|  |  |
| --- | --- |
| 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 |  |
| 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\_prev][yC\_prev] = = COPY\_PREV\_ROW && palette\_mode[ xC ][ yC − paletteRowOffset[ xC\_prev][ yC\_prev ] ] ! = ESCAPE){ |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex = paletteMap[ xC ][ yC − paletteRowOffset[ xC\_prev][ yC\_prev ]  ] |  |
| } |  |
| } |  |
| 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 ) { |  |
| 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 ] = = COPY\_PREV\_ROW ) { |  |
| adjustedRowMax = yC − 1 |  |
| adjustedRefRow = yC |  |
| if(palette\_mode[xC\_prev][yC\_prev] = = COPY\_PREV\_ROW){ |  |
| adjustedRowMax − − |  |
| adjustedRefRow = paletteRowOffset[ xC\_prev][ yC\_prev ] − 2 |  |
| } |  |
| **palette\_row\_offset** | ae(v) |
| palette\_row\_offset = palette\_row\_offset + 2 |  |
| } |  |
| 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 |  |
| paletteRowOffset[ xC ][ yC ] = 0 |  |
| } |  |
| else if (palette\_mode[ xC ][ yC ] == COPY\_ABOVE){ |  |
| palette\_mode[ xC ][ yC ] = COPY\_ABOVE |  |
| paletteMap[ xC ][ yC ] = paletteMap[ xC ][ y − 1 ] |  |
| paletteRowOffset[ xC ][ yC ] = 1 |  |
| }else{ |  |
| palette\_mode[ xC ][ yC ] = COPY\_PREV\_ROW |  |
| paletteMap[ xC ][ yC ] = paletteMap[ xC ][ y –palette\_row\_offset] |  |
| paletteRowOffset[ xC ][ yC ] = palette\_row\_offset |  |
| } |  |
| 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 mode semantics

**palette\_mode[** xC **][** yC **]** specifies the palette mode for the samples. The array indices xC, yC specify the location of the sample relative to the top-left luma sample of the picture. palette\_modeequal to 1 specifies thatthe sample is coded in COPY\_ABOVE mode, where the decoded sample value is equal to the sample value at the same location in the row above when the palette\_transpose flag is equal to 1 and is equal to the sample value at the same location in the column to the left when the palette\_transpose flag is equal to 0. palette\_mode equal to 0 specifies that the sample is coded in INDEX mode, where the sample's palette index (after necessary adjustments) is coded in the bitstream. palette\_mode equal to 2 specifies that the sample is coded COPY\_PREV\_ROW mode, where the decoded samples values is equal to the sample value in the previous row with the same horizontal position when the palette\_transpose flag is equal to 1 and is equla to the sample values in the previous column with the same vertical position when the palette\_transpose flag is equal to 0. Palette mode is defined only for blocks [Ed. (GJS): Something missing here. Perhaps it is only a full stop that is missing.]

**palette\_row\_offset[** xC **][** yC **]** specifies the reference row offset relative to the current sample when palette\_transpose flag is equal to 1, and the reference column offset relative to the current sample when palette\_transpose flag is equal to 0

9.3.3.1 General

Table 9‑38 – Syntax elements and associated binarizations

| **Syntax structure** | **Syntax element** | **Binarization** | |
| --- | --- | --- | --- |
| **Process** | **Input parameters** |
| 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\_mode | 9.3.3.14 | cMax = 2 |
| palette\_row\_offset | TB | cMax = adjustedRowMax |
| 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 |

9.3.3.14 Binarization process for palette\_mode

Input to this process is a request for a binarization for the syntax element palette\_mode, indexMax, scanPos and palette\_mode[xC\_prev][yC\_prev] specifying the palette mode for the previously coded sample in the current block.

Output of this process is the binarization of palette\_mode.

The binarization process for palette\_mode is derived as follows:

If indexMax is larger than 2, the following applies:

* indexMode flag is derived by invoking the FL binarization process spefified in subclause 9.3.3.5 with the input parameter set to 1, and the following applies:
  + If indexMode flag is equal to 0, palette\_mode is set equal to 0
  + Otherwise (indexMode flag is not equal to 0), the following applies:
    - If scanPos is equal to 0 or palette\_mode[xC\_prev][yC\_prev] is not equal to COPY\_ABOVE mode, aboveMode flag is derived by invoking the FL binarization process spefified in subclause 9.3.3.5 with the input parameter set to 1
      * If aboveMode flag is equal to 0, palette\_mode is set equal to 1
      * Othersie (aboveMode flag is equal to 1), palette\_mode is set equal to 2
    - Otherwise, palette\_mode is set equal to 0

Otherwise (indexMax is equal to or smaller than 2), the following applies:

* If scanPos is equal to 0 or palette\_mode[xC\_prev][yC\_prev] is not equal to COPY\_ABOVE mode, indexMode flag is derived by invoking the FL binarization process spefified in subclause 9.3.3.5 with the input parameter set to 1, and the following applies:
  + If indexMode is equal to 0, palette\_mode is set equal to 0
  + Otherwise (indexMode is not equal to 0), palette\_mode is set equal to 1
* Otherwise, palette\_mode is set equal to 0