

QUANTIZATION PARAMETER DERIVATION FOR PALETTE MODE

CROSS-REFERENCE TO RELATED APPLICATIONS

[001] Under the applicable patent law and/or rules pursuant to the Paris Convention, this application is made to timely claim the priority to and benefits of International Patent Application Nos. PCT/CN2019/106700 filed on September 19, 2019 and PCT/CN2019/108736 filed on September 27, 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

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

BACKGROUND

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

SUMMARY

[004] The disclosed techniques may be used by video or image decoder or encoder embodiments for quantization parameter derivation in palette mode coding and decoding.

[005] In an example aspect, a method of video processing is disclosed. The method includes determining, for a conversion between a current block of a video and a bitstream representation of the video, that the current block is coded using a palette mode and an escape symbol value is signaled in the bitstream representation, and performing, based on the determining, the conversion, wherein a clipped quantization parameter for the current block is used in the conversion, wherein the palette mode coding tool represents the current video block using a palette of representative color values, wherein the escape value is used for a sample of the current video block coded without using the representative color values, and wherein the clipped quantization parameter used for a chroma component of the video is derived based on a quantization parameter subsequent to a mapping operation of a quantization process or a dequantization process.

[0006] In another example aspect, a method of video processing is disclosed. The method includes determining, for a conversion between a current block of a video and a bitstream representation of the video, that the current block is coded using a palette mode and an escape symbol value is signaled in the bitstream representation, and performing, based on the determining, the conversion, wherein a clipped quantization parameter for the current block is used in the conversion, wherein the palette mode coding tool represents the current video block using a palette of representative color values, wherein the escape value is used for a sample of the current video block coded without using the representative color values, and wherein the clipped quantization parameter used for a chroma component of the video is derived based on a quantization parameter prior to a mapping operation of a quantization process or a dequantization process.

[0007] In yet another example aspect, the above-described method may be implemented by a video encoder apparatus that comprises a processor.

[0008] In yet another example aspect, the above-described method may be implemented by a video decoder apparatus that comprises a processor.

[0009] In yet another example aspect, these methods may be embodied in the form of processor-executable instructions and stored on a computer-readable program medium.

[0010] These, and other, aspects are further described in the present document.

BRIEF DESCRIPTION OF THE DRAWINGS

[0011] FIG. 1 shows an example of a block coded in palette mode.

[0012] FIG. 2 shows an example of use of palette predictor to signal palette entries.

[0013] FIG. 3 shows examples of horizontal and vertical traverse scans.

[0014] FIG. 4 shows an example coding of palette indices.

[0015] FIG. 5A and 5B show examples of a smallest chroma intra prediction unit (SCIPU).

[0016] FIG. 6 shows an example of repeated palette entries in the local dual tree case.

[0017] FIG. 7 shows an example of left and above blocks in the process of context derivation.

[0018] FIG. 8 is a block diagram of an example of a hardware platform used for implementing techniques described in the present document.

[0019] FIG. 9 is a block diagram of an example video processing system in which disclosed techniques may be implemented.

[0020] FIG. 10 is a block diagram that illustrates a video coding system in accordance with some embodiments of the present disclosure.

[0021] FIG. 11 is a block diagram that illustrates an encoder in accordance with some embodiments of the present disclosure.

[0022] FIG. 12 is a block diagram that illustrates a decoder in accordance with some embodiments of the present disclosure.

[0023] FIG. 13 shows a flowchart of an example method of video processing.

[0024] FIG. 14 shows a flowchart of another example method of video processing.

DETAILED DESCRIPTION

[0025] The present document provides various techniques that can be used by a decoder of image or video bitstreams to improve the quality of decompressed or decoded digital video or images. For brevity, the term “video” is used herein to include both a sequence of pictures (traditionally called video) and individual images. Furthermore, a video encoder may also implement these techniques during the process of encoding in order to reconstruct decoded frames used for further encoding.

[0026] Section headings are used in the present document for ease of understanding and do not limit the embodiments and techniques to the corresponding sections. As such, embodiments from one section can be combined with embodiments from other sections.

1. Summary

[0027] This document is related to video coding technologies. Specifically, it is related to index and escape symbols coding in palette coding. It may be applied to the existing video coding standard like HEVC, or the standard (Versatile Video Coding) to be finalized. It may be also applicable to future video coding standards or video codec.

2. Background

[0028] 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.

[0029] The latest version of VVC draft, i.e., Versatile Video Coding (Draft 6) could be found at: http://phenix.it-sudparis.eu/jvet/doc_end_user/documents/15_Gothenburg/wg11/JVET-O2001-v14.zip

[0030] The latest reference software of VVC, named VTM, could be found at:

[0031] https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/tags/VTM-5.0

2.1 Palette Mode in HEVC Screen Content Coding extensions (HEVC-SCC)

2.1.1 Concept of Palette Mode

[0032] The basic idea behind a palette mode is that the pixels in the CU are represented by a small set of representative colour values. This set is referred to as the palette. And it is also possible to indicate a sample that is outside the palette by signalling an escape symbol followed by (possibly quantized) component values. This kind of pixel is called escape pixel. The palette mode is illustrated in FIG. 1. As depicted in FIG. 1, for each pixel with three color components (luma, and two chroma components), an index to the palette is founded, and the block could be reconstructed based on the founded values in the palette.

2.1.2 Coding of the palette entries

[0033] For coding of the palette entries, a palette predictor is maintained. The maximum size of the palette as well as the palette predictor is signalled in the SPS. In HEVC-SCC, a `palette_predictor_initializer_present_flag` is introduced in the PPS. When this flag is 1, entries for initializing the palette predictor are signalled in the bitstream. The palette predictor is initialized at the beginning of each CTU row, each slice and each tile. Depending on the value of the `palette_predictor_initializer_present_flag`, the palette predictor is reset to 0 or initialized using the palette predictor initializer entries signaled in the PPS. In HEVC-SCC, a palette predictor initializer of size 0 was enabled to allow explicit disabling of the palette predictor initialization at the PPS level.

[0034] For each entry in the palette predictor, a reuse flag is signaled to indicate whether it is part of the current palette. This is illustrated in FIG. 2. The reuse flags are sent using run-length coding of zeros. After this, the number of new palette entries are signaled using Exponential Golomb (EG) code of order 0, i.e., EG-0. Finally, the component values for the new palette entries are signaled.

2.1.3 Coding of palette indices

[0035] The palette indices are coded using horizontal and vertical traverse scans as shown in FIG. 3. The scan order is explicitly signaled in the bitstream using the `palette_transpose_flag`. For the rest of the subsection it is assumed that the scan is horizontal.

[0036] The palette indices are coded using two palette sample modes: 'COPY_LEFT' and 'COPY_ABOVE'. In the 'COPY_LEFT' mode, the palette index is assigned to a decoded index. In the 'COPY_ABOVE' mode, the palette index of the sample in the row above is copied. For both 'COPY_LEFT' and 'COPY_ABOVE' modes, a run value is signaled which specifies the number of subsequent samples that are also coded using the same mode.

[0037] In the palette mode, the value of an index for the escape symbol is the number of palette entries. And, when escape symbol is part of the run in 'COPY_LEFT' or 'COPY_ABOVE' mode, the escape component values are signaled for each escape symbol. The coding of palette indices is illustrated in FIG. 4.

[0038] This syntax order is accomplished as follows. First the number of index values for the CU is signaled. This is followed by signaling of the actual index values for the entire CU using truncated binary coding. Both the number of indices as well as the index values are coded in bypass mode. This groups the index-related bypass bins together. Then the palette sample mode (if necessary) and run are signaled in an interleaved manner. Finally, the component escape values corresponding to the escape symbols for the entire CU are grouped together and coded in bypass mode. The binarization of escape symbols is EG coding with 3rd order, i.e., EG-3.

[0039] An additional syntax element, `last_run_type_flag`, is signaled after signaling the index values. This syntax element, in conjunction with the number of indices, eliminates the need to signal the run value corresponding to the last run in the block.

[0040] In HEVC-SCC, the palette mode is also enabled for 4:2:2, 4:2:0, and monochrome chroma formats. The signaling of the palette entries and palette indices is almost identical for all the chroma formats. In case of non-monochrome formats, each palette entry consists of 3

components. For the monochrome format, each palette entry consists of a single component. For subsampled chroma directions, the chroma samples are associated with luma sample indices that are divisible by 2. After reconstructing the palette indices for the CU, if a sample has only a single component associated with it, only the first component of the palette entry is used. The only difference in signaling is for the escape component values. For each escape symbol, the number of escape component values signaled may be different depending on the number of components associated with that symbol.

[0041] In addition, there is an index adjustment process in the palette index coding. When signaling a palette index, the left neighboring index or the above neighboring index should be different from the current index. Therefore, the range of the current palette index could be reduced by 1 by removing one possibility. After that, the index is signaled with truncated binary (TB) binarization.

[0042] The texts related to this part is shown as follows, where the CurrPaletteIndex is the current palette index and the adjustedRefPaletteIndex is the prediction index.

[0043] The variable PaletteIndexMap[xC][yC] specifies a palette index, which is an index to the array represented by CurrentPaletteEntries. The array indices xC, yC specify the location (xC, yC) of the sample relative to the top-left luma sample of the picture. The value of PaletteIndexMap[xC][yC] shall be in the range of 0 to MaxPaletteIndex, inclusive.

The variable adjustedRefPaletteIndex is derived as follows:

```

adjustedRefPaletteIndex = MaxPaletteIndex + 1
if( PaletteScanPos > 0 ) {
    xcPrev =
    x0 + TraverseScanOrder[ log2CbWidth ][ log2bHeight ][ PaletteScanPos - 1 ][ 0 ]
    ycPrev =
    y0 + TraverseScanOrder[ log2CbWidth ][ log2bHeight ][ PaletteScanPos - 1 ][ 1 ]
    if( CopyAboveIndicesFlag[ xcPrev ][ ycPrev ] == 0 ) {
        adjustedRefPaletteIndex = PaletteIndexMap[ xcPrev ][ ycPrev ] {      (7-157)
    }
    else {
        if( !palette_transpose_flag )
            adjustedRefPaletteIndex = PaletteIndexMap[ xC ][ yC - 1 ]
    }
}

```

```

else
    adjustedRefPaletteIndex = PaletteIndexMap[ xC - 1 ][ yC ]
}
}

```

[0044] When CopyAboveIndicesFlag[xC][yC] is equal to 0, the variable CurrPaletteIndex is derived as follows:

```

if( CurrPaletteIndex >= adjustedRefPaletteIndex )
    CurrPaletteIndex++

```

[0045] In addition, the run length elements in the palette mode are context coded. The related context derivation process described in JVET-O2011-vE is shown as follows.

Derivation process of ctxInc for the syntax element palette_run_prefix

Inputs to this process are the bin index binIdx and the syntax elements copy_above_palette_indices_flag and palette_idx_idc.

Output of this process is the variable ctxInc.

The variable ctxInc is derived as follows:

- If copy_above_palette_indices_flag is equal to 0 and binIdx is equal to 0, ctxInc is derived as follows:

$$\text{ctxInc} = (\text{palette_idx_idc} < 1) ? 0 : ((\text{palette_idx_idc} < 3) ? 1 : 2) \quad (9-69)$$

- Otherwise, ctxInc is provided by Table 1:

Table 1 – Specification of ctxIdxMap[copy_above_palette_indices_flag][binIdx]

binIdx	0	1	2	3	4	> 4
copy_above_palette_indices_flag == 1	5	6	6	7	7	bypass
copy_above_palette_indices_flag == 0	0, 1, 2	3	3	4	4	bypass

2.2 Palette Mode in VVC

2.2.1 Palette in Dual tree

[0046] In VVC, the dual tree coding structure is used on coding the intra slices, so the luma component and two chroma components may have different palette and palette indices. In addition, the two chroma component shares same palette and palette indices.

2.2.2 Palette as a separate mode

[0047] In JVET-N0258 and current VTM, the prediction modes for a coding unit can be MODE_INTRA, MODE_INTER, MODE_IBC and MODE_PLT. The binarization of prediction modes is changed accordingly.

[0048] When IBC is turned off, on I tiles, the first one bin is employed to indicate whether the current prediction mode is MODE_PLT or not. While on P/B tiles, the first bin is employed to indicate whether the current prediction mode is MODE_INTRA or not. If not, one additional bin is employed to indicate the current prediction mode is MODE_PLT or MODE_INTER.

[0049] When IBC is turned on, on I tiles, the first bin is employed to indicate whether the current prediction mode is MODE_IBC or not. If not, the second bin is employed to indicate whether the current prediction mode is MODE_PLT or MODE_INTRA. While on P/B tiles, the first bin is employed to indicate whether the current prediction mode is MODE_INTRA or not. If it's an intra mode, the second bin is employed to indicate the current prediction mode is MODE_PLT or MODE_INTRA. If not, the second bin is employed to indicate the current prediction mode is MODE_IBC or MODE_INTER.

[0050] The related texts in JVET-O2001-vE are shown as follows.

Coding unit syntax

coding_unit(x0, y0, cbWidth, cbHeight, cqtDepth, treeType, modeType) {	Descriptor
chType = treeType == DUAL_TREE_CHROMA ? 1 : 0	
if(slice_type != I sps_ibc_enabled_flag sps_palette_enabled_flag) {	
if(treeType != DUAL_TREE_CHROMA && !((cbWidth == 4 && cbHeight == 4) modeType == MODE_TYPE_INTRA) && !sps_ibc_enabled_flag)	
cu_skip_flag [x0][y0]	ac(v)
if(cu_skip_flag[x0][y0] == 0 && slice_type != I && !(cbWidth == 4 && cbHeight == 4) && modeType == MODE_TYPE_ALL)	
pred_mode_flag	ac(v)
if(((slice_type == I && cu_skip_flag[x0][y0] == 0) (slice_type != I && (CuPredMode[chType][x0][y0] != MODE_INTRA (cbWidth == 4 && cbHeight == 4 && cu_skip_flag[x0][y0] == 0)))) && cbWidth <= 64 && cbHeight <= 64 && modeType != MODE_TYPE_INTER && sps_ibc_enabled_flag && treeType != DUAL_TREE_CHROMA)	
pred_mode_ibc_flag	ac(v)

<pre> if(((slice_type == I (cbWidth == 4 && cbHeight == 4) sps_ibc_enabled_flag) && CuPredMode[x0][y0] == MODE_INTRA) (slice_type != I && !(cbWidth == 4 && cbHeight == 4) && !sps_ibc_enabled_flag && CuPredMode[x0][y0] != MODE_INTRA)) && sps_palette_enabled_flag && cbWidth <= 64 && cbHeight <= 64 && cu_skip_flag[x0][y0] == 0 && modeType != MODE_TYPE_INTER) </pre>	
pred_mode_plt_flag	ae(v)
}	
...	
}	

2.2.3 Palette mode syntax

palette_coding(x0, y0, cbWidth, cbHeight, startComp, numComps) {	Descriptor
palettePredictionFinished = 0	
NumPredictedPaletteEntries = 0	
for(predictorEntryIdx = 0; predictorEntryIdx < PredictorPaletteSize[startComp] && !palettePredictionFinished && NumPredictedPaletteEntries[startComp] < palette_max_size; predictorEntryIdx++) {	
palette_predictor_run	ae(v)
if(palette_predictor_run != 1) {	
if(palette_predictor_run > 1)	
predictorEntryIdx += palette_predictor_run - 1	
PalettePredictorEntryReuseFlags[predictorEntryIdx] = 1	
NumPredictedPaletteEntries++	
} else	
palettePredictionFinished = 1	
}	
if(NumPredictedPaletteEntries < palette_max_size)	
num_signalled_palette_entries	ae(v)
for(cIdx = startComp; cIdx < (startComp + numComps); cIdx++)	
for(i = 0; i < num_signalled_palette_entries; i++)	
new_palette_entries[cIdx][i]	ae(v)
if(CurrentPaletteSize[startComp] > 0)	
palette_escape_val_present_flag	ae(v)
if(MaxPaletteIndex > 0) {	
num_palette_indices_minus1	ae(v)
adjust = 0	
for(i = 0; i <= num_palette_indices_minus1; i++) {	
if(MaxPaletteIndex - adjust > 0) {	
palette_idx_idc	ae(v)
PaletteIndexIdc[i] = palette_idx_idc	
}	
adjust = 1	
}	
copy_above_indices_for_final_run_flag	ae(v)
palette_transpose_flag	ae(v)
}	

if(treeType != DUAL_TREE_CHROMA && palette_escape_val_present_flag) {	
if(cu_qp_delta_enabled_flag && !IsCuQpDeltaCoded) {	
cu_qp_delta_abs	ae(v)
if(cu_qp_delta_abs)	
cu_qp_delta_sign_flag	ae(v)
}	
}	
if(treeType != DUAL_TREE_LUMA && palette_escape_val_present_flag) {	
if(cu_chroma_qp_offset_enabled_flag && !IsCuChromaQpOffsetCoded) {	
cu_chroma_qp_offset_flag	ae(v)
if(cu_chroma_qp_offset_flag)	
cu_chroma_qp_offset_idx	ae(v)
}	
}	
remainingNumIndices = num_palette_indices_minus1 + 1	
PaletteScanPos = 0	
log2CbWidth = Log2(cbWidth)	
log2CbHeight = Log2(cbHeight)	
while(PaletteScanPos < cbWidth*cbHeight) {	
xC = x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos][0]	
yC = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos][1]	
if(PaletteScanPos > 0) {	
xcPrev = x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos - 1][0]	
ycPrev = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos - 1][1]	
}	
PaletteRunMinus1 = cbWidth * cbHeight - PaletteScanPos - 1	
RunToEnd = 1	
CopyAboveIndicesFlag[xC][yC] = 0	
if(MaxPaletteIndex > 0)	
if(((!palette_transpose_flag && yC > 0) (palette_transpose_flag && xC > 0)) && CopyAboveIndicesFlag[xcPrev][ycPrev] == 0)	
if(remainingNumIndices > 0 && PaletteScanPos < cbWidth*cbHeight - 1) {	
copy_above_palette_indices_flag	ae(v)
CopyAboveIndicesFlag[xC][yC] = copy_above_palette_indices_flag	
} else {	
if(PaletteScanPos == cbWidth * cbHeight - 1 && remainingNumIndices > 0)	
CopyAboveIndicesFlag[xC][yC] = 0	
else	
CopyAboveIndicesFlag[xC][yC] = 1	
}	
if(CopyAboveIndicesFlag[xC][yC] == 0) {	
currNumIndices = num_palette_indices_minus1 + 1 - remainingNumIndices	
PaletteIndexMap[xC][yC] = PaletteIndexIdx[currNumIndices]	
}	
if(MaxPaletteIndex > 0) {	

if(CopyAboveIndicesFlag[xC][yC] == 0)	
remainingNumIndices -- = 1	
if(remainingNumIndices > 0 CopyAboveIndicesFlag[xC][yC] != copy_above_indices_for_final_run_flag) {	
PaletteMaxRunMinus1 = cbWidth * cbHeight - PaletteScanPos - 1 - remainingNumIndices - copy_above_indices_for_final_run_flag	
RunToEnd = 0	
if(PaletteMaxRunMinus1 > 0) {	
palette_run_prefix	ae(v)
if((palette_run_prefix > 1) && (PaletteMaxRunMinus1 != (1 << (palette_run_prefix - 1))))	
palette_run_suffix	ae(v)
}	
}	
}	
runPos = 0	
while (runPos <= PaletteRunMinus1) {	
xR = x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos][0]	
yR = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][PaletteScanPos][1]	
if(CopyAboveIndicesFlag[xC][yC] == 0) {	
CopyAboveIndicesFlag[xR][yR] = 0	
PaletteIndexMap[xR][yR] = PaletteIndexMap[xC][yC]	
} else {	
CopyAboveIndicesFlag[xR][yR] = 1	
if (!palette_transpose_flag)	
PaletteIndexMap[xR][yR] = PaletteIndexMap[xR][yR - 1]	
else	
PaletteIndexMap[xR][yR] = PaletteIndexMap[xR - 1][yR]	
}	
runPos++	
PaletteScanPos ++	
}	
}	
if(palette_escape_val_present_flag) {	
for(cIdx = startComp; cIdx < (startComp + numComps); cIdx++)	
for(sPos = 0; sPos < cbWidth * cbHeight; sPos++) {	
xC = x0 + TraverseScanOrder[log2CbWidth][log2CbHeight][sPos][0]	
yC = y0 + TraverseScanOrder[log2CbWidth][log2CbHeight][sPos][1]	
if(PaletteIndexMap[cIdx][xC][yC] == MaxPaletteIndex) {	
palette_escape_val	ae(v)
PaletteEscapeVal[cIdx][xC][yC] = palette_escape_val	
}	
}	
}	
}	

2.2.4 Palette mode semantics

In the following semantics, the array indices x_0, y_0 specify the location (x_0, y_0) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture. The array indices x_C, y_C specify the location (x_C, y_C) of the sample relative to the top-left luma sample of the picture. The array index $startComp$ specifies the first colour component of the current palette table. $startComp$ equal to 0 indicates the Y component; $startComp$ equal to 1 indicates the Cb component; $startComp$ equal to 2 indicates the Cr component. $numComps$ specifies the number of colour components in the current palette table.

The predictor palette consists of palette entries from previous coding units that are used to predict the entries in the current palette.

The variable $PredictorPaletteSize[startComp]$ specifies the size of the predictor palette for the first colour component of the current palette table $startComp$. $PredictorPaletteSize$ is derived as specified in clause 8.4.5.3.

The variable $PalettePredictorEntryReuseFlags[i]$ equal to 1 specifies that the i -th entry in the predictor palette is reused in the current palette. $PalettePredictorEntryReuseFlags[i]$ equal to 0 specifies that the i -th entry in the predictor palette is not an entry in the current palette. All elements of the array $PalettePredictorEntryReuseFlags[i]$ are initialized to 0.

palette_predictor_run is used to determine the number of zeros that precede a non-zero entry in the array $PalettePredictorEntryReuseFlags$.

It is a requirement of bitstream conformance that the value of **palette_predictor_run** shall be in the range of 0 to ($PredictorPaletteSize - predictorEntryIdx$), inclusive, where $predictorEntryIdx$ corresponds to the current position in the array $PalettePredictorEntryReuseFlags$. The variable $NumPredictedPaletteEntries$ specifies the number of entries in the current palette that are reused from the predictor palette. The value of $NumPredictedPaletteEntries$ shall be in the range of 0 to $palette_max_size$, inclusive.

num_signalled_palette_entries specifies the number of entries in the current palette that are explicitly signalled for the first colour component of the current palette table $startComp$.

When **num_signalled_palette_entries** is not present, it is inferred to be equal to 0.

The variable $CurrentPaletteSize[startComp]$ specifies the size of the current palette for the first colour component of the current palette table $startComp$ and is derived as follows:

CurrentPaletteSize[startComp] = NumPredictedPaletteEntries +
num_signalled_palette_entries (7-155)

The value of CurrentPaletteSize[startComp] shall be in the range of 0 to palette_max_size, inclusive.

new_palette_entries[cIdx][i] specifies the value for the i-th signalled palette entry for the colour component cIdx.

The variable PredictorPaletteEntries[cIdx][i] specifies the i-th element in the predictor palette for the colour component cIdx.

The variable CurrentPaletteEntries[cIdx][i] specifies the i-th element in the current palette for the colour component cIdx and is derived as follows:

```

numPredictedPaletteEntries = 0
for( i = 0; i < PredictorPaletteSize[ startComp ]; i++ )
    if( PalettePredictorEntryReuseFlags[ i ] ) {
        for( cIdx = startComp; cIdx < ( startComp + numComps ); cIdx++ )
            CurrentPaletteEntries[ cIdx ][ numPredictedPaletteEntries ] =
PredictorPaletteEntries[ cIdx ][ i ]
            numPredictedPaletteEntries++
        }
    }
for( cIdx = startComp; cIdx < (startComp + numComps); cIdx++) (7-156)
    for( i = 0; i < num_signalled_palette_entries[startComp]; i++ )
        CurrentPaletteEntries[ cIdx ][ numPredictedPaletteEntries + i ] =
new_palette_entries[ cIdx ][ i ]

```

palette_escape_val_present_flag equal to 1 specifies that the current coding unit contains at least one escape coded sample. **escape_val_present_flag** equal to 0 specifies that there are no escape coded samples in the current coding unit. When not present, the value of **palette_escape_val_present_flag** is inferred to be equal to 1.

The variable MaxPaletteIndex specifies the maximum possible value for a palette index for the current coding unit. The value of MaxPaletteIndex is set equal to

CurrentPaletteSize[startComp] - 1 + **palette_escape_val_present_flag**.

num_palette_indices_minus1 plus 1 is the number of palette indices explicitly signalled or inferred for the current block.

When `num_palette_indices_minus1` is not present, it is inferred to be equal to 0.

palette_idx_idc is an indication of an index to the palette table, `CurrentPaletteEntries`. The value of `palette_idx_idc` shall be in the range of 0 to `MaxPaletteIndex`, inclusive, for the first index in the block and in the range of 0 to (`MaxPaletteIndex - 1`), inclusive, for the remaining indices in the block.

When `palette_idx_idc` is not present, it is inferred to be equal to 0.

The variable `PaletteIndexIdc[ i ]` stores the *i*-th `palette_idx_idc` explicitly signalled or inferred.

All elements of the array `PaletteIndexIdc[ i ]` are initialized to 0.

copy_above_indices_for_final_run_flag equal to 1 specifies that the palette indices of the last positions in the coding unit are copied from the palette indices in the row above if horizontal traverse scan is used or the palette indices in the left column if vertical traverse scan is used.

`copy_above_indices_for_final_run_flag` equal to 0 specifies that the palette indices of the last positions in the coding unit are copied from `PaletteIndexIdc[ num_palette_indices_minus1 ]`.

When `copy_above_indices_for_final_run_flag` is not present, it is inferred to be equal to 0.

palette_transpose_flag equal to 1 specifies that vertical traverse scan is applied for scanning the indices for samples in the current coding unit. `palette_transpose_flag` equal to 0 specifies that horizontal traverse scan is applied for scanning the indices for samples in the current coding unit.

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

The array `TraverseScanOrder` specifies the scan order array for palette coding.

`TraverseScanOrder` is assigned the horizontal scan order `HorTravScanOrder` if

`palette_transpose_flag` is equal to 0 and `TraverseScanOrder` is assigned the vertical scan order

`VerTravScanOrder` if `palette_transpose_flag` is equal to 1.

copy_above_palette_indices_flag equal to 1 specifies that the palette index is equal to the palette index at the same location in the row above if horizontal traverse scan is used or the same location in the left column if vertical traverse scan is used. `copy_above_palette_indices_flag` equal to 0 specifies that an indication of the palette index of the sample is coded in the bitstream or inferred.

The variable `CopyAboveIndicesFlag[ xC ][ yC ]` equal to 1 specifies that the palette index is copied from the palette index in the row above (horizontal scan) or left column (vertical scan).

`CopyAboveIndicesFlag[ xC ][ yC ]` equal to 0 specifies that the palette index is explicitly coded in the bitstream or inferred. The array indices `xC`, `yC` specify the location (`xC`, `yC`) of the

sample relative to the top-left luma sample of the picture. The value of $\text{PaletteIndexMap}[xC][yC]$ shall be in the range of 0 to $(\text{MaxPaletteIndex} - 1)$, inclusive. The variable $\text{PaletteIndexMap}[xC][yC]$ specifies a palette index, which is an index to the array represented by $\text{CurrentPaletteEntries}$. The array indices xC , yC specify the location (xC, yC) of the sample relative to the top-left luma sample of the picture. The value of $\text{PaletteIndexMap}[xC][yC]$ shall be in the range of 0 to MaxPaletteIndex , inclusive. The variable $\text{adjustedRefPaletteIndex}$ is derived as follows:

```

adjustedRefPaletteIndex = MaxPaletteIndex + 1
if( PaletteScanPos > 0 ) {
    xcPrev =
    x0 + TraverseScanOrder[ log2CbWidth ][ log2bHeight ][ PaletteScanPos - 1 ][ 0 ]
    ycPrev =
    y0 + TraverseScanOrder[ log2CbWidth ][ log2bHeight ][ PaletteScanPos - 1 ][ 1 ]
    if( CopyAboveIndicesFlag[ xcPrev ][ ycPrev ] == 0 ) {
        adjustedRefPaletteIndex = PaletteIndexMap[ xcPrev ][ ycPrev ] { (7-157)
    }
    else {
        if( !palette_transpose_flag )
            adjustedRefPaletteIndex = PaletteIndexMap[ xC ][ yC - 1 ]
        else
            adjustedRefPaletteIndex = PaletteIndexMap[ xC - 1 ][ yC ]
    }
}

```

When $\text{CopyAboveIndicesFlag}[xC][yC]$ is equal to 0, the variable CurrPaletteIndex is derived as follows:

```

if( CurrPaletteIndex >= adjustedRefPaletteIndex )
    CurrPaletteIndex++    (7-158)

```

palette_run_prefix, when present, specifies the prefix part in the binarization of PaletteRunMinus1 .

palette_run_suffix is used in the derivation of the variable PaletteRunMinus1 . When not present, the value of $\text{palette_run_suffix}$ is inferred to be equal to 0.

When RunToEnd is equal to 0, the variable PaletteRunMinus1 is derived as follows:

- If PaletteMaxRunMinus1 is equal to 0, PaletteRunMinus1 is set equal to 0.
- Otherwise (PaletteMaxRunMinus1 is greater than 0) the following applies:
 - If palette_run_prefix is less than 2, the following applies:

$$\text{PaletteRunMinus1} = \text{palette_run_prefix} \quad (7-159)$$

- Otherwise (palette_run_prefix is greater than or equal to 2), the following applies:

$$\text{PrefixOffset} = 1 \ll (\text{palette_run_prefix} - 1)$$

$$\text{PaletteRunMinus1} = \text{PrefixOffset} + \text{palette_run_suffix} \quad (7-160)$$

The variable PaletteRunMinus1 is used as follows:

- If CopyAboveIndicesFlag[xC][yC] is equal to 0, PaletteRunMinus1 specifies the number of consecutive locations minus 1 with the same palette index.
- Otherwise if palette_transpose_flag equal to 0, PaletteRunMinus1 specifies the number of consecutive locations minus 1 with the same palette index as used in the corresponding position in the row above.
- Otherwise, PaletteRunMinus1 specifies the number of consecutive locations minus 1 with the same palette index as used in the corresponding position in the left column.

When RunToEnd is equal to 0, the variable PaletteMaxRunMinus1 represents the maximum possible value for PaletteRunMinus1 and it is a requirement of bitstream conformance that the value of PaletteMaxRunMinus1 shall be greater than or equal to 0.

palette_escape_val specifies the quantized escape coded sample value for a component.

The variable PaletteEscapeVal[cIdx][xC][yC] specifies the escape value of a sample for which PaletteIndexMap[xC][yC] is equal to MaxPaletteIndex and

palette_escape_val_present_flag is equal to 1. The array index cIdx specifies the colour component. The array indices xC, yC specify the location (xC, yC) of the sample relative to the top-left luma sample of the picture.

It is a requirement of bitstream conformance that PaletteEscapeVal[cIdx][xC][yC] shall be in the range of 0 to $(1 \ll (\text{BitDepth}_Y + 1)) - 1$, inclusive, for cIdx equal to 0, and in the range of 0 to $(1 \ll (\text{BitDepth}_C + 1)) - 1$, inclusive, for cIdx not equal to 0.

2.3 Local Dual Tree in VVC

[0051] In typical hardware video encoders and decoders, processing throughput drops when a picture has more small intra blocks because of sample processing data dependency between neighbouring intra blocks. The predictor generation of an intra block requires top and left boundary reconstructed samples from neighbouring blocks. Therefore, intra prediction has to be sequentially processed block by block.

[0052] In HEVC, the smallest intra CU is 8x8 luma samples. The luma component of the smallest intra CU can be further split into four 4x4 luma intra prediction units (PUs), but the chroma components of the smallest intra CU cannot be further split. Therefore, the worst case hardware processing throughput occurs when 4x4 chroma intra blocks or 4x4 luma intra blocks are processed.

[0053] In VTM5.0, in single coding tree, since chroma partitions always follows luma and the smallest intra CU is 4x4 luma samples, the smallest chroma intra CB is 2x2. Therefore, in VTM5.0, the smallest chroma intra CBs in single coding tree is 2x2. The worst case hardware processing throughput for VVC decoding is only 1/4 of that for HEVC decoding. Moreover, the reconstruction process of a chroma intra CB becomes much more complex than that in HEVC after adopting tools including cross-component linear model (CCLM), 4-tap interpolation filters, position-dependent intra prediction combination (PDPC), and combined inter intra prediction (CIIP). It is challenging to achieve high processing throughput in hardware decoders. In this section, a method that improve the worst case hardware processing throughput is proposed.

[0054] The goal of this method is to disallow chroma intra CBs smaller than 16 chroma samples by constraining the partitioning of chroma intra CBs.

[0055] In single coding tree, a SCIPU is defined as a coding tree node whose chroma block size is larger than or equal to TH chroma samples and has at least one child luma block smaller than 4TH luma samples, where TH is set to 16 in this contribution. It is required that in each SCIPU, all CBs are inter, or all CBs are non-inter, i.e, either intra or IBC. In case of a non-inter SCIPU, it is further required that chroma of the non-inter SCIPU shall not be further split and luma of the SCIPU is allowed to be further split. In this way, the smallest chroma intra CB size is 16 chroma samples, and 2x2, 2x4, and 4x2 chroma CBs are removed. In addition, chroma scaling is not applied in case of a non-inter SCIPU. In addition, when luma blocks are further split and chroma blocks are not split, a local dual tree coding structure is constructed.

[0056] Two SCIPU examples are shown in FIGS. 5A and 5B. In FIG. 5A, one chroma CB of 8x4 chroma samples and three luma CBs (4x8, 8x8, 4x8 luma CBs) form one SCIPU because the ternary tree (TT) split from the 8x4 chroma samples would result in chroma CBs smaller than 16 chroma samples. In FIG. 5B, one chroma CB of 4x4 chroma samples (the left side of the 8x4 chroma samples) and three luma CBs (8x4, 4x4, 4x4 luma CBs) form one SCIPU, and the other one chroma CB of 4x4 samples (the right side of the 8x4 chroma samples) and two luma CBs (8x4, 8x4 luma CBs) form one SCIPU because the binary tree (BT) split from the 4x4 chroma samples would result in chroma CBs smaller than 16 chroma samples.

[0057] In the proposed method, the type of a SCIPU is inferred to be non-inter if the current slice is an I-slice or the current SCIPU has a 4x4 luma partition in it after further split one time (because no inter 4x4 is allowed in VVC); otherwise, the type of the SCIPU (inter or non-inter) is indicated by one signalled flag before parsing the CUs in the SCIPU.

[0058] By applying the above method, the worst case hardware processing throughput occurs when 4x4, 2x8, or 8x2 chroma blocks, instead of a 2x2 chroma blocks, are processed. The worst case hardware processing throughput is the same as that in HEVC and is 4X of that in VTM5.0.

2.4 Transform Skip(TS)

[0059] As in HEVC, the residual of a block can be coded with transform skip mode. To avoid the redundancy of syntax coding, the transform skip flag is not signalled when the CU level `MTS_CU_flag` is not equal to zero. The block size limitation for transform skip is the same to that for MTS in JEM4, which indicate that transform skip is applicable for a CU when both block width and height are equal to or less than 32. Note that implicit MTS transform is set to DCT2 when LFNST or MIP is activated for the current CU. Also the implicit MTS can be still enabled when MTS is enabled for inter coded blocks.

[0060] In addition, for transform skip block, minimum allowed Quantization Parameter (QP) is defined as $6 * (\text{internalBitDepth} - \text{inputBitDepth}) + 4$.

2.5 Alternative luma half-pel interpolation filters

[0061] In JVET-N0309, alternative half-pel interpolation filters are proposed.

[0062] The switching of the half-pel luma interpolation filter is done depending on the motion vector accuracy. In addition to the existing quarter-pel, full-pel, and 4-pel AMVR modes, a new

half-pel accuracy AMVR mode is introduced. Only in case of half-pel motion vector accuracy, an alternative half-pel luma interpolation filter can be selected.

[0063] For a non-affine non-merge inter-coded CU which uses half-pel motion vector accuracy (i.e., the half-pel AMVR mode), a switching between the HEVC/VVC half-pel luma interpolation filter and one or more alternative half-pel interpolation is made based on the value of a new syntax element `hpelIfIdx`. The syntax element `hpelIfIdx` is only signaled in case of half-pel AMVR mode. In case of skip/merge mode using a spatial merging candidate, the value of the syntax element `hpelIfIdx` is inherited from the neighbouring block.

3. Technical problems solved by technical solutions and embodiments described herein

1. The current binarization of escape symbols is not fix length, which may be suitable for a source with a uniform distribution.
2. Current palette coding design performs an index adjustment process to remove possible redundancy, which may introduce parsing dependency, e.g. when an escape value index is wrongly derived.
3. The reference index employed to derive the current index may need an encoder constraint which is not considered in the current design and not desirable for a codec design.
4. When local dualtree is enabled, previous block and current block's palette entries may have different number of color components. How to handle such a case is not clear.
5. The local dual tree and PLT could not be applied simultaneously since some palette entries may be repeated when coding from a single tree region to a dual tree region. One example is shown in FIG. 6.
6. Chroma QP table for `joint_cbr` mode may be restricted.

4. A listing of embodiments and solutions

[0064] The list below should be considered as examples to explain general concepts. These items should not be interpreted in a narrow way. Furthermore, these items can be combined in any manner.

[0065] The following examples may be applied on palette scheme in VVC and all other palette related schemes.

[0066] Modulo(x, M) is defined as $(x \% M)$ when x is an positive integer; otherwise, it is defined as $M - ((-x) \% M)$.

[0067] The following examples may be applied on palette scheme in VVC and all other palette related schemes.

1. Fixed-length coding may be applied to code escape symbols.
 - a. In one example, escape symbols may be signaled with fixed length binarization.
 - b. In one example, an escape symbol may be signaled in fixed length binarization using N bits.
 - c. In one example, the code length (e.g., N mentioned in bullet 1.b) to signal an escape symbol may depend on internal bit depth.
 - i. Alternatively, the code length to signal an escape symbol may depend on input bit depth.
 - ii. Alternatively, the code length to signal an escape symbol may depend on the difference between internal bit depth and input bit depth.
 - iii. In one example N is set equal to input/internal bit depth.
 - d. In one example, the code length to signal an escape symbol (e.g., N mentioned in bullet 1.b) may depend on the quantization parameter, i.e. Q_p .
 - i. In one example, the code length for signaling an escape symbol may be a function of quantization parameter, such as denoted by $f(Q_p)$.
 1. In one example, the function f may be defined as (internal bitdepth – $g(Q_p)$).
 2. In one example, N may be set to (internal bitdepth – $\max(16, (Q_p - 4)/6)$).
 3. In one example, N may be set to (internal bitdepth – $\max(Q_{pPrimeTsMin}, (Q_p - 4)/6)$), wherein q_p is the decoded quantization parameter and $Q_{pPrimeTsMin}$ is the minimum allowed quantization parameter for transform skip mode.
 4. Alternatively, furthermore, the code length N may be set to $\max(A, \text{internal bitDepth} - (\max(Q_{pPrimeTsMin}, Q_p) - 4)/6)$ wherein A is non-negative integer value, such as 0 or 1.
 - e. In the above examples, N may be greater than or equal to 0.
2. It is proposed to disable the usage of escape symbols in one video unit (e.g., a CU).

- a. Alternatively, furthermore, the signaling of indication of escape symbol presence is skipped.
- b. In one example, whether to enable/disable the usage of escape symbols may depend on the quantization parameters and/or bit depth.
 - i. In one example, if $(\text{internal bitDepth} - (\text{Max}(\text{QpPrimeTsMin}, \text{Qp}) - 4)/6)$ is no greater than 0, the usage of escape symbols may be disabled.
- 3. Variable length coding excluding EG with 3rd order may be applied to code escape symbols.
 - a. In one example, the binarization of an escape symbol may be truncated binary (TB) with an input parameter K.
 - b. In one example, the binarization of an escape symbol may be EG with Kth order wherein K is unequal to 3.
 - i. In one example, the binarization of an escape symbol may be EG with 0th order.
 - 1. Alternatively, in one example, the binarization of an escape symbol may be EG with 1th order.
 - 2. Alternatively, in one example, the binarization of an escape symbol may be EG with 2th order.
 - c. In above examples, K may be an integer number and may depend on
 - i. A message signaled in the SPS/VPS/PPS/picture header/slice header/tile group header/LCU row/group of LCUs/bricks.
 - ii. Internal bit depth
 - iii. Input bit depth
 - iv. Difference between internal bit depth and input depth
 - v. Block dimension of current block
 - vi. Current quantization parameter of current block
 - vii. Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)
 - viii. Coding structure (such as single tree or dual tree)
 - ix. Color component (such as luma component and/or chroma components)

4. Multiple binarization methods for coding escape symbols may be applied to a video unit (e.g., a sequence/picture/slice/tile/brick/subpicture/CTU row/CTU/CTB/CB/CU/a sub-region within a picture) and/or for one or multiple values of escape symbols.
 - a. In one example, how to select one of the multiple binarization methods may be signalled for the video unit and/or for one or multiple values of escape symbols.
 - b. In one example, how to select one of the multiple binarization methods may be derived for the video unit and/or for one or multiple values of escape symbols.
 - c. In one example, for one video unit and/or for one or multiple values of escape symbols, two or more binarization methods may be applied.
 - i. In one example, an index or a flag may be encoded/decoded to tell the selected binarization method.

In the following bullets, p may denote the symbol value of a color component, bd may denote bit-depth (e.g., the internal bit depth or input bit depth), ibd may denote input bit depth, and Qp may denote the quantization parameter for transform skip blocks or transform blocks. In addition, QPs for luma component and chroma component may be different or same. Bit depth may be associated with a given color component.

5. How to apply the quantization and/or inverse quantization process may depend on whether the block is coded with palette mode or not.
 - a. In one example, the quantization and/or inverse quantization process for escape symbols may be different from those used for normal intra/inter coded blocks with quantization applied.
6. The quantization and/or inverse quantization process for escape symbols may use bit-shifting.
 - a. In one example, right bit-shifting may be used for quantizing escape symbols.
 - i. In one example, the escape symbol may be signaled as $f(p, Qp)$ wherein p is the input symbol value (e.g., input luma/chroma sample value), and Qp is the derived quantization parameter for the corresponding color component.
 1. In one example, the function f may be defined as $p \gg g(Qp)$.

2. In one example, the function f may be defined as $(p + (1 \ll (g(QP) - 1))) \gg g(Qp)$.
3. In one example, the function f may be defined as $(0, (1 \ll bd) - 1, (p + (1 \ll (g(QP) - 1))) \gg g(Qp))$.
- ii. In one example, the escape symbol may be signaled as $h(p)$.
 1. In one example, the function h may be defined as $p \gg N$.
 2. In one example, the function h may be defined as $(p + (1 \ll (N - 1))) \gg N$.
 3. In one example, when `cu_transquant_bypass_flag` is equal to 1, N may be set to 0.
 4. In one example, when `cu_transquant_bypass_flag` is equal to 1, N may be equal to $(bd - ibd)$, where bd is internal bit-depth and ibd is input bit-depth.
 5. In one example, the function h may be defined as $\text{clip}(0, (1 \ll (bd - N) - 1, p \gg N)$, where bd is the internal bit depth for the current color component.
 6. In one example, the function h may be defined as $\text{clip}(0, (1 \ll (bd - N) - 1, (p + (1 \ll (N - 1))) \gg N)$, where bd is the internal bit depth for the current color component.
 7. In the above example, N may be in the range of $[0, (bd - 1)]$.
- b. In one example, left bit-shifting may be used for inverse quantizing escape symbols.
 - i. In one example, the escape symbol may be dequantized as $f(p, Qp)$, where p is the decoded escape symbol, and Qp is the derived quantization parameter for the corresponding color component.
 1. In one example, f may be defined as $p \ll g(Qp)$
 2. In one example, f may be defined as $(p \ll g(Qp)) + (1 \ll (g(Qp) - 1))$.
 - ii. In one example, the escape symbol may be reconstructed as $f(p, Qp)$, where p is the decoded escape symbol.

1. In one example, f may be defined as $\text{clip}(0, (1 \ll \text{bd}) - 1, p \ll g(Qp))$
2. In one example, f may be defined as $\text{clip}(0, (1 \ll \text{bd}) - 1, (p \ll g(Qp)) + (1 \ll (g(Qp) - 1)))$.
- iii. In one example, the escape symbol may be reconstructed as $h(p)$.
 1. In one example, the function h may be defined as $p \ll N$.
 2. In one example, the function h may be defined as $(p \ll N) + (1 \ll (N - 1))$
 3. In one example, when `cu_transquant_bypass_flag` is equal to 1, N may be set to 0.
 4. In one example, when `cu_transquant_bypass_flag` is equal to 1, N may be equal to $(\text{bd} - \text{ibd})$, where bd is internal bit-depth and ibd is input bit-depth.
 5. In one example, N is set to $(\max(QpPrimeTsMin, qP) - 4) / 6$, wherein qP is the decoded quantization parameter and $QpPrimeTsMin$ is the minimum allowed quantization parameter for transform skip mode.
 - a) In the above example, if both luma and chroma have transform skip modes, different minimum allowed quantization parameters for transform skip mode may be applied for different color components.
 6. Alternatively, for the above examples, N may be further clipped, such as $\min(\text{bd} - 1, N)$.
 7. In the above example, N may be in the range of $[0, (\text{bd} - 1)]$.
7. When applying left-shift as dequantization, reconstruction offset of an escape symbol p may depend on bitdepth information.
 - a. In one example, it may be dependent on the difference between internal bitdepth and input bitdepth, i.e. $\text{deltaBD} = \text{internal bidepth} - \text{input bitdepth}$.
 - b. When K is smaller or equal to deltaBD , the reconstructed value may be $p \ll K$.
 - c. When K is larger than deltaBD , the reconstruction value may be $(p \ll K) + (1 \ll (K - 1))$

- d. When K is smaller or equal to T_0 (e.g., $T_0=2$), the reconstructed value may be $p \ll K$.
 - e. When K is larger than T_1 (e.g., $T_1=2$), the reconstruction value may be $(p \ll K) + (1 \ll (K-1))$.
 - f. In one example, T_0 and T_1 in bullet d and e may be signalled in the bitstream, such as in sequence/picture/slice/tile/brick/subpicture-level.
 - g. In one example, the reconstruction value may be $(p \ll K) + ((1 \ll (K-1)) \gg \text{deltaBD} \ll \text{deltaBD})$.
 - h. In one example, the reconstruction value may be $((p \ll (K+1)) + (1 \ll K) \gg (\text{deltaBD}+1) \ll \text{deltaBD})$.
 - i. In one example, the deltaBD may be signaled in the bitstream, such as in sequence/picture/slice/tile/brick/subpicture-level.
 - j. In one example, which reconstruction value shall be used (e.g., bullets b to e) may depend on the quantization parameter of current block.
 - k. In one example, which reconstruction value shall be used (e.g., bullets b to e) may depend on the value of deltaBD .
 - l. In one example, K may be set to $g(Qp)$.
8. In the above examples, the following may apply:
- a. In one example, the escape symbols may be context coded.
 - b. In one example, the escape symbols may be bypass coded.
 - c. In one example, $g(Qp)$ may be defined as $(Qp-4)/6$ or $QP/8$.
 - i. Alternatively, $g(Qp)$ may be defined as $Qp/6$ or $QP/8$.
 - ii. Alternatively, $g(Qp)$ may be defined as $\max(16, Qp/6)$.
 - iii. Alternatively, $g(Qp)$ may be defined as $\max(16, (Qp-4)/6)$.
 - iv. Alternatively, $g(Qp)$ may be defined as $\max((bd - ibd)*6 + 4, (Qp-4)/6)$.
 - v. Alternatively, $g(Qp)$ may be defined as $\max(M, (Qp-4)/6)$.
 - 1. In one example, M may be signalled to the decoder.
 - vi. Alternatively, $g(Qp)$ may be defined as $\max((M, Qp)-4)/6$.
 - 1. In one example, M may be indicated in the SPS.
 - 2. In one example, same or different M may be applied on luma and chroma components.
 - 3. In one example, M may be equal to $(bd - ibd)*6 + 4$.
 - vii. Alternatively, $g(Qp)$ may be defined as $Qp/6$ or $QP/8$.
 - viii. Alternatively, $g(Qp)$ may be defined as $\max(16, Qp/6)$.

- ix. Alternatively, $g(Q_p)$ may be defined as $(\max(16, Q_p) - 4)/6$.
- d. In one example, the value of $g(Q_p)$ may be in the range of $[0, (bd - 1)]$.
- e. In one example, the max function $\max(a, i)$ may be defined as $(i \leq a ? a : i)$.
 - i. Alternatively, in one example, the max function $\max(a, i)$ may be defined as $(i < a ? a : i)$.
- f. In one example, N may be an integer number (e.g. 8 or 10) and may depend on
 - i. A message signaled in the SPS/VPS/PPS/picture header/slice header/tile group header/LCU row/group of LCUs/bricks.
 - ii. Internal bit depth
 - iii. Input bit depth
 - iv. Difference between internal bit depth and input depth
 - v. Block dimension of current block
 - vi. Current quantization parameter of current block
 - vii. Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)
 - viii. Coding structure (such as single tree or dual tree)
 - ix. Color component (such as luma component and/or chroma components)
 - x. Slice/tile group type and/or picture type
- g. In one example, N may be signaled to the decoder.
- 9. Q_p for escape values may be clipped.
 - a. In one example, the lowest Q_p applied to escape values may be equal to $\min_qp_prime_ts_minus4$.
 - b. In one example, the lowest Q_p applied to escape values may be related to $\min_qp_prime_ts_minus4$.
 - i. In one example, the lowest Q_p applied to escape values may be equal to $\min_qp_prime_ts_minus4 + 4$.
 - c. In one example, the lowest Q_p for each color component may be indicated in the SPS/PPS/VPD/DPS/Tile/Slice header.
 - d. In one example, the lowest Q_p applied to escape values may be $(bd - ibd) * 6 + 4$, where bd is the internal bit depth and ibd denotes the input bit depth for a certain color component.
 - e. In one example, the above examples may be applied to a certain color component.
- 10. In the above examples, the chroma Q_p for escape values may use the Q_p before/after mapping.

11. It is proposed to not use a reference index when deriving the current palette index in the palette mode.
- a. In one example, the palette index may be directly signaled without excluding the possibility of a reference index (e.g. `adjustedRefPaletteIndex`).
 - i. Alternatively, in one example, the encoder may be constrained to enable the reference index always being different from the current index. In such as case, the palette index may be signaled by excluding the possibility of a reference index.
 - b. In one example, the binarization of a palette index may be Truncated binary (TB) with using maximal palette index as a binarization input parameter.
 - c. In one example, the binarization of a palette index may be fixed length.
 - d. In one example, the binarization of a palette index may be EG with Kth order.
 - i. In one example, K may be an integer number (e.g. 1, 2 or 3) and may depend on
 - 1. A message signaled in the SPS/VPS/PPS/picture header/slice header/tile group header/LCU row/group of LCUs/bricks.
 - 2. Internal bit depth
 - 3. Input bit depth
 - 4. Difference between internal bit depth and input depth
 - 5. Block dimension of current block
 - 6. Current quantization parameter of current block
 - 7. Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)
 - 8. Coding structure (such as single tree or dual tree)
 - 9. Color component (such as luma component and/or chroma components)
 - e. In one example, the above examples may be applied only when the current block has one escape sample at least.
12. Current palette index may be signaled independent from the previous palette indices.
- a. In one example, whether and/or how to use previous palette indices may depend on whether there is escape sample(s) in the current block.

13. Derivation from an index for escape symbols to an index for non-escape symbols may be disallowed.
 - a. In one example, when escape symbols are applied and the palette index is not equal to the index for escape symbols, it may be disallowed to decode the symbols as an escape symbol.
14. Derivation from an index for non-escape symbols to an index for escape symbols may be disallowed.
 - a. In one example, when escape symbols are applied and the palette index is equal to the index for escape symbols, it may be disallowed to decode the symbols as a non-escape symbol.
15. A derived palette index may be capped by the current palette table size.
 - a. In one example, when the palette index is larger than `MaxPaletteIndex`, it may be modified to equal to `MaxPaletteIndex`.
16. A derived palette index may be capped by the current palette table size excluding the index for escape symbols.
 - a. In one example, when escape symbols are not applied and the palette index is larger than `MaxPaletteIndex`, it may be modified to equal to `MaxPaletteIndex`.
 - b. In one example, when escape symbols are applied and the palette index is larger than $(\text{MaxPaletteIndex} - 1)$, it may be modified to equal to $(\text{MaxPaletteIndex} - 1)$.
17. The index to indicate escape symbol may be disallowed to be modified.
 - a. In one example, index being equal to be `MaxPaletteIndex` may always indicate escape symbol when escape symbols present in the current block.
 - b. In one example, index not equal to be `MaxPaletteIndex` cannot be decoded as an index to indicate escape symbol.
18. It is proposed to code the difference between a reference index and current index
 - a. In one example, the difference equal to be 0 may be disallowed to be coded.
 - b. Alternatively, for the first index in a palette coded block, the index may be directly coded.
19. It is proposed to code the modulo of the difference between a reference index (denoted as R), and the current index (denoted as C)

- a. In one example, $I = \text{Modulo}(C-R, \text{MaxPaletteIndex})$ may be coded.
 - i. In one example, the index may be reconstructed as $\text{Modulo}(I+R, \text{MaxPaletteIndex})$
 - ii. In one example, $\text{Modulo}(C-R, \text{MaxPaletteIndex})$ equal to be 0 may be disallowed in the bitstream.
 - iii. In one example, truncated binary code with $c\text{Max} = \text{MaxPaletteIndex}$ may be used to code the value.
 - iv. Alternatively, for the first index in a palette coded block, the index may be directly coded.
 - b. In one example, $I = \text{Modulo}(C-R, \text{MaxPaletteIndex}) - 1$ may be coded.
 - i. In one example, the index may be reconstructed as $\text{Modulo}(I+1+R, \text{MaxPaletteIndex})$
 - ii. In one example, $\text{Modulo}(C-R, \text{MaxPaletteIndex}) - 1$ smaller than 0 may be disallowed in the bitstream.
 - iii. In one example, truncated binary code with $c\text{Max} = (\text{MaxPaletteIndex} - 1)$ may be used to code the value I.
 - iv. Alternatively, for the first index in a palette coded block, $\text{Modulo}(C-R, \text{MaxPaletteIndex})$ may be coded.
 - v. Alternatively, for the first index in a palette coded block, the index may be directly coded.
20. At the beginning of decoding a palette block, the reference index R may be set equal to -1
- a. Alternatively, the reference index R may be set equal to 0.
21. It is proposed to enable the palette mode and the local dual tree exclusively.
- a. In one example, the local dual tree may be not allowed when the palette mode is enabled.
 - i. Alternatively, in one example, the palette mode may be not allowed when the local dual tree is enabled.
 - b. In one example, the local dual tree is not enabled on a specific color format, such as 4:4:4.
 - c. In one example, palette mode may be disallowed when a coding tree is of `MODE_TYPE_INTRA`.

22. It is proposed to remove repeated palette entries in the palette prediction table when local dual tree is applied.
- a. In one example, the palette prediction table may be reset when local dual tree is applied.
 - i. Alternatively, in one example, the decoder may check all palette entries in the prediction table and remove repeated ones when local dual tree is applied.
 - ii. Alternatively, in one example, the encoder may add a constraint that is considering two palette entries different when three components of their entries are different.
23. When the current palette entry has a different number of color components from an entry the palette prediction table, the palette prediction table may be disallowed to be used.
- a. In one example, reused flags for all entries in the palette prediction table may be marked as true but may not be used for the current block when the current palette entry has a different number of color components from prediction.
 - b. In one example, reused flags for all entries in the palette prediction table may be marked as false when the current palette entry has a different number of color components from prediction.
24. When the prediction table and current palette table have different color component(s), the palette prediction table may be disallowed to be used.
- a. In one example, reused flags for all entries in the palette prediction table may be marked as true but may not be used for the current block when prediction table and current palette table have different color components.
 - b. In one example, reused flags for all entries in the palette prediction table may be marked as false when prediction table and current palette table have different color components.
25. The escape symbols may be predictively coded, such as based on previously coded escape symbols.
- a. In one example, an escape symbol of one component may be predicted by coded values in the same color component.

- i. In one example, the escape symbol may employ the previously one coded escape symbol in the same component as a predictor and the residue between them may be signaled.
- ii. Alternatively, the escape symbol may employ the previously K^{th} coded escape symbol in the same component as a predictor and the residue between them may be signaled.
- iii. Alternatively, the escape symbol may be predicted from multiple (e.g., K) coded escape symbols in the same component.
 - 1. In one example, K may be an integer number (e.g. 1, 2 or 3) and may depend on
 - a) A message signaled in the SPS/VPS/PPS/picture header/slice header/tile group header/LCU row/group of LCUs/bricks.
 - b) Internal bit depth
 - c) Input bit depth
 - d) Difference between internal bit depth and input depth
 - e) Block dimension of current block
 - f) Current quantization parameter of current block
 - g) Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)
 - h) Coding structure (such as single tree or dual tree)
 - i) Color component (such as luma component and/or chroma components)
 - b. In one example, an escape symbol of one component may be predicted by coded values of another component.
 - c. In one example, a pixel may have multiple color components, and if the pixel is treated as escape symbol, the value of one component may be predicted by the values of samples of other components.
 - i. In one example, the U component of an escape symbol may be predicted by the V component of that symbol.

- d. In one example, the above methods may be only applied to certain color component (e.g., on luma component or chroma components), or under certain conditions such as based on coded information.
26. The contexts for run-length coding in palette mode may depend on the palette index for indexing the palette entries.
- a. In one example, the palette index after the index adjustment process at the decoder (mentioned in section 2.1.3) may be employed to derive contexts for the prefix of a length element (e.g. `palette_run_prefix`).
 - b. Alternatively, in one example, the I defined in the bullet 13 may replace the palette index to derive contexts for the prefix of a length element (e.g. `palette_run_prefix`).
27. It is proposed to align the positions of left neighboring block and/or above neighboring block employed in the derivation process for the quantization parameter predictors with the positions of neighboring left block and/or above neighboring block used in the mode/MV (e.g., MPM) derivation.
- a. The positions of left neighboring block and/or above neighboring block employed in the derivation process for the quantization parameter may be aligned with that used in the merge/AMVP candidate list derivation process.
 - b. In one example, the positions of neighboring left block and/or above block employed in the derivation process for the quantization parameter may be the left/above neighboring blocks shown in FIG. 7.
28. Block-level QP difference may be sent independent of whether escape samples exist in the current block.
- a. In one example, whether and/or how to send block-level QP difference may follow blocks coded in other modes than palette.
 - b. In one example, block-level QP difference may be always not sent for a palette block.
 - c. In one example, block-level QP difference may be sent for a palette block when block width is larger than a threshold.
 - d. In one example, block-level QP difference may be sent for a palette block when block height is larger than a threshold.

- e. In one example, block-level QP difference may be sent for a palette block when block size is larger than a threshold.
 - f. In one example, the above examples may only apply to luma or chroma blocks.
29. One or more of the coded block flags (CBFs) (e.g., `cbf_luma`, `cbf_cb`, `cbf_cr`) for a palette block may be set equal to 1.
- a. In one example, the CBF for a palette block may be always set equal to 1.
 - b. One or more of the CBFs for a palette block may depend on whether escape pixels exist in the current block.
 - i. In one example, when a palette block has escape samples, its `cbf` may be set equal to 1.
 - ii. Alternatively, when a palette block does not have escape samples, its `cbf` may be set equal to 0.
 - c. Alternatively, when accessing a neighboring palette coded block, it may be treated as an intra coded block with CBF equal to 1.
30. The difference between luma and/or chroma QP applied to a palette block and QP derived for the block (e.g. Qp_Y or Qp'_Y in JVET-O2001-vE spec) may be set equal to a fixed value for palette blocks.
- a. In one example, the luma and/or chroma QP offset may be set equal to 0.
 - b. In one example, the chroma QP offsets for Cb and Cr may be different.
 - c. In one example, the luma QP offset and chroma QP offsets may be different.
 - d. In one example, the chroma QP offset(s) may be indicated in DPS/VPS/SPS/PPS/Slice/Brick/Tile header.
31. The number of palette indices explicitly signalled or inferred for the current block (e.g., `num_palette_indices_minus1+1`), denoted by `NumPIdx`, may be restricted to be greater than or equal to K.
- a. In one example, K may be determined based on the current palette size, the escape flag and/or other information of palette coded blocks. Let S be current palette size of a current block and E be the value of escape present flag (e.g., **`palette_escape_val_present_flag`**). Let `BlkS` be the current block size.
 - i. In one example, K may be set equal to S.
 - ii. Alternatively, in one example, K may be set equal to $S + E$.

- iii. Alternatively, in one example, K may be set equal to (Number of Predicted Palette entries + number of signalled palette entries + palette_escape_val_present_flag) (e.g., **NumPredictedPaletteEntries + num_signalled_palette_entries + palette_escape_val_present_flag**).
 - b. In one example, (NumPtlIdx minus K) instead of num_palette_indices_minus1 may be signalled/parsed.
 - i. Alternatively, furthermore, it may be signaled only when (S+E) is no smaller than 1.
 - ii. In one example, the value of (NumPtlIdx minus K) may be signaled with a binarization method that the binarized bin string may have a pre-fix (e.g., truncated unary) and/or a suffix with m-th EG code.
 - iii. In one example, the value of (NumPtlIdx minus K) may be signaled with a truncated binary binarization method.
 - iv. In one example, the value of (NumPtlIdx minus K) may be signaled with a truncated unary binarization method.
 - v. In one example, the value of (NumPtlIdx minus K) may be signaled with a m-th EG binarization method.
 - c. In one example, a conformance bitstream shall satisfy that NumPtlIdx is greater than or equal to K.
 - d. In one example, a conformance bitstream shall satisfy that NumPtlIdx is smaller than or equal to K'.
 - i. In one example, K' is set to (block width * block height).
- 32. Whether and/or how apply the above methods may be based on:
 - a. Video contents (e.g. screen contents or natural contents)
 - b. A message signaled in the DPS/SPS/VPS/PPS/APS/picture header/slice header/tile group header/ Largest coding unit (LCU)/Coding unit (CU)/LCU row/group of LCUs/TU/PU block/Video coding unit
 - c. Position of CU/PU/TU/block/Video coding unit
 - d. Block dimension of current block and/or its neighboring blocks
 - e. Block shape of current block and/or its neighboring blocks
 - f. Indication of the color format (such as 4:2:0, 4:4:4, RGB or YUV)

- g. Coding tree structure (such as dual tree or single tree)
- h. Slice/tile group type and/or picture type
- i. Color component (e.g. may be only applied on luma component and/or chroma component)
- j. Temporal layer ID
- k. Profiles/Levels/Tiers of a standard
- l. Whether the current block has one escape sample or not.
 - i. In one example, the above methods may be applied only when the current block has one escape sample at least.
- m. Whether current block is coded with lossless mode or not(e.g., `cu_transquant_bypass_flag`)
 - i. In one example, the above methods may be applied only when the current block is NOT coded with lossless mode.
- n. Whether lossless coding is enabled or not (e.g, `transquant_bypass_enabled`, `cu_transquant_bypass_flag`)
 - i. In one example, the above methods may be applied only when lossless coding is disabled.

BDPCM related

33. When one block is coded with BDPCM and it is split into multiple transform blocks or sub-blocks, the residual prediction may be done in block level, and signalling of residuals is done in sub-block/transform block level.
- a. Alternatively, furthermore, the reconstruction of one sub-block is disallowed in the reconstruction process of another sub-block.
 - b. Alternatively, the residual prediction and signalling of residuals is done in sub-block/transform block level.
 - i. In this way, the reconstruction of one sub-block may be utilized in the reconstruction process of another sub-block.

Chroma QP table related

34. For a given index, the value of the chroma QP table for `joint_cb_cr` mode may be constrained by both the value of the chroma QP table for Cb and the value of the chroma QP table for Cr.

- c. In one example, the value of the value of the chroma QP table for joint_cb_cr mode may be constrained between the value of the chroma QP table for Cb and the value of the chroma QP table for Cr, inclusive.

Deblocking related

- 35. MV comparison in deblocking may depend on whether the alternative half-pel interpolation filter is used(e.g. indicated by hpelIfIdx in the JVET-O2001-vE spec)
 - d. In one example, blocks using different interpolation filters may be treated as having different MVs.
 - e. In one example, a constant offset may be added to the MV difference for deblocking comparison when the alterbative half-pel interpolation filter is involved.

5. Embodiments

The embodiment is based on JVET-O2001-vE. The newly added texts are enclosed in double bolded parentheses, e.g., **{{a}}** indicates that “a” has been added. The deleted texts are enclosed in double bolded brackets, e.g., **[[b]]** indicates that “b” has been deleted.

5.1 Embodiment #1

Decoding process for palette mode

Inputs to this process are:

- a location (xCb, yCb) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current picture,
- a variable startComp specifies the first colour component in the palette table,
- a variable cIdx specifying the colour component of the current block,
- two variables nCbW and nCbH specifying the width and height of the current block, respectively.

Output of this process is an array recSamples[x][y], with $x = 0.. nCbW - 1$, $y = 0.. nCbH - 1$ specifying reconstructed sample values for the block.

Depending on the value of cIdx, the variables nSubWidth and nSubHeight are derived as follows:

- If cIdx is equal to 0, nSubWidth is set to 1 and nSubHeight is set to 1.
- Otherwise, nSubWidth is set to SubWidthC and nSubHeight is set to SubHeightC.

The (nCbW x nCbH) block of the reconstructed sample array recSamples at location (xCb, yCb) is represented by recSamples[x][y] with $x = 0..nCbW - 1$ and $y = 0..nCbH - 1$, and the value of recSamples[x][y] for each x in the range of 0 to nCbW - 1, inclusive, and each y in the range of 0 to nCbH - 1, inclusive, is derived as follows:

- The variables xL and yL are derived as follows:

$$xL = \text{palette_transpose_flag} ? x * nSubHeight : x * nSubWidth \quad (8-268)$$

$$yL = \text{palette_transpose_flag} ? y * nSubWidth : y * nSubHeight \quad (8-269)$$

- The variable bIsEscapeSample is derived as follows:

- If $\text{PaletteIndexMap}[xCb + xL][yCb + yL]$ is equal to MaxPaletteIndex and $\text{palette_escape_val_present_flag}$ is equal to 1, bIsEscapeSample is set equal to 1.

- Otherwise, bIsEscapeSample is set equal to 0.

- If bIsEscapeSample is equal to 0, the following applies:

$$\text{recSamples}[x][y] = \text{CurrentPaletteEntries}[cIdx][\text{PaletteIndexMap}[xCb + xL][yCb + yL]] \quad (8-270)$$

- Otherwise, if $\text{cu_transquant_bypass_flag}$ is equal to 1, the following applies:

$$\text{recSamples}[x][y] = \text{PaletteEscapeVal}[cIdx][xCb + xL][yCb + yL] \quad (8-271)$$

- Otherwise (bIsEscapeSample is equal to 1 and $\text{cu_transquant_bypass_flag}$ is equal to 0), the following ordered steps apply:

1. The quantization parameter qP is derived as follows:

- If cIdx is equal to 0,

$$qP = \text{Max}(0, Qp'Y) \quad (8-272)$$

- Otherwise, if cIdx is equal to 1,

$$qP = \text{Max}(0, Qp'Cb) \quad (8-273)$$

- Otherwise (cIdx is equal to 2),

$$qP = \text{Max}(0, Qp'Cr) \quad (8-274)$$

2. The variables bitDepth is derived as follows:

$$\text{bitDepth} = (cIdx == 0) ? \text{BitDepth}_Y : \text{BitDepth}_C \quad (8-275)$$

3. [[The list levelScale[] is specified as $\text{levelScale}[k] = \{ 40, 45, 51, 57, 64, 72 \}$ with $k = 0..5$.]]

4. The following applies:

```

        [[tmpVal = ( PaletteEscapeVal[ cIdx ][ xCb + xL ][ yCb + yL ] *
        levelScale[ qP%6 ] ) << ( qP / 6 ) + 32 ) >> 6 (8-276))]
    {{ T is set equal to (internal_bit_depth – input_bit_depth) for component cIdx
        Nbits = max(T, (qP – 4) / 6)
        – If Nbits is equal to T
            recSamples[ x ][ y ] = PaletteEscapeVal[ cIdx ][ xCb + xL ][ yCb + yL ] <<
Nbits
        – Otherwise
            recSamples[ x ][ y ] = (PaletteEscapeVal[ cIdx ][ xCb + xL ][ yCb + yL ] <<
Nbits) + (1 << (Nbits - 1) ) }}
        [[recSamples[ x ][ y ] = Clip3( 0, ( 1 << bitDepth ) – 1, tmpVal ) (8-277)]]

```

When one of the following conditions is true:

- cIdx is equal to 0 and numComps is equal to 1;
- cIdx is equal to 2;

the variable PredictorPaletteSize[startComp] and the array PredictorPaletteEntries are derived or modified as follows:

```

    for( i = 0; i < CurrentPaletteSize[ startComp ]; i++ )
        for( cIdx = startComp; cIdx < (startComp + numComps); cIdx++ )
            newPredictorPaletteEntries[ cIdx ][ i ] = CurrentPaletteEntries[ cIdx ][ i ]
    newPredictorPaletteSize = CurrentPaletteSize[ startComp ]
    for( i = 0; i < PredictorPaletteSize && newPredictorPaletteSize <
    PaletteMaxPredictorSize; i++ )
        if( !PalettePredictorEntryReuseFlags[ i ] ) {
            for( cIdx = startComp; cIdx < (startComp + numComps); cIdx++ ) (8-278)
                newPredictorPaletteEntries[ cIdx ][ newPredictorPaletteSize ] =
                PredictorPaletteEntries[ cIdx ][ i ]
            newPredictorPaletteSize++
        }
    for( cIdx = startComp; cIdx < ( startComp + numComps ); cIdx++ )
        for( i = 0; i < newPredictorPaletteSize; i++ )

```

PredictorPaletteEntries[cIdx][i] = newPredictorPaletteEntries[cIdx][i]

PredictorPaletteSize[startComp] = newPredictorPaletteSize

It is a requirement of bitstream conformance that the value of PredictorPaletteSize[startComp] shall be in the range of 0 to PaletteMaxPredictorSize, inclusive.

5.2 Embodiment #2

This embodiment describes palette index derivation.

Palette coding semantics

[[The variable adjustedRefPaletteIndex is derived as follows:

adjustedRefPaletteIndex = MaxPaletteIndex + 1

if(PaletteScanPos > 0) {

xcPrev =

x0 + TraverseScanOrder[log2CbWidth][log2bHeight][PaletteScanPos - 1][0]

ycPrev =

y0 + TraverseScanOrder[log2CbWidth][log2bHeight][PaletteScanPos - 1][1]

if(CopyAboveIndicesFlag[xcPrev][ycPrev] == 0) {

adjustedRefPaletteIndex = PaletteIndexMap[xcPrev][ycPrev] { (7-157)

}

else {

if(!palette_transpose_flag)

adjustedRefPaletteIndex = PaletteIndexMap[xC][yC - 1]

else

adjustedRefPaletteIndex = PaletteIndexMap[xC - 1][yC]

}

}

When CopyAboveIndicesFlag[xC][yC] is equal to 0, the variable CurrPaletteIndex is derived as follows:

if(CurrPaletteIndex >= adjustedRefPaletteIndex)

CurrPaletteIndex++]]

Binarization process for palette_idx_idc

Input to this process is a request for a binarization for the syntax element palette_idx_idc and the variable MaxPaletteIndex.

Output of this process is the binarization of the syntax element.

The variable cMax is derived as follows:

- [[If this process is invoked for the first time for the current block,]]cMax is set equal to MaxPaletteIndex.
- [[Otherwise (this process is not invoked for the first time for the current block), cMax is set equal to MaxPaletteIndex minus 1.]]

The binarization for the palette_idx_idc is derived by invoking the TB binarization process specified in clause 9.3.3.4 with cMax.

5.3 Embodiment #3

Table 9-77 – Syntax elements and associated binarizations

Syntax element	Input parameters	
	Process	
palette_escape_val	[[EG3]] {{FL}}	{{bitDepth - (Max(QpPrimeTsMin, Qp'Y) - 4)/6}}

8.4.5.3 Decoding process for palette mode

Inputs to this process are:

- a location (xCb, yCb) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current picture,
- a variable startComp specifies the first colour component in the palette table,
- a variable cIdx specifying the colour component of the current block,
- two variables nCbW and nCbH specifying the width and height of the current block, respectively.

Output of this process is an array recSamples[x][y], with $x = 0.. nCbW - 1$, $y = 0.. nCbH - 1$ specifying reconstructed sample values for the block.

Depending on the value of cIdx, the variables nSubWidth and nSubHeight are derived as follows:

...

- Otherwise (bIsEscapeSample is equal to 1 and cu_transquant_bypass_flag is equal to 0), the following ordered steps apply:

5. The quantization parameter qP is derived as follows:

- If cIdx is equal to 0,

$$qP = \text{Max}(0, Qp'Y) \quad (8-272)$$

- Otherwise, if cIdx is equal to 1,

$$qP = \text{Max}(0, Qp'Cb) \quad (8-273)$$
- Otherwise (cIdx is equal to 2),

$$qP = \text{Max}(0, Qp'Cr) \quad (8-274)$$

6. The variables bitDepth is derived as follows:

$$\text{bitDepth} = (\text{cIdx} == 0) ? \text{BitDepth}_Y : \text{BitDepth}_C \quad (8-275)$$

7. [[The list levelScale[] is specified as levelScale[k] = { 40, 45, 51, 57, 64, 72 } with k = 0..5.]]

8. The following applies:

$$\begin{aligned} & [[\text{tmpVal} = (\text{PaletteEscapeVal}[\text{cIdx}][\text{xCb} + \text{xL}][\text{yCb} + \text{yL}] * \\ & \text{levelScale}[\text{qP} \% 6]) << ((\text{qP} / 6) + 32) >> 6 \quad (8-276)] \\ & \{\{\text{shift} = (\text{max}(\text{QpPrimeTsMin}, \text{qP}) - 4) / 6 \\ & \text{tmpVal} = (\text{PaletteEscapeVal}[\text{cIdx}][\text{xCb} + \text{xL}][\text{yCb} + \text{yL}] << \text{shift}) \}\} \\ & \text{recSamples}[\text{x}][\text{y}] = \text{Clip3}(0, (1 << \text{bitDepth}) - 1, \text{tmpVal}) \quad (8-277) \end{aligned}$$

5.4 Embodiment #4

copy_above_palette_indices_flag equal to 1 specifies that the palette index is equal to the palette index at the same location in the row above if horizontal traverse scan is used or the same location in the left column if vertical traverse scan is used. **copy_above_palette_indices_flag** equal to 0 specifies that an indication of the palette index of the sample is coded in the bitstream or inferred.

...

The variable adjustedRefPaletteIndex is derived as follows:

$$\begin{aligned} & \text{adjustedRefPaletteIndex} = \text{MaxPaletteIndex} + 1 \\ & \text{if}(\text{PaletteScanPos} > 0 \{\&\& \text{!palette_escape_val_present_flag}\}) \{ \\ & \quad \text{xcPrev} = \\ & \quad \text{x0} + \text{TraverseScanOrder}[\log2CbWidth][\log2bHeight][\text{PaletteScanPos} - 1][0] \\ & \quad \text{ycPrev} = \\ & \quad \text{y0} + \text{TraverseScanOrder}[\log2CbWidth][\log2bHeight][\text{PaletteScanPos} - 1][1] \\ & \quad \text{if}(\text{CopyAboveIndicesFlag}[\text{xcPrev}][\text{ycPrev}] == 0) \{ \\ & \quad \quad \text{adjustedRefPaletteIndex} = \text{PaletteIndexMap}[\text{xcPrev}][\text{ycPrev}] \quad (7-157) \\ & \quad \} \\ & \quad \text{else} \{ \end{aligned}$$

```

    if( !palette_transpose_flag )
        adjustedRefPaletteIndex = PaletteIndexMap[ xC ][ yC - 1 ]
    else
        adjustedRefPaletteIndex = PaletteIndexMap[ xC - 1 ][ yC ]
    }
}

```

When CopyAboveIndicesFlag[xC][yC] is equal to 0, the variable CurrPaletteIndex is derived as follows:

```

if( CurrPaletteIndex >= adjustedRefPaletteIndex )
    CurrPaletteIndex++      (7-158)

```

5.5 Embodiment #5

Table 9-77 – Syntax elements and associated binarizations

Syntax element		
	Process	Input parameters
palette_escape_val	[[EG3]] {{FL}}	{{max(1, bitDepth - (Max(QpPrimeTsMin, Qp'Y) - 4)/6)}}}

8.4.5.3 Decoding process for palette mode

Inputs to this process are:

- a location (xCb, yCb) specifying the top-left luma sample of the current block relative to the top-left luma sample of the current picture,
- a variable startComp specifies the first colour component in the palette table,
- a variable cIdx specifying the colour component of the current block,
- two variables nCbW and nCbH specifying the width and height of the current block, respectively.

Output of this process is an array recSamples[x][y], with $x = 0..nCbW - 1$, $y = 0..nCbH - 1$ specifying reconstructed sample values for the block.

Depending on the value of cIdx, the variables nSubWidth and nSubHeight are derived as follows:

...

- Otherwise (bIsEscapeSample is equal to 1 and cu_transquant_bypass_flag is equal to 0), the following ordered steps apply:

9. The quantization parameter qP is derived as follows:

- If cIdx is equal to 0,

$$qP = \text{Max}(0, Qp'Y) \quad (8-272)$$

- Otherwise, if cIdx is equal to 1,

$$qP = \text{Max}(0, Qp'Cb) \quad (8-273)$$

- Otherwise (cIdx is equal to 2),

$$qP = \text{Max}(0, Qp'Cr) \quad (8-274)$$

10. The variables bitDepth is derived as follows:

$$\text{bitDepth} = (\text{cIdx} == 0) ? \text{BitDepth}_Y : \text{BitDepth}_C \quad (8-275)$$

11. [[The list levelScale[] is specified as levelScale[k] = { 40, 45, 51, 57, 64, 72 } with k = 0..5.]]

12. The following applies:

$$\begin{aligned} & \text{[[tmpVal} = (\text{PaletteEscapeVal}[\text{cIdx}][\text{xCb} + \text{xL}][\text{yCb} + \text{yL}] * \\ & \text{levelScale}[\text{qP}\%6]) << (\text{qP} / 6) + 32) >> 6 \quad (8-276)\text{]]} \\ & \{\{\text{shift} = \min(\text{bitDepth} - 1, (\max(\text{QpPrimeTsMin}, \text{qP}) - 4) / 6) \\ & \text{tmpVal} = (\text{PaletteEscapeVal}[\text{cIdx}][\text{xCb} + \text{xL}][\text{yCb} + \text{yL}] << \text{shift}) \}\} \\ & \text{resSamples}[\text{x}][\text{y}] = \text{Clip3}(0, (1 << \text{bitDepth}) - 1, \text{tmpVal}) \quad (8-277) \end{aligned}$$

5.6 Embodiment #6

This embodiment illustrates a design to skip transform shift for transform skip, and is based on JVET-O2001-vE.

8.7.2 Scaling and transformation process

Inputs to this process are:

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

Output of this process is the (nTbW)x(nTbH) array of residual samples resSamples[x][y] with x = 0..nTbW - 1, y = 0..nTbH - 1.

The variables bitDepth, bdShift and tsShift are derived as follows:

$$\begin{aligned} \text{bitDepth} &= (\text{cIdx} == 0) ? \text{BitDepth}_Y : \text{BitDepth}_C \\ & \quad (8-942) \end{aligned}$$

$$\text{bdShift} = \text{Max}(20 - \text{bitDepth}, 0) \quad (8-943)$$

$$\begin{aligned} & \ll \text{tsShift} = 5 + ((\text{Log2}(\text{nTbW}) + \text{Log2}(\text{nTbH})) / 2) \\ & (8-944) \end{aligned}$$

The variable codedCIdx is derived as follows:

- If cIdx is equal to 0 or TuCResMode[xTbY][yTbY] is equal to 0, codedCIdx is set equal to cIdx.
- Otherwise, if TuCResMode[xTbY][yTbY] is equal to 1 or 2, codedCIdx is set equal to 1.
- Otherwise, codedCIdx is set equal to 2.

The variable cSign is set equal to $(1 - 2 * \text{slice_joint_cbcr_sign_flag})$.

The (nTbW)x(nTbH) array of residual samples resSamples is derived as follows:

1. The scaling process for transform coefficients as specified in clause 8.7.3 is invoked with the transform block location (xTbY, yTbY), the transform block width nTbW and the transform block height nTbH, the colour component variable cIdx being set equal to codedCIdx and the bit depth of the current colour component bitDepth as inputs, and the output is an (nTbW)x(nTbH) array of scaled transform coefficients d.
2. The (nTbW)x(nTbH) array of residual samples r is derived as follows:
 - [[If transform_skip_flag[xTbY][yTbY] is equal to 1 and cIdx is equal to 0, the residual sample array values r[x][y] with $x = 0..nTbW - 1$, $y = 0..nTbH - 1$ are derived as follows:

$$r[x][y] = d[x][y] \ll \text{tsShift} \quad (8-945)$$
 - [[Otherwise (transform_skip_flag[xTbY][yTbY] is equal to 0 or and cIdx is not equal to 0),]] the transformation process for scaled transform coefficients as specified in clause 8.7.4.1 is invoked with the transform block location (xTbY, yTbY), the transform block width nTbW and the transform block height nTbH, the colour component variable cIdx and the (nTbW)x(nTbH) array of scaled transform coefficients d as inputs, and the output is an (nTbW)x(nTbH) array of residual samples r.
3. The intermediate residual samples res [x][y] with $x = 0..nTbW - 1$, $y = 0..nTbH - 1$ are derived as follows:

- {{If transform_skip_flag[xTbY][yTbY] is equal to 1 and cIdx is equal to 0, the following applies:

$$\text{res}[x][y] = d[x][y]$$

- {{Otherwise (transform_skip_flag[xTbY][yTbY] is equal to 0 or and cIdx is not equal to 0), the following applies:}}

$$\text{res}[x][y] = (r[x][y] + (1 \ll (\text{bdShift} - 1))) \gg \text{bdShift} \quad (8-946)$$

4. The residual samples resSamples[x][y] with $x = 0..nTbW - 1$, $y = 0..nTbH - 1$ are derived as follows:

- If cIdx is equal to codedCIdx, the following applies:

$$\text{resSamples}[x][y] = \text{res}[x][y] \quad (8-947)$$

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

$$\text{resSamples}[x][y] = \text{cSign} * \text{res}[x][y] \quad (8-948)$$

- Otherwise, the following applies:

$$\text{resSamples}[x][y] = (\text{cSign} * \text{res}[x][y]) \gg 1 \quad (8-949)$$

8.7.3 Scaling process for transform coefficients

...

The variable rectNonTsFlag is derived as follows:

$$\text{rect}[\text{NonTs}]\text{Flag} = ((\text{Log2}(nTbW) + \text{Log2}(nTbH)) \& 1) == 1 \quad (8-955)$$

$$[\text{transform_skip_flag}[xTbY][yTbY] = 0]$$

The variables bdShift, rectNorm and bdOffset are derived as follows:

- {{If transform_skip_flag[xTbY][yTbY] is equal to 1 and cIdx is equal to 0, the following applies:

$$\text{bdShift} = 10$$

- {{Otherwise, the following applies:}}

$$\text{bdShift} = \text{bitDepth} + ((\text{rect}[\text{NonTs}]\text{Flag} ? 1 : 0) + \quad (8-956)$$

$$(\text{Log2}(nTbW) + \text{Log2}(nTbH)) / 2 - 5 + \text{dep_quant_enabled_flag}$$

$$\text{bdOffset} = (1 \ll \text{bdShift}) \gg 1$$

(8-957)

The list $\text{levelScale}[j][k]$ is specified as $\text{levelScale}[j][k] = \{ \{ 40, 45, 51, 57, 64, 72 \}, \{ 57, 64, 72, 80, 90, 102 \} \}$ with $j = 0..1, k = 0..5$.

The $(nTbW) \times (nTbH)$ array dz is set equal to the $(nTbW) \times (nTbH)$ array $\text{TransCoeffLevel}[xTbY][yTbY][cIdx]$.

For the derivation of the scaled transform coefficients $d[x][y]$ with $x = 0..nTbW - 1, y = 0..nTbH - 1$, the following applies:

– The intermediate scaling factor $m[x][y]$ is derived as follows:

– If one or more of the following conditions are true, $m[x][y]$ is set equal to 16:

- $\text{sps_scaling_list_enabled_flag}$ is equal to 0.
- $\text{transform_skip_flag}[xTbY][yTbY]$ is equal to 1.

– Otherwise, the following applies:

$$m[x][y] = \text{ScalingFactor}[\text{Log2}(nTbW)][\text{Log2}(nTbH)][\text{matrixId}[x][y]],$$

with matrixId as specified in Table 7-5

(8-958)

– The scaling factor $ls[x][y]$ is derived as follows:

– If $\text{dep_quant_enabled_flag}$ is equal to 1, the following applies:

$$ls[x][y] = (m[x][y] * \text{levelScale}[\text{rect}[\text{NonTs}]\text{Flag}][qP + 1 \% 6]) << ((qP + 1) / 6)$$

(8-959)

– Otherwise ($\text{dep_quant_enabled_flag}$ is equal to 0), the following applies:

$$ls[x][y] = (m[x][y] * \text{levelScale}[\text{rect}[\text{NonTs}]\text{Flag}][qP \% 6]) << (qP / 6)$$

(8-960)

– When $\text{BdpcmFlag}[xTbY][yYbY]$ is equal to 1, $dz[x][y]$ is modified as follows:

– If $\text{BdpcmDir}[xTbY][yYbY]$ is equal to 0 and x is greater than 0, the following applies:

$$dz[x][y] = \text{Clip3}(\text{CoeffMin}, \text{CoeffMax}, dz[x - 1][y] + dz[x][y])$$

(8-961)

– Otherwise, if $\text{BdpcmDir}[xTbY][yYbY]$ is equal to 1 and y is greater than 0, the following applies:

$$dz[x][y] = \text{Clip3}(\text{CoeffMin}, \text{CoeffMax}, dz[x][y - 1] + dz[x][y])$$

(8-962)

– The value $\text{dnc}[x][y]$ is derived as follows:

$$\text{dnc}[x][y] = (dz[x][y] * ls[x][y] + \text{bdOffset}) >> \text{bdShift}$$

(8-963)

– The scaled transform coefficient $d[x][y]$ is derived as follows:

$$d[x][y] = \text{Clip3}(\text{CoeffMin}, \text{CoeffMax}, \text{dnc}[x][y])$$

(8-964)

[0068] FIG. 8 is a block diagram of a video processing apparatus 800. The apparatus 800 may be used to implement one or more of the methods described herein. The apparatus 800 may be embodied in a smartphone, tablet, computer, Internet of Things (IoT) receiver, and so on. The apparatus 800 may include one or more processors 802, one or more memories 804 and video processing hardware 806. The processor(s) 802 may be configured to implement one or more methods described in the present document. The memory (memories) 804 may be used for storing data and code used for implementing the methods and techniques described herein. The video processing hardware 806 may be used to implement, in hardware circuitry, some techniques described in the present document. In some implementations, the hardware 806 may be partially or completely a part of the processor 802, e.g., a graphics processor.

[0069] Some embodiments of the disclosed technology include making a decision or determination to enable a video processing tool or mode. In an example, when the video processing tool or mode is enabled, the encoder will use or implement the tool or mode in the processing of a block of video, but may not necessarily modify the resulting bitstream based on the usage of the tool or mode. That is, a conversion from the block of video to the bitstream representation of the video will use the video processing tool or mode when it is enabled based on the decision or determination. In another example, when the video processing tool or mode is enabled, the decoder will process the bitstream with the knowledge that the bitstream has been modified based on the video processing tool or mode. That is, a conversion from the bitstream representation of the video to the block of video will be performed using the video processing tool or mode that was enabled based on the decision or determination.

[0070] Some embodiments of the disclosed technology include making a decision or determination to disable a video processing tool or mode. In an example, when the video processing tool or mode is disabled, the encoder will not use the tool or mode in the conversion of the block of video to the bitstream representation of the video. In another example, when the video processing tool or mode is disabled, the decoder will process the bitstream with the knowledge that the bitstream has not been modified using the video processing tool or mode that was enabled based on the decision or determination.

[0071] FIG. 9 is a block diagram showing an example video processing system 900 in which various techniques disclosed herein may be implemented. Various implementations may include some or all of the components of the system 900. The system 900 may include input 902 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 902 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.

[0072] The system 900 may include a coding component 904 that may implement the various coding or encoding methods described in the present document. The coding component 904 may reduce the average bitrate of video from the input 902 to the output of the coding component 904 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 904 may be either stored, or transmitted via a communication connected, as represented by the component 906. The stored or communicated bitstream (or coded) representation of the video received at the input 902 may be used by the component 908 for generating pixel values or displayable video that is sent to a display interface 910. 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.

[0073] 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.

[0074] FIG. 10 is a block diagram that illustrates an example video coding system 100 that may utilize the techniques of this disclosure.

[0075] As shown in FIG. 10, video coding system 100 may include a source device 110 and a destination device 120. Source device 110 generates encoded video data which may be referred to as a video encoding device. Destination device 120 may decode the encoded video data generated by source device 110 which may be referred to as a video decoding device.

[0076] Source device 110 may include a video source 112, a video encoder 114, and an input/output (I/O) interface 116.

[0077] Video source 112 may include a source such as a video capture device, an interface to receive video data from a video content provider, and/or a computer graphics system for generating video data, or a combination of such sources. The video data may comprise one or more pictures. Video encoder 114 encodes the video data from video source 112 to generate a bitstream. The bitstream may include a sequence of bits that form a coded representation of the video data. The bitstream may include coded pictures and associated data. The coded picture is a coded representation of a picture. The associated data may include sequence parameter sets, picture parameter sets, and other syntax structures. I/O interface 116 may include a modulator/demodulator (modem) and/or a transmitter. The encoded video data may be transmitted directly to destination device 120 via I/O interface 116 through network 130a. The encoded video data may also be stored onto a storage medium/server 130b for access by destination device 120.

[0078] Destination device 120 may include an I/O interface 126, a video decoder 124, and a display device 122.

[0079] I/O interface 126 may include a receiver and/or a modem. I/O interface 126 may acquire encoded video data from the source device 110 or the storage medium/ server 130b. Video decoder 124 may decode the encoded video data. Display device 122 may display the decoded video data to a user. Display device 122 may be integrated with the destination device 120, or may be external to destination device 120 which be configured to interface with an external display device.

[0080] Video encoder 114 and video decoder 124 may operate according to a video compression standard, such as the High Efficiency Video Coding (HEVC) standard, Versatile Video Coding (VVC) standard and other current and/or further standards.

[0081] FIG. 11 is a block diagram illustrating an example of video encoder 200, which may be video encoder 114 in the system 100 illustrated in FIG. 10.

[0082] Video encoder 200 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 11, video encoder 200 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various

components of video encoder 200. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[0083] The functional components of video encoder 200 may include a partition unit 201, a predication unit 202 which may include a mode select unit 203, a motion estimation unit 204, a motion compensation unit 205 and an intra prediction unit 206, a residual generation unit 207, a transform unit 208, a quantization unit 209, an inverse quantization unit 210, an inverse transform unit 211, a reconstruction unit 212, a buffer 213, and an entropy encoding unit 214.

[0084] In other examples, video encoder 200 may include more, fewer, or different functional components. In an example, predication unit 202 may include an intra block copy(IBC) unit. The IBC unit may perform predication in an IBC mode in which at least one reference picture is a picture where the current video block is located.

[0085] Furthermore, some components, such as motion estimation unit 204 and motion compensation unit 205 may be highly integrated, but are represented in the example of FIG. 11 separately for purposes of explanation.

[0086] Partition unit 201 may partition a picture into one or more video blocks. Video encoder 200 and video decoder 300 may support various video block sizes.

[0087] Mode select unit 203 may select one of the coding modes, intra or inter, e.g., based on error results, and provide the resulting intra- or inter-coded block to a residual generation unit 207 to generate residual block data and to a reconstruction unit 212 to reconstruct the encoded block for use as a reference picture. In some example, Mode select unit 203 may select a combination of intra and inter predication (CIIP) mode in which the predication is based on an inter predication signal and an intra predication signal. Mode select unit 203 may also select a resolution for a motion vector (e.g., a sub-pixel or integer pixel precision) for the block in the case of inter-predication.

[0088] To perform inter prediction on a current video block, motion estimation unit 204 may generate motion information for the current video block by comparing one or more reference frames from buffer 213 to the current video block. Motion compensation unit 205 may determine a predicted video block for the current video block based on the motion information and decoded samples of pictures from buffer 213 other than the picture associated with the current video block.

[0089] Motion estimation unit 204 and motion compensation unit 205 may perform different operations for a current video block, for example, depending on whether the current video block is in an I slice, a P slice, or a B slice.

[0090] In some examples, motion estimation unit 204 may perform uni-directional prediction for the current video block, and motion estimation unit 204 may search reference pictures of list 0 or list 1 for a reference video block for the current video block. Motion estimation unit 204 may then generate a reference index that indicates the reference picture in list 0 or list 1 that contains the reference video block and a motion vector that indicates a spatial displacement between the current video block and the reference video block. Motion estimation unit 204 may output the reference index, a prediction direction indicator, and the motion vector as the motion information of the current video block. Motion compensation unit 205 may generate the predicted video block of the current block based on the reference video block indicated by the motion information of the current video block.

[0091] In other examples, motion estimation unit 204 may perform bi-directional prediction for the current video block, motion estimation unit 204 may search the reference pictures in list 0 for a reference video block for the current video block and may also search the reference pictures in list 1 for another reference video block for the current video block. Motion estimation unit 204 may then generate reference indexes that indicate the reference pictures in list 0 and list 1 containing the reference video blocks and motion vectors that indicate spatial displacements between the reference video blocks and the current video block. Motion estimation unit 204 may output the reference indexes and the motion vectors of the current video block as the motion information of the current video block. Motion compensation unit 205 may generate the predicted video block of the current video block based on the reference video blocks indicated by the motion information of the current video block.

[0092] In some examples, motion estimation unit 204 may output a full set of motion information for decoding processing of a decoder.

[0093] In some examples, motion estimation unit 204 may do not output a full set of motion information for the current video. Rather, motion estimation unit 204 may signal the motion information of the current video block with reference to the motion information of another video block. For example, motion estimation unit 204 may determine that the motion information of

the current video block is sufficiently similar to the motion information of a neighboring video block.

[0094] In one example, motion estimation unit 204 may indicate, in a syntax structure associated with the current video block, a value that indicates to the video decoder 300 that the current video block has the same motion information as the another video block.

[0095] In another example, motion estimation unit 204 may identify, in a syntax structure associated with the current video block, another video block and a motion vector difference (MVD). The motion vector difference indicates a difference between the motion vector of the current video block and the motion vector of the indicated video block. The video decoder 300 may use the motion vector of the indicated video block and the motion vector difference to determine the motion vector of the current video block.

[0096] As discussed above, video encoder 200 may predictively signal the motion vector. Two examples of predictive signaling techniques that may be implemented

[0097] by video encoder 200 include advanced motion vector predication (AMVP) and merge mode signaling.

[0098] Intra prediction unit 206 may perform intra prediction on the current video block. When intra prediction unit 206 performs intra prediction on the current video block, intra prediction unit 206 may generate prediction data for the current video block based on decoded samples of other video blocks in the same picture. The prediction data for the current video block may include a predicted video block and various syntax elements.

[0099] Residual generation unit 207 may generate residual data for the current video block by subtracting (e.g., indicated by the minus sign) the predicted video block(s) of the current video block from the current video block. The residual data of the current video block may include residual video blocks that correspond to different sample components of the samples in the current video block.

[00100] In other examples, there may be no residual data for the current video block for the current video block, for example in a skip mode, and residual generation unit 207 may not perform the subtracting operation.

[00101] Transform processing unit 208 may generate one or more transform coefficient video blocks for the current video block by applying one or more transforms to a residual video block associated with the current video block.

[00102] After transform processing unit 208 generates a transform coefficient video block associated with the current video block, quantization unit 209 may quantize the transform coefficient video block associated with the current video block based on one or more quantization parameter (QP) values associated with the current video block.

[00103] Inverse quantization unit 210 and inverse transform unit 211 may apply inverse quantization and inverse transforms to the transform coefficient video block, respectively, to reconstruct a residual video block from the transform coefficient video block. Reconstruction unit 212 may add the reconstructed residual video block to corresponding samples from one or more predicted video blocks generated by the predication unit 202 to produce a reconstructed video block associated with the current block for storage in the buffer 213.

[00104] After reconstruction unit 212 reconstructs the video block, loop filtering operation may be performed reduce video blocking artifacts in the video block.

[00105] Entropy encoding unit 214 may receive data from other functional components of the video encoder 200. When entropy encoding unit 214 receives the data, entropy encoding unit 214 may perform one or more entropy encoding operations to generate entropy encoded data and output a bitstream that includes the entropy encoded data.

[00106] FIG. 12 is a block diagram illustrating an example of video decoder 300 which may be video decoder 114 in the system 100 illustrated in FIG. 10.

[00107] The video decoder 300 may be configured to perform any or all of the techniques of this disclosure. In the example of FIG. 12, the video decoder 300 includes a plurality of functional components. The techniques described in this disclosure may be shared among the various components of the video decoder 300. In some examples, a processor may be configured to perform any or all of the techniques described in this disclosure.

[00108] In the example of FIG. 12, video decoder 300 includes an entropy decoding unit 301, a motion compensation unit 302, an intra prediction unit 303, an inverse quantization unit 304, an inverse transformation unit 305, and a reconstruction unit 306 and a buffer 307. Video decoder 300 may, in some examples, perform a decoding pass generally reciprocal to the encoding pass described with respect to video encoder 200 (FIG. 11).

[00109] Entropy decoding unit 301 may retrieve an encoded bitstream. The encoded bitstream may include entropy coded video data (e.g., encoded blocks of video data). Entropy decoding unit 301 may decode the entropy coded video data, and from the entropy decoded video data,

motion compensation unit 302 may determine motion information including motion vectors, motion vector precision, reference picture list indexes, and other motion information. Motion compensation unit 302 may, for example, determine such information by performing the AMVP and merge mode.

[00110] Motion compensation unit 302 may produce motion compensated blocks, possibly performing interpolation based on interpolation filters. Identifiers for interpolation filters to be used with sub-pixel precision may be included in the syntax elements.

[00111] Motion compensation unit 302 may use interpolation filters as used by video encoder 20 during encoding of the video block to calculate interpolated values for sub-integer pixels of a reference block. Motion compensation unit 302 may determine the interpolation filters used by video encoder 200 according to received syntax information and use the interpolation filters to produce predictive blocks.

[00112] Motion compensation unit 302 may use some of the syntax information to determine sizes of blocks used to encode frame(s) and/or slice(s) of the encoded video sequence, partition information that describes how each macroblock of a picture of the encoded video sequence is partitioned, modes indicating how each partition is encoded, one or more reference frames (and reference frame lists) for each inter-encoded block, and other information to decode the encoded video sequence.

[00113] Intra prediction unit 303 may use intra prediction modes for example received in the bitstream to form a prediction block from spatially adjacent blocks. Inverse quantization unit 303 inverse quantizes, i.e., de-quantizes, the quantized video block coefficients provided in the bitstream and decoded by entropy decoding unit 301. Inverse transform unit 303 applies an inverse transform.

[00114] Reconstruction unit 306 may sum the residual blocks with the corresponding prediction blocks generated by motion compensation unit 202 or intra-prediction unit 303 to form decoded blocks. If desired, a deblocking filter may also be applied to filter the decoded blocks in order to remove blockiness artifacts. The decoded video blocks are then stored in buffer 307, which provides reference blocks for subsequent motion compensation/intra prediction and also produces decoded video for presentation on a display device.

[00115] In some embodiments, the following methods are based on the listing of examples and embodiments enumerated above. In an example, these methods can be implemented using, but not limited to, the implementations shown in FIG. 8-12.

[00116] FIG. 13 is a flowchart of an example method for video processing. As shown therein, the method 1300 includes, at operation 1310, determining, for a conversion between a current block of a video and a bitstream representation of the video, that the current block is coded using a palette mode and an escape symbol value is signaled in the bitstream representation, such that a clipped quantization parameter for the current block is used in the conversion and the clipped quantization parameter used for a chroma component of the video is derived based on a quantization parameter subsequent to a mapping operation of a quantization process or a dequantization process.

[00117] The method 1300 further includes, at operation 1320, performing, based on the determining, the conversion.

[00118] FIG. 14 is a flowchart of an example method for video processing. As shown therein, the method 1400 includes, at operation 1410, determining, for a conversion between a current block of a video and a bitstream representation of the video, that the current block is coded using a palette mode and an escape symbol value is signaled in the bitstream representation, such that a clipped quantization parameter for the current block is used in the conversion and the clipped quantization parameter used for a chroma component of the video is derived based on a quantization parameter prior to a mapping operation of a quantization process or a dequantization process.

[00119] The method 1400 further includes, at operation 1420, performing, based on the determining, the conversion.

[00120] The following solutions may be implemented together with additional techniques described in items listed in the previous section (e.g., item 1) as preferred features of some embodiments.

[00121] 1. A method of video processing, comprising determining, for a conversion between a current block of a video and a bitstream representation of the video, that the current block is coded using a palette mode and an escape symbol value is signaled in the bitstream representation; and performing, based on the determining, the conversion, wherein a clipped quantization parameter for the current block is used in the conversion, wherein the palette mode

coding tool represents the current video block using a palette of representative color values, wherein the escape value is used for a sample of the current video block coded without using the representative color values, and wherein the clipped quantization parameter used for a chroma component of the video is derived based on a quantization parameter subsequent to a mapping operation of a quantization process or a dequantization process.

[00122] 2. A method of video processing, comprising determining, for a conversion between a current block of a video and a bitstream representation of the video, that the current block is coded using a palette mode and an escape symbol value is signaled in the bitstream representation; and performing, based on the determining, the conversion, wherein a clipped quantization parameter for the current block is used in the conversion, wherein the palette mode coding tool represents the current video block using a palette of representative color values, wherein the escape value is used for a sample of the current video block coded without using the representative color values, and wherein the clipped quantization parameter used for a chroma component of the video is derived based on a quantization parameter prior to a mapping operation of a quantization process or a dequantization process.

[00123] 3. The method of solution 1 or 2, wherein a lowest value of the clipped quantization parameter is based on a minimum allowed quantization parameter for a transform skip mode.

[00124] 4. The method of solution 3, wherein a lowest value of the clipped quantization parameter is based on `sps_min_qp_prime_ts`.

[00125] 5. The method of solution 4, wherein the lowest value of the clipped quantization parameter is equal to `QpPrimeTsMin`.

[00126] 6. The method of solution 1 or 2, wherein an indication of a lowest value of the clipped quantization parameter for each color component of the video is signaled in a sequence parameter set (SPS), picture parameter set (PPS), a video parameter set (VPS), a decoding parameter set (DPS), a tile, or a slice header in the bitstream representation.

[00127] 7. The method of solution 1 or 2, wherein a lowest value of the clipped quantization parameter is $(bd - ibd) \times 6 + 4$, wherein bd is an internal bit depth and ibd is an input bit depth for a color component of the video.

[00128] 8. The method of any of solutions 1 to 7, wherein the palette mode coding tool is applied to a certain color component of the video.

[00129] 9. The method of any of solutions 1 to 8, wherein performing the conversion comprises generating the bitstream representation from the current block.

[00130] 10. The method of any of solutions 1 to 8, wherein performing the conversion comprises generating the current block from the bitstream representation.

[00131] 11. An apparatus in a video system comprising a processor and a non-transitory memory with instructions thereon, wherein the instructions upon execution by the processor, cause the processor to implement the method in any one of solutions 1 to 10.

[00132] 12. A computer program product stored on a non-transitory computer readable media, the computer program product including program code for carrying out the method in any one of solutions 1 to 10.

[00133] The disclosed and other solutions, examples, embodiments, modules and the functional operations described in this document can be implemented in digital electronic circuitry, or in computer software, firmware, or hardware, including the structures disclosed in this document and their structural equivalents, or in combinations of one or more of them. The disclosed and other embodiments can be implemented as one or more computer program products, i.e., one or more modules of computer program instructions encoded on a computer readable medium for execution by, or to control the operation of, data processing apparatus. The computer readable medium can be a machine-readable storage device, a machine-readable storage substrate, a memory device, a composition of matter effecting a machine-readable propagated signal, or a combination of one or more them. The term “data processing apparatus” encompasses all apparatus, devices, and machines for processing data, including by way of example a programmable processor, a computer, or multiple processors or computers. The apparatus can include, in addition to hardware, code that creates an execution environment for the computer program in question, e.g., code that constitutes processor firmware, a protocol stack, a database management system, an operating system, or a combination of one or more of them. A propagated signal is an artificially generated signal, e.g., a machine-generated electrical, optical, or electromagnetic signal, that is generated to encode information for transmission to suitable receiver apparatus.

[00134] 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.

[00135] The processes and logic flows described in this document can be performed by one or more programmable processors executing one or more computer programs to perform functions by operating on input data and generating output. The processes and logic flows can also be performed by, and apparatus can also be implemented as, special purpose logic circuitry, e.g., an FPGA (field programmable gate array) or an ASIC (application specific integrated circuit).

[00136] Processors suitable for the execution of a computer program include, by way of example, both general and special purpose microprocessors, and any one or more processors of any kind of digital computer. Generally, a processor will receive instructions and data from a read only memory or a random-access memory or both. The essential elements of a computer are a processor for performing instructions and one or more memory devices for storing instructions and data. Generally, a computer will also include, or be operatively coupled to receive data from or transfer data to, or both, one or more mass storage devices for storing data, e.g., magnetic, magneto optical disks, or optical disks. However, a computer need not have such devices. Computer readable media suitable for storing computer program instructions and data include all forms of non-volatile memory, media and memory devices, including by way of example semiconductor memory devices, e.g., EPROM, EEPROM, and flash memory devices; magnetic disks, e.g., internal hard disks or removable disks; magneto optical disks; and CD ROM and DVD-ROM disks. The processor and the memory can be supplemented by, or incorporated in, special purpose logic circuitry.

[00137] While this patent document contains many specifics, these should not be construed as limitations on the scope of any subject matter or of what may be claimed, but rather as descriptions of features that may be specific to particular embodiments of particular techniques. Certain features that are described in this patent document in the context of separate

embodiments can also be implemented in combination in a single embodiment. Conversely, various features that are described in the context of a single embodiment can also be implemented in multiple embodiments separately or in any suitable subcombination. Moreover, although features may be described above as acting in certain combinations and even initially claimed as such, one or more features from a claimed combination can in some cases be excised from the combination, and the claimed combination may be directed to a subcombination or variation of a subcombination.

[00138] 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.

[00139] 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.