#### Palette syntax

|  |  |
| --- | --- |
| palette\_coding( x0, y0, nCbS ) { | Descriptor |
| palettePredictionFinished = 0 |  |
| NumPredictedPaletteEntries = 0 |  |
| for( i = 0; i < PredictorPaletteSize && !palettePredictionFinished &&  NumPredictedPaletteEntries < palette\_max\_size; i++ ) { |  |
| **palette\_predictor\_run** | ue(v) |
| if( palette\_predictor\_run != 1 ) { |  |
| if( palette\_predictor\_run > 1 ) |  |
| i += palette\_predictor\_run − 1 |  |
| PalettePredictorEntryReuseFlag[ i ] = 1 |  |
| NumPredictedPaletteEntries++ |  |
| } else |  |
| palettePredictionFinished = 1 |  |
| } |  |
| if( NumPredictedPaletteEntries < palette\_max\_size ) |  |
| **num\_signalled\_palette\_entries** | ue(v) |
| numComps = ( ChromaArrayType = = 0 ) ? 1 : 3 |  |
| for( cIdx = 0; cIdx < numComps; cIdx++ ) |  |
| for( i = 0; i < num\_signalled\_palette\_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( MaxPaletteIndex > 0) { |  |
| **palette\_transpose\_flag**[ x0 ][ y0 ] | ae(v) |
| **num\_palette\_indices\_idc** | ae(v) |
| for( i=0; i < NumPaletteIndices; i++ ) { |  |
| **palette\_index\_idc** | ae(v) |
| PaletteIndexIdc[ i ] = palette\_index\_idc |  |
| } |  |
| **last\_palette\_run\_type\_flag** | ae(v) |
| } |  |
| CurrNumIndices = 0 |  |
| PaletteScanPos = 0 |  |
| while( PaletteScanPos < nCbS \* nCbS ) { |  |
| xC = x0 + travScan[ PaletteScanPos ][ 0 ] |  |
| yC = y0 + travScan[ PaletteScanPos ][ 1 ] |  |
| if( PaletteScanPos > 0) { |  |
| xcPrev = x0 + travScan[ PaletteScanPos − 1 ][ 0 ] |  |
| ycPrev = y0 + travScan[ PaletteScanPos − 1 ][ 1 ] |  |
| } |  |
| PaletteRun = nCbS \* nCbS − PaletteScanPos − 1 |  |
| if( MaxPaletteIndex > 0 && CurrNumIndices < NumPaletteIndices ) { |  |
| if( ~~PaletteScanPos >= nCbS &&~~ scanPos = = 0 ||  (palette\_run\_type\_flag[ xcPrev ][ ycPrev ] != COPY\_ABOVE\_MODE   && PaletteScanPos < nCbS \* nCbS – 1)) { |  |
| **palette\_run\_type\_flag**[ xC ][ yC ] | ae(v) |
| } |  |
| readIndex = 0 |  |
| if( palette\_run\_type\_flag[ xC ][ yC ] = = COPY\_INDEX\_MODE &&   AdjustedMaxPaletteIndex > 0) { |  |
| readIndex = 1 |  |
| if (PaletteScanPos !=0 && palette\_run\_type\_flag[ xcPrev ][ ycPrev ] == COPY\_ABOVE\_MODE && PaletteIndexIdc[CurrNumIndices + readIndex] = = (indexMax – 1) && ((yR == y0) || PaletteIndexMap[ xR ][ yR − 1 ]==-1 )) |  |
| **palette\_index\_refinement\_flag** | ae(v) |
| } |  |
| maxPaletteRun = nCbS \* nCbS – PaletteScanPos – 1 |  |
| if( AdjustedMaxPaletteIndex > 0 &&   ( ( CurrNumIndices + readIndex ) < NumPaletteIndices | |   palette\_run\_type\_flag[ xC ][ yC ] != last\_palette\_run\_type\_flag ) ) |  |
| if( maxPaletteRun > 0 ) { |  |
| **palette\_run\_msb\_id\_plus1** | ae(v) |
| if( palette\_run\_msb\_id\_plus1 > 1 ) |  |
| **palette\_run\_refinement\_bits** | ae(v) |
| } |  |
| CurrNumIndices + = readIndex |  |
| } |  |
| runPos = 0 |  |
| while ( runPos < = paletteRun ) { |  |
| xR = x0 + travScan[ PaletteScanPos ][ 0 ] |  |
| yR = y0 + travScan[ PaletteScanPos ][ 1 ] |  |
| if(palette\_run\_type\_flag[ xC ][ yC ] = = COPY\_INDEX\_MODE ) { |  |
| PaletteSampleMode[ xR ][ yR ] = COPY\_INDEX\_MODE |  |
| PaletteIndexMap[ xR ][ yR ] = CurrPaletteIndex |  |
| } else { |  |
| PaletteSampleMode[ xR ][ yR ] = COPY\_ABOVE\_MODE |  |
| PaletteIndexMap[ xR ][ yR ] = (yR == y0) ? −1:PaletteIndexMap[ xR ][ yR − 1 ] |  |
| } |  |
| runPos++ |  |
| PaletteScanPos++ |  |
| } |  |
| } |  |
| if( palette\_escape\_val\_present\_flag ) { |  |
| sPos = 0 |  |
| while( sPos < nCbS \* nCbS ) { |  |
| xC = x0 + travScan[ sPos ][ 0 ] |  |
| yC = y0 + travScan[ sPos ][ 1 ] |  |
| if( PaletteIndexMap[ xC ][ yC ] =  = MaxPaletteIndex ) { |  |
| for( cIdx = 0; cIdx < numComps; cIdx++ ) |  |
| if( cIdx = = 0 | |   ( xR % 2 = = 0 && yR % 2 = = 0 && ChromaArrayType = = 1 ) | |  ( xR % 2 = = 0 && ChromaArrayType = = 2 ) | |   ChromaArrayType = = 3 ) { |  |
| **palette\_escape\_val** | ae(v) |
| PaletteEscapeVal[ cIdx ][ xC ][ yC ] = palette\_escape\_val |  |
| } |  |
| } |  |
| sPos++ |  |
| } |  |
| } |  |
| } |  |

#### Palette mode semantics

In the following semantics, the array indices x0, y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

The predictor palette consists of palette entries from previous coding units that are used to predict the entries in the current palette.

The variable PredictorPaletteSize specifies the size of the predictor palette. PredictorPaletteSize is derived as specified in clause 8.4.4.2.7.

The value of PredictorPaletteSize shall be in the range of 0 to max\_predictor\_palette\_size, inclusive.

The variable PalettePredictorEntryReuseFlag[ i ]equal to 1specifies that the i-th entry in the predictor palette is reused in the current palette. PalettePredictorEntryReuseFlag[ i ] equal to 0specifies that the i-th entry in the predictor palette is an entry in the current palette. All elements of the array PalettePredictorEntryReuseFlag[ i ] are initialized to be equal to zero.

**palette\_predictor\_run** specifies the number of zeros that precede a non-zero entry in the array predictor\_palette\_entry\_reuse\_flag. The value of palette\_predictor\_run shall be in the range of 0 to palette\_max\_predictor\_size, inclusive.

The variable NumPredictedPaletteEntries specifies the number of entries in the current palette that are reused from the predictor palette. The value of NumPredictedPaletteEntries shall be in the range of 0 to palette\_max\_size, inclusive.

**num\_signalled\_palette\_entries** specifies the number of entries in the current palette that are explicitly signalled.

When num\_signalled\_palette\_entries is not present, it is inferred to be equal to 0.

The variable CurrentPaletteSize specifies the size of the current palette and is derived as follows:

CurrentPaletteSize = NumPredictedPaletteEntries + num\_signalled\_palette\_entries (7‑72)

The value of CurrentPaletteSize shall be in the range of 0 to palette\_max\_size, inclusive.

**palette\_entry** specifiesthe value of a component in a palette entry for the current palette.

The variable PredictorPaletteEntries[ cIdx ][ i ]specifiesthe i-th element in the predictor palette for the colour component cIdx.

The variable numComps is derived as follows:

numComps = ( ChromaArrayType = = 0 ) ? 1 : 3 (7‑73)

The variable CurrentPaletteEntries[ cIdx ][ i ]specifiesthe i-th element in the current palette for the colour component cIdx and is derived as follows:

numPredictedPaletteEntries = 0  
for( i = 0; i < PredictorPaletteSize; i++ )   
 if( PalettePredictorEntryReuseFlag[ i ] **=** **=** 1 ) {  
 for( cIdx = 0; cIdx < numComps; cIdx++ )   
 CurrentPaletteEntries[ cIdx ][ numPredictedPaletteEntries ] = PredictorPaletteEntries[ cIdx ][ i ]  
 numPredictedPaletteEntries++   
 }  
for( cIdx = 0; cIdx < numComps; cIdx++ ) (7‑74)  
 for( i = 0; i < num\_signalled\_palette\_entries; i++ )   
 CurrentPaletteEntries[ cIdx ][ numPredictedPaletteEntries + i ] = palette\_entry

**palette\_escape\_val\_present\_flag** equal to 1 specifies that the current coding unit contains at least one escape coded sample. escape\_val\_present\_flag equal to 0 specifies that there are no escape coded samples in the current coding unit. When not present, the value of palette\_escape\_val\_present\_flag is inferred to be equal to 1.

The variable MaxPaletteIndex specifies the maximum possible value for a palette index for the current coding unit. The value of MaxPaletteIndex is set equal to CurrentPaletteSize – 1 + palette\_escape\_val\_present\_flag.

**cu\_qp\_delta\_palette\_abs** specifies the absolute value of the difference CuQpDeltaVal between the luma quantization parameter of the current coding unit and its prediction.

**cu\_qp\_delta\_palette\_sign\_flag** specifies the sign of CuQpDeltaVal as follows:

* If cu\_qp\_delta\_palette\_sign\_flag is equal to 0, the corresponding CuQpDeltaVal has a positive value.
* Otherwise (cu\_qp\_delta\_palette\_sign\_flag is equal to 1), the corresponding CuQpDeltaVal has a negative value.

When cu\_qp\_delta\_palette\_sign\_flag is not present, it is inferred to be equal to 0.

When cu\_qp\_delta\_palette\_abs is present, the variables IsCuQpDeltaCoded and CuQpDeltaVal are derived as follows:

IsCuQpDeltaCoded = 1 (7‑75)

CuQpDeltaVal = cu\_qp\_delta\_palette\_abs \* ( 1 − 2 \* cu\_qp\_delta\_palette\_sign\_flag ) (7‑76)

The value of CuQpDeltaVal shall be in the range of −( 26 + QpBdOffsetY / 2 ) to +( 25 + QpBdOffsetY / 2 ), inclusive.

**cu\_chroma\_qp\_palette\_offset\_flag**, when present and equal to 1, specifies that an entry in the cb\_qp\_offset\_list[ ] is used to determine the value of CuQpOffsetCb and a corresponding entry in the cr\_qp\_offset\_list[ ] is used to determine the value of CuQpOffsetCr. cu\_chroma\_qp\_palette\_offset\_flag equal to 0 specifies that these lists are not used to determine the values of CuQpOffsetCb and CuQpOffsetCr.

**cu\_chroma\_qp\_palette\_offset\_idx**, when present, specifies the index into the cb\_qp\_offset\_list[ ] and cr\_qp\_offset\_list[ ] that is used to determine the value of CuQpOffsetCb and CuQpOffsetCr. When present, the value of cu\_chroma\_qp\_palette\_offset\_idx shall be in the range of 0 to chroma\_qp\_offset\_list\_len\_minus1, inclusive. When not present, the value of cu\_chroma\_qp\_palette\_offset\_idx is inferred to be equal to 0.

When cu\_chroma\_qp\_palette\_offset\_flag is present, the following applies:

* The variable IsCuChromaQpOffsetCoded is set equal to 1.
* The variables CuQpOffsetCb and CuQpOffsetCr are derived as follows:
* If cu\_chroma\_qp\_palette\_offset\_flag is equal to 1, the following applies:

CuQpOffsetCb = cb\_qp\_offset\_list[ cu\_chroma\_qp\_offset\_idx ] (7‑77)

CuQpOffsetCr = cr\_qp\_offset\_list[ cu\_chroma\_qp\_offset\_idx ] (7‑78)

* Otherwise (cu\_chroma\_qp\_palette\_offset\_flag is equal to 0), CuQpOffsetCb and CuQpOffsetCr are both set equal to 0.

NOTE – When cu\_chroma\_qp\_offset\_enabled\_flag is equal to 0, CuQpOffsetCb and CuQpOffsetCr are not modified after being initialized to 0 for the slice as specified in clause 7.4.7.1.

**palette\_transpose\_flag**[ x0 ][ y0 ] equal to 1 specifies the transpose process is applied to the associated palette indices of the current coding unit. palette\_transpose\_flag[ x0 ][ y0 ] equal to 0 specifies the transpose process is not applied to the associated palette indices of the current coding unit. The array indices x0 and y0 specify the location ( x0, y0 ) of the top-left luma sample of the considered coding block relative to the top-left luma sample of the picture.

If palette\_transpose\_flag[ x0 ][ y0 ] is equal to 1, the reconstructed palette block is transposed as specified in clause 8.4.1.

**num\_palette\_indices\_idc** is an indication of the number of palette indices signalled for the current block.

When num\_palette\_indices\_idc is not present, it is inferred to be equal to 0.

The variable NumPaletteIndices specifies the number of palette indices signalled for the current block and is derived as follows:

if( num\_palette\_indices\_idc >= ( MaxPaletteIndex – 1) \* 32 )  
 NumPaletteIndices = num\_palette\_indices\_idc + 1  
else if(num\_palette\_indices\_idc % 32 = = 31)  
 NumPaletteIndices = MaxPaletteIndex – (num\_palette\_indices\_idc + 1 ) / 32 (7‑79)  
else  
 NumPaletteIndices = ( num\_palette\_indices\_idc / 32 ) \* 31 ) + (num\_palette\_indices\_idc % 32) + MaxPaletteIndex

**palette\_index\_idc** is an indication of an index to the array represented by currentPaletteEntries. The value of palette\_index\_idc shall be in the range of 0 to MaxPaletteIndex, inclusive, for the first index in the block and in the range of 0 to ( MaxPaletteIndex – 1 ), inclusive for the remaining indices in the block.

When palette\_index\_idc is not present, it is inferred to be equal to 0.

**palette\_last\_run\_type\_flag** specifies the last occurence of the palette\_run\_type\_flag within the block.

When palette\_last\_run\_type\_flag is not present, it is inferred to be equal to COPY\_INDEX\_MODE.

**palette\_run\_type\_flag**[ xC ][ yC ]equal to COPY\_ABOVE\_MODE specifies that the palette index is equal to the palette index at the same location in the row above. palette\_run\_type\_flag[ xC ][ yC ] equal to COPY\_INDEX\_MODE specifies that an indication of the palette index of the sample is coded in the bitstream. The array indices xC, yC specify the location ( xC , yC ) of the sample relative to the top-left luma sample of the picture.

The variable CurrNumIndices specifies the number of palette runs of type COPY\_INDEX\_MODE up to but not including the current scan position.

When palette\_run\_type\_flag is not present, it is inferred as follows:

* If CurrNumIndices is equal to NumPaletteIndices and MaxPaletteIndex is greater than 0, palette\_run\_type\_flag is inferred to be equal to COPY\_ABOVE\_MODE.
* Otherwise ( CurrNumIndices is less than NumPaletteIndices or MaxPaletteIndex is equal to 0 ) palette\_run\_type\_flag is inferred to be equal to COPY\_INDEX\_MODE

The variable AdjustedMaxPaletteIndex is derived as follows:

AdjustedMaxPaletteIndex = MaxPaletteIndex  
if( PaletteScanPos > 0 )   
 AdjustedMaxPaletteIndex − = 1

**palette\_index\_refinement\_flag** specifies whether the palette index should be incremented by 1.

When palette\_index\_refinement\_flag is not present, it is inferred to be equal to 0.

The variable PaletteIndexMap[ xC ][ yC ] specifes a palette index, which is an index to the array represented by CurrentPaletteEntries. The array indices xC, yC specify the location ( xC , yC ) of the sample relative to the top-left luma sample of the picture. The value of PaletteIndexMap[ xC ][ yC ] shall be in the range of -1~~0~~ to MaxPaletteIndex, inclusive.

The variable adjustedRefPaletteIndex is derived as follows:

adjustedRefPaletteIndex = MaxPaletteIndex + 1  
if( PaletteScanPos > 0 ) {   
 if( palette\_run\_type\_flag[xcPrev][ycPrev] != COPY\_ABOVE\_MODE ) {  
 xcPrev = xC + travScan[PaletteScanPos – 1][0]  
 ycPrev = yC + travScan[PaletteScanPos – 1][1]  
 adjustedRefPaletteIndex = PaletteIndexMap[xcPrev][ycPrev] (7‑80)  
 }  
 else if(scanPos >= nCbS && PaletteIndexMap[xC][yC − 1]!= -1 )  
 adjustedRefPaletteIndex = PaletteIndexMap[xC][yC − 1]

When PaletteSampleMode[ xC ][ yC ] is not equal to COPY\_ABOVE\_MODE, the variable CurrPaletteIndex is derived as follows:

CurrPaletteIndex = PaletteIndexIdc[CurrNumIndices]  
if( CurrPaletteIndex >= adjustedRefPaletteIndex )   
 CurrPaletteIndex +  = 1

The variable PaletteSampleMode[ xC ][ yC ] specifies whether the palette index is copied from the palette index in the row above or explicitly coded in the bitstream. The array indices xC, yC specify the location ( xC , yC ) of the sample relative to the top-left luma sample of the picture. The value of PaletteSampleMode[ xC ][ yC ] shall be one of COPY\_ABOVE\_MODE and COPY\_INDEX\_MODE.

If PaletteIndexMap[ xC ][ yC ] is equal to MaxPaletteIndex and palette\_escape\_val\_present\_flag is equal to 1, the sample is coded in escape mode and PaletteSampleMode[ xC ][ yC ] is set equal to ESCAPE\_MODE.

The variable paletteRun specifies the number of consecutive locations minus 1 with the same palette index as the position in the above row when palette\_run\_type\_flag is equal to COPY\_ABOVE\_MODE or specifies the number of consecutive locations minus 1 with the same palette index when palette\_run\_type\_flag is equal to COPY\_INDEX\_MODE.

**palette\_run\_msb\_id\_plus1** minus 1 specifies the index of the most significant bit in the binary representation of paletteRun. The value of palette\_run\_msb\_id\_plus1 shall be in the range of 0 to Floor( Log2( ( MaxPaletteRun ) ) + 1, inclusive.

**palette\_run\_refinement\_bits** specifies the refinement bits in the binary representation of paletteRun.

When palette\_run**\_**refinement\_bits is not present, it is inferred to be equal to 0.

When palette\_run\_msb\_id\_plus1 is present, the variable paletteRun is derived as follows:

* If palette\_run**\_**msb\_id\_plus1 is greater than 1,

paletteRun = ( 1 << ( palette\_run\_msb\_id\_plus1 − 1 ) ) + palette\_run**\_**refinement\_bits (7‑81)

* Otherwise ( palette\_run\_msb\_id\_plus1 equal to to 0 or 1 ) paletteRun is set equal to palette\_run\_msb\_id\_plus1.

**palette\_escape\_val** specifies the quantized escape coded sample value for a component.

The variable PaletteEscapeVal**[** cIdx **][** xC **][** yC **]** specifies escape value of a sample for which PaletteIndexMap[ xC ][ yC ] is equal to MaxPaletteIndex and palette\_escape\_val\_present\_flag is equal to 1. The array index cIdx specifies the colour component. The array indices xC, yC specify the location ( xC, yC ) of the sample relative to the top-left luma sample of the picture.

##### 

### 

### General decoding process for coding units coded in intra prediction mode

Inputs to this process are:

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

– a variable log2CbSize specifying the size of the current luma coding block.

Output of this process is a modified reconstructed picture before deblocking filtering.

The derivation process for quantization parameters as specified in clause is invoked with the luma location ( xCb, yCb ) as input.

A variable nCbS is set equal to 1  <<  log2CbSize.

Depending on the values of pcm\_flag[ xCb ][ yCb ], palette\_mode\_flag[ xCb ][ yCb ], and IntraSplitFlag, the decoding process for luma samples is specified as follows:

– If pcm\_flag[ xCb ][ yCb ] is equal to 1, the reconstructed picture is modified as follows:

SL[ xCb + i ][ yCb + j ] =   
 pcm\_sample\_luma[ ( nCbS \* j ) + i ]  <<  ( BitDepthY − PcmBitDepthY ), with i, j = 0..nCbS − 1 (8‑12)

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0), if palette\_mode\_flag[ xCb ][ yCb ] is equal to 1, the following order steps apply:

1. The decoding process for palette intra blocks as specified in clause 8.4.4.2.7 is invoked with the luma location ( xCb, yCb ), the variable cIdx set equal to 0, and nCbSX and nCbSY both set equal to nCbS as inputs, and the output is an nCbS x nCbS array of reconstructed palette sample values, recSamples[ x ][ y ], x, y = 0..nCbS − 1
2. The reconstructed picture is modified as follows:

– If palette\_transpose\_flag[ xCb ][ yCb ] is true,

SL[ yCb + y ][ xCb + x ] = recSamples[ x ][ y ] (8‑13)

– Otherwise (palette\_transpose\_flag[ xCb ][ yCb ] is false)

SL[ xCb + x ][ yCb + y ] = recSamples[ x ][ y ] (8‑14)

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, palette\_mode\_flag[ xCb ][ yCb ] is equal to 0 ), if IntraSplitFlag is equal to 0, the following ordered steps apply:

1. The derivation process for the intra prediction mode as specified in clause 8.4.2 is invoked with the luma location ( xCb, yCb ) as input.
2. If cu\_residual\_act\_flag[ xCb ][ yCb ] is equal to 1, the residual modification process for residual blocks using adaptive colour transform as specified in clause is invoked with the variable blkSize set equal to nCbS, the (nCbS)x(nCbS) array rY set equal to resSamplesL, the (nCbS)x(nCbS) array rCb set equal to resSamplesCb, and the (nCbS)x(nCbS) array rCr set equal to resSamplesCr as inputs, and the output are modified versions of the (nCbS)x(nCbS) arrays resSamplesL, resSamplesCb and resSamplesCr.
3. The general decoding process for intra blocks as specified in clause 8.4.4.1 is invoked with the luma location ( xCb, yCb ), the variable log2TrafoSize set equal to log2CbSize, the variable trafoDepth set equal to 0, the variable predModeIntra set equal to IntraPredModeY[ xCb ][ yCb ] and the variable cIdx set equal to 0, and variable controlParaAct set to (cu\_residual\_act\_flag[ xCb ][ yCb ] ? 2 : 3 ) as inputs, and the output is a modified reconstructed picture before deblocking filtering.

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0, palette\_mode\_flag[ xCb ][ yCb ] is equal to 0 and IntraSplitFlag is equal to 1), for the variable blkIdx proceeding over the values 0..3, the following ordered steps apply:

1. The variable xPb is set equal to xCb + ( nCbS  >>  1 ) \* ( blkIdx % 2 ).
2. The variable yPb is set equal to yCb + ( nCbS  >>  1 ) \* ( blkIdx / 2 ).
3. The derivation process for the intra prediction mode as specified in clause 8.4.2 is invoked with the luma location ( xPb, yPb ) as input.
4. If cu\_residual\_act\_flag[ xCb ][ yCb ] is equal to 1, the variable nSubCbS is set equal to ( nCbS  >> 1 ) and the residual modification process for residual blocks using adaptive colour transform as specified in clause 8.6.8 is invoked with the variable blkSize set equal to nSubCbS, the (nSubCbS)x(nSubCbS) array rY set equal to resSamplesL, the (nSubCbS)x(nSubCbS) array rCb set equal to resSamplesCb, and the (nSubCbS)x(nSubCbS) array rCr set equal to resSamplesCr as inputs, and the outputs are modified versions of the (nSubCbS)x(nSubCbS) arrays resSamplesL, resSamplesCb and resSamplesCr
5. The general decoding process for intra blocks as specified in clause 8.4.4.1 is invoked with the luma location ( xPb, yPb ), the variable log2TrafoSize set equal to log2CbSize − 1, the variable trafoDepth set equal to 1, the variable predModeIntra set equal to IntraPredModeY[ xPb ][ yPb ] and the variable cIdx set equal to 0, and variable controlParaACT set to ( cu\_residual\_act\_flag[ xCb ][ yCb ] ? 2 : 3 ) as inputs, and the output is a modified reconstructed picture before deblocking filtering.

When ChromaArrayType is not equal to 0, the following applies.

The variable log2CbSizeC is set equal to log2CbSize − ( ChromaArrayType  = =  3 ? 0 : 1 ).

Depending on the values of pcm\_flag[ xCb ][ yCb ] and IntraSplitFlag, the decoding process for chroma samples is specified as follows:

– If pcm\_flag[ xCb ][ yCb ] is equal to 1, the reconstructed picture is modified as follows:

SCb[ xCb / SubWidthC + i ][ yCb / SubHeightC + j ] =  
 pcm\_sample\_chroma[ ( nCbS / SubWidthC \* j ) + i ]  <<  ( BitDepthC − PcmBitDepthC ),  
 with i = 0..nCbS / SubWidthC − 1 and j = 0..nCbS / SubHeightC − 1 (8‑15)

SCr[ xCb / SubWidthC + i ][ yCb / SubHeightC + j ] =  
 pcm\_sample\_chroma[ ( nCbS / SubWidthC \* ( j + nCbS / SubHeightC ) ) + i ]  <<  
 ( BitDepthC − PcmBitDepthC ),  
 with i = 0..nCbS / SubWidthC − 1 and j = 0..nCbS / SubHeightC − 1 (8‑16)

– Otherwise (pcm\_flag[ xCb ][ yCb ] is equal to 0), if palette\_mode\_flag[ xCb ][ yCb ] is equal to 1 the following orderd steps apply:

1. The decoding process for palette intra blocks as specified in clause 8.4.4.2.7 is invoked with the chroma location ( xCb, yCb ), the variable cIdx set equal to 1, nCbSX set equal to nCbS / SubWidthC, and nCbSY set equal to nCbS / SubHeightC as inputs, and the output is an ( nCbS / SubWidthC ) x (nCbS / SubHeightC ) array of reconstructed palette sample values, recSamples[ x ][ y ], x = 0 … nCbS / SubWidthC − 1, y = 0..nCbS / SubHeightC − 1.
2. The reconstructed picture is modified as follows:

– If palette\_transpose\_flag[ xCb ][ yCb ] is true, SCb[ yCb + y ][ xCb + x ] is set equal to recSamples[ x ][ y ],

– Otherwise (palette\_transpose\_flag[ xCb ][ yCb ] is false), SCb[ xCb + x ][ yCb + y ] is set equal to recSamples[ x ][ y ].

1. The decoding process for palette intra blocks as specified in clause 8.4.4.2.7 is invoked with the chroma location ( xCb, yCb ), the variable cIdx set equal to 2, nCbSX set equal to nCbS / SubWidthC, and nCbSY set equal to nCbS / SubHeightC as inputs, and the output is an ( nCbS / SubWidthC ) x (nCbS / SubHeightC ) array of reconstructed palette sample values, recSamples[ x ][ y ], x = 0 … nCbS / SubWidthC − 1, y = 0..nCbS / SubHeightC − 1.
2. The reconstructed picture is modified as follows:

– If palette\_transpose\_flag[ xCb ][ yCb ] is true, SCr[ yCb + y ][ xCb + x ] is set equal to recSamples[ x ][ y ]

– Otherwise (palette\_transpose\_flag[ xCb ][ yCb ] is false), SCr[ xCb + x ][ yCb + y ] is set equal to recSamples[ x ][ y ]

##### Decoding process for palette mode

Inputs to this process are:

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

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

– two variables nCbSX and nCbSY specifying the width and height of the current block, respectively,

Output of this process is an array recSamples[ x ][ y ], with x = 0..nCbSX − 1,  y = 0..nCbSY − 1 specifying reconstructed sample values for the block.

Depending on the values of palette\_transpose\_flag[ xCb ][ yCb ], a variable bNeighbour is derived as follows:

1. The neighbouring locations ( xNbA, yNbA ) and ( xNbB, yNbB ) are set equal to ( xCb − 1, yCb ) and ( xCb, yCb − 1 ), respectively.
2. For X being replaced by either A or B,

* The availability derivation process for a block in z-scan order as specified in subclause  is invoked with the location ( xCurr, yCurr ) set equal to ( xCb, yCb ) and the neighbouring location ( xNbY, yNbY ) set equal to ( xNbX, yNbX ) as inputs, and the output is assigned to bNeighbourX. If palette\_transpose\_flag[ xCb ][ yCb ] is true, bNeighbourB is assigned to bNeighbour, otherwise bNeighbourA is assigned to bNeighbour.

Depending on the values of palette\_transpose\_flag[ xCb ][ yCb ], neighbouring samples array p[ x ] is derived as follows:

– If palette\_transpose\_flag[ xCb ][ yCb ] is true, each sample at the location ( xCb − 1, yCb + n ) is assigned to p[ n ], with n  = 0.. nCbS − 1.

– Otherwise (palette\_transpose\_flag[ xCb ][ yCb ] is false), each sample at the location ( xCb + n, yCb − 1) is assigned to p[ n ], with n  = 0.. nCbS − 1.

Depending on the value of cIdx, the variables nSubWidth and nSubHeight are derived as follows:

– If cIdx is equal to 0, nSubWidth is set to 1 and nSubHeight is set to 1,

– Otherwise, nSubWidth is set to SubWidthC and nSubHeight is set to SubHeightC.

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

For x = 0..nCbSX − 1,  y = 0..nCbSY − 1, recSamples[ x ][ y ] is set as follows:

The variables xLand yL are derived as follows:

xL= x \* nSubWidth (8‑65)

yL= y \* nSubHeight (8‑66)

The variable bIsEscapeSample is derived as follows:

– If PaletteIndexMap[ xCb + xL ][ yCb + yL ] is equal to MaxPaletteIndex and palette\_escape\_val\_present\_flag is equal to 1, bIsEscapeSample is set equal to 1:

– Otherwise, bIsEscapeSample is set equal to 0.

– If bIsEscapeSample is equal to 0, the following applies:

– If paletteIndexMap[ xCb + x ][ yCb + y ] is not equal to −1, the following applies:

recSamples[ x ][ y ] = CurrentPaletteEntries[ cIdx ][ PaletteIndexMap[ xCb + xL][ yCb + yL ] ],

– Otherwise, the following applies:

recSample[ cIdx ][ xCb + x ][ yCb + y ] = (y !=0) ? recSample[ cIdx ][ xCb + x ][ yCb + y − 1] : (bNeighbour ? p[ x ] : currentPaletteEntries[ cIdx ][0]) ,

– Otherwise, if cu\_transquant\_bypass\_flag is true, the following applies:

recSamples[ x ][ y ] = PaletteEscapeVal[ cIdx ][ xCb / nSubWidth + x ][ yCb / nSubHeight + y ],

– Otherwise (bIsEscapeSample is equal to 1 and cu\_transquant\_bypass\_flag is false), the following ordered steps apply:

1. The derivation process for quantization parameters as specified in clause 8.6.1 is invoked with the location ( xCb, yCb ) specifying the top-left sample of the current block relative to the top-left sample of the current picture.
2. The quantization parameter qP is derived as follows:

– If cIdx is equal to 0,

qP = max( 0, Qp′Y ) (‑)

– Otherwise, if cIdx is equal to 1,

qP = max( 0, Qp′Cb ) (‑)

– Otherwise (cIdx is equal to 2),

qP = max(0, Qp′Cr ) (‑)

1. The variables bitDepth is derived as follows:

bitDepth = ( cIdx = = 0 ) ? BitDepthY : BitDepthC  (‑)

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

tmpVal = ( PaletteEscapeVal[ cIdx ][ xCb / nSubWidth + x ][ yCb / nSubHeight + y ] \*

levelScale[ qP%6 ] ) << ( qP / 6 ) + 32 ) >> 6 (‑)

recSamples[ x ][ y ] = Clip3( 0, (1<< bitDepth) − 1, tmpVal ) (‑)

The variable numComps is derived as follows:

numComps = ( ChromaArrayType = = 0 ) ? 1 : 3 (‑)

The variables PredictorPaletteSize and PredictorPaletteEntries are modified as follows:

for( i = 0; i < CurrentPaletteSize; i++ )   
 for( cIdx = 0; cIdx < numComps; cIdx++ )   
 newPredictorPaletteEntries[ cIdx ][ i ] = CurrentPaletteEntries[ cIdx ][ i ]  
newPredictorPaletteSize = CurrentPaletteSize   
for( i = 0; i < PredictorPaletteSize && newPredictorPaletteSize < PaletteMaxPredictorSize; i++ )  
 if( PalettePredictorEntryReuseFlag[ i ] **!** **=** 1 ) {  
 for( cIdx = 0; cIdx < numComps; cIdx++ ) (‑)  
 newPredictorPaletteEntries[ cIdx ][ newPredictorPaletteSize ] = PredictorPaletteEntries[ cIdx ][ i ]  
 newPredictorPaletteSize ++   
 }  
for( cIdx = 0; cIdx < numComps; cIdx++ )   
 for( i = 0; i < newPredictorPaletteSize; i++ )   
 PredictorPaletteEntries[ cIdx ][ i ] = newPredictorPaletteEntries[ cIdx ][ i ]  
PredictorPaletteSize = newPredictorPaletteSize