INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11 MPEG2011/M12350
November 2011, Geneva, Switzerland

Source **Video**

Status **Approved**

Title Test Model under Consideration for HEVC based 3D video coding

Editors Heiko Schwarz, Krzysztof Wegner

This document summarizes the system structure and coding tools that are included in the test model under consideration and provides information on the HEVC-based 3DV software.

The coding tools that are not available in the first version of the software are marked using the label "integration tool".

Table of Contents

1 Data Format and System Description

3D video is represented using the Multiview Video plus Depth (MVD) format, in which a small number of captured views as well as associated depth maps are coded and the resulting bitstream packets are multiplexed into a 3D video bitstream. After decoding the video and depth data, additional intermediate views suitable for displaying the 3D content on an auto-stereoscopic display can be synthesized using depth-image-based rendering (DIBR) techniques. For the purpose of view synthesis, camera parameters are additionally included in the bitstream. The bitstream packets include header information, which signal, in connection with transmitted parameter sets, a view identifier and an indication whether the packet contains video or depth data. Sub-bitstreams containing only some of the coded components can be extracted by discarding bitstream packets that contain non-required data. One of the views, which is also referred to as the base view or the independent view, is coded independently of the other views and the depth data using a conventional HEVC video coder. The sub-bitstream containing the independent view can be decoded by an unmodified HEVC video decoder and displayed on a conventional 2D display. Optionally, the encoder can be configured in a way that a sub-bitstream representing two views without depth data can be extracted and independently decoded for displaying the 3D video on a conventional stereo display. The codec can also be used for coding multiview video signals without depth data. In that case alternative methods such as Image Domain Warping (IDW) may be used to generate a multiview signal. And, when using depth data, it can be configured in a way that the video pictures can be decoded independently of the depth data.

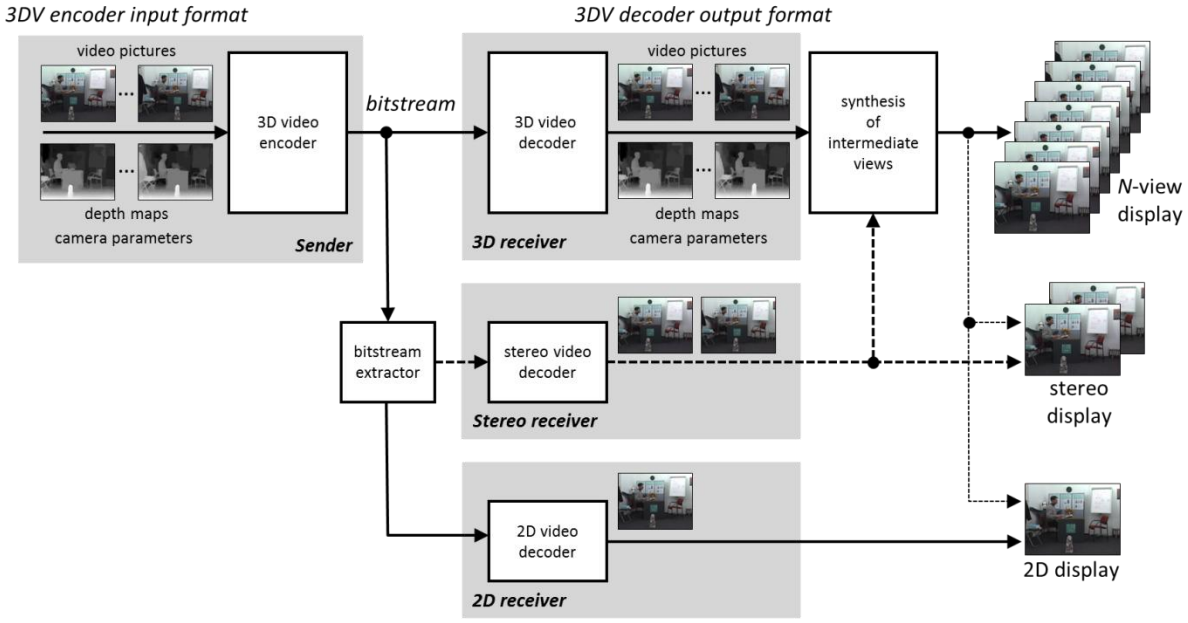

Figure 1: Overview of the system structure and the data format for the transmission of 3D video.

The basic concept of the system and data format is illustrated in Figure 1. In general the input signal for the encoder consists of multiple views, associated depth maps, and corresponding camera parameters. However, as described above, the codec can also be operated without depth data. The input component signals are coded using a 3D video encoder, which represents an extension of HEVC. At this, the base view is coded using an unmodified HEVC encoder. The 3D video encoder generates a bitstream, which represents the input videos and depth data in a coded format. If the bitstream is decoded using a 3D video decoder, the input videos, the associated

depth data, and camera parameters are reconstructed with the given fidelity. For displaying the 3D video on an autostereoscopic display, additional intermediate views are generated by a DIBR algorithm using the reconstructed views and depth data. If the 3D video decoder is connected to a conventional stereo display instead of to an autostereoscopic display, the view synthesizer can also generate a pair of stereo views, in case such a pair is not actually present in the bitstream. At this, it is possible to adjust the rendered stereo views to the stereo geometry of the viewing conditions. One of the decoded views or an intermediate view at an arbitrary virtual camera position can also be used for displaying a single view on a conventional 2D display.

The 3D video bitstream is constructed in a way that the sub-bitstream representing the coded representation of the base view can be extracted by simple means. The bitstream packets representing the base view can be identified by inspecting transmitted parameter sets and the packet headers. The sub-bitstream for the base view can be extracted by discarding all packets that contain depth data or data for the dependent views and, then, the extracted sub-bitstream can be directly decoded with an unmodified HEVC decoder and displayed on a conventional 2D video display.

The encoder can also be configured in a way that the sub-bitstream containing only two stereo views can be extracted and directly decoded using a stereo decoder. The encoder can also be configured in a way that the views can be generally decoded independently of the depth data. It is also possible to synthesize intermediate view using only the stereo sequences as input of the view synthesis.

A detailed description of the coding scheme is given in sec. 2. Depth-image-based rendering algorithms are described in sec. 3.

2 Coding Algorithm

In the following, the coding algorithm based on the MVD format, in which each video picture is associated with a depth map, is described. The coding algorithm can also be used for a multiview format without depth maps. The video pictures and, when present, the depth maps are coded access unit by access unit, as it is illustrated in Figure 2. An *access unit* includes all video pictures and depth maps that correspond to the same time instant. Non-VCL NAL units containing camera parameters may be additionally associated with an access unit. It should be noted that the coding order of access units doesn't need to be identical to the capture or display order. In general, the reconstructed data of already coded access units can be used for an efficient coding of the current access unit. Random access is enabled by so-called *random access units* or *instantaneous decoding refresh* (IDR) access units, in which the video pictures and depth maps are coded without referring to previously coded access units. Furthermore, an access unit doesn't reference any access unit that precedes the previous random access unit in coding order.

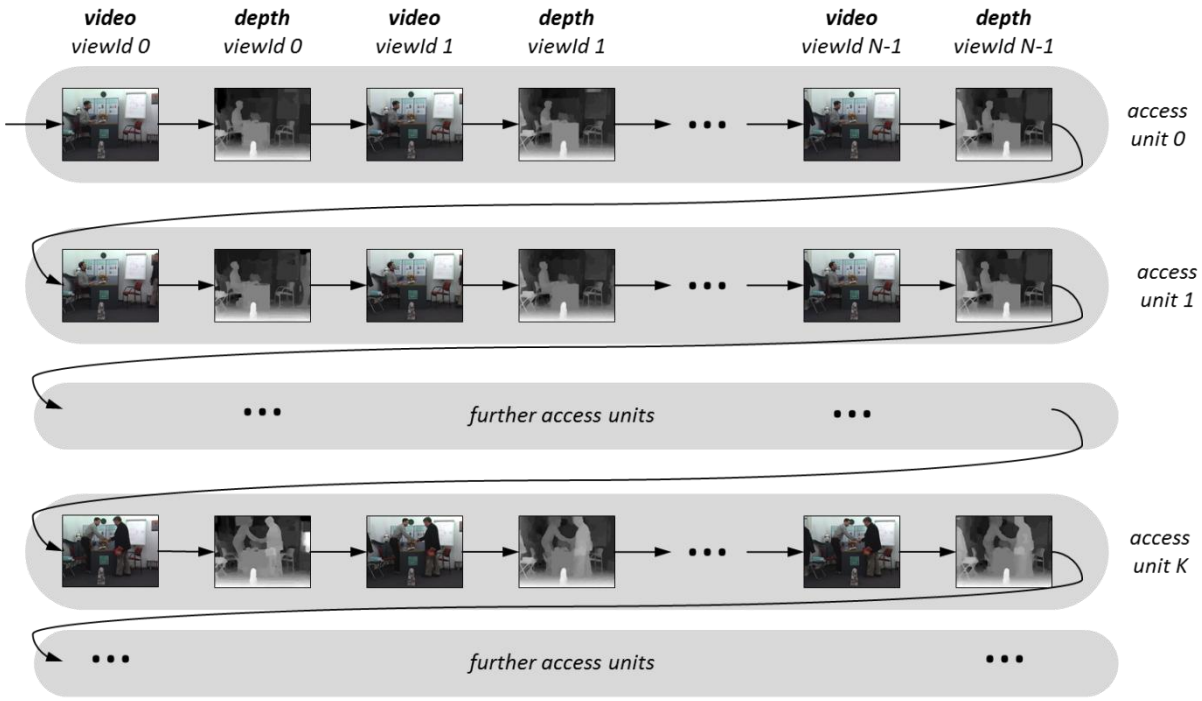

Figure 2: Access unit structure and coding order of view components.

The video pictures and depth maps corresponding to a particular camera position are indicated by a view identifier (viewId). All video pictures and depth maps that belong to the same camera position are associated with the same value of viewId. The view identifiers are used for specifying the coding order inside the access units and detecting missing views in error-prone environments. Inside an access unit, the video picture and, when present, the associated depth map with viewId equal to 0 are coded first, followed by the video picture and depth map with viewId equal to 1, etc. A video picture and depth map with a particular value of viewId are transmitted after all video pictures and depth maps with smaller values of viewId. For the independent view, the video picture is always coded before the associated depth map. For dependent views, the video picture may be coded before or after the associated depth map (i.e., the depth map with the same value of viewId). It should be noted that the value of viewId doesn't necessarily represent the arrangement of the cameras in the camera array. For ordering the reconstructed video pictures and depth map after decoding, each value of viewId is associated with another identifier called view order index (VOI). The view order index is a signed integer values, which specifies the ordering of the coded views from left to right. If a view A has a smaller value of VOI than a view B, the camera for view A is located left to the camera of view B. In addition, camera parameters required for converting depth values into disparity vectors are included in the bitstream. For the considered linear setup, the corresponding conversion parameters consist of a scale factor and an offset. The vertical component of a disparity vector is always equal to 0. The horizontal component is derived according to

$$d_v = (s * v + o) >> n,$$

where v is the depth sample value, s is the transmitted scale factor, o is the transmitted offset, and n is a shift parameter that depends on the required accuracy of the disparity vectors.

Each video sequence and depth sequence is associated with a separate sequence parameter set and a separate picture parameter set. The picture parameter set syntax, the NAL unit header syntax, and the slice header syntax for the coded slices haven't been modified for including a

mechanism by which the content of a coded slice NAL units can be associated with a component signal. Instead, the sequence parameter set syntax for all component sequences except for the base view has been extended. Theses sequences parameter sets contain the following additional parameters:

- the view identifier (indicates the coding order of a view);
- the depth flag (indicates whether video data or depth data are present);
- the view order index (indicates the location of the view relative to other coded views);
- an indicator specifying whether camera parameters are present in the sequence parameter set or in the slice headers;
- when camera parameters are present in an sequence parameter set, for each viewId value smaller than the current view identifier, a scale and an offset specifying the conversion of a depth sample of the current view to a horizontal disparity between the current view and the view with viewId;
- when camera parameters are present in an sequence parameter set, for each viewId value smaller than the current view identifier, a scale and an offset specifying the conversion of a depth sample of the view with viewId to a horizontal disparity between the current view and the view with viewId;

The sequence parameter set for the base view doesn't contain the additional parameters. Here, the view identifier is inferred to be equal to 0, the depth flag is inferred to be equal to 0, and the view order index is inferred to be equal to 0.

The sequence parameter sets for dependent views include a flag, which specifies whether the camera parameters are constant for a coded video sequence or whether they can change on a picture by picture basis. If this flag indicates that the camera parameters are constant for a coded video sequence, the camera parameters (i.e., the scale and offset values described above) are present in the sequence parameter set. Otherwise, the camera parameters are not present in the sequence parameter set, but instead the camera parameters are coded in the slice headers that reference the corresponding sequence parameter set.

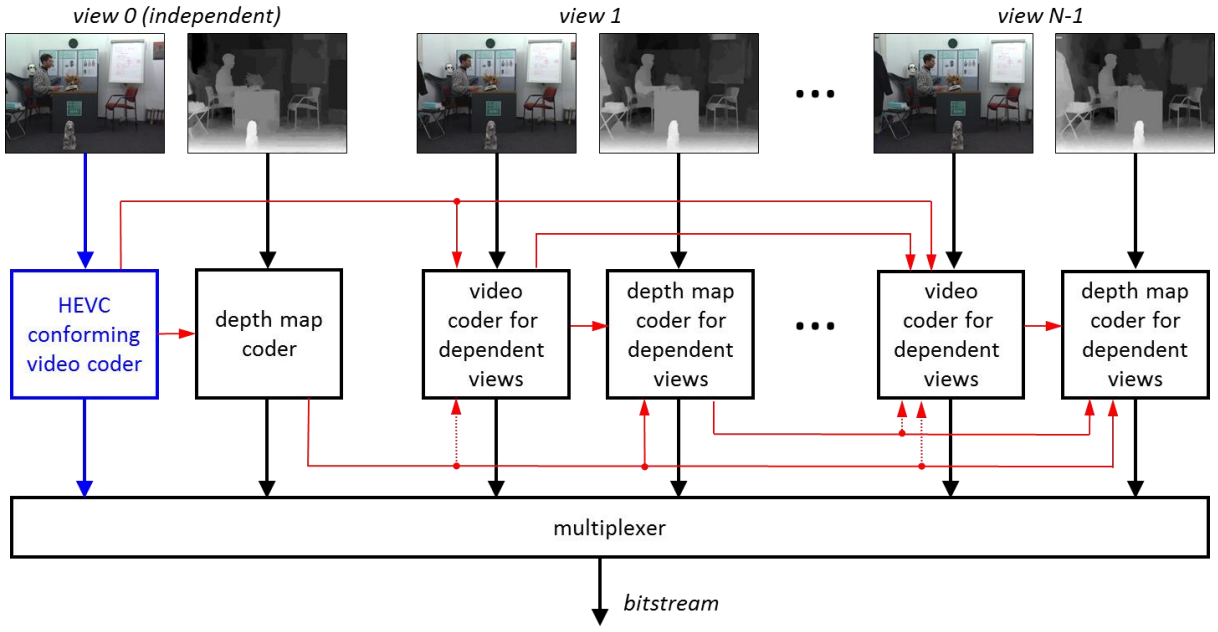

Figure 3: Basic codec structure with inter-component prediction (red arrows).

The basic structure of the 3D video codec is shown in the block diagram of Figure 3. In principle, each component signal is coded using an HEVC-based codec. The resulting bitstream packets, or more accurately, the resulting Network Abstraction Layer (NAL) units, are multiplexed to form the 3D video bitstream. The base or independent view is coded using an unmodified HEVC codec. Given the 3D video bitstream, the NAL units containing data for the base layer can be identified by parsing the parameter sets and NAL unit header of coded slice NAL units (up to the picture parameter set identifier). Based on these data, the sub-bitstream for the base view can be extracted and directly coded using a conventional HEVC decoder.

For coding the dependent views and the depth data, modified HEVC codecs are used, which are extended by including additional coding tools and inter-component prediction techniques that employ already coded data inside the same access unit as indicated by the red arrows in Figure 3. For enabling an optional discarding of depth data from the bitstream, e.g., for supporting the decoding of a stereo video suitable for conventional stereo displays, the inter-component prediction can be configured in a way that video pictures can be decoded independently of the depth data. A detailed description of the added coding tools is given in the following subsections.

2.1 Coding of the Independent View

The independent view, which is also referred to as the base view, is coded using an unmodified HEVC codec.

2.2 Coding of Dependent Views

For the dependent views, the same concepts and coding tools are used as for the independent view. However, additional tools have been integrated into the HEVC codec, which employ already coded data in other views for efficiently representing a dependent view. The additionally integrated tools are described in the following.

2.2.1 Disparity-compensated prediction

As a first coding tool for the dependent views, the well-known concept of disparity-compensated prediction (DCP), which is also used in MVC, has been added as an alternative to motion-compensated prediction (MCP). At this, MCP refers to an inter-picture prediction that uses already coded pictures of the same view, while DCP refers to an inter-picture prediction that uses already coded pictures of other views in the same access unit, as it is illustrated in Figure 4.

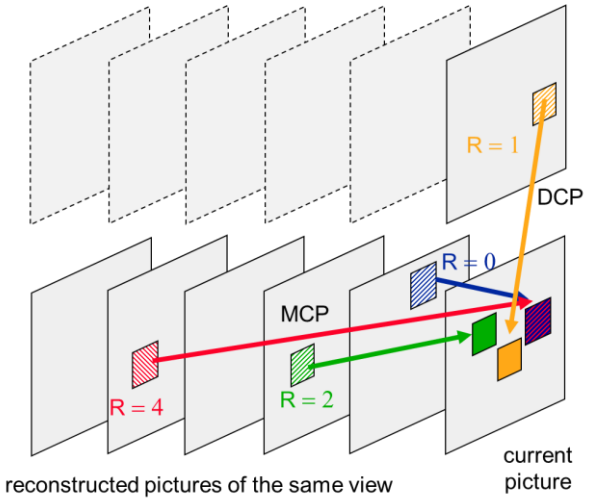

Figure 4: Disparity-compensated prediction as an alternative to motion-compensated prediction.

The macroblock syntax and decoding process haven't been changed for adding DCP, only the high-level syntax has been modified so that already coded video pictures of the same access unit can be inserted into the reference pictures lists. As illustrated in Figure 4, the transmitted reference picture index (R in the figure) signals whether an inter-coded blocks is predicted by MCP or DCP. The motion vector prediction is modified in a way that the motion vectors of motion-compensated blocks are predicted by only using the neighboring blocks that also use temporal reference pictures, while the disparity vectors of disparity-compensated blocks are predicted by only using the neighboring blocks that also use inter-view reference pictures.

2.2.2 View synthesis based inter-view prediction (integration tool)

The encoder and the decoder use the same inter-prediction view synthesis algorithm. The included view synthesis algorithm may be similar to the one investigated in the VSRS software. Basing on all already coded views, a new virtual view is synthesized in the position of the current view. Some regions of newly synthesized image are not available because they were occluded in previously coded views. Those disoccluded regions are identified and marked on a binary map, named availability map, which controls coding and decoding process. Coder and decoder simultaneously use this map to determine, whether given CU is coded or not. Because in a typical case most of the scene is the same in all of views, only small parts are disoccluded in subsequently coded views, and thus only small amount of CUs can be coded.

a) b) c)

Figure 5: a) The original side view, b) Disocclusion in the side view, and c) CUs selected by the rd-opt for coding in the side view.

2.2.2.1 Post processing in-loop filtering (integration tool)

A final step of view-synthesis prediction is reduction of artifacts in synthesized view. This post-processing consists of Depth-Gradient-based Loopback Filterer (DGLF) and Availability Deblocking Loopback Filter (ADLF).

The first one (DGLF), reduces texture artifacts introduced by DIBR technique in the areas of a sudden depth changes. In order to cope that the synthesized image is adaptively filtered with respect to depth gradient strengths. Large depth edges impose strong low-pass filtering of the synthesized texture, while flat depth regions are not filtered at all.

The latter (ADLF), reduces artifacts that are generated as a result of block CU-based coding. Shape of coded region not necessarily matches shape of binary availability map. This discrepancy is a source of artificial edges between those regions (Figure 5b) and c)) . The ADLF provides smooth transition between coded and synthesized regions by interpolating between them.

2.2.3 Inter-view motion prediction

The basic concept of the inter-view prediction of motion parameters is illustrated in Figure 6. For the following overview, it is assumed that an estimate of a pixel-wise depth map for the current picture is given. Below, it is described how such an estimate can be derived. For deriving candidate motion parameters for a current block in a dependent view, a sample location $\boldsymbol{x}$ in the middle of the block is selected and the associated depth value d is converted to a disparity vector. By adding the disparity vector to the sample location $\boldsymbol{x}$ a reference sample location $\boldsymbol{x}_R$ is obtained. The prediction block in the already coded picture in the reference view that covers the sample location $\boldsymbol{x}_R$ is used as the reference block. If this reference block is coded using MCP, the associated motion parameters can be used as candidate motion parameters for the current block in the current view. The derived disparity vector can also be directly used as a candidate disparity vector for DCP.

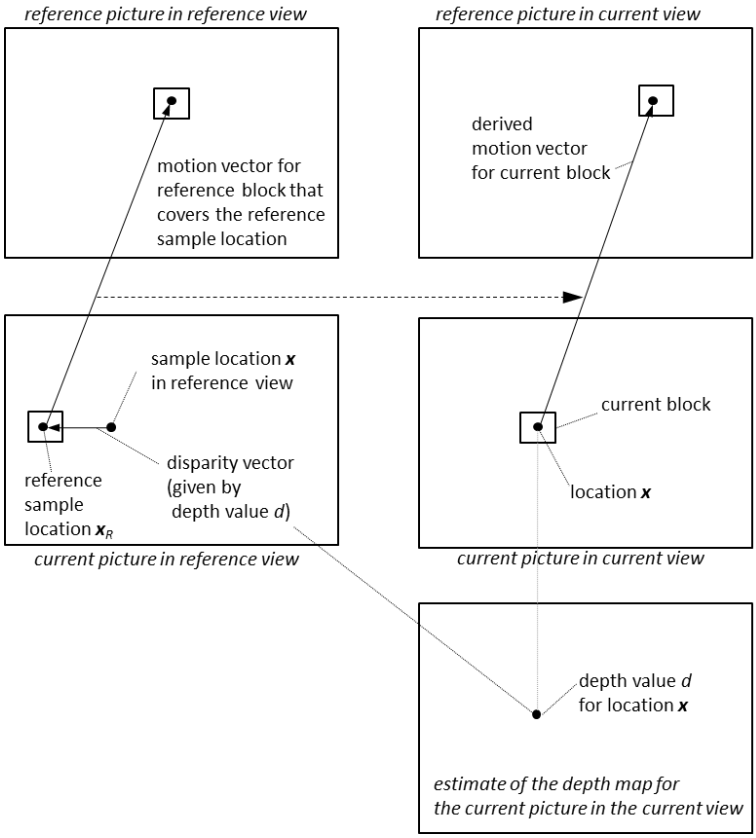

Figure 6: Basic principle of deriving motion parameters for a block in a current picture based on motion parameters in an already coded reference view and an estimate of the depth map for the current picture.

2.2.3.1 Derivation of Depth Map Estimates

The concept of inter-view motion prediction requires a depth map estimate for the current picture. Even if depth maps are coded, the depth map associated with a picture can be coded after the picture in order to enable coding techniques that employ the coded pictures for an efficient representation of the depth maps. In the following, two methods by which a suitable estimate for the depth map of the current picture can be derived based on already transmitted information are described. Both methods have been integrated in the codec, and one of the methods can be chosen by configuring the encoder accordingly. The used method is signaled in the sequence parameter sets for dependent views. This first method requires the transmission of depth data as part of the bitstream, and by using this method a decoder must decode the depth maps of previously coded views for decoding dependent views. The second method is also applicable if depth maps are not coded inside the bitstream, and if depth maps are coded, the decoding of the video pictures is independent of the depth maps.

Method 1: Depth map estimate based on already coded depth map

Since the depth map for a reference view is coded before the current picture, the reconstructed depth map is mapped into the coordinate system of the current picture for obtaining a suitable depth map estimate for the current picture. In Figure 7, such a mapping is illustrated for a simple depth map, which consists of a square foreground object and background with constant depth. For each sample of the given depth map, the depth sample value is converted into a sample-accurate disparity vector. Then, each sample of the depth map is displaced by the disparity vector. If two or more samples are displaced to the same sample location, the sample value that

represents the minimal distance from the camera (i.e., the sample with the larger value) is chosen. In general, the described mapping leads to sample locations in the target view to which no depth sample value is assigned (black area in the middle picture of Figure 7). These areas represent parts of the background that are uncovered due to the movement of the camera and can be filled using surrounding background sample values. Therefore, a hole filling algorithm, which processes the converted depth map line by line, is used. Each line segment that consists of successive sample location to which no value has been assigned is filled with the depth value of the two neighboring samples that represents a larger distance to the camera (i.e., the smaller depth value).

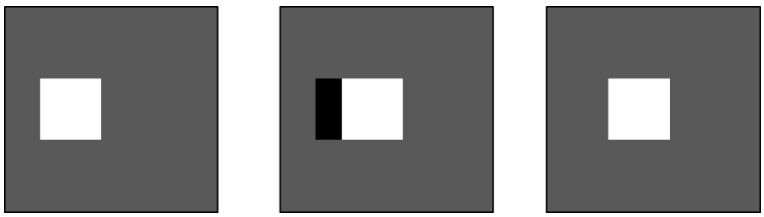

Figure 7: Mapping of a depth map into another view: (left) original depth map; (middle) converted depth map after displacing the orginal samples; (right) final converted depth map after filling of holes.

Method 2: Depth map estimate based on coded disparity and motion vectors

The above described method 1 is only applicable if depth maps are included in the bitstream, and by using this method, the video pictures (except the base view) cannot be decoded independently of the depth maps. In the following, a method for deriving depth map estimates that only uses data that are available in the coded representations of the video pictures is described.

In random access units, all blocks of the base view picture, are intra-coded. In the pictures of dependent views, most blocks are typically coded using DCP and the remaining blocks are intra-coded. When coding the first dependent view in a random access unit, no depth or disparity information is available. Hence, candidate disparity vectors are derived using a local neighborhood, i.e., by conventional motion vector prediction. But after coding the first dependent view in a random access unit, the transmitted disparity vectors are used for deriving a depth map estimate, as it is illustrated in Figure 8. Therefore, the disparity vectors used for DCP are converted into depth values and all depth samples of a disparity-compensated block are set equal to the derived depth value. The depth samples of intra-coded blocks are derived based on the depth samples of neighboring blocks; the used algorithm is similar to spatial intra prediction. If more than two views are coded, the obtained depth map is mapped into other views using the method described above and used as depth map estimate for deriving candidate disparity vectors.

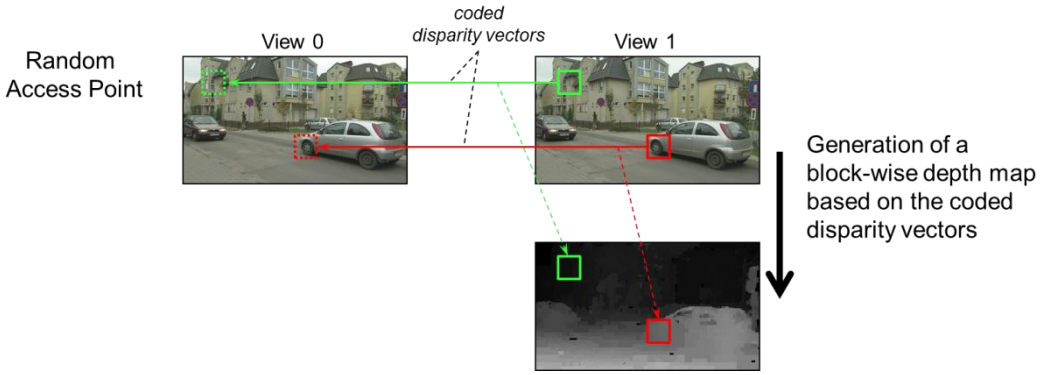

Figure 8: Generation of an initial depth map estimate after coding the first dependent view of a random access unit.

The depth map estimate for the picture of the first dependent view in a random access unit is used for deriving a depth map for the next picture of the first dependent view. The basic principle of the algorithm is illustrated in Figure 9. After coding the picture of the first dependent view in a random access unit, the derived depth map is mapped into the base view and stored together with the reconstructed picture. The next picture of the base view is typically inter-coded. For each block that is coded using MCP, the associated motion parameters are applied to the depth map estimate. A corresponding block of depth map samples is obtained by MCP with the same motion parameters as for the associated texture block; instead of a reconstructed video picture the associated depth map estimate is used as reference picture. In order to simplify the motion compensation and avoid the generation of new depth map values, the MCP for depth block doesn't involve any interpolation. The motion vectors are rounded to sample-precision before they are used. The depth map samples of intra-coded blocks are again determined on the basis of neighboring depth map samples. Finally, the depth map estimate for the first dependent view, which is used for the inter-view prediction of motion parameters, is derived by mapping the obtained depth map estimate for the base view into the first dependent view.

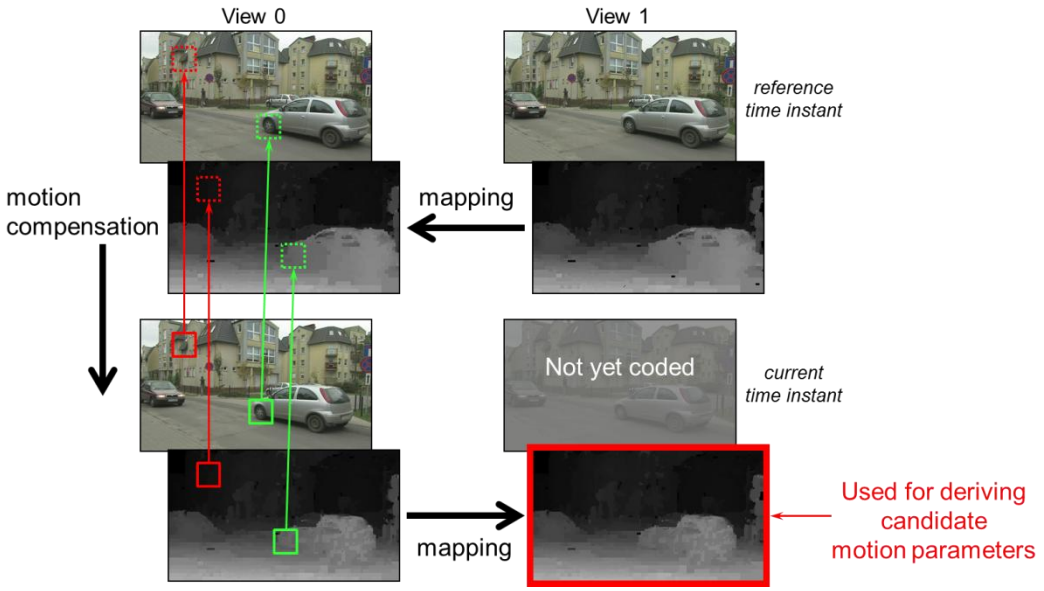

Figure 9: Derivation of a depth map estimate for the current picture using motion parameters of an already coded view of the same access unit.

After coding the second picture of the first dependent view, the estimate of the depth map is updated based on actually coded motion and disparity parameters, as it is illustrated in Figure 10. For blocks that are coded using DCP, the depth map samples are obtained by converting the disparity vector into a depth value. The depth map samples for blocks that are coded using MCP can be obtained by MCP of the previously estimated depth maps, similar as for the base view. In order to account for potential depth changes, new depth values are determined by adding a depth correction. The depth correction is derived by converting the difference between the motion vectors for the current block and the corresponding reference block of the base view into a depth difference. The depth values for intra-coded blocks are again determined by a spatial prediction. The updated depth map is mapped into the base view and stored together with the reconstructed picture. It is also used for deriving a depth map estimate for other views in the same access unit.

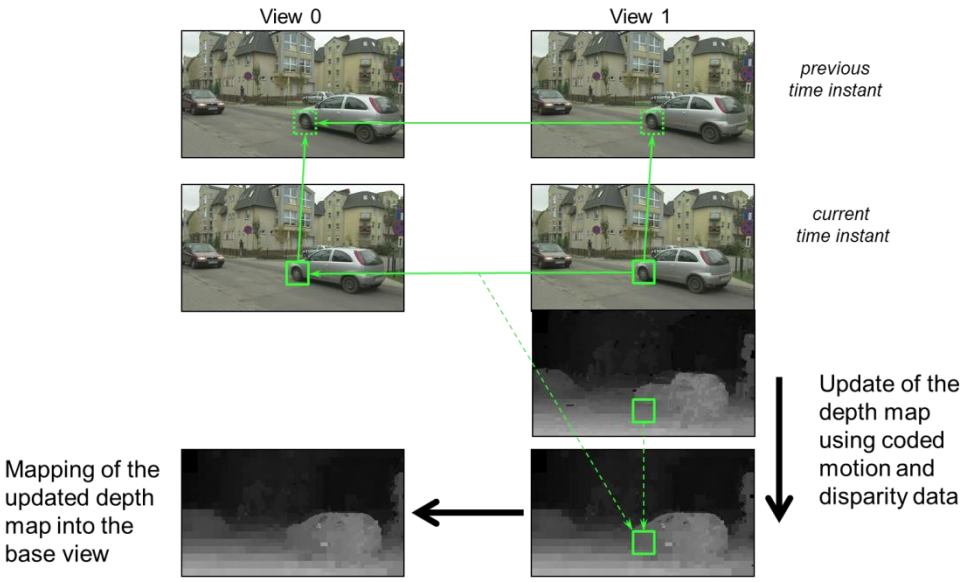

Figure 10: Update of depth map estimate for a dependent view based on coded motion and disparity vectors.

For all following pictures, the described process is repeated. After coding the base view picture, a depth map estimate for the base view picture is determined by MCP using the transmitted motion parameters. This estimate is mapped into the second view and used for the inter-view prediction of motion parameters. After coding the picture of the second view, the depth map estimate is updated using the actually used coding parameters. At the next random access unit, the inter-view motion parameter prediction is not used, and after decoding the first dependent view of the random access unit, the depth map is re-initialized as described above.

2.2.3.2 Usage of Inter-View Motion Parameter Prediction

In HEVC, two different modes for signaling the motion parameters for a block are specified. In the first mode, which is referred to as adaptive motion vector prediction (AMVP) mode, the number of motion hypotheses, the reference indices, the motion vector differences, and indications specifying the used motion vector predictors are coded in the bitstream. The second mode is referred to as merge mode. For this mode, only an indication is coded, which signals the set of motion parameters that are used for the block. The inter-view motion parameter prediction has been added to both modes, as will be described in the following.

Inter-view motion vector prediction in the AMVP mode

In the adaptive motion vector prediction (AMVP) mode, the number of motion hypotheses, the reference indices specifying the used reference pictures, the motion vector differences, and indexes specifying the used motion vector predictor are transmitted in the bitstream. For each motion hypothesis, a candidate list of motion vector predictors is derived based on the coded reference index. This list includes motion vectors of neighboring blocks that are associated with the same reference index as well as a motion vector predictor which is derived based on the motion parameters of the co-located block in a temporal reference picture. For including the inter-view motion parameter prediction, the AMVP mode has been extended in a way that an inter-view motion vector predictor is added to the candidate list. In our implementation it is inserted at the third position of the list. Based on the depth estimate for a middle sample of the current block, a disparity vector and a reference block in a reference view is determined as described above. If the reference index for the current block refers to an inter-view reference picture, the inter-view motion vector predictor is set equal to the corresponding disparity vector. If the current reference index refers to a temporal reference picture and the reference block uses a motion hypothesis that refers to the same access unit as the current reference index, the motion vector that is associated with this motion hypothesis is used as inter-view motion vector predictor. In all other cases, the inter-view motion vector predictor is marked as invalid and is not included in the list of motion vector predictor candidates.

Inter-view motion vector prediction in the merge mode (and skip mode)

In the merge mode of HEVC (as well as in the skip mode, which represents the merge mode without coding a residual signal), basically the same motion parameters (number of hypotheses, reference pictures, and motion vectors) as for a neighboring block are used. If a block is coded in the merge mode, a candidate list of motion parameters is derived, which includes the motion parameters of spatially neighboring blocks as well as motion parameters that are calculated based on the motion parameters of the co-located block in a temporal reference picture. The chosen motion parameters are signaled by transmitting an index into the candidate list. Similarly as for the AMVP mode, the candidate list of motion parameters is extended by a motion parameter set that is obtained using inter-view motion prediction, as described in the following. For each potential motion hypothesis, the first two reference indices of the reference picture list are investigated in the given order. A motion vector candidate for the reference index 0 is derived in the same way as for the AMVP mode. If the derived motion vector is valid, the reference index 0 and the derived motion vector are used for the considered hypothesis. Otherwise, the reference index 1 is tested in the same way. If it also results in an invalid motion vector, the motion hypothesis is marked as not available. In order to prefer temporal prediction, the order in which is reference indices are tested is reversed if the first index refers to an inter-view reference picture. The number of motion hypotheses for the inter-view motion parameter set is given by the number of available motion hypotheses. If all potential motion hypotheses are marked as not available, the inter-view candidate cannot be selected.

2.2.4 Depth-based motion parameter prediction (integration tool)

Depth-Based Motion Prediction (DBMP) is a new coding tool for multiview video coding which originates from the idea that motion fields of neighboring views in multiview sequence are highly correlated. DBMP provides an efficient representation of motion data in multiview video bitstreams that carry also depth/disparity maps. The motion information, such as motion vectors and reference indices, for each pixel of encoded coding unit (CU) is directly inferred with use of already coded disparity maps from encoded CUs in the neighboring views at the same temporal instance (Figure 11). This procedure is repeated

independently for every pixel of encoded CU. Consequently, motion vectors and reference indices for CU are not transmitted in the bitstream but are obtained from the reference view at the receiving side.

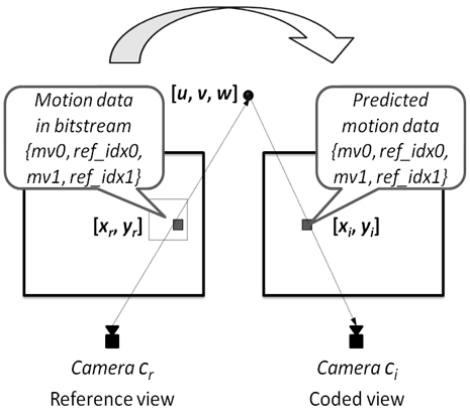

Figure 11: Independent derivation of motion information for each point of encoded CU from corresponding point in reference view.

2.2.5 Inter-view residual prediction

The basic principle of the inter-view residual prediction is illustrated in Figure 12. Similarly as for the inter-view motion prediction, the inter-view residual prediction is based on a depth map estimate for the current picture. The same depth map estimate as for the inter-view motion prediction is used. Depending on the encoder configuration, the depth map estimate is derived by one of the two methods described in sec. 2.2.3.1. Based on the depth map estimate, a disparity vector is determined for a current block and the residual block in the reference view that is referenced by the disparity vector is used for predicting the residual of the current block.

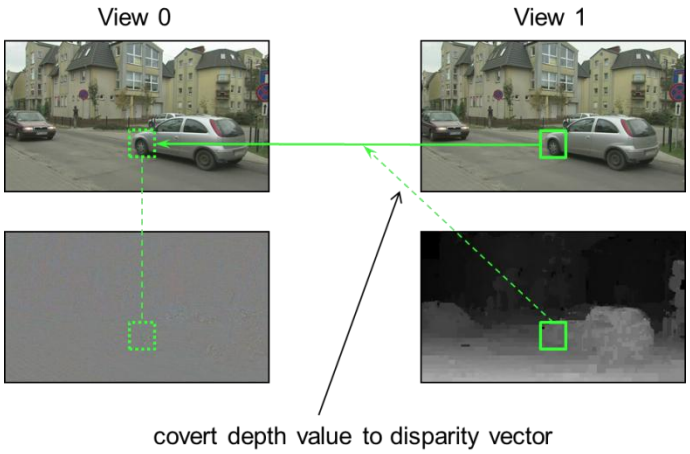

Figure 12: Basic concept for the inter-view residual prediction.

A more detailed illustration of the concept for deriving a reference block location inside the reference view is given in Figure 13. Inside the current block, a sample location ***x*** in the middle of the block is selected and the associated depth value *d* is converted to a disparity vector. The disparity vector is added to the location of the top-left sample of the current block yielding the location of the top-left sample of the reference block. Then, similar as for motion compensation,

the block of residual samples in a reference view that is located at the derived reference location is subtracted from the current residual and only the resulting difference signal is transform coded. If the disparity vector points to a sub-sample location, the residual prediction signal is obtained by interpolating the residual samples of the reference view using a bi-linear filter.

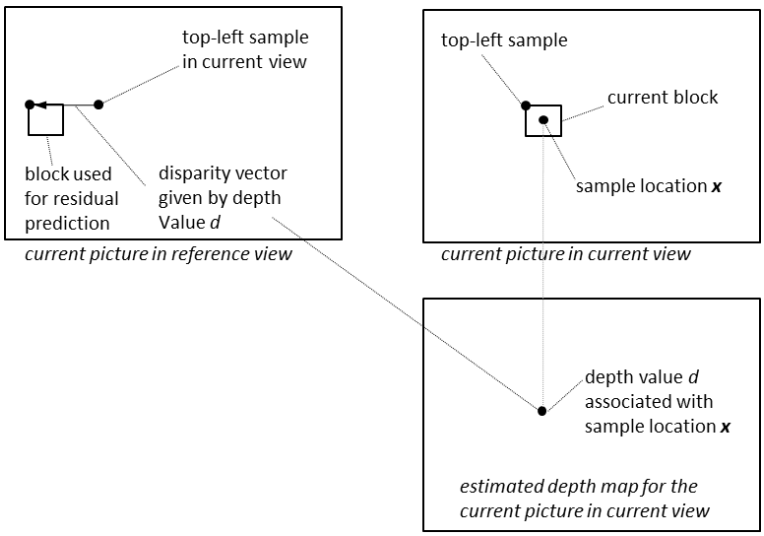

Figure 13: Derivation of the location of reference residual block.

The usage of the inter-view residual prediction can be adaptively selected on a block basis, or more accurately on a coding unit (CU) basis. For that purpose, if any sample of the potential reference residual signal is unequal to 0, a flag indicating the usage of inter-view residual prediction is transmitted as part of the CU syntax. If this flag is equal to 1, the current residual signal is predicted using the potentially interpolated reference residual signal and only the difference is transmitted using transform coding. Otherwise, the residual of the current block is conventionally coded using the HEVC transform coding.

2.2.6 Adjustment of QP of texture based on depth data (integration tool)

In order to improve perceptual quality of coded texture, a tool for bit assignment in the texture layer was developed. The basic idea is to increase texture quality of objects in the foreground and to increase compression factor (decrease texture quality) for objects in the background. The quality is adjusted in coding units (CUs) with use of quantization parameter QP that depends on the corresponding depth values. The QP adjustment is done simultaneously in coder and decoder so that no additional information is send. Described tool is disabled in the base view to preserve HEVC compatibility. The texture QP is modified in the following way:

$$QP' = QP - 2.6 + 8 \cdot \left(\frac{255 - \max\limits_{x,y \in CU} d_{x,y}}{256} \right)^2$$

Where QP' is adjusted QP value for a CU with corresponding disparity $d_{x,y}$.

2.3 Coding of Depth Maps

For the coding of depth maps, basically the same concepts of intra-prediction, motion-compensated prediction, disparity-compensated prediction, and transform coding as for the coding of the video pictures are used. However, some tools have been modified for depth maps, other tools have been generally disabled, and additional tools have been added.

As a first difference to the coding of video pictures, the inter-view motion and residual prediction as described in sec. 2.2.2 and sec. 2.2.4, respectively, are not used for depth coding. Instead, motion parameters are derived based on coded data in the associated video pictures as will be described in sec. 2.3.7 below. The other differences are described in the following subsections.

2.3.1 Disabled chrominance coding (integration tool)

Depth maps may be coded in 4:0:0 chroma sampling format.

2.3.2 Non-linear depth representation (integration tool)

As alternative representation of depth maps, the depth may be non-linearly scaled as described in the following.

The human perception of depth depends on absolute distance of viewed objects, therefore the internal depth representation is non-linear. Closer objects are represented more accurately than distant ones. Thanks to that, subjective quality of synthesized views is improved.

Internal depth sample values are defined by the following power-law expressions, similar as in the case of well known gamma correction:

$$depth\ value\ internal = \left(\frac{depth\ value\ external}{maximum\ value\ external}\right)^{exponent} \cdot maximum\ value\ internal$$

$$depth\ value\ external = \left(\frac{depth\ value\ internal}{maximum\ value\ internal}\right)^{1/exponent} \cdot maximum\ value\ external$$

Exponent is automatically chosen by the encoder with use of base QP for the depth and sent to decoder in the encoded bitstream:

$$exponent = clip\left(\left(QP_{depth} - 30\right) \cdot 0.0125 + 1.25\ ;\ 1.0\ ;\ 1.66\right)$$

Depth map samples are represented on increased number of bits with use of IBDI (Internal Bit Depth Increase) tool.

2.3.3 Z-near z-far compensated weighted prediction (integration tool)

Proposed znear-zfar compensation (ZZC) is a new coding tool for multiview video, designed especially for inter-frame depth map coding.

The concept of ZZC exploits the observation that frames from different views and time instances of encoded depth sequence may have different z_{near} and zfar parameters. The mentioned znear and zfar parameters describe range of depths represented in a gray-scale depth map. If znear and

zfar parameters are different for two frames, then given depth value is represented with different gray-scale values in those depth maps. Consequently, using one of such depth maps as a reference for the other one will result in a poor prediction.

To overcome this problem, a new ZZC coding tool is proposed. Prior to any inter-frame depth map prediction, each depth map that resides on the codec reference picture list is scaled, so that gray-scale depth values in scaled image and currently coded image refer to the same depth. As a result, depth maps with compensated z_{near} and z_{far} range are used for prediction.

Values used for prediction (instead of the original ones) are calculated as follows:

$$L_T = L_S \cdot \frac{z_{far\,S} - z_{near\,S}}{z_{far\,T} - z_{near\,T}} + 255 \cdot \frac{z_{near\,S} - z_{near\,T}}{z_{far\,T} - z_{near\,T}}$$

Where L_T is compensated disparity in range depth $z_{near\ T}$ to $z_{far\ T}$ and L_S is original disparity in depth range $z_{near\ S}$ and $z_{far\ S}$.

2.3.4 Modified motion compensation and motion vector coding

In contrast to natural video, depth maps are characterized by sharp edges and large regions with nearly constant values. The eight-tap interpolation filters that are used for motion-compensated interpolation in HEVC, can produce ringing artifacts at sharp edges in depth maps, which are visible as disturbing components in synthesized intermediate views. For avoiding this issue and for decreasing the encoder and decoder complexity, the motion-compensated prediction (MCP) as well as the disparity-compensated prediction (DCP) has been modified in a way that no interpolation is used. That means, for depth maps, the inter-picture prediction is always performed with full-sample accuracy. For the actual MCP or DCP, a block of samples in the reference picture is directly used as prediction signal without interpolating any intermediate samples. In order to avoid the transmission of motion and disparity vectors with an unnecessary accuracy, full-sample accurate motion and disparity vectors are used for coding depth maps. The transmitted motion vector differences are coded using full-sample instead of quarter-sample precision.

2.3.5 Disabling of in-loop filtering

The in-loop filters in the HEVC design have been particularly designed for the coding of natural video. For the coding of depth maps, these filters are less useful. In order to decrease the encoder and decoder complexity, the in-loop filters have been disabled for depth coding. This includes the following filters:

- the de-blocking filter;
- the adaptive loop filter (Wiener filter);
- the sample-adaptive loop filter.

2.3.6 Depth modeling modes

Depth maps are mainly characterized by sharp edges (which represent object borders) and large areas of nearly constant or slowly varying sample values (which represent object areas). While the HEVC intra prediction and transform coding is well-suited for nearly constant regions, it can result in significant coding artifacts at sharp edges, which are visible in synthesized intermediate views. For a better representation of edges in depth maps, four new intra prediction modes for depth coding are added. In all four modes, a depth block is approximated by a model that partitions the area of the block into two non-rectangular regions, where each region is represented by a constant value. The information required for such a model consists of two elements, namely the partition information, specifying the region each sample belongs to, and

the region value information, specifying a constant value for the samples of the corresponding region. Such a region value is referred to as constant partition value (CPV) in the following. Two different partition types are used, namely Wedgelets and Contours, which differ in the way the segmentation of the depth block is derived. The depth modeling modes are integrated as an alternative to the conventional intra prediction modes specified in HEVC. Similar as for the intra prediction modes, a residual representing the difference between the approximation and the original depth signal can be transmitted via transform coding. In the following, the approximation of depth blocks using the four new depth modeling modes is described in more detail.

It is differentiated between Wedgelet and Contour partitioning. For a Wedgelet partition, the two regions are defined to be separated by a straight line, as illustrated in Figure 14, in which the two regions are labeled with P_1 and P_2. The separation line is determined by the start point S and the end point P, both located on different borders of the block. For the continuous signal space (see Figure 14, left), the separation line can be described by the equation of a straight line. The middle image of Figure 14 illustrates the partitioning for the discrete sample space. Here, the block consists of an array of samples with size $u_B \times v_B$ and the start and end points correspond to border samples. Although the separation line can be described by a line equation as well, the definition of regions P_1 and P_2 is different here, as only complete samples can be assigned as part of either of the two regions. For employing Wedgelet block partitions in the coding process, the partition information is stored in the form of partition patterns. Such a pattern consists of an array of size $u_B \times v_B$ and each element contains the binary information whether the corresponding sample belongs to region P_1 or P_2. The regions P_1 and P_2 are represented by black and white samples in Figure 14 (right), respectively.

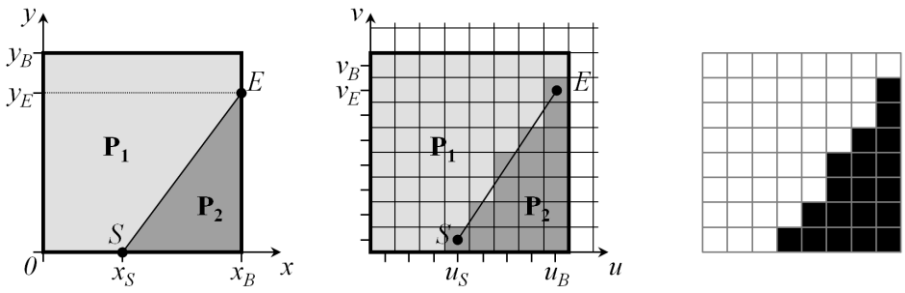

Figure 14: Wedgelet partition of a block: continuous (left) and discrete signal space (middle) with corresponding partition pattern (right).

Unlike for Wedgelets, the separation line between the two regions of a Contour partition of a block cannot be easily described by a geometrical function. As illustrated in Figure 15, the two regions P_1 and P_2 can be arbitrary shaped and even consist of multiple parts. Apart from that the properties of Contour and Wedgelet partitions are very similar. For employing Contour partitions in the coding process, the partition pattern (see example in Figure 15, right) is derived individually for each block from the signal of a reference block. Due to the lack of a functional description of the region separation line, no pattern lookup lists and consequently no search of the best matching partition are used for Contour partitions.

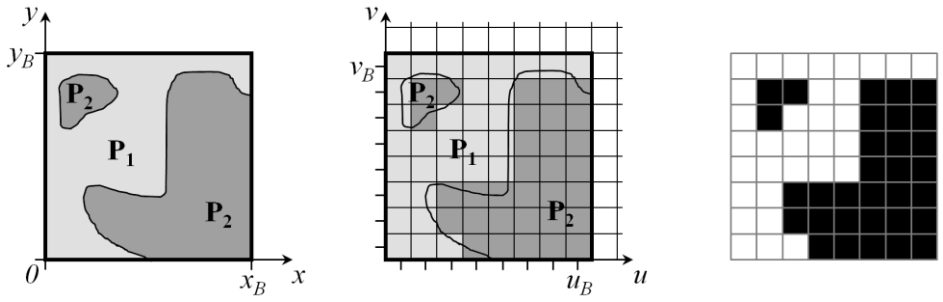

Figure 15: Contour partition of a block: continuous (left) and discrete signal space (middle) with corresponding partition pattern (right).

Apart from the partition information, either in form of a Wedgelet or a Contour partition, the second information required for modeling the signal of a depth block is the CPV of each of the two regions. For a given partition the best approximation is consequently achieved by using the mean value of the original depth signal of the corresponding region as the CPV.

Four depth-modeling modes, which mainly differ in the way the partitioning is derived and transmitted, have been added:

- Mode 1: Explicit Wedgelet signaling;
- Mode 2: Intra-predicted Wedgelet partitioning;
- Mode 3: Inter-component-predicted Wedgelet partitioning;
- Mode 4: Inter-component-predicted Contour partitioning.

These depth-modeling modes as well as the signaling of the modes and the constant partition values are described in the following four subsections.

2.3.6.1 Mode 1: Explicit Wedgelet Signalization

The basic principle of this mode is to find the best matching Wedgelet partition at the encoder and transmit the partition information in the bitstream. At the decoder the signal of the block is reconstructed using the transmitted partition information.

The Wedgelet partition information for this mode is not predicted. At the encoder, a search over a set of Wedglet partitions is carried out using the original depth signal of the current block as a reference. During this search, the Wedgelet partition that yields the minimum distortion between the original signal and the Wedgelet approximation is selected. The resulting prediction signal is then evaluated using the conventional mode decision process.

A fast search of the best matching partition is essential for employing Wedgelet models in the depth coding process. For this purpose, the patterns for all possible combinations of start and end point positions are generated and stored in a lookup table for each block size prior to the coding process. The Wedgelet pattern list contains only unique patterns. The resolution for the start and end positions used for generating the Wedgelet patterns depends on the block size. For 16x16 and 32x32 blocks, the possible start and end positions are restricted to locations with an accuracy of 2 samples. For 8x8 blocks, full-sample accuracy is used, and for 4x4 blocks, half-sample accuracy is used.

2.3.6.2 Mode 2: Intra-predicted Wedgelet Partitions

The basic principle of this mode is to predict the Wedgelet partition from data of previously coded blocks in the same picture, i.e. by intra-picture prediction. For a better approximation, the predicted partition is refined by varying the line end position. Only the offset to the line end

position is transmitted in the bitstream and at the decoder the signal of the block is reconstructed using the partition information that results from combining the predicted partition and the transmitted offset.

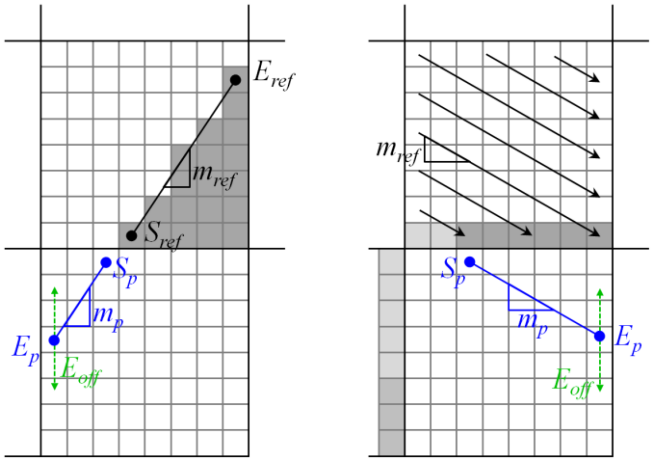

Figure 16: Intra prediction of Wedgelet partition (blue) for the scenarios that the above reference block is either of type Wedgelet partition (left) or regular intra direction (right).

The prediction process of this mode derives the line start position and the gradient from the information of previously coded blocks, i.e. the neighbor blocks left and above of the current block. Note that for some blocks one or both of the neighboring blocks are not available. In such a case the processing for this mode is carried out with setting the missing information to meaningful default values. As illustrated in Figure 16 two main prediction methods have to be distinguished: The first method covers the case when one of the two neighboring reference blocks is of type Wedgelet, shown in the example in Figure 16, left. The second method covers the case when the two neighboring reference blocks are not of type Wedgelet, but of type intra direction, which is the default intra coding type, shown in the example in Figure 16, right.

If the reference block is of type Wedgelet, the prediction process works as follows: The principle of this method is to continue the reference Wedgelet into the current block, which is only possible if the continuation of the separation line of the reference Wedgelet actually intersects the current block. Therefore, it is first checked whether it is possible to continue the reference Wedgelet. In case the check is positive, the start position S_p and the end position E_p are predicted by calculating the intersection points of the continued line with block border samples.

If the reference block is of type intra direction, the prediction process works as follows: First, the gradient is derived from the intra prediction direction. As the intra direction is only provided in the form of an abstract index, a mapping or conversion function is defined that associates each intra prediction mode with a gradient. Second, the start position S_p is derived from information that is also available at the decoder, namely the adjacent samples of the left and above neighboring block, by selecting the sample position with the maximum slope. Finally, the end position E_p is calculated from the start point and the gradient.

The line end position offset for refining the Wedgelet partition is not predicted, but searched within the estimation process at the encoder. For the search, candidate partitions are generated from the predicted Wedgelet partition and an offset value for the line end position E_{off}, as illustrated in Figure 16. By iterating over a range of offset values and comparing the distortion of the different resulting Wedgelet partitions, the offset value of the best matching Wedgelet partition is determined using a distortion measure.

2.3.6.3 Mode 3: Inter-component prediction of Wedgelet partitions

The basic principle of this mode is to predict the Wedgelet partition from a texture reference block, namely the co-located block of the associated video picture. This type of prediction is referred to as inter-component prediction. Unlike temporal or inter-view prediction, no motion or disparity compensation is used, as the texture reference picture shows the scene at the same time and from the same perspective. The Wedgelet partition information is not transmitted for this mode and consequently, the inter-component prediction uses the reconstructed video picture as a reference. For efficient processing, only the luminance signal of the reference block is taken into account, as this typically contains the most significant information for predicting the partition of a depth block, i.e. the edges between objects.

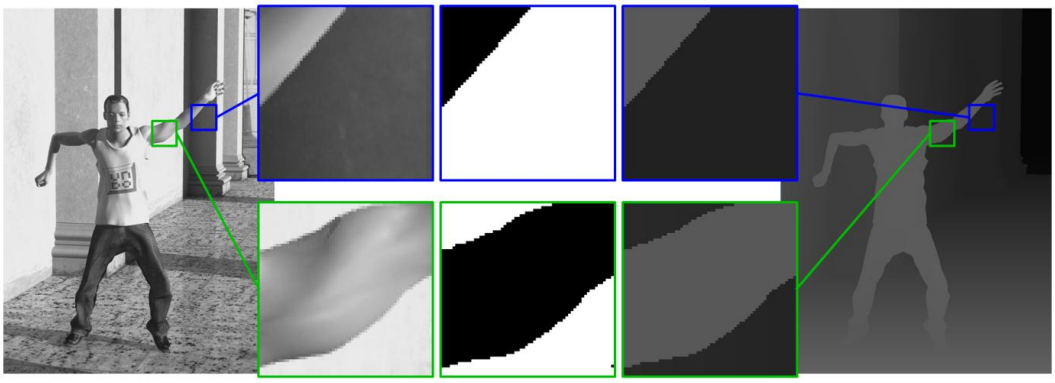

Figure 17: Prediction of Wedgelet (blue) and Contour (green) partition information from texture luma reference.

The prediction of a Wedgelet partition pattern from the texture reference is illustrated in the top row of Figure 17. For this purpose, a search over the set of possible Wedgelet partitions is carried out. The Wedgelet partition that yields the smallest distortion for the co-located texture block is used for approximating the current depth block.

2.3.6.4 Mode 4: Inter-component prediction of Contour partitions

The basic principle of this mode is to predict a Contour partition from a texture reference block by inter-component prediction. Like for the inter-component prediction of a Wedgelet partition pattern, the reconstructed luminance signal of the co-located block of the associated video picture is used as a reference, as illustrated in the bottom row of Figure 17. In contrast to Wedgelet partitions, the prediction of a Contour partition is realized by a thresholding method. Here, the mean value of the texture reference block is set as the threshold and depending on whether the value of a sample is above or below the sample position is marked as part of region P_1 or P_2 in the resulting Contour partition pattern.

2.3.6.5 Constant partition value coding

The method for CPV coding is the same for all four modes introduced above, as it does not distinguish between partition types, but rather assumes that a partition pattern is given for the current depth block. As illustrated in Figure 18, three types of CPVs are differentiated: original, predicted, and delta CPVs.

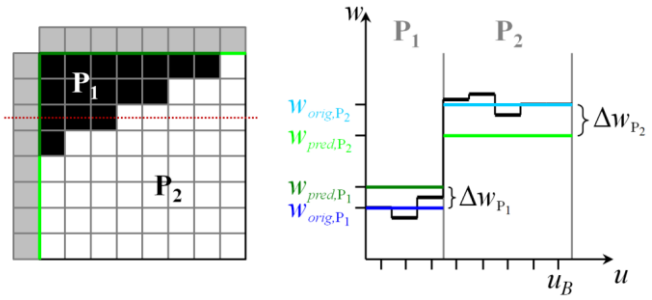

Figure 18: CPVs of block partitions: CPV prediction from adjacent samples of neighboring blocks (left) and cross section of block (right), showing relation between different CPV types.

The cross section of the block in Figure 18, right, schematically shows that the original CPVs are calculated as the mean value of the signal covered by the corresponding region. Although these values lead to the best approximation for the given partition, they are not available at the decoder as they require the original signal. Therefore prediction of CPVs is introduced. These predicted CPVs are derived from information that is also available at the decoder, namely the adjacent samples of the neighboring left and top block, as illustrated in Figure 18, left, where the green and light green line segments highlight the mapping of adjacent samples to the partitions. Again, the predicted CPVs are calculated as the mean value of the corresponding sample values. Depending on the similarity between original signal of the block and adjacent samples, the predicted and original CPVs may differ significantly. This difference is referred to as delta CPVs. By calculating the delta CPVs at the encoder and transmitting them in the bit stream, it is possible to reconstruct the CPVs at the decoder.

Although the distortion of the reconstructed signal is considerably reduced by the delta CPVs, the benefit of this approach is delimited by the additional bit rate required for transmitting the delta CPVs. Therefore, a linear quantization is introduced for the delta CPVs. This method is also used in transform coding and the step size of the quantization is set as a function of the QP. The delta CPVs are linearly quantized at the encoder and de-quantized before reconstruction at the decoder.

In case the distortion is not measured for the original depth, but for synthesized views, the delta CPV derivation process is extended by a minimum distortion search, which iterates over all possible delta CPV combinations for the two partitions. For the sake of efficient processing and signaling the range of tested values is limited. The search results in the combination of delta CPVs that causes the minimum distortion in synthesized views and for transmission these values are finally quantized.

2.3.6.6 Mode pre-selection

In the encoding process, for an intra-coded CU, one of the described depth modeling modes or one of the conventional intra prediction modes is selected. If a depth modeling mode is selected, the selected mode and the associated prediction data have to be signaled in the bitstream in addition to a syntax element that specifies the usage of a depth modeling mode. The following four depth modeling modes are defined:

- *Wedgelet_ModelIntra*: Intra modeling of Wedgelet block partition
- *Wedgelet_PredIntra*: Intra prediction of Wedgelet block partition
- *Wedgelet_PredTexture*: Inter-component prediction of Wedgelet block partition
- *Contour_PredTexture*: Inter-component prediction of Contour block partition

Each of the four modes can be applied with or without delta CPVs, resulting in eight different mode_IDs for signaling the decoder, which type of processing has to be applied for prediction and reconstruction of the block.

In order to reduce the encoder processing and signaling effort for block partition coding, a mode pre-selection is implemented, which excludes depth modeling modes that are very unlikely to be selected for the current block. One pre-selection method is to disable modes whose probability is very low for small block sizes. Therefore Wedgelet_PredIntra and Contour_PredTexture are disabled for block sizes smaller than 8×8. A second pre-selection method applies to the modes based on inter-component prediction, namely Wedgelet_PredTexture and Contour_PredTexture. This method adaptively disables the modes, if it is very unlikely that a meaningful partition pattern can be derived from the texture reference. Such blocks are characterized by having relatively constant pixel values without significant edges or contours, as illustrated by the upper example in Figure 19. For identifying them, their statistical dispersion is analyzed, namely the mean absolute deviation (MAD) of the luminance signal of the texture reference block. If the MAD value is below a certain threshold, the modes are disabled. Instead of a fixed value, the threshold is set as a function of the QP, which has the effect, that for higher QP values these two modes are excluded more frequently.

Figure 19: Mode preselection based on texture luma variance.

2.3.6.7 Signaling in the bitstream

The depth modeling modes are implemented as an additional set of block coding modes into the intra path of the 3D video codec. Therefore, an additional flag prior to the mode information is transmitted in the bitstream, signaling whether a block partition mode is used or not. In case this flag is not set, normal intra mode signaling follows. Otherwise, a mode_ID is signaled, which specifies the actual block partition mode and if delta CPVs are also transmitted or not. The number of bins required depends on the decision of the mode pre-selection methods, ranging from three bins, if all eight modes are enabled, to one bin, if the number of modes is reduced to two due to pre-selection decisions described in sec. 2.3.6.6.

Mode *Wedgelet_ModelIntra*: For this mode the Wedgelet partition information is explicitly signaled in the bitstream by the index of the corresponding pattern in the Wedgelet pattern lookup list. The index is signaled with a fixed number of bins. The number of bins used for transmitting the index is given by the size of the list of possible Wedgelet patterns.

Mode *Wedgelet_PredIntra*: For this mode only the refinement of the Wedgelet partition in terms of the line end position offset is signaled in the bitstream. A first bin indicates whether the offset is zero or not. If the offset is not zero, $k + 1$ additional bins follow for signaling offset

values in the range $\pm 2^k$, where the first bin represents the sign and the remaining k bins the absolute value of the offset. k is set equal to 2.

Mode *Wedgelet_PredTexture*: For this mode no additional signaling regarding the partition information is required.

Mode *Wedgelet_PredTexture*: For this mode no additional signaling regarding the partition information is required.

Delta CPVs: In case the delta CPVs are transmitted (which is signaled by the transmitted mode_ID), the two quantized values are signaled in the bitstream consecutively. For each CPV, a bin string consisting of the absolute value and the sign is transmitted. The sign is coded as a single bin, and the absolute value is coded using a truncated unary code (with 13 bins in the unary part and an exponential golomb code suffix).

2.3.7 Motion parameter inheritance

The basic idea behind the motion parameter inheritance (MPI) mode is that the motion characteristics of the video signal and its associated depth map should be similar, since they are both projections of the same scenery from the same viewpoint at the same time instant. Therefore, in order to enable efficient encoding of the depth map data, a new coding mode that allows inheritance of the treeblock subdivision into CUs and PUs and their corresponding motion parameters from the video signal has been introduced. Since the motion vectors of the video signal have quarter-sample accuracy, whereas for the depth map signal only full-sample accuracy is used, in the inheritance process the motion vectors are quantized to their nearest full-sample position. It can be adaptively decided for each block of the depth map, whether the motion data are inherited from the co-located region of the video signal or if new motion data are transmitted (cp. Figure 20). For signaling the MPI coding mode, the merge/skip mode syntax is used. The list of possible merge candidates has been extended in a way that, for depth map coding, the first merge candidate refers to merging with the corresponding block from the associated video signal.

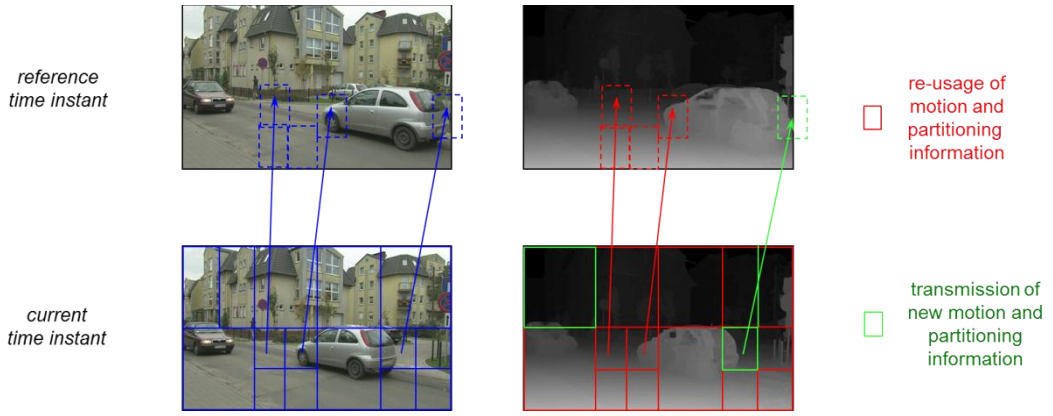

Figure 20: Illustration of the concept of motion parameter inheritance.

Independent of the partitioning of the video picture into its CUs, the MPI mode can be used at any level of the treeblock hierarchy for the depth map. If the MPI mode is indicated at a higher level of the depth map coding tree, corresponding to a CU size that is larger than the CU size that is used for the video signal, the CU/PU subdivision, together with the corresponding motion data, is inherited from the video signal. This makes it possible to specify once for a whole treeblock, typically corresponding to 64 × 64 image samples, that the whole partitioning of this

region into its CUs and PUs for the video signal is also applied to the depth map signal. In the other case, if MPI is indicated at a level of the coding tree that corresponds to the same or a smaller CU size than the CU size that is used for the video signal, only the motion data are inherited from the video signal. Since, when using MPI, not only the partitioning and the motion vectors, but also the reference picture indices are inherited from the video signal, it has to be ensured, that the depth maps that correspond to the video reference pictures are also available in the reference picture buffer for the depth map signal. The MPI mode is only possible, if the whole region of the video signal, that the motion data and partitioning are to be inherited from, is coded using inter prediction.

2.4 Encoder Control

For mode decision and motion estimation, a Lagrangian technique by which a cost measure $D + \lambda \cdot R$ is determined for each candidate mode or parameter, and the mode or parameter with the smallest cost measure is selected. D is the distortion that is obtained by coding the considered block in a particular mode or with a particular parameter, R is the number of bits that are required for representing a block in a given mode or that are required for coding a given parameter, and λ is the Lagrangian multiplier that is derived based on the used quantization parameter. As measure for the distortion, the sum of squared differences (SSD) or the sum of absolute differences (SAD) between the original and the reconstructed sample values is used (for the coding of depth maps this measure was modified as described below).

For the coding of depth maps, basically the same decision process is used. However, the distortion measure has been replaced with a measure that considers the distortion in synthesized intermediate views. This technique is described in the following subsection.

2.4.1 View Synthesis Optimization

The geometry information given by depth data is exploited only indirectly in the rendering process. Hence, the lossy coding of depth data causes distortions in the synthesized intermediate views. The depth map itself is not visible for a viewer. The efficiency of depth coding is improved by considering this property. As a consequence, the distortion measure for the mode decision process for depth maps is modified in a way that the synthesized view distortion is used instead of the depth map distortion. Therefore, the synthesized view distortion change (SVDC) is used as distortion measure.

The computation of the SVDC requires the usage of rendering functionalities in the encoding process. Since computational complexity is a critical factor in distortion calculation, a method, which is also referred to as renderer model, has been utilized that allows minimal re-rendering of parts of the synthesized view that are affected by a depth distortion. For this, a special renderer is included in the encoder, which supports the basic functionalities, shared by most rendering approaches, like sub-pixel accurate warping, hole filling and view blending.

2.4.1.1 Synthesized View Distortion Change (SVDC)

Since the encoding algorithm operates block-based, the mapping of depth distortion to the synthesized view distortion must be block-based as well. Moreover, the sum of partial distortions (of sub-blocks) must be equal to the overall distortion of a block in order to enable an independent distortion calculation for all partitions of a subdivided block, as hierarchical block structures are used in HEVC.

A relationship between a depth map s_D and a synthesized texture s'_T is created by the used view synthesis approach. However, disocclusions and occlusions prevent a bijective mapping of the

distorted areas in depth maps to distorted areas in the synthesized views. For example, areas in the synthesized view, which depend on depth data of a considered block, can become visible due to the distortions in other depth blocks; or vice versa, the distortion of a depth block has no effect on the synthesized view, since the block is occluded there. Hence, an exact mapping between the distortion of a block of the depth data and an associated distortion in the synthesized view is not possible considering only the depth data within a currently processed block.

For resolving this issue, the change of the overall distortion in a synthesized view depending on the change of the depth data within a block B is determined, while simultaneously also considering depth data outside the block B. For this purpose, the synthesized view distortion change (SVDC) is defined as distortion difference ΔD between two synthesized textures s'_T and $\tilde{s}_T$,

$$\Delta D = \widetilde{D} - D = \sum_{(x,y)\in I} \left[\tilde{s}_T(x,y) - {s'}_{T,Ref}(x,y)\right]^2 - \sum_{(x,y)\in I} \left[s'_T(x,y) - {s'}_{T,Ref}(x,y)\right]^2 \qquad (1)$$

${s'}_{T,Ref}$ denotes a reference texture rendered from original video and depth data. I represents the set of all samples in the synthesized view. To illustrate how the textures s'_T and $\tilde{s}'_T$ are obtained, the SVDC definition from eq. (1) is also depicted in Figure 21. s'_T denotes a texture rendered from a depth map s_D consisting of encoded depth data in already encoded blocks and original depth data in the other blocks. The current block B, for which the distortion has to be computed, contains original depth data as well. For the synthesis of the texture $\tilde{s}'_T$ a depth map $\tilde{s}_D$ is used that differs from the depth map s_D in that it contains the distorted depth data also for the current block B.

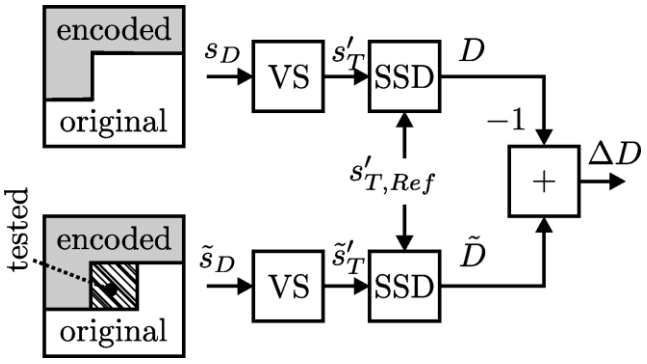

Figure 21: Definition of the SVDC related to the distorted depth data of the block *B* depicted by the hatched area in the bottom branch; VS denotes the view synthesis step and SSD stands for sum of squared differences.

The SVDC definition above is motivated by three reasons. First, an exact distortion measure is provided, therefore the overall distortion of the synthesized view and thereby disocclusions and occlusions are considered. Second, the measure is related to a block and third partial distortions are additive. For the latter two reasons, the change of the synthesized view distortion caused by a change of a depth block is employed instead of the total synthesized view distortion itself.

Figure 21 shows the SVDC definition for the extrapolation of virtual views from one input view only. However, the encoder side view synthesis algorithm supports also the interpolation of the texture s'_T from a left and a right view. Hence, rendering requires a left $s_{D,l}$ and a right $s_{D,r}$ depth map. To extend the SVDC computation to this two view case, the original depth map of the second view can be used when encoding the first depth map. Subsequently the first already

encoded depth map can be utilized for the SVDC computation when encoding the second depth map.

2.4.1.2 Efficient Computation of the SVDC

A straightforward approach to compute the SVDC would be the direct implementation of eq. (1). However, this would require the complete rendering of the synthesized textures s'_T and $\tilde{s}'_T$ and a rendering of a whole view is computational too complex to be feasible in a rate-distortion optimization process. To overcome this problem, a method which enables a fast computation of the SVDC and is integrated in the encoder.

Renderer Model

The renderer model provides three basic functionalities to the encoder: Initialization, partial re-rendering, and SVDC calculation.

- The **initialization** of the renderer model is carried out before the encoding of a depth map is started. In the initialization process, the complete synthesized view is rendered using the original input depth maps and the input textures. The input depth maps are stored as the renderer models depth states $s_{D,l}$ and $s_{D,r}$ and the rendered view as the synthesized view state s'_T. Intermediate variables used in the rendering process are also stored to enable a fast re-rendering.
- **Partial re-rendering** is carried out to update the renderer model when the encoding of a block B is finished and the final depth data for the block is known. For this purpose, the reconstructed depth data and the position of block B are signaled to the renderer model. The renderer model changes the block in the depth state $s_{D,l}$ or $s_{D,r}$ from original to coded data and re-renders only local parts of the synthesized view state s'_T and the intermediate variables that are affected by the change of the depth data. Thus, the renderer model is transferred to a state that is required to compute the SVDC for blocks of the depth data encoded subsequently.
- For the **computation of the SVDC**, the position and the depth data of a block B to be tested are provided to the renderer model. The renderer model then computes the SVDC as defined in eq. (1). Here, re-rendering followed by the computation of the sum of squared distortions SSD is carried out. However, instead of considering all positions $(x, y) \in I$ again only positions affected by the depth change are considered. Note that the re-rendering carried out here does not modify any state variables of the renderer model. Hence, the SVDC can be computed for multiple depth candidates successively without the need to re-render with original data in block B.

Re-Rendering and Error Calculation Algorithm

The main objective of the algorithm is a computational low complex distortion calculation or state transition, hence a low complex re-rendering of the parts of the synthesized view that are affected by a depth change in one of the input depth maps.

Conventional view synthesis consists of multiple steps such as warping of the input samples, interpolation at sub pixel positions, blending with a second view obtained similarly, and hole filling. Typically these steps are executed as independent algorithms that are applied successively using the results of the previous step. To enable fast re-rendering of only parts of the synthesized view, all steps are combined in single algorithm that can be applied pixel-wise to the input depth map. This allows a region-wise processing of the depth map, and thus an update of related regions in the synthesized view.

This process is illustrated in Figure 22 for an example for rendering from a left view to the right. Rendering is applied row wise, hence all depicted signals represent one row of input, intermediate, or output data. The single signals are from bottom to top: the left input texture $s_{T,l}$, a shifting chart, the texture synthesized from left $s'_{T,l}$ the texture synthesized from right $s'_{T,r}$, the blended texture s'_T, and the reference texture $s'_{T,Ref}$. The arrows denote the relationship between the single samples or sample positions of the signals. Dots shown in the shifting chart represent samples from the input view. Their horizontal position is equal to their position x' in the synthesized view. The vertical position shows their disparities. Since the depth is monotonically decreasing with increasing disparity, the top-most samples in the chart are the samples closest to the camera. Hence, it can be seen from the shifting chart which samples are occluded in the synthesized view.

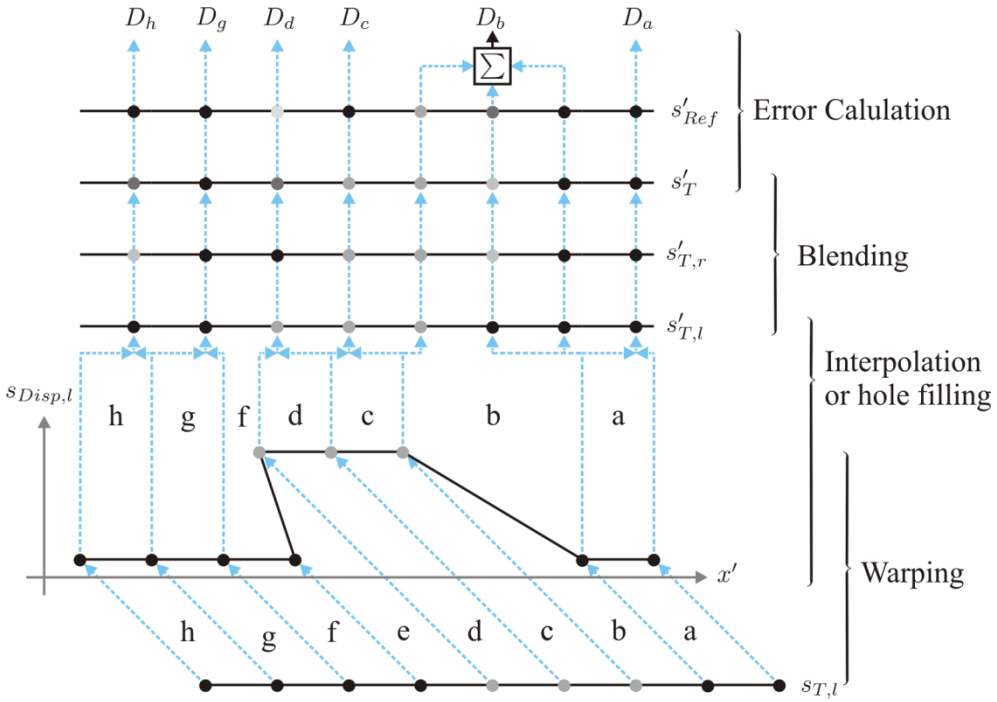

Figure 22: Example for the dependencies between input, intermediate and output signals of the rendering or error calculation step.

While a conventional view synthesis approach would carry out the single steps depicted from bottom to top for all samples in the intervals (a) to (g), the method supports an interval-wise processing. Hence, all steps are first conducted for interval (a) before continuing with interval (b). Re-rendering and error calculation are carried out by iterating only once over the input depth samples. If only the view synthesis distortion is calculated there is no need to store intermediate results in the state of the renderer model.

The boundaries of an interval in the output view are defined by the warped positions x'_s and x'_e of two neighboring input view samples at positions x_s and x_e. For warping, disparities are computed from the depth map as described in the beginning of sec. 4. Subsequently to the calculation of the interval boundaries, processing continues with interpolation, disocclusion handling, or occlusion handling:

- **Interpolation** is carried out in non-occluded ranges that are not disoccluded, as for example in the intervals (a,c,d,g,h). The accuracy of the warping is higher than the accuracy given by the sampling rate of synthesized view; hence an interpolation at the

full sample position x'_{FP} located between the interval boundaries x'_s and x'_e is carried out. For this, samples from an up-sampled version of the input texture $\hat{s}_{T,l}$ are mapped to the interpolation positions x'_{FP} in the synthesized view $s'_{T,l}$. The position $\hat{x}$ in the up-sampled view is derived from the distance of the interpolation position to the interval boundaries:

$$\hat{x} = 4 \cdot \left(\frac{x'_{FP} - x'_s}{x'_e - x'_s} + x_s \right) \tag{2}$$

The up-sampled view $\hat{s}_{T,l}$ is created in the initialization step by interpolating the input texture with quarter-sample accuracy using the FIR-filters specified for motion-compensated interpolation in HEVC.

- **Disocclusions**: If the width of the warped interval $x'_e - x'_s$ is greater than two times the width of the sampling distance, as for example for interval (b), a disocclusion is assumed in the synthesized view. Instead of interpolation, hole filling is carried. For this purpose, the samples in the interval are set equal to the value of the sample belonging to the right interval boundary $s_{T,l}(x_e)$ (which belongs to the background). If the leftmost full sample position within the interval is close to the left interval border, it is assumed that it belongs to the foreground and it is set equal to the value of the left interval boundary $s_{T,l}(x_s)$. Note, that the positions of disoccluded and filled samples are stored as additional information in the a filling map $s'_{F,l}$.
- **Occlusions:** Whether an interval is entirely occluded in the synthesized view, as for example interval (f), is determined by detecting if the interval boundaries are reversed ($x'_e < x'_s$), hence no complex z-buffering is required. To derive whether other samples left to interval (f) are occluded, the rendering process stores the position of the foreground edge. This stored position is then be utilized when processing the next intervals, for example interval (e), to determine which parts of theses intervals are occluded. If re-rendering does not start at the right image border, the position of the last foreground edge is recovered by carrying out a search to the right of the changed depth samples.

Sample values derived from interpolation or hole filling $s'_{T,l}$, are instantly combined with the texture sample values from a second view $s'_{T,r}$ synthesized the same way and stored as intermediate variable in the renderer model. The result is the sample value that is used in the final synthesized view s'_T.

The rendering model supports two different configurations. In the first configuration, a rendering process is considered that renders intermediate views using both surrounding actually coded views. The second configuration considers rendering processes by which an intermediate view is rendered mainly from one coded view; the other coded view is only used for rendering areas that are not present in the preferred coded view.

In the first configuration of the renderer model, the blending process is similar to that implemented in the VSRS software. Note that, although not depicted in Figure 22, a depth map $s'_{D,l}$ is rendered from $s_{D,l}$, when rendering $s'_{T,l}$, using full sample accuracy. This depth map is used in the blending step. The decision how blending is carried out depends on the filling of $s'_{F,l}$ or $s'_{F,r}$ and the rendered depth maps $s'_{D,l}$ and $s'_{D,r}$. While $s'_{F,l}$ and $s'_{D,l}$ have been obtained in the rendering process carried out before, $s'_{F,r}$ and $s'_{D,r}$ are stored as intermediate variables in the renderer model. The rules for determining the blended sample value $s'_T(x, y)$ from $s'_{T,l}(x, y)$ and $s'_{T,r}(x, y)$ are specified in the following:

- If the position (x, y) is disoccluded (as indicated by the filling map) in only one view, the sample value from the other view is used.
- Otherwise, if the position (x, y) is disoccluded in both views, the backmost sample value is used.
- Otherwise, if the depth difference retrieved from $s'_{D,l}(x, y)$ and $s'_{D,r}(x, y)$ is greater than a threshold, the front sample is used.
- Otherwise, a weighted average of $s'_{T,l}(x, y)$ and $s'_{T,r}(x, y)$, with a higher weight for the view that is closer to the virtual view position, is used.

For the second configuration of the renderer model, the intermediate view is mainly rendered from one view and only holes are filled from the other view. If assuming that $s'_{T,l}$ is the main view, the rules to determine the sample value $s'_T(x, y)$ from $s'_{T,l}(x, y)$ and $s'_{T,r}(x, y)$ are specified in the following:

- If $s'_{F,l}(x, y)$ indicates that there is no disocclusion at $s'_{T,l}(x, y)$, the sample value $s'_{T,l}(x, y)$ is used.
- Otherwise, if $s'_{F,r}(x, y)$ indicates that there is a disocclusion at $s'_{T,r}(x, y)$, the sample value $s'_{T,l}(x, y)$ is used.
- Otherwise, the average of $s'_{T,r}(x, y)$ and $s'_{T,l}(x, y)$ is used.

If only partial re-rendering is carried out, the result s'_T and all intermediate results are stored after the combination step and the processing of the interval is stopped. Otherwise, if the SVDC is determined, the distortion of the calculated value s'_T is computed by comparing it to the reference $s'_{T,Ref}$ in the next step.

To obtain the synthesized view distortion change the single intervals are rendered from right to left and the related distortions are summed up continuously. Moreover, and that is actually not depicted in Figure 22, the old per sample distortions of samples in the changed intervals are subtracted.

The renderer model only re-renders those parts of the synthesized view that are affected by the considered depth change. It has to be considered that in some cases not only the intervals related to the changed depth values must be re-rendered, but also some neighboring intervals. A reason is that neighboring intervals that are occluded before a depth change can become visible after the depth change. The algorithm detects such cases and continues rendering, until all change samples in the synthesized view are updated. The detection is carried out while warping by also considering the old shifted sample positions as they had been prior to the depth change and storing the left-most old position.

Chroma channels of the synthesized view are rendered together with the luma channel and are stored in the same resolution as luma. For this, up sampled versions of the chroma channels are created in the initialization step, which are later used for interpolation as described above. The sampling rate is increased by a factor of eight in horizontal direction and a factor of two in vertical direction using the interpolation FIR-filters that are specified for motion-compensated prediction in HEVC. However, the total distortion is obtained by a weighted sum of luma SVDC and chroma SVDC with a weight of 1 for luma and a weight of ¼ for each of the two chroma channels.

2.4.1.3 Integration of the Renderer Model in the Encoder Control

To enable rate-distortion optimization using the SVDC, the described renderer model is integrated in the encoding process for depth data. For this, the conventional distortion computation carried out is replaced with computation of the SVDC in all distortion computation steps related to the mode decision, coding unit (CU) partitioning, intra- and inter residual quadtree coding, motion parameter inheritance and merging. In order to reduce the computational complexity, the renderer model is not used for motion estimation or rate-distortion optimized quantization. Hence, only the inter prediction using the motion vector that has been obtained using the conventional motion search is compared to the intra and merge modes. Moreover, re-rendering is triggered in the renderer model when a final decision on the coding mode is taken within the encoder.

The usage of the synthesized view distortion in the rate-distortion decisions requires the adaptation of the Lagrange multiplier λ to obtain optimized coding results. This adaptation is carried out by adjusting the Lagrange multiplier using a constant factor. For this, the computation of rate-distortion cost J has been modified to

$$J = \Delta D + l_s \cdot \lambda \cdot \mathrm{R} = \Delta \mathrm{D} + \lambda_{\mathrm{l}} \cdot \mathrm{R} \qquad (3)$$

with ΔD denoting the change of global synthesized view distortion as provided by the renderer model, l_s denoting a constant scaling factor, and R denoting the rate for the current coding mode.

2.4.2 Optional Encoder Control for Renderable Regions in Dependent Views

As an optional encoding technique, a mechanism is integrated by which regions in dependent views that can be rendered based on the transmitted independent view and the associated depth maps are identified. These regions are encoded by employing a modified cost measure, which mainly considers the required bit rates. After decoding, the renderable regions can be identified in the same way as in the encoder and replaced by rendered versions.

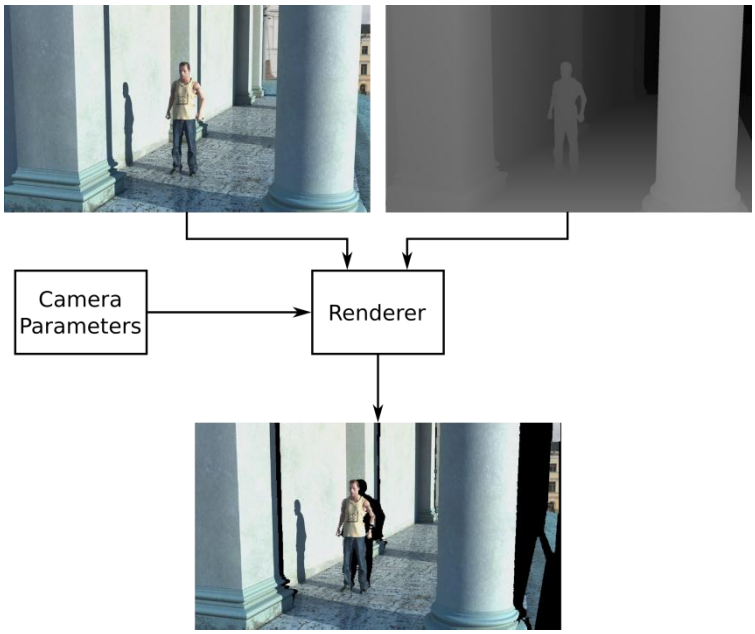

Figure 23: Rendering from a left camera position to a right camera position using depth maps.

The encoder identifies regions in the current frame that can be rendered from frames of the same time instance in a reference view based on the reconstructed depth maps of the reference view (see Figure 23). During the encoding process, the encoder checks for every CU, if all samples within that CU can be rendered. If all samples can be rendered, no residual is transmitted for this CU. In our HEVC-based codec, this means that for inter prediction the *no_residual_data_flag* for the CU is set equal to 1 or for intra-prediction the coded block flag of the TU's within the CU is set equal to 0. It should be noted that no syntax change is applied; only the encoder decision is modified.

Due to the quadtree structure in HEVC, the rate-distortion (RD-) costs are compared between different granularities of possible block subdivisions for the R-D optimization. Rendering artifacts have a different impact on the subjective image/video quality perception than coding artifacts and cannot be compared using conventional measurements, such as MSE or PSNR. Samples in renderable regions are not taken into account for calculating the distortion term in the R-D optimized encoder decisions. In Figure 24, the right image shows a block subdivision that is one level deeper than the ones in the left image. The gray area labels the samples that can be rendered and that are therefore not considered in the calculation of the distortion. Thus, for example, the upper left block in the right image is not considered at all. Hence, the costs being compared are $D_0 + \lambda R_0$ (left block subdivision) against $\mathrm{D}_1 + \lambda\,\mathrm{R}_1 + \mathrm{D}_2 + \lambda + \mathrm{D}_3 + \lambda\,\mathrm{R}_3 + \mathrm{D}_4 + \lambda\,\mathrm{R}_4$ (right block subdivision), where the distortions are only calculated based on the white shaded samples. E.g., the distortion of block B_1 is $D_1 = 0$. By this modification, blocks for which a subblock can be rendered are not automatically split, but also the entire block may be coded using a conventional coding mode if this improves the overall coding efficiency.

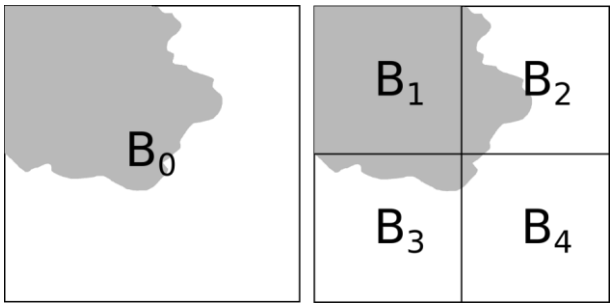

Figure 24: Distortion calculation on different tree depths. Renderable samples (gray shaded) are not taken into account.

For renderable blocks, the Lagrange multiplier λ is scaled by a factor $\mathrm{s} > 1$ and the calculation of the R-D costs is changed from $\mathrm{C} = \mathrm{D} + \lambda\mathrm{R}$ to $\mathrm{C} = \mathrm{D} + \mathrm{s}\lambda\mathrm{R}$.

2.4.3 Depth edge-based r-d optimization tuning (integration tool)

As alternative or in addition to the r-d optimization for depth maps as described in sec. 2.4.1, the following r-d optimization for depth coding is included. The purpose of this additional rd-opt is to reduce edge ringing artifacts in depth maps.

Normally cost is calculated as SAD between original and distorted (reconstructed) samples:

$$E_{x,y} = original_{x,y} - distorted_{x,y}$$

$$C = \sum_{x,y} |E_{x,y}|$$

$$C_{depth} = \sum_{x,y} |E_{x,y}| + \alpha \sum_{x,y} \left(H\{E_{x,y}\}\right)^2$$

Where $E_{x,y}$ is distortion, C is original distortion cost, C_{depth} is modified distortion cost for depth, and $H\{\cdot\}$ is a horizontal filter {1,-1} that detects vertical edges.

3 View Synthesis Algorithms

In the following, two view synthesis algorithms are described. Sec. 3.1 describes the fast 1-dimensional view synthesis algorithm that is part of the HEVC-based 3DV software. It is also referred to as "VSRS 1d fast mode". In sec. 3.2, an alternative view synthesis algorithm is described. This algorithm is also referred to as "VSRS" and was developed during the 3DV exploration experiments.

3.1 Fast 1-d View Synthesis (VSRS 1D Fast Mode)

An overview of the view synthesis method is depicted in Figure 25. The method supports the interpolation of a synthesized view form a left $s_{T,l}$ and right $s_{T,r}$ texture with corresponding depth maps $s_{D,l}$ and $s_{D,r}$. For this, two texture $s'_{T,l}$ and $s'_{T,r}$ are extrapolated from the left and the right view at the position of the virtual view. Subsequently, the similarity of $s'_{T,l}$ and $s'_{T,r}$ is enhanced before combining them to synthesized output view s'_T. The single processing steps are discussed in the following. Without the loss of generality steps carried out independently for both, the left and the right view, are discussed for the left view only.

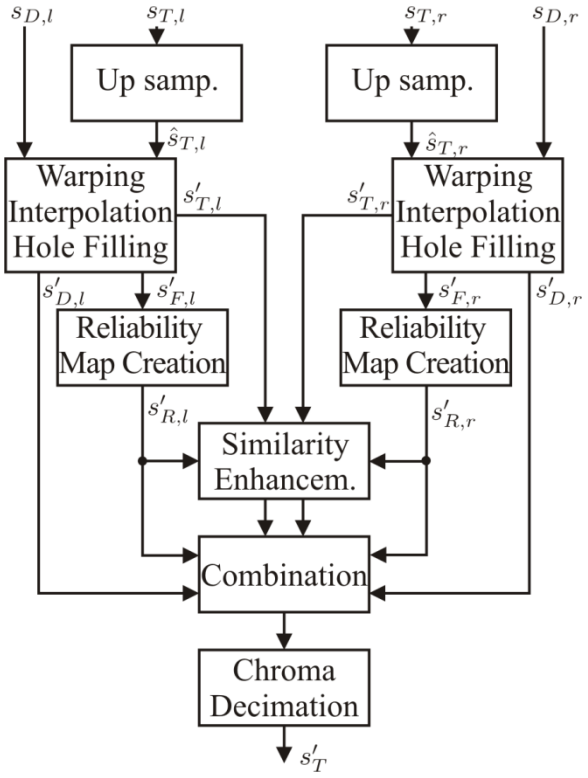

Figure 25: Processing steps of the view synthesis approach.

Similarly as the renderer model used in the encoder control (cp. sec. 2.4.1), the view synthesis algorithm supports two configurations. In the first configuration, which is referred to as interpolative rendering, an intermediate view is synthesized using both surrounding coded views. In the second configuration, which is referred to as non-interpolative rendering, an intermediate view is rendered mainly from one coded view; the other coded view is only used for rendering areas that are not present in the preferred coded view.

3.1.1 Upsampling of input video pictures

The luma channel of input texture $s_{T,l}$ is upsampled by a factor of four in horizontal direction. Chroma channels are upsampled by a factor of eight in horizontal direction and two in vertical direction. For upsampling, the FIR filters specified in HEVC for the purpose of motion-compensated interpolation are used. The resulting upsampled texture is denoted as $\hat{s}_{T,l}$.

3.1.2 Warping, interpolation and hole filling

Warping, interpolation and hole filling are carried out in a combined step. For warping disparities are computed as described in the beginning of sec. 2. Warping, interpolation and hole filling is carried out line wise and within a line interval wise. Processing direction is from left to right. An interval in the output view is defined by the warped positions x'_s and x'_e of two neighboring input view samples at positions x_s and x_e. Subsequently to the calculation of the interval boundaries, processing continues depending on the width of the interval.

- **Interpolation** is applied if the width of the warped interval $x'_e - x'_s$ is less than or equal to two times the sampling distance. An interpolation at the full sample position x'_{FP} located between the interval boundaries x'_s and x'_e is carried out. For this, samples from the up-sampled version of the input texture $\hat{s}_{T,l}$ are mapped to the interpolation positions x'_{FP} in the synthesized view $s'_{T,l}$. The position $\hat{x}$ in the up-sampled view is derived from the distance of the interpolation position to the interval boundaries:

$$\hat{x} = 4 \cdot \left(\frac{x'_{FP} - x'_s}{x'_e - x'_s} + x_s \right) \tag{4}$$

- **Disocclusions**: If the width of the warped interval $x'_e - x'_s$ is greater than two times the width of the sampling distance a disocclusion is assumed in the synthesized view. Instead of interpolation hole filling is carried. For this purpose samples in the interval are set to the value of sample belonging to the right interval boundary $s_{T,l}(x_e)$ (which belongs to the background). If the leftmost full sample position within the interval is close to the left interval border it is assumed that it belongs to the foreground and it is set to the value of the left interval boundary $s_{T,l}(x_s)$. Disoccluded and filled sample position are stored in the filling map $s'_{F,l}$.

- **Occlusions:** If the boundaries of an interval are reversed ($x'_e < x'_s$) the interval is occluded in the synthesized view. Rendering at a full sample position close to x'_e might be carried out, if the next interval is not occluded and x'_e belongs to a foreground object. Moreover, the algorithm uses the property that occluded background intervals are automatically overwritten by foreground objects in the synthesized view $s'_{T,l}$, due to the processing direction from left to right.

Chroma channels of the synthesized view are rendered together with luma channel and stored in the same resolution as luma. Moreover, if interpolative rendering is used, also a depth map $s'_{D,l}$ is

extrapolated with full sample accuracy from the input depth map $s_{D,l}$ within the steps described above.

3.1.3 Reliability map creation

In this step the filling map $s'_{F,l}$ is converted to the reliability map $s'_{R,l}$. If interpolative rendering is used, positions marked as disocclusions in $s'_{F,l}$ are mapped to a reliability of 0. In areas located right to a disocclusion with a width of six samples the reliability is linearly increased from 0 to 255 from left to right in horizontal direction. All other samples are assigned with a reliability of 255. If non-interpolative rendering is used, positions marked as disocclusions in $s'_{F,l}$ are mapped to a reliability of 0. All other samples are assigned with a reliability of 255.

3.1.4 Similarity enhancement

In this step the histogram of $s'_{T,l}$ is adapted to the histogram of $s'_{T,r}$. For this purpose a look up table (LUT) realizing a function f is created, that is subsequently applied to map the samples of $s'_{T,l}$ to adapt their values.

The function f and the corresponding LUT are obtained by approximately solving

$$h\left[f\left(s'_{T,l}\right)\right] = h\left[s'_{T,r}\right] \tag{5}$$

where $h[\ldots]$ denotes the histogram only regarding samples at positions (x, y) with reliabilities $s'_{R,l}(x, y)$ and $s'_{R,r}(x, y)$ of 255. Chroma channels are treated in the same way.

3.1.5 Combination

$s'_{T,l}$ and $s'_{T,r}$ are combined to obtain the synthesized output view in this step.

In the interpolative rendering mode is used, the decision how blending is carried out depends on the reliability maps $s'_{R,l}$ or $s'_{R,r}$ and the rendered depth maps $s'_{D,l}$ and $s'_{D,r}$. The rules for determining the blended sample value $s'_T(x, y)$ from $s'_{T,l}(x, y)$ and $s'_{T,r}\ (x, y)$ are given in the following:

- If position (x, y) is disoccluded (reliability of 0) in only one view, the sample value from the other view is used.
- Otherwise, if position (x, y) is disoccluded in both views, the backmost sample value is used.
- Otherwise, if the depth difference retrieved form $s'_{D,l}(x, y)$ and $s'_{D,r}(x, y)$ is above a threshold, the front sample is used.
- Otherwise, if one sample is not reliable with a value of 255, a weighted average with the given reliabilities as weights is used.
- Otherwise, a weighted average of $s'_{T,l}(x, y)$ and $s'_{T,r}\ (x, y)$ with a higher weight for the view that is closer to the virtual view position is used.

If the non-interpolative rendering mode is used, the intermediate view is mainly rendered from one view are utilized and only holes are filled from the other view. Assuming $s'_{T,l}$ is the main view, the rules for determining the sample value $s'_T(x, y)$ from $s'_{T,l}(x, y)$ and $s'_{T,r}\ (x, y)$ are given in the following:

- If $s'_{R,l}(x, y)$ is equal to 255 or $s'_{R,r}(x, y)$ is equal 0, the sample value $s'_{T,l}(x, y)$ is used.

- Otherwise, if $s'_{R,l}(x,y)$ is equal to 0, the sample value $s'_{T,r}(x,y)$ is used.
- Otherwise, a weighted average with the given reliabilities as weights is used.

3.1.6 Chroma decimation

To convert the 4:4:4 YUV representation obtained by rendering to the required 4:2:0 output, chroma channels are decimated by a factor of two in horizontal and vertical direction using the FIR filter (1;2;1).

3.2 VSRS (alternative view synthesis algorithm)

The VSRS algorithm was developed during the MPEG 3DV Exploration Experiments. VSRS takes two reference views and two depth maps as input to generate a synthesized virtual view. The intrinsic and extrinsic camera parameters are required and 1D parallel and non-parallel camera setups are supported.

The software has two main modes referred to as "General mode" and "1D mode". The reference views are reprojected to the target viewpoint using pixel-by-pixel mapping based on 3D warping in "General mode", or horizontal pixel shifting in "1D mode".

3.2.1 General mode

In the general mode, virtual views are generated by a technique referred to as "3D warping". This process involves two steps. At first the original view (reference view) is projected into 3D world space using the corresponding reference depth map. Then the 3D space points are projected into the image plane of the "virtual" view. For this, the intrinsic camera parameters A, and extrinsic camera parameters E=[R|t] are required. The intrinsic matrix A, transforms the 3D camera coordinates to its 2D image coordinates. The extrinsic matrix E=[R|t] transforms the world coordinates to camera coordinates, which is composed of rotation matrix R and translation vector t. The two-step warping can be formulated in two equations as in eq. (6) and (8). First a pixel (u_r, v_r) in the reference view is warped to the world coordinates (X_w, Y_w, Z_w), using the depth of the reference view:

$$\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} = R_{3\times 3}^{-1} \left(z_r A_{3\times 3}^{-1} \begin{pmatrix} u_r \\ v_r \\ 1 \end{pmatrix} - t_{3\times 1} \right) \tag{6}$$

where subscript r indicates the reference view and z_r is the depth value in the reference view at location (u_r, v_r) calculated from

$$z = \frac{1}{\frac{v}{255}\left(\frac{1}{Z_{near}} - \frac{1}{Z_{far}}\right) + \frac{1}{Z_{far}}} \tag{7}$$

where v is an 8-bit intensity of the depth map value. It is noted that the values z, Z_{near}, and Z_{far} are assumed to be either all positive or all negative values.

Then the 3D point is mapped to the virtual view:

$$z_v \begin{pmatrix} u_v \\ v_v \\ 1 \end{pmatrix} = A_{3\times3} \left(R_{3\times3} \begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} + t_{3\times1} \right) \quad (8)$$

where subscript v refers to the virtual view.

The general mode is based on a "reverse warping" algorithm. Instead of forward warping the left and right reference views to the virtual location, the left and right depth maps are warped to the virtual view location. Then after filtering, these depth maps are used to warp the reference views to the virtual view. This results in a higher rendering quality of the final synthesized view. Figure 26 depicts the flow diagram of the general mode.

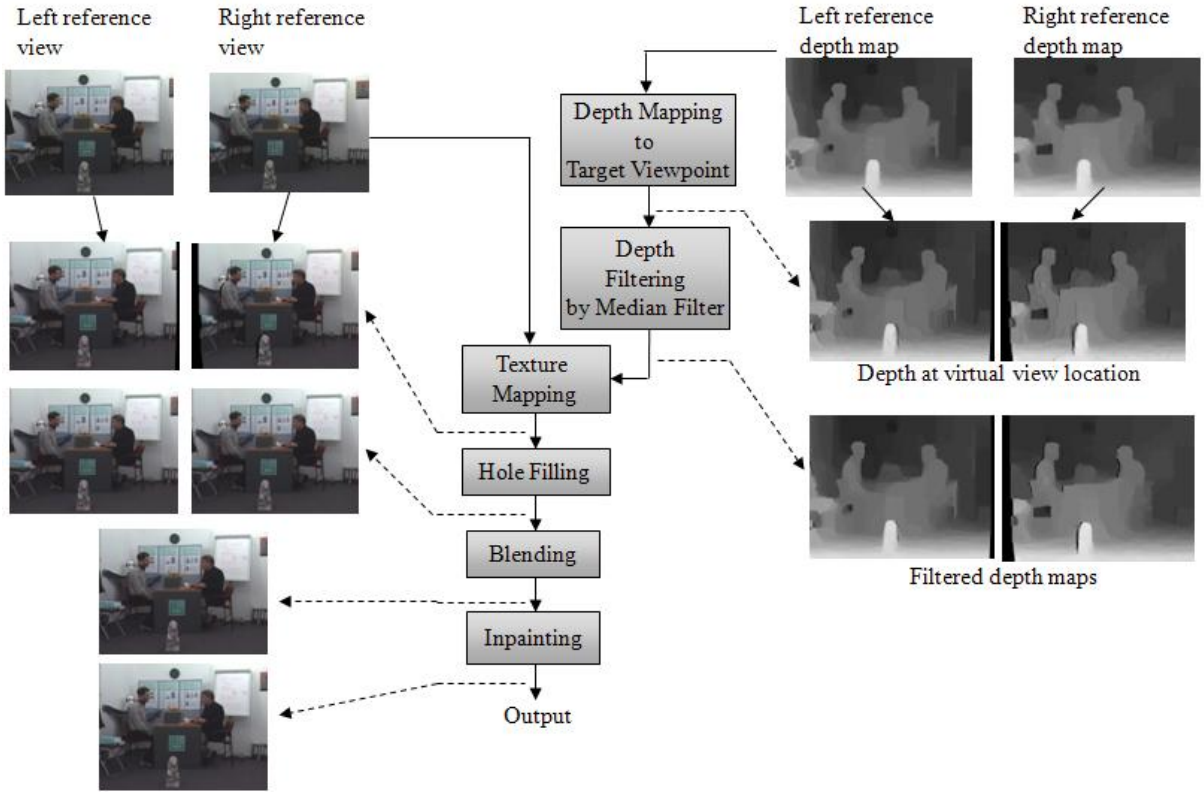

Figure 26: Flow diagram for VSRS general mode.

The steps of VSRS general mode are briefly described below:

1. First, the two depth maps are mapped to the target viewpoint. E.g. the left reference depth is warped to the virtual view location using eq. (6) and (8). If multiple pixels warp to the same location in the virtual view, then the pixel closest to the camera wins, so foreground pixels will occlude background pixels. The right depth map is also warped in a similar way. We denote these warped depth maps as D_L' and D_R', respectively.
2. The mapped depth maps D_L' and D_R' may contain small holes. Small holes which are caused by rounding to integer coordinates are filled by a series of median filtering. Furthermore, binary masks for each side are maintained to indicate larger holes, for example caused by occlusions that remain after filtering. During the following steps, these binary masks are used and updated if necessary (for example during hole filling in step 4).

3. Next, the left and right texture reference views are mapped to the target viewpoint using the filtered depth map D_L' and D_R'. So two texture images at the target viewpoint are obtained, one generated from the left reference view and the other from the right reference view. We denote them here as V_L' and V_R', respectively. Note that D_L' is used to warp the left reference, and D_R' is used to warp the right reference.
4. Hole areas in the mapped texture images V_L' and V_R', which are caused by occlusion, are filled by pixels from the other mapped texture image. So holes in V_L' are filled from non-hole areas in V_R' and vice versa.
5. Next, these two virtual images are blended. The general mode has two modes of blending: Blending-on and Blending-off. The Blending-on mode is a weighted blending based on the baseline distance. So pixels from the reference camera which is closer to the virtual view are assigned a higher weight, based on the baseline ratio. In Blending-off mode, all pixels visible in the closer reference view are copied to the virtual view, and only hole areas are filled from the farther reference view. During this step, the binary masks are merged to form one mask indicating remaining holes which are inpainted in the next step.
6. Any remaining holes after blending are filled by an inpainting algorithm using the binary mask. Inpainting algorithms can be used to reconstruct damaged portions of images. Generally a mask is used to indicate which image regions need to be inpainted. Next, color information is propagated inward from the region boundaries, i.e., the known image information is used to fill in the missing areas. An inpainting example is show in Figure 27.

Additionally, VSRS contains a Boundary Noise Removal algorithm. In this mode, the binary maps indicating holes caused by occlusion, are used to identify object boundaries. After identifying the background side of the holes based on the depth, the holes are expanded into the background. Then these areas in V_L' and V_R' are filled from the opposite reference view. This reduces noise around object boundaries, where foreground pixels are falsely projected into background objects due to depth errors.

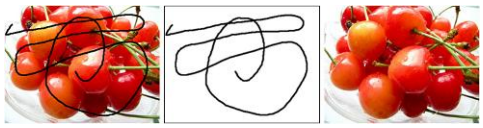

Figure 27: Inpainting: "damaged" image, mask, and result after inpainting.

3.2.2 1-d mode

VSRS provides a second synthesis mode other than the general "3D warping" as described above: 1D mode. This mode is implemented with assumptions that the optical axes of camera are in parallel and the views are rectified such that no vertical disparities exist. Under the assumption of 1D mode, formulations can be simpler than in the general case:

- The rotation matrix for every camera is identical to each other.
- The translation vectors of all cameras share the same translation in Y and Z directions, that is, T_y and T_z are constant for every view.
- As a consequence $z_v = z_r$
- Views are corrected (distortion and vertical disparity are null), so vertical position of intersection of optical axis in sensors is constant

So the $A_{3\times3}$ matrix has the following form $A_{3\times3}=\begin{bmatrix} fu & 0 & du \\ 0 & fv & dv \\ 0 & 0 & 1 \end{bmatrix}$, where fu and fv denote the horizontal and vertical focal length in pixels; du and dv the position of intersection with the optical axis in image (dv is constant among cameras).

Then eq. (8) given for the general case can be simplified as,

$$u_{v,t+1} = u_{r,t+1} - \frac{f(t_{x,r}-t_{x,v})}{z_{r,t+1}} \quad \text{and} \quad v_v = v_r \qquad (9)$$

The equation above is used to “warp” pixels from real views to the virtual one.

Figure 28 depicts the flow diagram of the VSRS 1D-mode.

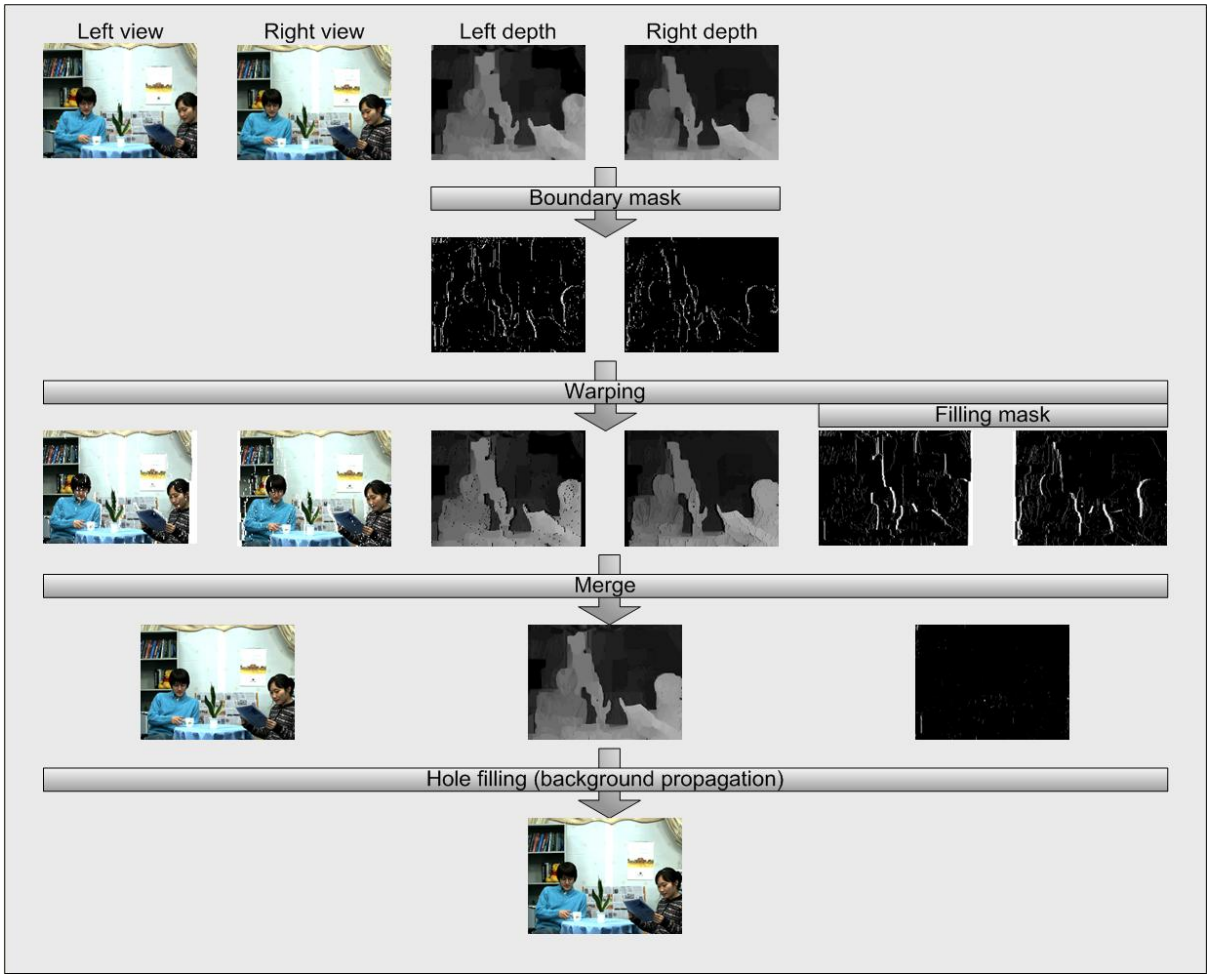

Figure 28: Flow diagram for VSRS 1D mode.

The algorithm proceeds as follows:

1. In a preliminary phase,
    a. The chroma components are upsampled to 4:4:4 format (for implementation simplicity).

   b. For suppressing transient depth errors, the depth maps can be temporally filtered according to the variations of the color information if the TemporalImprovementOption is chosen.
   c. The color video may be further upsampled, if sub-pixel precision is specified in the configuration file, for example, half-pixel or quarter-pixel.

2. During the warping process, the reference views and the depth maps are mapped to the target viewpoint using eq. (9), which is a 1D shifting on the samples. For each reference view, a binary mask is maintained indicating whether a pixel in the targeted map is filled or not (hole pixel). The warping procedure is also controlled by the splatting switch in configuration file. When splatting is selected, each pixel in the reference view may be mapped to two sample locations. Besides, two enhancement processing on warping (corresponding to CleanNoiseOption and WarpEnhancementOption) suppress some synthesis artifacts due to the texture-depth misalignment at object boundaries (which causes foreground pixels scattered to the background) and wrongly categorized holes in the foreground (which makes background pixels appear in the foreground). Warping of the unreliable pixels (which probably yield artifacts) is forbidden accordingly.
3. Two warped images from left and right reference views are obtained from last step, which are then merged to a single image. This operation is also applied on warped depth maps and filling masks. In case of conflicts (two pixels present for the same target position), the MergingOption specified by the user is applied in the following way.
   a. Z-buffer only: Take the pixel closest to camera always.
   b. Averaging only: Mix colors using weights in reverse proportional to the distance of the virtual camera from the left and right reference views
   c. Adaptive merging: Use either the proximity criterion (a) if depth level difference is greater than a threshold or, (b) if depth levels are too similar, uses the weighting method.
4. Hole areas in the warped images are filled by propagating the background pixels into the hole along the horizontal row.
5. Final view image is downsampled to original size if necessary and transformed to 4:2:0 format for output purposes.

Additionally, VSRS 1D mode can use the boundary noise removal algorithm already described as final processing step in the section dedicated to the general mode.

4 Software

As initial software for the HEVC-based 3DV project, the software that has been used for the submissions m22570, m22571, and m22668 is chosen. The software is based on the HEVC reference software, version HM 3.0.

4.1 Software repository

The source code for the software will be available in the MPEG SVN repository. An initial version of the software is already available in the following SVN repository.

https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSoftware/

For tool integration a branch for a company can be obtained by contacting:

gerhard.tech@hhi.fraunhofer.de,
kwegner@multimedia.edu.pl

4.2 Build System

The software can be build under linux using make. For windows, solutions for different versions of Microsoft Visual Studio are provided.

4.3 Software Structure

The 3D-HEVC Test Model Software includes several applications and libraries for encoding, decoding and view synthesis:

- Applications:
  - TAppEncoder, executable for bit stream generation
  - TAppDecoder, executable for reconstruction.
  - TAppRenderer, executable view synthesis
- Libraries:
  - TAppCommon, library for handling encoder, decoder and renderer options and camera parameters
  - TLibEncoder, encoding functionalities
  - TLibDecoder, decoding functionalities
  - TLibRenderer, renderer functionalities
  - TLibCommon, common functionalities
  - TLibVideoIO, video input/output functionalities