|  |  |
| --- | --- |
| **Joint Collaborative Team on 3D Video Coding Extensions**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  7th Meeting: San Jose, USA, 11 Jan. – 17 Jan. 2013 | Document: JCT3V-G0106 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **CE3: Results on Depth-based Block Partitioning (DBBP)** | | |
| *Status:* | Input Document | | |
| *Purpose:* | Proposal | | |
| *Author(s) or Contact(s):* | Fabian Jäger Institut für Nachrichtentechnik RWTH Aachen University  Jacek Konieczny, Giovanni Cordara Huawei Technologies Munich Office, European Research Center | Tel: Email:  Tel: Email: | +49 (0) 241 80 27678 [jaeger@ient.rwth-aachen.de](mailto:jaeger@ient.rwth-aachen.de)  +49 (0) 89 158834 4334 [jacek.konieczny@huawei.com](mailto:jacek.konieczny@huawei.com) |
| *Source:* | RWTH Aachen University, Huawei Technologies | | |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Abstract

In the proposed method, an arbitrarily shaped block partitioning for the collocated texture block is derived based on a binary segmentation mask computed from the collocated (virtual) depth map. Each of the two partitions (resembling foreground and background) is motion compensated and afterwards merged based on the depth-based segmentation mask.  
Simulation results show the benefit of the proposed tools with 0.2% BD-rate savings on the video component (w.r.t. total bitrate).

# Block Partitioning Algorithm

The proposed algorithm for using depth information for block partitioning in texture views consists of three sequential steps, which will be described in more detail in the following section.

## Depth Segmentation

In an initial step the collocated depth block of the current coded tree block (CTB) of the texture component is segmented into two arbitrarily shaped segments. As the depth component is coded after the corresponding texture view in the current CTC, a virtual depth map is derived from the base view’s reconstructed depth and shifted by a disparity vector, which is itself derived from the neighboring blocks (by means of DoNBDV).

The segmentation of the (virtual) depth map is performed based on a very simple thresholding mechanism where the threshold is computed from the mean depth value.

Here, defines the width/height of the current texture block and resembles the already coded, corresponding depth map of the reference view’s texture frame.

Afterwards, a binary segmentation mask is generated based on as follows.

The resulting binary segmentation mask defines the shape of the partitioning of the texture block. While motion or disparity compensation in a modern video coder (e.g. in HEVC) is performed on a block-basis, an arbitrarily shaped block partitioning typically requires pixel-based compensation. This concept is applied in pixel-based view-synthesis prediction (VSP), which warps each pixel position based on the corresponding depth value to the position of the particular reference frame. By this fine-grained compensation approach higher order deformations can be approximated. To achieve this amount of precision for the prediction stage of a video coder, pixel-based VSP introduces very high complexity compared to conventional block-based motion/disparity compensation. This is mainly due to the irregular access to the reference buffer and the pixel-wise conversion between depth and disparity.

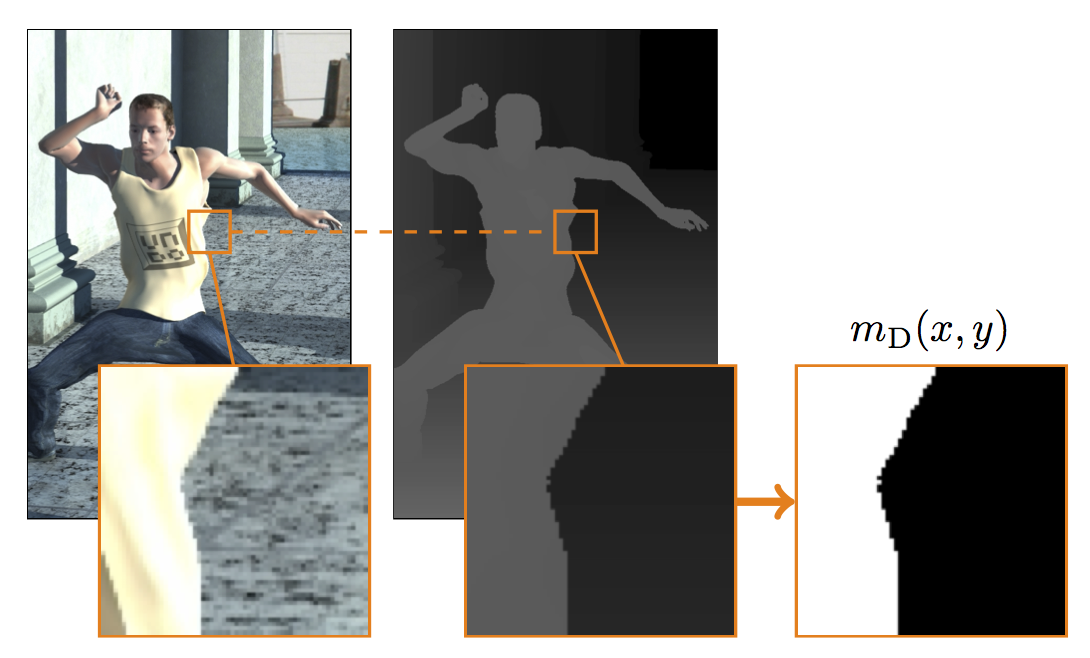


Figure 1: Cropped frame from *Undo\_Dancer* test sequence with magnified component blocks of an example coding unit. The collocated depth block segments the block into foreground and background, which is defined by a binary mask .

To overcome this issue of VSP, the proposed depth-based block-partitioning (DBBP) scheme still uses block-based compensation in the prediction stage, as it is described in the following subsection.

## Block-based Compensation

In the proposed DBBP scheme, the actual motion or disparity compensation is performed on a partitioning, which means that the full CTB is shifted by the coded vector information. This full-size motion/disparity compensation is performed twice, once for each segment, and results in two prediction signals and .

Consequently, two sets of vector information need to be coded for a DBBP block. The assumption behind this approach is that a texture block is typically segmented into foreground and background based on the collocated depth block. These two depth layers can then be compensated independently by their own sets of motion or disparity vectors.

## Merging of Prediction Signals

After having generated two full-size prediction signals and for a DBBP-coded block, the segmentation mask is used to merge these into the final prediction signal for the current texture CTB.

By merging the two prediction signals, shape information from the depth map allows to independently compensate foreground and background objects in the same texture block. At the same time, DBBP does not require pixel-wise motion/disparity compensation. Memory access to the reference buffers is always regular (block-based) for DBBP-coded blocks in contrast to approaches like VSP. Moreover, DBBP always uses full-size blocks for compensation. This is preferable with respect to complexity, because of the higher probability of finding the data in the memory cache.

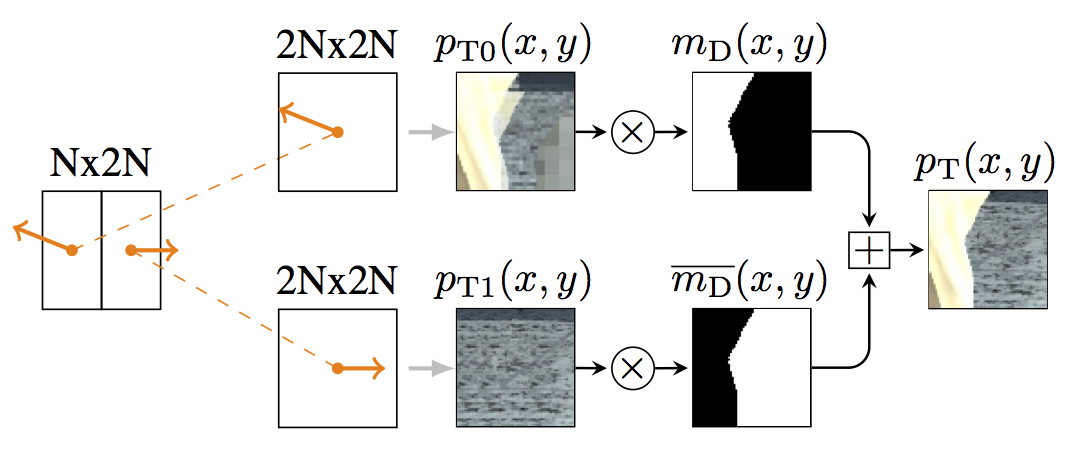


Figure 2: DBBP merging process: For each of the two decoded motion parameters a motion compensation is performed. The resulting prediction signals and are combined using the DBBP mask .

# Coding of Partitioning Information

As described in the previous section, DBBP requires coding two sets of motion information, one for each segment. A modern video coder, such as HEVC, allows using rectangular, non-square partitioning modes within a coding tree unit (CTU) for finer-grained motion compensation. For each of the partitions in a CTU a separate set of motion information is coded. This coding scheme is reused in the proposed depth-based block partitioning.

After the encoder has derived the optimal motion/disparity information for each DBBP segment, this information is mapped into one of the available rectangular, non-square partitioning modes of HEVC. This includes asymmetric motion partitioning modes [1], which were introduced for HEVC. The mapping of the binary segmentation mask to one of the 6 available two-segment partitioning modes is performed by a correlation analysis. For each of the available partitioning modes ) 2 binary masks and are generated, where is the negation of . To find the best matching partitioning mode for the current depth-based segmentation mask , the following algorithm is performed:

The Boolean variable defines whether the derived segmentation mask needs to be inverted or not. This might be necessary in some cases where the indexing of the conventional partitioning schemes is complementary to the indexing in the segmentation mask. In the conventional partitioning modes, index 0 always defines the partition in the top-left corner of the current block, while the same index in the segmentation mask defines the segment with the lower depth values (background objects). To align the positioning of the corresponding sets of motion information between and , the indexing in is inverted, if is set.

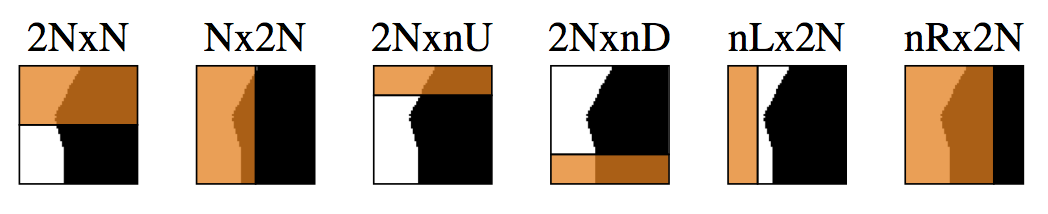


Figure 4: Superposition of conventional partitioning modes (gray) and depth-based block partitioning (black/white). The best matching partitioning () mode is selected for storing motion information.

After having found the best matching conventional partitioning mode, motion information is stored and coded according to this optimal mode . Succeeding coding units (CUs) can access the already coded motion information conventionally when deriving motion vector candidates for advanced motion vector prediction (AMVP) or motion vector merging.

A single flag is added to the coding syntax to signal to the decoder that a block uses DBBP for prediction. An obvious choice would be to send this flag for all of the conventional partitioning modes. But this approach would result in unwanted coding overhead for blocks that do not use DBBP. Therefore, the partitioning mode for DBBP-coded blocks is always set to before coding the partitioning mode. Afterwards, only for -partitioned blocks, the DBBP flag is coded in the bitstream. For all other partitioning modes, the DBBP flag is not coded. At the decoder side, the true (conventional) partitioning is derived after having decoded the DBBP flag. All further processing steps at the decoder remain the same as in the HEVC base specification.



By this modified coding scheme the number of DBBP flags for CUs not using DBBP can be reduced significantly.

# Simulation Results

The proposed method was implemented into HTM 9.0r1 and all simulations follow the common test conditions.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | video 0 | video 1 | video 2 | video PSNR / video bitrate | video PSNR / total bitrate | synth PSNR / total bitrate | enc time | dec time |
| Balloons | 0,0% | -0,4% | -0,2% | -0,1% | -0,1% | -0,1% | 103,2% | 100,4% |
| Kendo | 0,0% | -0,5% | -0,4% | -0,2% | -0,1% | -0,1% | 102,9% | 99,2% |
| Newspaper\_CC | 0,0% | -0,3% | -0,2% | -0,1% | -0,1% | 0,0% | 100,4% | 101,8% |
| GT\_Fly | 0,0% | -0,5% | -0,6% | -0,2% | -0,2% | -0,1% | 103,6% | 101,4% |
| Poznan\_Hall2 | 0,0% | -0,2% | -0,1% | -0,2% | -0,2% | 0,0% | 105,2% | 100,0% |
| Poznan\_Street | 0,0% | -0,6% | -0,8% | -0,2% | -0,2% | -0,2% | 104,6% | 99,7% |
| Undo\_Dancer | 0,0% | -1,1% | -1,1% | -0,3% | -0,3% | -0,2% | 103,1% | 102,3% |
| Shark | 0,0% | -0,6% | -0,7% | -0,2% | -0,2% | -0,1% | 103,3% | 101,7% |
| 1024x768 | 0,0% | -0,4% | -0,2% | -0,1% | -0,1% | -0,1% | 102,2% | 100,5% |
| 1920x1088 | 0,0% | -0,6% | -0,7% | -0,2% | -0,2% | -0,1% | 104,0% | 101,0% |
| **average** | **0,0%** | **-0,5%** | **-0,5%** | **-0,2%** | **-0,2%** | **-0,1%** | **103,3%** | **100,8%** |

# Cross Check

The cross check of the proposed DBBP coding scheme was performed by LG Electronics. They investigated the required source code modifications and ran the simulations for verification of the presented results.

In their investigation they did not find any problems with the source code. Their simulation results perfectly match with those presented in this document.

# Conclusion

In this contribution a novel partitioning scheme for texture views is proposed that derives a fine-grained segmentation mask from previously coded depth information. By segmenting the texture block into foreground and background, both depth layers can be motion compensated independently before they are merged to form the final prediction signal.

The presented simulations results clearly show that there is a strong potential in this approach. Consequently, adoption of the proposed coding scheme to the working draft of the 3D extension of HEVC is recommended.

# Working Draft Changes

The proposed working draft modifications are as follows.

**H.7.3.2.1.2 Video parameter set extension 2 syntax**

|  |  |
| --- | --- |
| vps\_extension2( ) { | **Descriptor** |
| … |  |
| if ( !VpsDepthFlag[ layerId ] ) { |  |
| **iv\_res\_pred\_flag**[ layerId ] | u(1) |
| **depth\_refinement\_flag**[ layerId ] | u(1) |
| **view\_synthesis\_pred\_flag**[ layerId ] | u(1) |
| **use\_dbbp\_flag**[ layerId ] | u(1) |
| } |  |
| … |  |
| } |  |

**H.7.3.8.5 Coding unit syntax**

|  |  |
| --- | --- |
| coding\_unit( x0, y0, log2CbSize, ctDepth ) { | **Descriptor** |
| … |  |
| if( ( CuPredMode[ x0 ][ y0 ] != MODE\_INTRA | | log2CbSize ==Log2MinCbSize ) &&!predPartModeFlag ) |  |
| **part\_mode** | ae(v) |
| if( use\_dbbp\_flag && PartMode == PART\_2NxN ) { |  |
| **dbbp\_flag** | u(1) |
| if( dbbp\_flag ) |  |
| PartMode = virtual\_part\_mode(x0, y0, log2CbSize, ctDepth ) |  |
| } |  |
| … |  |
| } |  |

**H.7.4.3.1.2 Video parameter set extension 2 semantics**

**use\_dbbp\_flag**[ layerId ]equal to 1 specifies that depth-based block partitioning is allowed for the layer with nuh\_layer\_id equal to layerId. use\_dbbp\_flag equal to 0 specifies that depth-based block partitioning is not allowed in this layer.

**H.7.4.9.5 Coding unit semantics**

**dbbp\_flag** equal to 1 specifies that depth-based block partitioning is used for the current coding unit.dbbp\_flag equal to 0 specifies that depth-based block partitioning is not used for the current coding unit. When not present, dbbp\_flag is inferred to be equal to 0.

virtual\_part\_mode(x0, y0, log2CbSize, ctDepth ) derives the actual partitioning mode for the current coding unit that uses depth-based block partitioning (has dbbp\_flag equal to 1). The corresponding derivation process of PartMode is specified in section H.8.5.5.3

**H.8.5** **Decoding process for coding units coded in inter prediction mode**

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 variable log2CbSize specifying the size of the current coding block.

Output of this process is a modified reconstructed picture before deblocking filtering.

The derivation process for quantization parameters as specified in subclause 8.6.1 is invoked with the luma location ( xC, yC ) as input.

The variable nCSL is set equal to 1 << log2CbSize and the variable nCSC is set equal to ( 1 << log2CbSize ) >> 1.

Decoding process for coding units coded in inter prediction mode consists of following ordered steps:

1. When iv\_mv\_pred\_flag[ nuh\_layer\_id ] is equal to 1, or iv\_res\_pred\_flag[ nuh\_layer\_id ] is equal to 1 or view\_synthesis\_pred\_flag[ nuh\_layer\_id ] is equal to 1, the derivation process for disparity vectors as specified in subclause H.8.5.3.1 is invoked with the luma locations ( xC , yC ), the coding block size nCSL as the inputs.
2. When dbbp\_flag is equal to 1, the inter prediction process, the decoding process for the residual signal and the derivation of the reconstructed samples described in the point following point 3 are applied for each of the two DBBP partitions of the current coding block with PartMode temporarily set to PART\_2Nx2N and all the inter prediction data set according to the prediction data of the DBBP partition that is currently being processed.
3. The inter prediction process as specified in subclause H.8.5.1 is invoked with the luma location ( xC, yC ), and the luma coding block size log2CbSize as the inputs and the outputs are three arrays predSamplesL, predSamplesCb, predSamplesCr.
4. When dbbp\_flag is equal to 1, results of the inter prediction process determined for each of the two DBBP partitions (arrays predSamplesL, predSamplesCb, predSamplesCr) are merged by the DBBP merging process specified in subclause H.8.5.5.4.
5. The decoding process for the residual signal of coding units coded in inter prediction mode specified in subclause H.8.5.3 is invoked with the luma location ( xC, yC ), luma coding block size log2CbSize as the inputs and the outputs are three arrays resSamplesL, resSamplesCb, resSamplesCr.
6. The reconstructed samples of the current coding unit are derived as follows.

The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the luma coding block location ( xC, yC ), the variable nS set equal to nCSL, the variable cIdx set equal to 0, the (nCSL)x(nCSL) array predSamples set equal to predSamplesL, and the (nCSL)x(nCSL) array resSamples set equal to resSamplesL as the inputs.

The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the chroma coding block location ( xC/2, yC/2 ), the variable nS set equal to nCSC, the variable cIdx set equal to 1, the (nCSC)x(nCSC) array predSamples set equal to predSamplesCb, and the (nCSC)x(nCSC) array resSamples set equal to resSamplesCb as the inputs.

The picture reconstruction process prior to in-loop filtering for a colour component as specified in subclause 8.6.5 is invoked with the chroma coding block location ( xC/2, yC/2 ), the variable nS set equal to nCSC, the variable cIdx set equal to 2, the (nCSC)x(nCSC) array predSamples set equal to predSamplesCr, and the (nCSC)x(nCSC) array resSamples set equal to resSamplesCr as the inputs.

**H.8.5.5.3 Derivation Process of Partitioning Mode from Disparity Map**

Inputs to this process are:

* a location ( xCb, yCb ) specifying the top-left sample of the current luma coding block relative to the top left sample of the current picture,
* the size (width and height) of this prediction block, nCbS,

Outputs of this process are:

* a partitioning mode PartMode for the current coding unit

The derivation process for a disparity vector from temporal neighbour block as specified in subclause H.8.5.5.1 is invoked with the luma location ( xCb, yCb ), and the variable nCbS as inputs, and the outputs are the flag availableDV, the disparity vector mvDisp and the reference view order index refViewIdx.

The derivation process for a disparity sample array as specified in subclause H.8.5.5.2 is invoked with the luma locations xCb, yCb, the disparity vector mvDisp, the view identifier refViewIdx, the variable nPSW equal to nCbS, the variable nPSH equal to nCbS, and the variable splitFlag equal to 0 as the inputs, and the output is the array disparitySamples of size nCbS x nCbS.

The partitioning mode PartMode is derived from the array disparitySamples as follows:

dispSum = 0  
for ( y = 0; y < nCbS ; y+=4 )  
 for( x = 0; x < nCbS ; x+=4 )  
 dispSum +=  disparitySamples[ x ][ y ]  
dispMean = dispSum >> (2 \* ( log2( nCbS ) - 2 ))

ampAvail = nCbS > 8

for ( y = 0; y < nCbS ; y++ )  
 for( x = 0; x < nCbS ; x++ ) {   
 segmentIdx = disparitySamples[ x ][ y ] > dispMean  
 if( x < (nCbS >> 1) )  
 matchedPartSum[ 0 ][ segmentIdx ]++  
 else  
 matchedPartSum[ 0 ][ 1 – segmentIdx ]++  
 if( y < (nCbS >> 1) )  
 matchedPartSum[ 1 ][ segmentIdx ]++  
 else  
 matchedPartSum[ 1 ][ 1 – segmentIdx ]++   
  
 if( ampAvail ) {  
 if( y < (nCbS >> 2) )  
 matchedPartSum[ 2 ][ segmentIdx ]++   
 else  
 matchedPartSum[ 2 ][ 1 – segmentIdx ]++   
 if( y < (nCbS >> 2 + nCbS >> 1) )   
 matchedPartSum[ 3 ][ segmentIdx ]++   
 else  
 matchedPartSum[ 3 ][ 1 – segmentIdx ]++   
 if( x < (nCbS >> 2) )  
 matchedPartSum[ 4 ][ segmentIdx ]++  
 else  
 matchedPartSum[ 4 ][ 1 – segmentIdx ]++   
 if( x < (nCbS >> 2 + nCbS >> 1) )  
 matchedPartSum[ 5 ][ segmentIdx ]++  
 else  
 matchedPartSum[ 5 ][ 1 – segmentIdx ]++   
 }  
 }

The variable maxMatchSum is set equal to 0.  
The mapping PartModeMap[ p ] of an index p to the corresponding partition mode is specified as follows:  
PartModeMap[ 0..5 ] = { SIZE\_Nx2N, SIZE\_2NxN, SIZE\_2NxnU, SIZE\_2NxnD, SIZE\_nLx2N, SIZE\_nRx2N }

for ( p = 0; p < 6; p++ )  
 for( i = 0; i < 2; i++ ) {  
 if( matchedPartSum[ p ][ i ] > maxMatchSum ) {  
 maxMatchSum = matchedPartSum[ p ][ i ]  
 PartMode = PartModeMap[ p ]   
 }  
 }

**H.8.5.5.4 Derivation Process of Segmentation Mask from Disparity Map**

Note to editors: A very similar process is already in the WD for DMM4 and can potentially be reused.

Inputs to this process are:

* a location ( xCb, yCb ) specifying the top-left sample of the current luma coding block relative to the top left sample of the current picture,
* the size (width and height) of this prediction block, nCbS,

Outputs of this process are:

* a binary segmentation mask array segMask of size nCbS x nCbS

The derivation process for a disparity vector from temporal neighbour block as specified in subclause H.8.5.5.1 is invoked with the luma location ( xCb, yCb ), and the variable nCbS as inputs, and the outputs are the flag availableDV, the disparity vector mvDisp and the reference view order index refViewIdx.

The derivation process for a disparity sample array as specified in subclause H.8.5.5.2 is invoked with the luma locations xCb, yCb, the disparity vector mvDisp, the view identifier refViewIdx, the variable nPSW equal to nCbS, the variable nPSH equal to nCbS, and the variable splitFlag equal to 0 as the inputs, and the output is the array disparitySamples of size nCbS x nCbS.

The average disparity value dispMean of the disparity sample array disparitySamples is computed as specified in the following:

dispSum = 0  
for ( y = 0; y < nCbS ; y++ )  
 for( x = 0; x < nCbS ; x++ )  
 dispSum +=  disparitySamples[ x ][ y ]  
dispMean = dispSum >> (2 \* log2( nCbS ))

The boolean variable invertMask is derived as follows:

invertMask = disparitySamples[ 0 ][ 0 ] > dispMean

The binary segmentation mask array segMask is derived as specified in the following:

for ( y = 0; y < nCbS ; y++ )  
 for( x = 0; x < nCbS ; x++ ) {  
 if( invertMask )  
 segMask[ x ][ y ] = disparitySamples[ x ][ y ] <= dispMean  
 else  
 segMask[ x ][ y ] = disparitySamples[ x ][ y ] > dispMean  
 }

**H.8.5.5.5 Merging process for blocks coded with inter DBBP prediction mode**

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 variable log2CbSize specifying the size of the current coding block,
* (nCSL)x(nCSL) arrays predSamplesL0, predSamplesL1,
* (nCSC)x(nCSC) arrays predSamplesCb0, predSamplesCr0,predSamplesCb1, predSamplesCr1

Outputs of this process are three arrays:

* predSamplesL of size(nCSL)x(nCSL),
* predSamplesCb and predSamplesCr of size(nCSC)x(nCSC)

Merging process for coding blocks coded in inter DBBP prediction mode consists of following ordered steps:

1. The derivation process of segmentation mask from disparity map as specified in subclause H.8.5.5.4 is invoked with the luma location ( xC, yC ), and the variable nCbS as inputs, and the outputs are the segmentation mask segMask of size xCb x yCb.
2. Based on segMask, the following process is used to compute the final prediction arrays predSamplesL, predSamplesCb and predSamplesCr:

for ( y = 0; y < nCSL ; y++ )  
 for( x = 0; x < nCSL ; x++ ) {  
 if( y==0 ) t = segMask[ y+1 ][ x ] else t = segMask [ y-1 ][ x ]  
 if( x==0 ) l = segMask[ y ][ x+1 ] else l = segMask [ y ][ x-1 ]  
 if( y==nCSL-1 ) b = segMask[ y-1 ][ x ] else b = segMask [ y+1 ][ x ]  
 if( x== nCSL-1 ) r = segMask[ y ][ x-1 ] else r = segMask [ y ][ x+1 ]  
 doBlend = !((t&l&b&r) || (!t&!l&!b&!r))  
 if( doBlend ) {  
 predSamplesL[ x ][ y ] = ( predSamplesL0[ x ][ y ] + predSamplesL1[ x ][ y ] + 1 ) >> 1  
 } else {  
 if( segMask[ x ][ y ] == 0 )  
 predSamplesL[ x ][ y ] = predSamplesL0[ x ][ y ]  
 else  
 predSamplesL[ x ][ y ] = predSamplesL1[ x ][ y ]  
 }  
 }

for ( y = 0; y < nCSC ; y++ )  
 for( x = 0; x < nCSC ; x++ ) {  
 if( y==0 ) t = segMask[ (y+1)\*2 ][ x\*2 ] else t = segMask [ (y-1)\*2 ][ x\*2 ]  
 if( x==0 ) l = segMask[ y\*2 ][ (x+1)\*2 ] else l = segMask [ y\*2 ][ (x-1)\*2 ]  
 if( y==nCSL-1 ) b = segMask[ (y-1)\*2 ][ x\*2 ] else b = segMask [ (y+1)\*2 ][ x\*2 ]  
 if( x== nCSL-1 ) r = segMask[ y\*2 ][ (x-1)\*2 ] else r = segMask [ y\*2 ][ (x+1)\*2 ]  
 doBlend = !((t&l&b&r) || (!t&!l&!b&!r))  
 if( doBlend ) {  
 predSamplesCb[ x ][ y ] = ( predSamplesCb0[ x ][ y ] + predSamplesCb1[ x ][ y ] + 1 ) >> 1  
 predSamplesCr[ x ][ y ] = ( predSamplesCr0[ x ][ y ] + predSamplesCr1[ x ][ y ] + 1 ) >> 1  
 } else {  
 if( segMask[ x << 1 ][ y << 1 ] == 0 ) {  
 predSamplesCb[ x ][ y ] = predSamplesCb0[ x ][ y ]  
 predSamplesCr[ x ][ y ] = predSamplesCr0[ x ][ y ]  
 } else {  
 predSamplesCb[ x ][ y ] = predSamplesCb1[ x ][ y ]  
 predSamplesCr[ x ][ y ] = predSamplesCr1[ x ][ y ]  
 }  
 }  
 }

# Patent rights declaration

**RWTH Aachen University may have current or pending patent rights relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**

**Huawei Technologies may have current or pending patent rights relating to the technology described in this contribution and, conditioned on reciprocity, is prepared to grant licenses under reasonable and non-discriminatory terms as necessary for implementation of the resulting ITU-T Recommendation | ISO/IEC International Standard (per box 2 of the ITU-T/ITU-R/ISO/IEC patent statement and licensing declaration form).**

**References**

[1] I.-K. Kim, S. Lee, M.-S. Cheon, T. Lee, and J. Park, “Coding efficiency improvement of HEVC using asymmetric motion partitioning,” *Broadband Multimedia Systems and Broadcasting (BMSB), 2012 IEEE International Symposium on*, pp. 1–4, 2012.