General picture parameter set RBSP syntax

|  |  |
| --- | --- |
| pic\_parameter\_set\_rbsp( ) { | Descriptor |
| **…** |  |
| **pps\_extension\_present\_flag** | u(1) |
| if( pps\_extension\_present\_flag ) { |  |
| **pps\_range\_extensions\_flag** | u(1) |
| **pps\_scc\_extension\_flag** | u(1) |
| **pps\_extension\_6bits** | u(6) |
| } |  |
| if( pps\_range\_extensions\_flag ) |  |
| pps\_range\_extensions( ) |  |
| if ( pps\_scc\_extension\_flag ) |  |
| **plt\_escape\_color\_pred\_list\_size\_minus1** | u(8) |
| if( pps\_extension\_7bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **pps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Palette mode syntax

|  |  |
| --- | --- |
| palette\_coding( x0, y0, nCbS ) { | Descriptor |
| **palette\_transpose\_flag** | ae(v) |
| **palette\_share\_flag**[ x0 ][ y0 ] | ae(v) |
| if( palette\_share\_flag[ x0 ][ y0 ] ) { |  |
| palette\_size = previousPaletteSize |  |
| for( n = 0; n < palette\_size; n++ ) |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| palette\_entries[ cIdx ][ n ] = previousPaletteEntries[ cIdx ][ n ] |  |
| } else { |  |
| numPredPreviousPalette = 0 |  |
| for( i = 0; i < previousPaletteStuffingSize; i++ ) |  |
| previous\_palette\_entry\_flag[ i ] = 0 |  |
| palette\_last\_group = 0 |  |
| for( i = 0; i < previousPaletteStuffingSize && !palette\_last\_group &&  numPredPreviousPalette < max\_palette\_size; i++ ) { |  |
| lastPossibleGroupFlag = ( i + 4 >= previousPaletteStuffingSize ) |  |
| lastIdx = min( i + 4, previousPaletteStuffingSize ) − 1 |  |
| if( i > 3 && !lastPossibleGroupFlag ) |  |
| **palette**\_**all\_zeros\_in\_group** | ae(v) |
| Else |  |
| palette\_all\_zeros\_in\_group = 0 |  |
| if( palette\_all\_zeros\_in\_group ) |  |
| i += 4 |  |
| else { |  |
| numOnesInGroup = 0 |  |
| for( idx = i; idx <= lastIdx && numPredPreviousPalette < max\_palette\_size;  idx++ ) { |  |
| if ( idx = = lastIdx && numOnesInGroup = = 0 ) |  |
| previous\_palette\_entry\_flag[ idx ] = 1 |  |
| Else |  |
| **previous\_palette\_entry\_flag**[ idx ] | ae(v) |
| if ( previous\_palette\_entry\_flag[ idx ] ) { |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| palette\_entries[ cIdx ][ numPredPreviousPalette ] =   previousPaletteEntries[ cIdx ][ idx ] |  |
| numPredPreviousPalette++ |  |
| numOnesInGroup++ |  |
| } |  |
| } |  |
| if( !palette\_all\_zeros\_in\_group &&   !lastPossibleGroupFlag && numPredPreviousPalette < max\_palette\_size ) |  |
| **palette**\_**last\_group** | ae(v) |
| } |  |
| } |  |
| if( numPredPreviousPalette < max\_palette\_size) |  |
| **num\_signalled\_palette\_entries** | ae(v) |
| for( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| for( i = 0; i < num\_signalled\_palette\_entries; i++ ) |  |
| **palette\_entries**[ cIdx ][ numPredPreviousPalette + i ] | ae(v) |
| palette\_size = numPredPreviousPalette + num\_signalled\_palette\_entries |  |
| } |  |
| **palette\_escape\_val\_present\_flag** | ae(v) |
| if( palette\_escape\_val\_present\_flag ) |  |
| indexMax = palette\_size |  |
| Else |  |
| indexMax = palette\_size – 1 |  |
| escapeClrPredListSize=0 |  |
| for( i = 0; i < previousPaletteStuffingSize && escapeClrPredListSize <= plt\_escape\_color\_pred\_list\_size\_minus1; i++ ) |  |
| if( previous\_palette\_entry\_flag[ i ] = = 0 ) |  |
| escapeClrPredListSize++ |  |
| scanPos = 0 |  |
| while( scanPos < nCbS \* nCbS ) { |  |
| xC = x0 + travScan[ scanPos ][ 0 ] |  |
| yC = y0 + travScan[ scanPos ][ 1 ] |  |
| if( scanPos > 0) { |  |
| xC\_prev = x0 + travScan[ scanPos − 1 ][ 0 ] |  |
| yC\_prev = y0 + travScan[ scanPos − 1 ][ 1 ] |  |
| } |  |
| if( scanPos > = nCbS && palette\_mode[xC\_prev][yC\_prev] ! = COPY\_ABOVE ) |  |
| **palette\_mode**[ xC ][ yC ] | ae(v) |
| if( palette\_mode[ xC ][ yC ] ! = COPY\_ABOVE ) { |  |
| adjustedIndexMax = indexMax |  |
| adjustedRefIndex = indexMax + 1 |  |
| } |  |
| if( scanPos > 0 && palette\_mode[xC\_prev][yC\_prev] ! = ESCAPE ) { |  |
| if( palette\_mode[xC\_prev][yC\_prev] = = INDEX ) { |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex = paletteMap[ xC\_prev ][ yC\_prev ] |  |
| } |  |
| if( scanPos > = nCbS && palette\_mode[ xC\_prev ][ yC\_prev ] = = COPY\_ABOVE   && palette\_mode[ xC ][ yC − 1 ] ! = ESCAPE ) { |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex = paletteMap[ xC ][ yC − 1 ] |  |
| } |  |
| } |  |
| if(palette\_mode[ xC ][ yC ] ! = COPY\_ABOVE ) { |  |
| if( adjustedIndexMax > 0 ) |  |
| **palette\_index** | ae(v) |
| if( palette\_index > = adjustedRefIndex ) |  |
| palette\_index++ |  |
| if( palette\_index = = palette\_size ) { |  |
| if( escapeClrPredListSize || palette\_size) |  |
| **escape\_clr\_pred\_flag**[xC][yC] | ae(v) |
| if( escape\_clr\_pred\_flag[xC][yC] ) { |  |
| if(escapeClrPredListSize && palette\_size) |  |
| **escape\_clr\_pred\_type**[xC][yC] | ae(v) |
| If( (escape\_clr\_pred\_type == 0 && palette\_size >1) || (escape\_clr\_pred\_type == 1  && escapeClrPredListSize >1 ) ) |  |
| **escape\_clr\_pred\_idx\_coded**[xC][yC] | ae(v) |
| **escape\_clr\_residual\_flag**[xC][yC] | ae(v) |
| if( escape\_clr\_residual\_flag[xC][yC] ) { |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| **escape\_clr\_residual\_abs\_val**[cIdx][xC][yC] | ae(v) |
| if(escape\_clr\_residual\_absolute\_val[cIdx][xC][yC] ) |  |
| **escape\_clr\_residual\_sign**[cIdx][xC][yC] | ae(v) |
| } |  |
| } |  |
| } else { |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| **palette\_escape\_val** | ae(v) |
| paletteEscapeVal[ cIdx ][ xC ][ yC ] = palette\_escape\_val |  |
| } |  |
| } |  |
| if( !escape\_clr\_pred\_flag[xC][yC] || escape\_clr\_residual\_flag[xC][yC]) |  |
| escapeClrPredListSize = min (escapeClrPredListSize,  plt\_escape\_color\_pred\_list\_size\_minus1) +1 |  |
| palette\_mode[ xC ][ yC ] = ESCAPE |  |
| scanPos++ |  |
| } |  |
| } |  |
| if( palette\_mode[xC][yC] ! = ESCAPE ) { |  |
| **palette\_run** | ae(v) |
| runPos = 0 |  |
| runMode = palette\_mode[ xC ][ yC ] |  |
| while ( runPos < = palette\_run ) { |  |
| xC = x0 + travScan[ scanPos ][ 0 ] |  |
| yC = y0 + travScan[ scanPos ][ 1 ] |  |
| if( palette\_mode[ xC ][ yC ] = = INDEX ) { |  |
| palette\_mode[ xC ][ yC ] = INDEX |  |
| paletteMap[ xC ][ yC ] = palette\_index |  |
| } else { |  |
| palette\_mode[ xC ][ yC ] = COPY\_ABOVE |  |
| paletteMap[ xC ][ yC ] = paletteMap[ xC ][ y − 1 ] |  |
| } |  |
| runPos++ |  |
| scanPos++ |  |
| } |  |
| } |  |
| } |  |
| previousPaletteSize = palette\_size |  |
| current\_size = palette\_size |  |
| for( i = 0; i < palette\_size; i++ ) |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| tempPaletteEntries[ cIdx ][ i ] = palette\_entries[ cIdx ][ i ] |  |
| for( i = 0; i < previousPaletteStuffingSize && current\_size < max\_palette\_predictor\_size;   i++ ) |  |
| if( previous\_palette\_entry\_flag[ i ] = = 0 ) { |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| tempPaletteEntries[ cIdx ][ current\_size ] = previousPaletteEntries[ cIdx ][ i ] |  |
| current\_size++ |  |
| } |  |
| previousPaletteStuffingSize = current\_size |  |
| previousPaletteEntries = tempPaletteEntries |  |
| } |  |

##### General picture parameter set RBSP semantics

**pps\_scc\_extension\_flag** equal to 1 specifies that the screen content coding extensions related syntax elements are present in the PPS RBSP syntax structure. pps\_scc\_extensions\_flag equal to 0 specifies that those syntax elements are not present. When not present, the value of pps\_scc\_extensions\_flag is inferred to be equal to 0.

**plt\_escape\_color\_pred\_list\_size\_minus1** plus 1 specifies the maximum size of escape color list for escape color prediction in the coding unit coded in palette mode.

#### Palette mode semantics

**escape\_clr\_pred\_flag**[xC][yC]equal to 1 specifies that the escape color is coded in predictive mode. escape\_clr\_pred\_flagequal to 0 specifies that the escape color is coded without using prediction. When not present, the value of escape\_clr\_pred\_flag[xC][yC] is inferred to be equal to 0.

**escape\_clr\_pred\_type**[xC][yC] equal to 1 specifies that the predictor of escape color is from escape color list. escape\_clr\_pred\_typeequal to 0 specifies that the predictor of escape color is from palette table. When not present, the value of escape\_clr\_pred\_type[xC][yC] is derived as follows :

escape\_clr\_pred\_type[xC][yC] = (palette\_size >0) ? 0 : 1

**escape\_clr\_pred\_idx\_coded**[xC][yC] specifies how to derive the index of escape color predictor depending on the value of escape\_clr\_pred\_type[xC][yC]. If escape\_clr\_pred\_type[xC][yC] is 0, escape\_clr\_pred\_idx\_coded[xC][yC] is used to derive the index in the palette table. If escape\_clr\_pred\_type[xC][yC] is 1, escape\_clr\_pred\_idx\_coded[xC][yC] is used to derive the index in the escape color predictor list. When not present, the value of escape\_clr\_pred\_idx\_coded[xC][yC] is inferred to be equal to 0. The escapeClrPredIdx[xC][yC] is derived as follows,

if(escape\_clr\_pred\_type[xC][yC] ==0 )

escapeClrPredIdx[xC][yC] = palette\_size-1 - escape\_clr\_pred\_idx\_coded[xC][yC]

else

escapeClrPredIdx[xC][yC] = escape\_clr\_pred\_idx\_coded[xC][yC]

The predictor for escape color reconstruction of component cIdx is derived as follows,

The list escapeClrList is intialized as follows,

escapeClrPredListSize = 0;

for( i=0; i < previousPaletteStuffingSize && escapeClrPredListSize <= plt\_escape\_color\_pred\_list\_size\_minus1; i++) {

if(previous\_palette\_entry\_flag[ i ]==0) {

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

escapeClrList[j][ escapeClrPredListSize ] = previousPaletteEntries[j][i];

escapeClrPredListSize ++;

}

}

**escape\_clr\_residual\_flag**[xC][yC]equal to 1 specifies that there are non-zero residuals for escape color reconstruction of three components when escape color is coded with prediction. escape\_clr\_residual\_flag[xC][yC]equal to 0 specifies that residuals are zero for three components. When not present, the value of escape\_clr\_residual\_flag[xC][yC] is inferred to be equal to 0.

**escape\_clr\_residual\_abs\_val**[cIdx][xC][yC] specifies the absolute value of residual of component cIdx. When not present, the value of escape\_clr\_residual\_absolute\_val[cIdx][xC][yC] is inferred to be equal to 0.

**escape\_clr\_residual\_sign**[cIdx][xC][yC] equal to 1 specifies the residual value of component cIdx is negative. escape\_clr\_residual\_sign[cIdx][xC][yC] equal to 0 specifies the residual value of component cIdx is positive. When not present, the value of escape\_clr\_residual\_sign[cIdx][xC][yC] is inferred to be equal to 0.

##### 8.4.5.2.8 Decoding process for palette mode

Inputs to this process are:

– a location ( xCb, yCb ) specifying the top-left sample of the current block relative to the top-left sample of the current picture,

– a variable cIdx specifying the colour component of the current block,

– a variable nCbS specifying the size of the current block,

– An array palette\_mode[ xCb + x ][ yCb + y ], with x = 0..nCbS − 1 and  y = 0..nCbS − 1, specifying the palette mode for each sample in the current block

– The array paletteMap[ xCb + x ][ yCb + y ], with x = 0..nCbS − 1,  y = 0..nCbS − 1, specifying the palette index for each sample in the current block for which the palette mode is not equal to ESCAPE, and

– The array palette\_escape\_val[ xCb + x ][ yCb + y ], with x = 0..nCbS − 1 and  y = 0..nCbS − 1, specifying the quantized escape value for each sample in the current block for which the palette mode is equal to ESCAPE.

Output of this process array recSamples[ x ][ y ], with x = 0..nCbS − 1,  y = 0..nCbS − 1, specifying a reconstructed sample values for the palette block.

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

– If cIdx is equal to 0, recSamples corresponds to the reconstructed picture sample array SL and the function clipCidx1 corresponds to Clip1Y.

– Otherwise, if cIdx is equal to 1, recSamples corresponds to the reconstructed chroma sample array SCb and the function clipCidx1 corresponds to Clip1C.

– Otherwise (cIdx is equal to 2), recSamples corresponds to the reconstructed chroma sample array SCr and the function clipCidx1 corresponds to Clip1C.

The (nCbS x nCbS) block of the reconstructed sample array recSamples at location ( xCb, yCb ) is derived as follows:

For x = 0..nCbS − 1,  y = 0..nCbS − 1, recSample[ cIdx ][ yCb + y ][ xCb + x ] is set as follows:

– If palette\_mode[ xCb + x ][ yCb + y ] is not equal to ESCAPE, the following applies:

If palette\_transpose\_flag is true, the following applies:

recSample[ cIdx ][ yCb + y ][ xCb + x ] = palette\_entries[ cIdx ][ paletteMap[ xCb + x ][ yCb + y ] ],

– Otherwise (palette\_transpose\_flag is false), the following applies:

recSample[ cIdx ][ xCb + x ][ yCb + y ] = palette\_entries[ cIdx ][ paletteMap[ xCb + x ][ yCb + y ] ],

– Otherwise (ESCAPE), if escape\_clr\_pred\_flag[ xCb + x ][ yCb + y ] is 0,

If cu\_transquant\_bypass\_flag is true, the following applies:

If palette\_transpose\_flag is true, the following applies:

recSample[ cIdx ][ yCb + y ][ xCb + x ] = palette\_escape\_val[ cIdx ][ xCb + x ][ yCb + y ],

– Otherwise (palette\_transpose\_flag is false), the following applies:

recSample[ cIdx ][ xCb + x ][ yCb + y ] = palette\_escape\_val[ cIdx ][ xCb + x ][ yCb + y ],

Otherwise if palette\_mode[ xCb + x ][ yCb + y ] is equal to ESCAPE and cu\_transquant\_bypass\_flag is false , the following ordered steps apply:

1. The derivation process for quantization parameters as specified in subclause 8.6.1 is invoked assuming that the current block is the first block in the slice, availableA is equal to FALSE, availableB is equal to FALSE, and CuQpDeltaVal is equal to 0.
2. The quantization parameter qP is derived as follows:

– If cIdx is equal to 0, qP is set to Qp′Y

– Otherwise, if cIdx is equal to 1, qP is set to Qp′Cb

– Otherwise (cIdx is equal to 2), qP is set to Qp′Cr.

1. The variable bdShift is derived as follows:

bitShift = ( ( cIdx = = 0 ) ? BitDepthY : BitDepthC)  + Log2( nTbS ) − 5 (8‑XX)

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

If palette\_transpose\_flag is true, the following applies

recSamples[cIdx][ yCb + y ][ xCb + x ] = Clip3( 0, 1 << bdShift − 1, ( ( palette\_escape\_val[ cIdx ][ xCb + x ][ yCb + y ]\* 16 \* levelScale[ qP%6 ]  <<  (qP / 6 ) ) + ( 1  <<  ( bdShift − 1 ) ) )  >>  bdShift ),

– Otherwise (palette\_transpose\_flag is false), the following applies

recSamples[cIdx][ xCb + x ][ yCb + y ] = Clip3( 0, 1 << bdShift − 1, ( ( palette\_escape\_val[ cIdx ][ xCb + x ][ yCb + y ]\* 16 \* levelScale[ qP%6 ]  <<  (qP / 6 ) ) + ( 1  <<  ( bdShift − 1 ) ) )  >>  bdShift )

– Otherwise ( escape\_clr\_pred\_flag[ xCb + x ][ yCb + y ] is 1 ) , the following applies:

The variables escapeClrPredictor[cIdx] and escapeClrResidual[cIdx] for the cIdx-th component are derived as follows,

escapeClrPredictor[cIdx] = (escape\_clr\_pred\_type[xCb+x][yCb+y]==0) ?

palette\_entries[ cIdx ][ escapeClrPredIdx[xCb+x][yCb+y] ] :

escapeClrList[cIdx][escapeClrPredIdx[xCb+x][yCb+y]]

escapeClrResidual[cIdx] = escape\_clr\_residual\_sign[cIdx][xCb+x][yCb+y] ?

-escape\_clr\_residual\_abs\_val[cIdx][xCb+x][yCb+y]  :

escape\_clr\_residual\_abs\_val[cIdx][xCb+x][yCb+y]

If cu\_transquant\_bypass\_flag is true, the following applies:

If palette\_transpose\_flag is true, the following applies:

recSample[ cIdx ][ yCb + y ][ xCb + x ] = ( escapeClrPredictor[cIdx] +

escapeClrResidual[cIdx] ),

– Otherwise (palette\_transpose\_flag is false), the following applies:

recSample[ cIdx ][ xCb + x ][ yCb + y ] = (escapeClrPredictor[cIdx] +

escapeClrResidual[cIdx] ),

Otherwise (cu\_transquant\_bypass\_flag is false), the following ordered steps apply:

1. The derivation process for quantization parameters as specified in subclause 8.6.1 is invoked assuming that the current block is the first block in the slice, availableA is equal to FALSE, availableB is equal to FALSE, and CuQpDeltaVal is equal to 0.
2. The quantization parameter qP is derived as follows:

– If cIdx is equal to 0, qP is set to Qp′Y

– Otherwise, if cIdx is equal to 1, qP is set to Qp′Cb

– Otherwise (cIdx is equal to 2), qP is set to Qp′Cr.

1. The variable bdShift is derived as follows:

bitShift = ( ( cIdx = = 0 ) ? BitDepthY : BitDepthC)  + Log2( nTbS ) − 5 (8‑XX)

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

If palette\_transpose\_flag is true, the following applies

recSamples[cIdx][ yCb + y ][ xCb + x ] = Clip3( 0, 1 << bdShift − 1, (( ( escapeClrResidual[ cIdx ]\* 16 \* levelScale[ qP%6 ]  <<  (qP / 6 ) ) + ( 1  <<  ( bdShift − 1 ) ) )  >>  bdShift ) + escapeClrPredictor[cIdx]),

Otherwise (palette\_transpose\_flag is false), the following applies

recSamples[cIdx][ xCb + x ][ yCb + y ] = Clip3( 0, 1 << bdShift − 1, (( ( escapeClrResidual[ cIdx ]\* 16 \* levelScale[ qP%6 ]  <<  (qP / 6 ) ) + ( 1  <<  ( bdShift − 1 ) ) )  >>  bdShift ) + escapeClrPredictor[cIdx])

– If palette\_mode[ xCb + x ][ yCb + y ] is equal to ESCAPE, the escape color predictors list (escapeClrList) is updated as follows,

if (escape\_clr\_pred\_flag[ xCb + x ][ yCb + y ]==0 || escape\_clr\_residual\_flag[ xCb + x ][ yCb + y ] ==1) {

for(i=min(escapeClrPredListSize, plt\_escape\_color\_pred\_list\_size\_minus1) ; i>0; i--)

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

escapeClrList[j][i] = escapeClrList[j][i-1] ;

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

escapeClrList[j][0] = palette\_transpose\_flag? recSamples[j][ yCb + y ][ xCb + x ] : recSamples[j][ xCb + x ][ yCb + y ];

escapeClrPredListSize = min(escapeClrPredListSize, plt\_escape\_color\_pred\_list\_size\_minus1)+1;

}

#### Initialization process for context variables

1. Table 9‑4 – Association of ctxIdx and syntax elements for each initializationType in the initialization process

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Syntax structure** | **Syntax element** | **ctxTable** | **initType** | | |
| **0** | **1** | **2** |
| palette\_coding( ) | escape\_clr\_pred\_flag | Table 9‑41 | 0 | 0 | 0 |
| escape\_clr\_pred\_type | Table 9‑42 | 0 | 0 | 0 |
| escape\_clr\_pred\_idx\_coded | Table 9‑43 | 0 | 0 | 0 |
| escape\_clr\_residual\_flag | Table 9‑44 | 0 | 0 | 0 |
| escape\_clr\_residual\_abs\_val | Table 9‑45 | 0..2 | 0..2 | 0..2 |

Table 9‑41 – Values of initValue for ctxIdx of escape\_clr\_pred\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of** escape\_clr\_pred\_flag | |
| **0** |  |
| **initValue** | 154 |  |

Table 9‑42 – Values of initValue for ctxIdx of escape\_clr\_pred\_type

|  |  |
| --- | --- |
| **Initialization variable** | **ctxIdx of** escape\_clr\_pred\_type |
| **0** |
| **initValue** | 154 |

Table 9‑43 – Values of initValue for ctxIdx of escape\_clr\_pred\_idx\_coded

|  |  |
| --- | --- |
| **Initialization variable** | **ctxIdx of** escape\_clr\_pred\_idx\_coded |
| **0** |
| **initValue** | 154 |

Table 9‑44 – Values of initValue for ctxIdx of escape\_clr\_residual\_flag

|  |  |  |
| --- | --- | --- |
| **Initialization variable** | **ctxIdx of** escape\_clr\_residual\_flag | |
| **0** |  |
| **initValue** | 154 |  |

Table 9‑45 – Values of initValue for escape\_clr\_residual\_abs\_val

|  |  |  |  |
| --- | --- | --- | --- |
| **Initialization variable** | **ctxIdx of escape\_clr\_residual\_abs\_val** | | |
| **0** | **1** | **2** |
| **initValue** | **154** | **154** | **154** |

### Binarization process

Table 9‑38 – Syntax elements and associated binarizations

|  |  |  |  |
| --- | --- | --- | --- |
| palette\_coding( ) | previous\_palette\_entry\_flag[] | FL | cMax = 1 |
| palette\_share\_flag | FL | cMax = 1 |
| palette\_num\_signalled\_entries | TR | cMax = 31, cRiceParam = 0 |
| palette\_entries | FL | cMax = cIdx = = 0 ? ( (1<<BitDepthY) − 1 ) : ( (1<<BitDepthC) − 1 ) |
| palette\_transpose\_flag | FL | cMax = 1 |
| palette\_escape\_val | 9.3.3.12 | cIdx, qP |
| escape\_val\_present\_flag | FL | cMax = 1 |
| palette\_run\_type\_flag | FL | cMax = 1 |
| palette\_index | TB | cMax = adjustedIndexMax |
| palette\_run | 9.3.3.13 | - |
| palette\_all\_zeros\_in\_group | FL | cMax = 1 |
| palette\_last\_group | FL | cMax = 1 |
| escape\_clr\_pred\_flag | FL | cMax = 1 |
| escape\_clr\_pred\_type | FL | cMax = 1 |
| escape\_clr\_pred\_idx\_coded | 9.3.3.14 |  |
| escape\_clr\_residual\_flag | FL | cMax = 1 |
| escape\_clr\_residual\_abs\_val | 9.3.3.15 |  |
| escape\_clr\_residual\_sign | FL | cMax = 1 |

#### Binarization process for escape\_clr\_pred\_idx\_coded

Input to this process is a request for a binarization for the syntax element escape\_clr\_pred\_idx\_coded.

Output of this process is the binarization of escape\_clr\_pred\_idx\_coded as specified in Table 9‑47.

Table 9‑47 – Binarization for escape\_clr\_pred\_idx\_coded

|  |  |
| --- | --- |
| **escape\_clr\_pred\_idx\_coded** | Codeword |
| 0 | 0 |
| >0 | TB |

The maximum parameter cMax for TB binarization is derived as follows

if ( escape\_clr\_pred\_type ==0 )

cMax = palette\_size – 1

else

cMax = escapeClrPredListSize -1

#### 9.3.3.15 Binarization process for escape\_clr\_residual\_abs\_val

Input to this process is a request for a binarization for the syntax element escape\_clr\_residual\_abs\_val.

Output of this process is the binarization of escape\_clr\_residual\_abs\_val as specified in Table 9‑48.

Table 9‑48(a) – Binarization for escape\_clr\_residual\_abs\_val if cu\_transquant\_bypass\_flag is 0

|  |  |
| --- | --- |
| **escape\_clr\_residual\_abs\_val** | Codeword |
| 0 | 0 |
| 1 | 10 |
| 2 | 110 |
| 3 | 111 |

Table 9‑48(b) – Binarization for escape\_clr\_residual\_abs\_val if cu\_transquant\_bypass\_flag is 1

|  |  |
| --- | --- |
| **escape\_clr\_residual\_abs\_val** | Codeword |
| 0 | 0 |
| 1 | 10 |
| 2 | 110 |
| >2 | Prefix = '111', Suffix = TR code , cRiceParam = 2 |

| Table 9‑43 – Assignment of ctxInc to syntax elements with context coded bins | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Syntax element** | **binIdx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| … |  |  |  |  |  |  |
| palette\_mode\_flag | 0 | na | na | na | Na | Na |
| palette\_share\_flag | 0 | na | na | na | Na | na |
| previous\_palette\_entry\_flag | bypass | na | na | na | na | Na |
| palette\_num\_signalled\_entries | bypass | bypass | bypass | bypass | bypass | bypass |
| palette\_entries | bypass | bypass | bypass | bypass | bypass | bypass |
| palette\_transpose\_flag | 0, 1 (subclause 9.3.4.2.2) | na | na | na | na | Na |
| palette\_escape\_val | bypass | bypass | bypass | bypass | bypass | bypass |
| escape\_val\_present\_flag | bypass | na | na | na | na | na |
| palette\_run\_type\_flag | 0, 1 (subclause 9.3.4.2.2) | na | Na | na | Na | Na |
| palette\_index | bypass | bypass | bypass | bypass | bypass | bypass |
| palette\_run | 0 | 1 | 2 | bypass | bypass | bypass |
| palette\_all\_zeros\_in\_group | bypass | na | na | na | na | na |
| palette\_last\_group | bypass | na | na | na | na | na |
| escape\_clr\_pred\_flag | 0 | na | na | na | na | na |
| escape\_clr\_pred\_type | 0 | na | na | na | na | na |
| escape\_clr\_pred\_idx\_coded | 0 | bypass | bypass | bypass | bypass | bypass |
| escape\_clr\_residual\_flag | 0 | na | na | na | na | na |
| escape\_clr\_residual\_abs\_val | 0 | 1 | 2 | bypass | bypass | bypass |
| escape\_clr\_residual\_sign | bypass | na | na | na | na | na |
| … |  |  |  |  |  |  |