#### Adaptive loop filter unit syntax

|  |  |
| --- | --- |
| alf\_unit(rx, ry, cIdx, lcuIdx, endrX, oneUnitFlag ) { | Descriptor |
| if( oneUnitFlag ) { |  |
| if( lcuIdx = = 0 ) { |  |
| **alf\_lcu\_enable\_flag**[ cIdx ][ ry ][ rx ] | u(1) |
| if( alf\_lcu\_enable\_flag[ cIdx ][ ry ][ rx ] ) |  |
| alf\_info( rx, ry, cIdx ) |  |
| } |  |
| } else { |  |
| if( !alf\_repeat\_row \_flag[ cIdx ] ) { |  |
| if( rx = = 0 || alfRun[ cIdx ][ ry ][ rx ] < 0 ) { |  |
| if( lcuIdx –numCtbInWidth < 0 ) |  |
| alfRun[ cIdx ][ ry ][ rx ] = 0 + **alf\_run\_diff** | u(v) |
| else |  |
| alfRun[ cIdx ][ ry ][ rx ] = alfRun[ cIdx ][ ry − 1 ][ rx ]+ **alf\_run\_diff** | s(v) |
|  |  |
|  |  |
| if( ry > 0 && ( lcuIdx − numCtbInWidth > = 0 | | alfAcrossSlice) ) |  |
| **alf\_merge\_up\_flag** | u(1) |
| if( !alf\_merge\_up\_flag ) { |  |
| **alf\_lcu\_enable\_flag**[ cIdx ][ ry ][ rx ] | u(1) |
| if( alf\_lcu\_enable\_flag[ cIdx ][ ry ][ rx ] ) |  |
| alf\_info( rx, ry, cIdx ) |  |
| } |  |
| } |  |
| alfRun[ cIdx ][ ry ][ rx+ 1 ] = alfRun[ cIdx ][ ry ][ rx ] − 1 |  |
| } else |  |
| alfRun[ cIdx ][ ry ][ rx ] = alfRun[ cIdx ][ ry − 1 ][ rx ] |  |
| } |  |
| } |  |

#### Derivation process for filter index array for luma samples

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 log2CbSize specifying the size of the current coding unit.

Output of this process is the two-dimensional filter index array of (nS)x(nS), fIdx.

A variable nS is set equal to ( 1 << log2CbSize ).

The boundary padding process specified in subclause is invoked with the luma location ( xC, yC ), the size of coding unit log2CbSize and the chroma component index cIdx set equal to 0, and the output is assigned to the luma sample array s’’.

The filter index array fIdx is specified in the follows:

1. For x, y = 0…(nS-1), when one of the following condition is true

* x is equal to ( ((x>>2) <<2)+1) or ( ((x>>2) <<2)+2)
* y is equal to ( ((y>>2) <<2)+1) or ( ((y>>2) <<2)+2)

The variables varTempH[ x ][ y ], varTempV[ x ][ y ] and varTemp1[ x ][ y ] are derived as

varTempH[ x ][ y ] = | ( s’’[ xC+x, yC+y ] << 1 ) – s’’[ xC+x−1, yC+y ] – s’’[ xC+x+1, yC+y ] | (8‑375)  
varTempV[ x ][ y ] = | ( s’’[ xC+x, yC+y ] << 1 ) – s’’[ xC+x, yC+y−1 ] – s’’[ xC+x, yC+y+1 ] | (8‑376)

#### 8.7.3.5 Filtering process for chroma samples

Inputs of this process are:

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

– a variable log2CbSize specifying the size of the current coding unit.

– a variable cIdx specifying the chroma component index.

Output of this process is the filtered reconstruction of chroma picture.

The boundary padding process specified in subclause is invoked with the chroma location ( xC, yC ), the size of coding unit log2CbSize and the chroma component index cIdx, and the output is assigned to the luma sample array s’’.

A variable nS is set equal to ( 1 << log2CbSize ).

A variable lcuHeight is set equal to ( 1 << ( Log2CtbSize − 1 ) ) and a variable vbLine is set equal to lcuHeight − 2.

Filtered samples of chroma picture recFiltPicture[ xC + x ][ yC + y ] with x, y = 0..(nS)−1, are derived as the following ordered steps:

1. A variable dist2VB is derived as follows.

dist2VB = ( ( yC + y ) % lcuHeight – vbLine ) (8‑384)

1. A variable dist2VB is modified as follows.

* If dist2VB is less than –vbLine+2 and yC is larger than 2, dist2VB is set equal to dist2VB+lcuHeight,
* Otherwise, if yC+lcuHeight is greater than or equal to pic\_height\_in\_luma\_samples >> 1, dist2VB is set equal to 5.

1. The variable alfPrecisionBit is derived as specified in , horPos[ i ] and verPos[ i ] are specified in and , respectively.
2. The following applies.

recFiltPictureC[ xC + x ][ yC + y ] = Σi(s’’[ xC + x + horPos[ i ], yC + y + verPos[ i ] ] \* cC [ i ])   
 with i=0..18

recFiltPictureC[ xC + x][ yC + y ] = ( recFiltPictureC[xC + x][yC + y] +   
 ( 1 << ( alfPrecisionBit − 1) ) ) >> alfPrecisionBit