#### Palette syntax

|  |  |
| --- | --- |
| palette\_coding( x0, y0, nCbS ) { | Descriptor |
| **palette\_share\_flag**[ x0 ][ y0 ] | ae(v) |
| if( !palette\_share\_flag[ x0 ][ y0 ] ) { |  |
| palettePredictionFinished = 0 |  |
| paletteNumPredictedEntries = 0 |  |
| for( i = 0; i < PredictorPaletteSize  && !palettePredictionFinished &&  paletteNumPredictedEntries < palette\_max\_size; i++ ) { |  |
| **palette\_predictor\_run** | ae(v) |
| if( palette\_predictor\_run != 1 ) |  |
| if( palette\_predictor\_run > 1 ) |  |
| i += palette\_predictor\_run − 1 |  |
| PalettePredictorEntryReuseFlag[ i ] = 1 |  |
| paletteNumPredictedEntries ++ |  |
| } else |  |
| palettePredictionFinished = 1 |  |
| } |  |
| if( paletteNumPredictedEntries < palette\_max\_size ) |  |
| **palette\_num\_signalled\_entries** | ae(v) |
| for( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| for( i = 0; i < palette\_num\_signalled\_ entries; i++ ) |  |
| **palette\_entry** | ae(v) |
| } |  |
| if( currentPaletteSize != 0 ) |  |
| **palette\_escape\_val\_present\_flag** | ae(v) |
| if( palette\_escape\_val\_present\_flag ) { |  |
| if( cu\_qp\_delta\_enabled\_flag && !IsCuQpDeltaCoded ) { |  |
| **cu\_qp\_delta\_palette\_abs** | ae(v) |
| if( cu\_qp\_delta\_palette\_abs ) |  |
| **cu\_qp\_delta\_palette\_sign\_flag** | ae(v) |
| } |  |
| if( cu\_chroma\_qp\_offset\_enabled\_flag && !IsCuChromaQpOffsetCoded ) { |  |
| **cu\_chroma\_qp\_palette\_offset\_flag** | ae(v) |
| if( cu\_chroma\_qp\_offset\_flag && chroma\_qp\_offset\_list\_len\_minus1 > 0 ) |  |
| **cu\_chroma\_qp\_palette\_offset\_idx** | ae(v) |
| } |  |
| } |  |
| if( indexMax > 0) |  |
| **palette\_transpose\_flag** | ae(v) |
| scanPos = 0 |  |
| while( scanPos < nCbS \* nCbS ) { |  |
| xC = x0 + travScan[ scanPos ][ 0 ] |  |
| yC = y0 + travScan[ scanPos ][ 1 ] |  |
| if( scanPos > 0) { |  |
| xcPrev = x0 + travScan[ scanPos − 1 ][ 0 ] |  |
| ycPrev = y0 + travScan[ scanPos − 1 ][ 1 ] |  |
| } |  |
| if( indexMax > 0 && ~~scanPos > = nCbS &&~~ ( scanPos = = 0 | | palette\_run\_type\_flag[ xcPrev ][ ycPrev ]  ! = COPY\_ABOVE\_MODE) ) { |  |
| **palette\_run\_type\_flag**[ xC ][ yC ] | ae(v) |
| } |  |
| if( palette\_run\_type\_flag[ xC ][ yC ] = = COPY\_INDEX\_MODE &&   adjustedIndexMax > 0) |  |
| **palette\_index\_idc** | ae(v) |
| if( indexMax > 0 ) { |  |
| maxPaletteRun = nCbS \* nCbS – scanPos – 1 |  |
| if( maxPaletteRun > 0 ) { |  |
| **palette\_run\_msb\_id\_plus1** | ae(v) |
| if( palette\_run\_msb\_id\_plus1 > 1 ) |  |
| **palette\_run\_refinement\_bits** | ae(v) |
| } |  |
| } else |  |
| paletteRun = nCbS \* nCbS − 1 |  |
| runPos = 0 |  |
| while ( runPos < = paletteRun ) { |  |
| xR = x0 + travScan[ scanPos ][ 0 ] |  |
| yR = y0 + travScan[ scanPos ][ 1 ] |  |
| if( palette\_run\_type\_flag[ xC ][ yC ] = = COPY\_INDEX\_MODE &&   paletteIndex = = indexMax ) { |  |
| PaletteSampleMode[ xR ][ yR ] = ESCAPE\_MODE |  |
| PaletteIndexMap[ xR ][ yR ] = paletteIndex |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| **palette\_escape\_val** | ae(v) |
| PaletteEscapeVal[ cIdx ][ xR ][ yR ] = palette\_escape\_val |  |
| } |  |
| } else if(palette\_run\_type\_flag[ xC ][ yC ] = = COPY\_INDEX\_MODE ) { |  |
| PaletteSampleMode[ xR ][ yR ] = COPY\_INDEX\_MODE |  |
| PaletteIndexMap[ xR ][ yR ] = paletteIndex |  |
| } else { |  |
| PaletteSampleMode[ xR ][ yR ] = COPY\_ABOVE\_MODE |  |
| if (scanPos < nCbS) |  |
| PaletteIndexMap[ xR ][ yR ] = 0 |  |
| else |  |
| PaletteIndexMap[ xR ][ yR ] = PaletteIndexMap[ xR ][ yR − 1 ] |  |
| if(scanPos > 0 && scanPos < nCbS && palette\_run\_type\_flag[xC\_prev][yC\_prev] == COPY\_INDEX\_MODE){ |  |
| if (PaletteIndexMap[xC\_prev][yC\_prev] ! = 0) |  |
| PaletteIndexMap[ xC ][ yC ] = 0 |  |
| else |  |
| PaletteIndexMap[ xC ][ yC ] = 1 |  |
| }else |  |
| if (scanPos > = nCbS && palette\_run\_type\_flag[xC\_prev][yC\_prev] == COPY\_INDEX\_MODE) { |  |
| if (PaletteIndexMap[xC\_prev][yC\_prev] == PaletteIndexMap[ xC ][ yC − 1 ]){ |  |
| if (PaletteIndexMap[xC\_prev][yC\_prev] ! = 0) |  |
| PaletteIndexMap[ xC ][ yC ] = 0 |  |
| else |  |
| PaletteIndexMap[ xC ][ yC ] = 1 |  |
| } |  |
| } |  |
| } |  |
| runPos++ |  |
| scanPos++ |  |
| } |  |
| } |  |
| } |  |

Table 9‑43 – Assignment of ctxInc to syntax elements with context coded bins

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| palette\_run\_type\_flag | 0, 1, 2 (subclause 9.4.1.2.x) | na | Na | na | na | na |

##### 9.4.1.2.x Derivation process of ctxInc for the syntax element palette\_run\_type\_flag

Inputs to this process are the luma location ( x0, y0 ) specifying the top-left luma sample of the current luma block relative to the top-left sample of the current picture, the current sample location ( xC, yC ) specifying the sample relative to the top-left luma sample of the picture, the sample location ( xC\_prev, yC\_prev ) specifying the sample relative to the previous sample(in decoding order) of the current luma sample of the picture, and the syntax element palette\_run\_type\_flag.

Output of this process is ctxInc.

The assignment of ctxInc for the syntax element palette\_run\_type\_flag is specified as follows:

* If palette\_run\_type\_flag[xC\_prev][yC\_prev] is equal to COPY\_INDEX\_MODE, and PaletteIndexMap[xC\_prev][yC\_prev] is equal to PaletteIndexMap[ xC ][ yC − 1 ], ctxInc is derived as follows:

ctxInc = 2 (9‑x)

* Otherwise, ctxInc is derived as follows:

ctxInc = (palette\_run\_type\_flag[xC][yC-1] == COPY\_ABOVE\_MODE) ? 1: 0 (9‑x)