### Deblocking filter process

A conditional filtering process shall be performed on a treeblock basis after the completion of the picture construction process prior to deblocking filter process for the entire decoded picture (as specified in subclauses XXX and YYY) [Ed.: (WJ) those subclauses seem not defined yet], with all treeblocks in a picture processed in order of increasing treeblock addresses.

Each treeblock is processed on a coding unit basis with the same order as decoding process. For each coding unit, vertical edges are filtered first, starting with the edge on the left-hand side of the coding unit proceeding through the edges towards the right-hand side of the coding unit in their geometrical order. Then, the horizontal edges are filtered starting with the edge on the top of the coding unit proceeding through the edges towards the bottom of the coding unit in their geometrical order.

Sample values above of the current coding unit that may have already been modified by the filtering of horizontal edges of deblocking filter process operation on previous coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit shall be used as inputs to the deblocking filter process on the current coding unit and may be further modified during the filtering of the current coding unit. Sample values to the left of the current coding unit may be modified by the filtering of vertical edge and may be further modified by the filtering of horizontal edges.

Sample values modified during filtering of vertical edges are used as input for the filtering of the horizontal edges. For sample values modified by both filtering of horizontal edges and filtering of vertical edges, filtering of horizontal edges is applied after filtering of vertical edges.

The deblocking filter process shall be applied to all prediction unit edges and transform unit edges of a picture, except edges at the boundary of the picture, any edges for which the deblocking filter process is disabled by disable\_deblocking\_filter\_idc and any edges coinside with slice boundaries when loop\_filter\_across\_slice\_flag is equal to 0. For the transform units and prediction units with edges smaller than 8 samples in either vertical or horizontal direction, only the edges lying on the 8x8 sample grid are filtered.

When disable\_deblocking\_filter\_idc is not equal to 1, the deblocking filter process is invoked as the following ordered steps for each coding unit with the same order as decoding process.

1. The coding unit size nS is set equal to 1 << log2CUSize.
2. The variables FilterInternalEdgesFlag, FilterLeftCuEdgeFlag and FilterTopCuEdgeFlag are derived as follows.

* The variable FilterInternalEdges is set equal to 1.
* If the left boundary of current coding unit is the left boundary of the picture or if the left boundary of current coding unit is the left boundary of the slice and loop\_filter\_across\_slice\_flag is equal to 0, the variable FilterLeftCuEdgeFlag is set equal to 0, otherwise set equal to 1.
* If the top boundary of current coding unit is the top boundary of the picture or if the top boundary of current coding unit is the top boundary of the slice and loop\_filter\_across\_slice\_flag is equal to 0, the variable FilterTopCuEdgeFlag is set equal to 0, otherwise set equal to 1.

1. All elements of two-dimensional array of size (nS)x(nS), horEdgeFlags and verEdgeFlags are initialized to zero.
2. The derivation process of transform unit boundary specified in subclause are invoked with the luma location ( xB, yB ) set equal to ( 0, 0 ), the transform unit width log2TrafoWidth set equal to log2CUSize, the transform unit height log2TrafoHeight set equal to log2CUSize and the variable trafoDepth set equal to 0 as the inputs and the modified horEdgeFlags and verEdgeFlags as outputs.
3. The derivation process of prediction unit boundary specified in subclause are invoked with the coding unit size log2CUSize and the prediction partition mode PartMode as inputs, and the modified horEdgeFlags and verEdgeFlags as outputs.
4. The derivation process of the boundary filtering strength specified in subclause 8.6.1.3 is invoked with the luma location ( xC, yC ), the coding unit size log2CUSize, horEdgeFlags and verEdgeFlags as inputs and an array of size (2)x(nS)x(nS), bS as output.
5. The filtering process for coding unit specified in subclause 8.6.1.4 are invoked with the luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture, the coding unit size log2CUSize and the array bS as inputs and the modified reconstructued picture as output.

#### Derivation process of transform unit boundary

Inputs of this process are:

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top‑left luma sample of the current coding unit,

– a variable log2TrafoWidth specifying the width of the current block,

– a variable log2TrafoHeight specifying the height of the current block,

– a variable trafoDepth.

Outputs of this process are:

– two-dimensional arrays of (nS)x(nS), horEdgeFlags and verEdgeFlags.

Depending on split\_transform\_flag[ xB ][ yB ][ trafoDepth ], the following applies:

– If split\_transform\_flag[ xB ][ yB ][ trafoDepth ] is equal to 1, the following ordered steps apply:

1. The variables xB1, yB1, xB2, yB2, xB3 and yB3 are derived as follows.
   * If InterTUSplitDirection is equal to 2, the following applies.
   * The variable xB1 is set equal to xB + ( ( 1 << log2TrafoWidth ) >> 1 ).
   * The variable yB1 is set equal to yB.
   * The variable xB2 is set equal to xB.
   * The variable yB1 is set equal to yB + ( ( 1 << log2TrafoHeight ) >> 1 ).
   * The variable xB3 is set equal to xB1.
   * The variable yB3 is set equal to yB2.
   * The variable log2TrafoWidth1 is set equal to log2TrafoWidth − 1.
   * The variable log2TrafoHeight1 is set equal to log2TrafoHeight − 1.
   * Otherwise (InterTUSplitDirection is equal to 0 or 1), the following applies.
   * The variable xB1 is set equal to xB + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB1 is set equal to yB + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable xB2 is set equal to xB1 + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB2 is set equal to yB1 + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable xB3 is set equal to xB2 + ((1 << (log2TrafoWidth)) >> 2) \* InterTUSplitDirection.
   * The variable yB3 is set equal to yB2 + ((1 << (log2TrafoHeight)) >> 2) \* (1 − InterTUSplitDirection).
   * The variable log2TrafoWidth1 is set equal to (log2TrafoWidth − 2) \* InterTUSplitDirection.
   * The variable log2TrafoHeight1 is set equal to (log2TrafoHeight − 2) \* (1 − InterTUSplitDirection).
2. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB, yB ), the variable log2TrafoWidth set equal to log2TrafoWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
3. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB1, yB1 ), the variable log2TrafoWidth set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
4. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB2, yB2 ), the variable log2TrafoWidth set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.
5. The deriviation process of transform unit boundary as specified in this subclause is invoked with the luma location ( xB3, yB3 ), the variable log2TrafoWidth1 set equal to log2TrafoSizeWidth1, the variable log2TrafoHeight set equal to log2TrafoHeight1 and the variable trafoDepth1 set equal to trafoDepth + 1 as inputs and the outputs are the modified versions of two arrays, horEdgeFlags and verEdgeFlags.

– Otherwise (split\_transform\_flag[ xB ][ yB ][ trafoDepth ] is equal to 0), the following applies:

* If yB is equal to zero, horEdgeFlags[ xB + k ][ yB ] is set equal to FilterTopCuEdgeFlag, otherwise horEdgeFlags[ xB + k ][ yB ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoWidth ) – 1.
* If xB is equal to zero, verEdgeFlags[ xB ][ yB + k ] is set equal to FilterLeftCuEdgeFlag, otherwise verEdgeFlags[ xB ][ yB + k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2TrafoHeight ) – 1.

#### Derivation process of prediction unit boundary

Inputs of this process are:

– a variable log2CUSize specifying the coding unit size,

– a prediction partition mode PartMode.

Outputs of this process are:

– two-dimensional arrays of (nS)x(nS), horEdgeFlags and verEdgeFlags.

Depending on PartMode, the following applies:

– If PartMode is equal to PART\_2NxN or PART\_NxN, horEdgeFlags[ k ][ 1 << ( log2CUSize – 1 ) ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_Nx2N or PART\_NxN, verEdgeFlags[ 1 << ( log2CUSize – 1 ) ][ k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_2NxnU, horEdgeFlags[ k ][ (1 << ( log2CUSize – 1 )) – (1 << ( log2CUSize – 2 )) ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_2NxnD, horEdgeFlags[ k ][ (1 << ( log2CUSize – 1 )) + (1 << ( log2CUSize – 2 )) ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_nLx2N, verEdgeFlags[ (1 << ( log2CUSize – 1 )) – (1 << ( log2CUSize – 2 )) ][ k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

– If PartMode is equal to PART\_nRx2N, verEdgeFlags[ (1 << ( log2CUSize – 1 )) + (1 << ( log2CUSize – 2 )) ][ k ] is set equal to FilterInternalEdgesFlag for k = 0.. ( 1 << log2CUSize ) – 1.

#### Derivation process of boundary filtering strength

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top-left luma sample of the current picture,

– a variable log2CUSize specifying the size of the current coding unit,

– a two-dimensional arrays of size (nS)x(nS), horEdgeFlags and verEdgeFlags.

Output of this process are three arrays of size (2)x(nS)x(nS), bS, bSCb, bSCr specifying the boundary filtering strength for luma and two chroma components respectively.

Let ( xEk, yEj ) with k = 0..nE-1 and j = 0..nE-1 specify a set of edge sample locations where nE is set equal to ( ( 1 << log2CUSize ) >> 2 ), xE0 = 0, yE0 = 0, xEk+1 = xEk + 4 and yEj+1 = yEj + 4.

For ( xEk, yEj ) with k = 0..nE-1 and j = 0..nE-1, the following applies.

* If horEdgeFlags[ xEk ][ yEj ] is equal to 1,
* Set sample p0 = recPicture[ xC + xEk ][ yC + yEj – 1 ] and q0 = recPicture[ xC + xEk ][ yC + yEj ].
* The variable filterDir is set equal to 1.
* Otherwise, if verEdgeFlags[ xEk ][ yEj ] is equal to 1,
* Set sample p0 = recPicture[ xC + xEk – 1 ][ yC + yEj ] and q0 = recPicture[ xC + xEk ][ yC + yEj ].
* The variable filterDir is set equal to 0.
* Depending on the value of filterDir, the variable bS[ filterDir ][ xEk ][ yEj ] is derived as follows.
* If the block edge is also a coding unit edge and the following condition is true, the variable bS[ 0 ][ xEk ][ yEj ] is set equal to 4.
* The sample p0 or q0 is in a coding unit coded with intra prediction mode
* Otherwise, if the following condition is true, the variable bS[ filterDir ][ xEk ][ yEj ] is set equal to 3.
* The sample p0 or q0 is in a coding unit coded with intra prediction mode
* Otherwise, if the block edge is also a transform unit edge and the following condition is true, the variable bS[ filterDir ][ xEk ][ yEj ] is set equal to 2.
* The sample p0 or q0 is in a transform unit which contains non-zero transform coefficient level.
* Otherwise, if any of the following conditions are true, the variable bS[ filterDir ][ xEk ][ yEj ] is set equal to 1.
* The prediction unit containing sample p0 has different reference pictures or a different number of motion vectors with the prediction unit containing the sample q0.

NOTE – The determination of whether the reference pictures used for the two prediction are the same or different is based on which pictures are referenced, without regard to whether a prediction is formed using an index into list 0 or an index into list 1, and also without regard to whether or not the index position within a reference picture list is different or not.

* One motion vector is used to predict the prediction unit containing sample p0, one motion vector is used to predict the prediction unit containing sample q0, and the absolute difference between the horizontal or vertical component of the motion vector used is greater than or equal to 4 in units of quarter luma samples.
* [Ed.: (WJ) needs to be checked again whether this condition covers all 2-motion cases] Two motion vectors are used to predict the prediction unit containing sample p0, two motion vectors are used to predict the prediction unit containing sample q0, and at least one of the motion vector pairs corresponding the same reference pictures and the different boundary samples p0 and q0 satisfies the following condition:

1. The absolute difference between the horizontal or vertical component of a motion vector used in the prediction of the two prediction units is greater than or equal to 4 in units of quarter luma samples.

* Otherwise, the variable bS[ filterDir ][ xEk ][ yEj ] is set equal to 0.
* Depending on the value of filterDir, the variables bSCb[ filterDir ][ xEk ][ yEj ] and bSCr[ filterDir ][ xEk ][ yEj ] are derived as follows.
* If the block edge is also a coding unit edge and the following condition is true, the variables bSCb[ 0 ][ xEk ][ yEj ] and bSCr[ filterDir ][ xEk ][ yEj ] are set equal to 4.
* The sample p0 or q0 is in a coding unit coded with intra prediction mode
* Otherwise, if the following condition is true, the variables bSCb[ 0 ][ xEk ][ yEj ] and bSCr[ filterDir ][ xEk ][ yEj ] are set equal to 3.
* The sample p0 or q0 is in a coding unit coded with intra prediction mode
* Otherwise,

- If the block edge is also a transform unit edge and the following condition is true, the variable bSCb[ filterDir ][ xEk ][ yEj ] is set equal to 2,

-The sample p0 or q0 is in a transform unit which contains non-zero chroma component Cb transform coefficient level.

-Otherwise, the variable bS[ filterDir ][ xEk ][ yEj ] is set equal to 0.

- If the block edge is also a transform unit edge and the following condition is true, the variable bSCr[ filterDir ][ xEk ][ yEj ] is set equal to 2.

-The sample p0 or q0 is in a transform unit which contains non-zero chroma component Cr transform coefficient level.

-Otherwise, the variable bS[ filterDir ][ xEk ][ yEj ] is set equal to 0.

#### Filtering process for coding unit

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a variable log2CUSize specifying the coding unit size,

– three arrays bS, bSCb and bSCr specifying the boundary filtering strength for luma and two chroma components respectively.

Output of this process is:

– modified reconstruction of the picture.

The filtering process for luma edges in the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << ( log2CUSize – 3 ).
2. All elements of the three-dimensional array of size (2)x(nD)x(nD), dEdge are initialized to zero.
3. All elements of the three-dimensional array of size (2)x(nD)x(1<<log2CUSize), dSample are initialized to zero.
4. All elements of the three-dimensional array of size (2)x(nD)x(nD), bStrength are initialized to zero.
5. For xDk set equal to xC+( k << 3 ), k=0..nD – 1, the following applies:

* For yDm set equal to yC+( m << 3 ), m=0..nD – 1, the following ordered steps apply:
  + 1. Boundary filtering strength bSVer is derived as follows:

bSVer = Max( bS[ 0 ][ xDk ][ yDm + i ] ) for i = 0..7 (8‑428)

* + 1. bStrength[1][k][m] is set equal to bSVer.
    2. Decision process for luma block edge in subclause 8.6.1.4.1 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 1, and the boundary filtering strength bSVer as inputs and the decision dEdge[1][k][m] and an array dS of size (8) as outputs.
    3. dSample[1][k][(m<<3)+i] is set equal to dS[i] for i=0..7.
    4. Boundary filtering strength bSHor is derived as follows:

bSHor = Max( bS[ 1 ][ xDk + i ][ yDm ] ) for i = 0..7 (8‑429)

* + 1. bStrength[0][k][m] is set equal to bSHor.
    2. Decision process for luma block edge in subclause 8.6.1.4.1 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 0, the boundary filtering strength bSHor as inputs, the decision dEdge[0][k][m] and an array dS of size (8) as outputs.
    3. dSample[0][m][(k<<3)+i] is set equal to dS[i] for i=0..7.

1. For xDk set equal to xC+( k << 3 ), k=0..nD - 1, the following applies:

* For yDm set equal to yB+( m << 3 ), m=0..nD – 1, the following ordered steps apply:
  + 1. dS[i] is set equal to dSample[1][k][(m<<3)+i] for i=0..7.
    2. Filtering process for luma block edge in subclause 8.6.1.4.2 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 1, the boundary filtering strength bStrength[1][k][m], the decision dEdge[1][k][m], and the array of size (8), dS as inputs and the modified luma picture buffer as outputs.

1. For yDm set equal to yC+( m << 3 ), m=0..nD - 1, the following applies:

* For xDk set equal to xC+( k << 3 ), k=0..nD – 1, the following ordered steps apply:
  + 1. If xDk is equal to 0, the parameter xPOS is set equal to 1. If xDk is equal to xB+( ( nD – 1) << 3 ) xPOS is set equal to 2. Otherwise xPOS is set to 0.
    2. dS[i] is set equal to dSample[0][m][ (k << 3) + i ] for i = 0..7.
    3. Filtering process for luma block edge in subclause 8.6.1.4.2 is invoked with the luma location of the coding unit ( xC, yC ), the luma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 0, the boundary filtering strength bStrength[0][k][m], the decision dEdge[0][k][m], and the array of size (8), dS, xPOS, dSL[m][], dEL[m], bSL[m], and tCL[m], as inputs and the modified luma picture buffer as output.
* The elements of the two dimensional array of size (3)x(nD), dSL are set as follows. dSL[m][0], dSL[m][1], and dSL[m][2], are set equal to dS[5], dS[6] and dS[7].
* The elements of the array of size (nD), dEL are set as follows. dEL[m] is set equal to dEdge[0][k][m].
* The elements of the array of size (nD), bSL are set as follows. bSL[m] is set equal to bStrength[0][k][m].
* The elements of the array of size (nD), tC are set as follows. tCL[m] is set equal to tc.

The filtering process for chroma edges in the current coding unit consists of the following ordered steps:

1. The variable nD is set equal to 1 << ( Max( log2CUSize, 4 ) – 4 ).
2. For xDk set equal to ( xC / 2 )+( k << 3 ), k=0..nD – 1, the following applies:

* For yDm set equal to ( yC / 2)+( m << 2 ), m=0..nD\*2 – 1, the following ordered steps apply:

1. Boundary filtering strength bSCbVer1 and bSCbVer2 are derived as follows:

bSCbVer1 = Max( bSCb[ 0 ][ xDk\*2 ][ yDm\*2 + i ] ) for i = 0..3 (8‑430)

bSCbVer2 = Max( bSCb[ 0 ][ xDk\*2 ][ yDm\*2 + i ] ) for i = 4..7 (8‑431)

Boundary filtering strength bSCrVer1 and bSCrVer2 are derived as follows: bSCrVer1 = Max( bSCr[ 0 ][ xDk\*2 ][ yDm\*2 + i ] ) for i = 0..3 (8‑432)

bSCrVer2 = Max( bSCr[ 0 ][ xDk\*2 ][ yDm\*2 + i ] ) for i = 4..7 (8‑433)

1. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 1, a chroma component index cIdx set equal to 1 and the boundary filtering strength bSCbVer1 and bSCbVer2 as inputs and the modified chroma picture buffer as output.
2. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 1, a chroma component index cIdx set equal to 2 and the boundary filtering strength bSCrVer1 and bSCrVer2 as inputs and the modified chroma picture buffer as output.
3. For yDm set equal to ( yC / 2 )+( m << 3 ), m=0..nD – 1, the following applies:

* For xDk set equal to ( xC / 2 )+( k << 2 ), k=0..nD\*2 – 1, the following ordered steps apply:

1. If xDk is equal to 0, the parameter xPOS is set equal to 1. If xDk is equal to xB+( ( nD\*2 – 1) << 2 ) xPOS is set equal to 2. Otherwise xPOS is set to 0.
2. Boundary filtering strength bSCbHor1 and bSCbHor2 are derived as follows:

bSCbHor1 = Max( bSCb[ 1 ][ xDk\*2 + i ][ yDm\*2 ] ) for i = 0..3 (8‑434)

bSCbHor2 = Max( bSCb[ 1 ][ xDk\*2 + i ][ yDm\*2 ] ) for i = 4..7 (8‑435)

Boundary filtering strength bSCrHor1 and bSCrHor2 is derived as follows:

bSCrHor1 = Max( bSCr[ 1 ][ xDk\*2 + i][ yDm\*2 ] ) for i = 0..3 (8‑436)

bSCrHor2 = Max( bSCr[ 1 ][ xDk\*2 + i][ yDm\*2 ] ) for i = 4..7 (8‑437)

1. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 0, a chroma component index cIdx set equal to 1 and the boundary filtering strength bSCbHor1 and bSCbHor2, xPOS, bSL[m] and tCL[m] as inputs and the modified chroma picture buffer as output.
2. Filtering process for chroma block edge in subclause 8.6.1.4.3 is invoked with the chroma location of the coding unit ( xC/2, yC/2 ), the chroma location of the block ( xDk, yDm ), a variable verticalEdgeFlag set equal to 0, a chroma component index cIdx set equal to 2 and the boundary filtering strength bSCrHor1 and bSCrHor2, xPOS, bSL[m] and tCL[m] as inputs and the modified chroma picture buffer as output.

##### Decision process for luma block edge

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top left luma sample of the current coding unit,

– a variable verticalEdgeFlag,

– a variable bS specifying the boundary filtering strength,

Output of this process is:

– variables dE, dEp1 and dEq1 containing decisions,

– one-dimensional array of size (8), dS containing decisions.

Let s’ represent the luma sample array recPictureL of the current picture.

A variables β is specified as Table 8‑15 with luma quantization parameter qPL as input.

A variable tC is specified as follows:

– If bS is greater than 2, the variable tC is specified as Table 8‑15 with luma quantization parameter Clip3(0, 55, qPL + 2 ) as input,

– Otherwise (bS is equal or less than 2), the variable tC is specified as Table 8‑15 with luma quantization parameter qPL as input.

Depending on verticalEdgeFlag, the following applies:

– If verticalEdgeFlag is equal to 1, the following ordered steps apply:

1. The sample values pi,k and qi,k with i = 0..3 and k = 2,5 are derived as follows:

qi,k = s’[ xC + xB +i, yC + yB + k ] (8‑438)

pi,k = s’[ xC + xB – i – 1, yC + yB + k ] (8‑439)

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

dp = | p2,2 – 2\*p1,2 + p0,2 | + | p2,5 – 2\*p1,5 + p0,5 | (8‑440)

dq = | q2,2 – 2\*q1,2 + q0,2 | + | q2,5 – 2\*q1,5 + q0,5 | (8‑441)

d = dp + dq (8‑441)

1. The variables dE, dEp1 and dEq1 are set equal to 0.
2. If bS is not equal to 0 and d is less than β, the following ordered steps apply:
3. for each sample location ( xC + xB, yC + yB + k ), k = 0..7, the following ordered steps apply:
   1. The decision process for a luma sample specified in subclause 8.6.1.4.4 is invoked with sample values pi,k, qi,k with i = 0..3, the boundary filtering strength bS and the variables d, β and tC as inputs and a decision dSam as output.
   2. The variable dS[k] is set equal to dSam
4. The variable dE is set equal to 1.
5. If dp is less than ( β + ( β >> 1 ) ) >> 3, the variable dEp1 is set equal to 1.
6. If dq is less than ( β + ( β >> 1 ) ) >> 3, the variable dEq1 is set equal to 1.

– Otherwise (verticalEdgeFlag is equal to 0), the following ordered steps apply:

1. The sample values pi,k and qi,k with i = 0..3 and k = 2,5 are derived as follows:

qi,k = s’[ xC + xB +k, yC + yB + i ] (8‑442)

pi,k = s’[ xC + xB +k, yC + yB – i – 1 ] (8‑443)

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

dp = | p2,2 – 2\*p1,2 + p0,2 | + | p2,5 – 2\*p1,5 + p0,5 | (8‑444)

dq = | q2,2 – 2\*q1,2 + q0,2 | + | q2,5 – 2\*q1,5 + q0,5 | (8‑445)

d = dp + dq (8‑446)

1. The variables dE, dEp1 and dEq1 are set equal to 0.
2. If bS is not equal to 0 and d is less than β, the following ordered steps apply:
3. For each sample location ( xC + xB + k, yC + yB ), k = 0..7, the following ordered steps apply:
   1. The decision process for a luma sample specified in subclause 8.6.1.4.4 is invoked with sample values pi,k, qi,k with i = 0..3, the boundary filtering strength bS and the variables d, β and tC as inputs and a decision dSam as output.
   2. The variable dS[k] is set equal to dSam.
4. The variable dE is set equal to 1.
5. If dp is less than ( β + ( β >> 1 ) ) >> 3, the variable dEp1 is set equal to 1.
6. If dq is less than ( β + ( β >> 1 ) ) >> 3, the variable dEq1 is set equal to 1.

##### Filtering process for luma block edge

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left luma sample of the current coding unit relative to the top left luma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left luma sample of the current block relative to the top left luma sample of the current coding unit,

– a variable verticalEdgeFlag,

– a variable bS specifying the boundary filtering strength,

– variables dE, dEp1 and dEq1 containing decisions,

– one-dimensional array of size (8), dS containing decisions,

– a variable bSL,

– a variable tCL,

Output of this process is:

– modified reconstruction of the picture.

Let s’ represent the luma sample array recPictureL of the current picture.

Depending on pcm\_flag, a variable β is specified as follows:

– If pcm\_flag is equal to 1, the variables β is specified as Table 8‑11 with luma quantization parameter 0 as input.

– Otherwise, the variables β is specified as Table 8‑11 with luma quantization parameter qPL as input.

A variable tC is specified as follows:

– If bS is greater than 2, the variable tC is specified as Table 8‑11 with luma quantization parameter Clip3(0, 55, qPL + 2 ) as input,

– Otherwise (bS is equal or less than 2), the variable tC is specified as Table 8‑11 with luma quantization parameter qPL as input.

Depending on verticalEdgeFlag, the following applies:

– If verticalEdgeFlag is equal to 1, the following ordered steps apply:

1. The sample values pi,k and qi,k with i = 0..3 and k = 0..7 are derived as follows:

qi,k = s’[ xC + xB +i, yC + yB + k ] (8‑447)

pi,k = s’[ xC + xB – i – 1, yC + yB + k ] (8‑448)

1. If dE is not equal to 0, for each sample location ( xC + xB, yC + yB + k ), k = 0..7, the following ordered steps apply:
2. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values pi,k, qi,k with i = 0..3, the decision dS[k], variables dEp1 and dEq1, the boundary filtering strength bS and the variable tC as inputs and the number of filtered samples nDp and nDq from each side of the block boundary, and the filtered sample values pi’ and qj’ as outputs.
3. The filtered sample values pi’ and qj’ with i = 0..nDp – 1, j = 0..nDq – 1 replace the corresponding samples inside the sample array s’ as follows:

s’[ xC + xB +j, yC + yB + k ] = qj’ (8‑449)

s’[ xC + xB – i – 1, yC + yB + k ] = pi’ (8‑450)

– Otherwise (verticalEdgeFlag is equal to 0), the following ordered steps apply:

1. If xPOS is equal to 1, the parameters ks and ke are set to -3 and 4 respectively. If xD is equal to 2, the parameters ks and ke are set to 0 and 4 respectively. Otherwise ks and ke are set to 0 and 7 respectively.
2. The sample values pi,k and qi,k with i = 0..3 and k = ks..ke are derived as follows:

qi,k = s’[ xC + xB +k, yC + yB + i ] (8‑451)

pi,k = s’[ xC + xB +k, yC + yB – i – 1 ] (8‑452)

1. If xPOS is less than 0 and dEL is not equal to 0, for each sample location ( xC + xB + k, yC + yB ), k = -3..-1, the following ordered steps apply:
2. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values pi,k, qi,k with i = 0..3, decision dSL[k+3], variables dEp1 and dEq1, the boundary filtering strength bSL and the variable tCL as inputs and the number of filtered samples nDp and nDq from each side of the block boundary and the filtered sample values pi’ and qj’ as outputs.
3. The filtered sample values pi’ and qj’ with i = 0..nDp – 1, j = 0..nDq – 1 replace the corresponding samples inside the sample array s’ as follows:

s’[ xC + xB +k, yC + yB + j ] = qj’ (8‑453)

s’[ xC + xB +k, yC + yB – i – 1 ] = pi’ (8‑454)

1. If dE is not equal to 0, for each sample location ( xC + xB + k, yC + yB ), k = 0.. ke , the following ordered steps apply:
2. The filtering process for a luma sample specified in subclause 8.6.1.4.5 is invoked with sample values pi,k, qi,k with i = 0..3, decision dS[k], variables dEp1 and dEq1, the boundary filtering strength bS and the variable tC as inputs and the number of filtered samples nDp and nDq from each side of the block boundary and the filtered sample values pi’ and qj’ as outputs.
3. The filtered sample values pi’ and qj’ with i = 0..nDp – 1, j = 0..nDq – 1 replace the corresponding samples inside the sample array s’ as follows:

s’[ xC + xB +k, yC + yB + j ] = qj’ (8‑455)

s’[ xC + xB +k, yC + yB – i – 1 ] = pi’ (8‑456)

##### Filtering process for chroma block edge

[Ed.: (WJ) cIdx cannot be 0 here]

Inputs of this process are:

– a luma location ( xC, yC ) specifying the top-left chroma sample of the current coding unit relative to the top left chroma sample of the current picture,

– a luma location ( xB, yB ) specifying the top-left chroma sample of the current block relative to the top left chroma sample of the current coding unit,

– a variable verticalEdgeFlag,

– variables bS1 and bS2 specifying the boundary filtering strength,

– a variable cIdx specifying the chroma component index.

– a variable xPOS,

– two variables tC1 and tC2

Output of this process is:

– modified reconstruction of the picture.

Let s’ be a variable specifying chroma sample array which is derived as follows.

– If cIdx is equal to 1, s’ represents the chroma sample array recPictureCb of the current picture.

– Otherwise (cIdx is equal to 2), s’ represents the chroma sample array recPictureCr of the current picture.

A variable β is specified as Table 8‑15 with chroma quantization parameter qPC as input

The variable tC1 is specified as follows:

– If bS1 is greater than 2, the variable tC1 is specified as Table 8‑15 with chroma quantization parameter Clip3(0, 55, qPC + 2 ) as input,

– Otherwise (bS1 is equal or less than 2), the variable tC1 is specified as Table 8‑15 with chroma quantization parameter qPC as input.

The variable tC2 is specified as follows:

– If bS2 is greater than 2, the variable tC2 is specified as Table 8‑15 with chroma quantization parameter Clip3(0, 55, qPC + 2 ) as input,

– Otherwise (bS2 is equal or less than 2), the variable tC2 is specified as Table 8‑15 with chroma quantization parameter qPC as input.

Depending on verticalEdgeFlag, the following applies:

– If verticalEdgeFlag is equal to 1, for each sample location ( xC + xB, yC + yB + k ), k = 0..7, the following ordered steps apply:

1. The sample values pi and qi with i = 0..1 are derived as follows:

qi,k = s’[ xC + xB +i, yC + yB + k ] (8‑457)

pi,k = s’[ xC + xB – i – 1, yC + yB + k ] (8‑458)

1. Two variables d1 and d2 are derived as follows:

d1 = |p0,1 – p1,1| + |q0,1 – q1,1| + |p0,2 – p1,2| + |q0,2 – q1,2|

d2 = |p0,5 – p1,5| + |q0,5 – q1,5| + |p0,6 – p1,6| + |q0,6 – q1,6|

1. If bS1 is greater than 0, and d1 is less than β, for each sample location ( xC + xB, yC + yB + k ), k = 0..3, the following ordered steps apply:
2. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values pi,k, qi,k, with i = 0..1, the boundary filtering strength bS1 and the variable tC1 as inputs and the filtered sample values p0,k’ and q0,k’ as outputs.
3. The filtered sample values p0,k’ and q0,k’ replace the corresponding samples inside the sample array s’ as follows:

s’[ xC + xB , yC + yB + k ] = q0,k’ (8‑459)

s’[ xC + xB – 1, yC + yB + k ] = p0,k’ (8‑460)

1. If bS2 is greater than 0, and d2 is less than β, for each sample location ( xC + xB, yC + yB + k ), k = 4..7, the following ordered steps apply:
2. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values pi,k, qi,k, with i = 0..1, the boundary filtering strength bS2 and the variable tC2 as inputs and the filtered sample values p0,k’ and q0,k’ as outputs.
3. The filtered sample values p0,k’ and q0,k’ replace the corresponding samples inside the sample array s’ as follows:

s’[ xC + xB , yC + yB + k ] = q0,k’ (8‑461)

s’[ xC + xB – 1, yC + yB + k ] = p0,k’ (8‑462)

Otherwise (verticalEdgeFlag is equal to 0), the following ordered steps apply:

1. If xPOS is equal to 1, the parameters ks and ke are set to -1 and 2 respectively. If xD is equal to 2, the parameters ks and ke are set to 0 and 2 respectively. Otherwise ks and ke are set to 0 and 3 respectively.
2. The sample values pi and qi with i = 0..1 and k = ks..ke are derived as follows:

qi = s’[ xC + xB +k, yC + yB + i ] (8‑463)

pi = s’[ xC + xB +k, yC + yB – i – 1 ] (8‑464)

1. Two variables d1 and d2 are derived as follows:

d1 = |p0,1 – p1,1| + |q0,1 – q1,1| + |p0,2 – p1,2| + |q0,2 – q1,2|

d2 = |p0,5 – p1,5| + |q0,5 – q1,5| + |p0,6 – p1,6| + |q0,6 – q1,6|

1. If xPOS is less than 0, and if bS1 is greater than 0, and d1 is less than β, for each sample location ( xC + xB - 1, yC + yB ), the following ordered steps apply:
2. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values pi, qi, with i = 0..1, the boundary filtering strength bS1 and the variable tC1 as inputs and the filtered sample values p0’ and q0’ as outputs.
3. The filtered sample values p0’ and q0’ replace the corresponding samples inside the sample array s’ as follows:

s’[ xC + xB +k, yC + yB ] = q0’ (8‑465)

s’[ xC + xB +k, yC + yB – i – 1 ] = pi’ (8‑466)

1. If bS1 is greater than 0 and d1 is less than β, for each sample location ( xC + xB + k, yC + yB ), k = 0.. ke, the following ordered steps apply:
2. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values pi, qi, with i = 0..1, the boundary filtering strength bS1 and the variable tC1 as inputs and the filtered sample values p0’ and q0’ as outputs.
3. The filtered sample values p0’ and q0’ replace the corresponding samples inside the sample array s’ as follows:

s’[ xC + xB +k, yC + yB ] = q0’ (8‑467)

s’[ xC + xB +k, yC + yB – 1 ] = p0’ (8‑468)

1. If bS2 is greater than 0 and d2 is less than β, for each sample location ( xC + xB + k, yC + yB ), k = 4.. 4+ke, the following ordered steps apply:
2. The filtering process for a sample specified in subclause 8.6.1.4.6 is invoked with sample values pi, qi, with i = 0..1, the boundary filtering strength bS2 and the variable tC2 as inputs and the filtered sample values p0’ and q0’ as outputs.
3. The filtered sample values p0’ and q0’ replace the corresponding samples inside the sample array s’ as follows:

s’[ xC + xB +k, yC + yB ] = q0’ (8‑469)

s’[ xC + xB +k, yC + yB – 1 ] = p0’ (8‑470)

##### Decision process for a luma sample

[Ed: (WJ) no filtering when bS is equal to 0]

Inputs of this process are:

– sample values, pi and qi with i = 0..2,

– a variable bS specifying the boundary filtering strength,

– variables d, β and tC.

Output of this process is:

– a variable dSam containing a decision

When the variable bS is not equal to 0, the following applies:

– If d is less than ( β >> 2 ), | p3 – p0 | + | q0 – q3 | is less than ( β >> 3 ) and | p0 – q0 | is less than ( 5\*tC + 1 ) >> 1, dSam is set equal to 1.

– Otherwise, dSam is set equal to 0.

##### Filtering process for a luma sample

Inputs of this process are:

– sample values, pi and qi with i = 0..3,

– a variable dSam containing a decision,

– variables dEp1 and dEq1 containing decisions to filter pixels p1 and q1 respectively,

– a variable tC.

Output of this process is:

– number of filtered samples nDp and nDq,

– filtered sample values, pi’ and qj’ with i = 0..nDp – 1, j = 0..nDq – 1

Depending on dSam, the following applies:

– When the variable dSam is equal to 1, the following strong filtering applies while nDp and nDq are set equal to 3:

p0’ = Clip1Y( ( p2 + 2\*p1 + 2\*p0 + 2\*q0 + q1 + 4 ) >> 3 ) (8‑450)

p1’ = Clip1Y( ( p2 + p1 + p0 + q0 + 2 ) >> 2 ) (8‑451)

p2’ = Clip1Y( ( 2\*p3 + 3\*p2 + p1 + p0 + q0 + 4 ) >> 3 ) (8‑452)

q0’ = Clip1Y( ( p1 + 2\*p0 + 2\*q0 + 2\*q1 + q2 + 4 ) >> 3 ) (8‑453)

q1’ = Clip1Y( ( p0 + q0 + q1 + q2 + 2 ) >> 2 ) (8‑454)

q2’ = Clip1Y( ( p0 + q0 + q1 + 3\*q2 + 2\*q3 + 4 ) >> 3 ) (8‑455)

– Otherwise, the following weak filtering applies while nDp and nDq are set equal to 0:

Δ = ( 9 \* ( q0 –  p0 ) – 3 \* ( q1 – p1 ) + 8 ) >> 4 (8‑456)

* + When abs(Δ) is less than tc\*10, the following ordered steps apply:
    1. The filtered sample values p0’ and q0’ are specified as follows:

Δ = Clip3( -tc, tc, Δ ) (8‑456)

p0’ = Clip1Y( p0 + Δ ) (8‑457)

q0’ = Clip1Y( q0 - Δ ) (8‑458)

* + 1. If dEp1 is equal to 1, the filtered sample value pi’ is specified as follows:

Δp = Clip3( -(tc >> 1), tc >> 1, ( ( ( p2 + p0 + 1 ) >> 1 ) – p1 + Δ ) >>1 ) (8‑458)

pi’ = Clip1Y( p1 + Δp ) (8‑458)

* + 1. If dEq1 is equal to 1, the filtered sample value qi’ is specified as follows:

Δq = Clip3( -(tc >> 1), tc >> 1, ( ( ( q2 + q0 + 1 ) >> 1 ) – q1 – Δ ) >>1 ) (8‑458)

qi’ = Clip1Y( q1 + Δq ) (8‑458)

* + 1. nDp is set equal to dEp1+1 and nDq is set equal to dEq1+1.

Each of the filtered sample values, pi’ with i = 0..nDp-1, is substituted by the corresponding input sample value pi if all of the following conditions are true.

– pi is a sample of an I\_PCM block.

– pcm\_loop\_filter\_disable\_flag value is equal to 1.

Similarly, each of the filtered sample values, qj’ with j = 0..nDq-1, is substituted by the corresponding input sample value qj if all of the following conditions are true.

– qj is a sample of an I\_PCM block.

– pcm\_loop\_filter\_disable\_flag value is equal to 1.

[Ed. (WJ): for PCM case, deblocking filter applies first and the filtered pixels are restored. Rather than this, it’s better to skip the filtering itself for PCM samples since first filtering is actually not needed.]

Table 8‑15 – Derivation of threshold variables β and tC from input Q

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Q** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| **β** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 6 | 7 | 8 |
| **tC** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| **Q** | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 |
| **β** | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 |
| **tC** | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 4 |
| **Q** | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 |  |
| **β** | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 | 54 | 56 | 58 | 60 | 62 | 64 | 64 | 64 | 64 | 64 |  |
| **tC** | 5 | 5 | 6 | 6 | 7 | 8 | 9 | 9 | 10 | 10 | 11 | 11 | 12 | 12 | 13 | 13 | 14 | 14 |  |

##### Filtering process for a chroma sample

[Ed: (WJ) no filtering when bS is equal or less than 2]

Inputs of this process are:

– sample values, pi and qi with i = 0..1,

– a variable bS specifying the boundary filtering strength.

– a variable tC.

Output of this process is:

– The filtered sample values, p0’ and q0’.

When the variable bS is greater than 0, the filtered sample values p0’ and q0’ are derived by

Δ = Clip3( -tC, tC, ( ( ( ( q0 – p0 ) << 2 ) + p1 – q1 + 4 ) >> 3 ) ) (8‑461)

p0’ = Clip1C( p0 + Δ ) (8‑462)

q0’ = Clip1C( q0 - Δ ) (8‑463)

The filtered sample value, p0’ is substituted by the corresponding input sample value p0 if all of the following conditions are true.

– p0 is a sample of an I\_PCM block.

– pcm\_loop\_filter\_disable\_flag value is equal to 1.

Similary, the filtered sample value, q0’ is substituted by the corresponding input sample value q0 if all of the following conditions are true.

– q0 is a sample of an I\_PCM block.

– pcm\_loop\_filter\_disable\_flag value is equal to 1.

[Ed. (WJ): for PCM case, deblocking filter applies first and the filtered pixels are restored. Rather than this, it’s better to skip the filtering itself for PCM samples since first filtering is actually not needed.]