## Fixes for palette\_share\_flag and palette\_escape\_val\_present\_flag

#### Palette mode syntax

|  |  |
| --- | --- |
| palette\_coding( x0, y0, nCbS ) { | Descriptor |
| **palette\_transpose\_flag** | ae(v) |
| if ( previousPaletteSize > 0 ) |  |
| **palette\_share\_flag**[ x0 ][ y0 ] | ae(v) |
| if( palette\_share\_flag[ x0 ][ y0 ] ) { |  |
| palette\_size = previousPaletteSize |  |
| for( n = 0; n < palette\_size; n++ ) |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| palette\_entries[ cIdx ][ n ] = previousPaletteEntries[ cIdx ][ n ] |  |
| } else { |  |
| numPredPreviousPalette = 0 |  |
| for( i = 0; i < previousPaletteStuffingSize; i++ ) |  |
| previous\_palette\_entry\_flag[ i ] = 0 |  |
| palette\_last\_group = 0 |  |
| for( i = 0; i < previousPaletteStuffingSize && !palette\_last\_group &&  numPredPreviousPalette < max\_palette\_size; i++ ) { [Ed. YY: max\_palette\_size probably needs to be signalled] |  |
| lastPossibleGroupFlag = ( i + 4 >= previousPaletteStuffingSize ) |  |
| lastIdx = min( i + 4, previousPaletteStuffingSize ) − 1 |  |
| if( i > 3 && !lastPossibleGroupFlag ) |  |
| **palette**\_**all\_zeros\_in\_group** | ae(v) |
| else |  |
| palette\_all\_zeros\_in\_group = 0 |  |
| if( palette\_all\_zeros\_in\_group ) |  |
| i += 4 |  |
| else { |  |
| numOnesInGroup = 0 |  |
| for( idx = i; idx <= lastIdx && numPredPreviousPalette < max\_palette\_size;  idx++ ) { |  |
| if ( idx = = lastIdx && numOnesInGroup = = 0 ) |  |
| previous\_palette\_entry\_flag[ idx ] = 1 |  |
| else |  |
| **previous\_palette\_entry\_flag**[ idx ] | ae(v) |
| if ( previous\_palette\_entry\_flag[ idx ] ) { |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| palette\_entries[ cIdx ][ numPredPreviousPalette ] =   previousPaletteEntries[ cIdx ][ idx ] |  |
| numPredPreviousPalette++ |  |
| numOnesInGroup++ |  |
| } |  |
| } |  |
| if( !palette\_all\_zeros\_in\_group &&   !lastPossibleGroupFlag && numPredPreviousPalette < max\_palette\_size ) |  |
| **palette**\_**last\_group** | ae(v) |
| } |  |
| } |  |
| if( numPredPreviousPalette < max\_palette\_size) |  |
| **num\_signalled\_palette\_entries** | ae(v) |
| for( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| for( i = 0; i < num\_signalled\_palette\_entries; i++ ) |  |
| **palette\_entries**[ cIdx ][ numPredPreviousPalette + i ] | ae(v) |
| palette\_size = numPredPreviousPalette + num\_signalled\_palette\_entries |  |
| } |  |
| if ( palette\_size > 0 ) |  |
| **palette\_escape\_val\_present\_flag** | ae(v) |
| if( palette\_escape\_val\_present\_flag ) |  |
| indexMax = palette\_size |  |
| else |  |
| indexMax = palette\_size − 1 |  |
| scanPos = 0 |  |
| while( scanPos < nCbS \* nCbS ) { |  |
| xC = x0 + travScan[ scanPos ][ 0 ] |  |
| yC = y0 + travScan[ scanPos ][ 1 ] |  |
| if( scanPos > 0) { |  |
| xC\_prev = x0 + travScan[ scanPos − 1 ][ 0 ] |  |
| yC\_prev = y0 + travScan[ scanPos − 1 ][ 1 ] |  |
| } |  |
| if( scanPos > = nCbS && palette\_mode[xC\_prev][yC\_prev] ! = COPY\_ABOVE ) |  |
| **palette\_mode**[ xC ][ yC ] | ae(v) |
| if( palette\_mode[ xC ][ yC ] ! = COPY\_ABOVE ) { |  |
| adjustedIndexMax = indexMax |  |
| adjustedRefIndex = indexMax + 1 |  |
| } |  |
| if( scanPos > 0 && palette\_mode[xC\_prev][yC\_prev] ! = ESCAPE ) { |  |
| if( palette\_mode[xC\_prev][yC\_prev] = = INDEX ) { |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex = paletteMap[ xC\_prev ][ yC\_prev ] |  |
| } |  |
| if( scanPos > = nCbS && palette\_mode[ xC\_prev ][ yC\_prev ] = = COPY\_ABOVE   && palette\_mode[ xC ][ yC − 1 ] ! = ESCAPE ) { |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex = paletteMap[ xC ][ yC − 1 ] |  |
| } |  |
| } |  |
| if(palette\_mode[ xC ][ yC ] ! = COPY\_ABOVE ) { |  |
| if( adjustedIndexMax > 0 ) |  |
| **palette\_index** | ae(v) |
| if( palette\_index > = adjustedRefIndex ) |  |
| palette\_index++ |  |
| if( palette\_index = = palette\_size ) { |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| **palette\_escape\_val** | ae(v) |
| paletteEscapeVal[ cIdx ][ xC ][ yC ] = palette\_escape\_val |  |
| } |  |
| palette\_mode[ xC ][ yC ] = ESCAPE |  |
| scanPos++ |  |
| } |  |
| } |  |
| if( palette\_mode[xC][yC] ! = ESCAPE ) { |  |
| **palette\_run** | ae(v) |
| runPos = 0 |  |
| runMode = palette\_mode[ xC ][ yC ] |  |
| while ( runPos < = palette\_run ) { |  |
| xC = x0 + travScan[ scanPos ][ 0 ] |  |
| yC = y0 + travScan[ scanPos ][ 1 ] |  |
| if( palette\_mode[ xC ][ yC ] = = INDEX ) { |  |
| palette\_mode[ xC ][ yC ] = INDEX |  |
| paletteMap[ xC ][ yC ] = palette\_index |  |
| } else { |  |
| palette\_mode[ xC ][ yC ] = COPY\_ABOVE |  |
| paletteMap[ xC ][ yC ] = paletteMap[ xC ][ y − 1 ] |  |
| } |  |
| runPos++ |  |
| scanPos++ |  |
| } |  |
| } |  |
| } |  |
| previousPaletteSize = palette\_size |  |
| current\_size = palette\_size |  |
| for( i = 0; i < palette\_size; i++ ) |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| tempPaletteEntries[ cIdx ][ i ] = palette\_entries[ cIdx ][ i ] |  |
| for( i = 0; i < previousPaletteStuffingSize && current\_size < max\_palette\_predictor\_size;   i++ ) |  |
| if( previous\_palette\_entry\_flag[ i ] = = 0 ) { |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| tempPaletteEntries[ cIdx ][ current\_size ] = previousPaletteEntries[ cIdx ][ i ] |  |
| current\_size++ |  |
| } |  |
| previousPaletteStuffingSize = current\_size |  |
| previousPaletteEntries = tempPaletteEntries |  |
| } |  |

**7.4.9.6 Palette mode semantics**

**palette\_share\_flag**[ x0 ][ y0 ] equal to 1 specifies that the palette size is equal to previous previousPaletteEntries and the entire palette entries are the same as the previous palette entries.

When palette\_share\_flag[ x0 ][ y0 ] is not present, it is inferred to be equal to 0.

…

**escape\_val\_present\_flag** equal to 1 specifies that the current block may contain some escape coded samples. escape\_val\_present\_flag equal to 0 specifies that there are no escape coded samples in the current block.

When escape\_val\_present\_flagis not present, it is inferred to be equal to 1.

## Inferred PCM mode

|  |  |
| --- | --- |
| palette\_coding( x0, y0, nCbS ) { | Descriptor |
| **palette\_transpose\_flag** | ae(v) |
| **palette\_share\_flag**[ x0 ][ y0 ] | ae(v) |
| if( palette\_share\_flag[ x0 ][ y0 ] ) { |  |
| palette\_size = previousPaletteSize |  |
| for( n = 0; n < palette\_size; n++ ) |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| palette\_entries[ cIdx ][ n ] = previousPaletteEntries[ cIdx ][ n ] |  |
| } else { |  |
| numPredPreviousPalette = 0 |  |
| for( i = 0; i < previousPaletteStuffingSize; i++ ) |  |
| previous\_palette\_entry\_flag[ i ] = 0 |  |
| palette\_last\_group = 0 |  |
| for( i = 0; i < previousPaletteStuffingSize && !palette\_last\_group &&  numPredPreviousPalette < max\_palette\_size; i++ ) { [Ed. YY: max\_palette\_size probably needs to be signalled] |  |
| lastPossibleGroupFlag = ( i + 4 >= previousPaletteStuffingSize ) |  |
| lastIdx = min( i + 4, previousPaletteStuffingSize ) − 1 |  |
| if( i > 3 && !lastPossibleGroupFlag ) |  |
| **palette**\_**all\_zeros\_in\_group** | ae(v) |
| else |  |
| palette\_all\_zeros\_in\_group = 0 |  |
| if( palette\_all\_zeros\_in\_group ) |  |
| i += 4 |  |
| else { |  |
| numOnesInGroup = 0 |  |
| for( idx = i; idx <= lastIdx && numPredPreviousPalette < max\_palette\_size;  idx++ ) { |  |
| if ( idx = = lastIdx && numOnesInGroup = = 0 ) |  |
| previous\_palette\_entry\_flag[ idx ] = 1 |  |
| else |  |
| **previous\_palette\_entry\_flag**[ idx ] | ae(v) |
| if ( previous\_palette\_entry\_flag[ idx ] ) { |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| palette\_entries[ cIdx ][ numPredPreviousPalette ] =   previousPaletteEntries[ cIdx ][ idx ] |  |
| numPredPreviousPalette++ |  |
| numOnesInGroup++ |  |
| } |  |
| } |  |
| if( !palette\_all\_zeros\_in\_group &&   !lastPossibleGroupFlag && numPredPreviousPalette < max\_palette\_size ) |  |
| **palette**\_**last\_group** | ae(v) |
| } |  |
| } |  |
| if( numPredPreviousPalette < max\_palette\_size) |  |
| **num\_signalled\_palette\_entries** | ae(v) |
| for( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| for( i = 0; i < num\_signalled\_palette\_entries; i++ ) |  |
| **palette\_entries**[ cIdx ][ numPredPreviousPalette + i ] | ae(v) |
| palette\_size = numPredPreviousPalette + num\_signalled\_palette\_entries |  |
| } |  |
| if( palette\_size == 0 ) |  |
| { |  |
| while( !byte\_aligned( ) ) |  |
| **plt\_alignment\_zero\_bit** |  |
| pcm\_sample( x0, y0, log2CbSize ) |  |
| } |  |
| else { |  |
| **palette\_escape\_val\_present\_flag** | ae(v) |
| if( palette\_escape\_val\_present\_flag ) |  |
| indexMax = palette\_size |  |
| else |  |
| indexMax = palette\_size − 1 |  |
| scanPos = 0 |  |
| while( scanPos < nCbS \* nCbS ) { |  |
| xC = x0 + travScan[ scanPos ][ 0 ] |  |
| yC = y0 + travScan[ scanPos ][ 1 ] |  |
| if( scanPos > 0) { |  |
| xC\_prev = x0 + travScan[ scanPos − 1 ][ 0 ] |  |
| yC\_prev = y0 + travScan[ scanPos − 1 ][ 1 ] |  |
| } |  |
| if( scanPos > = nCbS && palette\_mode[xC\_prev][yC\_prev] ! = COPY\_ABOVE ) |  |
| **palette\_mode**[ xC ][ yC ] | ae(v) |
| if( palette\_mode[ xC ][ yC ] ! = COPY\_ABOVE ) { |  |
| adjustedIndexMax = indexMax |  |
| adjustedRefIndex = indexMax + 1 |  |
| } |  |
| if( scanPos > 0 && palette\_mode[xC\_prev][yC\_prev] ! = ESCAPE ) { |  |
| if( palette\_mode[xC\_prev][yC\_prev] = = INDEX ) { |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex = paletteMap[ xC\_prev ][ yC\_prev ] |  |
| } |  |
| if( scanPos > = nCbS && palette\_mode[ xC\_prev ][ yC\_prev ] = = COPY\_ABOVE   && palette\_mode[ xC ][ yC − 1 ] ! = ESCAPE ) { |  |
| adjustedIndexMax − = 1 |  |
| adjustedRefIndex = paletteMap[ xC ][ yC − 1 ] |  |
| } |  |
| } |  |
| if(palette\_mode[ xC ][ yC ] ! = COPY\_ABOVE ) { |  |
| if( adjustedIndexMax > 0 ) |  |
| **palette\_index** | ae(v) |
| if( palette\_index > = adjustedRefIndex ) |  |
| palette\_index++ |  |
| if( palette\_index = = palette\_size ) { |  |
| for( cIdx = 0; cIdx < 3; cIdx++ ) { |  |
| **palette\_escape\_val** | ae(v) |
| paletteEscapeVal[ cIdx ][ xC ][ yC ] = palette\_escape\_val |  |
| } |  |
| palette\_mode[ xC ][ yC ] = ESCAPE |  |
| scanPos++ |  |
| } |  |
| } |  |
| if( palette\_mode[xC][yC] ! = ESCAPE ) { |  |
| **palette\_run** | ae(v) |
| runPos = 0 |  |
| runMode = palette\_mode[ xC ][ yC ] |  |
| while ( runPos < = palette\_run ) { |  |
| xC = x0 + travScan[ scanPos ][ 0 ] |  |
| yC = y0 + travScan[ scanPos ][ 1 ] |  |
| if( palette\_mode[ xC ][ yC ] = = INDEX ) { |  |
| palette\_mode[ xC ][ yC ] = INDEX |  |
| paletteMap[ xC ][ yC ] = palette\_index |  |
| } else { |  |
| palette\_mode[ xC ][ yC ] = COPY\_ABOVE |  |
| paletteMap[ xC ][ yC ] = paletteMap[ xC ][ y − 1 ] |  |
| } |  |
| runPos++ |  |
| scanPos++ |  |
| } |  |
| } |  |
| } |  |
| } |  |
| previousPaletteSize = palette\_size |  |
| current\_size = palette\_size |  |
| for( i = 0; i < palette\_size; i++ ) |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| tempPaletteEntries[ cIdx ][ i ] = palette\_entries[ cIdx ][ i ] |  |
| for( i = 0; i < previousPaletteStuffingSize && current\_size < max\_palette\_predictor\_size;   i++ ) |  |
| if( previous\_palette\_entry\_flag[ i ] = = 0 ) { |  |
| for ( cIdx = 0; cIdx < 3; cIdx++ ) |  |
| tempPaletteEntries[ cIdx ][ current\_size ] = previousPaletteEntries[ cIdx ][ i ] |  |
| current\_size++ |  |
| } |  |
| previousPaletteStuffingSize = current\_size |  |
| previousPaletteEntries = tempPaletteEntries |  |
| } |  |

**7.4.9.6 Palette mode semantics**

**escape\_val\_present\_flag** equal to 1 specifies that the current block may contain some escape coded samples. escape\_val\_present\_flag equal to 0 specifies that there are no escape coded samples in the current block.

When escape\_val\_present\_flagis not present, it is inferred to be equal to 1.

**plt\_alignment\_zero\_bit** is a bit equal to 0.

## 