### Adaptive loop filter process

#### General

Inputs of this process are the reconstructed picture sample array prior to adaptive loop filter recPictureL and, when ChromaArrayType is not equal to 0, the arrays recPictureCb and recPictureCr.

Outputs of this process are the modified reconstructed picture sample array after adaptive loop filter alfPictureL and, when ChromaArrayType is not equal to 0, the arrays ccAlfPictureCb and ccAlfPictureCr.

The sample values in the modified reconstructed picture sample array after adaptive loop filter alfPictureL and, when ChromaArrayType is not equal to 0, the arrays alfPictureCb and alfPictureCr are initially set equal to the sample values in the reconstructed picture sample array prior to adaptive loop filter recPictureL and, when ChromaArrayType is not equal to 0, the arrays recPictureCb and recPictureCr, respectively.

The following ordered steps apply:

* + For every coding tree unit with luma coding tree block location ( rx, ry ), where rx = 0..PicWidthInCtbsY − 1 and ry = 0..PicHeightInCtbsY − 1, the following applies:
  + When alf\_ctb\_flag[ 0 ][ rx ][ ry ] is equal to 1, the coding tree block filtering process for luma samples as specified in clause 8.8.5.2 is invoked with recPictureL, alfPictureL, and the luma coding tree block location ( xCtb, yCtb ) set equal to ( rx  <<  CtbLog2SizeY, ry  <<  CtbLog2SizeY ) as inputs, and the output is the modified filtered picture alfPictureL.
  + When ChromaArrayType is not equal to 0 and alf\_ctb\_flag[ 1 ][ rx ][ ry ] is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 8.8.5.4 is invoked with recPicture set equal to recPictureCb, alfPicture set equal to alfPictureCb, the chroma coding tree block location ( xCtbC, yCtbC ) set equal to ( ( rx  <<  CtbLog2SizeY ) / SubWidthC, ( ry  <<  CtbLog2SizeY ) / SubHeightC ), and the alternative chroma filter index altIdx set equal to alf\_ctb\_filter\_alt\_idx[ 0 ][ rx ][ ry ] as inputs, and the output is the modified filtered picture alfPictureCb.
  + When ChromaArrayType is not equal to 0 and alf\_ctb\_flag[ 2 ][ rx ][ ry ] is equal to 1, the coding tree block filtering process for chroma samples as specified in clause 8.8.5.4 is invoked with recPicture set equal to recPictureCr, alfPicture set equal to alfPictureCr, the chroma coding tree block location ( xCtbC, yCtbC ) set equal to ( ( rx  <<  CtbLog2SizeY ) / SubWidthC, ( ry  <<  CtbLog2SizeY ) / SubHeightC ), and the alternative chroma filter index altIdx set equal to alf\_ctb\_filter\_alt\_idx[ 1 ][ rx ][ ry ] as inputs, and the output is the modified filtered picture alfPictureCr.
  + When ChromaArrayType is not equal to 0, the sample values in the arrays ccAlfPictureCb and ccAlfPictureCr are set equal to the sample values in the arrays alfPictureCb and alfPictureCr, respectively.
  + For every coding tree unit with luma coding tree block location ( rx, ry ), where rx = 0..PicWidthInCtbsY − 1 and ry = 0..PicHeightInCtbsY − 1, the following applies:
  + When ChromaArrayType is not equal to 0 and alf\_ctb\_cc\_cb\_idc[ rx ][ ry ] is not equal to 0, the cross-component filtering process as specified in clause 8.8.5.7 is invoked with recPictureL set equal to recPictureL, alfPictureC set equal to alfPictureCb, the chroma coding tree block location ( xCtbC, yCtbC ) set equal to ( ( rx << CtbLog2SizeY ) / SubWidthC, ( ry << CtbLog2SizeY  )/ SubHeightC ) ), the luma coding tree block location ( xCtb, yCtb ) set equal to ( rx  <<  CtbLog2SizeY, ry  <<  CtbLog2SizeY ), ccAlfWidth set equal to ( 1 << CtbLog2SizeY ) / SubWidthC, ccAlfHeight set equal to ( 1<< CtbLog2SizeY ) / SubHeightC, and the cross-component filter coefficients CcAlfCoeff[ j ] set equal to CcAlfApsCoeffCb[ slice\_cc\_alf\_cb\_aps\_id ][ alf\_ctb\_cc\_cb\_idc[ rx ][ ry ] − 1 ][ j ], with j = 0..6, as inputs, and the output is the modified filtered picture ccAlfPictureCb.
  + When ChromaArrayType is not equal to 0 and alf\_ctb\_cc\_cr\_idc[ rx ][ ry ] is not equal to 0, the cross-component filtering process as specified in clause 8.8.5.7 is invoked with recPictureL set equal to recPictureL, alfPictureC set equal to alfPictureCr, the chroma coding tree block location ( xCtbC, yCtbC ) set equal to ( ( rx << CtbLog2SizeY ) / SubWidthC, ( ry << CtbLog2SizeY  )/ SubHeightC ) ), ccAlfWidth set equal to ( 1 << CtbLog2SizeY ) / SubWidthC, ccAlfHeight set equal to ( 1<< CtbLog2SizeY ) / SubHeightC, and the cross-component filter coefficients CcAlfCoeff[ j ] set equal to CcAlfApsCoeffCr[ slice\_cc\_alf\_cr\_aps\_id ][ alf\_ctb\_cc\_cr\_idc[ rx ][ ry ] − 1 ][ j ], with j = 0..6, as inputs, and the output is the modified filtered picture ccAlfPictureCr.

#### Coding tree block filtering process for luma samples

Inputs of this process are:

* a reconstructed luma picture sample array recPicture prior to the adaptive loop filtering process,
* a filtered reconstructed luma picture sample array alfPictureL,
* a luma location ( xCtb, yCtb ) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture.

Output of this process is the modified filtered reconstructed luma picture sample array alfPictureL.

The derivation process for filter index clause 8.8.5.3 is invoked with the location ( xCtb, yCtb ) and the reconstructed luma picture sample array recPicture as inputs, and filtIdx[ x ][ y ] and transposeIdx[ x ][ y ] with x, y = 0..CtbSizeY − 1 as outputs.

For the derivation of the filtered reconstructed luma samples alfPictureL[ x ][ y ], each reconstructed luma sample inside the current luma coding tree block recPicture[ x ][ y ] is filtered as follows with x, y = 0..CtbSizeY − 1:

* + The array of luma filter coefficients f[ j ] and the array of luma clipping values c[ j ] corresponding to the filter specified by filtIdx[ x ][ y ] is derived as follows with j = 0..11:
  + If AlfCtbFiltSetIdxY[ xCtb  >>  CtbLog2SizeY ][ yCtb  >>  CtbLog2SizeY ] is less than 16, the following applies:

i = AlfCtbFiltSetIdxY[ xCtb  >>  CtbLog2SizeY ][ yCtb  >>  CtbLog2SizeY ] (1454)

f[ j ] = AlfFixFiltCoeff[ AlfClassToFiltMap[ i ][ filtIdx[ x ][ y ] ] ][ j ] (1455)

c[ j ] = 2BitDepth (1456)

* + Otherwise (AlfCtbFiltSetIdxY[ xCtb  >>  CtbLog2SizeY ][ yCtb  >>  CtbLog2SizeY ] is greater than or equal to 16, the following applies:

i = slice\_alf\_aps\_id\_luma[ AlfCtbFiltSetIdxY[ xCtb  >>  CtbLog2SizeY ][ yCtb  >>  CtbLog2SizeY ] − 16 ]  
 (1457)

f[ j ] = AlfCoeffL[ i ][ filtIdx[ x ][ y ] ][ j ] (1458)

c[ j ] = AlfClipL[ i ][ filtIdx[ x ][ y ] ][ j ] (1459)

* + The luma filter coefficients and clipping values index idx are derived depending on transposeIdx[ x ][ y ] as follows:
  + If transposeIndex[ x ][ y ] is equal to 1, the following applies:

idx[ ] = { 9, 4, 10, 8, 1, 5, 11, 7, 3, 0, 2, 6 } (1460)

* + Otherwise, if transposeIndex[ x ][ y ] is equal to 2, the following applies:

idx[ ] = { 0, 3, 2, 1, 8, 7, 6, 5, 4, 9, 10, 11 } (1461)

* + Otherwise, if transposeIndex[ x ][ y ] is equal to 3, the following applies:

idx[ ] = { 9, 8, 10, 4, 3, 7, 11, 5, 1, 0, 2, 6 } (1462)

* + Otherwise, the following applies:

idx[ ] = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 } (1463)

* + The locations ( hx + i, vy + j ) for each of the corresponding luma samples ( x, y ) inside the given array recPicture of luma samples with i, j = −3..3 are derived as follows:

hx + i = Clip3( 0, pic\_width\_in\_luma\_samples − 1, xCtb + x + i ) (1464)

vy + j = Clip3( 0, pic\_height\_in\_luma\_samples − 1, yCtb + y + j ) (1465)

* + The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with ( xCtb, yCtb ) ( x, y ), and 2 as inputs.
  + The variables hx + i and vy + j are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with ( xCtb, yCtb ), ( hx + i, vy + j ), 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.
  + The variable applyAlfLineBufBoundaryis derived as follows:
  + If the bottom boundary of the current coding tree block is the bottom boundary of current picture and pic\_height\_in\_luma\_samples − yCtb  <=  CtbSizeY − 4, applyAlfLineBufBoundary is set equal to 0:
  + Otherwise, applyAlfLineBufBoundary is set equal to 1.
  + The vertical sample position offsets y1, y2, y3 and the variable alfShiftY are specified in Table 45 according to the vertical luma sample position y and applyAlfLineBufBoundary.
  + The variable curr is derived as follows:

curr = recPicture[ hx ][ vy ] (1466)

* + The variable sum is derived as follows:

sum = f[ idx[ 0 ] ]   \* (  Clip3( −c[ idx[ 0 ] ], c[ idx[ 0 ] ],     recPicture[ hx ][ vy + y3 ] − curr ) +  
 Clip3( −c[ idx[ 0 ] ], c[ idx[ 0 ] ],     recPicture[ hx ][ vy − y3 ] − curr ) ) +  
 f[ idx[ 1 ] ]   \* (  Clip3( −c[ idx[ 1 ] ], c[ idx[ 1 ] ],     recPicture[ hx + 1 ][ vy + y2 ] − curr ) +  
 Clip3( −c[ idx[ 1 ] ], c[ idx[ 1 ] ],     recPicture[ hx − 1 ][ vy − y2 ] − curr ) ) +  
 f[ idx[ 2 ] ]   \* (  Clip3( −c[ idx[ 2 ] ], c[ idx[ 2 ] ],     recPicture[ hx ][ vy + y2 ] − curr ) +  
 Clip3( −c[ idx[ 2 ] ], c[ idx[ 2 ] ],     recPicture[ hx ][ vy − y2 ] − curr ) ) +  
 f[ idx[ 3 ] ]   \* (  Clip3( −c[ idx[ 3 ] ], c[ idx[ 3 ] ],     recPicture[ hx − 1 ][ vy + y2 ] − curr ) +  
 Clip3( −c[ idx[ 3 ] ], c[ idx[ 3 ] ],     recPicture[ hx + 1 ][ vy − y2 ] − curr ) ) +  
 f[ idx[ 4 ] ]   \* (  Clip3( −c[ idx[ 4 ] ], c[ idx[ 4 ] ],     recPicture[ hx + 2 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 4 ] ], c[ idx[ 4 ] ],     recPicture[ hx − 2 ][ vy − y1 ] − curr ) ) +  
 f[ idx[ 5 ] ]   \* (  Clip3( −c[ idx[ 5 ] ], c[ idx[ 5 ] ],     recPicture[ hx + 1 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 5 ] ], c[ idx[ 5 ] ],     recPicture[ hx − 1 ][ vy − y1 ] − curr ) ) +  
 f[ idx[ 6 ] ]   \* (  Clip3( −c[ idx[ 6 ] ], c[ idx[ 6 ] ],     recPicture[ hx ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 6 ] ], c[ idx[ 6 ] ],     recPicture[ hx ][ vy − y1 ] − curr ) ) + (1467)  
 f[ idx[ 7 ] ]   \* (  Clip3( −c[ idx[ 7 ] ], c[ idx[ 7 ] ],     recPicture[ hx − 1 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 7 ] ], c[ idx[ 7 ] ],     recPicture[ hx + 1 ][ vy − y1 ] − curr ) ) +  
 f[ idx[ 8 ] ]   \* (  Clip3( −c[ idx[ 8 ] ], c[ idx[ 8 ] ],     recPicture[ hx − 2 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ idx[ 8 ] ], c[ idx[ 8 ] ],     recPicture[ hx + 2 ][ vy − y1 ] − curr ) ) +  
 f[ idx[ 9 ] ]   \* (  Clip3( −c[ idx[ 9 ] ], c[ idx[ 9 ] ],     recPicture[ hx + 3 ][ vy ] − curr ) +  
 Clip3( −c[ idx[ 9 ] ], c[ idx[ 9 ] ],     recPicture[ hx − 3 ][ vy ] − curr ) ) +  
 f[ idx[ 10 ] ] \* (  Clip3( −c[ idx[ 10 ] ], c[ idx[ 10 ] ], recPicture[ hx + 2 ][ vy ] − curr ) +  
 Clip3( −c[ idx[ 10 ] ], c[ idx[ 10 ] ], recPicture[ hx − 2 ][ vy ] − curr ) ) +  
 f[ idx[ 11 ] ] \* (  Clip3( −c[ idx[ 11 ] ], c[ idx[ 11 ] ], recPicture[ hx + 1 ][ vy ] − curr ) +  
 Clip3( −c[ idx[ 11 ] ], c[ idx[ 11 ] ], recPicture[ hx − 1 ][ vy ] − curr ) )

sum = curr + ( ( sum + 64 )  >>  alfShiftY  ) (1468)

* + The modified filtered reconstructed luma picture sample alfPictureL[ xCtb + x ][ yCtb + y ] is derived as follows:

alfPictureL[ xCtb + x ][ yCtb + y ] = Clip3( 0, ( 1  <<  BitDepth ) − 1, sum ) (1469)

Table 45 – Specification of y1, y2, y3 and alfShiftY according to the vertical luma sample position y and applyAlfLineBufBoundary

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Condition | alfShiftY | y1 | y2 | y3 |
| ( y  = =  CtbSizeY − 5 | | y  = =  CtbSizeY − 4 ) &&  ( applyAlfLineBufBoundary  = =  1 ) | 10 | 0 | 0 | 0 |
| ( y  = =  CtbSizeY − 6 | | y  = =  CtbSizeY − 3 ) &&  ( applyAlfLineBufBoundary  = =  1 ) | 7 | 1 | 1 | 1 |
| ( y  = =  CtbSizeY − 7 | | y  = =  CtbSizeY − 2 ) &&  ( applyAlfLineBufBoundary  = =  1 ) | 7 | 1 | 2 | 2 |
| otherwise | 7 | 1 | 2 | 3 |

#### Derivation process for ALF transpose and filter index for luma samples

Inputs of this process are:

* a luma location ( xCtb, yCtb ) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
* a reconstructed luma picture sample array recPicture prior to the adaptive loop filtering process.

Outputs of this process are

* the classification filter index array filtIdx[ x ][ y ] with x, y = 0..CtbSizeY − 1,
* the transpose index array transposeIdx[ x ][ y ] with x, y = 0..CtbSizeY − 1.

The variables ac[ x ][ y ], sumH[ x ][ y ], sumV[ x ][ y ], sumD0[ x ][ y ], sumD1[ x ][ y ] and sumOfHV[ x ][ y ] with x, y = 0..( CtbSizeY − 1 )  >>  2 are derived as follows:

* + The variables x4 and y4 are set as ( x  <<  2 ) and ( y  <<  2 ), respectively.
* The variables minY, maxY, and ac[ x ][ y ] are derived as follows:
* If y4 is equal to ( CtbSizeY − 8 ) and one of the following condition is true, minY is set equal to −2, maxY is set equal to 3, and ac[ x ][ y ] is set equal to 3.
  + The bottom boundary of the current coding tree block is the bottom boundary of the picture and pic\_height\_in\_luma\_samples − yCtb > CtbSizeY − 4.
  + The bottom boundary of the current coding tree block is not the bottom boudary of the picture.
* Otherwise, if y4 is equal to ( CtbSizeY − 4 ) and one of the following condition is true, minY is set equal to 0, maxY is set equal to 5, and ac[ x ][ y ] is set equal to 3.
  + The bottom boundary of the current coding tree block is the bottom boundary of the picture and pic\_height\_in\_luma\_samples − yCtb > CtbSizeY − 4.
  + The bottom boundary of the current coding tree block is not the bottom boudary of the picture
* Otherwise, minY is set equal to −2 and maxY is set equal to 5, and ac[ x ][ y ] is set equal to 2.
  + The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with ( xCtb, yCtb ), ( x4, y4 ) and 2 as inputs.
  + The locations ( hx4 + i, vy4 + j ) for each of the corresponding luma samples inside the given array recPicture of luma samples with i, j = −3..6 are derived as follows:

hx4 + i = Clip3( 0, pic\_width\_in\_luma\_samples − 1, xCtb + x4 + i ) (1470)

vy4 + j = Clip3( 0, pic\_height\_in\_luma\_samples − 1, yCtb + y4 + j ) (1471)

* The variables hx4 + i and vy4 + j are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with ( xCtb, yCtb ), ( hx4 + i, vy4 + j ), the variable isChroma set equal to 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.
  + The variables filtH[ i ][ j ], filtV[ i ][ j ], filtD0[ i ][ j ] and filtD1[ i ][ j ] with i, j = −2..5 are derived as follows:
  + If both i and j are even numbers or both i and j are not even numbers, the following applies:

filtH[ i ][ j ] = Abs( ( recPicture[ hx4 + i ][ vy4 + j ]  <<  1 ) − recPicture[ hx4 +  i − 1 ][ vy4 +j ] − (1472)  
  recPicture[ hx4 + i + 1 ][ vy4 + j ] )

filtV[ i ][ j ] = Abs( ( recPicture[ hx4 + i ][ vy4 + j ]  <<  1 ) − recPicture[ hx4 + i ][ vy4 + j − 1 ] − (1473)  
  recPicture[ hx4 + i ][ vy4 + j + 1 ] )

filtD0[ i ][ j ] = Abs( ( recPicture[ hx4 + i ][ vy4 + j ]  <<  1 ) − recPicture[ hx4 + i − 1 ][ vy4 + j − 1 ] − (1474)  
 recPicture[ hx4 + i + 1 ][ vy4 + j + 1 ] )

filtD1[ i ][ j ] = Abs( ( recPicture[ hx4 + i ][ vy4 + j ]  <<  1 ) − recPicture[ hx4 + i + 1 ][ vy4 + j − 1 ] − (1475)  
 recPicture[ hx4 + i − 1 ][ vy4 + j + 1 ] )

* + Otherwise, filtH[ i ][ j ], filtV[ i ][ j ], filtD0[ i ][ j ] and filtD1[ i ][ j ] are set equal to 0.
  + The variables sumH[ x ][ y ], sumV[ x ][ y ], sumD0[ x ][ y ], sumD1[ x ][ y ] and sumOfHV[ x ][ y ] are derived as follows:

sumH[ x ][ y ] = ΣiΣj filtH[ i ][ j ], with i = −2..5, j = minY..maxY (1476)

sumV[ x ][ y ] = ΣiΣj filtV[ i ][ j ], with i = −2..5, j = minY..maxY (1477)

sumD0[ x ][ y ] = ΣiΣj filtD0[ i ][ j ], with i = −2..5, j = minY..maxY (1478)

sumD1[ x ][ y ] = ΣiΣj filtD1[ i ][ j ], with i = −2..5, j = minY..maxY (1479)

sumOfHV[ x ][ y ] = sumH[ x ][ y ] + sumV[ x ][ y ] (1480)

The classification filter index array filtIdx and transpose index array transposeIdx are derived by the following steps:

1. The variables dir1[ x ][ y ], dir2[ x ][ y ] and dirS[ x ][ y ] with x, y = 0..CtbSizeY − 1 are derived as follows:

* The variables hv1, hv0 and dirHV are derived as follows:
* If sumV[ x  >>  2 ][ y  >>  2 ] is greater than sumH[ x  >>  2 ][ y  >>  2 ], the following applies:

hv1 = sumV[ x  >>  2 ][ y  >>  2 ] (1481)

hv0 = sumH[ x  >>  2 ][ y  >>  2 ]  (1482)

dirHV = 1 (1483)

* Otherwise, the following applies:

hv1 = sumH[ x  >>  2 ][ y  >>  2 ] (1484)

hv0 = sumV[ x  >>  2 ][ y  >>  2 ] (1485)

dirHV = 3 (1486)

* The variables d1, d0 and dirD are derived as follows:
* If sumD0[ x  >>  2 ][ y  >>  2 ] is greater than sumD1[ x  >>  2 ][ y  >>  2 ], the following applies:

d1 = sumD0[ x  >>  2 ][ y  >>  2 ] (1487)

d0 = sumD1[ x  >>  2 ][ y  >>  2 ] (1488)

dirD = 0 (1489)

* Otherwise, the following applies:

d1 = sumD1[ x  >>  2 ][ y  >>  2 ] (1490)

d0 = sumD0[ x  >>  2 ][ y  >>  2 ] (1491)

dirD = 2 (1492)

* The variables hvd1, hvd0, are derived as follows:

hvd1 = ( d1 \* hv0 > hv1 \* d0 ) ? d1 : hv1 (1493)

hvd0 = ( d1 \* hv0 > hv1 \* d0 ) ? d0 : hv0 (1494)

* The variables dirS[ x ][ y ], dir1[ x ][ y ] and dir2[ x ][ y ] derived as follows:

dir1[ x ][ y ] = ( d1 \* hv0 > hv1 \* d0 ) ? dirD : dirHV (1495)

dir2[ x ][ y ] = ( d1 \* hv0 > hv1 \* d0 ) ? dirHV : dirD (1496)

dirS[ x ][ y ] = ( hvd1 \*2 > 9 \* hvd0 ) ? 2 : ( ( hvd1 > 2 \* hvd0 ) ? 1 : 0 ) (1497)

1. The variable avgVar[ x ][ y ] with x, y = 0..CtbSizeY − 1 is derived as follows:

varTab[ ] = { 0, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 4 } (1498)

avgVar[ x ][ y ] = varTab[ Clip3( 0, 15, ( sumOfHV[ x  >>  2 ][ y  >>  2 ] \* (1499)  
 ac[ x  >>  2 ][ y  >>  2 ] )  >>  ( BitDepth − 1 ) ) ]

1. The classification filter index array filtIdx[ x ][ y ] and the transpose index array transposeIdx[ x ][ y ] with x = y = 0..CtbSizeY − 1 are derived as follows:

transposeTable[ ] = { 0, 1, 0, 2, 2, 3, 1, 3 }

transposeIdx[ x ][ y ] = transposeTable[ dir1[ x ][ y ] \* 2 + ( dir2[ x ][ y ]  >>  1 ) ]

filtIdx[ x ][ y ] = avgVar[ x ][ y ]

* When dirS[ x ][ y ] is not equal 0, filtIdx[ x ][ y ] is modified as follows:

filtIdx[ x ][ y ] += ( ( ( dir1[ x ][ y ] & 0x1 )  <<  1 ) + dirS[ x ][ y ] ) \* 5 (1500)

#### Coding tree block filtering process for chroma samples

Inputs of this process are:

* a reconstructed chroma picture sample array recPicture prior to the adaptive loop filtering process,
* a filtered reconstructed chroma picture sample array alfPicture,
* a chroma location ( xCtbC, yCtbC ) specifying the top-left sample of the current chroma coding tree block relative to the top left sample of the current picture,
* an alternative chroma filter index altIdx.

Output of this process is the modified filtered reconstructed chroma picture sample array alfPicture.

The width and height of the current chroma coding tree block ctbWidthC and ctbHeightC is derived as follows:

ctbWidthC = CtbSizeY / SubWidthC (1501)

ctbHeightC = CtbSizeY / SubHeightC (1502)

For the derivation of the filtered reconstructed chroma samples alfPicture[ x ][ y ], each reconstructed chroma sample inside the current chroma coding tree block recPicture[ x ][ y ] is filtered as follows with x = 0..ctbWidthC − 1, y = 0..ctbHeightC − 1:

* + The locations ( hx + i, vy + j ) for each of the corresponding chroma samples ( x, y ) inside the given array recPicture of chroma samples with i, j = −2..2 are derived as follows:

hx + i = Clip3( 0, pic\_width\_in\_luma\_samples / SubWidthC − 1, xCtbC + x + i ) (1503)

vy + j = Clip3( 0, pic\_height\_in\_luma\_samples / SubHeightC − 1, yCtbC + y + j ) (1504)

* + The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with ( xCtbC \* SubWidthC, yCtbC \* SubHeightC ), ( x \* SubWidthC, y \*SubHeightC ) and SubHeightC as inputs.
  + The variables hx + i and vy + j are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with ( xCtb, yCtb ), ( hx + i, vy + j ), the variable isChroma set equal to 1, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.
  + The variable applyAlfLineBufBoundary is derived as follows:
  + If the bottom boundary of the current coding tree block is the bottom boundary of the picture and pic\_height\_in\_luma\_samples − ( yCtbC \* SubHeightC ) < CtbSizeY – 2\*SubHeightC, applyAlfLineBufBoundary is set equal to 0.
  + Otherwise, applyAlfLineBufBoundary is set equal to 1.
  + The vertical sample position offsets y1, y2 and the variable alfShiftC are specified in Table 45 according to the vertical chroma sample position y and applyAlfLineBufBoundary.
  + The variable curr is derived as follows:

curr = recPicture[ hx ][ vy ] (1505)

* + The array of chroma filter coefficients f[ j ] and the array of chroma clipping values c[ j ] is derived as follows with j = 0..5:

f[ j ] = AlfCoeffC[ slice\_alf\_aps\_id\_chroma ][ altIdx ][ j ] (1506)

c[ j ] = AlfClipC[ slice\_alf\_aps\_id\_chroma ][ altIdx ][ j ] (1507)

* + The variable sum is derived as follows:

sum = f[ 0 ] \* (  Clip3( −c[ 0 ], c[ 0 ], recPicture[ hx ][ vy + y2 ] − curr ) +  
 Clip3( −c[ 0 ], c[ 0 ], recPicture[ hx ][ vy− y2 ] − curr ) ) +  
 f[ 1 ] \* ( Clip3( −c[ 1 ], c[ 1 ], recPicture[ hx + 1 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ 1 ], c[ 1 ], recPicture[ hx − 1 ][ vy − y1 ] − curr ) ) +  
 f[ 2 ] \* ( Clip3( −c[ 2 ], c[ 2 ], recPicture[ hx ][ vy + y1 ] − curr ) +  
 Clip3( −c[ 2 ], c[ 2 ], recPicture[ hx ][ vy − y1 ] − curr ) ) + (1508)  
 f[ 3 ] \* ( Clip3( −c[ 3 ], c[ 3 ], recPicture[ hx − 1 ][ vy + y1 ] − curr ) +  
 Clip3( −c[ 3 ], c[ 3 ], recPicture[ hx + 1 ][ vy − y1 ] − curr ) ) +  
 f[ 4 ] \* ( Clip3( −c[ 4 ], c[ 4 ], recPicture[ hx + 2 ][ vy ] − curr ) +  
 Clip3( −c[ 4 ], c[ 4 ], recPicture[ hx − 2 ][ vy ] − curr ) ) +  
 f[ 5 ] \* ( Clip3( −c[ 5 ], c[ 5 ], recPicture[ hx + 1 ][ vy ] − curr ) +  
 Clip3( −c[ 5 ], c[ 5 ], recPicture[ hx − 1 ][ vy ] − curr ) )

sum = curr + ( ( sum + 64 )  >>  alfShiftC ) (1509)

* + The modified filtered reconstructed chroma picture sample alfPicture[ xCtbC + x ][ yCtbC + y ] is derived as follows:

alfPicture[ xCtbC + x ][ yCtbC + y ] = Clip3( 0, ( 1  <<  BitDepth ) − 1, sum ) (1510)

Table 46 – Specification of y1, y2 and alfShiftC according to the vertical chroma sample position y and applyAlfLineBufBoundary

|  |  |  |  |
| --- | --- | --- | --- |
| Condition | alfShiftC | y1 | y2 |
| ( y = = ctbHeightC − 2 | | y = = ctbHeightC − 3 ) &&  ( applyAlfLineBufBoundary = = 1 ) | 10 | 0 | 0 |
| ( y = = ctbHeightC − 1 | | y = = ctbHeightC − 4 ) &&  ( applyAlfLineBufBoundary = = 1 ) | 7 | 1 | 1 |
| Otherwise | 7 | 1 | 2 |

#### ALF boundary position derivation process

Inputs of this process are:

* + a luma location ( xCtb, yCtb ) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
  + a luma location ( x, y ) specifying the current sample relative to the top-left sample of the current luma coding tree block.
* scaling factor scaleHeight.

Output of this process are:

* + the left vertical boundary position clipLeftPos,
  + the right vertical boundary position clipRightPos,
  + the above horizontal boundary position clipTopPos,
  + the below horizontal boundary position clipBottomPos,
  + the top left boundary flag clipTopLeftFlag,
  + the bottom right boundary flag clipBotRightFlag.

The variables clipLeftPos, clipRightPos, clipTopPos and clipBottomPos are set equal to −128.

The variables clipTopLeftFlag and clipBotRightFlag are both set equal to 0.

The variable clipTopPos is modified as follows:

* + If y − ( CtbSizeY – 2\*scaleHeight ) is greater than or equal to 0, the variable clipTopPos is set equal to yCtb + CtbSizeY – 2\*scaleHeight.
  + Otherwise, if VirtualBoundariesPresentFlag is equal to 1, and yCtb + y − VirtualBoundaryPosY[ n ] is greater than or equal to 0 and less than 3 for any n = 0..NumHorVirtualBoundaries − 1, the following applies:

clipTopPos = VirtualBoundaryPosY[ n ] (1511)

* + Otherwise, if y is less than 3 and one or more of the following conditions are true, the variable clipTopPos is set equal to yCtb:
    - The top boundary of the current coding tree block is the top boundary of the tile, and pps\_loop\_filter\_across\_tiles\_enabled\_flag is equal to 0.
    - The top boundary of the current coding tree block is the top boundary of the slice, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0.
    - The top boundary of the current coding tree block is the top boundary of the subpicture, and loop\_filter\_across\_subpic\_enabled\_flag[ CurrSubpicIdx ] is equal to 0.

The variable clipBottomPos is modified as follows:

* + If VirtualBoundariesPresentFlag is equal to 1, VirtualBoundaryPosY[ n ] is not equal to pic\_height\_in\_luma\_samples − 1 or 0, and VirtualBoundaryPosY[ n ] − yCtb − y is greater than 0 and less than 5 for any n = 0..NumHorVirtualBoundaries − 1, the following applies:

clipBottomPos = VirtualBoundaryPosY[ n ] (1512)

* + Otherwise, if CtbSizeY – 2\*scaleHeight − y is greater than 0 and is less than 5, the variable clipBottomPos is set equal to yCtb + CtbSizeY – 2\*scaleHeight.
  + Otherwise, if CtbSizeY − y is less than 5, and one or more of the following conditions are true, the variable clipBottomPos is set equal to yCtb + CtbSizeY:
    - The bottom boundary of the current coding tree block is the bottom boundary of the tile, and pps\_loop\_filter\_across\_tiles\_enabled\_flag is equal to 0.
    - The bottom boundary of the current coding tree block is the bottom boundary of the slice, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0.
    - The bottom boundary of the current coding tree block is the bottom boundary of the subpicture, and loop\_filter\_across\_subpic\_enabled\_flag[ CurrSubpicIdx ] is equal to 0.

The variable clipLeftPos is modified as follows:

* + If VirtualBoundariesPresentFlag is equal to 1, and xCtb + x − VirtualBoundaryPosX[ n ] is greater than or equal to 0 and less than 3 for any n = 0..NumVerVirtualBoundaries − 1, the following applies:

clipLeftPos = VirtualBoundaryPosX[ n ] (1513)

* + Otherwise, if x is less than 3, and one or more of the following conditions are true, the variable clipLeftPos is set equal to xCtb:
    - The left boundary of the current coding tree block is the left boundary of the tile, and pps\_loop\_filter\_across\_tiles\_enabled\_flag is equal to 0.
    - The left boundary of the current coding tree block is the left boundary of the slice, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0.
    - The left boundary of the current coding tree block is the left boundary of the subpicture, and loop\_filter\_across\_subpic\_enabled\_flag[ CurrSubpicIdx ] is equal to 0.

The variable clipRightPos is modified as follows:

* + If VirtualBoundariesPresentFlag is equal to 1, and VirtualBoundaryPosX[ n ]− xCtb − x is greater than 0 and less than 5 for any n = 0..NumVerVirtualBoundaries − 1, the following applies:

clipRightPos = VirtualBoundaryPosX[ n ] (1514)

* + Otherwise, if CtbSizeY − x is less than 5, and one or more of the following conditions are true, the variable clipRightPos is set equal to xCtb + CtbSizeY:
    - The right boundary of the current coding tree block is the right boundary of the tile, and loop\_filter\_across\_tiless\_enabled\_flag is equal to 0.
    - The right boundary of the current coding tree block is the right boundary of the slice, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0.
    - The right boundary of the current coding tree block is the right boundary of the subpicture, and loop\_filter\_across\_subpic\_enabled\_flag[ CurrSubpicIdx ] is equal to 0.

The variable clipTopLeftFlag and clipBotRightFlag are modified as following:

* + If the coding tree block covering the luma position ( xCtb, yCtb ) and the coding tree block covering the luma position ( xCtb − CtbSizeY, yCtb − CtbSizeY) belong to different slices, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0, clipTopLeftFlag is set equal to 1.
  + If the coding tree block covering the luma position ( xCtb, yCtb ) and the coding tree block covering the luma position ( xCtb + CtbSizeY, yCtb + CtbSizeY) belong to different slices, and pps\_loop\_filter\_across\_slices\_enabled\_flag is equal to 0, clipBotRightFlag is set equal to 1.

#### ALF sample padding process

Inputs of this process are:

* + a luma location ( xCtb, yCtb ) specifying the top-left sample of the current luma coding tree block relative to the top left sample of the current picture,
  + a luma location ( x, y ) specifying the neighbouring sample relative to the top-left sample of the current picture,
  + a flag isChroma specifiying whether the colour componenet is chroma component or not,
  + the left vertical boundary position clipLeftPos,
  + the right vertical boundary position clipRightPos,
  + the above horizontal boundary position clipTopPos,
  + the below horizontal boundary position clipBottomPos,
  + the top left boundary flag clipTopLeftFlag,
  + the bottom right boundary flag clipBotRightFlag.

Outputs of this process are:

* + modified luma location ( x, y ) specifying the neighbouring sample relative to the top-left sample of the current picture,

The variables picWidth, picHeight, xCtbCur, yCtbCur, CtbSizeHor, CtbSizeVer, topBry, botBry, leftBry and rightBry are derived as follows:

picWidth = isChroma ? pic\_width\_in\_luma\_samples / SubWidthC : pic\_width\_in\_luma\_samples (1515)

picHeight = isChroma ? pic\_height\_in\_luma\_samples / SubHeightC : pic\_height\_in\_luma\_samples (1516)

xCtbCur = isChroma ? xCtb / SubWidthC : xCtb (1517)

yCtbCur = isChroma ? yCtb / SubHeightC : yCtb (1518)

ctbSizeHor = isChroma ? CtbSizeY / SubWidthC : CtbSizeY (1519)

ctbSizeVer = isChroma ? CtbSizeY / SubHeightC : CtbSizeY (1520)

topBryPos = isChroma ? clipTopPos / SubHeightC : clipTopPos (1521)

botBryPos = isChroma ? clipBottomPos / SubHeightC : clipBottomPos (1522)

leftBryPos = isChroma ? clipLeftPos / SubWidthC : clipLeftPos (1523)

rightBryPos = isChroma ? clipRightPos / SubWidthC : clipRightPos (1524)

The variables ( x , y ) is modified as follows:

* + When topBryPos is not less than 0, the following applies:

y = Clip3( topBryPos, picHeight − 1, y ) (1525)

* + When botBryPos is not less than 0, the following applies:

y = Clip3( 0, botBryPos − 1, y ) (1526)

* + When leftBryPos is not less than 0, the following applies:

x = Clip3( leftBryPos, picWidth − 1,  x ) (1527)

* + When rightBryPos is not less than 0, the following applies:

x = Clip3( 0, rightBryPos − 1, x ) (1528)

* + ( x, y ) is set equal to ( xCtbCur, y ) if all of the followig conditions are true:
    - clipTopLeftFlag is equal to true
    - topBryPos is less than 0 and leftBryPos is less than 0
    - x is less than xCtbCur and y is less than yCtbCur
  + ( x, y ) is set equal to ( xCtbCur + CtbSizeHor − 1, y ) if all of the followig conditions are true:
    - clipBotRightFlag is equal to true
    - botBryPos is less than 0 and rightBryPos is less than 0
    - x is greater than xCtbCur + CtbSizeHor − 1 and y is greater than yCtbCur + CtbSizeVer − 1

#### Cross-component filtering process

Inputs of this process are:

1. a reconstructed luma picture sample array recPictureL prior to the luma adaptive loop filtering process,
2. a filtered reconstructed chroma picture sample array alfPictureC,
3. a chroma location ( xCtbC, yCtbC ) specifying the top-left sample of the current chroma coding tree block relative to the top-left sample of the current picture,
4. a CTB width ccAlfWidth in chroma samples,
5. a CTB height ccAlfHeight in chroma samples,
6. cross-component filter coefficients CcAlfCoeff[ j ], with j = 0..6.

Output of this process is the modified filtered reconstructed chroma picture sample array ccAlfPicture.

For the derivation of the filtered reconstructed chroma samples ccAlfPicture[ xCtbC + x ][ yCtbC + y ], each reconstructed chroma sample inside the current chroma block of samples alfPictureC[ xCtbC + x ][ yCtbC + y ] with x = 0..ccAlfWidth − 1, y = 0..ccAlfHeight − 1, is filtered as follows:

* + The luma location ( xL, yL ) corresponding to the current chroma sample at chroma location ( xCtbC + x, yCtbC + y ) is set equal to ( ( xCtbC + x ) \* SubWidthC, ( yCtbC + y ) \* SubHeightC ).
  + The luma locations ( hx + i, vy + j ) with i = −1..1, j = −1..2 inside the array recPictureL are derived as follows:

hx + i = Clip3( 0, pic\_width\_in\_luma\_samples − 1, xL + i ) (1529)

vy + j = Clip3( 0, pic\_height\_in\_luma\_samples − 1, yL + j ) (1530)

* The variables clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag are derived by invoking the ALF boundary position derivation process as specified in clause 8.8.5.5 with ( xCtbC \* SubWidthC, yCtbC \* SubHeightC ), ( x \* SubWidthC, y \*SubHeightC ) and 2 as inputs.
* The variables hx + i and vy + j are modified by invoking the ALF sample padding process as specified in clause 8.8.5.6 with ( xCtbC \* SubWidthC, yCtbC \* SubHeightC ), ( hx + i, vy + j ), the variable isChroma set equal to 0, clipLeftPos, clipRightPos, clipTopPos, clipBottomPos, clipTopLeftFlag and clipBotRightFlag as input.
* The variable applyAlfLineBufBoundary is derived as follows:
  + If the bottom boundary of the current coding tree block is the bottom boundary of current picture and pic\_height\_in\_luma\_samples − yCtbC \* SubHeightC is less then or equal to CtbSizeY − 4, applyAlfLineBufBoundary is set equal to 0.
  + Otherwise, applyAlfLineBufBoundary is set equal to 1.
  + The vertical sample position offsets yP1 and yP2 are specified in Table 47 according to the vertical luma sample position (y \* subHeightC ) and applyAlfLineBufBoundary.
  + The variable curr is derived as follows:

curr = alfPictureC[ xCtbC + x][ yCtbC + y ] (1531)

* + The array of cross-component filter coefficients f[ j ] is derived as follows with j = 0..6:

f[ j ] = CcAlfCoeff[ j ] (1532)

* + The variable sum is derived as follows:

sum =  f[ 0 ] \* ( recPictureL[ hx ][ vy - yP1 ] − recPictureL[ hx ][ vy ] ) +  
 f[ 1 ] \* ( recPictureL[ hx-1 ][ vy ] − recPictureL[ hx ][ vy ] ) +  
 f[ 2 ] \* ( recPictureL[ hx+1 ][ vy ] − recPictureL[ hx ][ vy ] ) + (1533) f[ 3 ] \* ( recPictureL[ hx-1 ][ vy+yP1 ] − recPictureL[ hx ][ vy ] ) +  
 f[ 4 ] \* ( recPictureL[ hx ][ vy+yP1 ] − recPictureL[ hx ][ vy ] ) +  
 f[ 5 ] \* ( recPictureL[ hx+1 ][ vy+yP1 ] − recPictureL[ hx ][ vy ] ) +   
 f[ 6 ] \* ( recPictureL[ hx ][ vy + yP2 ] − recPictureL[ hx ][ vy ] )

scaledSum = Clip3( −( 1 << ( BitDepth − 1 ) ), ( 1 << ( BitDepth  − 1 ) ) − 1, ( sum + 64 ) >> 7) (1534)

sum = curr + scaledSum (1535)

* + The modified filtered reconstructed chroma picture sample ccAlfPicture[ xCtbC + x ][ yCtbC + y ] is derived as follows:

ccAlfPicture[ xCtbC + x ][ yCtbC + y ] = Clip3( 0, ( 1 << BitDepth ) − 1, sum ) (1536)

Table 47 – Specification of yP1 and yP2 according to the vertical luma sample position ( y \* subHeightC ) and applyAlfLineBufBoundary

|  |  |  |
| --- | --- | --- |
| Condition | yP1 | yP2 |
| ( y \* subHeightC  = =  CtbSizeY − 5  | |  y \* subHeightC  = =  CtbSizeY − 4 ) && applyAlfLineBufBoundary  = =  1 | 0 | 0 |
| ( y \* subHeightC  = =  CtbSizeY − 6  | |  y \* subHeightC  = =  CtbSizeY − 3 ) && applyAlfLineBufBoundary  = =  1 | 1 | 1 |
| Otherwise | 1 | 2 |