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

ISO/IEC JTC1/SC29/WG11 MPEG2015/N15349
June 2015, Poland, Warsaw

Source **Requirements**
Status **Approved**
Title **FTV software framework**
Editors Gauthier Lafruit, gauthier.lafruit@ulb.ac.be
Krzysztof Wegner, kwegner@multimedia.edu.pl
Masayuki Tanimoto, tanimoto@nagoya-u.jp

1 Introduction

This document gives an overview of the processing steps that are involved in multi-view video coding using DERS (Depth Estimation Reference Software), VSRS (View Synthesis Reference Software) and 3D-HEVC (3D High Efficiency Video Coding) reference software tools that have been developed throughout the MPEG-FTV (Free viewpoint TV) and MPEG video coding exploration and standardization activities.
Since these tools have continuously evolved, targeting a plethora of requirements throughout their evolution, it has become increasingly challenging – especially for non-experts in the field - to properly use the tools, understanding their input/output data formats and settings.
This document is a quick start throughout examples of multi-view test sequences with arbitrary multi-camera arrangements, aimed at addressing the new challenges of Super-Multi-View (SMV) and Free Navigation (FN) in the FTV Call for Evidence [CfE FTV document].
Annex 10 describes how the user can run the script files for coding and/or view synthesis in SMV and FN applications, with the test sequences and camera parameters that are used in the CfE.

2 Multi-camera Processing Pipeline

The multi-view processing pipeline is basically composed of three steps: Acquisition, Transmission, and Displaying.

The figures in this section provide the complete processing flow, with green arrows and red blocks corresponding to the main processing kernels, blue arrows addressing pre-processing steps, and yellow arrows providing configuration parameters.
The pre-processing includes the camera parameters extraction and depth/disparity estimation. The actual compression is performed by a multi-view codec like MV-HEVC (without depth) or

3D-HEVC (with depth). The final rendering typically involves some view interpolation, synthesizing additional views typically not present in the decoded bitstream.
All figures are connected to each other through the circled numbers in green, brown and yellow at the borders of each figure, and will be further explained in the remainder of the document, referencing specific modules with the black italic numbers indicated in these figures.

2.1 Acquisition

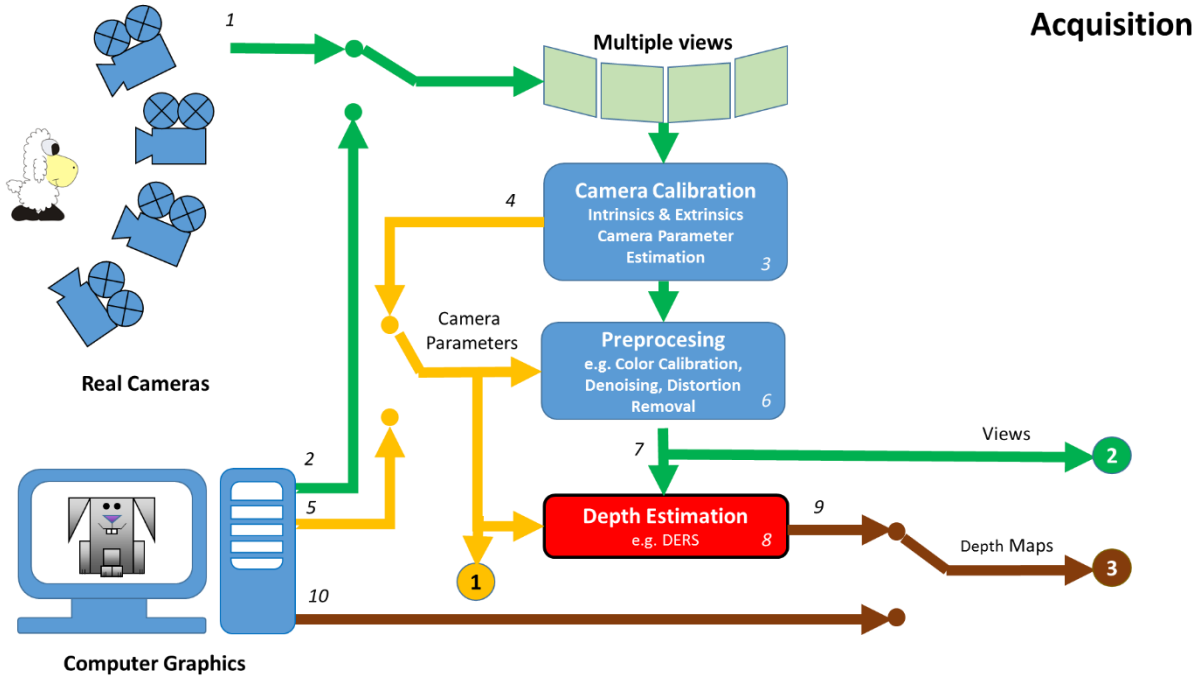


Figure 1: Main acquisition processing flow

In Figure 1, multi-view video can be directly acquired by a multi-camera system (*1*) or it can be generated artificially (*2*) by computer software (e.g. Blender). Based on the captured sequences, the camera parameters (*4*) are estimated through special camera calibration software (*3*). Alternatively, in the case of computer generated sequences (*2*), the camera parameters (*5*) can be provided directly from the renderer that created the sequences. Commonly, camera parameters are provided as intrinsic and extrinsic matrices, one per view. Additionally it may consist of image distortion coefficients for each view. Acquired sequences are preprocessed (*6*) in order to color calibrate all views, remove image distortion and denoise the sequences. Undistorted and color corrected views (*7*) are used to estimate the depth maps (*9*) through depth estimation software (*8*).
Alternatively, depth maps (*10*) can directly be obtained during the view generation, e.g. in the case of computer graphics generated sequences, as the depth map is usually generated during rendering for proper occlusion handling.
The end result of the full acquisition step is a number of color calibrated views (**2**), a number of depth maps (**3**) (commonly one per view) and a set of camera parameters (**1**) (one per view).

2.1.1 Camera parameters

Figure 2 gives the main processing steps for the camera parameter extraction.

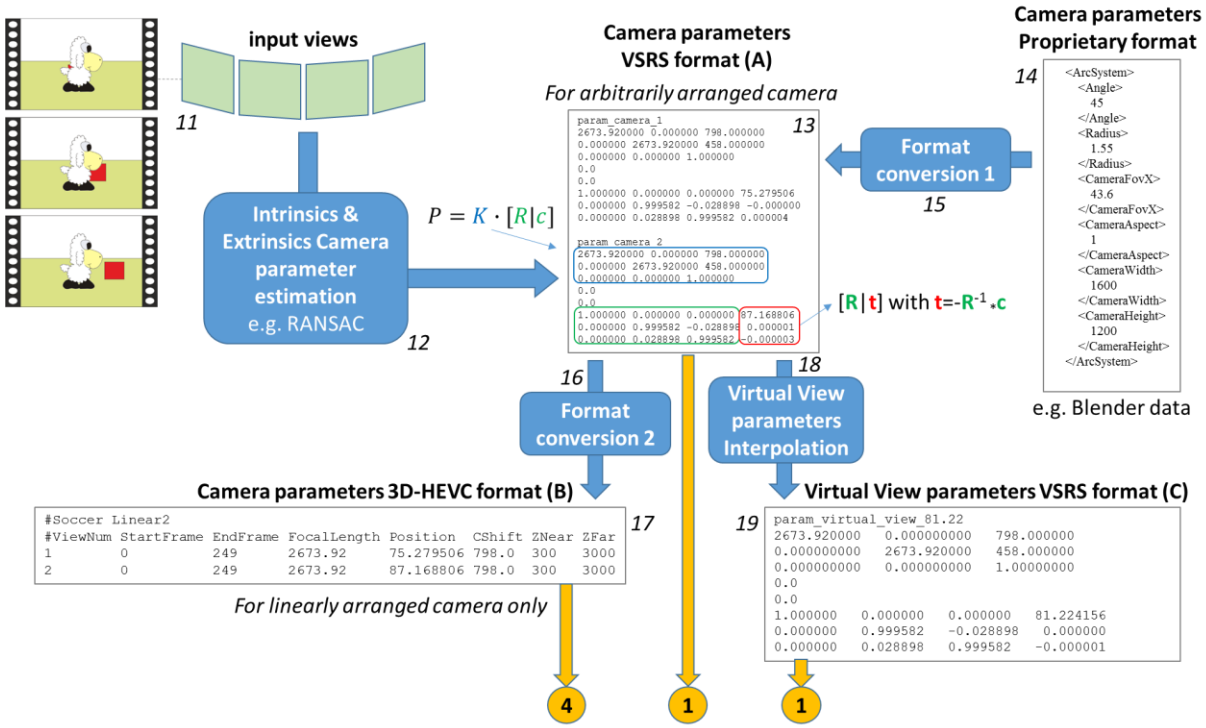


Figure 2: Various camera parameter formats used in the FTV framework

Starting from the input views (*11*), all camera parameters are estimated with calibration software (*12*) (not made available by MPEG), providing the camera parameter in VSRS compatible format A (*13*). For each camera section (starting with camera name e.g. *param_camera_1*) the first 3 lines give the camera intrinsics (i.e. the parameters of each camera individually: focal length, principal point, etc. in pixel units), while the last 3 lines give the camera extrinsics (i.e. the camera rotation and translation, in world coordinates). Note, that the VSRS camera format does not follow the extrinsics format commonly used in literature (i.e. [Rotation R | local translation c]), but rather uses the camera position in global world coordinates system (i.e. [Rotation R| global translation t = $-R^{-1} \cdot c$]). More information can be found in Annex 1, which has been copied from [W9595].

As the views might also be rendered from a 3D model by 3D design software, the camera parameters are known, and typically provided in some proprietary format (*14*), e.g. Holografika's XML format. Such camera parameters should be converted with an appropriate script (*5*) into VSRS compatible format. A script example for converting Holografika's XML camera parameters into VSRS is given in Annex 2.

A camera parameter file B (*17*) is also required through (*23*, Figure 8) for proper compression of the multi-view data with 3D-HEVC. As 3D-HEVC has been designed strictly for linear camera setups (input views should be registered properly, i.e. all epipolar lines should be horizontal) the 3D-HEVC camera parameter format contains only a subset of the information available in general VSRS camera parameter format A (*13*): only one component of the translation vector is considered, and the camera rotation matrix is absent. Additionally, the 3D-HEVC camera

parameter file B also contains the *Znear* and *Zfar* parameters that are linked to the depth information of a given view. More information is given in the following section.

The camera parameters in VSRS format can readily be used in DERS (Depth Estimation Reference Software) through (*15*), and VSRS (View Synthesis Reference Software) through (*34*, Figure 3).

2.1.2 Preprocessing

As shown in the figure below, the multi-view input data (*1*, *2*) is typically color calibrated (6), denoised and image distortion is removed. Note that the input data (captured by cameras or generated by the computer) can be raw (3 bytes RGB per pixel – (*2*) or Bayer mosaic (*1*)), whereas further processing in the chain requires the data to be in YUV420 format (*3*, *6*, *32*) with per frame all the luminance Y values, followed by the chrominance U and V values. All frames are concatenated into one YUV file per camera view.

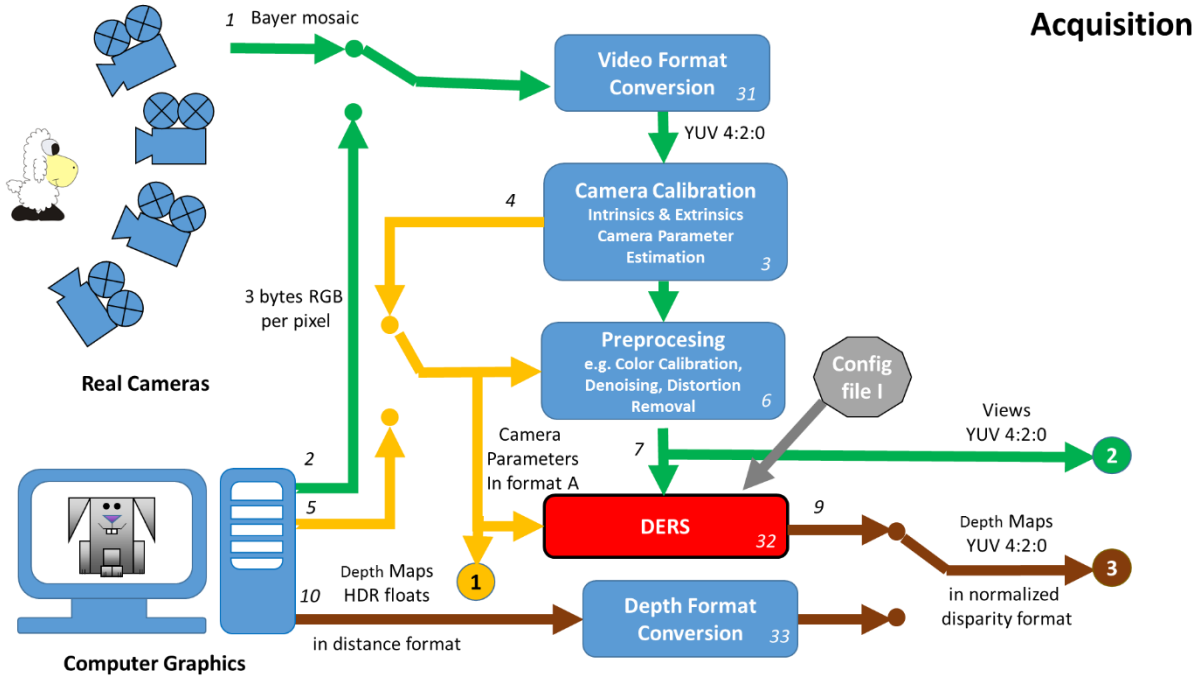


Figure 3: Video data preparation and depth/disparity estimation

2.1.3 Depth/Disparity estimation

The depth map is an image where each pixel of a view provides information about the distance of the observed point to the camera. The depth information can be provided in various formats. The most commonly used are: direct distance to the objects, normalized distance, stereoscopic disparity, normalized disparity.

The most natural way of storing a depth map is its direct distance to the object, where each value of the depth map represents a distance between the observed object and the camera plane. Distances are often stored as floating point numbers. The HDR file format used by Blender is an example of such a depth map format.

Because potential huge dynamics of distances can occur in a given scene, the direct distance can be normalized to represent the numbers in an efficient way. Commonly, 3D graphic software use

Z-buffer data normalized to the clipping range, with commonly used *Znear* and *Zfar* parameters. Stereoscopic disparity refers to pixel position differences of an object seen from two viewpoints. A simplified relation between disparity d and distance z is given in the following equation

$$d = f \cdot \frac{B}{z} \tag{1}$$

where f is the focal length of the camera acquiring the image, and B is the baseline of the used stereo pair.

Generally, disparity can be in a range of 0 to infinity. However, in practical cases disparity is limited to a certain range: $d_{min} \div d_{max}$. The d_{min} is related through equation 1 to *zFar*, and d_{max} to *zNear*. For practical reasons, instead of providing d_{min}, *and* d_{max}, *zFar* and *zNear* value are used. Then distance z to the object can be obtained from normalized disparity sample v through the following equation

$$z = \frac{1}{\frac{v}{v_{max}} \cdot \left(\frac{1}{zNear} - \frac{1}{zFar}\right) + \frac{1}{zFar}}$$

where v_{max} is 255 for 8 bit depth maps and 65535 for 16 bit depth maps. MPEG commonly uses depth maps in normalized disparity format.

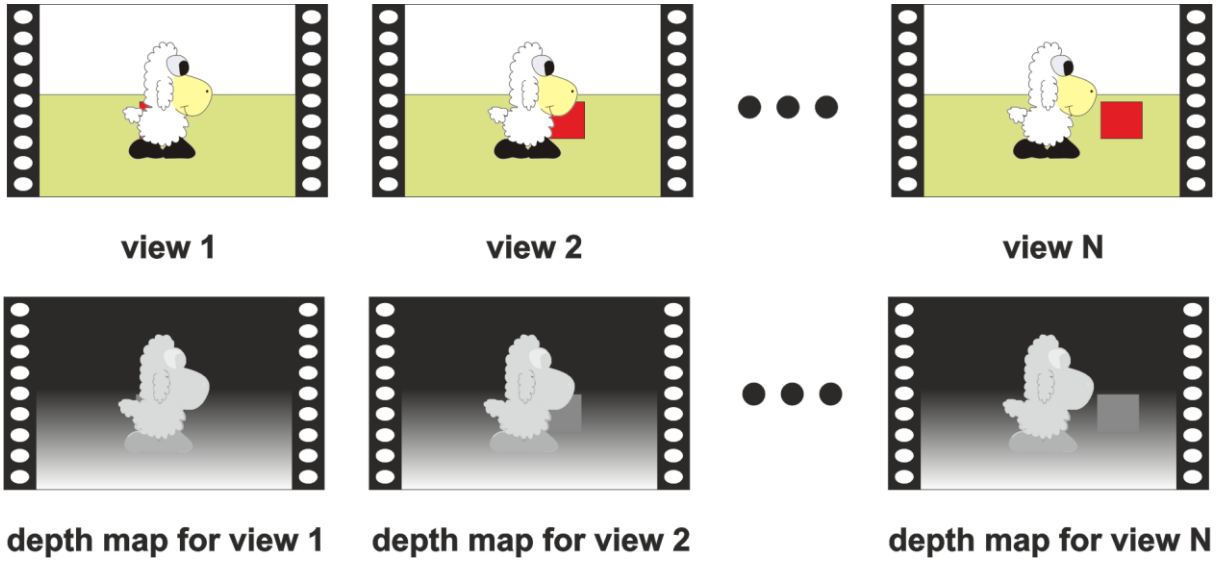


Figure 4: Views and correspondent depth maps

Depth maps are commonly estimated algorithmically through depth estimation software (*8*, Figure) (possibly proprietary) based on the analysis of the input views (*7*). Alternatively it can be obtained as an intermediate result for view rendering in computer generated sequences (*10*).

2.2.1.1 Depth Estimation Reference Software

Throughout the years, MPEG has been developing Depth Estimation Reference Software (DERS) which has constantly been improved in order to provide state-of-the-are depth estimation results. Currently DERS is not limited to linear view arrangements any more: it can estimate depth maps out of three arbitrary arranged input views. More details can be found in [DERS] and [DERSManual]. Figure 5 gives a typical example of a depth estimation configuration file (config file I at step (32) in Figure 3**Błąd! Nie można odnaleźć źródła**

odwołania.) . In DERS, depth is estimated with a graph cut algorithm on 3 successive cameras (left, center, right), which are given at the top of the configuration file. It outputs depth maps in 4:2:0 YUV file where the Y component contains normalized disparity of the center view. Additionally it outputs two scaling values *zNear* and *zFar* used to normalize the disparity.

```
#=============== INPUT PARAMETERS ================
DepthType                               0               # 0...Depth from camera, 1...Depth from the origin of 3D space
SourceWidth                             1600            # Input frame width
SourceHeight                            1200            # Input frame height
StartFrame                              0               # Starting frame
TotalNumberOfFrames                     3              # Total number of input frames
LeftCameraName                          param_uh_line2_1 # Name of left camera
CenterCameraName                        param_uh_line2_2 # Name of center camera
RightCameraName                         param_uh_line2_3 # Name of right camera
MinimumValueOfDisparitySearchRange      1                # Minimum value of disparity search range. This value is not always same
as all pairs of views.
MaximumValueOfDisparitySearchRange      55               # Maximum value of disparity search range. This value is not always same
as all pairs of views.
MinimumValueOfDisparityRange            1                # Minimum value of disparity range. This value is smaller than or equal
to minimum value of disparity search ranges for all views.
MaximumValueOfDisparityRange            55              # Maximum value of disparity range. This value is larger than or equal to
maximum value of disparity search ranges for all views.
SmoothingCoefficient                    1.0             # Smoothing coefficient to compute depth maps
FileLeftViewImage                       C:\Users\glafruit\Data\MPEG\Test_sequences\UHasselt_soccer_dataset\Line2\yuv\eye1.yuv
# Name of left input video
FileCenterViewImage                     C:\Users\glafruit\Data\MPEG\Test_sequences\UHasselt_soccer_dataset\Line2\yuv\eye2.yuv
# Name of center input video
FileRightViewImage                      C:\Users\glafruit\Data\MPEG\Test_sequences\UHasselt_soccer_dataset\Line2\yuv\eye3.yuv
# Name of right input video
FileOutputDepthMapImage
C:\Users\glafruit\Data\MPEG\Test_sequences\UHasselt_soccer_dataset\Line2\yuv\depth_eye_123d.yuv       # Name of output depth map
file
FileCameraParameter                      ..\camera_parameter_files\cam_param_uh_line2.txt                              # Name of
text file which includes camera parameters

BaselineBasis                               1                  # 0...minimum baseline, 1...maximum baseline, 2...left baseline,
3...right baseline
Precision                               1               # 1...Integer-pel, 2...Half-pel, 4...Quater-pel
SearchLevel                             1               # 1...Integer-pel, 2...Half-pel, 4...Quater-pel
Filter                                  1               # 0...(Bi)-linear, 1...(Bi)-Cubic, 2...MPEG-4 AVC 6-tap
MatchingMethod                           0               # 0...Conventional, 1...Disparity-based, 2...Homography-based, 3...Soft-
segmentation-based

#========== Temporal Enhancement ==========
TemporalEnhancement                     0               # 0...Off, 1...On
Threshold                               2.00            # Threshold of MAD

#========== Size of Matching Block (ignored when MatchingMethod = 3) ==========
MatchingBlock                           3               # 1...Pixel matching, 3...3x3 block matching

#========== Softsegmentation (for MatchingMethod = 3 only) ==========
SoftDistanceCoeff                       10.0            # SoftSegmentation Distance Coefficient
SoftColorCoeff                          20.0            # SoftSegmentation Color Coefficient
SoftBlockWidth                          11              # SoftSegmentation Block Width
SoftBlockHeight                         11              # SoftSegmentation Block Height

#========== Segmentation ==========
ImageSegmentation                   0                   # 0...Off, 1...On
SmoothingCoefficient2               1.00                # Smoothing coefficient to compute depth maps
SegmentationMethod                      3                       # 1...mean shift algorithm, 2...phyramid segmentation, 3...K mean
clustering
MaxCluster                          24                  # Positive Integer Value

#=====================================================
#========== Semi-automatic Depth Estimation ==========
#=====================================================
DepthEstimationMode                      0                      #0...automatic Depth Estimation; 1...Semi-automatic mode1; 2...Semi-
automatic mode2; 3...Reference Depth mode

#---- For DepthEstimationMode = 1 or 2  ------
FileCenterManual                     ..\semi_automatic_input\champagne_41    #Path and filename prefix of the manual input files

#---- For DepthEstimationMode = 2  ------
ThresholdOfDepthDifference              10              # Threshold value of depth difference
MovingObjectsBSize                      1               # 0: small  1: medium  2: large
MotionSearchBSize                       0               # 0: narrow 1: medium 2: wide

#---- For DepthEstimationMode = 3  ------
RefDepthCameraName      param_uh_line2_2                # camera parameter name
RefDepthFile            rdepth_eye2.yuv                 # filename reference depthmap video
```

Figure 5: Example configuration file for DERS

2.2.1.2 Recommended DERS settings

For sequences with wide camera baseline or a big depth range it is recommended to compile the DERS with POZNAN_16BIT_DEPTH flag, to support estimation of extended 16 bit depth maps.

2.2.1.3 Computer generated depth maps

When using Computer Generated Images/Projections from 3D synthetic data as multiview input, a depth map can be extracted in floating point EXR or HDR format by using the Z output of a Render layer in the composition graph. In this case no depth estimation software is then needed as ground truth depth is generated as a side product of the rendering process.

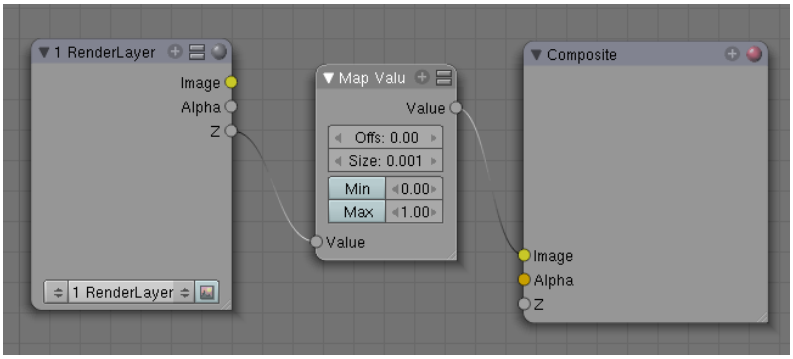


Figure 6: Example composition graph for outputting depth values from Blender. Please note the scaling node that uniformly divides the depth values before writing them to a file to avoid clamped values. exactly measured.

Currently it is required to convert (*33*, Figure) distance base depth maps to normalized disparity depth maps stored as a 4:2:0 YUV file.

2.3 Multiview compression

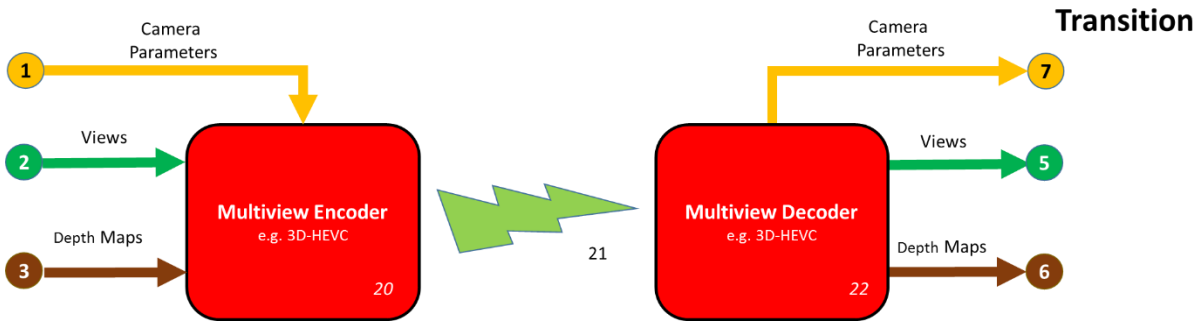


Figure 7: Texture and Depth/Disparity data encoding, decoding and rendering in Super-MultiView and Free Navigation

The set of views along with the corresponding depth maps, if present, are encoded by a Multiview Encoder (*20*) e.g. 3D-HEVC. The encoder may use view redundancy, as well as depth redundancy, as well as inter-component redundancy in order to efficiently code the multi-view sequence. With the help of camera parameters, view synthesis prediction can be used as well to further enhance the compression efficiency. The Multiview Decoder (*22*) decodes the encoded bitstreams according to the parameters encoded in the bitstream and outputs decoded views and depth maps (if existing).

2.3.1 3D-HEVC: depth-based High Efficiency Video Coding

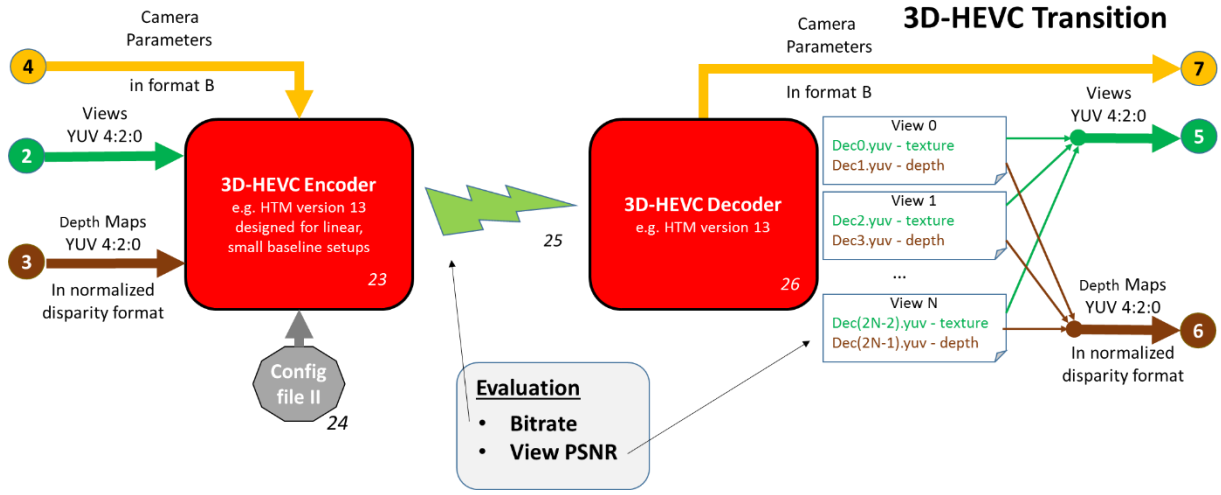


Figure 8: Texture and Depth/Disparity data encoding, decoding.

The bitstream is decoded, recovering the camera parameters, the views and depth maps in the final displaying step, as shown in the figure above.

Since 3D-HEVC has been designed strictly for linear camera arrangements (no rotation matrix in the camera parameter file (B)**Błąd! Nie można odnaleźć źródła odwołania.**), good compression performances can only be guaranteed for registered/aligned camera views.

For compatibility reasons, we are currently using the HTM13 reference software of 3D-HEVC available from https://hevc.hhi.fraunhofer.de/svn/svn_3DVCSoftware/tags/HTM-13.0 (NOT the latest software documented on https://hevc.hhi.fraunhofer.de/HM-doc/).
For properly using HTM13 for MPEG-FTV, the source code modification are required according to Annex 5.

The 3D-HEVC codec (HTM13) takes textures (**2** – the video data in YUV 420 file format), depth/disparities (**3** – normalized disparity maps in YUV 420 file format) and camera parameters (**4** – in B format), together with a config file (*24* - example portions are given in Figure), to create a single coded bitstream (*25*) containing compressed versions of the video and depth data. Be aware that different and intelligently chosen quantization parameters *QP* should be used for texture and depth, in order to provide the highest quality of the rendered views at fixed bitrate.

```
#======== Legend for comments =====================
# (m) specification per layer/dimension/layerset possible
# (c) cyclic repetition of values, if not given for all layers/dimensions/layersets. (e.g. 5 layers and 1 2 3 -> 1 2 3 1 2  )

#======== File I/O =====================

InputFile_0                   : ..\Data\4.yuv
InputFile_1                   : ..\Data\d4.yuv
InputFile_2                   : ..\Data\3.yuv
InputFile_3                   : ..\Data\d3.yuv
InputFile_4                   : ..\Data\2.yuv
InputFile_5                   : ..\Data\d2.yuv
InputFile_6                   : ..\Data\1.yuv
InputFile_7                   : ..\Data\d1.yuv
InputFile_8                   : ..\Data\5.yuv
InputFile_9                   : ..\Data\d5.yuv
```

```
InputFile_10                  : ..\Data\6.yuv
InputFile_11                  : ..\Data\d6.yuv
InputFile_12                  : ..\Data\7.yuv
InputFile_13                  : ..\Data\d7.yuv

BitstreamFile                 : stream.bit

ReconFile_0                   : rec_4.yuv
ReconFile_1                   : rec_4d.yuv
ReconFile_2                   : rec_3.yuv
ReconFile_3                   : rec_3d.yuv
ReconFile_4                   : rec_2.yuv
ReconFile_5                   : rec_2d.yuv
ReconFile_6                   : rec_1.yuv
ReconFile_7                   : rec_1d.yuv
ReconFile_8                   : rec_5.yuv
ReconFile_9                   : rec_5d.yuv
ReconFile_10                  : rec_6.yuv
ReconFile_11                  : rec_6d.yuv
ReconFile_12                  : rec_7.yuv
ReconFile_13                  : rec_7d.yuv

FramesToBeEncoded             : 250           # Number of frames to be coded
FrameRate                     : 25          # Frame Rate per second
SourceWidth                   : 1600        # Input  frame width
SourceHeight                  : 1200        # Input  frame height
NumberOfLayers                : 14
TargetEncLayerIdList          :             # Layer Id in Nuh to be encoded, (empty:-> all layers will be encode)

QP                         : 30  39                                # quantization parameter (view depth)

#======== VPS ============================
ScalabilityMask               : 3                                     # Scalability Mask               ( 2: View Scalability, 3:
View + Depth Scalability )
DimensionIdLen                : 4 4                                   # Number of bits to store Ids,  per scalability dimension,
(m)
ViewOrderIndex                : 0 0 1 1 2 2 3 3 4 4 5 5 6 6     # ViewOrderIndex, per layer (m)
DepthFlag                     : 0 1 0 1 0 1 0 1 0 1 0 1 0 1           # DepthFlag  (m)
LayerIdInNuh                  : 0                                     # Layer Id in NAL unit header, (0: no explicit signalling,
otherwise per layer ) (m)
SplittingFlag                 : 0                                     # Splitting Flag
ViewId                        : 3 2 1 0 4 5 6                         # ViewId, per ViewOrderIndex (m)

#======== VPS / Layer sets ================
VpsNumLayerSets               : 9                                     # Number of layer sets    # of View + 2
LayerIdsInSet_0               : 0                                     # Indices in VPS of layers in layer set 0
LayerIdsInSet_1               : 0 1                                   # Indices in VPS of layers in layer set 1
LayerIdsInSet_2               : 0 1 2 3                               # Indices in VPS of layers in layer set 2
LayerIdsInSet_3               : 2 3 4 5                               # Indices in VPS of layers in layer set 3
LayerIdsInSet_4               : 4 5 6 7                               # Indices in VPS of layers in layer set 3
LayerIdsInSet_5               : 0 1 8 9                               # Indices in VPS of layers in layer set 2
LayerIdsInSet_6               : 8 9 10 11                             # Indices in VPS of layers in layer set 3
LayerIdsInSet_7               : 10 11 12 13                 # Indices in VPS of layers in layer set 3
LayerIdsInSet_8               : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 # Indices in VPS of layers in layer set 4

#======== VPS / Output layer sets ================
DefaultTargetOutputLayerIdc    : 0              # Specifies output layers of layer sets, 0: output all layers, 1: output highest
layer, 2: specified by LayerIdsInDefOutputLayerSet
#OutputLayerSetIdx             : 2 3          # Indices of layer sets used to derive additional output layer sets
#LayerIdsInAddOutputLayerSet_0 : 2 3          # Indices in VPS of output layers in additional output layer set 0
#LayerIdsInAddOutputLayerSet_1 : 4 5          # Indices in VPS of output layers in additional output layer set 1
…

#======== VPS / Dependencies ================
DirectRefLayers_1             : 0           # Indices in VPS of direct reference layers
DirectRefLayers_2             : 0 1         # Indices in VPS of direct reference layers
DirectRefLayers_3             : 1 2         # Indices in VPS of direct reference layers
…

DependencyTypes_1             : 2           # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion
DependencyTypes_2             : 2 2         # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion
DependencyTypes_3             : 2 2         # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion
…

#========== Camera parameters ==========
BaseViewCameraNumbers         : 4 3 2 1 5 6 7                                                 #  camera  numbers  of  coded
views ( in coding order per view )
CameraParameterFile           : ..\cam_param_Soccer_Linear2_for_HTM13.txt  # camera parameter file
CodedCamParsPrecision         : 5                                     # precision used for coding of camera parameters (in units
of 2^(-x) luma samples)

…

#======== Coding Structure =============
IntraPeriod                   : 24          # Period of I-Frame ( -1 = only first)
DecodingRefreshType           : 1           # Random Accesss 0:none, 1:CRA, 2:IDR, 3:Recovery Point SEI
GOPSize                       : 8           # GOP Size (number of B slice = GOPSize-1)
```

```
#                          QPfactor      betaOffsetDiv2   #ref_pics_active  reference pictures     deltaRPS     reference idcs
ilPredLayerIdc       refLayerPicPosIl_L1
#            Type   POC  QPoffset      tcOffsetDiv2       temporal_id       #ref_pics                      predict      #ref_idcs
#ActiveRefLayerPics     refLayerPicPosIl_L0
Frame1:       B      8       1       0.442      0           0           0           4         4       -8  -10  -12  -16       0
0
Frame2:     B     4      2      0.3536    0         0         0         2       3      -4 -6   4           1      4      5      1 1 0 0 1
0
…

#========== depth coding tools ==========
VSO                                 : 1                # use of view synthesis optimization for depth coding
IntraWedgeFlag                      : 1
IntraContourFlag                    : 1                # use of intra-view prediction mode
IntraSdcFlag                        : 1
DLT                                 : 1
QTL                                 : 1
QtPredFlag                          : 1
InterSdcFlag                        : 1                # use of inter sdc
MpiFlag                             : 1
IntraSingleFlag                     : 1                # use of single depth mode

#========== view synthesis optimization (VSO) ==========
#VSOConfig                      : [cx0 B(cc1) I(s0.5)][cx1 B(oo0) B(cc2) I(s0.5 s1.5)][cx2 B(oo1) B(cc3) I(s1.5 s2.5)][cx3 B(oo2)
B(oo4) I(s2.5 s3.5)][cx4 B(cc3) B(oo5) I(s3.5 s4.5)][cx5 B(cc4) B(oo6) I(s4.5 s5.5)][cx6 B(cc5) I(s5.5)] # VSO configuration
string
VSOConfig                   : [cx0 B(cc1) I(s0.25 s0.5 s0.75)][cx1 B(oo0) B(cc2) I(s0.25 s0.5 s0.75 s1.25 s1.5 s1.75)][cx2 B(oo1)
B(cc3) I(s1.25 s1.5 s1.75 s2.25 s2.5 s2.75)][cx3 B(oo2) B(oo4) I(s2.25 s2.5 s2.75 s3.25 s3.5 s3.75)][cx4 B(cc3) B(oo5) I(s3.25
s3.5 s3.75 s4.25 s4.5 s4.75)][cx5 B(cc4) B(oo6) I(s4.25 s4.5 s4.75 s5.25 s5.5 s5.75)][cx6 B(cc5) I(s5.25 s5.5 s5.75)] # VSO
configuration string
#VSOConfig                  : [cx0 B(cc1) I(s0.25 s0.5 s0.75)][cx1 B(oo0) B(oo2) I(s0.25 s0.5 s0.75 s1.25 s1.5 s1.75)][cx2 B(cc1)
I(s1.25 s1.5 s1.75)] # VSO configuration string
#VSOConfig                  : [ox0 B(cc1) I(s0.25 s0.5 s0.75)][cx1 B(oo0) B(oo2) I(s0.25 s0.5 s0.75 s1.25 s1.5 s1.75)][ox2 B(cc1)
I(s1.25 s1.5 s1.75)] # VSO configuration string for FCO = 1
WVSO                        : 1                                  # use of WVSO (Depth distortion metric with a weighted depth
fidelity term)
VSOWeight                 : 10                                # weight of VSO ( in SAD case, cf. squared in SSE case )
VSDWeight                 : 1                                 # weight of VSD ( in SAD case, cf. squared in SSE case )
DWeight                    : 1                                  # weight of depth distortion itself ( in SAD case, cf. squared
in SSE case )
UseEstimatedVSD           : 1                                  # Model based VSD estimation instead of rendering based for some
encoder decisions
```

Figure 9: Example of configuration file of 3D-HEVC (HTM13)

The 3D-HEVC decoder (*26*) will decode this single stream into 2.N output layers, where N is the number of encoded views, as shown in Figure 8**Błąd! Nie można odnaleźć źródła odwołania.**, i.e. the even layers contain the video data (texture), while the odd layers contain the depth data.

Be aware that the views and depth maps are not necessarily coded in the same order as their original view order, cf. the top of the example configuration file of Figure. Currently recommended 3D-HEVC configuration uses following coding structure in the view direction.

Coding structure in view direction:	P <- P <- P <- I -> P -> P -> P
Input view number from left:	1, 2, 3, 4, 5, 6, 7
Coding order:	6, 4, 2, 0, 8, 10, 12
Decoded view data number:	6, 4, 2, 0, 8, 10, 12

Input depth map number from left:	d1, d2, d3, d4, d5, d6, d7
Coding order:	7, 5, 3, 1, 9, 11, 13
Decoded depth map number:	7, 5, 3, 1, 9, 11, 13

It means that the middle view is encoded first, then its depth, then the views to the left of the middle view followed by the depth of the views to the left, and as a last view to the right of the middle view followed by its depth. Therefore, reordering of the output files from the 3D-HEVC decoder is necessary (check the validity of the reordering by observing the gradual translation of

objects in successive images for linear arrangements) and should be done before using VSRS in the final rendering step (Figure).

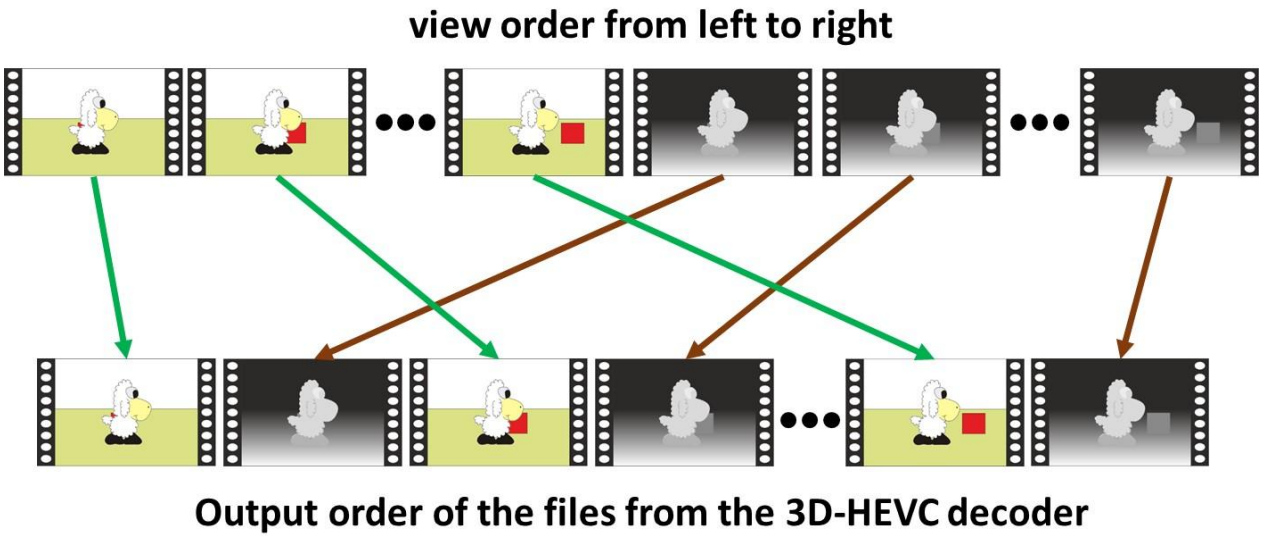


Figure 10: Reordering of the files after decoding

2.3.1.1 Recommended 3D-HEVC settings

Figure 9 shows an example of recommended 3D-HEVC setting for Free navigation case. In case of Super multi-view, an example of recommended 3D-HECV setting is in Annex 8.

2.4 View Synthesis and rendering

The display (*30*) in Figure may request some views to be provided by the system. Requested view positions are provided to the Virtual View parameters interpolator (27), which interpolates the camera parameters for the requested view positions based of the transmitted ones. Alternatively, intrinsic and extrinsic camera parameters of the requested views can be provided explicitly. View Synthesis creates the requested views based on the decoded views (**5**) and the depth maps (**6**). The synthesized views (*29*) are displayed to the viewer on the 3D display.

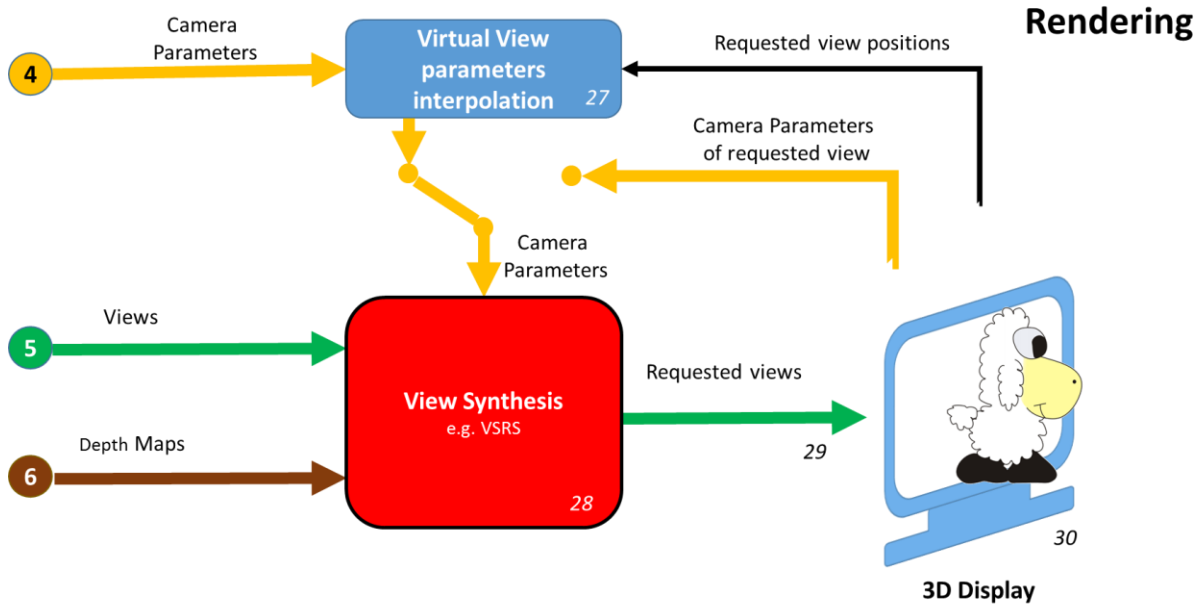


Figure 11: Rendering framework

2.4.1 Virtual View Camera Parameter Interpolation

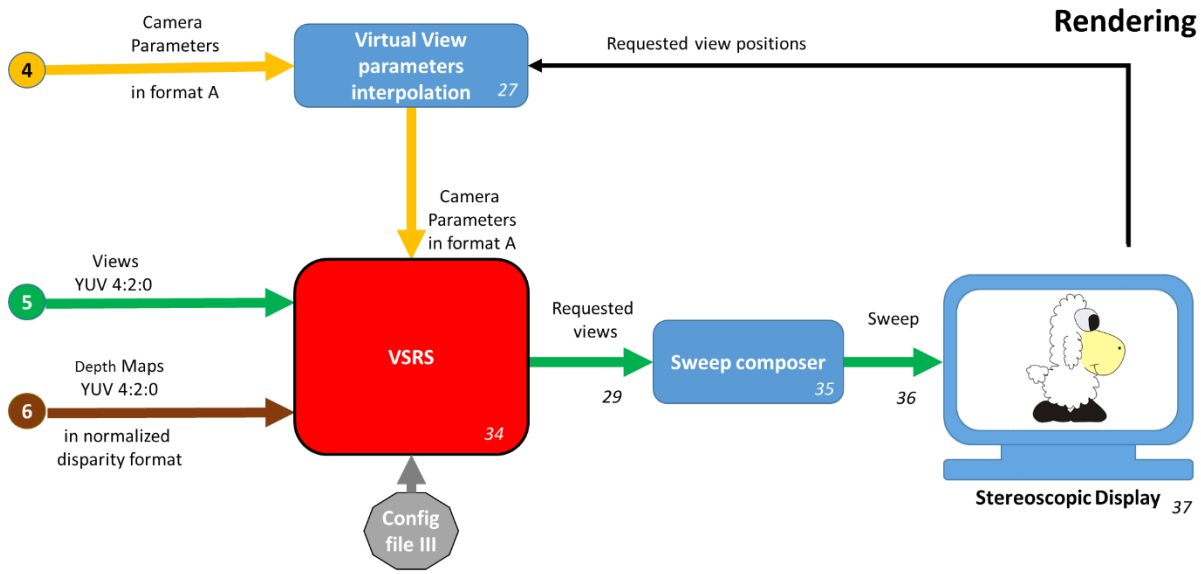


Figure 3: Currently used rendering pipeline

Camera parameters for virtual views can be generated by interpolation from existing views. Such a virtual view parameters C in format A in Figure 2 have to be provided to VSRS in order to render the views requested by the display. These parameters in (*27*, Figure 3) are calculated through rotational and translational interpolation of the physical camera parameters (**1**). In principle, this involves a spherical interpolation based on quaternions, but spline interpolation with an Excel script gives satisfactory results, as explained in Annex 3. These values calculated in Excel can be readily copied into the Virtual View Camera parameter file (C), which is VSRS compatible, as shown in Figure .

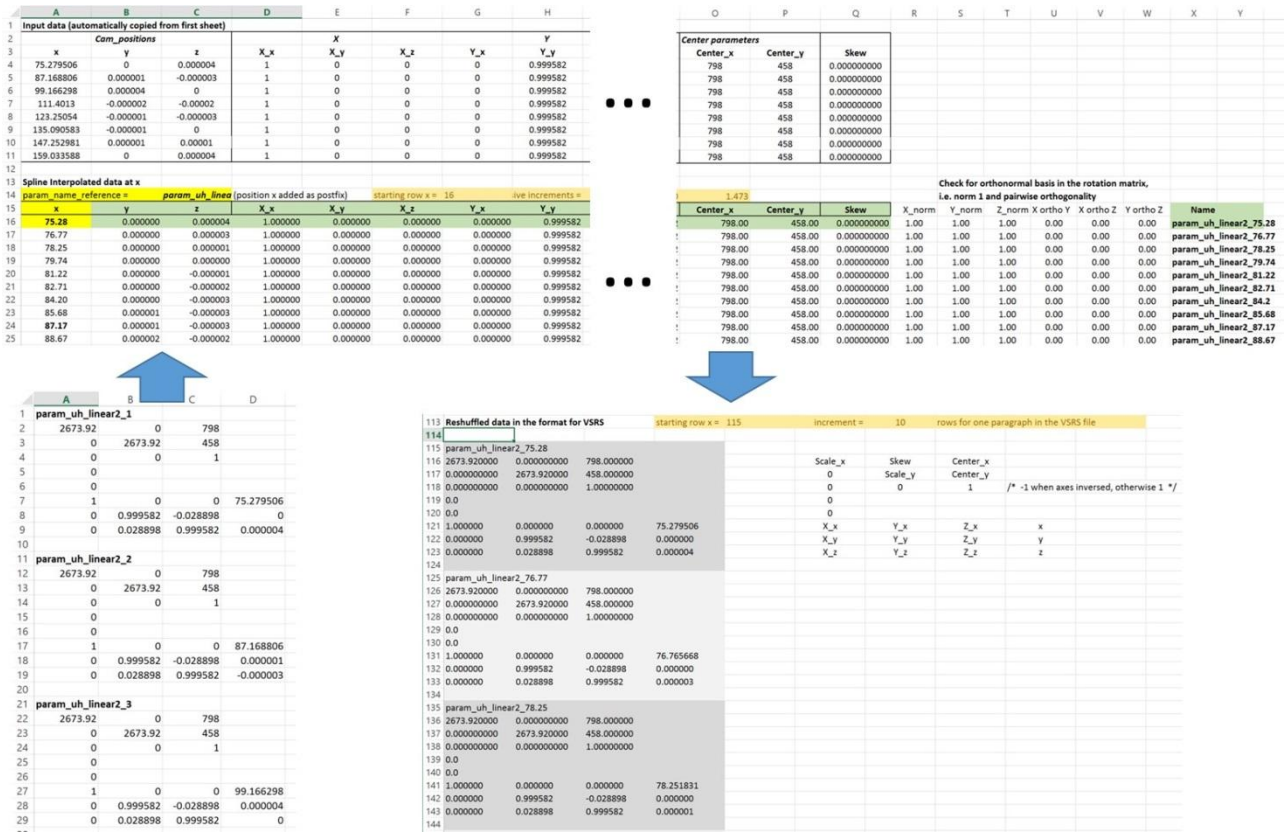


Figure 13: Excel sheet camera parameter interpolation

Suggested virtual views camera parameters file is structured as follows. It should start with the name followed by the position of the nearest real view e.g. 1 and ended with the index of the virtual view position in between real cameras e.g. number in range 1 to N for N virtual views in between two real views. The proposed virtual camera parameters name is "*param_virtual_view_VV.xx*", where VV is the real camera number closed to the left view, and xx is an index of subsequent virtual views as exemplified in Figure 14 for the arc1 soccer sequence.

```
param_uh_arc1_2
2965.270000 -0.000737 -960.003000
0.000000 2965.270000 -540.002000
0.000000 0.000000 -1.000000
0.0
0.0
0.995531 -0.092387 -0.019586 35.201600
-0.046285 -0.296534 -0.953900 -54.125100
-0.082320 -0.950543 0.299485 12.593000

param_uh_arc1_3
2996.470000 -0.000556 -960.002000
0.000000 2996.470000 -540.002000
0.000000 0.000000 -1.000000
0.0
0.0
0.994945 0.099968 -0.009532 43.556600
0.020080 -0.291051 -0.956497 -54.015600
0.098393 -0.951470 0.291587 12.664400

param_uh_arc1_44.6
2999.624265   -0.000450513   -960.0016179
0.000000.000000   2999.624108   -540.001831
0.000000   0.000000   -1.000000
0.000000
0.000000
0.993404886   0.122161527   -0.011077616   44.6
0.025264322   -0.290645534   -0.956780523   -53.92204311
0.119970121   -0.950016464   0.290643237   12.60201349

param_uh_arc1_45.6
3002.107406   -0.000336801   -960.001112
0.000000   3002.10707   -540.0015749
0.000000   0.000000   -1.000000
0.000000
0.000000
0.991571297   0.142489419   -0.013407117   45.6
0.029143208   -0.290363046   -0.957015399   -53.80369338
0.139993711   -0.948231259   0.289821479   12.52136025
```

Figure 14: Physical and virtual camera naming convention

The video file, that at the end of the processing chain (*29*) corresponds to a virtual view in Figure 14, will typically be named following the convention "*virtual_VV.xx_from_Y_Z*", where (Y,Z) is optional information regarding the left and right views used for the view synthesis with VSRS.

2.4.2 View Synthesis Reference Software

MPEG has been developing View Synthesis Reference Software (VSRS) for synthesizing virtual views (*34*) from two input views and two correspondence depth maps in normalized disparity format. Although synthesized view can be positioned at any place in 3D space, commonly it is positioned in-between input views. The same VSRS module is also used as a component in the 3D-HEVC codec for view prediction of existing camera views. The VSRS software can be downloaded from http://wg11.sc29.org/svn/repos/Explorations/FTV
More information about using the VSRS software can be found in Annex 6 and in VSRS manual [VSRS].

An example configuration file (config file III from Figure 3) for VSRS is given in Figure 55. There is clearly a reference to a left and right camera view, from which the virtual view will be calculated. Depending on the position of this virtual view, different left and right camera references should be used.

```
#================ Input Parameters ================
DepthType                       1                       # 0...Depth from camera, 1...Depth from the origin of 3D space
SourceWidth                     1600                    # Input frame width
SourceHeight                    1200                    # Input frame height
StartFrame                      0                                               # Starting frame #
TotalNumberOfFrames             1                     # Total number of input frames
LeftNearestDepthValue           300             # Nearest depth value of left image from camera or the origin of 3D space
LeftFarthestDepthValue          3000            # Farthest depth value of left image from camera or the origin of 3D space
RightNearestDepthValue          300             # Nearest depth value of right image from camera or the origin of 3D space
RightFarthestDepthValue         3000            # Farthest depth value of right image from camera or the origin of 3D space
CameraParameterFile                 cam_param_uh_line2.txt          # Name of text file which includes real and virtual camera
parameters
LeftCameraName                  param_uh_line2_1              # Name of real left camera
VirtualCameraName               param_uh_line2_2             # Name of virtual camera
RightCameraName                 param_uh_line2_3              # Name of real right camera
LeftViewImageName               ../yuv/1.yuv                # Name of left input video
RightViewImageName              ../yuv/3.yuv                # Name of right input video
LeftDepthMapName                ../depthyuv/depth-1-8bit-zn300-zf3000.yuv          # Name of left depth map video
RightDepthMapName               ../depthyuv/depth-3-8bit-zn300-zf3000.yuv          # Name of right depth map video
OutputVirtualViewImageName      virtual2.yuv       # Name of output virtual view video

ColorSpace                      0                       # 0...YUV, 1...RGB
Precision                       2                       # 1...Integer-pel, 2...Half-pel, 4...Quater-pel
Filter                          1                       # 0...(Bi)-linear, 1...(Bi)-Cubic, 2...MPEG-4 AVC

BoundaryNoiseRemoval                                        1                                                               #
Boundary Noise Removal: Updated By GIST

SynthesisMode                   1                       # 0...General, 1...1D parallel

#---- General mode ------
ViewBlending                  1                       # 0...Blend left and right images, 1...Not Blend

#---- 1D mode      ------
#---- In this example, all parameters below are commented and default values will be taken ----
#SplattingOption        2                # 0: disable; 1: Enable for all pixels; 2: Enable only for boundary pixels. Default: 2
#BoundaryGrowth         40               # A parameter to enlarge the boundary area with SplattingOption = 2. Default: 40
#MergingOption          2                 # 0: Z-buffer only; 1: Averaging only; 2: Adaptive merging using Z-buffer and averaging.
Default: 2
#DepthThreshold         75               # A threshold is only used with MergingOption = 2. Range: 0 ~ 255. Default: 75
#HoleCountThreshold     30               # A threshold is only used with MergingOption = 2. Range: 0 ~ 49. Default: 30
```

Figure 55: View Synthesis configuration file example

2.4.3 Recommended VSRS settings

Parameter	Value	Comments
DepthType	1	Defines depth from the origin of 3D space
{Left,Right}NearestDepthValue	-	Obtained directly from depth input data or from the output of DERS
{Left,Right}FarthestDepthValue	-	
ColorSpace	0	YUV
Precision	4	Quarter-pixel
Filter	1	Bi-cubic
BoundaryNoiseRemoval	0	Not applied
SynthesisMode	0/1	0: general camera configuration 1: cameras with 1D parallel configuration
ViewBlending	1	View Blending set to off

ViewBlending will blend the left and right camera view in creating the virtual view. For test sequences that are not color calibrated, ViewBlending masks the luminance change in the transitions between left/right camera pairs in the quality evaluation sweeping tests of Section **Błąd! Nie można odnaleźć źródła odwołania.**. However, ghosting silhouettes are then also more pronounced.

2.4.4 Legacy display support

As the super-multiview displays are not yet widely available, the quality of the sequence can be assessed on classical stereoscopic displays. From all views required (*29*) by the super-multiview display, a single sweep though all of the view is created before display (*35*) (Figure 66)

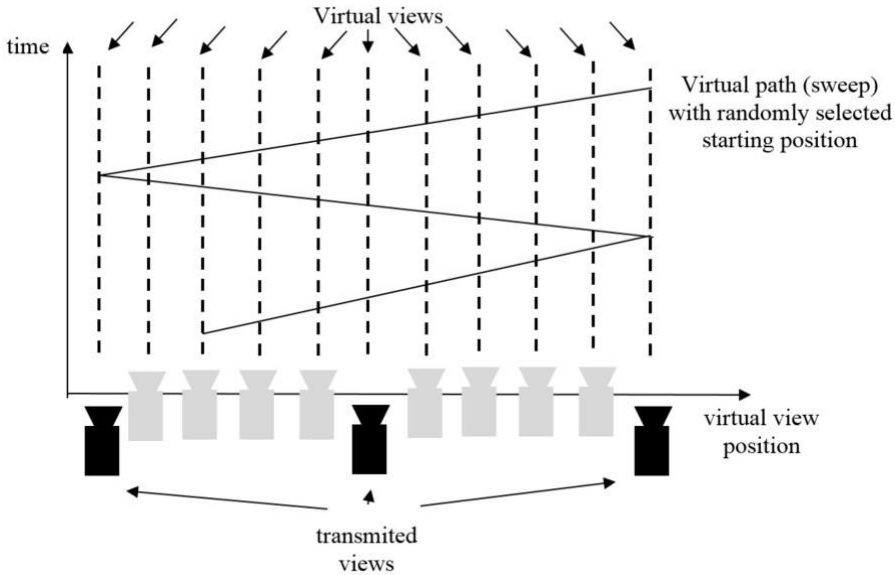


Figure 6: Subjective sweeping test in Free Navigation

3 Annex 1: VSRS Camera parameter format

3.1 Camera parameters principles

The camera parameters are subdivided into two categories:

- the camera intrinsics, i.e. the parameters of each camera individually: camera center c position in world coordinate system, principal point *p* and focal length *f* in pixel units, as shown in the top part of Figure 76 (picture from [CV1])
- the camera extrinsics, i.e. the camera rotation R and translation T, expressed in world coordinates, as shown in the bottom part of Figure 76

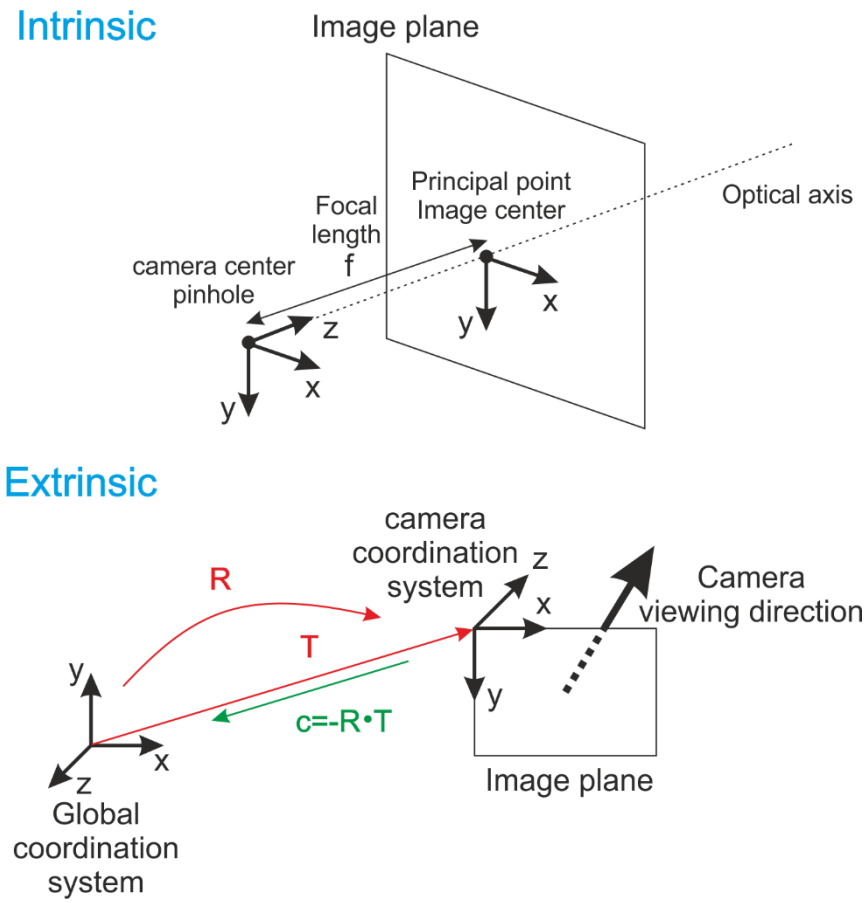


Figure 77: Intrinsics and extrinsics of cameras

Literature often uses the [R|c] matrix at the bottom of Figure 77 for the extrinsics. However, DERS/VSRS use a slightly different format, replacing c by the camera position $T = - R^{-1} * c$, according to [W9595], which annex A is copied in the next section for completeness.

3.2 [N9595] Annex A: Specification of Camera Parameters

Camera parameters shall be specified as rotation matrix **R**, translation vector **t** and intrinsic matrix **A** for each camera i. Values shall be given in floating point precision as accurate as possible. More illustration and explanation is given in Annex B below.

The extrinsic camera parameters **R** and **t** shall be specified according to a right-handed coordinate system. The upper left corner of an image shall be the origin for corresponding image/camera coordinates, i.e., the (0,0) coordinate, with all other corners of the image having non-negative coordinates. With these specifications, a 3-dimensional world point, **wp**=[x y z]^T is mapped into a 2-dimensional camera point, **cp** = s * [u v 1] ^{T}, for the i-th camera according to:

$$s * \mathbf{cp}(i) = \mathbf{A}(i) * \mathbf{R}^{-1}(i) * [\mathbf{wp} - \mathbf{t}(i)]$$

where **A**(i) denotes the intrinsic camera parameters, **R**$^{-1}$(i) denotes the inverse of the rotation matrix **R**(i) and s is an arbitrary scaling chosen to make the third coordinate of **cp** equal to one.

The rotation matrix **R**(i) for i-th camera is represented as follows.

r_11[i]	**r_12**[i]	**r_13**[i]
r_21[i]	**r_22**[i]	**r_23**[i]
r_31[i]	**r_32**[i]	**r_33**[i]

The translation vector **t**(i) for i-th camera is represented as follows:

t_1[i]
t_2[i]
t_3[i]

The rotation matrix **R** and the translation vector **t** define the position and orientation of the corresponding camera with respect to the world coordinate system. The components of the rotation matrix **R** are function of the rotations about the three coordinate axes.

focal_length_x[i] specifies the focal length of the i-th camera in the horizontal direction units of pixels.
focal_length_y[i] specifies the focal length of the i-th camera in the vertical direction in units of pixels.
principal_point_x[i] specifies the principal point of the i-th camera in the horizontal direction units of pixels.
principal_point_y[i] specifies the principal point of the i-th camera in the vertical direction in units of pixels.
radial_distortion[i] specifies the radial distortion coefficient of the i-th camera.

The intrinsic matrix **A**(i) for i-th camera is represented as follows:

focal_length_x[i]	**radial_distortion**[i]	**principal_point_x**[i]
0	**focal_length_y**[i]	**principal_point_y**[i]
0	0	1

4 Annex 2: Blender XML to VSRS Camera parameter conversion script

```
import os, fileinput, string, math, time, sys, shutil, glob, subprocess
from xml.dom import minidom


def HoloVizio2MpegConvertor_LinearLayout(xmlFileName, mpegParamFileName):
# Converting XML Linear HoloVizio 3D Camera system file
# to MPEG camera parameter file format:
# MPEG format:
#         camera_name
#         Intrinsic parameter matrix K 3x3
#         0.0  "Origin for corresponding image"
#         0.0
#         extrinsic parameter matrix P = [R T] 3x4

      parametersFromXmlFile = ['count','PathLength', 'CameraDistance', 'CameraFovX',
\
            'CameraAspect', 'CameraWidth', 'CameraHeight']
      parameterValuesFromXmlFile = [None]* len(parametersFromXmlFile)
# Extract Parameters from XML HoloVizio file:
      xmlFile = minidom.parse(xmlFileName)
      for p in range(0, len(parametersFromXmlFile)):
            if (p == 0):
                  CameraLayout = xmlFile.getElementsByTagName("CameraLayout")
                  parameterValue = int(CameraLayout[0].getAttribute("count"))
            else:
                  #print xmlFile.getElementsByTagName(parametersFromXmlFile[p])
                  parameterValue                                                =
float(xmlFile.getElementsByTagName(parametersFromXmlFile[p])[0].firstChild.data)
            parameterValuesFromXmlFile[p] = float(parameterValue)
      numOfCamera = int(parameterValuesFromXmlFile[0])
      width = parameterValuesFromXmlFile[5]
      height = parameterValuesFromXmlFile[6]
      focalLength                                                               =
0.5*width/(math.tan(math.radians(parameterValuesFromXmlFile[3]/2)));
      cameraInterval = parameterValuesFromXmlFile[1]/numOfCamera;
# Write MPEG Camera Parameter file:
      mpegParamFile = open(mpegParamFileName, 'w')
      for i in range(0, numOfCamera):
            mpegParamFile.write ('\nparam_ptm%02d \n' % i)
            K = [[focalLength,0,width/2], \
                  [0,focalLength,height/2],[0,0,1]];
            P = [[1, 0,0,(0.5*cameraInterval*(i-float(numOfCamera-1)/2))],[0, 1, 0,
0],[0, 0, 1, 0]]
            for r in range(0, len(K)):
                  for c in range(0, len(K)):
                        mpegParamFile.write(str(K[r][c]))
                        mpegParamFile.write(' ')
                  mpegParamFile.write('\n')
            mpegParamFile.write('0.0\n')
            mpegParamFile.write('0.0\n')
            for r in range(0, len(P[:])):
                  for c in range(0, len(P[:][1])):
                        mpegParamFile.write(str(P[r][c]))
                        mpegParamFile.write(' ')
                  mpegParamFile.write('\n')
      mpegParamFile.close()
      return (numOfCamera, width, height)


def HoloVizio2MpegConvertor_ArcLayout(xmlFileName, mpegParamFileName):
# Converting XML Arc HoloVizio 3D Camera system file
# to MPEG camera parameter file format:
# MPEG format:
#         camera_name
#         Intrinsic parameter matrix K 3x3
#         0.0  "Origin for corresponding image"
```

```
#         0.0
#         extrinsic parameter matrix P = [R T] 3x4

        firstCameraLocationInDegree = -22.5
        angularResolution = 0.5
        parametersFromXmlFile = ['count','Angle', 'Radius', 'CameraFovX', \
                'CameraWidth', 'CameraHeight']
        parameterValuesFromXmlFile = [None]* len(parametersFromXmlFile)
# Extract Parameters from XML HoloVizio file:
        xmlFile = minidom.parse(xmlFileName)
        for p in range(0, len(parametersFromXmlFile)):
                if (p == 0):
                        CameraLayout = xmlFile.getElementsByTagName("CameraLayout")
                        parameterValue = int(CameraLayout[0].getAttribute("count"))
                else:
                        parameterValue                                                  =
float(xmlFile.getElementsByTagName(parametersFromXmlFile[p])[0].firstChild.data)
                parameterValuesFromXmlFile[p] = float(parameterValue)
        numOfCamera = int(parameterValuesFromXmlFile[0])
        angle = parameterValuesFromXmlFile[1]
        radius = parameterValuesFromXmlFile[2]
        cameraFovX = parameterValuesFromXmlFile[3]
        width = parameterValuesFromXmlFile[4]
        height = parameterValuesFromXmlFile[5]
        focalLength = 0.5*width/(math.tan(math.radians(cameraFovX/2)));
        beta = firstCameraLocationInDegree
# Write MPEG Camera Parameter file:
        mpegParamFile = open(mpegParamFileName, 'w')
        for i in range(0, numOfCamera):
                mpegParamFile.write ('\nparam_ptm%02d \n' % i)
                K = [[focalLength,0,width/2], \
                        [0,focalLength,height/2],[0,0,1]]
                xt = radius*math.sin(math.radians(beta))
                yt = 0;
                zt = -radius*math.cos(math.radians(beta))
                P = [[math.cos(math.radians(beta)), 0, math.sin(math.radians(beta)), xt]\
                        ,[0, 1, 0, yt]\
                        ,[-math.sin(math.radians(beta)),  0,  math.cos(math.radians(beta)),
zt]]
                beta  += angularResolution
                for r in range(0, len(K)):
                        for c in range(0, len(K)):
                                mpegParamFile.write(str(K[r][c]))
                                mpegParamFile.write(' ')
                        mpegParamFile.write('\n')
                mpegParamFile.write('0.0\n')
                mpegParamFile.write('0.0\n')
                for r in range(0, len(P[:])):
                        for c in range(0, len(P[:][1])):
                                mpegParamFile.write(str(P[r][c]))
                                mpegParamFile.write(' ')
                        mpegParamFile.write('\n')
        mpegParamFile.close()
        return (numOfCamera, width, height)


HoloVizio2MpegConvertor_LinearLayout('BigBuckBunny_Flowers_Linear.xml',
'BigBuckBunny_Flowers_Linear.txt')
HoloVizio2MpegConvertor_ArcLayout('BigBuckBunny_Flowers_Arc.xml',
'BigBuckBunny_Flowers_Arc.txt')
```

5 Annex 3: Virtual View Camera parameter interpolation with Excel

5.1 Spherical Linear Interpolation

To create virtual views, also their corresponding camera parameters should be properly provided. For doing so, the rotation R and translation T of Figure 7 should be smoothly interpolated between all successive camera viewpoints. While a spline interpolation between the camera positions will clearly create virtual intermediate positions properly, it's not obvious that spline interpolation on the matrix columns of the rotation matrix R will be adequate. We will show in this annex that it is.

Computer graphics textbooks (e.g. [3DMaths2011]) explain that a rotation with angle θ around an axis **A** can be represented by a quaternion **q** (i.e. a complex number with 1 real and 3 imaginary components):

$$\mathbf{q} = \cos\frac{\theta}{2} + \mathbf{A}\sin\frac{\theta}{2}$$

Hence, a smooth rotational transition from the rotation matrix R1 with quaternion **q1** of a camera view to the rotation matrix R2 with quaternion **q2** of the next camera view can be obtained with a spherical linear interpolation along a spherical arc, as shown in Figure 8, with the transformation formula at the right side of Figure 9.

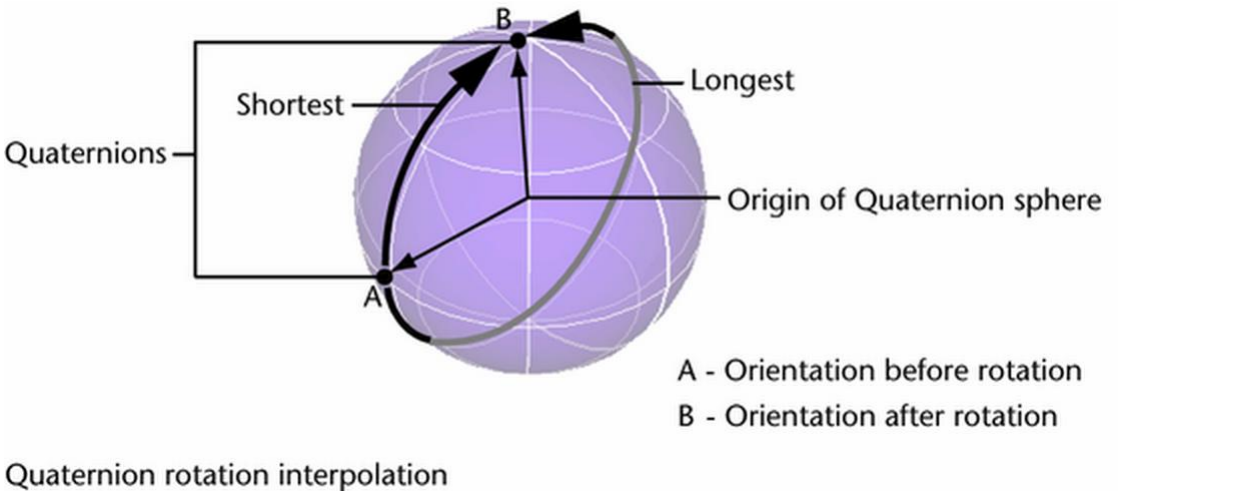
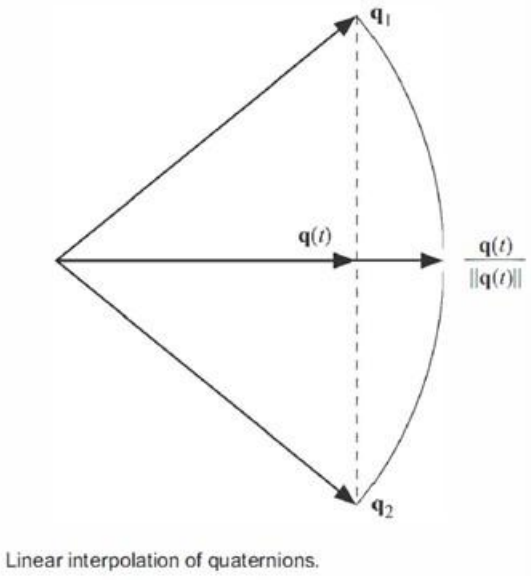
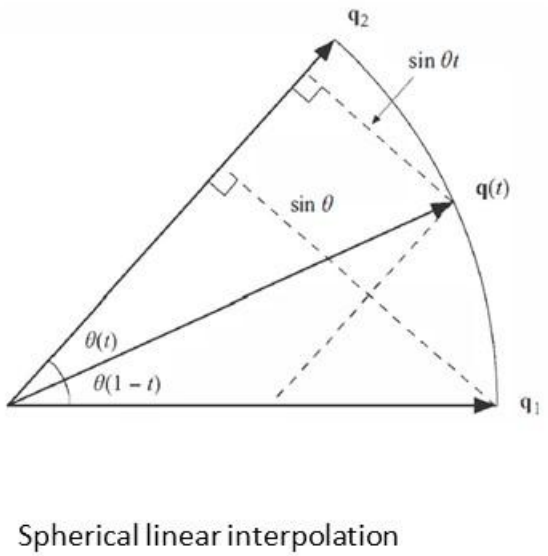


Figure 8: Spherical linear interpolation along a spherical arc





$$\mathbf{q}(t) = (1-t)\mathbf{q}_1 + t\mathbf{q}_2$$

Normalized: $$\mathbf{q}(t) = \frac{(1-t)\mathbf{q}_1 + t\mathbf{q}_2}{\|(1-t)\mathbf{q}_1 + t\mathbf{q}_2\|}$$

$$\mathbf{q}(t) = \frac{\sin\theta(1-t)}{\sin\theta}\mathbf{q}_1 + \frac{\sin\theta t}{\sin\theta}\mathbf{q}_2$$

The angle θ is given by

$$\theta = \cos^{-1}(\mathbf{q}_1 \cdot \mathbf{q}_2),$$

Figure 9: Linear (left) and spherical linear interpolation (right) on quaternions

The so obtained quaternion **q**=<w,x,y,z> corresponding to an intermediate virtual view at parametric position t (0<t<1) can readily be transformed into the corresponding rotation matrix R, using the following transformation formula:

$$\mathbf{R} = \begin{bmatrix} 1-2y^2-2z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & 1-2x^2-2z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & 1-2x^2-2y^2 \end{bmatrix}$$

Interestingly, since any projection onto a plane of the spherical arc in Figure 8 is an ellipse, any good interpolation mechanism on an ellipse will be able to mimic the spherical linear interpolation properly. Spline interpolation perfectly satisfies this condition as exemplified in the left side of Figure 20. Applying spline interpolation on the rotation matrix components yields the results of Figure 101 for the example of the Soccer Arc1 sequence.

Also note in Figure 20 that spline interpolation also preserves a linear behavior, which can hence handle linear camera arrays properly.
For this and all above reasons, spline interpolation has been selected to calculate the virtual viewpoints' camera parameters.

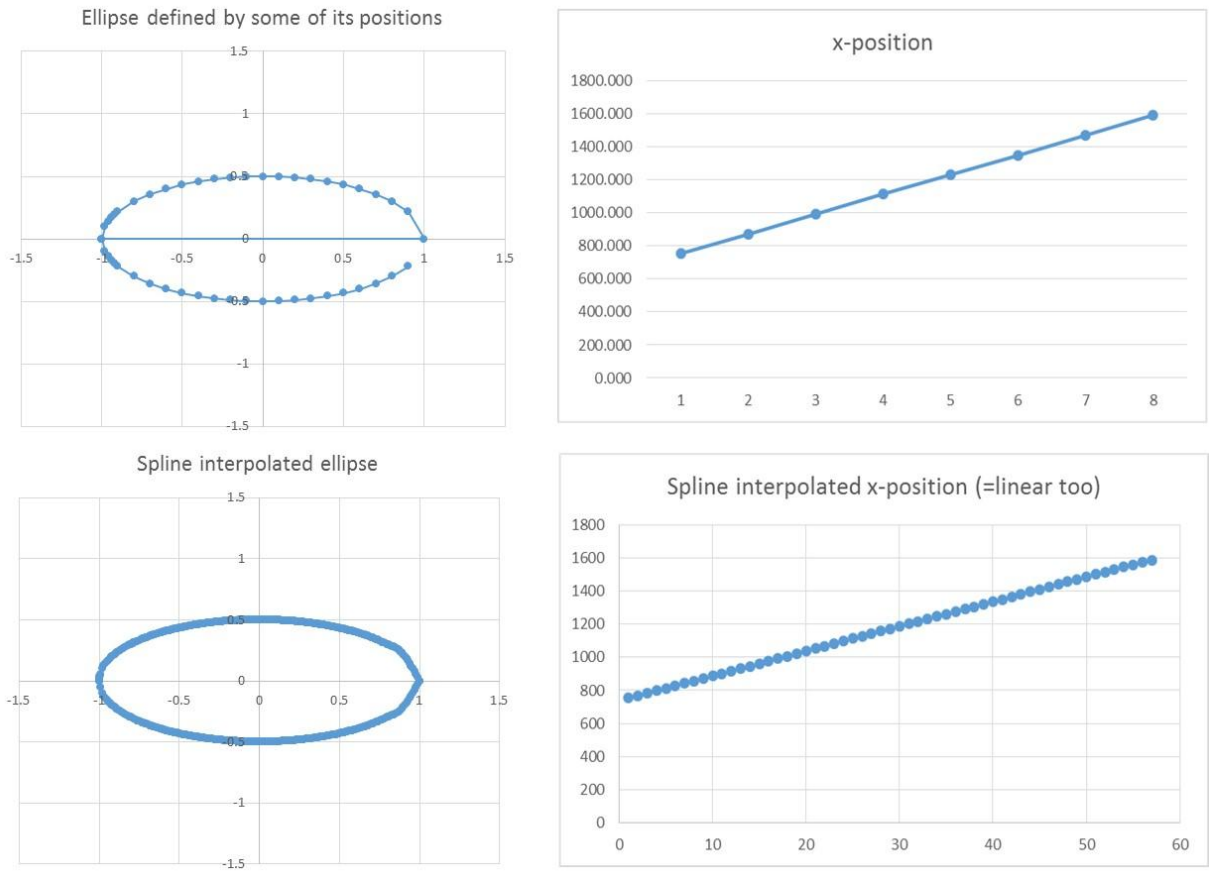
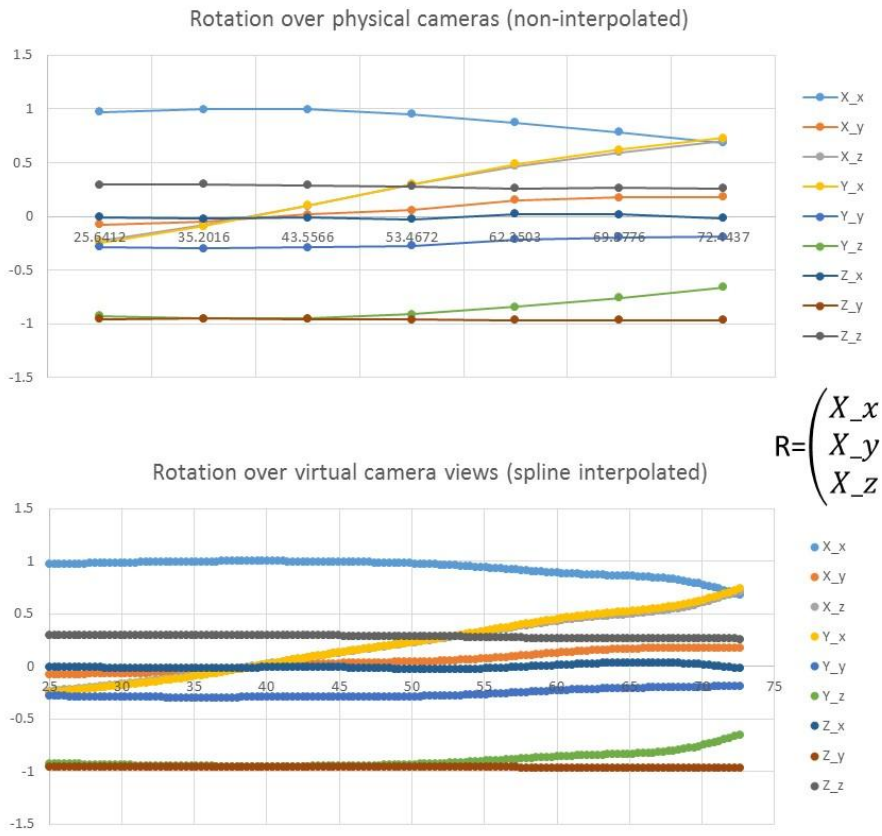


Figure 20: Spline interpolation on an ellipse (left) and a straight line (right)



$$R=\begin{pmatrix} X\_x & Y\_x & Z\_x \\ X\_y & Y\_y & Z\_y \\ X\_z & Y\_z & Z\_z \end{pmatrix}$$

Figure 10: Spline interpolation for calculating the rotation matrix of the virtual viewpoints

5.2 Excel sheet for virtual camera parameters

All virtual camera parameters are calculated with Excel using the spline interpolation explained in previous section. The excel spline macro v3.2 downloadable from http://srs1-cubic-spline-for-excel.software.informer.com/download/ has been used for this purpose. How to use this macro is further explained in https://www.youtube.com/watch?v=HpMjdnk-Drg

The physical camera parameters should be input in the Cam_parameter_input_file sheet, and virtual camera parameters are output in the Virtual_Views_parameters sheet, following the flow given in **Błąd! Nie można odnaleźć źródła odwołania.**.

The output in the light/dark grey zones can be copied to a txt file that can be readily used in VSRS.
Note at the top-right side of **Błąd! Nie można odnaleźć źródła odwołania.**2 that names are automatically created for these virtual views, using the naming convention "virtual_XX.xx_from_Y_Z", where XX.xx is the translation corresponding to the virtual camera and (Y,Z) are the left and right views used for the view synthesis with VSRS.

Also note the X_norm, Y_norm and the Z_norm columns in the excel sheet that are all 1, clearly indicating that the spline interpolation preserves the norm of the rotation basis vectors corresponding to the columns of the rotation matrix.

The X_ortho_Y, Y_ortho_Z and X_ortho_Z columns confirm the scalar product between these basis vectors is zero, clearly indicating the that spline interpolation preserves the orthogonality of the basis vectors in the rotation matrix.

Hence, the Excel sheet well approximates the interpolation on the physical camera parameters to obtain the virtual camera parameters with a simple to use tool.

6 Annex 4: Y to YUV420 conversion script

The Y to YUV420 transformation relies on the principle that in YUV420 there are – per frame – half as many UV data values (set to the default value of 128 = no chrominance) as the number of Y values. Hence, it is sufficient to read each Y-frame and add an array of value 128 with the appropriate size (cf. UV = bytearray([128]*(NpixelsPerFrame>>1)*NbytesPerElement) to create a YUV420 file.

```
# ----------- Read the input file ------------
# all data elements are 1 byte(s) long
NbytesPerElement = 1
# size of one frame in number of pixels
NpixelsPerFrame = 1600 * 1200
# first, last and no. of frames to read
StartFrame = 0         # starts at 0 for the first frame
NumberOfFrames = 1267  # NumberOfFrames>500 heavily slows down the processing (couple
of minutes)
StopFrame = StartFrame+NumberOfFrames-1
```

```
# ----------- File definition ----------------
FilenameBase = 'images/depth-7-8bit-zn300-zf3000'
f_in = open(FilenameBase + '.yuv', 'rb')
f_out = open(FilenameBase +'_Y420_'+ str(StartFrame) +'_'+ str(StopFrame) +'.yuv',
'wb')

# UV as a chunck of successive values of 128
UV = bytearray([128]*(NpixelsPerFrame>>1)*NbytesPerElement)

# start
print('Processing started')

# go to the first frame
f_in.seek((StartFrame)*NpixelsPerFrame*NbytesPerElement)

# Copy frame per frame, adding UV components per frame
for i in range(NumberOfFrames):
    # copy one frame to the output
    OneFrame = f_in.read(NpixelsPerFrame*NbytesPerElement)
    f_out.write(OneFrame)
    # add U and V component per frame
    f_out.write(UV)

# close all files
f_in.close()
f_out.close()

# done
print('Done!')
```

7 Annex 5: Code changes for use of HTM13 in MPEG-FTV

Following lines of code should be commented out of 3D-HEVC for proper interfacing with the DERS/VSRS workflow explained in the current document:

```
In TLibCommon/TypeDef.h, line 57,
#define HEVC_EXT 2 // 3D-HEVC mode // define 2 for 3D-HEVC mode.

In TAppEncoder/TAppEncTop.cpp Line 1336:
Int* aiIdx2DepthValue = (Int*) calloc(uiMaxDepthValue+1, sizeof(Int)); //Fix
Fix Owieczka +1 Added

In TLibEncoder/TEncCavlc.cpp line 444:
WRITE_CODE(uiNumDepthValues_coded,   9,   "num_depth_values_in_dlt[i]");   //
num_entry //Fix Fix Owieczka 8->9

In TLibDecoder/TDecCavlc.cpp line 470:
READ_CODE(9, uiNumDepthValues, "num_depth_values_in_dlt[i]");   //> num_entry
//Fix Fix Owieczka 8->9

In /Lib/TLibDecoder/TDecCAVLC.cpp,
line 235: // assert (uiCode <= 15); // comment it out
line 771: // assert (uiCode <= 15); // comment it out

In TLibCommon/TComSlice.h,
line 3039: // assert (psId < m_maxId); // comment it out
```

8 Annex 6: Use of VSRS

The following steps are needed to properly configure MPEG's View Synthesis Reference Software (VSRS) for the tests to be performed in the Free-viewpoint TV (FTV) ad-hoc group. This configuration has been tested on 64-bit Windows 7 with Visual Studio 10, but should be easily adaptable to other Windows versions.

- Download the reference software from SVN repository (including the *opencv* folder) using MPEG user and password: http://wg11.sc29.org/svn/repos/Explorations/FTV.
- Go to folder *vsrs/trunk/windows* and open the Visual Studio file corresponding to your version.
- In Visual Studio, open the file **version.h**, located under project *ViewSynLibStaticVC10*, and check that the global definitions are

```
//#define POZNAN_GENERAL_HOMOGRAPHY
//#define POZNAN_16BIT_DEPTH
#define POZNAN_DEPTHMAP_CHROMA_FORMAT 400
```

  Note that the second line should be uncommented in case 16-bit depth maps are being used, and the third line should be commented if depth maps are provided in a 420 format.
- Build the solution, which will result in having the working VSRS executable file in **vsrs/trunk/windows/x64/ViewSynVC10.exe**
- Once VSRS is compiled, it can be executed from command line using the configuration file as its first and only parameter, that is, as in `ViewSynVC10.exe <configuration_file.cfg>`.
- Before running the program, check that configuration file is fine, especially regarding the following options:

Parameter	Value	Comments
DepthType	1	Defines depth from the origin of 3D space
{Left,Right}NearestDepthValue	-	Obtained directly from depth input data or from the output of DERS
{Left,Right}FarthestDepthValue	-	
ColorSpace	0	YUV
Precision	4	Quarter-pixel
Filter	1	Bi-cubic
BoundaryNoiseRemoval	0	Not applied
SynthesisMode	0/1	0: general camera configuration 1: cameras with 1D parallel configuration

- Besides, check that the camera parameters file corresponds to the depth maps being used, and that these are in the VSRS expected format (where white values means objects closer to the camera). For instance, there are several depth maps of sequence SoccerArc on MPEG repository, but the right ones are located under **/MPEG-04/Part02-Visual/FTV_AhG/UHasselt_Soccer/Arc1/Disparity-yuv400**.

9 Annex 7: Python code for multi-threaded VSRS

To automate view synthesis process with the use of VSRS a python script have been developed. For further details please see M36506.

10 Annex 8: example 3D-HEVC configuration file for super multi-view case

Since all configuration file of 3D-HEVC encoder for 40 view is quite long, only key line have been puted below. Entre file can be found have been attached to this document.

```
#======== Legend for comments =====================
# (m) specification per layer/dimension/layerset possible
# (c) cyclic repetition of values, if not given for all layers/dimensions/layersets. (e.g. 5 layers and 1 2 3 -> 1 2 3 1 2  )

#======== File I/O =====================
InputFile_0                : D:\FTV\Butterfly_MV5-45\Data\BBB_Butterfly_cam0025.yuv
InputFile_1                : D:\FTV\Butterfly_MV5-45\Data\BBB_Butterfly_cam0024.yuv
InputFile_2                : D:\FTV\Butterfly_MV5-45\Data\BBB_Butterfly_cam0023.yuv

• • •
InputFile_37               : D:\FTV\Butterfly_MV5-45\Data\BBB_Butterfly_cam0042.yuv
InputFile_38               : D:\FTV\Butterfly_MV5-45\Data\BBB_Butterfly_cam0043.yuv
InputFile_39               : D:\FTV\Butterfly_MV5-45\Data\BBB_Butterfly_cam0044.yuv

BitstreamFile              : stream.bit

ReconFile_0                : rec_25.yuv
ReconFile_1                : rec_24.yuv
ReconFile_2                : rec_23.yuv

• • •
ReconFile_37               : rec_42.yuv
ReconFile_38               : rec_43.yuv
ReconFile_39               : rec_44.yuv

NumberOfLayers             : 40          # number of layers to be coded

TargetEncLayerIdList       :             # Layer Id in Nuh to be encoded, (empty:-> all layers will be encode)

FramesToBeEncoded          : 120         # Number of frames to be coded
FrameRate                  : 24          # Frame Rate per second
SourceWidth                : 1280        # Input  frame width
SourceHeight               : 768         # Input  frame height

QP                         : 30                       # quantization parameter (view only)

#======== VPS ============================
ScalabilityMask            : 2           # Scalability Mask          ( 2: View Scalability, shall be 2 for MV-HEVC )
DimensionIdLen             : 6           # Number of bits to store Ids,  per scalability dimension, (m)
ViewOrderIndex             : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39  # ViewOrderIndex, per layer (m)
DepthFlag                  : 0 0 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 # DepthFlag  (m)
LayerIdInNuh               : 0           # Layer Id in NAL unit header, (0: no explicit signalling, otherwise per layer ) (m)
SplittingFlag              : 0           # Splitting Flag
ViewId                     : 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39   # ViewId, per ViewOrderIndex (m)

#======== VPS / Layer sets ================
VpsNumLayerSets            : 41          # Number of layer sets
LayerIdsInSet_0            : 0           # Indices in VPS of layers in layer set 0
LayerIdsInSet_1            : 0 1         # Indices in VPS of layers in layer set 1

• • •
LayerIdsInSet_37           : 36 37
LayerIdsInSet_38           : 37 38
LayerIdsInSet_39           : 38 39
LayerIdsInSet_40           : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39


#======== VPS / Output layer sets ================

DefaultTargetOutputLayerIdc    : 0       # Specifies output layers of layer sets, 0: output all layers, 1: output highest layer, 2: specified by LayerIdsInDefOutputLayerSet
#OutputLayerSetIdx             : 2 3     # Indices of layer sets used to derive additional output layer sets
#LayerIdsInAddOutputLayerSet_0 : 2 3     # Indices in VPS of output layers in additional output layer set 0
#LayerIdsInAddOutputLayerSet_1 : 4 5     # Indices in VPS of output layers in additional output layer set 1

#======== VPS / PTLI ================
#Profile                   : main main 3d-main       # Profile indication in VpsProfileTierLevel, per VpsProfileTierLevel syntax structure  (m)
#Level                     : none none none          # Level   indication in VpsProfileTierLevel, per VpsProfileTierLevel syntax structure  (m)
#Tier                      : main main main          # Tier    indication in VpsProfileTierLevel, per VpsProfileTierLevel syntax structure  (m)
#InblFlag                  : 0   0   0               # Inbl    indication in VpsProfileTierLevel, per VpsProfileTierLevel syntax structure  (m)

ProfileTierLevelIdx_0      : 1    # VpsProfileTierLevel indices of layers in output layer set 0 (m) (should be -1, when layer is not necessary)
ProfileTierLevelIdx_1      : 1 2  # VpsProfileTierLevel indices of layers in output layer set n (m) (should be -1, when layer is not necessary)
ProfileTierLevelIdx_2      : 1 2  # VpsProfileTierLevel indices of layers in output layer set n (m) (should be -1, when layer is not necessary)
```

```
• • •
ProfileTierLevelIdx_37          : 1 2  # VpsProfileTierLevel indices of layers in output layer set n (m) (should be -1, when layer is not necessary)
ProfileTierLevelIdx_38          : 1 2  # VpsProfileTierLevel indices of layers in output layer set n (m) (should be -1, when layer is not necessary)
ProfileTierLevelIdx_39          : 1 2  # VpsProfileTierLevel indices of layers in output layer set n (m) (should be -1, when layer is not necessary)
ProfileTierLevelIdx_40          : 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2  # VpsProfileTierLevel indices

#======== VPS / Dependencies ================
DirectRefLayers_1         : 0        # Indices in VPS of direct reference layers
DirectRefLayers_2         : 1        # Indices in VPS of direct reference layers
DirectRefLayers_3         : 2        # Indices in VPS of direct reference layers

• • •
DirectRefLayers_37        : 36       # Indices in VPS of direct reference layers
DirectRefLayers_38        : 37       # Indices in VPS of direct reference layers
DirectRefLayers_39        : 38       # Indices in VPS of direct reference layers

DependencyTypes_1         : 2        # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion
DependencyTypes_2         : 2        # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion
DependencyTypes_3         : 2        # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion

• • •
DependencyTypes_37        : 2        # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion
DependencyTypes_38        : 2        # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion
DependencyTypes_39        : 2        # Dependency types of direct reference layers, 0: Sample 1: Motion 2: Sample+Motion

#========== Camera parameters ==========
BaseViewCameraNumbers       : 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 # camera numbers of coded views ( in coding order per view )
CameraParameterFile         : D:\FTV\Butterfly_3D5-44\cam_param_Butterfly_Arc_view5-44_for_HTM13.txt  # camera parameter file
CodedCamParsPrecision       : 5        # precision used for coding of camera parameters (in units of 2^(-x) luma samples)

#======== Unit definition ================
MaxCUWidth              : 64      # Maximum coding unit width in pixel
MaxCUHeight             : 64      # Maximum coding unit height in pixel
MaxPartitionDepth       : 4       # Maximum coding unit depth
QuadtreeTULog2MaxSize      : 5       # Log2 of maximum transform size for
                                     # quadtree-based TU coding (2...6)
QuadtreeTULog2MinSize      : 2       # Log2 of minimum transform size for
                                     # quadtree-based TU coding (2...6)
QuadtreeTUMaxDepthInter    : 3
QuadtreeTUMaxDepthIntra    : 3

#======== Coding Structure =============
IntraPeriod             : 24      # Period of I-Frame ( -1 = only first)
DecodingRefreshType        : 1       # Random Accesss 0:none, 1:CDR, 2:IDR
GOPSize                 : 8       # GOP Size (number of B slice = GOPSize-1)

#                QPfactor   betaOffsetDiv2  #ref_pics_active  reference pictures   deltaRPS   reference idcs     ilPredLayerIdc    refLayerPicPosIl_L1
#       Type  POC QPoffset   tcOffsetDiv2    temporal_id    #ref_pics          predict   #ref_idcs     #ActiveRefLayerPics   refLayerPicPosIl_L0

Frame1:   B  8  1   0.442   0    0    0    4   4   -8 -10 -12 -16   0   0
Frame2:   B  4  2   0.3536  0    0    0    2   3   -4 -6 4       1   4   5   1 1 0 0 1    0
Frame3:   B  2  3   0.3536  0    0    0    2   4   -2 -4 2 6     1   2   4   1 1 1 1      0
Frame4:   B  1  4   0.68    0    0    0    2   4   -1 1 3 7      1   1   5   1 0 1 1 1    0
Frame5:   B  3  4   0.68    0    0    0    2   4   -1 -3 1 5     1  -2   5   1 1 1 1 0    0
Frame6:   B  6  3   0.3536  0    0    0    2   4   -2 -4 -6 2    1  -3   5   1 1 1 1 0    0
Frame7:   B  5  4   0.68    0    0    0    2   4   -1 -5 1 3     1   1   5   1 0 1 1 1    0
Frame8:   B  7  4   0.68    0    0    0    2   4   -1 -3 -7 1    1  -2   5   1 1 1 1 0    0

FrameI_l1:  P   0   3   0.442   0    0    0    1   0              0                   1    0    0    -1
Frame1_l1:  B   8   4   0.442   0    0    0    4   4   -8 -10 -12 -16   0                 1    0    1    -1
Frame2_l1:  B   4   5   0.3536  0    0    0    3   3   -4 -6 4       1   4   5   1 1 0 0 1    1    0    1    -1
Frame3_l1:  B   2   6   0.3536  0    0    0    3   4   -2 -4 2 6     1   2   4   1 1 1 1      1    0    1    -1
Frame4_l1:  B   1   7   0.68    0    0    0    3   4   -1 1 3 7      1   1   5   1 0 1 1 1    1    0    1    -1
Frame5_l1:  B   3   7   0.68    0    0    0    3   4   -1 -3 1 5     1  -2   5   1 1 1 1 0    1    0    2    -1
Frame6_l1:  B   6   6   0.3536  0    0    0    3   4   -2 -4 -6 2    1  -3   5   1 1 1 1 0    1    0    2    -1
Frame7_l1:  B   5   7   0.68    0    0    0    3   4   -1 -5 1 3     1   1   5   1 0 1 1 1    1    0    2    -1
Frame8_l1:  B   7   7   0.68    0    0    0    3   4   -1 -3 -7 1    1  -2   5   1 1 1 1 0    1    0    2    -1

FrameI_l2:  P   0   3   0.442   0    0    0    1   0              0                   1    0    0    -1
Frame1_l2:  B   8   4   0.442   0    0    0    4   4   -8 -10 -12 -16   0                 1    0    1    -1
Frame2_l2:  B   4   5   0.3536  0    0    0    3   3   -4 -6 4       1   4   5   1 1 0 0 1    1    0    1    -1
Frame3_l2:  B   2   6   0.3536  0    0    0    3   4   -2 -4 2 6     1   2   4   1 1 1 1      1    0    1    -1
Frame4_l2:  B   1   7   0.68    0    0    0    3   4   -1 1 3 7      1   1   5   1 0 1 1 1    1    0    1    -1
Frame5_l2:  B   3   7   0.68    0    0    0    3   4   -1 -3 1 5     1  -2   5   1 1 1 1 0    1    0    2    -1
Frame6_l2:  B   6   6   0.3536  0    0    0    3   4   -2 -4 -6 2    1  -3   5   1 1 1 1 0    1    0    2    -1
Frame7_l2:  B   5   7   0.68    0    0    0    3   4   -1 -5 1 3     1   1   5   1 0 1 1 1    1    0    2    -1
Frame8_l2:  B   7   7   0.68    0    0    0    3   4   -1 -3 -7 1    1  -2   5   1 1 1 1 0    1    0    2    -1

• • •
FrameI_l38:  P   0   3   0.442   0    0    0    1   0              0                   1    0    0    -1
Frame1_l38:  B   8   4   0.442   0    0    0    4   4   -8 -10 -12 -16   0                 1    0    1    -1
Frame2_l38:  B   4   5   0.3536  0    0    0    3   3   -4 -6 4       1   4   5   1 1 0 0 1    1    0    1    -1
Frame3_l38:  B   2   6   0.3536  0    0    0    3   4   -2 -4 2 6     1   2   4   1 1 1 1      1    0    1    -1
```

```
Frame4_l38:  B   1    7    0.68    0       0       0       3     4     -1  1  3 7        1     1     5     1 0 1 1 1        1        0        1        -1
Frame5_l38:  B   3    7    0.68    0       0       0       3     4     -1 -3  1 5        1     -2    5     1 1 1 1 0        1        0        2        -1
Frame6_l38:  B   6    6    0.3536  0       0       0       3     4     -2 -4 -6 2        1     -3    5     1 1 1 1 0        1        0        2        -1
Frame7_l38:  B   5    7    0.68    0       0       0       3     4     -1 -5  1 3        1     1     5     1 0 1 1 1        1        0        2        -1
Frame8_l38:  B   7    7    0.68    0       0       0       3     4     -1 -3 -7 1        1     -2    5     1 1 1 1 0        1        0        2        -1

FrameI_l39:  P   0    3    0.442   0       0       0       1     0                 0                                1        0        0        -1
Frame1_l39:  B   8    4    0.442   0       0       0       4     4     -8 -10 -12 -16    0                                1        0        1        -1
Frame2_l39:  B   4    5    0.3536  0       0       0       3     3     -4 -6  4          1     4     5     1 1 0 0 1        1        0        1        -1
Frame3_l39:  B   2    6    0.3536  0       0       0       3     4     -2 -4  2 6        1     2     4     1 1 1 1          1        0        1        -1
Frame4_l39:  B   1    7    0.68    0       0       0       3     4     -1  1  3 7        1     1     5     1 0 1 1 1        1        0        1        -1
Frame5_l39:  B   3    7    0.68    0       0       0       3     4     -1 -3  1 5        1     -2    5     1 1 1 1 0        1        0        2        -1
Frame6_l39:  B   6    6    0.3536  0       0       0       3     4     -2 -4 -6 2        1     -3    5     1 1 1 1 0        1        0        2        -1
Frame7_l39:  B   5    7    0.68    0       0       0       3     4     -1 -5  1 3        1     1     5     1 0 1 1 1        1        0        2        -1
Frame8_l39:  B   7    7    0.68    0       0       0       3     4     -1 -3 -7 1        1     -2    5     1 1 1 1 0        1        0        2        -1


#========== Motion Search =============
FastSearch                  : 1          # 0:Full search  1:TZ search
SearchRange                 : 96         # (0: Search range is a Full frame)
BipredSearchRange           : 4          # Search range for bi-prediction refinement
HadamardME                  : 1          # Use of hadamard measure for fractional ME
FEN                         : 1          # Fast encoder decision
FDM                         : 1          # Fast Decision for Merge RD cost

#======== Quantization =============

MaxDeltaQP                  : 0          # CU-based multi-QP optimization
MaxCuDQPDepth               : 0          # Max depth of a minimum CuDQP for sub-LCU-level delta QP
DeltaQpRD                   : 0          # Slice-based multi-QP optimization
RDOQ                        : 1          # RDOQ
RDOQTS                      : 1          # RDOQ for transform skip

#========== Deblock Filter ============
DeblockingFilterControlPresent: 1        # Dbl control params present (0=not present, 1=present) (mc)
LoopFilterOffsetInPPS       : 0          # Dbl params: 0=varying params in SliceHeader, param = base_param + GOP_offset_param; 1=constant params in PPS, param = base_param)
LoopFilterDisable           : 0 1        # Disable deblocking filter (0=Filter, 1=No Filter) (mc)
LoopFilterBetaOffset_div2   : 0          # base_param: -6 ~ 6
LoopFilterTcOffset_div2     : 0          # base_param: -6 ~ 6
DeblockingFilterMetric      : 0          # blockiness metric (automatically configures deblocking parameters in bitstream)

#========== Misc. ============
InternalBitDepth            : 8          # codec operating bit-depth

#========== Coding Tools =================
SAO                         : 1 0        # Sample adaptive offset  (0: OFF, 1: ON) (mc)
AMP                         : 1          # Asymmetric motion partitions (0: OFF, 1: ON)
TransformSkip               : 1          # Transform skipping (0: OFF, 1: ON)
TransformSkipFast           : 1          # Fast Transform skipping (0: OFF, 1: ON)
SAOLcuBoundary              : 0          # SAOLcuBoundary using non-deblocked pixels (0: OFF, 1: ON)

#============ Slices ================
SliceMode                : 0             # 0: Disable all slice options.
                                         # 1: Enforce maximum number of LCU in an slice,
                                         # 2: Enforce maximum number of bytes in an 'slice'
                                         # 3: Enforce maximum number of tiles in a slice
SliceArgument            : 1500          # Argument for 'SliceMode'.
                                         # If SliceMode==1 it represents max. SliceGranularity-sized blocks per slice.
                                         # If SliceMode==2 it represents max. bytes per slice.
                                         # If SliceMode==3 it represents max. tiles per slice.

LFCrossSliceBoundaryFlag : 1             # In-loop filtering, including ALF and DB, is across or not across slice boundary.
                                         # 0:not across, 1: across

#============ PCM ================
PCMEnabledFlag                  : 0      # 0: No PCM mode
PCMLog2MaxSize                  : 5      # Log2 of maximum PCM block size.
PCMLog2MinSize                  : 3      # Log2 of minimum PCM block size.
PCMInputBitDepthFlag            : 1      # 0: PCM bit-depth is internal bit-depth. 1: PCM bit-depth is input bit-depth.
PCMFilterDisableFlag            : 0      # 0: Enable loop filtering on I_PCM samples. 1: Disable loop filtering on I_PCM samples.

#============ Tiles ================
TileUniformSpacing              : 0      # 0: the column boundaries are indicated by TileColumnWidth array, the row boundaries are indicated by TileRowHeight array
#UniformSpacingIdc              : 0      # 0: the column boundaries are indicated by ColumnWidth array, the row boundaries are indicated by RowHeight array
                                         # 1: the column and row boundaries are distributed uniformly
NumTileColumnsMinus1            : 0      # Number of columns in a picture minus 1
TileColumnWidthArray            : 2 3    # Array containing ColumnWidth values in units of LCU (from left to right in picture)
NumTileRowsMinus1               : 0      # Number of rows in a picture minus 1
TileRowHeightArray              : 2      # Array containing RowHeight values in units of LCU (from top to bottom in picture)
LFCrossTileBoundaryFlag         : 1      # In-loop filtering is across or not across tile boundary.
                                         # 0:not across, 1: across

#============ WaveFront ================
WaveFrontSynchro                : 0      # 0:  No WaveFront synchronisation (WaveFrontSubstreams must be 1 in this case).
                                         # >0: WaveFront synchronises with the LCU above and to the right by this many LCUs.

#========== Quantization Matrix =================
ScalingList                 : 0          # ScalingList 0 : off, 1 : default, 2 : file read
```

```
ScalingListFile             : scaling_list.txt      # Scaling List file name. If file is not exist, use Default Matrix.

#============ Lossless ================
TransquantBypassEnableFlag    : 0  # Value of PPS flag.
CUTransquantBypassFlagForce                  : 0 # Constant lossless-value signaling per CU, if TransquantBypassEnableFlag is 1.

#============ Rate Control ======================
RateControl                   : 0             # Rate control: enable rate control
TargetBitrate                 : 1000000       # Rate control: target bitrate, in bps
KeepHierarchicalBit           : 1             # Rate control: keep hierarchical bit allocation in rate control algorithm
LCULevelRateControl           : 1             # Rate control: 1: LCU level RC; 0: picture level RC
RCLCUSeparateModel            : 1             # Rate control: use LCU level separate R-lambda model
InitialQP                     : 0             # Rate control: initial QP
RCForceIntraQP                : 0             # Rate control: force intra QP to be equal to initial QP

#========== multiview coding tools ==========
IvMvPredFlag                  : 1 1           # Inter-view motion prediction
IvResPredFlag                 : 1             # Advanced inter-view residual prediction (0:off, 1:on)
IlluCompEnable                : 1             # Enable Illumination compensation ( 0: off, 1: on )   (v/d)
IlluCompLowLatencyEnc         : 0             # Enable low-latency Illumination compensation encoding( 0: off, 1: on )
ViewSynthesisPredFlag         : 0             # View synthesis prediction
DepthRefinementFlag           : 0             # Disparity refined by depth DoNBDV
IvMvScalingFlag               : 1             # Interview motion vector scaling
Log2SubPbSizeMinus3           : 0             # Log2 of sub-PU size minus 3 for IvMvPred (0 ... 3) and smaller than or equal to log2(maxCUSize)-3
Log2MpiSubPbSizeMinus3        : 0             # Log2 of sub-PU size minus 3 for MPI (0 ... 3) and smaller than or equal to log2(maxCUSize)-3
DepthBasedBlkPartFlag         : 0             # Depth-based Block Partitioning

#========== depth coding tools ==========
VSO                           : 0             # use of view synthesis optimization for depth coding
IntraWedgeFlag                : 0
IntraContourFlag              : 0             # use of intra-view prediction mode
IntraSdcFlag                  : 0
DLT                           : 0
QTL                           : 0
QtPredFlag                    : 0
InterSdcFlag                  : 0                     # use of inter sdc
MpiFlag                       : 0
IntraSingleFlag               : 0             # use of single depth mode

#========== view synthesis optimization (VSO) ==========
#VSOConfig                    : [cx0 B(cc1) I(s0.25 s0.5 s0.75)][cx1 B(oo0) B(oo2) I(s0.25 s0.5 s0.75 s1.25 s1.5 s1.75)][cx2 B(cc1) I(s1.25 s1.5 s1.75)] # VSO configuration string
#VSOConfig                    : [ox0 B(cc1) I(s0.25 s0.5 s0.75)][cx1 B(oo0) B(oo2) I(s0.25 s0.5 s0.75 s1.25 s1.5 s1.75)][ox2 B(cc1) I(s1.25 s1.5 s1.75)] # VSO configuration string for FCO = 1
WVSO                          : 0                     # use of WVSO (Depth distortion metric with a weighted depth fidelity term)
VSOWeight                     : 0                     # weight of VSO ( in SAD case, cf. squared in SSE case )
VSDWeight                     : 0                     # weight of VSD ( in SAD case, cf. squared in SSE case )
DWeight                       : 0                     # weight of depth distortion itself ( in SAD case, cf. squared in SSE case )
UseEstimatedVSD               : 0                     # Model based VSD estimation instead of rendering based for some encoder decisions



### DO NOT ADD ANYTHING BELOW THIS LINE ###
### DO NOT DELETE THE EMPTY LINE BELOW ###
```

11 Annex 9: Inter-view prediction related parameters for HTM

In the configuration of HTM software, there are four parameters to assign the inter-view prediction structure for each view. They are DirectRefLayers_x, activeRefLayerPics, ilPredLayerIdc, refLayerPicPosIl_L0, refLayerPicPosIl_L1.

DirectRefLayers_x sepecifies the layerID of the reference layers of the layer x,

activeRefLayerPics specifies the number of reference pictures used for inter-layer prediction of the current picture. E.g. 0: means non inter-view reference picture 1: means one inter-view reference picture,

ilPredLayerIdc specifies the indices of the direct reference layers in DirectRefLayers_x.

refLayerPicPosIl_L0 and **refLayerPicPosIl_L1** specify the positions to which the inter-layer reference pictures are inserted to list 0 and 1, respectively. When the number is negative the position is related to the end of the list. E.g. 0 means first position of the list , -2 means the last but one position of the list.

Example:

If we have the configurations like follows:

```
.........
DirectRefLayers_7 : 1 3 5      # Layers 1, 3 and 5 are direct reference layers of layer 7
.......
            activeRefLayerPics ilPredLayerIdc refLayerPicPosIl_L0  refLayerPicPosIl_L1
Frame2_l7:    2                      0 2            0 -1                  -1 1
```

1 3 5 : specifies the pictures of layer7 can refer the pictures of layer1, layer3 and layer5.

Inter view prediction structure of the picture 2 in a GOP of layer7:

2 : specifies the frame has two inter-view reference pictures.
0 2 : specifies the two inter-view reference pictures are respectively from layer1 and layer5.
0 -1 : specifies the reference picture from layer1 is inserted to the first position of the list0 and the reference picture from layer5 is inserted to the last position of the list0.
1 2 : specifies the reference picture from layer1 is inserted to the second position of the list0 and the reference picture from layer5 is inserted to the third position of the list0.

12 Annex 10: SMV and FN software user guidelines

This annex describes how the user can run the script files for coding and/or view synthesis in SMV and FN applications, along following steps:

- Creating depth maps with DERS
- Generating 3D-HEVC bitstreams from the input views and (optional) depth maps
- Decoding the 3D-HEVC bitstreams
- Synthesizing new virtual viewpoints that were not present in the bitstream (e.g. for FN) with VSRS

The scripts are written for the test sequences and camera parameters that are used in the CfE for FTV (SMV and FN) [CfE FTV document] issued at the Warsaw 112th MPEG meeting.

[Here comes a workflow diagram with numbers]
[Here comes the scripts usage and parameters for each step in the above workflow]

References

[N15095] Gauthier Lafruit, Krzysztof Wegner, Masayuki Tanimoto, "Draft Call for Evidence on FTV," ISO/IEC JTC1/SC29/WG11 MPEG2015/N15095, Geneva, Switzerland, February 2015.

[CV1] "Computer Vision course: CISC 4/689," University of Louisville.

[CV2] Robert Collins, "Computer Vision I: CSE486," Penn State University

[3DMaths2011] John Flynt, Eric Lengyel, "Mathematics for 3D Game programming and Computer Graphics," Course Technology, 2011.

[W9595] "Call for Contributions on 3D Video Test Material (Update)," ISO/IEC JTC1/SC29/WG11 MPEG 2008/N9595, Antalya, Turkey, January 2008.

[DERSManual] Krzysztof Wegner, Olgierd Stankiewicz, „DERS Software Manual", ISO/IEC JTC1/SC29/WG11 MPEG2014/M34302 July 2014, Sapporo, Japan.

[VSRS] Krzysztof Wegner, Olgierd Stankiewicz, Masayuki Tanimoto, Marek Domanski, „Enhanced View Synthesis Reference Software (VSRS) for

Free-viewpoint Television", ISO/IEC JTC1/SC29/WG11 MPEG2013/M31520 October 2013, Geneva, Switzerland.

[DERS] Krzysztof Wegner, Olgierd Stankiewicz, Masayuki Tanimoto, Marek Domanski, „Enhanced Depth Estimation Reference Software (DERS) for Free-viewpoint Television", ISO/IEC JTC1/SC29/WG11 MPEG2013/M31518 October 2013, Geneva, Switzerland.

[CfE FTV document]