|  |  |
| --- | --- |
| **Joint Collaborative Team on Video Coding (JCT-VC)**  **of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11**  17th Meeting: Valencia, ES, 27 March – 4 April 2014 | Document: JCTVC-Q0037 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **Description of screen content coding technology proposal by InterDigital** | | |
| *Status:* | Input Document to JCT-VC | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Xiaoyu Xiu, Chia-Ming Tsai, Yuwen He, Yan Ye 9710 Scranton Rd, #250 San Diego, CA 92121 | Tel: Email: | +1-858-210-4830 {xiaoyu.xiu,chiaming.tsai,yuwen.he,yan.ye}@interdigital.com |
| *Source:* | InterDigital Communications, LLC | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

This proposal is InterDigital Communications’ response to the Call for Proposals (CfP) for coding of screen content, jointly issued by ISO/IEC JTC1/SC29/WG11 (MPEG) and ITU-T SG16 Q6 (VCEG). The goal of this proposal is to develop dedicated coding technologies to the characteristics of screen content for future extensions of HEVC for screen content coding. Two main technologies, namely improved palette coding and adaptive residue color space conversion, are proposed based on the current framework of HEVC Range Extensions.

Compared to the CfP anchors, for lossy coding, the proposed solution achieves the average {G, B, R} BD-rate reductions of {16.3%, 15.9%, 15.8%}, {13.4%, 13.2%, 13.1%} and {13.8%, 13.6%, 13.4%} for AI, RA and LD, respectively, in RGB coding, and the average luma BD-rate reductions of 13.2%, 9.2% and 7% for AI, RA and LD, respectively, in YCbCr coding. For lossless coding, the average bit-rate savings of the proposed solution are 14.8%, 16.2% and 16.7% for AI, RA and LD, respectively, in RGB coding, and 13.8%, 10.2% and 9.3% for AI, RA and LD, respectively, in YCbCr coding.

The performance improvement for screen content sequences (video sequences in the category “text & graphics with motion”) is significantly higher. For lossy coding, the proposed solution achieves average {G, B, R} BD-rate reductions of {28.9%, 28.4%, 28.3%}, {21.7%, 21.1%, 21.3%} and {18.2%, 17.5%, 17.8%} for AI, RA and LD, respectively, in RGB coding, and average luma BD-rate reductions of 22.7%, 15.7% and 10.3% for AI, RA and LD, respectively, in YCbCr coding. For lossless coding, the bit-rate savings of the proposed solution are 27.8%, 25.8% and 25.9% for AI, RA and LD, respectively, in RGB coding, and are 28%, 21.5% and 19.4% for AI, RA and LD, respectively, in YCbCr coding.

# Contents

[Abstract 1](#_Toc382833963)

[Contents 1](#_Toc382833964)

[1 Introduction 3](#_Toc382833965)

[2 Algorithm description 4](#_Toc382833966)

[2.1 CTU/TU/PU partitioning 4](#_Toc382833967)

[2.2 Inter-picture prediction 4](#_Toc382833968)

[2.3 Intra-picture prediction 5](#_Toc382833969)

[2.4 In-loop filtering 5](#_Toc382833970)

[2.4.1 De-blocking filter 5](#_Toc382833971)

[2.4.2 SAO 5](#_Toc382833972)

[2.5 Spatial transforms 5](#_Toc382833973)

[2.6 Other specific compression tools for screen content 5](#_Toc382833974)

[2.6.1 Improved palette coding mode 5](#_Toc382833975)

[2.6.2 Adaptive residue color space conversion 10](#_Toc382833976)

[2.7 Quantization 11](#_Toc382833977)

[2.8 Motion/mode/parameter coding 11](#_Toc382833978)

[2.8.1 Merge/skip 11](#_Toc382833979)

[2.8.2 Motion vector coding 11](#_Toc382833980)

[2.8.3 Intra prediction mode coding 11](#_Toc382833981)

[2.8.4 Other parameter coding 11](#_Toc382833982)

[2.9 Entropy coding of residual / transform coefficients 11](#_Toc382833983)

[2.10 High-layer syntax 12](#_Toc382833984)

[2.11 Additional algorithmic description discussion topic #1 12](#_Toc382833985)

[2.12 Additional algorithmic description discussion topic #2 12](#_Toc382833986)

[3 Syntax and semantics description 13](#_Toc382833987)

[3.1 Syntax 13](#_Toc382833988)

[3.1.1 Sequence parameter set syntax 13](#_Toc382833989)

[3.1.2 Picture parameter set syntax 14](#_Toc382833990)

[3.1.3 Coding unit syntax 15](#_Toc382833991)

[3.1.4 Palette coding syntax 17](#_Toc382833992)

[3.2 Semantics 18](#_Toc382833993)

[3.2.1 Sequence parameter set semantics 18](#_Toc382833994)

[3.2.2 Picture parameter set semantics 18](#_Toc382833995)

[3.2.3 Coding unit semantics 18](#_Toc382833996)

[3.2.4 Palette coding semantics 18](#_Toc382833997)

[4 Compression performance discussion 21](#_Toc382833998)

[4.1 Lossy test condition 21](#_Toc382833999)

[4.2 Mathematically lossless test condition 22](#_Toc382834000)

[5 Complexity analysis 23](#_Toc382834001)

[5.1 Encoding time and measurement methodology 23](#_Toc382834002)

[5.2 Decoding time and measurement methodology and comparison vs. anchor bitstreams decoded by HM 25](#_Toc382834003)

[5.3 Description of computing platform used to determine encoding and decoding times reported in sections 5.1 and 5.2 26](#_Toc382834004)

[5.4 Expected memory usage of encoder 27](#_Toc382834005)

[5.5 Expected memory usage of decoder 27](#_Toc382834006)

[5.6 Complexity characteristics of encoder motion estimation and partitioning selection 27](#_Toc382834007)

[5.7 Complexity characteristics of decoder motion compensation 28](#_Toc382834008)

[5.8 Complexity characteristics of encoder intra-frame prediction type and partitioning selection 28](#_Toc382834009)

[5.9 Complexity characteristics of decoder intra-frame prediction operation 28](#_Toc382834010)

[5.10 Complexity characteristics of encoder inter-component prediction tools 28](#_Toc382834011)

[5.11 Complexity characteristics of decoder inter-component prediction tools 28](#_Toc382834012)

[5.12 Complexity characteristics of transforms specific in screen content coding 28](#_Toc382834013)

[5.13 Complexity characteristics of encoder related to other tools specific in screen content coding 28](#_Toc382834014)

[5.14 Complexity characteristics of decoder related to other tools specific in screen content coding 28](#_Toc382834015)

[5.15 Complexity characteristics of encoder entropy coding operation related to tools/parameters specific in screen content coding 29](#_Toc382834016)

[5.16 Complexity characteristics of decoder entropy decoding operation related to tools/parameters specific in screen content coding 29](#_Toc382834017)

[5.17 Degree of capability for encoder parallel processing 29](#_Toc382834018)

[5.18 Degree of capability for decoder parallel processing 29](#_Toc382834019)

[5.19 Additional complexity discussion topic #1 29](#_Toc382834020)

[5.20 Additional complexity discussion topic #2 29](#_Toc382834021)

[5.21 … 29](#_Toc382834022)

[6 Software implementation description 29](#_Toc382834023)

[7 Highlighted aspects discussion 29](#_Toc382834024)

[8 Closing remarks 29](#_Toc382834025)

[9 Patent rights declaration(s) 30](#_Toc382834026)

[References 30](#_Toc382834027)

# Introduction

As the HEVC range extensions [1] comes close to finalization, a joint Call for Proposals on the screen content coding extensions of HEVC was recently issued by ISO/IEC JTC1/SC29/WG11 (MPEG) and by ITU-T SG16 Q.6 (VCEG) [2]. This proposal is InterDigital Communications’ response to the CfP. This proposal is based on the Range Extension Draft 6 reference platform [1], with the addition of two main new coding tools specifically for screen content coding:

1. **Improved palette coding**: the palette coding mode proposed in [3] is incorporated into this proposal, and 3 main improvements are made on top of the existing palette coding mode, including improved palette table prediction, improved palette index coding, and improved escape color coding.
2. **Adaptive residual color space conversion**: for RGB sequences, the prediction residuals are adaptively converted and coded in the YCgCo color space.

In addition, the 8-tap fractional pixel interpolation filters are applied to all 3 color components for lossy coding. Figure 1 and Figure 2 show the encoder and decoder block diagrams of the proposed SCC codec, respectively. The highlighted boxes are new coding blocks added on top of the Range Extensions reference platform.



Figure 1. Block diagram of the proposed encoder



Figure 2. Block diagram of the proposed

Compared to the CfP anchors, for lossy coding, the proposed solution achieves the average {G, B, R} BD-rate reductions of the proposed solution are {16.3%, 15.9%, 15.8%}, {13.4%, 13.2%, 13.1%} and {13.8%, 13.6%, 13.4%} for AI, RA and LD, respectively, in RGB coding, and the average luma BD-rate reductions of 13.2%, 9.2% and 7% for AI, RA and LD, respectively, in YCbCr coding. For lossless coding, the average bit-rate savings of the proposed solution are 14.8%, 16.2% and 16.7% for AI, RA and LD, respectively, in RGB coding, and 13.8%, 10.2% and 9.3% for AI, RA and LD, respectively, in YCbCr coding.

The performance improvement for screen content sequences (video sequences in the category “text & graphics with motion”) is significantly higher. For lossy coding, the proposed solution achieves {G, B, R} BD-rate reduction of {28.9%, 28.4%, 28.3%}, {21.7%, 21.1%, 21.3%} and {18.2%, 17.5%, 17.8%} for AI, RA and LD in RGB coding, and luma BD-rate reduction of 22.7%, 15.7% and 10.3% for AI and RA and LD, respectively, in YCbCr coding. For lossless coding of the video sequences in the same category, bit rate reductions of 27.8%, 25.8% and 25.9% are achieved for AI, RA and LD, respectively, in RGB coding, and 28%, 21.5% and 19.4% for AI, RA and LD, respectively, in YCbCr coding.

# Algorithm description

## CTU/TU/PU partitioning

Same as HEVC range extension draft 6 [1].

## Inter-picture prediction

The inter prediction process in this proposal is the same as in Rext draft 6 [1] with only one exception: for lossy coding of 4:4:4 YUV and RGB video, the 8-tap interpolation filters are applied to all 3 color components. A flag is signaled in PPS to indicate this change, as presented in Section 3.

## Intra-picture prediction

Same as HEVC range extension draft 6 [1].

## In-loop filtering

### De-blocking filter

The de-blocking filtering process for all existing coding modes in this proposal is kept the same as that in Rext draft 6 [1]. For CUs coded using the new palette coding mode (described in Section 2.6.1), the de-blocking filter is not applied.

### SAO

Same as HEVC range extension draft 6 [1].

## Spatial transforms

Same as HEVC range extension draft 6 [1].

## Other specific compression tools for screen content

### Improved palette coding mode

The palette coding mode proposed in [3] is used in this response. The palette coding mode in [3] is signaled at the CU level. For each CU coded with palette mode, a palette table is generated by selecting a set of representative colors from that CU. At pixel locations where the colors already exist in the palette table, only the palette indices are transmitted. At pixel locations where the colors do not exist in the palette table, the color values are considered escape colors, and they are quantized (if lossy coding) and coded without prediction.

In this response, the following changes are proposed to further improve the coding efficiency of palette coding mode:

1. For palette table prediction, a representative color dictionary is proposed to store the representative colors from previously coded palette tables in previous CUs. Additionally, a palette table skip mode is proposed to directly re-use the representative color dictionary for the current CU.
2. For palette index coding, 3 improvements are proposed, including BWT based index grouping, transition mode, and palette index value mapping.
3. For escape color coding, an escape color prediction method is proposed and used in lossless coding.
4. At the encoder side, an improved palette table generation method is proposed to generate the representative colors in the palette table.

#### Palette table prediction

To reduce signaling overhead of the palette tables, palette table prediction based on representative color dictionary is proposed in this response. The proposed representative color dictionary stores the representative colors from previous CUs coded in palette mode. When coding the current CU, the representative color dictionary is used to predict the representative colors in the current palette table. After the current CU is coded, the representative color dictionary is updated by: 1) adding the representative colors in the current CU into the dictionary, and 2) removing old representative colors from the dictionary. A maximum dictionary size is maintained. Given the strong correlation between the colors from neighboring CUs, the latest representative colors from the current CU are placed at the beginning of the updated dictionary. In addition, pruning operation is performed to remove all redundant elements such that only unique representative colors are kept in the dictionary. Let denote the maximum size of the palette tables, the maximum size of the representative color dictionary is also set equal to . Figure 3 illustrates the proposed representative color dictionary update process, where the size of the representative color dictionary and the size of the current palette table are set to 8 and 5, respectively.

When predicting the palette table of the current CU using the representative color dictionary, one flag for each element in the palette table (i.e., each representative color) is sent to indicate if this element is predicted from the corresponding element in the representative color dictionary or not. In order to further reduce the signaling overhead of palette tables, a palette table skip mode is proposed to indicate that the representative color dictionary is directly re-used as the palette table of the current CU. The palette table skip mode is decided based on R-D cost optimization. As presented in Section 3, when palette table skip mode is chosen, only one flag is sent to represent the entire palette table of the current CU.



Figure 3. The update process of the representative color dictionary

#### Palette index coding

The palette index coding method in [3] is improved in this response. Our improvements include 3 parts: 1) BWT-based palette index grouping, 2) transition mode, and 3) palette index mapping.

##### BWT-based index grouping

The Burrows–Wheeler transform (BWT) [4] is used to increase the correlation between adjacent positions in the palette index map. The BWT is applied before encoding the palette index map. After scanning each position as a 1-D index sequence, this sequence is fed into the BWT process and the end position is also coded as BWT side-information (overhead) for decoder side reconstruction. Due to the side information needed for BWT reconstruction, a 1-bit flag for each CU is used to indicate if the BWT is performed or not.

An example of BWT process is given in Figure 4. Assume the input string is “BANANA”. After the first and second BWT encoding steps, the input string “BANANA” is rotated and sorted, as shown in Figure 4 (a). Then, the final column “NNBAAA” is extracted as the output string. Besides the output string, the end of block position also needs to be transmitted to allow perfect reconstruction at the decoder side. The decoding process contains two steps: 1) the transformed string is appended to the decoding table, and 2) the decoding table is sorted.

Step 1) and 2) are iterated k rounds, where k is the string length. At the last round (k-th round), the string with the end of block position (“A” in this example) at the end is the original string. In Figure 4(b), the decoding table is initialized to be empty. At the first round, the transformed string “NNBAAA” is appended to the empty decoding table, and after sorting, the first column of decoding table becomes “AAABNN”. At the second round, the string “NNBAAA” is appended to the decoding table again, and the decoding table is sorted. This process continues for 6 rounds. After 6 rounds, the 4th row of the decoding table is the decoded string, because its last block position “A” corresponds to the signaled end of block position (EOB = 3), . With the proposed BWT based index grouping method, the entropy coding process only needs to signal three characters, “N”, “B” and “A”, and three corresponding run values, 1, 0 and 2.



(a)



(b)

**Figure 4. An example of (a) BWT encoding and (b) BWT decoding. The “A” represents the end of string.**

##### Transition mode

The transition mode has been previously proposed in [5] for palette index coding. The conventional palette coding modes (e.g. run/copy mode) in [3] derive palette index from immediate left or top neighbor of the current position. In comparison, the transition mode in [5] allows the palette index of the current position to be copied from a previously occurring pattern of palette indices. In this response, the same transition mode as in [5] is implemented. To mitigate the overhead of using the transition mode, the transition mode is enabled only when the palette table size of current CU is larger than 14. Besides, the transition mode is not used in combination with the BWT based palette index grouping method, as proposed in 2.6.1.2.1.

##### Palette index value mapping

Palette index values are transmitted only when signaling run mode, and each index value is variable length coded. To reduce the magnitude of the coded palette index value, the palette index value is mapped before being coded.

Denote the last compared palette index (abridged as LCPI) value as , the current index value as , and the mapped index as . The mapped index value is calculated as follows:

The mapped index is then coded using variable length coding. At the decoder, the received index is compared to the last index . If , the decoded index value = . Note that, the LCPI value is not the same as the last palette index value. The LCPI position is calculated as follows:

,

where is denoted as the current CU width, and is defined as the target distance for copy index values in the transition mode.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| (a) | (b) | (c) |

**Figure 5. Three examples on how the last compared palette index (LCPI) value is derived. The run, copy and transition modes are marked in green, orange and blue, respectively. The uncoded palette indices are marked in white. The current coding position is marked in red dash, and the LCPI position is marked in red bold. (a) If the previous palette mode is run mode, then the LCPI is the previous position. (b) If the previous palette mode is copy mode, then the LCPI is the upper position. (c) If the previous palette mode is transition mode, then the LCPI is the position with the distance of 7 in front of the current position.**

In [3], the escape color index is set to the last (i.e., largest) index in the palette table. To reduce the magnitude of the index value of escape color pixels, in our implementation, the index value of escape color pixels is set to 1. The other entries in the palette table are shifted backward by 1 position. Mapping of the escape color index is shown in Figure 5.



**Figure 5. The escape color index mapping process**

#### Escape color prediction

To better code the escape colors, escape color prediction is proposed in this response. In the proposed method, a flag is signaled to indicate if every color component of the current escape color pixel is the same as the previous escape color pixel or not. And if not (flag is equal to 0), the current escape color value is predicted from an entry in the palette table. The index value of the table entry used for prediction, along with the prediction difference (including absolute difference and the sign bit), are signaled to the decoder. The readers are referred to Section 3 for detailed syntax.

In this response, the proposed escape color prediction method is applied to lossless coding only. For lossy coding, the escape color is coded directly without any prediction, in the same manner as in [3] . In our current encoder implementation, the palette table entry used for escape color prediction is the table entry with the minimal prediction error, where the prediction error is calculated by summing the prediction error of each color component.

#### Palette table generation (encoder only)

In this proposal, the palette table generation consists of two main steps: 1) representative color initialization, and 2) representative color refinement.

The first step, representative color initialization, is the same as in [3]. The histogram of the colors in the current CU is calculated to select an initial set of representative colors with highest frequency of occurrence, where is the palette table size. In the case of lossy coding, a quantization process is also employed to divide the whole color spectrum into separate quantization zones. Colors that fall in the same quantization zone as one of the initial representative colors will be quantized to that representative color. For all other colors, they will be coded as escape colors.

The initial representative colors derived in the first step may not give the smallest distortion for all the colors that fall in the same quantization zone. Therefore, we introduce a second step to refine the initial representative colors. Specifically, for each quantization zone, the refined representative color is calculated as the weighted average of all the colors within that quantization zone, as follows



where is the refined representative color,  is the i-th color in that quantization zone, and is the occurrence percentage of . This way, the average distortion between the refined representative color and the colors in the quantization zone it represents is minimized.

The quantization process and the representative color refinement process are only applied to lossy coding. For lossless coding, the simple histogram based selection is employed to generate the palette table, i.e., the first colors with highest percentages of occurrence are selected as the representative colors.

### Adaptive residue color space conversion

When coding in RGB color space, there are a lot of redundancies between three color components, which may be inefficient for video compression. Cross-component prediction (CCP) was adopted to HEVC range extension [1] to exploit such redundancy and improve coding efficiency. CCP uses the residue of G component to predict the residues of B and R components. Although CCP is effective in improving coding efficiency with relatively simple cross color component dependency, it cannot completely remove the correlation among RGB components.

For RGB video coding, we propose to apply adaptive color space conversion in the residual domain. Specifically, we propose to convert residuals from RGB to YCgCo color space before transform and quantization at the encoder. At the decoder side, after inverse quantization and inverse transform, the reconstructed residuals are converted from YCgCo back to RGB. For lossless coding, we use the YCgCo space described in Equations (2) and (3); and for lossy coding, we use the YCgCo space described in Equations (4) and (5).









The proposed adaptive residual color conversion is signaled at the CU level. A flag sps\_residual\_csc\_enabled\_flag is added in SPS to indicate if the adaptive residual color conversion is enabled at the sequence level. The proposed adaptive residual color conversion is only used for RGB lossy and lossless coding. At the CU level, a flag cu\_ycgco\_residue\_flag is added to signal if adaptive residual color conversion is on or off. In order to speed up encoding, the flag cu\_ycgco\_residue\_flag is checked only if there are non-zero reconstructed residual coefficients in the RGB color space in the CU. The readers are referred to Section 3 for detailed syntax.

The proposed adaptive residual color space conversion can be applied whether transform skip mode is on or off. And in the current implementation, the proposed method is applied only to inter prediction residuals and INTRA\_BC prediction residuals, but not to intra prediction residuals.



Figure 6. Encoder decision flow chart for of the proposed adaptive residual color conversion

## Quantization

Same as HEVC range extension draft 6 [1].

## Motion/mode/parameter coding

Same as HEVC range extension draft 6 [1].

### Merge/skip

Same as HEVC range extension draft 6 [1].

### Motion vector coding

Same as HEVC range extension draft 6 [1].

### Intra prediction mode coding

Same as HEVC range extension draft 6 [1].

### Other parameter coding

Same as HEVC range extension draft 6 [1].

## Entropy coding of residual / transform coefficients

Same as HEVC range extension draft 6 [1].

## High-layer syntax

Same as HEVC range extensions draft 6 [1] with the exceptions listed in section 3.

## Additional algorithmic description discussion topic #1

## Additional algorithmic description discussion topic #2

# Syntax and semantics description

All the proposed syntax and semantics changes are described based on the HEVC range extension draft 6. All changes are highlighted in yellow.

## Syntax

### Sequence parameter set syntax

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor |
| **sps\_video\_parameter\_set\_id** | u(4) |
| **……** |  |
| **sps\_extension\_present\_flag** | u(1) |
| if( sps\_extension\_present\_flag ) { |  |
| for( i = 0; i < 1; i++ ) |  |
| **sps\_extension\_flag**[ i ] | u(1) |
| **sps\_extension\_7bits** | u(7) |
| if( sps\_extension\_flag[ 0 ] ) { |  |
| **transform\_skip\_rotation\_enabled\_flag** | u(1) |
| **transform\_skip\_context\_enabled\_flag** | u(1) |
| **intra\_block\_copy\_enabled\_flag** | u(1) |
| **implicit\_rdpcm\_enabled\_flag** | u(1) |
| **explicit\_rdpcm\_enabled\_flag** | u(1) |
| **extended\_precision\_processing\_flag** | u(1) |
| **intra\_smoothing\_disabled\_flag** | u(1) |
| **high\_precision\_offsets\_enabled\_flag** | u(1) |
| **fast\_rice\_adaptation\_enabled\_flag** | u(1) |
| **cabac\_bypass\_alignment\_enabled\_flag** | u(1) |
| **sps\_residual\_csc\_enabled\_flag** | u(1) |
| **sps\_palette\_mode\_enabled\_flag** | u(1) |
| } |  |
| if( sps\_extension\_7bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **sps\_extension\_data\_flag** | u(1) |
| } |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

### Picture parameter set syntax

|  |  |
| --- | --- |
| pic\_parameter\_set\_rbsp( ) { | Descriptor |
| **pps\_pic\_parameter\_set\_id** | ue(v) |
| **pps\_seq\_parameter\_set\_id** | ue(v) |
| **……** |  |
| **pps\_extension\_present\_flag** | u(1) |
| if( pps\_extension\_present\_flag ) { |  |
| for( i = 0; i < 1; i++ ) |  |
| **pps\_extension\_flag**[ i ] | u(1) |
| **pps\_extension\_7bits** | u(7) |
| } |  |
| if( pps\_extension\_flag[ 0 ] ) { |  |
| if( transform\_skip\_enabled\_flag ) |  |
| **log2\_max\_transform\_skip\_block\_size\_minus2** | ue(v) |
| **cross\_component\_prediction\_enabled\_flag** | u(1) |
| **chroma\_qp\_adjustment\_enabled\_flag** | u(1) |
| if( chroma\_qp\_adjustment\_enabled\_flag ) { |  |
| **diff\_cu\_chroma\_qp\_adjustment\_depth** | ue(v) |
| **chroma\_qp\_adjustment\_table\_size\_minus1** | ue(v) |
| for( i = 0; i <= chroma\_qp\_adjustment\_table\_size\_minus1; i++ ) { |  |
| **cb\_qp\_adjustment**[ i ] | se(v) |
| **cr\_qp\_adjustment**[ i ] | se(v) |
| } |  |
| } |  |
| **log2\_sao\_offset\_scale\_luma** | ue(v) |
| **log2\_sao\_offset\_scale\_chroma** | ue(v) |
| **pps\_chroma\_default\_filter\_disabled\_flag** | u(1) |
| } |  |
| if( pps\_extension\_7bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **pps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

### Coding unit syntax

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize ) { | Descriptor |
| **……** |  |
| if( cu\_skip\_flag[ x0 ][ y0 ] ) |  |
| prediction\_unit( x0, y0, nCbS, nCbS ) |  |
| else { |  |
| if( intra\_block\_copy\_enabled\_flag ) |  |
| **intra\_bc\_flag**[ x0 ][ y0 ] | ae(v) |
| if( slice\_type != I && !intra\_bc\_flag[ x0 ][ y0 ] ) |  |
| **pred\_mode\_flag** | ae(v) |
| if( sps\_palette\_mode\_enabled\_flag && CuPredMode[ x0 ][ y0 ] == MODE\_INTRA && !intra\_bc\_flag[ x0 ][ y0 ]) |  |
| **palette\_mode\_flag[ x0 ][ y0 ]** | ae(v) |
| if( (CuPredMode[ x0 ][ y0 ] != MODE\_INTRA | | intra\_bc\_flag[ x0 ][ y0 ] | |   log2CbSize = = MinCbLog2SizeY) && !palette\_mode\_flag[ x0 ][ y0 ] ) |  |
| **part\_mode** | ae(v) |
| if( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ) { |  |
| if( PartMode = = PART\_2Nx2N && pcm\_enabled\_flag &&   !intra\_bc\_flag[ x0 ][ y0 ] && !palette\_mode\_flag[ x0 ][ y0 ] &&  log2CbSize >= Log2MinIpcmCbSizeY &&  log2CbSize <= Log2MaxIpcmCbSizeY ) |  |
| **pcm\_flag**[ x0 ][ y0 ] | ae(v) |
| if( pcm\_flag[ x0 ][ y0 ] ) { |  |
| while( !byte\_aligned( ) ) |  |
| **pcm\_alignment\_zero\_bit** | f(1) |
| pcm\_sample( x0, y0, log2CbSize ) |  |
| } else if( intra\_bc\_flag[ x0 ][ y0 ] ) { |  |
| mvd\_coding( x0, y0, 2) |  |
| if( PartMode = = PART\_2NxN ) |  |
| mvd\_coding( x0, y0 + ( nCbS / 2 ), 2) |  |
| else if( PartMode = = PART\_Nx2N ) |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0, 2) |  |
| else if( PartMode = = PART\_NxN ) { |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0, 2) |  |
| mvd\_coding( x0, y0 + ( nCbS / 2 ), 2) |  |
| mvd\_coding( x0 + ( nCbS / 2 ), y0 + ( nCbS / 2 ), 2) |  |
| } |  |
| }else if( palette\_mode\_flag[ x0 ][ y0 ] ) |  |
| palette\_coding ( x0, y0, nCbS, nCbS ) |  |
| else { |  |
| pbOffset = ( PartMode = = PART\_NxN ) ? ( nCbS / 2 ) : nCbS |  |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| **prev\_intra\_luma\_pred\_flag**[ x0 + i ][ y0 + j ] | ae(v) |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| if( prev\_intra\_luma\_pred\_flag[ x0 + i ][ y0 + j ] ) |  |
| **mpm\_idx**[ x0 + i ][ y0 + j ] | ae(v) |
| Else |  |
| **rem\_intra\_luma\_pred\_mode**[ x0 + i ][ y0 + j ] | ae(v) |
| if( ChromaArrayType = = 3 ) |  |
| for( j = 0; j < nCbS; j = j + pbOffset ) |  |
| for( i = 0; i < nCbS; i = i + pbOffset ) |  |
| **intra\_chroma\_pred\_mode**[ x0 + i ][ y0 + j ] | ae(v) |
| else if( ChromaArrayType != 0 ) |  |
| **intra\_chroma\_pred\_mode**[ x0 ][ y0 ] | ae(v) |
| } |  |
| } else { |  |
| **……** |  |
| } |  |
| if( !pcm\_flag[ x0 ][ y0 ] ) { |  |
| if( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA &&   !( PartMode = = PART\_2Nx2N && merge\_flag[ x0 ][ y0 ] ) | |   ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA && intra\_bc\_flag[ x0 ][ y0 ] ) ) |  |
| **rqt\_root\_cbf** | ae(v) |
| if( rqt\_root\_cbf ) { |  |
| if(sps\_residual\_csc\_enabled\_flag) |  |
| **cu\_ycgco\_residual\_flag** | ae(v) |
| MaxTrafoDepth = ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA ?   ( max\_transform\_hierarchy\_depth\_intra + IntraSplitFlag ) :   max\_transform\_hierarchy\_depth\_inter ) |  |
| transform\_tree( x0, y0, x0, y0, log2CbSize, 0, 0 ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

### Palette coding syntax

|  |  |
| --- | --- |
| palette\_coding ( x0, y0, nCbWidth, nCbHeight ) { | Descriptor |
| **palette\_table\_skipped\_flag[ x0 ][ y0 ]** | ae(v) |
| if( !palette\_table\_skipped\_flag[ x0 ][ y0 ] ) { |  |
| **palette\_size\_minus1[ x0][ y0 ]** | ae(v) |
| for( n = 0; n < RepColorDictionarySize; n ++ ) |  |
| **used\_by\_curr\_palette\_table\_pred[ n ]** | ae(v) |
| for( n = numPaletteColorPred; n < PaletteTableSize; n++) { |  |
| for( cIdx = 0; cIdx < 3; cIdx++) |  |
| **palette\_color\_value[ x0 ][ y0 ][n][ cIdx ]** | ae(v) |
| } |  |
| } |  |
| **palette\_bwt\_flag[ x0 ][ y0 ]** | ae(v) |
| if( palette\_bwt\_flag[ x0 ][ y0 ] ) |  |
| **palette\_bwt\_eob\_pos[ x0 ][ y0 ]** | ae(v) |
| scanPos = 0 |  |
| while ( scanPos < nCbWidth \* nCbHeight ) { |  |
| **palette\_mode[ x0 ][ y0 ][ scanPos ]** | ae(v) |
| if( palette\_mode == RUN\_MODE ) |  |
| **mapped\_palette\_index[ x0 ][ y0 ][ scanPos ]** | ae(v) |
| **run[ x0 ][ y0 ][ scanPos ]** | ae(v) |
| scanPos += run |  |
| scanPos ++ |  |
| } |  |
| scanPos = 0 |  |
| while ( scanPos < nCbWidth \* nCbHeight ) { |  |
| if ( palette\_index[ x0 ][ y0 ][ scanPos ] == PaletteTableSize ) { |  |
| if ( cu\_transquant\_bypass\_flag ) { |  |
| **escape\_color\_reuse\_flag[ x0 ][ y0 ][ scanPos ]** | ae(v) |
| if( !escape\_color\_pred\_flag[ x0 ][ y0 ][ scanPos ] ) { |  |
| **pred\_palette\_table\_index[ x0 ][ y0 ][ scanPos ]** | ae(v) |
| for( cIdx = 0; cIdx < 3; cIdx ++ ) { |  |
| **escape\_color\_residual\_abs\_level[ x0 ][ y0 ][ scanPos ][ cIdx ]** | ae(v) |
| if( escape\_color\_resi\_abs\_level[ x0 ][ y0 ][ scanPos ][ cIdx ] ) |  |
| **escape\_color\_residual\_sign\_flag[ x0 ][ y0 ][ scanPos ][ cIdx ]** | ae(v) |
| } |  |
| } |  |
| } else { |  |
| for( cIdx = 0; cIdx < 3; cIdx ++ ) |  |
| **escape\_color\_value[ x0 ][ y0 ][ scanPos][ cIdx ]** | ae(v) |
| } |  |
| } |  |
| scanPos ++ |  |
| } |  |
| } |  |

## Semantics

### Sequence parameter set semantics

**sps\_residual\_csc\_enabled\_flag** equal to 1 specifies that residual color space conversion coding tool is enabled in the CVS. sps\_residual\_csc\_enabled\_flag equal to 0 specifies that the residual color space conversion is disabled in the CVS. When not presented, the value of sps\_residual\_csc\_enabled\_flag is inferred to be 0.

**sps\_palette\_mode\_enabled\_flag** equal to 1 specifies that palette coding tool is enabled in the CVS. sps\_palette\_mode\_enabled\_flag equal to 0 specifies that palette coding tool is disabled in the CVS. When not presented, the value of sps\_palette\_mode\_enabled\_flag is inferred to be 0.

### Picture parameter set semantics

**pps\_chroma\_default\_filter\_disabled\_flag** equal to 1 specifies that the luma interpolation filters are applied to interpolate the chroma components at fractional pixel positions in the slices referring to this PPS. pps\_chroma\_default\_filter\_disabled\_flag equal to 0 specifies that the chroma interpolation filters are not changed for the slices referring to this PPS. When not presented, the value of pps\_chroma\_default\_filter\_disabled\_flag is inferred to be 0.

### Coding unit semantics

**palette\_mode\_flag**[ x0 ][ y0 ] equal to 1 specifies that the current coding unit is coded in palette mode. palette\_mode\_flag[ x0 ][ y0 ] equals to 0 specifies the current coding unit is not coded in palette mode. The array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the current coding block relative to the top-left luma sample of the picture. When palette\_mode\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**cu\_ycgco\_residual\_flag** specifies whether the residue of the current coding unit is represented in the YCgCo color space or not. When cu\_ycgco\_residual\_flag is not present, it is inferred to be equal to 0.

### Palette coding semantics

In the following semantics, the array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

**palette\_table\_skipped\_flag**[ x0 ][ y0 ] equal to 1 specifies that for the current coding unit, when palette coding mode is applied, no more syntax elements of palette table signaling are parsed after palette\_table\_skipped\_flag[ x0 ][ y0 ]. When palette\_table\_skipped\_flag[ x0 ][ y0 ] equal to 0 specifies that palette table signaling is not skipped.

**palette\_size\_minus1**[ x0][ y0 ] plus 1 specifies the number of representative color elements in palette table of the current coding unit.

The variable PaletteTableSize is derived as follows:

PaletteTableSize = palette\_size\_minus1[ x0 ][ y0 ] + 1

**used\_by\_curr\_palette\_table\_pred**[ n ] equal to 1 specifies that the n-th element of representative-color dictionary is used to predict one element of the palette table of the current coding unit. used\_by\_curr\_palette\_table\_pred[ n ] equal to 0 specifies that the n-th element of representative-color dictionary is not used to predict the palette table of the current coding unit.

**palette\_color\_value**[ x0 ][ y0 ][ n ][ cIdx ] specifies the color value of the n-th representative-color dictionary for color component cIdx.

Depending on the used\_by\_curr\_palette\_table\_pred[ n ] and palette\_color\_value[ x0 ][ y0 ][ n ][ cIdx ], the variables PaletteTableColors[ x0 ][ y0 ][ i ] and NumPaletteColorPred are derived as follows:

numPaletteColorPred = 0

for( i = 0; i < RepColorDictionarySize; i++) {

if( used\_by\_curr\_palette\_table\_pred[ i ] ) {

for( cIdx = 0; cIdx < 3; cIdx ++ )

PaletteTableColors[ x0 ][ y0 ][ numPaletteColorPred ][cIdx] = RepColorDictionary[ i ][ cIdx ]

numPaletteColorPred ++

}

}

n = 0

for( i = numPaletteColorPred; i < PaletteTableSize; i++ ) {

for( cIdx = 0; cIdx < 3; cIdx ++)

PaletteTableColors[ x0][ y0 ][ i ][ cIdx ] = palette\_color\_value[ n ][ cIdx ]

n ++

}

**palette\_bwt\_flag**[ x0 ][ y0 ] equals to 1 whether the BWT-based palette index grouping process is applied to the associated palette indices of the current CU or not.

**palette\_bwt\_eob\_pos**[ x0 ][ y0 ] specifies the end of block position used in the BWT-based palette index grouping process. When palette\_bwt\_eob\_pos[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

**palette\_mode**[ x0 ][ y0 ][ scanPos ] specifies the palette index coding mode at the scan position scanPos of the current coding unit.

Depending on palette\_bwt\_flag[ x0 ][ y0 ] and palette\_mode[ x0 ][ y0 ][ scanPos ], the variable PaletteIndexCodingMode[ x0 ][ y0 ][ scanPos ] is derived as defined in Table 1.

**Table 1 Name association of palette index coding mode**

|  |  |  |
| --- | --- | --- |
| **palette\_bwt\_flag[ x0 ][ y0 ]** | **palette\_mode[ x0 ][ y0 ][scanPos]** | **PaletteIndexCodingMode[ x0 ] [ y0 ][ scanPos ]** |
| 0 | 0 | RUN\_MODE |
| 1 | COPY\_MODE |
| 1 | 0 | RUN\_MODE |
| 1 | COPY\_MODE |
| 2 | TRANSITION\_MODE |

**mapped\_palette\_idx**[ x0 ][ y0 ][ scanPos ] represents the mapped index value at the scan position scanPos of the current coding unit.

Depending on mapped\_palette\_index[ x0 ][ y0 ][ scanPos ], the variable PaletteIndexMap[ x0 ][ y0 ][ scanPos ], i.e., the palette index at the scan position scanPos, is derived as follows:

The variable RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos ] is derived as follows:

* If mapped\_palette\_index[ x0 ][ y0 ][ scanPos ] is equal to 0, RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos ] is set equal to 0.
* Otherwise, if mapped\_palette\_index[ x0 ][ y0 ][ scanPos ] is equal to 1, RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos ] is set equal to palette\_size\_minus1[ x0 ][ y0 ].
* Otherwise, RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos ] is set equal to mapped\_palette\_index[ x0 ][ y0 ][ scanPos ] + 1.

The variable TargetDist is derived as follows:

* If PaletteIndexCodingMode[ x0 ] [ y0 ][ scanPos ] is equal to RUN\_MODE, TargetDist is set equal to 1.
* Otherwise, if PaletteIndexCodingMode[ x0 ] [ y0 ][ scanPos ] is equal to COPY\_MODE, TargetDist is set equal to nCbWidth.
* Otherwise, i.e., PaletteIndexCodingMode[ x0 ] [ y0 ][ scanPos ] is equal to TRANSITION\_MODE, TargetDist is set equal to TransDist.

The following is then applied:

* If scanPos is equal to 0, PaletteIndexMap[ x0 ][ y0 ][ scanPos ] is set equal to RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos ].
* Otherwise, if RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos ] is no smaller than RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos - TargetDist ], PaletteIndexMap[ x0 ][ y0 ][ scanPos ] is set equal to RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos ] + 1.
* Otherwise, PaletteIndexMap[ x0 ][ y0 ][ scanPos ] is set equal to RelocatedPaletteIndexMap[ x0 ][ y0 ][ scanPos ].

**run**[ x0 ][ y0 ][ scanPos ] specifies the number of consecutive positions that have the same palette index value as that of the scan position scanPos.

Depending on PaletteIndexCodingMode[ x0 ] [ y0 ][ scanPos ] and run[ x0 ][ y0 ][ scanPos ], the variable PaletteIndexMap[ x0 ][ y0 ][ i ], where i is from scanPos to scanPos + run[ x0 ][ y0 ][ scanPos ] – 1, is derived as follows:

* If PaletteIndexCodingMode[ x0 ] [ y0 ][ scanPos ] is equal to RUN\_MODE, the following is applied:

for( i = scanPos + 1; i < scanPos + run[ x0 ][ y0 ][ scanPos ]; i++) {

PaletteIndexMap [ x0 ] [ y0 ] [ i ] = PaletteIndexMap [ x0 ] [ y0 ] [ i - 1 ]

PaletteTransTable[ PaletteIndexMap [ x0 ] [ y0 ] [ i ] ][ 1 ] = paletteTransTable[ PaletteIndexMap [ x0 ] [ y0 ] [ i ] ][ 0 ]

PaletteTransTable[ PaletteIndexMap [ x0 ][ y0 ][ i ] ][ 0 ] = i

}

* Otherwise, if PaletteIndexCodingMode[ x0 ] [ y0 ][ scanPos ] is equal to COPY\_MODE, the following is applied:

for( i = scanPos; i < scanPos + run[ x0 ][ y0 ][ scanPos ]; i++) {

PaletteIndexMap[ x0 ][ y0 ][ i ] = PaletteIndexMap[ x0 ][ y0 ][ i – nCbS ]

PaletteTransTable[ PaletteIndexMap [ x0 ][ y0 ][ i ] ][ 1 ] = paletteTransTable[ PaletteIndexMap [ x0 ][ y0 ][ i ] ][ 0 ]

PaletteTransTable[PaletteIndexMap [ x0 ][ y0 ][ i ] ][ 0 ] = i

}

* Otherwise, i.e., PaletteIndexCodingMode[ x0 ] [ y0 ][ scanPos ] is equal to TRANSITION\_MODE, the following is applied:

TransDist = scanPos – 1 – PaletteTransTable[ PaletteIndexMap [ x0 ][ y0 ][ scanPos – 1 ] ][ 1 ]

for( i = scanPos; i < scanPos + run[ x0 ][ y0 ][ scanPos ]; i++ ) {

PaletteIndexMap [ x0 ][ y0 ][ i ] = PaletteIndexMap [ x0 ][ y0 ][ i – TransDist ]

PaletteTransTable[ PaletteIndexMap [ x0 ][ y0 ][ i ] ][ 1 ] = paletteTransTable[ PaletteIndexMap [ x0 ][ y0 ][ i ] ][ 0 ]

PaletteTransTable[PaletteIndexMap [ x0 ][ y0 ][ i ] ][ 0 ] = i

}

**escape\_color\_reuse\_flag**[ x0 ][ y0 ][ scanPos ] equal to 1 specifies that the escape color at the scan position scanPos is set equal to the escape color coded most recently. escape\_color\_reuse\_flag[ x0 ][ y0 ][ scanPos ] equal to 0 specifies the scan position scanPos is not equal to the escape color coded most recently.

**pred\_palette\_table\_index**[ x0 ][ y0 ][ scanPos ] specifies the entry of the palette table of the current coding unit that is used to predict the escape color at the scan position scanPos.

**escape\_color\_resi\_abs\_level**[ x0 ][ y0 ][ scanPos ][ cIdx ] specifies the absolute difference between the escape color at the scan position scanPos of the current coding unit and its prediction for the color component cIdx.

**escape\_color\_resi\_sign\_flag**[ x0 ][ y0 ][ scanPos ][ cIdx ] specifies the sign of the difference between the escape color at the scan position scanPos of the current coding unit and its prediction for the color component cIdx.

**escape\_color\_value**[ x0 ][ y0 ][ scanPos ][ cIdx ] specifies the value of the escape color at the scan position scanPos of the current coding unit for color component cIdx.

# Compression performance discussion

The compression performance is reported using the test conditions and the measurement metrics as required by the CfP [2]. The file sizes of all 360 bit-streams (including lossless coding and lossy coding) satisfy the CfP requirement that they not exceed the file sizes of the anchor bit-streams. Summary of compression performance of the proposed system will be discussed in this section. Readers are referred to the accompanying spreadsheets for further detail.

## Lossy test condition

Table 2 shows the average BD-rate improvement of the proposed solution using lossy coding configuration. For RGB sequences, the average BD-rate reductions for {G, B, R} components of the proposed solution are {16.3%, 15.9%, 15.8%}, {13.4%, 13.2%, 13.1%} and {13.8%, 13.6%, 13.4%} for AI, RA and LD respectively. For YCbCr sequences, the average luma BD-rate reductions are 13.2%, 9.2% and 7% for AI, RA and LD respectively.

When only considering the typical screen content sequences (the 1080p and 720p video sequences in the category of “text & graphics with motion”), for the RGB format, the corresponding average BD-rate reductions for {G, B, R} components are {28.9%, 28.4%, 28.3%}, {21.7%, 21.1%, 21.3%} and {18.2%, 17.5%, 17.8%} for AI, RA and LD respectively; for the YCbCr format, the average BD-rate reductions for the luma component are 22.7%, 15.7% and 10.3% for AI, RA and LD respectively.

**Table 2 BD-rate reductions of the proposed solution under lossy coding configuration**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **All Intra** | | |
|  | Y | U | V |
| RGB, text & graphics with motion, 1080p | -35.2% | -35.9% | -35.1% |
| RGB, text & graphics with motion,720p | -22.5% | -20.9% | -21.4% |
| RGB, mixed content, 1440p | -10.6% | -9.4% | -9.3% |
| RGB, mixed content, 1080p | -12.6% | -12.4% | -11.9% |
| RGB, Animation, 720p | -0.7% | -1.1% | -1.1% |
| YUV, text & graphics with motion, 1080p | -30.3% | -34.0% | -34.0% |
| YUV, text & graphics with motion,720p | -15.1% | -21.7% | -24.5% |
| YUV, mixed content, 1440p | -9.3% | -14.2% | -14.2% |
| YUV, mixed content, 1080p | -10.8% | -13.9% | -13.7% |
| YUV, Animation, 720p | -0.6% | -1.3% | -1.1% |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Random Access** | | |
|  | Y | U | V |
| RGB, text & graphics with motion, 1080p | -22.6% | -23.7% | -23.1% |
| RGB, text & graphics with motion,720p | -20.8% | -18.4% | -19.4% |
| RGB, mixed content, 1440p | -9.4% | -7.7% | -7.8% |
| RGB, mixed content, 1080p | -12.0% | -12.8% | -12.0% |
| RGB, Animation, 720p | -2.3% | -3.7% | -3.3% |
| YUV, text & graphics with motion, 1080p | -17.9% | -24.5% | -24.3% |
| YUV, text & graphics with motion,720p | -13.4% | -19.8% | -24.1% |
| YUV, mixed content, 1440p | -7.2% | -14.2% | -14.1% |
| YUV, mixed content, 1080p | -7.3% | -11.4% | -11.1% |
| YUV, Animation, 720p | -0.4% | -1.4% | -1.3% |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Low delay B** | | |
|  | Y | U | V |
| RGB, text & graphics with motion, 1080p | -18.1% | -18.9% | -18.3% |
| RGB, text & graphics with motion,720p | -18.3% | -16.0% | -17.2% |
| RGB, mixed content, 1440p | -13.7% | -11.6% | -11.7% |
| RGB, mixed content, 1080p | -13.6% | -14.0% | -13.1% |
| RGB, Animation, 720p | -5.3% | -7.5% | -6.7% |
| YUV, text & graphics with motion, 1080p | -11.4% | -17.1% | -17.1% |
| YUV, text & graphics with motion,720p | -9.1% | -14.1% | -18.4% |
| YUV, mixed content, 1440p | -8.2% | -15.6% | -15.8% |
| YUV, mixed content, 1080p | -6.0% | -9.0% | -8.5% |
| YUV, Animation, 720p | -0.5% | -0.6% | -0.6% |

## Mathematically lossless test condition

Table 3 shows the average bit-rate savings of the proposed solution using lossless coding configuration. Compared to the lossless anchors, the average bit-rate savings of the proposed solution are 14.8%, 16.2% and 16.7% for AI, RA and LD, respectively, in RGB coding, and 13.8%, 10.2% and 9.3% for AI, RA and LD, respectively, in YCbCr coding.

When only considering the typical screen content sequences (the 1080p and 720p video sequences in the category of “text & graphics with motion”) the average bit-rate savings of the proposed solution are 27.8%, 25.8% and 25.9% for AI, RA and LD respectively in RGB coding, and28%, 21.5% and 19.4% for AI, RA and LD respectively, in YCbCr coding.

**Table 3 Bit-rate savings of the proposed system under lossless coding configuration**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **AI** | | | |
|  | Bit-rate saving (Total) | Bit-rate saving (Average) | Bit-rate saving (Min) | Bit-rate saving (Max) |
|  |
| RGB, text & graphics with motion, 1080p | 34.5% | 36.2% | 26.1% | 49.0% |
| RGB, text & graphics with motion,720p | 18.4% | 19.3% | 2.3% | 36.9% |
| RGB, mixed content, 1440p | 6.1% | 6.4% | 5.4% | 7.3% |
| RGB, mixed content, 1080p | 11.4% | 10.3% | 7.6% | 13.0% |
| RGB, Animation, 720p | 1.6% | 1.6% | 1.6% | 1.6% |
| YUV, text & graphics with motion, 1080p | 35.0% | 36.9% | 22.9% | 51.9% |
| YUV, text & graphics with motion,720p | 17.0% | 19.1% | 1.5% | 42.7% |
| YUV, mixed content, 1440p | 4.3% | 4.7% | 2.6% | 6.8% |
| YUV, mixed content, 1080p | 8.6% | 7.9% | 6.2% | 9.5% |
| YUV, Animation, 720p | 0.4% | 0.4% | 0.4% | 0.4% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **RA** | | | |
|  | Bit-rate saving (Total) | Bit-rate saving (Average) | Bit-rate saving (Min) | Bit-rate saving (Max) |
|  |
| RGB, text & graphics with motion, 1080p | 24.8% | 34.4% | 23.8% | 50.2% |
| RGB, text & graphics with motion,720p | 8.7% | 17.1% | 5.7% | 36.4% |
| RGB, mixed content, 1440p | 8.9% | 8.9% | 8.3% | 9.4% |
| RGB, mixed content, 1080p | 13.3% | 11.5% | 9.5% | 13.5% |
| RGB, Animation, 720p | 9.0% | 9.0% | 9.0% | 9.0% |
| YUV, text & graphics with motion, 1080p | 17.6% | 28.5% | 16.1% | 44.8% |
| YUV, text & graphics with motion,720p | 5.0% | 14.4% | 1.5% | 41.6% |
| YUV, mixed content, 1440p | 1.9% | 1.9% | 1.6% | 2.2% |
| YUV, mixed content, 1080p | 6.7% | 4.5% | 2.0% | 7.0% |
| YUV, Animation, 720p | 1.8% | 1.8% | 1.8% | 1.8% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **LB** | | | |
|  | Bit-rate saving (Total) | Bit-rate saving (Average) | Bit-rate saving (Min) | Bit-rate saving (Max) |
|  |
| RGB, text & graphics with motion, 1080p | 25.1% | 34.5% | 24.4% | 50.7% |
| RGB, text & graphics with motion,720p | 9.1% | 17.2% | 6.2% | 34.4% |
| RGB, mixed content, 1440p | 10.0% | 10.0% | 9.5% | 10.5% |
| RGB, mixed content, 1080p | 13.3% | 12.0% | 10.4% | 13.5% |
| RGB, Animation, 720p | 9.9% | 9.9% | 9.9% | 9.9% |
| YUV, text & graphics with motion, 1080p | 16.3% | 25.8% | 15.3% | 39.1% |
| YUV, text & graphics with motion,720p | 4.3% | 12.9% | 1.6% | 38.7% |
| YUV, mixed content, 1440p | 1.9% | 1.8% | 1.3% | 2.4% |
| YUV, mixed content, 1080p | 6.7% | 4.2% | 1.5% | 7.0% |
| YUV, Animation, 720p | 1.8% | 1.8% | 1.8% | 1.8% |

# Complexity analysis

## Encoding time and measurement methodology

Table 4 and Table 5 shows the relative encoding time for each sequence in both RGB and YCbCr formats when compared to the anchor under the lossy and lossless coding configurations respectively. The encoding time of the anchor and the proposed one is measured using the Linux cluster system as specified in Section 5.3.

**Table 4 Average relative encoding time of each sequence in comparison to the anchor under lossy coding configuration**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Color format** | **Sequence** | **AI** | **RA** | **LD** |
| RGB | Flyinggraphicstext | 2.18 | 2.04 | 1.85 |
| Desktop | 2.14 | 2.29 | 1.86 |
| Console | 2.01 | 2.18 | 1.90 |
| WebBrowsing | 2.92 | 2.35 | 2.45 |
| Map | 2.91 | 2.16 | 2.35 |
| Programming | 2.62 | 2.02 | 2.43 |
| SlideShow | 4.96 | 3.27 | 3.28 |
| BasketballScreen | 2.86 | 2.16 | 3.12 |
| MissionControlClip2 | 2.30 | 1.90 | 2.38 |
| MissionControlClip3 | 2.14 | 2.10 | 1.57 |
| SocialNetworkMap | 2.20 | 1.67 | 1.71 |
| Robot | 1.98 | 1.61 | 1.35 |
| YCbCr | Flyinggraphicstext | 1.87 | 1.07 | 1.06 |
| Desktop | 2.04 | 1.39 | 1.20 |
| Console | 1.80 | 1.24 | 1.11 |
| WebBrowsing | 2.64 | 1.61 | 1.38 |
| Map | 2.53 | 1.50 | 1.30 |
| Programming | 2.28 | 1.36 | 1.23 |
| SlideShow | 4.24 | 2.26 | 1.68 |
| BasketballScreen | 2.55 | 1.64 | 1.39 |
| MissionControlClip2 | 2.73 | 1.60 | 1.35 |
| MissionControlClip3 | 2.56 | 1.49 | 1.31 |
| SocialNetworkMap | 2.42 | 1.19 | 1.12 |
| Robot | 2.43 | 1.17 | 1.08 |

**Table 5 Average relative encoding time of each sequence in comparison to the anchor under lossless coding configuration**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Color format** | **Sequence** | **AI** | **RA** | **LD** |
| RGB | Flyinggraphicstext | 3.43 | 1.70 | 2.12 |
| Desktop | 2.35 | 2.17 | 1.35 |
| Console | 2.16 | 1.37 | 1.25 |
| WebBrowsing | 3.11 | 1.76 | 1.61 |
| Map | 3.05 | 1.81 | 1.55 |
| Programming | 2.83 | 1.75 | 1.66 |
| SlideShow | 4.20 | 1.96 | 1.86 |
| BasketballScreen | 4.53 | 2.41 | 2.38 |
| MissionControlClip2 | 5.01 | 2.17 | 2.03 |
| MissionControlClip3 | 4.17 | 2.52 | 2.03 |
| SocialNetworkMap | 3.88 | 2.39 | 2.31 |
| Robot | 2.97 | 1.84 | 1.86 |
| YCbCr | Flyinggraphicstext | 2.17 | 1.23 | 1.19 |
| Desktop | 2.28 | 1.47 | 1.29 |
| Console | 2.08 | 1.25 | 1.16 |
| WebBrowsing | 2.90 | 1.77 | 1.55 |
| Map | 2.81 | 1.66 | 1.51 |
| Programming | 2.48 | 1.32 | 1.30 |
| SlideShow | 4.26 | 1.96 | 1.70 |
| BasketballScreen | 5.48 | 2.95 | 2.69 |
| MissionControlClip2 | 5.61 | 2.67 | 2.19 |
| MissionControlClip3 | 3.68 | 2.71 | 1.79 |
| SocialNetworkMap | 3.55 | 1.67 | 1.76 |
| Robot | 2.28 | 1.24 | 1.19 |

## Decoding time and measurement methodology and comparison vs. anchor bitstreams decoded by HM

Table 6 and Table 7 shows the relative decoding time for each sequence in both RGB and YCbCr formats when compared to the anchor under the lossy and lossless coding configuration. The decoding time of the anchor and the proposed one is measured using the Linux cluster system as specified in Section 5.3. For lossy coding configuration, the average decoding time of the proposed solution is around 86%, 86% and 89% of that of the anchor, for AI, RA and LD respectively. When it comes to lossless coding, the corresponding relative decoding time of the proposed solution is 83%, 86% and 83% for AI, RA, and LD, respectively.

**Table 6 Average relative decoding time of each sequence in comparison to the anchor using lossy coding configuration**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Color format** | **Sequence** | **AI** | **RA** | **LD** |
| RGB | Flyinggraphicstext | 1.12 | 0.87 | 0.88 |
| Desktop | 1.25 | 1.00 | 1.13 |
| Console | 1.28 | 1.08 | 1.09 |
| WebBrowsing | 1.17 | 1.02 | 0.78 |
| Map | 1.11 | 0.92 | 0.91 |
| Programming | 1.08 | 0.99 | 0.95 |
| SlideShow | 1.10 | 1.00 | 0.74 |
| BasketballScreen | 1.11 | 0.99 | 0.75 |
| MissionControlClip2 | 1.32 | 1.10 | 0.97 |
| MissionControlClip3 | 1.35 | 0.99 | 1.22 |
| SocialNetworkMap | 1.44 | 1.01 | 1.00 |
| Robot | 1.35 | 1.01 | 1.14 |
| YCbCr | Flyinggraphicstext | 1.23 | 1.27 | 1.08 |
| Desktop | 1.26 | 1.47 | 1.34 |
| Console | 1.27 | 1.55 | 1.30 |
| WebBrowsing | 1.09 | 1.36 | 1.33 |
| Map | 1.09 | 1.31 | 1.27 |
| Programming | 1.10 | 1.33 | 1.20 |
| SlideShow | 1.10 | 1.29 | 1.26 |
| BasketballScreen | 1.04 | 1.29 | 1.48 |
| MissionControlClip2 | 1.04 | 1.32 | 1.64 |
| MissionControlClip3 | 1.05 | 1.47 | 1.44 |
| SocialNetworkMap | 1.21 | 1.26 | 1.27 |
| Robot | 0.88 | 1.27 | 1.25 |

**Table 7 Average relative decoding time of each sequence in comparison to the anchor using lossless coding configuration**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Color format** | **Sequence** | **AI** | **RA** | **LD** |
| RGB | Flyinggraphicstext | 0.87 | 0.60 | 0.77 |
| Desktop | 0.65 | 0.93 | 0.56 |
| Console | 0.73 | 0.75 | 0.75 |
| WebBrowsing | 0.73 | 0.77 | 0.80 |
| Map | 0.65 | 0.78 | 0.72 |
| Programming | 0.79 | 0.80 | 0.81 |
| SlideShow | 0.87 | 0.77 | 0.87 |
| BasketballScreen | 1.05 | 1.02 | 0.76 |
| MissionControlClip2 | 0.88 | 1.01 | 0.88 |
| MissionControlClip3 | 0.82 | 0.95 | 0.91 |
| SocialNetworkMap | 0.83 | 0.83 | 0.90 |
| Robot | 0.89 | 0.80 | 0.80 |
| YCbCr | Flyinggraphicstext | 0.69 | 0.84 | 0.83 |
| Desktop | 0.65 | 0.74 | 0.75 |
| Console | 0.71 | 0.63 | 0.71 |
| WebBrowsing | 0.65 | 0.75 | 0.83 |
| Map | 0.73 | 0.80 | 0.79 |
| Programming | 0.80 | 0.77 | 0.81 |
| SlideShow | 0.83 | 0.84 | 0.78 |
| BasketballScreen | 1.45 | 1.09 | 1.32 |
| MissionControlClip2 | 1.48 | 1.28 | 1.11 |
| MissionControlClip3 | 1.03 | 1.13 | 0.96 |
| SocialNetworkMap | 0.89 | 1.27 | 0.92 |
| Robot | 0.86 | 0.84 | 0.89 |

## Description of computing platform used to determine encoding and decoding times reported in sections 5.1 and 5.2

The Linux cluster system as specified in Table 8 is used for simulations. The Linux cluster system consists of two categories of CPUs of slightly varying performance (< 10% difference).

**Table 8 Technical specification of the Linux cluster system**

|  |  |
| --- | --- |
| CPU | Intel Xeon E5-2650 / Clock speed 2.00GHz  Intel Xeon X5675 / Clock speed 3.07GHz |
| Memory for each core | 2 GB |
| HDD characteristics | -Hard Drive Capacity: 300GB  -Interface: Serial Attached SCSI  -External Data Transfer Rate: 6.0 GB/sec  -Rotational Speed: 10,000 rpm |
| Compiler | GCC version 4.4.5 |
| Operating System (OS) | 64 bit Linux |

## Expected memory usage of encoder

Table 9 shows the peak memory usage of the proposed encoder for different combinations of coding configurations and video resolutions. The peak memory usage of the proposed encoder ranges from 3.6GB (for 2560x1440 resolution, RA) to 483MB (for 1280x720 resolution, AI). In addition, based on our observations, there is no obvious difference of memory usage between lossy and lossless coding configurations, and/or between RGB and YCbCr color spaces.

**Table 9 The peak memory usage of the proposed encoder**

|  |  |  |
| --- | --- | --- |
| **Coding configuration** | **Resolution** | **Peak encoder memory usage** |
| AI | 1920x1080 | 962MB |
| 1280x720 | 483MB |
| 2560x1440 | 1.6GB |
| RA | 1920x1080 | 3.3GB |
| 1280x720 | 1.6GB |
| 2560x1440 | 3.6GB |
| LD | 1920x1080 | 2.4GB |
| 1280x720 | 1.2GB |
| 2560x1440 | 3.5GB |

## Expected memory usage of decoder

Table 10 shows the peak memory usage of the proposed decoder for different combinations of coding configurations and video resolutions. In specific, the peak memory usage of the proposed decoder ranges from 2GB (for 2560x1440 resolution, RA) to 158MB (for 1280x720 resolution, AI). In addition, based on our observations, there is no obvious difference of memory usage between lossy and lossless coding configurations, and/or between RGB and YCbCr color formats.

**Table 10 Peak memory usage of the proposed decoder**

|  |  |  |
| --- | --- | --- |
| **Coding configuration** | **Resolution** | **Peak encoder memory usage** |
| AI | 1920x1080 | 186MB |
| 1280x720 | 158MB |
| 2560x1440 | 462MB |
| RA | 1920x1080 | 1.2GB |
| 1280x720 | 585MB |
| 2560x1440 | 2GB |
| LD | 1920x1080 | 1GB |
| 1280x720 | 498MB |
| 2560x1440 | 1.5GB |

## Complexity characteristics of encoder motion estimation and partitioning selection

Same as HM-12.1\_RExt-5.1 software.

## Complexity characteristics of decoder motion compensation

Same as HM-12.1\_RExt-5.1 software.

## Complexity characteristics of encoder intra-frame prediction type and partitioning selection

Same as HM-12.1\_RExt-5.1 software.

## Complexity characteristics of decoder intra-frame prediction operation

Same as HM-12.1\_RExt-5.1 software.

## Complexity characteristics of encoder inter-component prediction tools

Same as HM-12.1\_RExt-5.1 software.

## Complexity characteristics of decoder inter-component prediction tools

Same as HM-12.1\_RExt-5.1 software.

## Complexity characteristics of transforms specific in screen content coding

Same as HM-12.1\_RExt-5.1 software.

## Complexity characteristics of encoder related to other tools specific in screen content coding

The complexity characteristics of the improved palette coding in this response are similar to those of the palette coding mode in [3].

The proposed adaptive residue color space conversion can be implemented using only addition and shifting operations. Table 11 summarizes the number of arithmetic operations used by forward and backward color space conversion for both lossy and lossless coding configurations.

**Table 11 Number of arithmetic operations of adaptive color space conversion**

|  |  |  |  |
| --- | --- | --- | --- |
| **Coding configuration** | **color space conversion** | **Addition** | **Shift** |
| Lossy coding | RGB to YCgCo | 4 | 4 |
| YCgCo to RGB | 4 | 0 |
| Lossless coding | RGB to YCgCo | 4 | 2 |
| YCgCo to RGB | 4 | 2 |

In addition, for lossy coding of 4:4:4 RGB and YCbCr video, the 8-tap fractional pixel interpolation filters are applied to all three components. In comparison, HM-12.1\_RExt-5.1 applies 8-tap filters to G (Y) component and 4-tap filters to B and R (Cb and Cr) components.

## Complexity characteristics of decoder related to other tools specific in screen content coding

As discussed as in Section 5.13. For adaptive color space conversion, only YCgCo to RGB conversion is needed at the decoder side.

## Complexity characteristics of encoder entropy coding operation related to tools/parameters specific in screen content coding

The encoder entropy coding complexity is similar to that of HM-12.1\_RExt-5.1 software.

## Complexity characteristics of decoder entropy decoding operation related to tools/parameters specific in screen content coding

The decoder entropy decoding complexity is similar to that of HM-12.1\_RExt-5.1 software.

## Degree of capability for encoder parallel processing

The degree of capability for encoder parallel processing of the proposed solution is expected to be the same as that of HM-12.1\_RExt-5.1 software.

## Degree of capability for decoder parallel processing

The degree of capability for decoder parallel processing of the proposed solution is expected to be the same as that of HM-12.1\_RExt-5.1 software.

## Additional complexity discussion topic #1

## Additional complexity discussion topic #2

## …

# Software implementation description

# Highlighted aspects discussion

# Closing remarks

In this response to the SCC CfP, two main coding tools specifically designed for screen content coding are proposed: the improved palette coding mode, and the adaptive residual color space conversion. All aspects of the proposed coding system, including detailed algorithm descriptions, performance results and complexity analysis, have been provided.

In conclusion, sufficient information about the proposed system has been provided to demonstrate the validity of the proposed coding tools. It is in our opinion that these coding tools provide good coding performance and complexity tradeoff, and should be included in the collaborative phase of the SCC extensions.

# Patent rights declaration(s)

**InterDigital Communications LLC may have IPR relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**

# References

1. D. Flynn, M, Naccari, C. Rosewarne, J. Sole, G. Sullivan, T. Suzuki, High Efficiency Video Coding (HEVC) Range Extensions Text Specification Draft 6. Document no JCTVC-P1005, January 2014.
2. Joint call for proposals for coding of screen content, ISO/IEC JTC-1/SC29/WG11 N14175, January 2014.
3. W. Pu, X. Guo, P. Onno, P.-L. Lai, J. Xu, Suggested Software for the AHG on Investigation of Palette Mode Coding Tools, Document no JCTVC-P0303, January 2014.
4. M. Burrows and D. J. Wheeler, A Block-sorting Lossless Data Compression Algorithm, Technical report SRC 124, May 1994.
5. C. Gisquet, G. Laroche, P. Onno, Non-RCE4: Transition copy mode for Palette mode, , Document no JCTVC-P0115, January 2014.