#### 7.3.8.6 Palette coding syntax

|  |  |
| --- | --- |
| palette\_coding( x0, y0, cbWidth, cbHeight, startComp, numComps ) { | Descriptor |
| palettePredictionFinished = 0 |  |
| NumPredictedPaletteEntries = 0 |  |
| for( predictorEntryIdx = 0; predictorEntryIdx < PredictorPaletteSize[ startComp ] &&  !palettePredictionFinished &&   NumPredictedPaletteEntries[ startComp ] < palette\_max\_size; predictorEntryIdx++ ) { |  |
| **palette\_predictor\_run** | ae(v) |
| if( palette\_predictor\_run != 1 ) { |  |
| if( palette\_predictor\_run > 1 ) |  |
| predictorEntryIdx += palette\_predictor\_run − 1 |  |
| PalettePredictorEntryReuseFlags[ predictorEntryIdx ] = 1 |  |
| NumPredictedPaletteEntries++ |  |
| } else |  |
| palettePredictionFinished = 1 |  |
| } |  |
| if( NumPredictedPaletteEntries < palette\_max\_size ) |  |
| **num\_signalled\_palette\_entries** | ae(v) |
| for( cIdx = startComp; cIdx < ( startComp + numComps); cIdx++ ) |  |
| for( i = 0; i < num\_signalled\_palette\_entries; i++ ) |  |
| **new\_palette\_entries**[ cIdx ][ i ] | ae(v) |
| if( CurrentPaletteSize[ startComp ] > 0 ) |  |
| **palette\_escape\_val\_present\_flag** | ae(v) |
| if( MaxPaletteIndex > 0 ) { |  |
| **num\_palette\_indices\_minus1** | ae(v) |
| adjust = 0 |  |
| for( i = 0; i <= num\_palette\_indices\_minus1; i++ ) { |  |
| if( MaxPaletteIndex − adjust > 0 ) { |  |
| **palette\_idx\_idc** | ae(v) |
| PaletteIndexIdc[ i ] = palette\_idx\_idc |  |
| } |  |
| adjust = 1 |  |
| } |  |
| **copy\_above\_indices\_for\_final\_run\_flag** | ae(v) |
| **palette\_transpose\_flag** | ae(v) |
| } |  |
| if( treeType != DUAL\_TREE\_CHROMA && palette\_escape\_val\_present\_flag ) { |  |
| if( cu\_qp\_delta\_enabled\_flag && !IsCuQpDeltaCoded ) { |  |
| **cu\_qp\_delta\_abs** | ae(v) |
| if( cu\_qp\_delta\_abs ) |  |
| **cu\_qp\_delta\_sign\_flag** | ae(v) |
| } |  |
| } |  |
| if( treeType != DUAL\_TREE\_LUMA && palette\_escape\_val\_present\_flag ) { |  |
| if( cu\_chroma\_qp\_offset\_enabled\_flag && !IsCuChromaQpOffsetCoded ) { |  |
| **cu\_chroma\_qp\_offset\_flag** | ae(v) |
| if( cu\_chroma\_qp\_offset\_flag ) |  |
| **cu\_chroma\_qp\_offset\_idx** | ae(v) |
| } |  |
| } |  |
| remainingNumIndices = num\_palette\_indices\_minus1 + 1 |  |
| PaletteScanPos = 0 |  |
| log2CbWidth = Log2( cbWidth ) |  |
| log2CbHeight = Log2( cbHeight ) |  |
| while( PaletteScanPos < cbWidth\*cbHeightt ) { |  |
| xC = x0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos ][ 0 ] |  |
| yC = y0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos ][ 1 ] |  |
| if( PaletteScanPos > 0 ) { |  |
| xcPrev = x0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos − 1 ][ 0 ] |  |
| ycPrev = y0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos − 1 ][ 1 ] |  |
| } |  |
| PaletteRunMinus1 = cbWidth \* cbHeight − PaletteScanPos − 1 |  |
| RunToEnd = 1 |  |
| CopyAboveIndicesFlag[ xC ][ yC ] = 0 |  |
| if( MaxPaletteIndex > 0 ) |  |
| if( ( ( !palette\_transpose\_flag && yC > 0 ) | | ( palette\_transpose\_flag && xC > 0 ) )   && CopyAboveIndicesFlag[ xcPrev ][ ycPrev ] = = 0 ) |  |
| if( remainingNumIndices > 0 && PaletteScanPos < cbWidth\* cbHeight − 1 ) { |  |
| **copy\_above\_palette\_indices\_flag** | ae(v) |
| CopyAboveIndicesFlag[ xC ][ yC ] = copy\_above\_palette\_indices\_flag |  |
| } else { |  |
| if( PaletteScanPos = = cbWidth \* cbHeight − 1 && remainingNumIndices > 0 ) |  |
| CopyAboveIndicesFlag[ xC ][ yC ] = 0 |  |
| else |  |
| CopyAboveIndicesFlag[ xC ][ yC ] = 1 |  |
| } |  |
| if ( CopyAboveIndicesFlag[ xC ][ yC ] = = 0 ) { |  |
| currNumIndices = num\_palette\_indices\_minus1 + 1 − remainingNumIndices |  |
| CurrPaletteIndex = PaletteIndexIdc[ currNumIndices ] |  |
| } |  |
| if( MaxPaletteIndex > 0 ) { |  |
| if( CopyAboveIndicesFlag[ xC ][ yC ] = = 0 ) |  |
| remainingNumIndices − = 1 |  |
| if( remainingNumIndices > 0 | | CopyAboveIndicesFlag[ xC ][ yC ] !=  copy\_above\_indices\_for\_final\_run\_flag ) { |  |
| PaletteMaxRunMinus1 = cbWidth \* cbHeight − PaletteScanPos − 1 −   remainingNumIndices − copy\_above\_indices\_for\_final\_run\_flag |  |
| RunToEnd = 0 |  |
| if( PaletteMaxRunMinus1 > 0 ) { |  |
| **palette\_run\_prefix** | ae(v) |
| if( ( palette\_run\_prefix > 1 ) && ( PaletteMaxRunMinus1 !=  ( 1  <<  ( palette\_run\_prefix – 1 ) ) ) ) |  |
| **palette\_run\_suffix** | ae(v) |
| } |  |
| } |  |
| } |  |
| runPos = 0 |  |
| while ( runPos <= PaletteRunMinus1 ) { |  |
| xR = x0 +  TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos ][ 0 ] |  |
| yR = y0 + TraverseScanOrder[ log2CbWidth ][ log2CbHeight ][ PaletteScanPos ][ 1 ] |  |
| if( CopyAboveIndicesFlag[ xC ][ yC ] = = 0 ) { |  |
| CopyAboveIndicesFlag[ xR ][ yR ] = 0 |  |
| PaletteIndexMap[ xR ][ yR ] = CurrPaletteIndex |  |
| } else { |  |
| CopyAboveIndicesFlag[ xR ][ yR ] = 1 |  |
| if ( !palette\_transpose\_flag ) |  |
| PaletteIndexMap[ xR ][ yR ] = PaletteIndexMap[ xR ][ yR − 1 ] |  |
| else |  |
| PaletteIndexMap[ xR ][ yR ] = PaletteIndexMap[ xR − 1 ][ yR ] |  |
| } |  |
| runPos++ |  |
| PaletteScanPos ++ |  |
| } |  |
| } |  |
| if( palette\_escape\_val\_present\_flag ) { |  |
| for( cIdx = startComp; cIdx < ( startComp + numComps ); cIdx++ ) |  |
| for( sPos = 0; sPos < cbWidth\* cbHeight; sPos++ ) { |  |
| xC = x0 + TraverseScanOrder[ log2CbWidth][ log2CbHeight ][ sPos ][ 0 ] |  |
| yC = y0 + TraverseScanOrder[ log2CbWidth][ log2CbHeight ][ sPos ][ 1 ] |  |
| if( PaletteIndexMap[ cIdx ][ xC ][ yC ] = = MaxPaletteIndex ) { |  |
| **palette\_escape\_val** | ae(v) |
| PaletteEscapeVal[ cIdx ][ xC ][ yC ] = palette\_escape\_val |  |
| } |  |
| } |  |
| } |  |
| } |  |

### 8.5.8 Decoding process for the residual signal of coding blocks coded in inter prediction mode

Inputs to this process are:

* a sample location ( xTb0, yTb0 ) specifying the top-left sample of the current transform block relative to the top‑left sample of the current picture,
* a variable nTbW specifying the width of the current transform block,
* a variable nTbH specifying the height of the current transform block,
* a variable cIdx specifying the colour component of the current block.

Output of this process is an (nTbW)x(nTbH) array resSamples.

The maximum transform block width maxTbWidth and height maxTbHeight are derived as follows:

maxTbWidth = ( cIdx  = =  0 ) ? MaxTbSizeY : MaxTbSizeY / SubWidthC (8‑883)

maxTbHeight = ( cIdx  = =  0 ) ? MaxTbSizeY : MaxTbSizeY / SubHeightC (8‑884)

The luma sample location is derived as follows:

( xTbY, yTbY ) = ( cIdx  = =  0 ) ? ( xTb0, yTb0 ) : ( xTb0 \* SubWidthC, yTb0 \* SubHeightC ) (8‑885)

Depending on maxTbSize, the following applies:

* If nTbW is greater than maxTbWidth or nTbH is greater than maxTbHeight, the following ordered steps apply.

1. The variables newTbW and newTbH are derived as follows:

newTbW = ( nTbW  >  maxTbWidth ) ? ( nTbW / 2 ) : nTbW (8‑886)

newTbH = ( nTbH   >  maxTbHeight ) ? ( nTbH / 2 ) :  nTbH (8‑887)

1. The decoding process process for the residual signal of coding units coded in inter prediction mode as specified in this clause is invoked with the location ( xTb0, yTb0 ), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and the variable cIdx as inputs, and the output is a modified reconstructed picture before in-loop filtering.
2. When nTbW is greater than maxTbWidth, the decoding process process for the residual signal of coding units coded in inter prediction mode as specified in this clause is invoked with the location ( xTb0, yTb0 ) set equal to ( xTb0 + newTbW, yTb0 ), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and the variable cIdx as inputs, and the output is a modified reconstructed picture .
3. When nTbH is greater than maxTbHeight, the decoding process process for the residual signal of coding units coded in inter prediction mode as specified in this clause is invoked with the location ( xTb0, yTb0 ) set equal to ( xTb0, yTb0 + newTbH ), the transform block width nTbW set equal to newTbW and the height nTbH set equal to newTbH, and the variable cIdx as inputs, and the output is a modified reconstructed picture before in-loop filtering.
4. When nTbW is greater than maxTbWidth and nTbH is greater than maxTbHeight, the decoding process process for the residual signal of coding units coded in inter prediction mode as specified in this clause is invoked with the location ( xTb0, yTb0 ) set equal to ( xTb0 + newTbW, yTb0 + newTbH ), the transform block width nTbW set equal to newTbW and height nTbH set equal to newTbH, and the variable cIdx as inputs, and the output is a modified reconstructed picture before in-loop filtering.

* Otherwise, if cu\_sbt\_flag is equal to 1, the following applies:
* The variables sbtMinNumFourths, wPartIdx and hPartIdx are derived as follows:

sbtMinNumFourths = cu\_sbt\_quad\_flag  ?  1  :  2 (8‑888)

wPartIdx = cu\_sbt\_horizontal\_flag ? 4 : sbtMinNumFourths (8‑889)

hPartIdx = !cu\_sbt\_horizontal\_flag ? 4 : sbtMinNumFourths (8‑890)

* The variables xPartIdx and yPartIdx are derived as follows:
* If cu\_sbt\_pos\_flag is equal to 0, xPartIdx and yPartIdx are set equal to 0.
* Otherwise (cu\_sbt\_pos\_flag is equal to 1), the variables xPartIdx and yPartIdx are derived as follows:

xPartIdx = cu\_sbt\_horizontal\_flag ? 0 : ( 4 − sbtMinNumFourths ) (8‑891)

yPartIdx = !cu\_sbt\_horizontal\_flag ? 0 : ( 4 − sbtMinNumFourths ) (8‑892)

* The variables xTbYSub, yTbYSub, xTb0Sub, yTb0Sub, nTbWSub and nTbHSub are derived as follows:

xTbYSub = xTbY + ( ( nTbW \* ( ( cIdx  = =  0 ) ? 1: SubWidthC ) \* xPartIdx / 4 ) (8‑893)

yTbYSub = yTbY + ( ( nTbH \* ( ( cIdx  = =  0 ) ? 1: SubHeightC ) \* yPartIdx / 4 ) (8‑894)

xTb0Sub = xTb0 + ( nTbW \* xPartIdx / 4 ) (8‑895)

yTb0Sub = yTb0 + ( nTbH \* yPartIdx / 4 ) (8‑896)

nTbWSub = nTbW \* wPartIdx / 4 (8‑897)

nTbHSub = nTbH \* hPartIdx / 4 (8‑898)

* The scaling and transformation process as specified in clause 8.7.2 is invoked with the luma location ( xTbYSub , yTbYSub ), the variable cIdx, nTbWSub and nTbHSub as inputs, and the output is an ( nTbWSub )x( nTbHSub ) array resSamplesTb.
* The residual samples resSamples[ x ][ y ] with x = 0..nTbW − 1, y = 0..nTbH − 1 are set equal to 0.
* The residual samples resSamples[ x ][ y ] with x = xTb0Sub..xTb0Sub + nTbWSub – 1, y = yTb0Sub..yTb0Sub + nTbHSub – 1 are derived as follows:

resSamples[ x ][ y ] = resSamplesTb[ x – xTb0Sub ][ y – yTb0Sub ] (8‑899)

* Otherwise, the scaling and transformation process as specified in clause 8.7.2 is invoked with the luma location ( xTbY, yTbY ), the variable cIdx, the transform width nTbW and the transform height nTbH as inputs, and the output is an (nTbW)x(nTbH) array resSamples.

#### 8.6.2.4 Derivation process for IBC history-based block vector candidates

Inputs to this process are:

* a block vector candidate list bvCandList,
* a variable isInSmr specifying whether the current coding unit is inside a shared merging candidate region,
* the number of available block vector candidates in the list numCurrCand.

Outputs to this process are:

* the modified block vector candidate list bvCandList,
* the modified number of motion vector candidates in the list numCurrCand.

The variables isPrunedA1 and isPrunedB1 are set both equal to FALSE.

For each candidate in smrHmvpIbcCandList[ hMvpIdx ] with index hMvpIdx = 1..smrNumHmvpIbcCand, the following ordered steps are repeated until numCurrCand is equal to MaxNumIbcMergeCand:

1. The variable sameMotion is derived as follows:
   * If all of the following conditions are true for any block vector candidate N with N being A1 or B1, sameMotion and isPrunedN are both set equal to TRUE:
   * hMvpIdx is equal to 1.
   * The candidate HmvpIbcCandList[NumHmvpIbcCand − hMvpIdx] is equal to the block vector candidate N.
   * isPrunedN is equal to FALSE.
   * Otherwise, sameMotion is set equal to FALSE.
2. When sameMotion is equal to FALSE, the candidate HmvpIbcCandList[NumHmvpIbcCand − hMvpIdx] is added to the block vector candidate list as follows:

bvCandList[ numCurrCand++ ] = HmvpIbcCandList[ NumHmvpIbcCand − hMvpIdx ] (8‑916)

##### 8.8.3.6.3 Decision process for chroma block edges

This process is only invoked when ChromaArrayType is not equal to 0.

Inputs to this process are:

* a chroma picture sample array recPicture,
* a chroma location ( xCb, yCb ) specifying the top-left sample of the current chroma coding block relative to the top-left chroma sample of the current picture,
* a chroma location ( xBl, yBl ) specifying the top-left sample of the current chroma block relative to the top-left sample of the current chroma coding block,
* a variable edgeType specifying whether a vertical (EDGE\_VER) or a horizontal (EDGE\_HOR) edge is filtered,
* a variable cIdx specifying the colour component index,
* a variable cQpPicOffset specifying the picture-level chroma quantization parameter offset,
* a variable bS specifying the boundary filtering strength,
* a variable maxFilterLengthCbCr.

Outputs of this process are

* the modified variable maxFilterLengthCbCr,
* the variable tC.

The variable maxK is derived as follows:

* If edgeType is equal to EDGE\_VER, the following applies:

maxK = ( SubHeightC = = 1 ) ? 3 : 1 (8‑1124)

* Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

maxK = ( SubWidthC = = 1 ) ? 3 : 1 (8‑1125)

The values pi and qi with i = 0.. maxFilterLengthCbCr and k = 0..maxK are derived as follows:

* If edgeType is equal to EDGE\_VER, the following applies::

qi,k = recPicture[ xCb + xBl + i ][ yCb + yBl + k ] (8‑1126)

pi,k = recPicture[ xCb + xBl − i − 1 ][ yCb + yBl + k ] (8‑1127)

subSampleC = SubHeightC (8‑1128)

* Otherwise (edgeType is equal to EDGE\_HOR), the following applies:

qi,k = recPicture[ xCb + xBl + k ][ yCb + yBl + i ] (8‑1129)

pi,k = recPicture[ xCb + xBl + k ][ yCb + yBl − i − 1 ] (8‑1130)

subSampleC = SubWidthC (8‑1131)

The variables QpQ and QpP are set equal to the QpY values of the coding units which include the coding blocks containing the sample q0,0 and p0,0, respectively.

The variable QpC is derived as follows:

qPi = Clip3( 0, 63, ( ( QpQ + QpP + 1 )  >>  1 ) + cQpPicOffset ) (8‑1132)

QpC = ChromaQpTable[ cIdx − 1 ][ qPi ] (8‑1133)

NOTE – The variable cQpPicOffset provides an adjustment for the value of pps\_cb\_qp\_offset or pps\_cr\_qp\_offset, according to whether the filtered chroma component is the Cb or Cr component. However, to avoid the need to vary the amount of the adjustment within the picture, the filtering process does not include an adjustment for the value of slice\_cb\_qp\_offset or slice\_cr\_qp\_offset nor (when cu\_chroma\_qp\_offset\_enabled\_flag is equal to 1) for the value of CuQpOffsetCb, CuQpOffsetCr, or CuQpOffsetCbCr.

The value of the variable β′ is determined as specified in Table 8‑18 based on the quantization parameter Q derived as follows:

Q = Clip3( 0, 63, QpC + ( slice\_beta\_offset\_div2  <<  1 ) ) (8‑1134)

where slice\_beta\_offset\_div2 is the value of the syntax element slice\_beta\_offset\_div2 for the slice that contains sample q0,0.

The variable β is derived as follows:

β = β′ \* ( 1  <<  ( BitDepthC − 8 ) ) (8‑1135)

The value of the variable tC′ is determined as specified in Table 8‑18 based on the chroma quantization parameter Q derived as follows:

Q = Clip3( 0, 65, QpC + 2 \* ( bS − 1 ) + ( slice\_tc\_offset\_div2  <<  1 ) ) (8‑1136)

where slice\_tc\_offset\_div2 is the value of the syntax element slice\_tc\_offset\_div2 for the slice that contains sample q0,0.

The variable tC is derived as follows:

tC = ( BitDepthC < 10 ) ? ( tC′ + 2 ) >> ( 10 − BitDepthC ) : tC′ \* ( 1  <<  ( BitDepthC − 8 ) ) (8‑1137)

When maxFilterLengthCbCr is equal to 1 and bS is not equal to 2, maxFilterLengthCbCr is set equal to 0.

When maxFilterLengthCbCr is equal to 3, the following ordered steps apply:

1. The variables n1, dpq0, dpq1, dp, dq and d are derived as follows:

n1 = ( subSampleC = = 2 ) ? 1 : 3 (8‑1138)

dp0 = Abs( p2,0 − 2 \* p1,0 + p0,0 ) (8‑1139)

dp1 = Abs( p2,n1 − 2 \* p1,n1 + p0,n1 ) (8‑1140)

dq0 = Abs( q2,0 − 2 \* q1,0 + q0,0 ) (8‑1141)

dq1 = Abs( q2,n1 − 2 \* q1,n1 + q0,n1 ) (8‑1142)

dpq0 = dp0 + dq0 (8‑1143)

dpq1 = dp1 + dq1 (8‑1144)

dp = dp0 + dp1 (8‑1145)

dq = dq0 + dq1 (8‑1146)

d = dpq0 + dpq1 (8‑1147)

1. The variables dSam0 and dSam1 are both set equal to 0.
2. When d is less than β, the following ordered steps apply:
3. The variable dpq is set equal to 2 \* dpq0.
4. The variable dSam0 is derived by invoking the decision process for a chroma sample as specified in clause 8.8.3.6.8 for the sample location ( xCb + xBl, yCb + yBl ) with sample values p0,0, p3,0, q0,0, and q3,0, the variables dpq, β and tC as inputs, and the output is assigned to the decision dSam0.
5. The variable dpq is set equal to 2 \* dpq1.
6. The variable dSam1 is modified as follows:

* If edgeType is equal to EDGE\_VER, for the sample location ( xCb + xBl, yCb + yBl + n1 ), the decision process for a chroma sample as specified in clause 8.8.3.6.8 is invoked with sample values p0,n1, p3,n1, q0,n1, and q3,n1, the variables dpq, β and tC as inputs, and the output is assigned to the decision dSam1.
* Otherwise (edgeType is equal to EDGE\_HOR), for the sample location ( xCb + xBl + n1, yCb + yBl ), the decision process for a chroma sample as specified in clause 8.8.3.6.8 is invoked with sample values p0,n1, p3,n1, q0,n1 and q3,n1, the variables dpq, β and tC as inputs, and the output is assigned to the decision dSam1.

1. The variable maxFilterLengthCbCr is modified as follows:

* If dSam0 is equal to 1 and dSam1 is equal to 1, maxFilterLengthCbCr is set equal to 3.
* Otherwise, maxFilterLengthCbCr is set equal to 1.