|  |  |
| --- | --- |
| **Joint Collaborative Team on 3D Video Coding Extensions**  **of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11**  5th Meeting: Vienna, AT, 27 July – 02 Aug. 2013 | Document: JCT3V-E0118 |

|  |  |  |  |
| --- | --- | --- | --- |
| *Title:* | **CE3-related: Depth-based Block Partitioning** | | |
| *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 |
| *Source:* | RWTH Aachen University, Huawei Technologies | | |

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

# Abstract

In the proposed method, the dependent views' depth information is coded before the texture component to be able to utilize this coded depth data. Based on a binary segmentation mask computed from the reconstructed depth map, an arbitrarily shaped block partitioning for the collocated texture block is derived. 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 0.8% BD-rate savings on the video component (w.r.t. total bitrate).

# Depth-First Coding Order

In the latest version of the common test conditions texture is always coded before the depth component for each view. There are coding tools (e.g. DMM3/4, QTLPC and motion inheritance) utilizing already coded texture data to increase coding efficiency of the collocated depth map. When looking at the bitrate distribution of a typical 3D video bitstream (based on CTC sequences and configuration), it is obvious that the dependent views’ depth maps occupy only a very small portion (2,96%) of the total bitrate and consequently it might be better to use the coded depth information to increase coding efficiency for the dependent views texture components (16,10%).

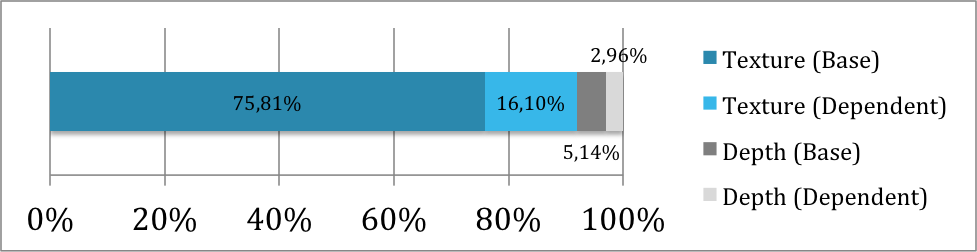


Figure 1: Bitrate distribution for an average bitstream that is generated following the CTC and using HTM 6.0 reference software.

Moreover, the current CTC does not provide stereo backwards compatibility since view synthesis prediction (VSP) requires depth map data to be available when decoding dependent views texture (and depth). Hence, using a depth-first coding order for the dependent views would not introduce an additional restriction regarding backwards compatibility.

# 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. The segmentation 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 current 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 CTB. 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 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, 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.

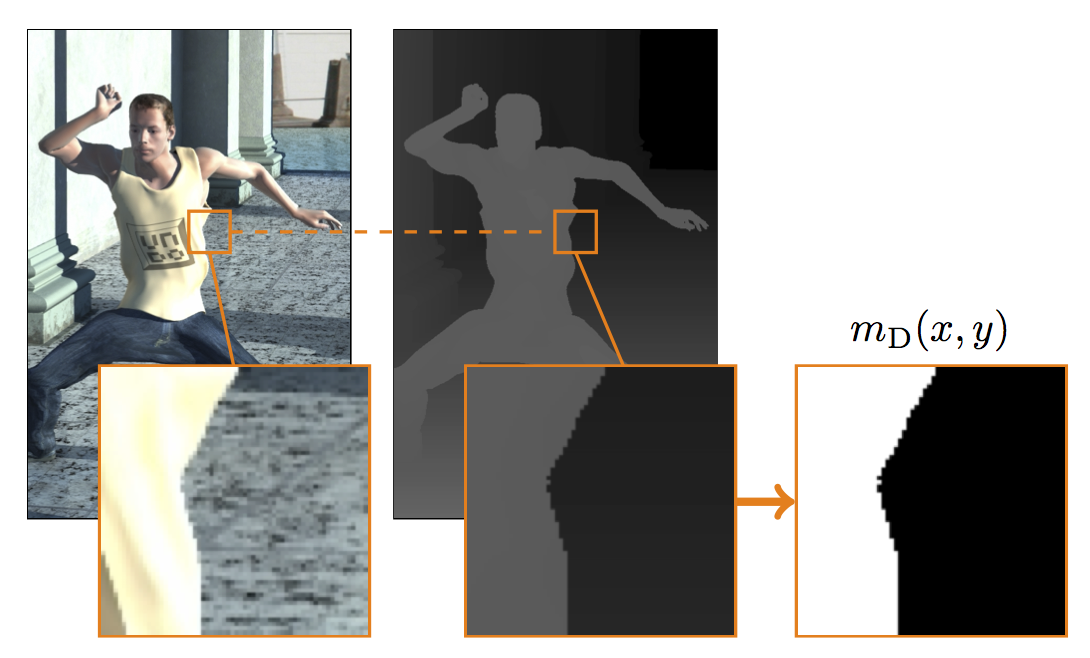


Figure 2: Cropped frame from *Undo\_Dancer* test sequence with magnified component blocks of an example coding unit. The collocated depth block segments the block into foreground and background, which is defined by a binary mask .

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 CTB. 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.

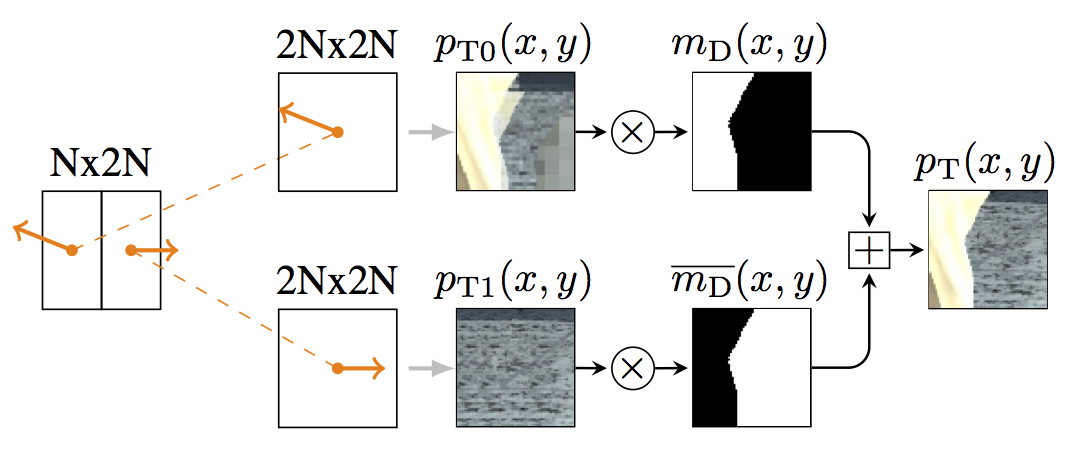


Figure 3: DBBP merging process: For each of the two decoded motion parameters a motion compensation is performed. The resulting prediction signals and are combined using the DBBP mask .

# Coding of Partitioning Information

As described in the previous section, DBBP requires to code 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 [5], 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.

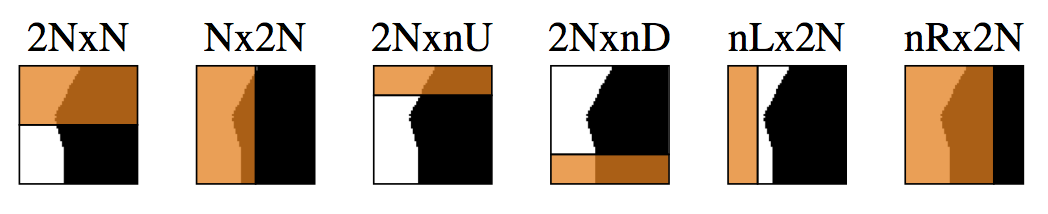


Figure 4: Superposition of conventional partitioning modes (gray) and depth-based block partitioning (black/white). The best matching partitioning () mode is selected for storing motion information.

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 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. This modified part of the decoding process is described in the following algorithm:



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 6.1. As DBBP requires depth-first coding order for the dependent views, flexible coding order (FCO) was enabled in the encoder for the tested simulations.

## DBBP vs. HTM 6.1 (Depth-First)

In this simulation the anchor is HTM 6.1 with depth-first (DF) coding order for the dependent views. The same coding order is used for the tested simulation. Consequently, the results show the benefit coming from adding DBBP to such a coding configuration.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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% | -2,40% | -2,42% | -1,07% | -1,0% | -0,8% | 104,2% | 99,2% |
| Kendo | 0,0% | -4,99% | -5,71% | -2,28% | -2,0% | -1,7% | 102.3% | 101,2% |
| Newspaper\_CC | 0,0% | -1,64% | -2,15% | -0,79% | -0,7% | -0,6% | 105,8% | 100,5% |
| GT\_Fly | 0,0% | -0,79% | -1,13% | -0,24% | -0,2% | -0,2% | 104,0% | 100,3% |
| Poznan\_Hall2 | 0,0% | -0,56% | -2,01% | -0,67% | -0,6% | -0,3% | 104,7% | 100,1% |
| Poznan\_Street | 0,0% | -1,23% | -1,47% | -0,45% | -0,4% | -0,4% | 103,9% | 99,9% |
| Undo\_Dancer | 0,0% | -3,07% | -3,98% | -0,93% | -0,9% | -0,8% | 102,0% | 102,2% |
| 1024x768 | 0,0% | -3,01% | -3,43% | -1,38% | -1,2% | -1,0% | 105,0% | 100,3% |
| 1920x1088 | 0,0% | -1,41% | -2,15% | -0,57% | -0,5% | -0,4% | 103,7% | 100,6% |
| **average** | **0,0%** | **-2,10%** | **-2,70%** | **-0,92%** | **-0,8%** | **-0,7%** | **104,1%** | **100,5%** |

## DBBP vs. HTM 6.1 (CTC)

In this simulation the anchor is HTM 6.1 configured according to the common test conditions (CTC). For the tested simulation depth-first coding order in combination with DBBP is used.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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% | -0,3% | -1,4% | -0,3% | 2,0% | 1,2% | 104,2% | 100,9% |
| Kendo | 0,0% | -0,9% | -1,3% | -0,4% | 2,8% | 2,1% | 105,0% | 101,0% |
| Newspaper\_CC | 0,0% | -0,5% | -1,0% | -0,3% | 2,2% | 0,8% | 104,6% | 101,8% |
| GT\_Fly | 0,0% | -1,1% | -1,5% | -0,3% | 1,8% | 1,6% | 105,1% | 102,2% |
| Poznan\_Hall2 | 0,0% | -0,2% | -1,1% | -0,3% | 3,3% | 2,8% | 105,2% | 100,4% |
| Poznan\_Street | 0,0% | -1,1% | -0,8% | -0,3% | 1,6% | 1,7% | 102,3% | 100,9% |
| Undo\_Dancer | 0,0% | -3,5% | -5,1% | -1,2% | 0,7% | 0,6% | 102,8% | 101,2% |
| 1024x768 | 0,0% | -0,6% | -1,2% | -0,3% | 2,3% | 1,4% | 104,6% | 101,2% |
| 1920x1088 | 0,0% | -1,5% | -2,1% | -0,5% | 1,8% | 1,7% | 103,9% | 101,2% |
| **average** | **0,0%** | **-1,1%** | **-1,8%** | **-0,4%** | **2,0%** | **1,5%** | **104,2%** | **101,2%** |

## DBBP vs. HTM 6.1 (Same VSO)

In this simulation the anchor is HTM 6.1 configured according to the common test conditions (CTC) with a modified VSO configuration string, which is the same as for the depth-first coding configuration. For the tested simulation depth-first coding order in combination with DBBP is used.

The used VSO configuration string is the following:

[ox0 B(cc1) I(s0.25 s0.5 s0.75)][cx1 B(oo0) B(oo2) I(s0.25 s0.5 s0.75 s1.25 s1.5 s1.75)][ox2 B(cc1) I(s1.25 s1.5 s1.75)]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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% | -0,3% | -1,4% | -0,3% | 1,6% | 0,9% | 102,3% | 100,9% |
| Kendo | 0,0% | -0,9% | -1,3% | -0,4% | 2,1% | 1,6% | 102,8% | 101,2% |
| Newspaper\_CC | 0,0% | -0,5% | -1,0% | -0,3% | 1,7% | 0,6% | 104,2% | 100,9% |
| GT\_Fly | 0,0% | -1,1% | -1,5% | -0,3% | 1,0% | 0,8% | 104,6% | 101,8% |
| Poznan\_Hall2 | 0,0% | -0,2% | -1,1% | -0,3% | 2,3% | 2,0% | 105,0% | 101,0% |
| Poznan\_Street | 0,0% | -1,1% | -0,8% | -0,3% | 1,0% | 1,0% | 104,2% | 101,1% |
| Undo\_Dancer | 0,0% | -3,5% | -5,1% | -1,2% | 0,2% | 0,0% | 105,2% | 100,4% |
| 1024x768 | 0,0% | -0,6% | -1,2% | -0,3% | 1,8% | 1,0% | 103,1% | 101,0% |
| 1920x1088 | 0,0% | -1,5% | -2,1% | -0,5% | 1,1% | 1,0% | 104,8% | 101,1% |
| **average** | **0,0%** | **-1,1%** | **-1,8%** | **-0,4%** | **1,4%** | **1,0%** | **104,0%** | **101,0%** |

# Cross Check

The cross check of the proposed DBBP coding scheme was performed by Ghent University. They investigated the 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, the group should consider to add a depth-first coding configuration to the common test conditions or to establish a new Core Experiment on depth-first coding tools. For the AVC-based 3D extension a depth-first coding order is regarded as being more efficient. Similar observations might be made for the HEVC-based 3D extension as soon as texture-dependent coding tools are modified and new depth-dependent coding tools utilize already coded depth data.

# 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).**1