**JCTVC-S0178 Solution 1**

**7.3.8.8 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++ ) { [Ed. YY: max\_palette\_size probably needs to be signalled] |  |
| 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 |  |
| if ( palette\_size > 2){ |  |
| **use\_transition\_flag** | ae(v) |
| } |  |
| 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[0] = indexMax + 1 |  |
| adjustedRefIndex[1] = indexMax + 1 |  |
| } |  |
| if( scanPos > 0 && palette\_mode[xC\_prev][yC\_prev] ! = ESCAPE ) { |  |
| if( palette\_mode[xC\_prev][yC\_prev] = = INDEX ) { |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex[0] = 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[0]= paletteMap[ xC ][ yC − 1 ] |  |
| } |  |
| } |  |
| if(indexMax > 2){ |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex[1] = tran\_idx |  |
| } |  |
| if(palette\_mode[ xC ][ yC ] ! = COPY\_ABOVE { |  |
| if( adjustedIndexMax > 1 ) { |  |
| if (use\_transition\_flag){ |  |
| **equal\_to\_transition\_flag** | ae(v) |
| if ( equal\_to\_transition\_flag == 1){ |  |
| palette\_index **=**  tran\_idx |  |
| } |  |
| else{ |  |
| **palette\_index** | ae(v) |
| if (palette\_index < tran\_idx || ( (tran\_idx <= adjustedRefIndex[0]) && palette\_index == tran\_idx))) |  |
| palette\_index ­­­­­− − |  |
| if( palette\_index > = adjustedRefIndex ) |  |
| palette\_index++ |  |
| else{ |  |
| **palette\_index** | ae(v) |
| if( palette\_index > = adjustedRefIndex ) |  |
| palette\_index++ |  |
| } |  |
| } |  |
| ~~if( palette\_index > = adjustedRefIndex )~~ |  |
| ~~palette\_index++~~ |  |
| if( palette\_index = = palette\_size ) { |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| **palette\_escape\_val** | ae(v) |
| paletteEscapeVal[ cIdx ][ xC ][ yC ] = palette\_escape\_val |  |
| } |  |
| 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 |  |
| } |  |

##### 7.4.9.6 Palette semantics

**use\_transition\_flag**[ x0 ][ y0 ] equal to 1 specifies that the transition copy index prediction is used in the current block. use\_transition\_flag[ x0 ][ y0 ] equal to 0 specifies that the transition copy index prediction is not used in the curent block.

##### equal\_to\_transition\_flag equal to 1 specifies that the palette index of current sample is equal to the transition copy index prediction value. equal\_to\_transition\_flag equal to 0 specifies that the palette index of the current sample is not equal to the transition copy index prediction value.

##### 8.4.5.2.X Decoding process for transition index table propagation

Inputs to this process are:

– The array prev\_TC\_index[pilot\_index][candidate\_index], with pilot\_index = 0..31,  candidate\_index = 0 and 1, specifying transition index table of previous CU.

Output of this process

– The array TC\_table[pilot\_index][candidate\_index], with pilot\_index = 0..31,  candidate\_index = 0 and 1, specifying transition index table of current CU.

For the derivation of the transition index table TC\_index[ pilot ][ cand ] with pilot = 0..31,  cand = 0 and 1, the following applies:

– If the current CU is the first CU of the slice, TC\_index[pilot][cand] = 0, with pilot = 0..31,  cand = 0 and 1.

– Otherwise, the following applies, with pilot = 0..31,  cand = 0 and 1:

The color map table color\_map[ c ] with c = 0…31 is derived as follows:

for( idx = i; idx <= 32; idx++ )

{

color\_map[idx] = -1;

if ( previous\_palette\_entry\_flag[ idx ] )

{color\_map[idx] = numPredPreviousPalette}

numPredPreviousPalette++

}

For the derivation of the transition index table TC\_index[ pilot ][ cand ] , the following applies with pilot = 0..31,  cand = 0 and 1:

TC\_index[color\_map[pilot]][cand] = 0.

And, the following applies, with pilot = 0..31:

– If color\_map[pilot] is not equal to -1, TC\_index[color\_map[pilot]][cand] = prev\_TC\_index[pilot][cand],  
 , cand = 0 and 1.

And, the following applies, with pilot = 0..31,  cand = 0 and 1:

– If color\_map[TC\_index[pilot][cand]] is equal to -1, TC\_index[pilot][cand] = 0.

– Otherwise, TC\_index[pilot][cand] = color\_map[TC\_index[pilot][cand]].

##### 8.4.5.2.X Decoding process for transition index derivation

Inputs to this process are:

– a location ( x, y ) specifying the current sample relative to the top-left sample of the current block,

– a location ( x\_prev , y\_prev  ) specifying the previous scaned sample relative to the top-left sample of the current block,

– a variable indexMax specifying the maximal index value of the current block,

– a variable adjustedRefIndex[0] specifying the redundant index value of the location ( x, y ),

– a variable palette\_index specifying the decoded index value of the location ( x, y ),

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

– The array paletteMap[ x ][ 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 [Ed. (GJS): Check "is not" versus "is not equal to" (generally).]

– The array palette\_mode[ x ][ y ], with x = 0..nCbS − 1,  y = 0..nCbS − 1, specifying the palette run mode.

– The array TC\_table[pilot\_index][candidate\_index], with pilot\_index = 0..31,  candidate\_index = 0 and 1, specifying transition index table of current CU.

Output of this process tran\_idx, specifying a prediction index value of transition copy mode.

Depending on the palette\_mode[ xCb][ yCb], the following assignments are made:

– If palette\_mode[ x ][ y ] is equal to INDEX or INDEX\_TC,

tran\_idx\_tmp = palette\_mode[paletteMap[ x ][ y ]][ y%2 ].

Depending on the adjustedRefIndex[0], the following assignments are made:

– If adjustedRefIndex[0] is equal to 0, tran\_idx\_refine = 1

– Otherwise, tran\_idx\_refine = 0

Depending on the value of tran\_idx\_tmp, indexMax, adjustedRefIndex[0], palette\_mode[ xC\_prev ][ yC\_prev ], and paletteMap[xCb][ yCb- 1], the following assignments are made:

– If tran\_idx\_tmp is larger than indexMax,

tran\_idx = tran\_idx\_refine.

– Otherwise, if tran\_idx\_tmp is equal to adjustedRefIndex[0],

tran\_idx = tran\_idx\_refine.

– Otherwise, if palette\_mode[ xC\_prev ][ yC\_prev ] is equal to COPY\_ABOVE, paletteMap[xCb][ yCb- 1] is not equal to paletteMap[ xC\_prev][ yC\_prev], and tran\_idx\_tmp is equal to paletteMap[xCb][ yCb- 1],

tran\_idx = paletteMap[ xC\_prev][ yC\_prev].

– Otherwise,

tran\_idx = tran\_idx\_tmp.

##### 8.4.5.2.X Decoding process for transition index table updating

Inputs to this process are:

– a location ( x, y ) specifying the current sample relative to the top-left sample of the current block,

– a location ( x\_prev , y\_prev  ) specifying the previous scaned sample relative to the top-left sample of the current block,

– a variable palette\_index specifying the decoded index value of the location ( x, y ),

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

– The array paletteMap[ x ][ 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 [Ed. (GJS): Check "is not" versus "is not equal to" (generally).]

– The array palette\_mode[ x ][ y ], with x = 0..nCbS − 1,  y = 0..nCbS − 1, specifying the palette run mode.

– The array TC\_table[pilot\_index][candidate\_index], with pilot\_index = 0..31,  candidate\_index = 0 and 1, specifying transition index table of current CU.

Depending on the palette\_mode[  x ][ y ] and x, the following assignments are made:

– If x % nCbS is not equal to 0,

palette\_mode[paletteMap[ x\_prev][ y\_prev]][ y%2 ] = palette\_index, and

palette\_mode[palette\_index][ (y+1)%2 ] = paletteMap[ x\_prev][ y\_prev].

### Binarization process

#### 9.3.3.1 General

Input to this process is a request for a syntax element.

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

Table 9‑38 specifies the type of binarization process associated with each syntax element and corresponding inputs.

The specification of the truncated Rice (TR) binarization process, the k-th order Exp-Golomb (EGk) binarization process, limited k-th order Exp-Golomb (EGk) binarization process the fixed-length (FL) binarization process, and truncated binarization process are given in subclauses 9.3.3.2 through 9.3.3.6 respectively. Other binarizations are specified in subclauses 9.3.3.7 through 9.3.3.11. [Ed. (GJS): Should that last cross-reference be to subclause 9.3.3.13?]

Table 9‑38 – Syntax elements and associated binarizations

|  |  |  |  |
| --- | --- | --- | --- |
| palette\_coding( ) | previous\_palette\_entry\_flag[] | FL | cMax = 1 |
| palette\_share\_flag | FL | cMax = 1 |
| use\_transition\_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\_mode\_flag | FL | cMax = 1 |
| equal\_to\_transition\_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 |