* + - 1. General slice header syntax

|  |  |
| --- | --- |
| slice\_header( ) { | **Descriptor** |
| **first\_slice\_in\_pic\_flag** | u(1) |
| if( RapPicFlag ) |  |
| **no\_output\_of\_prior\_pics\_flag** | u(1) |
| **pic\_parameter\_set\_id** | ue(v) |
| if( !first\_slice\_in\_pic\_flag ) |  |
| **slice\_address** | u(v) |
| if( dependent\_slice\_enabled\_flag && !first\_slice\_in\_pic\_flag ) |  |
| **dependent\_slice\_flag** | u(1) |
| if( !dependent\_slice\_flag ) { |  |
| **slice\_type** | ue(v) |
| if( output\_flag\_present\_flag ) |  |
| **pic\_output\_flag** | u(1) |
| if( separate\_colour\_plane\_flag = = 1 ) |  |
| **colour\_plane\_id** | u(2) |
| if( !IdrPicFlag ) { |  |
| **pic\_order\_cnt\_lsb** | u(v) |
| **short\_term\_ref\_pic\_set\_sps\_flag** | u(1) |
| if( !short\_term\_ref\_pic\_set\_sps\_flag ) |  |
| short\_term\_ref\_pic\_set( num\_short\_term\_ref\_pic\_sets ) |  |
| else |  |
| **short\_term\_ref\_pic\_set\_idx** | u(v) |
| if( long\_term\_ref\_pics\_present\_flag ) { |  |
| if( num\_long\_term\_ref\_pics\_sps > 0 ) |  |
| **num\_long\_term\_sps** | ue(v) |
| **num\_long\_term\_pics** | ue(v) |
| for( i = 0; i < num\_long\_term\_sps + num\_long\_term\_pics; i++ ) { |  |
| if( i < num\_long\_term\_sps ) |  |
| **lt\_idx\_sps**[ i ] | u(v) |
| else { |  |
| **poc\_lsb\_lt**[ i ] | u(v) |
| **used\_by\_curr\_pic\_lt\_flag**[ i ] | u(1) |
| } |  |
| **delta\_poc\_msb\_present\_flag**[ i ] | u(1) |
| if( delta\_poc\_msb\_present\_flag[ i ] ) |  |
| **delta\_poc\_msb\_cycle\_lt**[ i ] | ue(v) |
| } |  |
| } |  |
| } |  |
| if( sample\_adaptive\_offset\_enabled\_flag ) { |  |
| **slice\_sao\_luma\_flag** | u(1) |
| **slice\_sao\_chroma\_flag** | u(1) |
| } |  |
| if( slice\_type = = P | | slice\_type = = B ) { |  |
| if( sps\_temporal\_mvp\_enable\_flag ) |  |
| **slice\_temporal\_mvp\_enable\_flag** | u(1) |
| **num\_ref\_idx\_active\_override\_flag** | u(1) |
| if( num\_ref\_idx\_active\_override\_flag ) { |  |
| **num\_ref\_idx\_l0\_active\_minus1** | ue(v) |
| if( slice\_type = = B ) |  |
| **num\_ref\_idx\_l1\_active\_minus1** | ue(v) |
| } |  |
| if( lists\_modification\_present\_flag ) |  |
| ref\_pic\_list\_modification( ) |  |
| if( slice\_type = = B ) |  |
| **mvd\_l1\_zero\_flag** | u(1) |
| if( cabac\_init\_present\_flag ) |  |
| **cabac\_init\_flag** | u(1) |
| if( slice\_temporal\_mvp\_enable\_flag ) { |  |
| if( slice\_type = = B ) |  |
| **collocated\_from\_l0\_flag** | u(1) |
| if( ( collocated\_from\_l0\_flag && num\_ref\_idx\_l0\_active\_minus1 > 0 )  | | ( !collocated\_from\_l0\_flag &&  num\_ref\_idx\_l1\_active\_minus1 > 0 ) ) |  |
| **collocated\_ref\_idx** | ue(v) |
| } |  |
| If (iv\_mv\_pred\_flag[ nuh\_layer\_id ] || iv\_res\_pred\_flag[ nuh\_layer\_id ]) { |  |
| If (slice\_type==B) |  |
| **refview\_from\_l0\_flag** | u(1) |
| If ((refview\_from\_l0\_flag==1 && num\_ref\_idx\_l0\_active\_minus1 > 0 )  | | ( refview\_from\_l0\_flag==0 &&  num\_ref\_idx\_l1\_active\_minus1 > 0 ) ) |  |
| **refview\_ref\_idx** | ue(v) |
| } |  |
| if( ( weighted\_pred\_flag && slice\_type = = P) | |  ( weighted\_bipred\_flag && slice\_type = = B ) ) |  |
| pred\_weight\_table( ) |  |
| else if ( layer\_id ) |  |
| **slice\_ic\_enable\_flag** [ Ed.(GT): Should be moved to extension.. ] | u(1) |
| **five\_minus\_max\_num\_merge\_cand** | ue(v) |
| } |  |
| **slice\_qp\_delta** | se(v) |
| if( pic\_slice\_level\_chroma\_qp\_offsets\_present\_flag ) { |  |
| **slice\_cb\_qp\_offset** | se(v) |
| **slice\_cr\_qp\_offset** | se(v) |
| } |  |
| if( deblocking\_filter\_control\_present\_flag ) { |  |
| if( deblocking\_filter\_override\_enabled\_flag ) |  |
| **deblocking\_filter\_override\_flag** | u(1) |
| if( deblocking\_filter\_override\_flag ) { |  |
| **slice\_header\_disable\_deblocking\_filter\_flag** | u(1) |
| if( !slice\_header\_disable\_deblocking\_filter\_flag ) { |  |
| **beta\_offset\_div2** | se(v) |
| **tc\_offset\_div2** | se(v) |
| } |  |
| } |  |
| } |  |
| if( loop\_filter\_across\_slices\_enabled\_flag &&  ( slice\_sao\_luma\_flag | | slice\_sao\_chroma\_flag | |   !disable\_deblocking\_filter\_flag ) ) |  |
| **slice\_loop\_filter\_across\_slices\_enabled\_flag** | u(1) |
| } |  |
| if( tiles\_enabled\_flag | | entropy\_coding\_sync\_enabled\_flag ) { |  |
| **num\_entry\_point\_offsets** | ue(v) |
| if( num\_entry\_point\_offsets > 0 ) { |  |
| **offset\_len\_minus1** | ue(v) |
| for( i = 0; i < num\_entry\_point\_offsets; i++ ) |  |
| **entry\_point\_offset**[ i ] | u(v) |
| } |  |
| } |  |
| if( slice\_header\_extension\_present\_flag ) { |  |
| **slice\_header\_extension\_length** | ue(v) |
| slice\_header\_extension |  |
| slice\_header\_extension2\_flag | u(1) |
| if ( slice\_header\_extension2\_flag ) { |  |
| slice\_header\_extension2\_length | ue(v) |
| for( i = 0; i < slice\_header\_extension2\_length; i++) |  |
| **slice\_header\_extension2\_data\_byte**[ i ] | u(8) |
| } |  |
| } |  |
| byte\_alignment( ) |  |
| } |  |

* + - 1. General slice header semantics

……

**refview\_from\_l0\_flag** equal to 1 specifies that the reference view used for inter-view motion prediction and inter-view residual prediction is derived from reference picture list 0. refview\_from\_l0\_flag equal to 0 specifies that the collocated picture used for inter-view motion prediction and inter-view residual prediction is derived from reference picture list 1. When refview\_from\_l0\_flag is not present, it is inferred to be equal to 1.

**refview\_ref\_idx** specifies the reference index of the reference view used for inter-view motion prediction and inter-view residual prediction.

When slice\_type is equal to P or when slice\_type is equal to B and refview\_from\_l0 is equal to 1, refview\_ref\_idx refers to a picture in list 0, and the value of refview\_ref\_idx shall be in the range of 0 to num\_ref\_idx\_l0\_active\_minus1, inclusive.

When slice\_type is equal to B and refview\_from\_l0 is equal to 0, refview\_ref\_idx refers to a picture in list 1, and the value of refview\_ref\_idx shall be in the range of 0 to num\_ref\_idx\_l1\_active\_minus1, inclusive.

It is a requirement of bitstream conformance that the picture referred to by refview\_ref\_idx shall be the same for all slices of a coded picture.

1. * + - 1. Derivation process for a temporal inter-view motion vector candidate

This process is not invoked when iv\_mv\_pred\_flag[ nuh\_layer\_id ] is equal to 0.

Inputs to this process are:

* a luma location ( xP, yP ) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
* variables nPSW and nPSH specifying the width and the height, respectively, of the current prediction unit,
* a prediction list indication X,
* a reference view index refViewIdx.
* a disparity vector mvDisp,
* a flag mergeFlag specifying whether a merge candidate is derived,
* a reference index refIdxLX specifying a reference picture in the reference picture list RefPicListLX.

Outputs of this process are:

* a flag availableFlagLXInterView specifying whether the temporal inter-view motion vector candidate is available,
* a temporal inter-view motion vector candidate mvLXInterView (if availableFlagLXInterView is equal to 1),
* a reference index refIdxLX specifying a reference picture in the reference picture list RefPicListLX,

The flag availableFlagLXInterView is set equal to 0, both components of mvLXInterView are set equal to 0.

When X is equal to 1 and the current slice is not a B slice the whole decoding process specified in this subclause terminates.

The reference layer luma location ( xRef, yRef ) is derived by

* 1. xRef = Clip3( 0, PicWidthInSamplesL – 1, xP + ( ( nPSW – 1 ) >> 1 ) + ( ( mvDisp[ 0 ] + 2 ) >> 2 ) ) (‑161)  
     yRef = Clip3( 0, PicHeightInSamplesL – 1, yP + ( ( nPSH – 1 ) >> 1 ) + ( ( mvDisp[ 1 ] + 2 ) >> 2 )) (‑162)

Let refCU be the coding unit that covers the luma location ( xRef, yRef ) in the view component with ViewIdx equal to refViewIdx.

In software, the refCU is the coding unit in refViewPic which is derived as follows:

* If slice\_type is equal to B and refview\_from\_l0\_flag is equal to 0, the variable refViewPic specifies the picture that contains the refCU as specified by RefPicList1[ refview\_ref\_idx ].
* Otherwise (slice\_type is equal to B and refview\_from\_l0\_flag is equal to 1 or slice\_type is equal to P), the variable refViewPic specifies the picture that contains the refCU as specified by RefPicList0[ refview\_ref\_idx ].

[Ed. (CY): What is implemented in the software has no impact on the final results in common test condition, so it is preferred to have the current text and software changes may be desirable. ]

When the variable PredMode for the coding unit refCU is equal to MODE\_SKIP or MODE\_INTER, the following ordered steps apply, for Y in the range of X to (1 – X), inclusive:

* 1. The variable refPredFlagLY is set equal to the prediction utilization flag predFlagLY of the prediction unit refPU.
  2. The variable refRefIdxLY, is set equal to the reference index refIdxLY of the prediction unit refPU.
  3. The variable refMvLY is set equal to the motion vector mvLY of the prediction unit refPU.
  4. The variable refRefPicListLY, is set equal to the reference picture list RefPicListLY of the prediction unit refCU.
  5. When refPredFlagLY is equal to 1, the following applies for each i from ( mergeFlag ? 0 : refIdxLX ) to ( mergeFlag ? num\_ref\_idx\_lX\_active\_minus1 : refIdxLX), inclusive:
     + When availableFlagLXInterView is equal to 0, and the picture order count of the picture refRefPicListLY[ refRefIdxLY ] is equal to the picture order count of the picture RefPicListLX[ i ], the flag availableFlagLXInterView is set equal to 1 and the following applies.
       - The motion vector mvLXInterView is derived by:
         1. mvLXInterView[ 0 ] = refMvLY[ 0 ] (‑)  
            mvLXInterView[ 1 ] = refMvLY[ 1 ] (‑)
       - When mergeFlag is equal to 1, the reference index refIdxLX is derived by:
         1. refIdxLX = i (‑)

1. * + - 1. Inter-view residual prediction process

The process is only invoked if res\_pred\_flag is equal to 1.

Inputs to this process are:

* a luma location ( xC, yC ) specifying the top-left sample of the current luma coding block relative to the top left luma sample of the current picture,
* a luma location ( xP, yP ) of the top-left luma sample of the current prediction unit relative to the top-left luma sample of the current picture,
* a variable nCS specifying the size of the current luma coding block,
* variables nPSW and nPSH specifying the width and the height, respectively, of the current prediction unit,prediction list utilization flags, predFlagL0 and predFlagL1,
* a (nPSW)x(nPSH) array predSamplesL of luma prediction samples,
* two (nPSW / 2)x(nPSH / 2) arrays predSamplesCb and predSamplesCr of chroma prediction samples.

Output of this process are:

* a modified version of the (nPSW)x(nPSH) array predSamplesL,
* a modified versions of the (nPSW / 2)x(nPSH / 2) arrays predSamplesCb and predSamplesCr.

The derivation process for a disparity vector as specified in subclause is invoked with the luma locations ( xC, yC ) and ( xP, yP ), the coding block size nCS, the variables nPSW and nPSH, the partition index partIdx and the variable deriveFromDepthFlag being equal to 0, as the inputs and the outputs are the view order index refViewIdx, the flag availableDV and the disparity vector mvDisp.[Ed. (GT) partIdx is missing as input to this subclause.].

In software, the refViewIdx is set equal to ViewId(refViewPic), the refViewPic is derived as follows:

* If slice\_type is equal to B and refview\_from\_l0\_flag is equal to 0, the variable refViewPic specifies the picture that contains the refCU as specified by RefPicList1[ refview\_ref\_idx ].
* Otherwise (slice\_type is equal to B and refview\_from\_l0\_flag is equal to 1 or slice\_type is equal to P), the variable refViewPic specifies the picture that contains the refCU as specified by RefPicList0[ refview\_ref\_idx ].

Let refResSamplesL be the (PicWidthInSamplesL)x(PicHeightInSamplesL) array of luma residual samples ResSamplesL: of the view component with ViewIdx equal to refViewIdx. Let refResSamplesCb and refResSamplesCr be the (PicWidthInSamplesL / 2)x(PicHeightInSamplesL / 2) arrays of Cb and Cr residual samples ResSamplesCb and ResSamplesCr, respectively, for inter-coded coding units for the view component with ViewIdx equal to refViewIdx.

When the flag availableDV is equal to 0 the whole decoding process of this sub-clause terminates.

For y proceeding over the values 0..(nPSH – 1) and x proceeding over the values 0..(nPSW – 1), the following ordered steps apply.

* 1. The variables xR0, xR1, yR0, yR1 and w0, w1, w2, and w3 are derived by
     + 1. xR0 = Clip3( 0, PicWidthInSamplesL – 1, xP + x + (mvDisp[ 0 ] >> 2 ) ) (‑)  
          xR1 = Clip3( 0, PicWidthInSamplesL – 1, xP + x + (mvDisp[ 0 ] >> 2 ) + 1 ) (‑)  
          yR0 = Clip3( 0, PicHeightInSamplesL – 1, yP + y + (mvDisp[ 1 ] >> 2 ) ) (‑)  
          yR1 = Clip3( 0, PicHeightInSamplesL – 1, yP + y + (mvDisp[ 1 ] >> 2 ) + 1 ) (‑)  
          w0 = 4 – mvDisp[ 0 ] + ( ( mvDisp[ 0 ] >> 2 ) << 2 ) (‑)  
          w1 = mvDisp[ 0 ] − ( ( mvDisp[ 0 ] >> 2 ) << 2 ) (‑)  
          w2 = 4 – mvDisp[ 1 ] + ( ( mvDisp[ 1 ] >> 2 ) << 2 ) (‑)  
          w3 = mvDisp[ 1 ] − ( ( mvDisp[ 1 ] >> 2 ) << 2 ) (‑)
  2. The sample predSamplesL[ x ][ y ] is modified by
     + 1. deltaL1 = ( w0 \* refResSamplesL[ xR0 ][ yR0 ] + w1 \* refResSamplesL[ xR1 ][ yR0 ] + 4 ) >> 3 (‑)  
          deltaL2 = ( w0 \* refResSamplesL[ xR0 ][ yR1 ] + w1 \* refResSamplesL[ xR1 ][ yR1 ] + 4 ) >> 3 (‑)  
          deltaL = ( w2 \* deltaL1 + w3 \* deltaL2 + 4 ) >> 3 (‑)  
          predSamplesL[ x ][ y ] = predSamplesL[ x ][ y ] + deltaL (‑)

For y proceeding over the values 0..(nPSH / 2 – 1) and x proceeding over the values 0..(nPSW / 2 – 1), the following ordered steps are specified:

* 1. The variables xR0, xR1, yR0, yR1 and w0, w1, w2, and w3 are derived by
     + 1. xR0 = Clip3( 0, PicWidthInSamplesL / 2 – 1, xP / 2 + x + (mvDisp[ 0 ] >> 3 ) ) (‑)  
          xR1 = Clip3( 0, PicWidthInSamplesL / 2 – 1, xP / 2 + x + (mvDisp[ 0 ] >> 3 ) + 1 ) (‑)  
          yR0 = Clip3( 0, PicHeightInSamplesL / 2 – 1, yP / 2 + y + (mvDisp[ 1 ] >> 3 ) ) (‑)  
          yR1 = Clip3( 0, PicHeightInSamplesL / 2 – 1, yP / 2 + y + (mvDisp[ 1 ] >> 3 ) + 1 ) (‑)  
          w0 = 8 – mvDisp[ 0 ] + ( (mvDisp[ 0 ] >> 3 ) << 3 ) (‑)  
          w1 = mvDisp[ 0 ] − ( (mvDisp[ 0 ] >> 3 ) << 3 ) (‑)  
          w2 = 8 – mvDisp[ 1 ] + ( (mvDisp[ 1 ] >> 3 ) << 3 ) (‑)  
          w3 = mvDisp[ 1 ] − ( (mvDisp[ 1 ] >> 3 ) << 3 ) (‑)
  2. The sample predSamplesCb[ x ][ y ] is modified by
     + 1. deltaCb1 = ( w0 \* refResSamplesCb[ xR0 ][ yR0 ] + w1 \* refResSamplesCb[ xR1 ][ yR0 ] + 8 ) >> 4 (‑)  
          deltaCb2 = ( w0 \* refResSamplesCb[ xR0 ][ yR1 ] + w1 \* refResSamplesCb[ xR1 ][ yR1 ] + 8 ) >> 4 (‑)  
          deltaCb  = ( w2 \* deltaCb1 + w3 \* deltaCb2 + 8 ) >> 4 (‑)  
          predSamplesCb[ x ][ y ] = predSamplesCb[ x ][ y ] + deltaCb  (‑)
  3. The sample predSamplesCr[ x ][ y ] is modified by
     + 1. deltaCr1 = ( w0 \* refResSamplesCr[ xR0 ][ yR0 ] + w1 \* refResSamplesCr[ xR1 ][ yR0 ] + 8 ) >> 4 (‑)  
          deltaCr2 = ( w0 \* refResSamplesCr[ xR0 ][ yR1 ] + w1 \* refResSamplesCr[ xR1 ][ yR1 ] + 8 ) >> 4 (‑)  
          deltaCr = ( w2 \* deltaCr1+ w3 \* deltaCr2) + 8 ) >> 4 (‑)  
          predSamplesCr[ x ][ y ] = predSamplesCr[ x ][ y ] + deltaCr  (‑)