sample block; and

[0304] performing, using the block vector, the conversion based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, wherein the rule specifies that the block vector (BVx, BVy) is valid in case that (1) one or more samples from the sample block are outside the current picture and/or (2) one or more samples from the sample block are outside at least one coding tree unit (CTU) associated with the current video block, and/or (3) one or more samples from the sample block fail to be reconstructed.

[0305] L3. The method of clause L2, wherein, upon identifying that the block vector (BVx, BVy) is valid, a prediction sample with a location (A, B) from reference samples in a buffer is determined at least according to a size of the buffer and/or the block vector (BVx, BVy).

[0306] L4. The method of any one or more of clauses L1 or L3, wherein the reference samples in the buffer corresponds to reconstructed samples of a region of the current picture.

[0307] L5. The method of clause L4, wherein the region includes a coding tree unit (CTU) row associated with the current video block.

[0308] L6. The method of any one or more of clauses L1-L5, wherein the block vector (BVx, BVy) is determined to be valid regardless of whether the location (P, Q) computed according to the block vector (BVx, BVy) and the upper-left position (x, y) of the current video block is outside a boundary of a picture.

[0309] L7. The method of clause L6, wherein the block vector (BVx, BVy) is valid regardless of whether $x+BVx<0$ or $x+BVx>0$.

[0310] L8. The method of clause L6, wherein the block vector (BVx, BVy) is valid regardless of whether $x+W+BVx > W_{pic}$ or $x+W+BVx < W_{pic}$, where W denotes a width of the current video block and W_{pic} denotes a width of the picture.

[0311] L9. The method of clause L6, wherein the block vector (BVx, BVy) is valid regardless of whether $y+BVy<0$ or $y+BVy>0$.

[0312] L10. The method of clause L6, wherein the block vector (BVx, BVy) is valid regardless of whether $x+H+BVx > H_{pic}$ or $x+H+BVx < H_{pic}$, where H denotes a height of the current video block and H_{pic} denotes a height of the picture.

[0313] L11. The method of any one or more of clauses L1-L5, wherein the block vector (BVx, BVy) is valid regardless of whether the location (P, Q) computed according to the block vector (BVx, BVy) and the upper-left position (x, y) of the current video block is outside a coding tree unit including the current video block.

[0314] L12. The method of clause L11, wherein the block vector (BVx, BVy) is valid regardless of whether $y+BVy<floor(y/H_{ctu})* H_{ctu}$ or $y+BVy>floor(y/H_{ctu})* H_{ctu}$, where H_{ctu} denotes a height of the coding tree unit and floor(a) is the largest integer no greater than a.

[0315] L13. The method of clause L11, wherein the block vector (BVx, BVy) is valid regardless of whether $y+H+BVy<floor(y/H_{ctu})*H_{ctu}$ or $y+H+BVy>floor(y/H_{ctu})*H_{ctu}$, where H denotes a height of the current video block, H_{ctu} denotes a height of the coding tree unit, and floor(a) is the largest integer no greater than a.

[0316] L14. The method of any one or more of clauses L1-L5, wherein the block vector (BVx, BVy) is valid regardless of whether the location (P, Q) computed according to the block vector (BVx, BVy) and the upper-left position (x, y) of the current video block is outside a coding tree unit including the current video block and (n-1) coding tree units along a left direction.

[0317] L15. The method of clause L14, wherein the block vector (BVx, BVy) is valid regardless of whether $x+BVx<floor(x/W_{ctu})*W_{ctu}-(n-1)*W_{ctu}$ or $x+BVx>floor(X/W_{ctu})*W_{ctu}-(n-1)*W_{ctu}$, where W_{ctu} denotes a weight of the coding tree unit and floor(a) is the largest integer no greater than a.

[0318] L16. The method of clause L14, wherein the block vector (BVx, BVy) is valid regardless of whether $x+W+BVx>floor(X/W_{ctu})*W_{ctu}+W_{ctu}$ or $x+W+BVx<floor(X/W_{ctu})*W_{ctu}+W_{ctu}$, where W denotes a width of the current video block, W_{ctu} denotes a weight of the coding tree unit, and floor(a) is the largest integer no greater than a.

[0319] L17. The method of any one or more of clauses L1-L5, wherein the block vector (BVx, BVy) is valid regardless of whether the location (P, Q) computed according to the block vector (BVx, BVy) and the upper-left position (x, y) of the current video block is outside a current CTU row including a current coding tree unit including the current video block.

[0320] L18. The method of clause L17, wherein the block vector (BVx, BVy) is valid regardless of whether Y+BVy<floor(Y/ Hctu)* Hctu or Y+H+BVy>=floor(Y/ Hctu)*Hctu+ Hctu, wherein, W_{ctu} and H_{ctu} denote a width and height of a CTU respectively and floor(a) is the largest integer no greater than a.

[0321] L19. The method of any one or more of clauses L1-L5, wherein the block vector (BVx, BVy) is determined to be valid regardless of whether a sample fails to be reconstructed.

[0322] L20. The method of clause L19, wherein the block vector (BVx, BVy) is valid regardless of whether isRec(x+BVx, y+ BVy) is false, where isRec(x,y) is true if pixel (x,y) is reconstructed by an intra block copy mode.

[0323] L21. The method of clause L19, wherein the block vector (BVx, BVy) is valid regardless of whether isRec(x+BVx+W-1, y+BVy) is false, where isRec(x,y) is true if pixel (x, y) is reconstructed by an intra block copy mode and W denotes a width of the current video block.

[0324] L22. The method of clause L19, wherein the block vector (BVx, BVy) is valid regardless of whether isRec(x+BVx, y+BVy+H-1) is false, where isRec(x,y) is true if pixel (x, y) is reconstructed by an intra block copy mode and H denotes a height of the current video block.

[0325] L23. The method of clause L19, wherein the block vector (BVx, BVy) is valid regardless of whether isRec(x+BVx+W-1, y+BVy+H-1) is false, where isRec(x, y) is true if pixel (x, y) is reconstructed by an intra block copy mode, W denotes a width of the current video block, and H denotes a height of the current video block.

[0326] L24. The method of any one or more of clauses L1-L5 wherein the block vector (BVx, BVy) is determined to be valid regardless of whether the current video block is included in a first coding tree unit of a coding tree unit row.

[0327] L25. The method of any one or more of clauses L1-L5, wherein the block vector (BVx, BVy) is determined to be valid when all the following conditions are satisfied: (i) x + BVx >= 0, (ii) y + BVy >= floor(y / Hctu), and (iii) isRec(x + BVx + W - 1, y + BVy + H - 1) is true, where isRec(x, y) is true if sample (x, y) is reconstructed by an intra block copy mode, W denotes a width of the current video block, H denotes a height of the current video block, where floor(a) is the largest integer no greater than a.

[0328] L26. The method of clause L25, wherein the block vector is a located in a first CTU in a CTU row.

[0329] L27. The method of clause L3, wherein the prediction sample with the location (A, B) is determined according to the size of the buffer, the block vector (BVx, BVy), and the upper-left position (x, y).

[0330] L28. The method of clause L27, wherein the prediction sample with the location (A, B) comprises a predication sample with a location computed according to ((X+BVx)%Wbuf, (Y+BVy)%Hbuf), wherein Wbuf and Hbuf denote a width of the buffer and a height of the buffer respectively.

[0331] L29. The method of any one or more of clauses L1-L28, wherein the conversion is performed in an intra block copy mode.

[0332] M1. A visual media processing method, comprising:

[0333] performing a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data,

[0334] wherein, the conversion is based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, and

[0335] wherein a virtual buffer of a defined size is used for tracking availability of the reference samples for deriving the prediction block.

[0336] M2. The method of clause M1, wherein the virtual buffer is maintained using a virtual pipeline data unit (VPDU), and wherein a size of the virtual buffer is $m*W_{VPDU}$ x $n*H_{VPDU}$, where W_{VPDU} and H_{VPDU} denote a width and a height of the VPDU.

[0337] M3. The method of clause M2, wherein m=4 and n=2.

[0338] M4. The method of clause M2, wherein m and/or n are based at least in part on a resolution of a picture associated with the current video block or a size of a coding tree unit including the current video block.

[0339] M5. The method of clause M2, wherein m and/or are predefined quantities.

[0340] M6. The method of clause M2, wherein m and/or are signaled as fields in the bitstream representation.

[0341] M7. The method of clause M1, wherein a sample in the current video block is mapped to $(x\%(m*W_{VPDU}), y\%(n*H_{VPDU}))$ in the virtual buffer, where the sample in the current video block is located at (x, y) relative to a upper-left corner of a picture; "x%y" is defined as y = x – y*floor(x/y), where floor(a) is the largest integer no greater than a, and W_{VPDU} and H_{VPDU} denote a width and a height of the VPDU.

[0342] M8. The method of clause M1, further comprising:

[0343] using an array for tracking availabilities of samples stored in the virtual buffer.

[0344] M9. The method of clause M8, wherein the array includes a flag to indicate if one or more samples stored in the buffer are used for prediction in an intra block copy mode.

[0345] M10. The method of clause M8, wherein the array corresponds to one or more VPDUs of size 3x2.

[0346] M11. The method of clause M8, wherein the array corresponds to one or more VPDUs of size 4x2.

[0347] M12. The method of clause M1, wherein a subset of samples stored in the virtual buffer are flagged as unavailable for prediction.

[0348] M13. The method of clause M12, wherein, the subset of samples flagged as unavailable for prediction are based on a position of a most-recently processed VPDU.

[0349] M14. The method of clause M13, wherein the samples are flagged as unavailable at a beginning of processing a VPDU.

[0350] M15. The method of clause M14, wherein, if yPrevVPDU%(n*H_{VPDU}) is equal to 0, then the subset of samples located at positions (x, y) are flagged as unavailable, wherein x lies in a first predetermined range and y lies in a second predetermined range, where (xPrevVPDU, yPrevVPDU) denotes an upper-left corner of a coding tree unit of a most-recently processed VPDU, and W_{VPDU} and H_{VPDU} denote a width and a height of the VPDU.

[0351] M16. The method of clause M15, wherein the first range is expressed as [xPrevVPDU - 2W_{VPDU}+ 2mW_{VPDU})% mW_{VPDU}, ((xPrevVPDU - 2*W_{VPDU} + 2*m*W_{VPDU})% (m*W_{VPDU}))-1+W_{VPDU}] and the second range is expressed as [yPrevVPDU%(n*H_{VPDU}), (yPrevVPDU%(n*H_{VPDU}))-1+H_{VPDU}].

[0352] M17. The method of clause M15, wherein the first range is expressed as [xPrevVPDU - 2*W_{VPDU}+ 2*m*W_{VPDU})% mW_{VPDU}, ((xPrevVPDU - 2*W_{VPDU}+ 2*m*W_{VPDU})% (m*W_{VPDU}))-1+W_{VPDU}] and the second range is expressed as [yPrevVPDU%(n*H_{VPDU}), (yPrevVPDU%(n*H_{VPDU}))-1+ H_{VPDU}].

[0353] M18. The method of clause M14, wherein, if yPrevVPDU%(n*H_{VPDU}) is not equal to 0, then the subset of samples located at positions (x, y) are flagged as unavailable, wherein x lies in a first predetermined range and y lies in a second predetermined range, where (xPrevVPDU, yPrevVPDU) denotes an upper-left corner of a coding tree unit of a most-recently processed VPDU, and W_{VPDU} and H_{VPDU} denote a width and a height of the VPDU.

[0354] M19. The method of clause M18, wherein the first range is expressed as [xPrevVPDU - W_{VPDU}+ 2*m*W_{VPDU})% (m*W_{VPDU}), ((xPrevVPDU - W_{VPDU}+ 2*m*W_{VPDU})% (m*W_{VPDU}))-1+W_{VPDU}] and the second range is expressed as [yPrevVPDU%(n*H_{VPDU}), (yPrevVPDU%(n*H_{VPDU}))-1+ H_{VPDU}].

[0355] M20. The method of clause M18, wherein the first range is expressed as [xPrevVPDU - W_{VPDU} + 2*m*W_{VPDU})% mW_{VPDU}, ((xPrevVPDU - W_{VPDU}+ 2*m*W_{VPDU})% (m*W_{VPDU}))-1+W_{VPDU}] and the second range is expressed as [yPrevVPDU%(n*H_{VPDU}), (yPrevVPDU%(n*H_{VPDU}))-1+ H_{VPDU}].

[0356] M21. The method of clause M12, wherein, when the coding tree includes VPDUs, the subset of samples flagged as unavailable for prediction are based on a position of a most-recently processed coding tree unit.

[0357] M22. The method of clause M21, wherein the samples are flagged as unavailable at a beginning of processing a coding tree unit.

[0358] M23. The method of any one or more of clauses M1-M22, further comprising:

[0359] determining validity of a block vector corresponding to the current video block based on an upper-left position of the current video block, a bottom-left position of the current video block, and a bottom-right position of the current video block, wherein the determining excludes use of a upper-right position of the current video block.

[0360] M24. The method of any one or more of clauses M1-M23, wherein the conversion is performed in an intra block copy mode.

[0361] N1. A visual media processing method, comprising:

[0362] maintaining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, a buffer comprising reference samples from the current picture for a derivation of a prediction block of the current video block,

[0363] wherein one or more reference samples in the buffer that are marked unavailable for the derivation have values outside of a pixel value range.

[0364] N2. The method of clause N1, wherein the pixel value range is expressed as [0, 1<<(bit_depth) – 1], where bit_depth is a positive integer.

[0365] N3. The method of clause N2, wherein bit_depth is a precision used for processing the sample.

[0366] N4. The method of clause N1, wherein the method further comprises:

[0367] initializing a set of samples in the buffer to a predetermined value indicative of unavailability of the set of samples.

[0368] N5. The method of clause N4, wherein the predetermined value is -1.

[0369] N6. The method of any one or more of clauses N4-N5, wherein locations of the set of samples and/or whether to initialize the set of the samples to the predetermined value is/are based on one or more of: a position of the current video block, a size of the current video block, a size of a VPDU including the current video block, and/or a size of a coding tree unit including the current video block.

[0370] N7. The method of clause N6, wherein, if (xCb%vSize) is equal to 0 and (yCb%vSize) is equal to 0, the set of samples are marked as unavailable, where xCb, yCb denote a position of the current video block relative to the sample and vSize= min(ctbSize, 64), where ctbSize denotes a width or a height of the coding tree unit.

[0371] N8. The method of clause N1, wherein, if a size of the current video block is less than min(ctbSize, 64), the set of samples in the buffer are marked unavailable, where ctbSize denotes a width or a height of the coding tree unit.

[0372] N9. The method of clause N8, wherein locations of the plurality of samples are related to a size of a VPDU.

[0373] N10. The method of clause N8, wherein locations of the set of samples are related to a size of the coding tree unit including the current video block.

[0374] N11. The method of clause N4, wherein the set of samples in the buffer have with positions expressed as (x%wIbcBuf, y%hIbcBuf), where x = xV, …,xV+ctbSize-1 and y=yV,…,yV+ctbSize-1, and xV, yV denote the upper-left position of a VPDU relative to an upper-left position of a picture, where ctbSize denotes a size of the coding tree unit including the current video block, and wIbcBuf and hIbcBuf denote a buffer width and a buffer height.

[0375] N12. The method of clause N11, wherein the set of samples in the buffer are initalized to -1.

[0376] N13. The method of clause N4, wherein the set of samples are initialized at a beginning of decoding a video unit.

[0377] N14. The method of any one or more of clauses N1-N13, wherein the conversion is performed in an intra block copy mode.

[0378] O1. A visual media processing method, comprising:

[0379] performing a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data using a buffer comprising reference samples from the current picture for derivation of a prediction block of the current video block,

[0380] wherein the conversion is based according to rule which specifies that, for the bitstream representation to conform the rule, a reference sample in the buffer is to satisfy a bitstream conformance constraint.

[0381] O2. The method of clause O1, wherein the bitstream conformance constraint is based on at least one of (1) a value of the reference sample in the buffer and/or (2) an availability information of the sample in the buffer.

[0382] O3. The method of any one or more of clauses O1-O2, wherein the bitstream conformance constraint specifies that the bitstream representation is non-conforming, if the sample in the buffer has a value outside of a pixel range.

[0383] O4. The method of clause O3, wherein the range is [K0, K1], wherein K0 is set to 0 and K1 is set to (1<<BitDepth-1), where BitDepth represents a precision of a prediction sample.

[0384] O5. The method of any one or more of clauses O1-O2, wherein the bitstream conformance constraint specifies that the bitstream representation is non-conforming, if the availability information of the sample in the buffer indicates that the sample is unavailable for the current video block.

[0385] O6. The method of any one or more of clauses O1-O2, wherein the sample is a luma sample, and wherein the bitstream conformance constraint specifies that the bitstream representation is non-conforming, if the availability information of the sample in the buffer indicates that the sample is unavailable for the current video block and a single tree partitioning is used for the current video block.

[0386] O7. The method of clause any one or more of clauses O1-O6, further comprising:

[0387] marking the availability information of the sample according to the value of the sample in the buffer.

[0388] O8. The method of clause O7, wherein, if the value of the sample lies in an interval denoted as [K0, K1], the availability information of the sample is marked as available.

[0389] O9. The method of clause O8, wherein K0 is set to 0 and K1 is set to (1<<BitDepth-1), where BitDepth is represents a precision of a prediction sample.

[0390] O10. The method of any one or more of clauses O1-O8, wherein the bitstream conformance constraint is further based on a partitioning type and a tree type of a coding unit associated with the current video block.

[0391] O11. The method of clause O10, wherein, if the partitioning type is dual tree and the tree type is single tree, then the bitstream conformance constraint specifies checking if all color components of the sample are marked as unavailable.

[0392] O12. The method of clause O10, wherein if the partitioning type is dual tree and the tree type is dual tree, then the bitstream conformance constraint specifies excludes checking if a chroma component of the sample is marked as unavailable.

[0393] O13. The method of any one or more of clauses O1-O12, wherein the conversion is performed in an intra block copy mode.

[0394] XX. The method of any of clauses L1-XX, wherein the conversion includes generating the bitstream representation from the current video block.

[0395] XX. The method of any of clauses L1-XX, wherein the conversion includes generating pixel values of the current video block from the bitstream representation.

[0396] XX. A video encoder apparatus comprising a processor configured to implement a method recited in any one or more of clauses L1-XX.

[0397] XX. A video decoder apparatus comprising a processor configured to implement a method recited in any one or more of clauses L1-XX.

[0398] XX. A computer readable medium having code stored thereon, the code embodying processor-executable instructions for implementing a method recited in any of or more of clauses L1-XX.

[0399] 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.

[0400] 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.

[0401] 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.

[0402] 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.

[0403] 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).

[0404] 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.

[0405] 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.

[0406] 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.

[0407] 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.

[0408] Only a few implementations and examples are described and other implementations, enhancements and variations can be made based on what is described and illustrated in this patent document.

CLAIMS

What is claimed is:

1. A visual media processing method, comprising:

determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the current video block, a block vector (BVx,BVy), wherein validity of the block vector (BVx, BVy) is independent of (1) a location (P, Q) of a sample block and/or (2) whether a sample at the location (P,Q) is reconstructed, and/or (3) a location of the current video block, wherein, the block vector (BVx, BVy) represents a pixel displacement between the current video block and the sample block; and

performing, using the block vector, the conversion in an intra block copy mode which is based on a reconstructed block located in same video region with the current video block comprising reference samples used for deriving a prediction block of the current video block, wherein, during the conversion, a prediction sample with a location (A, B) from reference samples in a buffer is determined at least according to a size of the buffer and/or the block vector (BVx, BVy).

2. A visual media processing method, comprising:

determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, whether a block vector (BVx, BVy) corresponding to the current video block is valid according to a rule, wherein the block vector (BVx, BVy) represents a pixel displacement between the current video block and a sample block; and

performing, using the block vector, the conversion based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, wherein the rule specifies that the block vector (BVx, BVy) is valid in case that (1) one or more samples from the sample block are outside the current picture and/or (2) one or more samples from the sample block are outside at least one coding tree unit (CTU) associated with the current video block, and/or (3) one or more samples from the sample block fail to be reconstructed.

3. The method of claim 2, wherein, upon identifying that the block vector (BVx, BVy) is valid, a prediction sample with a location (A, B) from reference samples in a buffer is determined at least according to a size of the buffer and/or the block vector (BVx, BVy).

4. The method of any one or more of claims 1 or 3, wherein the reference samples in the buffer corresponds to reconstructed samples of a region of the current picture.

5. The method of claim 4, wherein the region includes a coding tree unit (CTU) row associated with the current video block.

6. The method of any one or more of claims 1-5, wherein the block vector (BVx, BVy) is determined to be valid regardless of whether the location (P, Q) computed according to the block vector (BVx, BVy) and the upper-left position (x, y) of the current video block is outside a boundary of a picture.

7. The method of claim 6, wherein the block vector (BVx, BVy) is valid regardless of whether $x+BVx<0$ or $x+BVx>0$.

8. The method of claim 6, wherein the block vector (BVx, BVy) is valid regardless of whether $x+W+BVx > W_{pic}$ or $x+W+BVx < W_{pic}$, where W denotes a width of the current video block and W_{pic} denotes a width of the picture.

9. The method of claim 6, wherein the block vector (BVx, BVy) is valid regardless of whether $y+BVy<0$ or $y+BVy>0$.

10. The method of claim 6, wherein the block vector (BVx, BVy) is valid regardless of whether $x+H+BVx > H_{pic}$ or $x+H+BVx < H_{pic}$, where H denotes a height of the current video block and H_{pic} denotes a height of the picture.

11. The method of any one or more of claims 1-5, wherein the block vector (BVx, BVy) is valid regardless of whether the location (P, Q) computed according to the block vector (BVx, BVy) and the upper-left position (x, y) of the current video block is outside a coding tree unit including the current video block.

12. The method of claim 11, wherein the block vector (BVx, BVy) is valid regardless of whether $y+BVy<floor(y/H_{ctu})* H_{ctu}$ or $y+BVy>floor(y/H_{ctu})* H_{ctu}$, where H_{ctu} denotes a height of the coding tree unit and floor(a) is the largest integer no greater than a.

13. The method of claim 11, wherein the block vector (BVx, BVy) is valid regardless of whether $y+H+BVy<floor(y/H_{ctu})*H_{ctu}$ or $y+H+BVy>floor(y/H_{ctu})*H_{ctu}$, where H denotes a height of the current video block, H_{ctu} denotes a height of the coding tree unit, and floor(a) is the largest integer no greater than a.

14. The method of any one or more of claims 1-5, wherein the block vector (BVx, BVy) is valid regardless of whether the location (P, Q) computed according to the block vector (BVx, BVy) and the upper-left position (x, y) of the current video block is outside a coding tree unit including the current video block and (n-1) coding tree units along a left direction.

15. The method of claim 14, wherein the block vector (BVx, BVy) is valid regardless of whether $x+BVx<floor(x/W_{ctu})*W_{ctu}-(n-1)*W_{ctu}$ or $x+BVx>floor(X/W_{ctu})*W_{ctu}-(n-1)*W_{ctu}$, where W_{ctu} denotes a weight of the coding tree unit and floor(a) is the largest integer no greater than a.

16. The method of claim 14, wherein the block vector (BVx, BVy) is valid regardless of whether $x+W+BVx>floor(X/W_{ctu})*W_{ctu}+W_{ctu}$ or $x+W+BVx<floor(X/W_{ctu})*W_{ctu}+W_{ctu}$, where W denotes a width of the current video block, W_{ctu} denotes a weight of the coding tree unit, and floor(a) is the largest integer no greater than a.

17. The method of any one or more of claims 1-5, wherein the block vector (BVx, BVy) is valid regardless of whether the location (P, Q) computed according to the block vector (BVx, BVy) and the upper-left position (x, y) of the current video block is outside a current CTU row including a current coding tree unit including the current video block.

18. The method of claim 17, wherein the block vector (BVx, BVy) is valid regardless of whether $Y+BVy<floor(Y/ Hctu)*$ Hctu or $Y+H+BVy>=floor(Y/ Hctu)*Hctu+$ Hctu, wherein, W_{ctu} and H_{ctu} denote a width and height of a CTU respectively and floor(a) is the largest integer no greater than a.

19. The method of any one or more of claims 1-5, wherein the block vector (BVx, BVy) is determined to be valid regardless of whether a sample fails to be reconstructed.

20. The method of claim 19, wherein the block vector (BVx, BVy) is valid regardless of whether isRec(x+BVx, y+ BVy) is false, where isRec(x,y) is true if pixel (x,y) is reconstructed by an intra block copy mode.

21. The method of claim 19, wherein the block vector (BVx, BVy) is valid regardless of whether isRec(x+BVx+W-1, y+BVy) is false, where isRec(x,y) is true if pixel (x, y) is reconstructed by an intra block copy mode and W denotes a width of the current video block.

22. The method of claim 19, wherein the block vector (BVx, BVy) is valid regardless of whether isRec(x+BVx, y+BVy+H-1) is false, where isRec(x,y) is true if pixel (x, y) is reconstructed by an intra block copy mode and H denotes a height of the current video block.

23. The method of claim 19, wherein the block vector (BVx, BVy) is valid regardless of whether isRec(x+BVx+W-1, y+BVy+H-1) is false, where isRec(x, y) is true if pixel (x, y) is reconstructed by an intra block copy mode, W denotes a width of the current video block, and H denotes a height of the current video block.

24. The method of any one or more of claims 1-5 wherein the block vector (BVx, BVy) is determined to be valid regardless of whether the current video block is included in a first coding tree unit of a coding tree unit row.

25. The method of any one or more of claims 1-5, wherein the block vector (BVx, BVy) is determined to be valid when all the following conditions are satisfied: (i) x + BVx >= 0, (ii) y + BVy >= floor(y / Hctu), and (iii) isRec(x + BVx + W - 1, y + BVy + H - 1) is true, where isRec(x, y) is true if sample (x, y) is reconstructed by an intra block copy mode, W denotes a width of the current video block, H denotes a height of the current video block, where floor(a) is the largest integer no greater than a.

26. The method of claim 25, wherein the block vector is a located in a first CTU in a CTU row.

27. The method of claim 3, wherein the prediction sample with the location (A, B) is determined according to the size of the buffer, the block vector (BVx, BVy), and the upper-left position (x, y).

28. The method of claim 27, wherein the prediction sample with the location (A, B) comprises a predication sample with a location computed according to ((X+BVx)%Wbuf, (Y+BVy)%Hbuf), wherein Wbuf and Hbuf denote a width of the buffer and a height of the buffer respectively.

29. The method of any one or more of claims 1-28, wherein the conversion is performed in an intra block copy mode.

30. The method of any of claims 1-29, wherein the conversion includes generating the bitstream representation from the current video block.

31. The method of any of claims 1-29, wherein the conversion includes generating pixel values of the current video block from the bitstream representation.

32. A video encoder apparatus comprising a processor configured to implement a method recited in any one or more of claims 1-29.

33. A video decoder apparatus comprising a processor configured to implement a method recited in any one or more of claims 1-29.

34. A computer readable medium having code stored thereon, the code embodying processor-executable instructions for implementing a method recited in any one or more of claims 1-29.

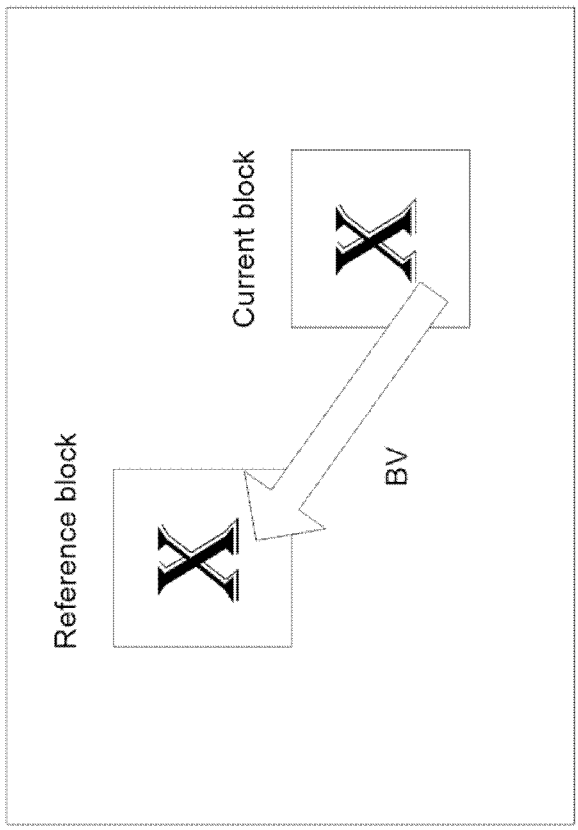

FIG. 1

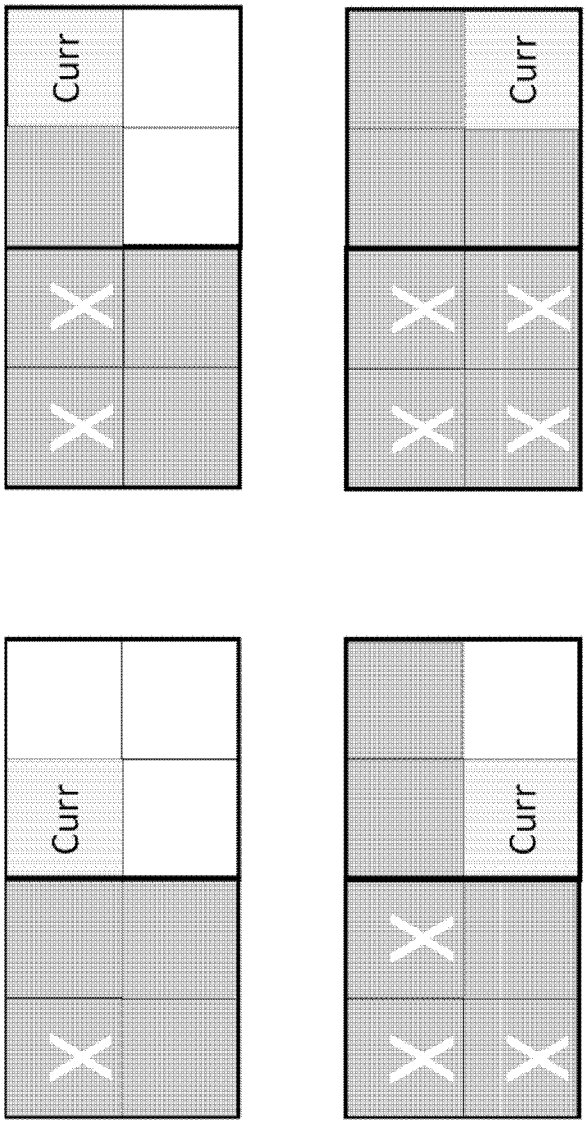

FIG. 2

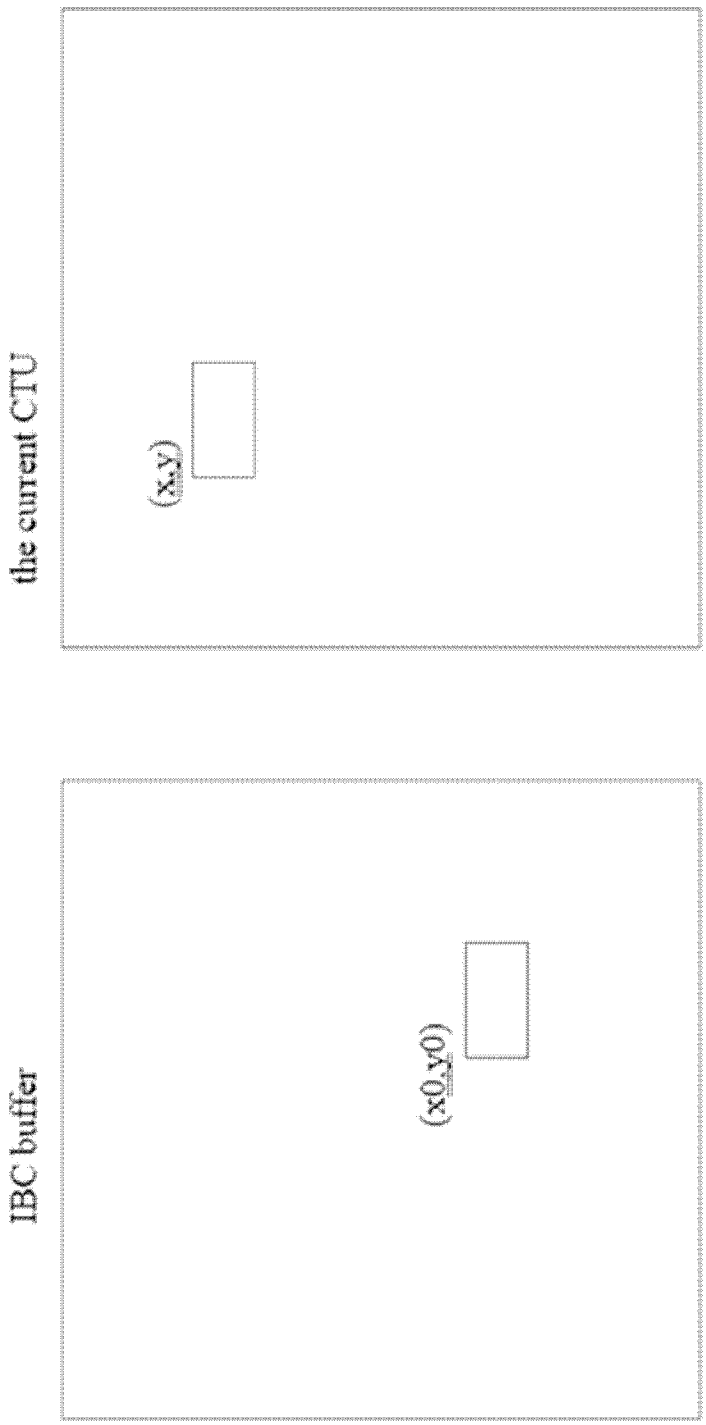

FIG. 3

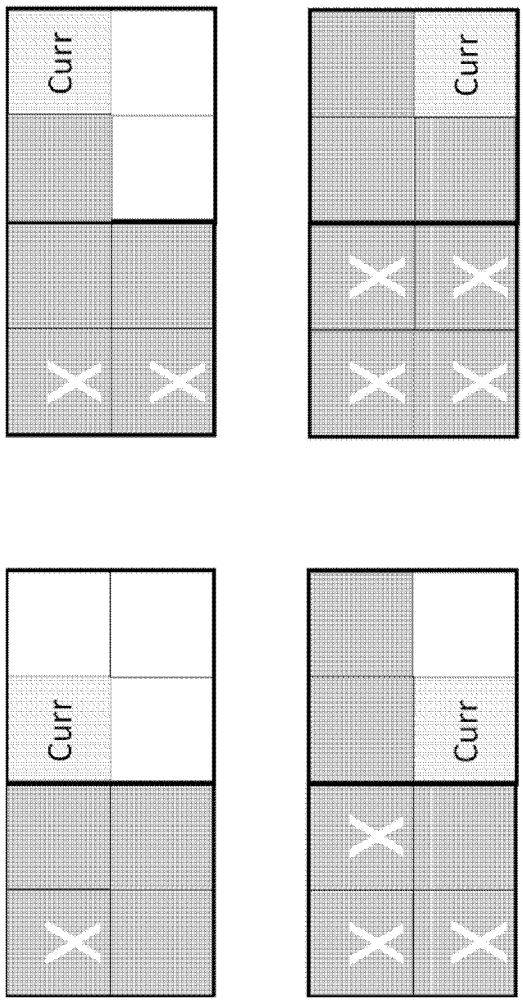

FIG. 4

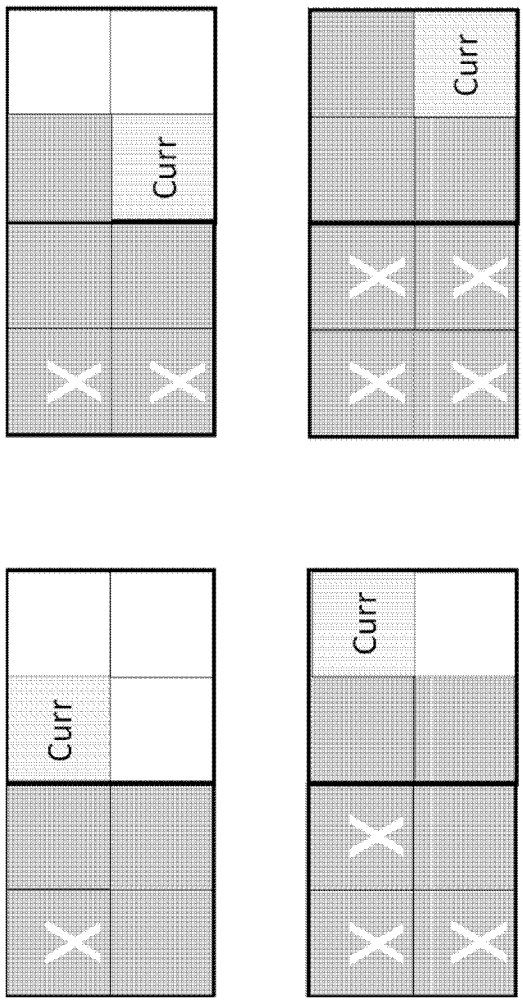

FIG. 5

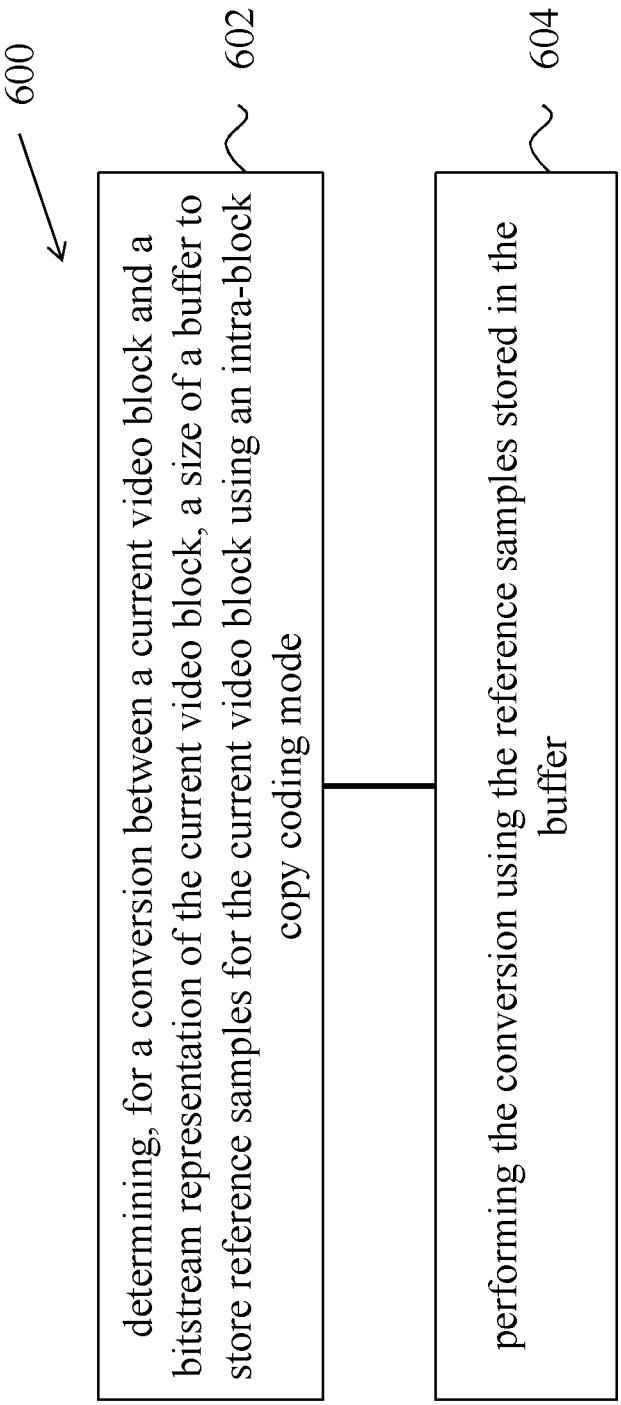
600
602
determining, for a conversion between a current video block and a bitstream representation of the current video block, a size of a buffer to store reference samples for the current video block using an intra-block copy coding mode
604
performing the conversion using the reference samples stored in the buffer

FIG. 6

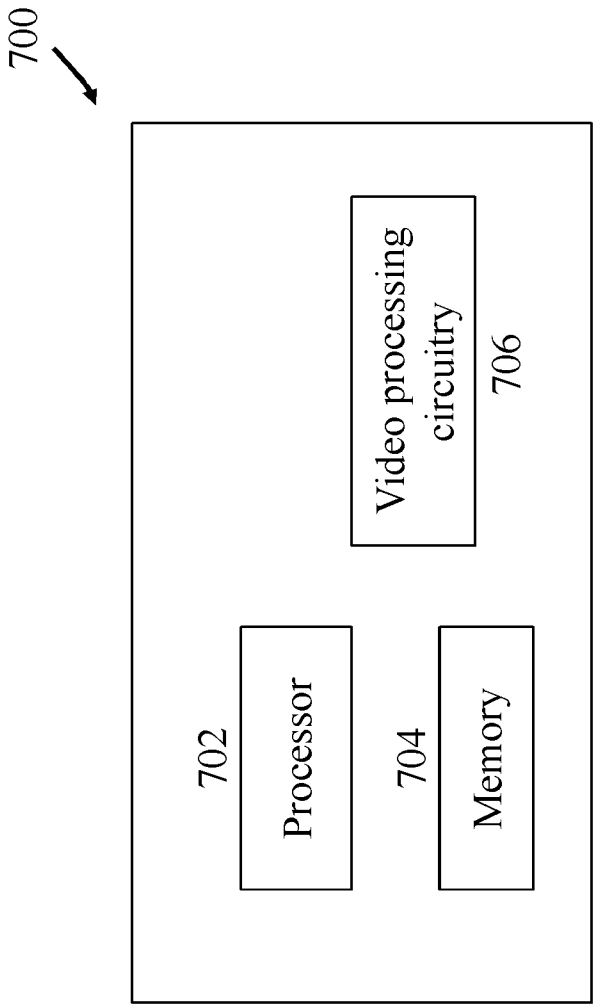
700
Video processing circuitry
706
Processor
702
Memory
704

FIG. 7

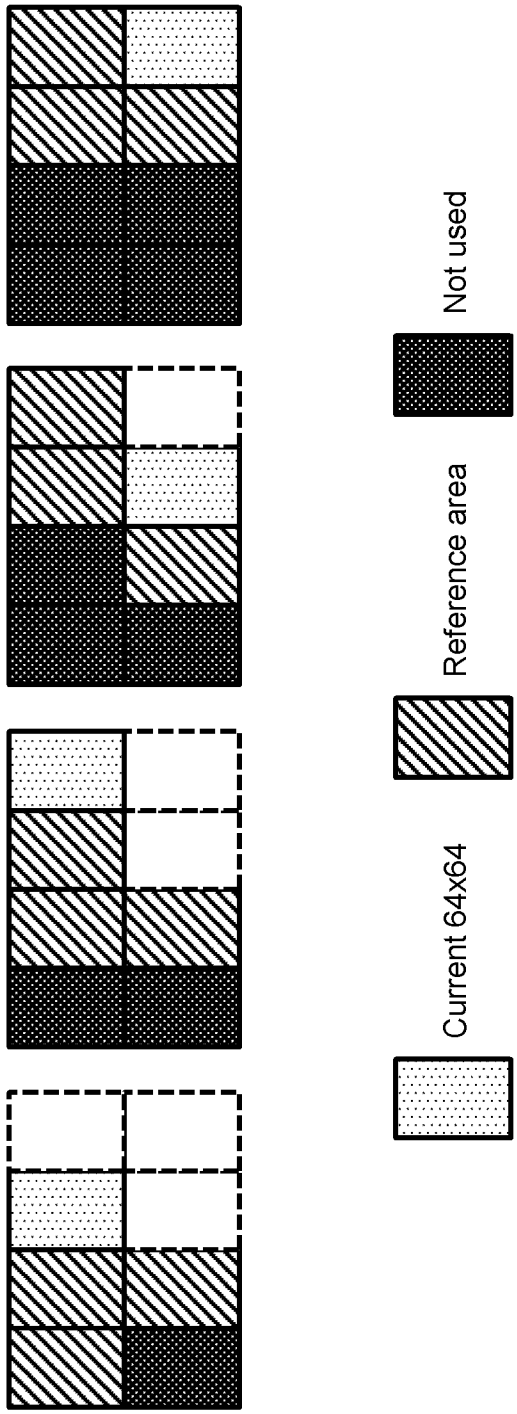

FIG. 8

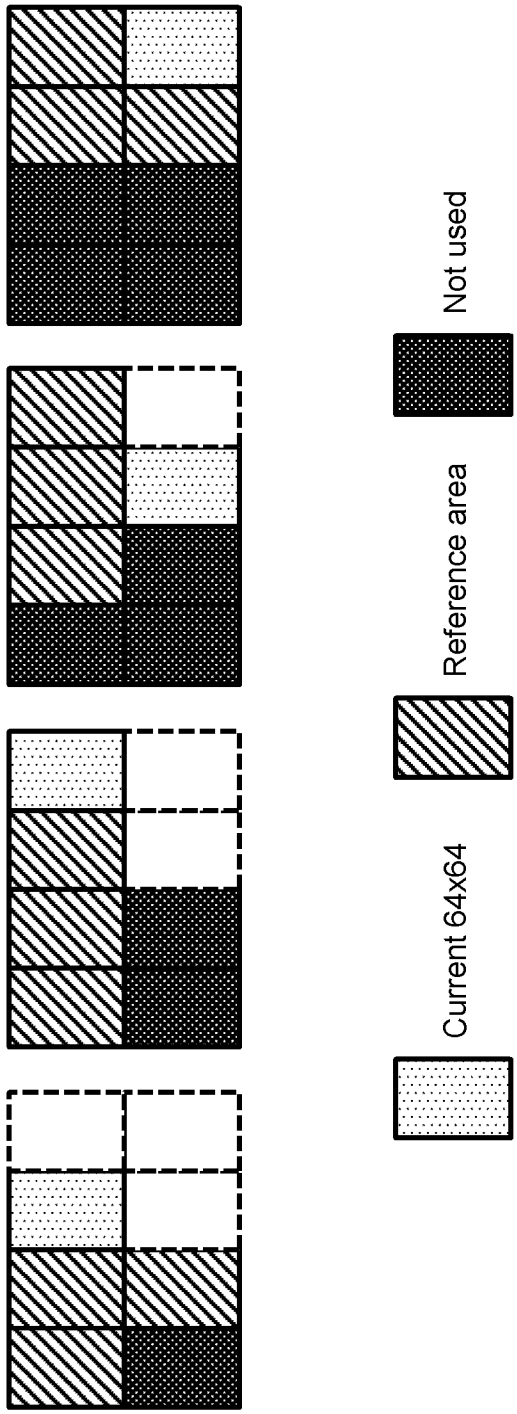

FIG. 9

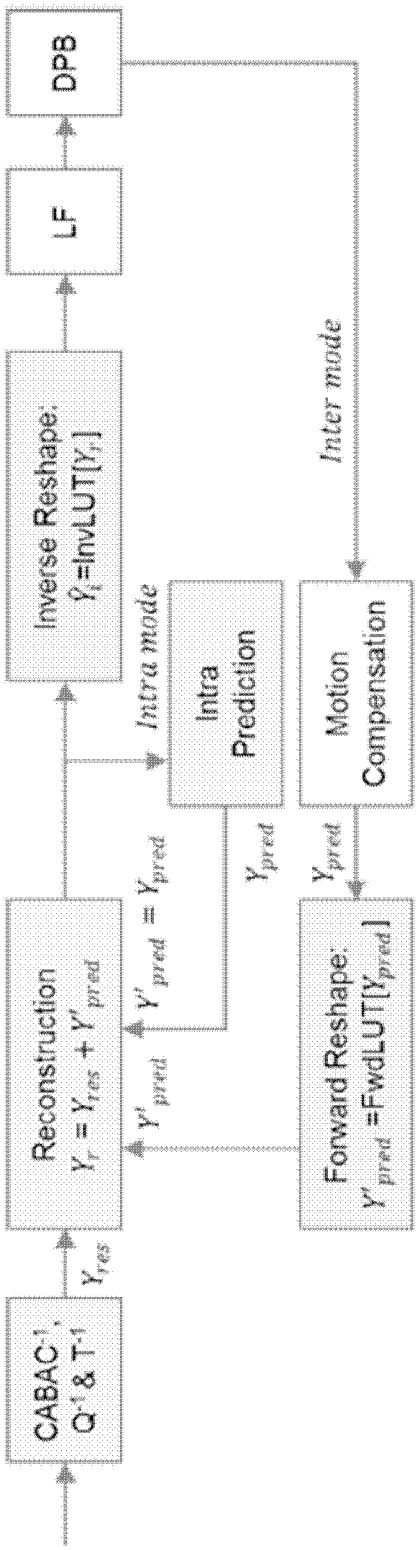
CABAC-1, Q-1 & T-1
Yres
Reconstruction
Yr = Yres + Y'pred
Y'pred
Y'pred = Ypred
Ypred
Inverse Reshape:
Yi=InvLUT[Yr]
LF
DPB
Intra mode
Intra Prediction
Inter mode
Motion Compensation
Ypred
Forward Reshape:
Y'pred =FwdLUT[Ypred]

FIG. 10

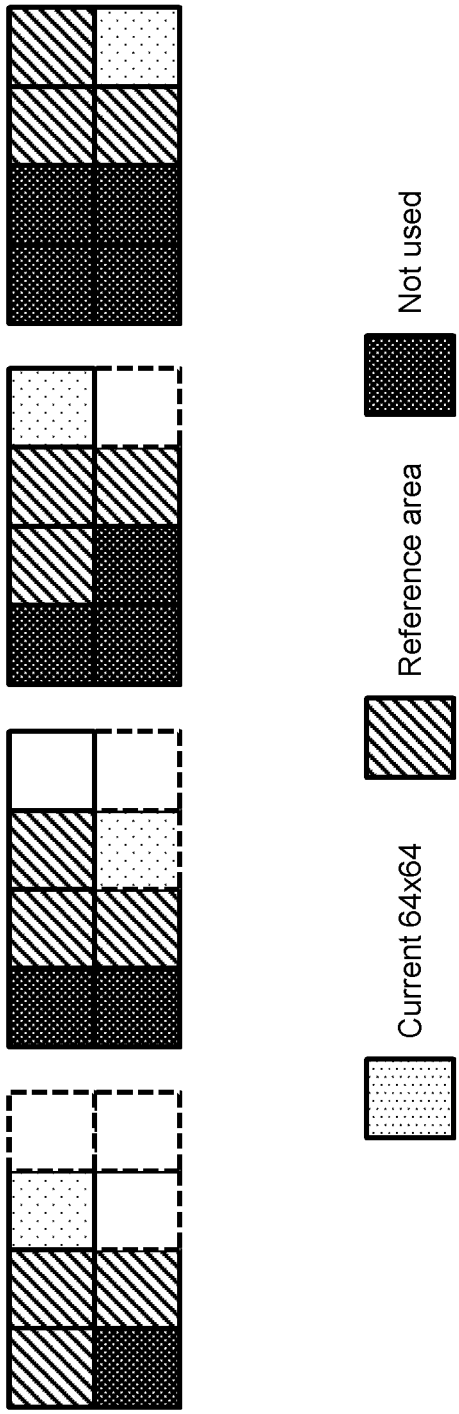

FIG. 11

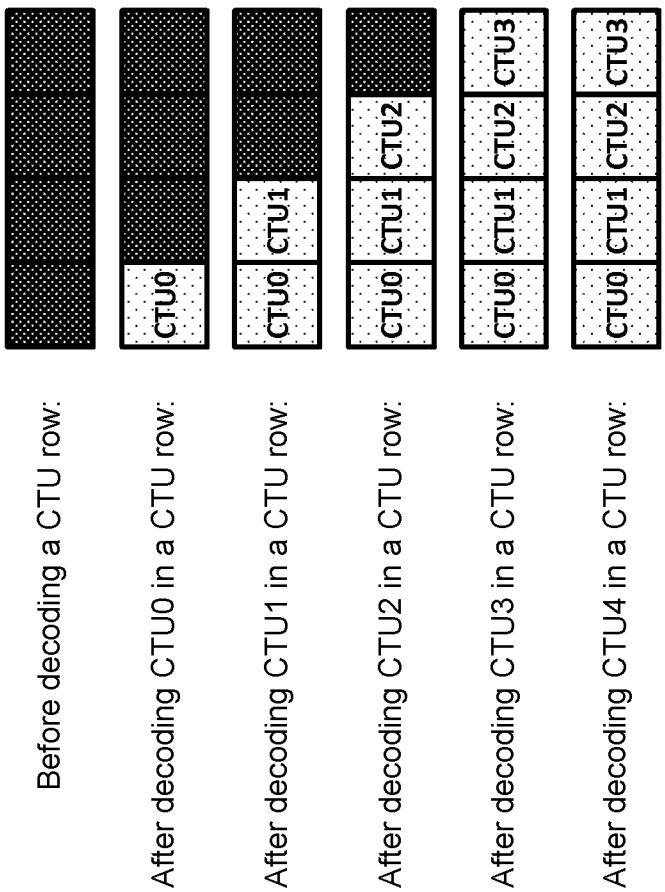
Before decoding a CTU row:
After decoding CTU0 in a CTU row:
CTU0
After decoding CTU1 in a CTU row:
CTU0
CTU1
After decoding CTU2 in a CTU row:
CTU0
CTU1
CTU2
After decoding CTU3 in a CTU row:
CTU0
CTU1
CTU2
CTU3
After decoding CTU4 in a CTU row:
CTU0
CTU1
CTU2
CTU3

FIG. 12

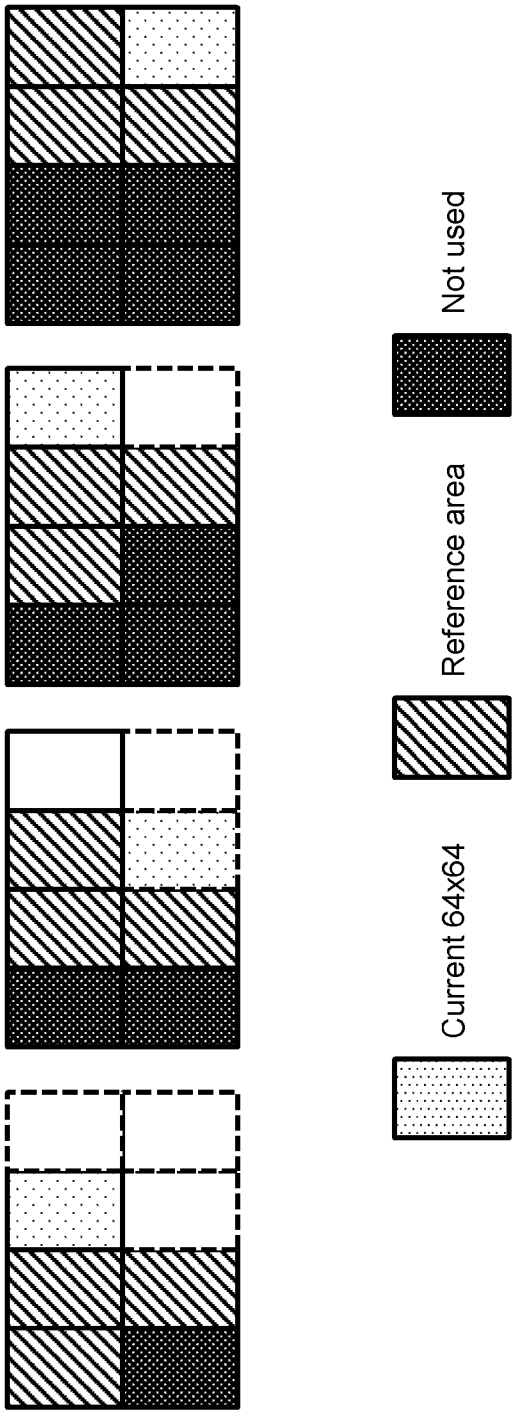

FIG. 13

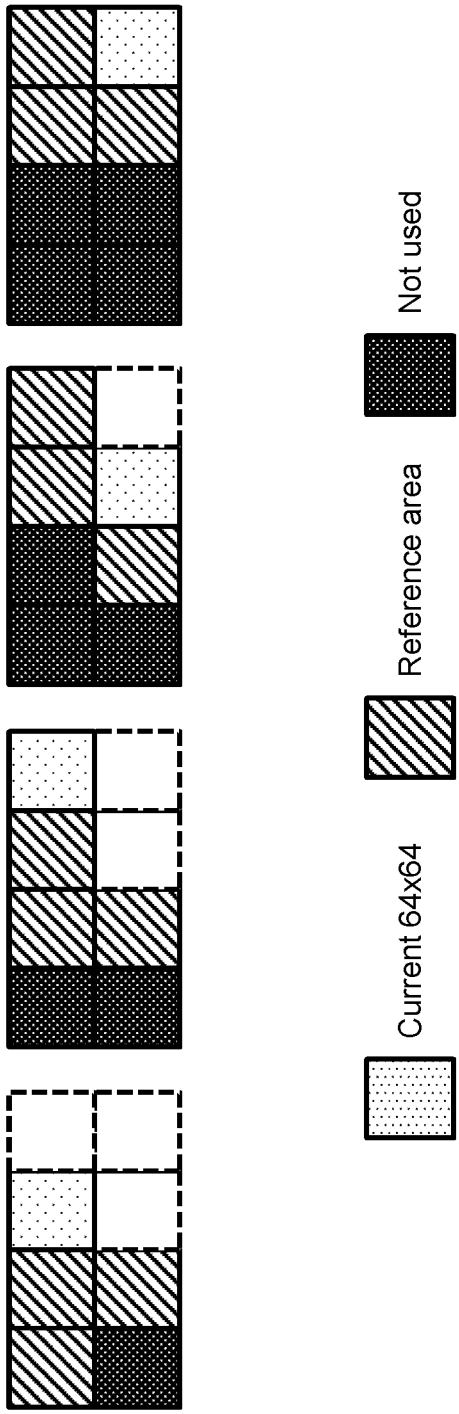

FIG. 14

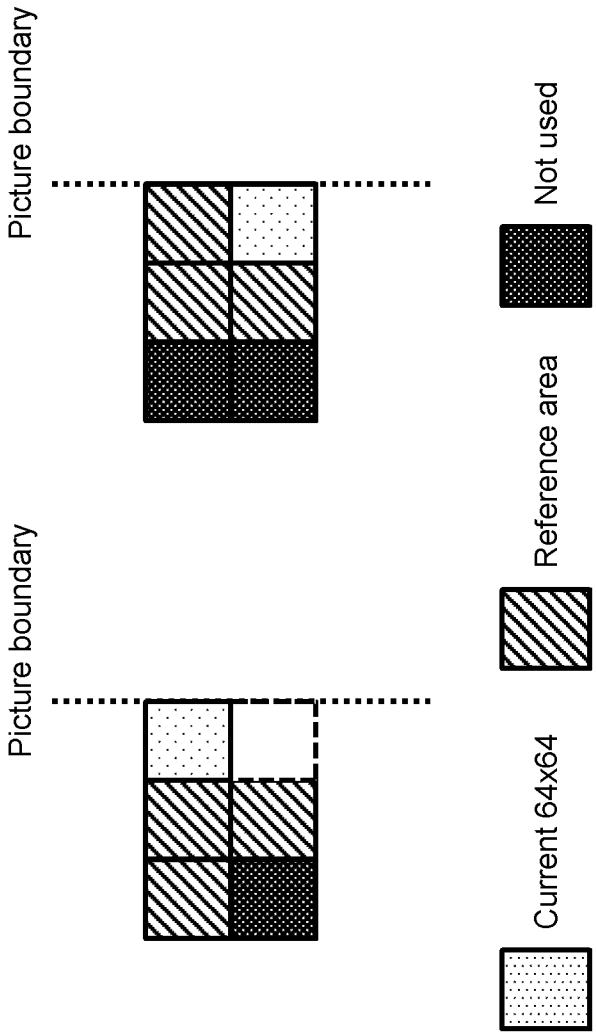
Picture boundary
Picture boundary
Not used
Reference area
Current 64x64

FIG. 15

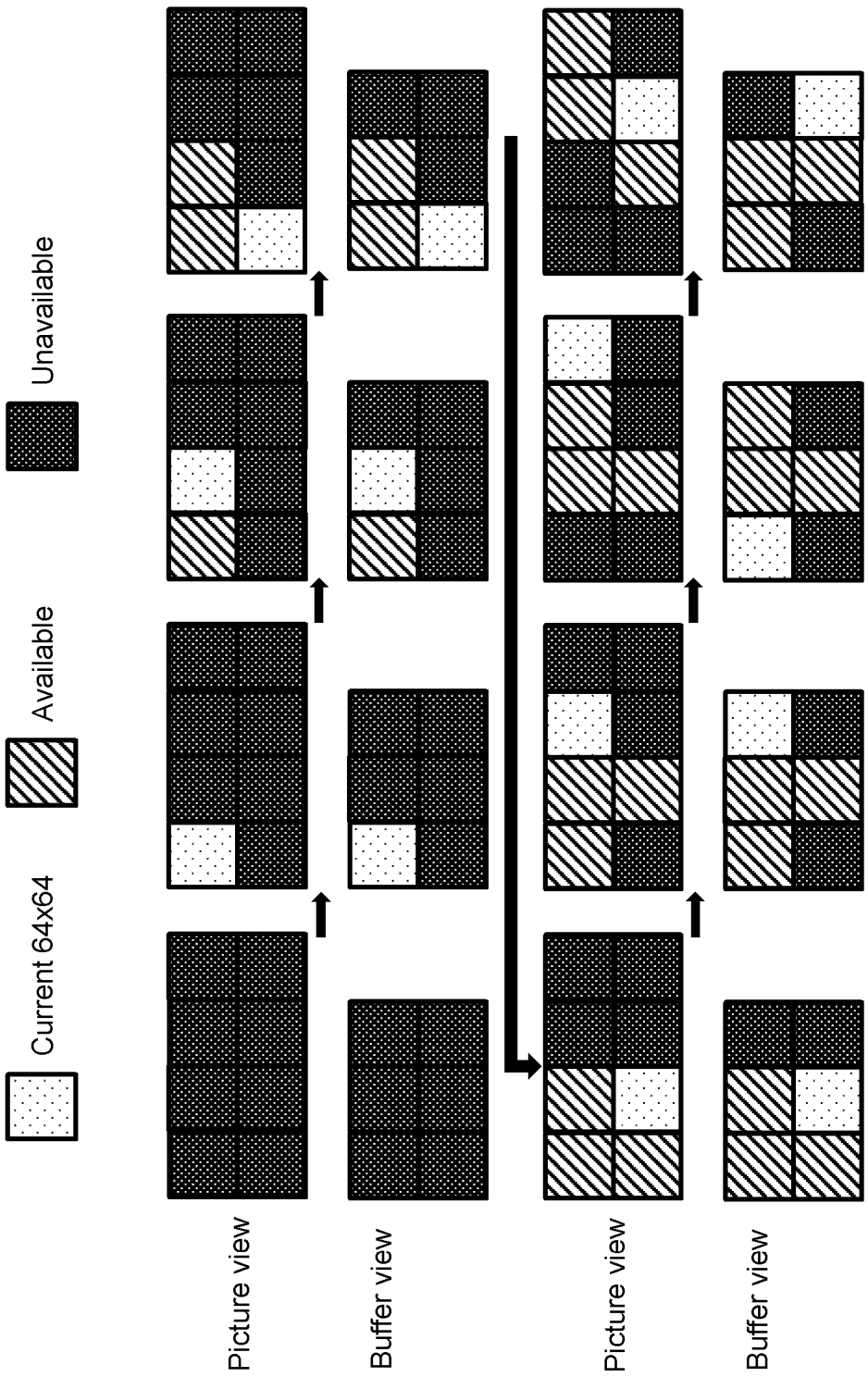
Current 64x64
Available
Unavailable
Picture view
Buffer view
Picture view
Buffer view

FIG. 16

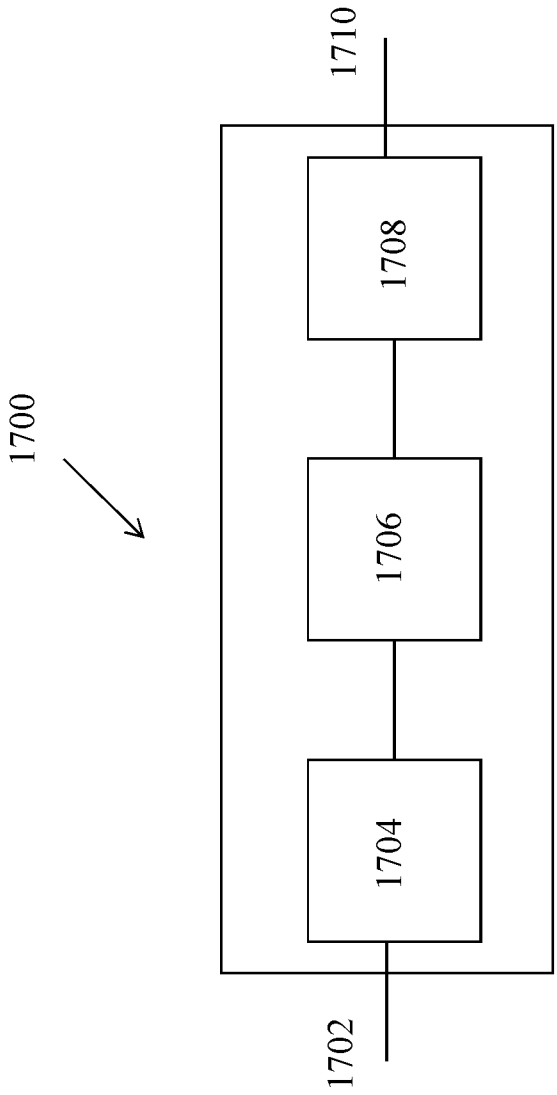

FIG. 17

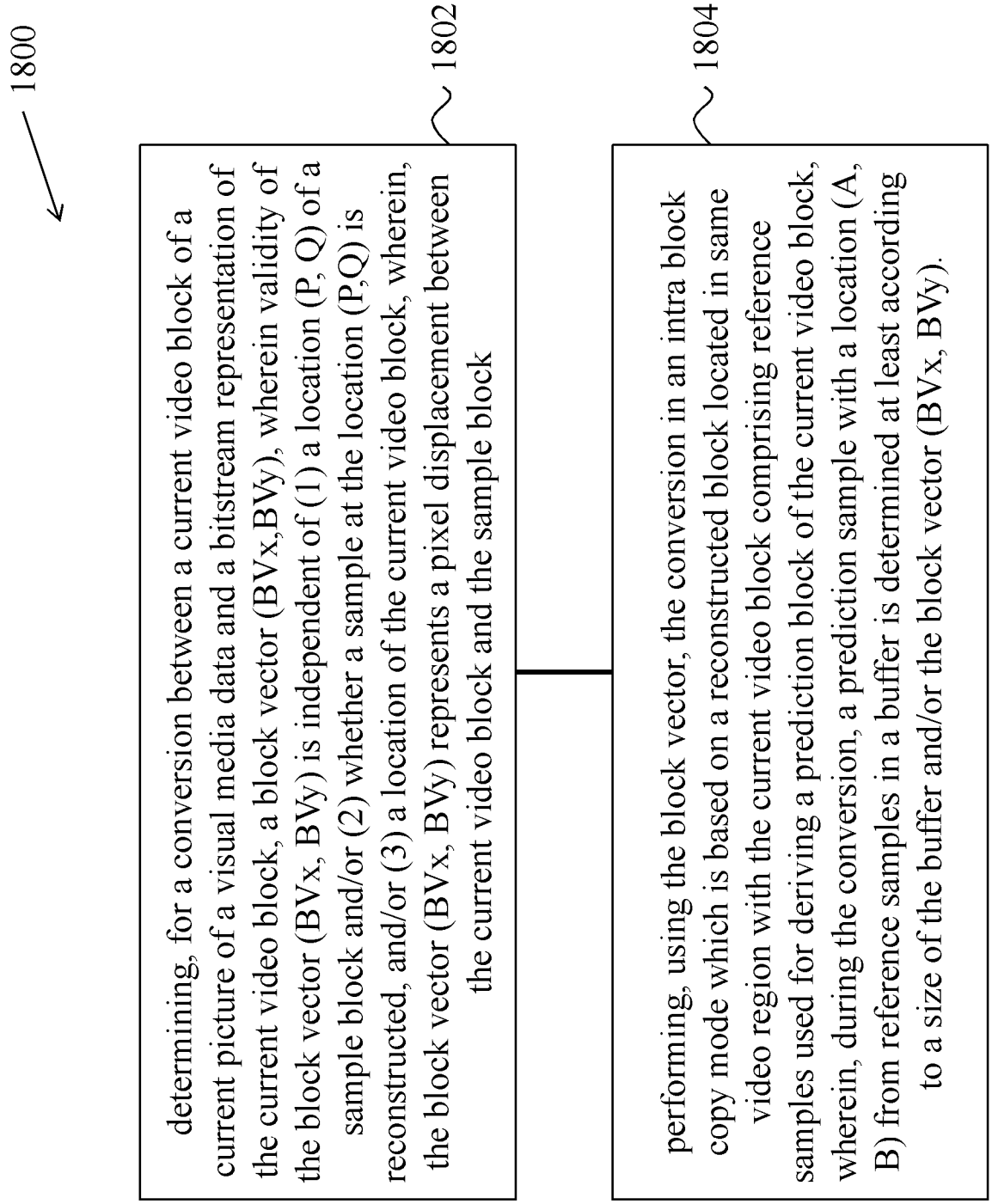
1800
determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the current video block, a block vector (BVx,BVy), wherein validity of the block vector (BVx, BVy) is independent of (1) a location (P, Q) of a sample block and/or (2) whether a sample at the location (P,Q) is reconstructed, and/or (3) a location of the current video block, wherein, the block vector (BVx, BVy) represents a pixel displacement between the current video block and the sample block
1802
performing, using the block vector, the conversion in an intra block copy mode which is based on a reconstructed block located in same video region with the current video block comprising reference samples used for deriving a prediction block of the current video block, wherein, during the conversion, a prediction sample with a location (A, B) from reference samples in a buffer is determined at least according to a size of the buffer and/or the block vector (BVx, BVy).
1804

FIG. 18

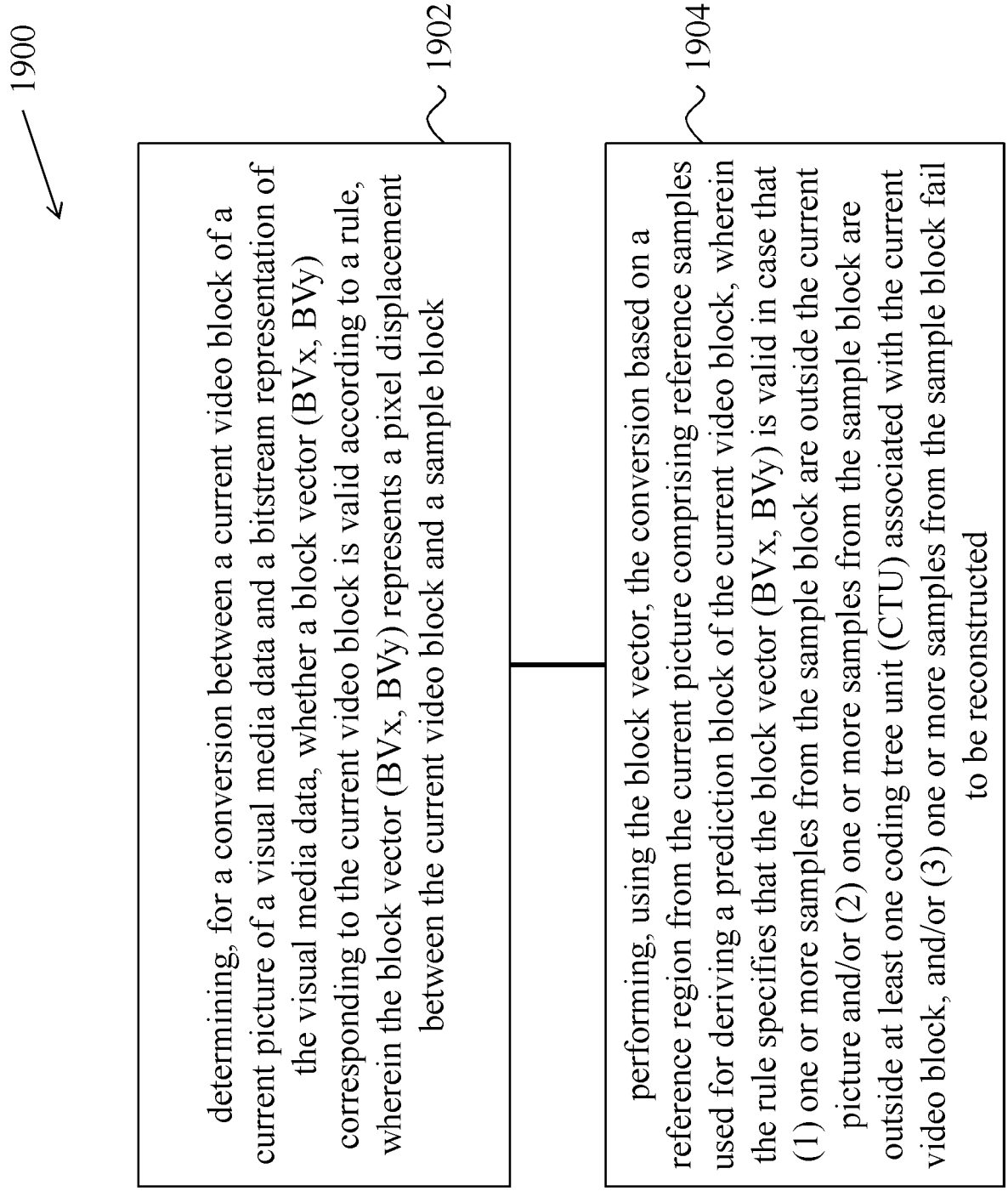
1900
determining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, whether a block vector (BVx, BVy) corresponding to the current video block is valid according to a rule, wherein the block vector (BVx, BVy) represents a pixel displacement between the current video block and a sample block
1902
performing, using the block vector, the conversion based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, wherein the rule specifies that the block vector (BVx, BVy) is valid in case that (1) one or more samples from the sample block are outside the current picture and/or (2) one or more samples from the sample block are outside at least one coding tree unit (CTU) associated with the current video block, and/or (3) one or more samples from the sample block fail to be reconstructed
1904

FIG. 19

2000
performing a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, wherein, the conversion is based on a reference region from the current picture comprising reference samples used for deriving a prediction block of the current video block, and wherein a virtual buffer of a defined size is used for tracking availability of the reference samples for deriving the prediction block
2002

FIG. 20

2100
maintaining, for a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data, a buffer comprising reference samples from the current picture for a derivation of a prediction block of the current video block, wherein one or more reference samples in the buffer that are marked unavailable for the derivation have values outside of a pixel value range
2102

FIG. 21

2200
performing a conversion between a current video block of a current picture of a visual media data and a bitstream representation of the visual media data using a buffer comprising reference samples from the current picture for derivation of a prediction block of the current video block, wherein the conversion is based according to rule which specifies that, for the bitstream representation to conform the rule, a reference sample in the buffer is to satisfy a bitstream conformance constraint
2202

FIG. 22

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2020/098471

A. CLASSIFICATION OF SUBJECT MATTER

H04N 19/103(2014.01)i; H04N 19/159(2014.01)i; H04N 19/176(2014.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

H04N

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

CNABS, CNTXT, CNKI, EPTXT, USTXT, WOTXT, VEN, JVET, JCTVC, IEEE: intraBC, IBC, Intra block copy, block vector, BV, search, constraint, restrain, Screen Content Coding, SCC

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	CN 105659602 A (MICROSOFT CORP) 08 June 2016 (2016-06-08) description, paragraphs [0112]-[0147] and figures 7-11B	1-4, 19-23, 27-34
Y	CN 105659602 A (MICROSOFT CORP) 08 June 2016 (2016-06-08) description, paragraphs [0112]-[0147] and figures 7-11B	1-34
X	CN 105765974 A (MICROSOFT CORP) 13 July 2016 (2016-07-13) description, paragraphs [0108]-[0143] and figures 7-11B	1-4, 19-23, 27-34
Y	CN 105765974 A (MICROSOFT CORP) 13 July 2016 (2016-07-13) description, paragraphs [0108]-[0143] and figures 7-11B	1-34
Y	Joel Sole et al. "HEVC Screen Content Coding Core Experiment 1 (SCCE1): Intra Block Copying Extensions" *Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 17th Meeting: Valencia, ES,* No. JCTVC-Q1121, 18 April 2014 (2014-04-18), section 3	1-34
Y	Joel Sole et al. "SCCE1: Test 1.1 – Intra block copy with different search areas" *Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11 18th Meeting: Sapporo, JP,* No. JCTVC-R0184, 02 July 2014 (2014-07-02), section 1	1-34

☑ Further documents are listed in the continuation of Box C. ☑ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
02 September 2020	**10 October 2020**
Name and mailing address of the ISA/CN **National Intellectual Property Administration, PRC** **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088** **China**	Authorized officer **QIN,Juxiu**
Facsimile No. **(86-10)62019451**	Telephone No. **86-(010)-62412043**

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2020/098471

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	CN 105474645 A (QUALCOMM INC) 06 April 2016 (2016-04-06) the whole document	1-34
A	US 2014301465 A1 (TEXAS INSTRUMENTS INC) 09 October 2014 (2014-10-09) the whole document	1-34

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2020/098471

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
CN 105659602 A	08 June 2016	WO 2015054813 A1	23 April 2015
		EP 3058736 B1	27 February 2019
		CN 105659602 B	08 October 2019
		US 2016241858 A1	18 August 2016
		EP 3058736 A4	31 May 2017
		EP 3058736 A1	24 August 2016
CN 105765974 A	13 July 2016	US 2016241868 A1	18 August 2016
		JP 2016539542 A	15 December 2016
		RU 2016114182 A	18 October 2017
		US 10582213 B2	03 March 2020
		BR 112016007151 A2	12 September 2017
		US 2020177910 A1	04 June 2020
		EP 3058739 A4	16 August 2017
		MX 2016004705 A	18 July 2016
		JP 6359101 B2	18 July 2018
		EP 3058739 A1	24 August 2016
		CA 2924763 A1	23 April 2015
		AU 2013403224 A1	31 March 2016
		CA 2928495 A1	23 April 2015
		RU 2654129 C2	16 May 2018
		CN 105765974 B	02 July 2019
		KR 20160072181 A	22 June 2016
		WO 2015054811 A1	23 April 2015
		AU 2013403224 B2	18 October 2018
		EP 3058739 B1	07 August 2019
CN 105474645 A	06 April 2016	WO 2015031253 A1	05 March 2015
		US 10313682 B2	04 June 2019
		KR 20160047486 A	02 May 2016
		US 2015055703 A1	26 February 2015
		CN 105474645 B	28 May 2019
		JP 2016534649 A	04 November 2016
		EP 3039870 A1	06 July 2016
US 2014301465 A1	09 October 2014	None	