INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 4
MPEG VIDEO CODING

ISO/IEC JTC 1/SC 29/WG 4 **m 68843**
July 2024, Sapporo

Title: A new approach to geometry features signaling

Source: **PUT: Jakub Stankowski, Błażej Szydełko, Adrian Dziembowski**
ETRI: Jun Young Jeong, Gwangsoon Lee

1 Abstract

This document presents a new approach to geometry features and a new type of Extended Geometry Assistance SEI – quadtree geometry features (QTGF). With the new assistance type, new geometry block encoding modes are proposed: rectangular arbitrary, wedgelet, bilateral, and gradient. The DRFC (Depth Range Features Codec) software has been created from scratch, which can produce bitstream compliant with current EGA SEI syntax (block-based geometry features) and provides implementation of the proposal. The recommendations are: (1) to adopt the proposal, (2) to include the DRFC software in the IVDE repository.

2 Geometry features codec software – DRFC

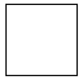
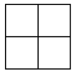
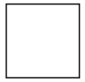
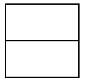
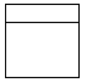
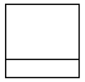
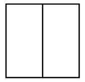
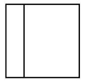
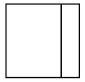
We have developed a codec for EGA SEI that is capable of encoding and decoding geometry features. The main feature of the DRFC encoder is the ability to control the selection of modes by adjusting the Lambda parameter, as in classic video encoders.

The manual of DRFC software will be submitted later.

3 New assistance type: QTGF – Quadtree geometry features

Our proposal uses a quadtree structure to represent geometry features. In contrast to BBGF, only square blocks can be recursively split into four smaller blocks. This simplifies encoding and decoding processes, and enhances context derivation from neighboring blocks. This also prevents splitting excessively thin blocks and leads to maintaining uniform sizes for better prediction and compression.

Each leaf node in the quadtree is encoded using several predefined modes, including modes derived from BBGF and adding five new ones:

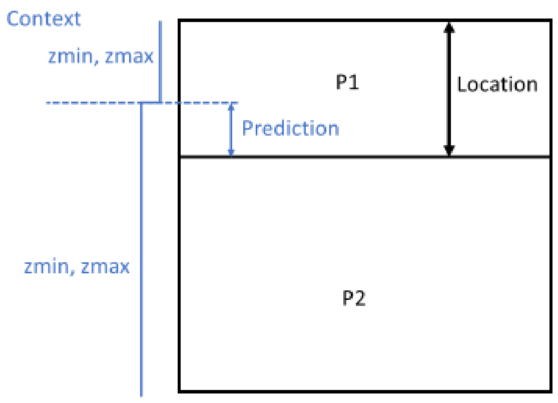
Name	Skip	Split (NxN)	Uniform	Wedge	Bilateral	Gradient	Part 2NxN	Part 2NxnU	Part 2NxnD	Part 2NxA	Part Nx2N	Part nLx2N	Part nRx2N	Part Ax2N
														
qtf_skip_flag (inter only)	1	0	0	0	0	0	0	0	0	0	0	0	0	0
qtf_quadsplit_flag (CanSplit)		1	0	0	0	0	0	0	0	0	0	0	0	0
qtf_partitioning_flag (CanUseParts)			0	0	0	0	1	1	1	1	1	1	1	1
qtf_uniform_flag (EnableWedge || EnableBilateral)			1	0	0	0								
qtf_pattern_flag (EnableWedge && EnableBilateral)				1	1	0								
qtf_bilateral_flag				0	1									
qtf_part_direction_flag							0	0	0	0	1	1	1	1
qtf_part_symmetric_flag (CanUseAsymmetric)							1	0	0	0	1	0	0	0
qtf_part_arbitrary_flag (CanUseArbitrary)								0	0	1		0	0	1
qtf_first_block_bigger (CanUseAsymmetric)								0	1			0	1	

3.1 Rectangular arbitrary splits

In this mode block can be split into two rectangular parts in an arbitrary ratio (vertically or horizontally):

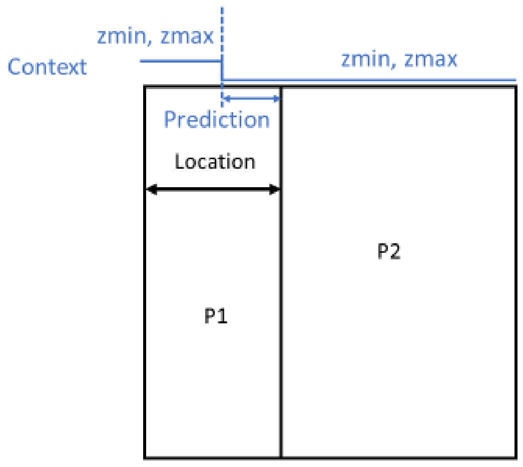
3.1.1 Horizontal arbitrary split mode – 2NxA

The position of the splitting line can be signaled by a prediction from edge the left block, or sent as it is with reference to the top line of the block.



3.1.2 Vertical arbitrary split mode – Ax2N

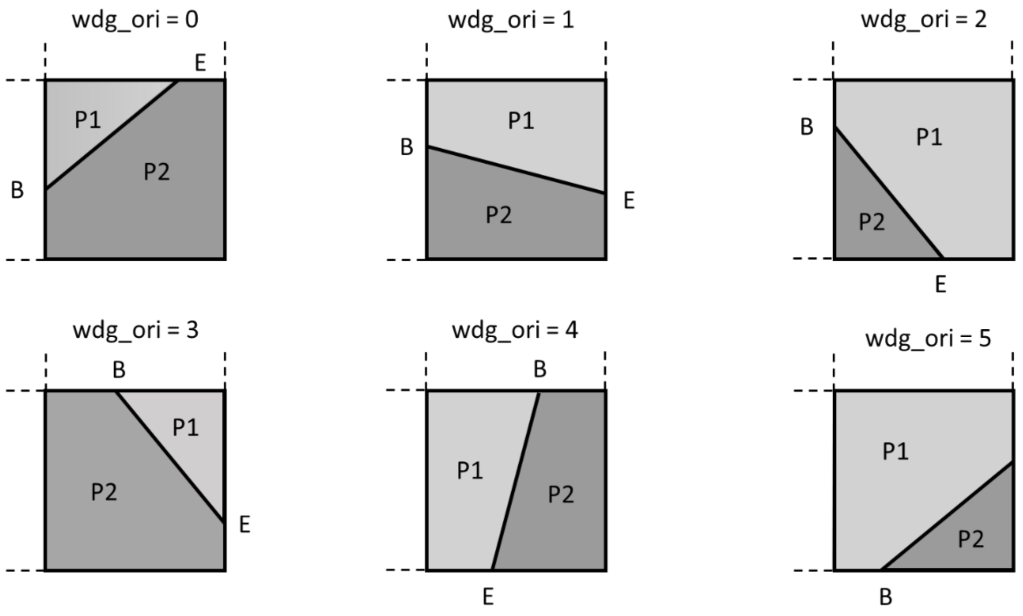
The position of the splitting line can be signaled by a prediction from edge in the top block, or sent as it is with reference to the left column of the block.



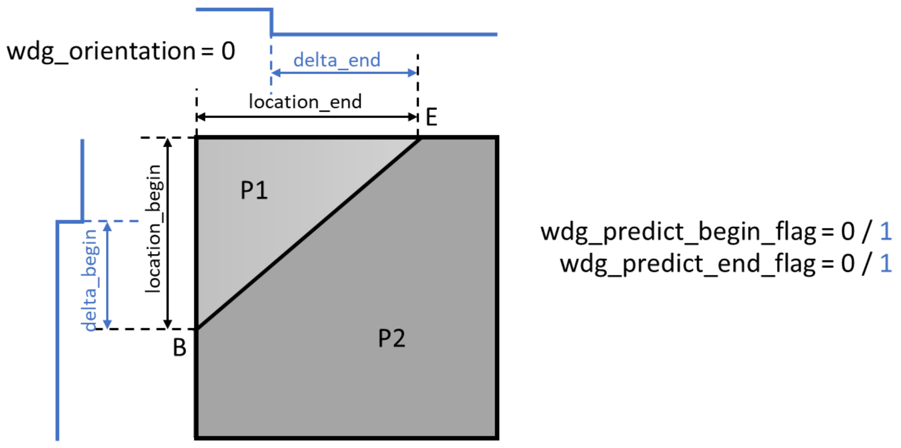
3.2 Non-uniform modes

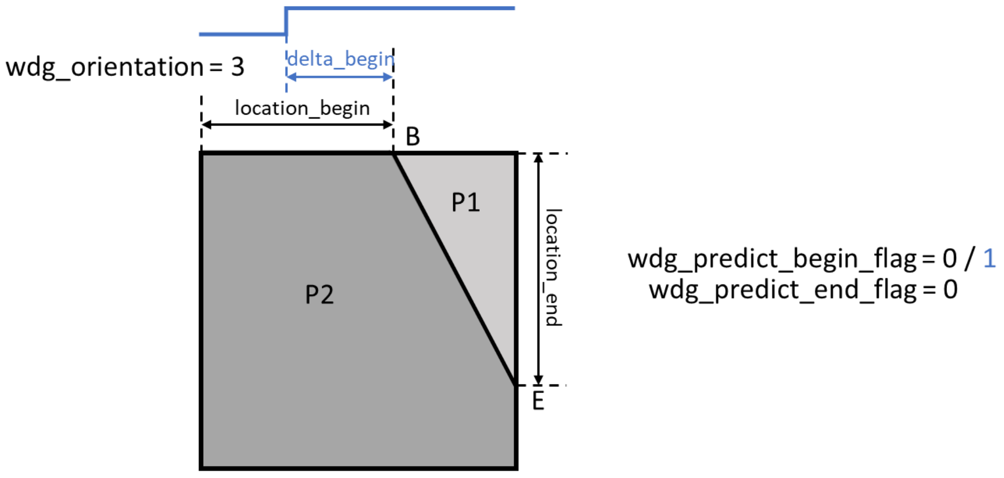
3.2.1 Wedgelet split mode

In the wedgelet mode, the block is divided by line along the edge of the depth. The orientation and coordinates of the *B* and *E* points are encoded (prediction from left or above block is possible).



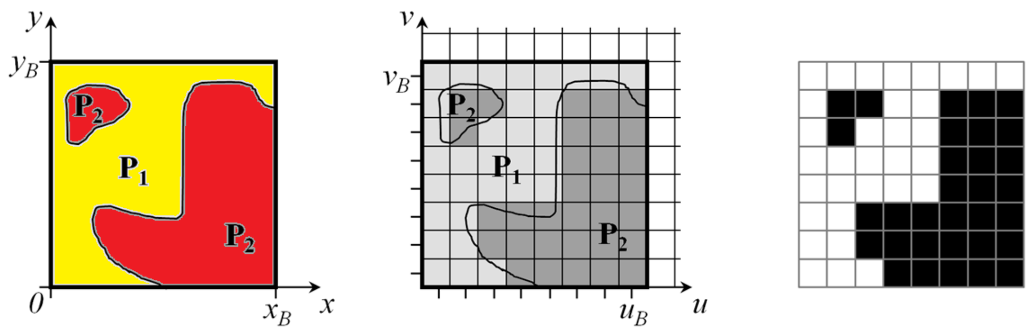
Values *B* and *E* may be predicted or signaled directly, depending on the orientation:





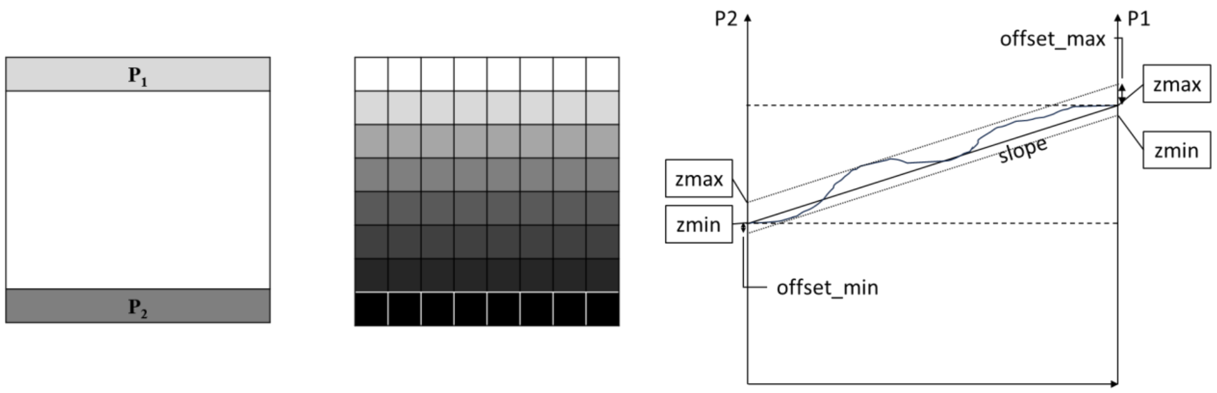
3.2.2 Bilateral split mode

In this mode, the partitioning is based on the edges in the texture. Splitting is done by thresholding the luminance component of the decoded texture block and assigning samples to the corresponding parts of P1 and P2. Only the value of the threshold is sent.



3.2.3 Gradient block mode

In this mode, the block is signaled as a gradient (horizontal or vertical). A depth range is transmitted for the two extremes of the block P1 and P2. The orientation, slope sign and offsets are also transmitted. In the decoding process, the area between parts P1 and P2 is filled in.



4 Syntax & semantics

F.1.1.1 Quadtree geometry features: initial block grid syntax

quadtree_geometry_features(v) {	
qtgf_depth_plane_height_minus1[v]	u(16)
qtgf_depth_plane_width_minus1[v]	u(16)
qtgf_depth_bit_depth_minus8[v]	u(3)
qtgf_quant_step_minus1[v]	ue(v)
qtgf_overwrite_unit_settings[v]	u(1)
if(qtgf_overwrite_unit_settings) {	
qtgf_log2_root_size_minus_def[v]	se(v)
qtgf_max_quadsplit_depth_minus_def[v]	se(v)
qtgf_log2_min_unit_size_minus_def[v]	se(v)
qtgf_log2_min_symmetric_minus_def[v]	se(v)
qtgf_log2_min_asymmetric_minus_def[v]	se(v)
qtgf_log2_min_arbitrary_minus_def[v]	se(v)
qtgf_log2_min_wedge_minus_def[v]	se(v)
qtgf_log2_min_bilateral_minus_def[v]	se(v)
qtgf_log2_min_gradient_minus_def[v]	se(v)
}	
RootSize = 1 << (qtgf_log2_root_size_minus_def[v] + *PredefRootSize*)	
MaxQuadsplitDepth = qtgf_max_quadsplit_depth_minus_def[v] + *PredefMaxQuadsplitDepth*)	
MinUnitSize = 1 << (qtgf_log2_min_unit_size_minus_def[v] + *PredefMinUnitSize*)	
MinSymmetric = 1 << (qtgf_log2_min_symmetric_minus_def[v] + *PredefMinSymmetric*)	
MinAsymmetric = 1 << (qtgf_log2_min_asymmetric_minus_def[v] + *PredefMinAsymmetric*)	
MinArbitrary = 1 << (qtgf_log2_min_arbitrary_minus_def[v] + *PredefMinArbitrary*)	
MinWedge = 1 << (qtgf_log2_min_wedge_minus_def[v] + *PredefMinWedge*)	
MinBilateral = 1 << (qtgf_log2_min_bilateral_minus_def[v] + *PredefMinBilateral*)	
MinGradient = 1 << (qtgf_log2_min_gradient_minus_def[v] + *PredefMinGradient*)	
if(*MinBilateral* <= *RootSize*) {	
qtgf_log2_bilateral_thr_step_minus1[v]	ue(v)
qtgf_bilateral_bit_depth_minus8[v]	u(3)
}	
qtgf_is_frame_inter[v]	u(1)
qtgf_use_bac[v]	u(1)
for(h = 0; h <= qtgf_depth_plane_height_minus1[v]; h += *RootSize*) {	
for(w = 0; w <= (qtgf_depth_plane_width_minus1[v]; w += *RootSize*) {	
quadtree_function(v, h, w, 0)	
}	
}	
}	

qtgf_depth_plane_width_minus1[v] plus 1 and **qtgf_depth_plane_height_minus1**[v] plus 1 specify the horizontal and vertical resolutions, respectively, of the camera projection planes for view v, expressed in coded luma samples, for which geometry assistance parameters are signaled.

qtgf_depth_bit_depth_minus8[v] plus 8 specifies the bit depth of the samples of the depth for view v.

qtgf_quant_step_minus1[v] plus 1 specifies the quantization step for the suggested geometry range boundaries for view v.

qtgf_overwrite_unit_settings[v] equal to 1 indicates that the bitstream contains syntax elements changing predefined values of QTGF limits for view v.

qtgf_log2_root_size_minus_def[v] plus *PredefinedRootSize* (specified in Table P) specifies the initial size of a block in view v. When not present, the value of qtgf_log2_root_size_minus_def[v] is inferred to be equal to 0.

qtgf_max_quadsplit_depth_minus_def[v] plus *PredefinedMaxQuadsplitDepth* (specified in Table P) specifies the maximum number of quadsplits of each block for view v. When not present, the value of qtgf_max_quadsplit_depth_minus_def[v] is inferred to be equal to 0.

qtgf_log2_min_unit_size_minus_def[v] plus *PredefinedMinUnitSize* (specified in Table P) specifies the minimum size of a block for view v. When not present, the value of qtgf_log2_min_unit_size_minus_def[v] is inferred to be equal to 0.

qtgf_log2_min_symmetric_minus_def[v] plus *PredefinedMinSymmetric* (specified in Table P) specifies the minimum size of a block in view v, for which the symmetrical partitioning may be signaled. When not present, the value of qtgf_log2_min_symmetric_minus_def[v] is inferred to be equal to 0.

qtgf_log2_min_asymmetric_minus_def[v] plus *PredefinedMinAsymmetric* (specified in Table P) specifies the minimum size of a block in view v, for which the asymmetrical partitioning may be signaled. When not present, the value of qtgf_log2_min_asymmetric_minus_def[v] is inferred to be equal to 0.

qtgf_log2_min_arbitrary_minus_def[v] plus *PredefinedMinArbitrary* (specified in Table P) specifies the minimum size of a block in view v, for which the arbitrary partitioning may be signaled. When not present, the value of qtgf_log2_min_arbitrary_minus_def[v] is inferred to be equal to 0.

qtgf_log2_min_wedge_minus_def[v] plus *PredefinedMinWedge* (specified in Table P) specifies the minimum size of a block in view v, for which the wedgelet partitioning may be signaled. When not present, the value of qtgf_log2_min_wedge_minus_def[v] is inferred to be equal to 0.

qtgf_log2_min_bilateral_minus_def[v] plus *PredefinedMinBilateral* (specified in Table P) specifies the minimum size of a block in view v, for which the bilateral partitioning may be signaled. When not present, the value of qtgf_log2_min_bilateral_minus_def[v] is inferred to be equal to 0.

qtgf_log2_min_gradient_minus_def[v] plus *PredefinedMinGradient* (specified in Table P) specifies the minimum size of a block in view v, for which the gradient-based features may be signaled. When not present, the value of qtgf_log2_min_gradient_minus_def[v] is inferred to be equal to 0.

qtgf_log2_bilateral_thr_step_minus1[v] specifies the quantization step for bilateral selection threshold for view v. *BilateralThresholdStep* = 1 << (qtgf_log2_bilateral_thr_step_minus1[v] + 1). When not present, the value of qtgf_log2_bilateral_thr_step_minus1[v] is inferred to be equal to 0.

qtgf_bilateral_bit_depth_minus8[v] plus 8 specifies the bit depth of the samples of the texture for view v. When not present, the value of qtgf_bilateral_bit_depth_minus8[v] is inferred to be equal to 0.

qtgf_is_frame_inter[v] equal to 1 indicates that the current frame for view v allows for inter-coded units. qtgf_is_frame_inter[v] equal to 0 indicates that the current frame is intra-coded.

qtgf_use_bac[v] equal to 1 indicates that the quadtree function is encoded using a binary arithmetic coder. Otherwise, it is encoded using UVLC coder.

NOTE: Only the UVLC coding is supported yet.

Table P – Predefined QTGF limits

Limit name	**Predefined value**
PredefinedRootSize	7
PredefinedMaxQuadsplitDepth	4
PredefinedMinUnitSize	3
PredefinedMinSymmetric	3
PredefinedMinAsymmetric	3
PredefinedMinArbitrary	3
PredefinedMinWedge	4
PredefinedMinBilateral	6
PredefinedMinGradient	4

F.1.1.2 Quadtree geometry features: quadtree function

quadtree_function(v, i, j, k) {	
if(qtgf_is_frame_inter[v]) {	
qtf_skip_flag[v][i][j][k]	u(1)
}	
if(!qtf_skip_flag[v][i][j][k]) {	
CurrentBlockSize = *RootSize* >> k	
if(k <= *MaxQuadsplitDepth* && *CurrentBlockSize* > *MinUnitSize*) {	
qtf_quadsplit_flag[v][i][j][k]	u(1)
}	
if(qtf_quadsplit_flag[v][i][j][k]) {	
quadtree_function(v, i, j, k + 1)	
quadtree_function(v, i + *CurrentSize* >> 1, j, k + 1)	
quadtree_function(v, i, j + *CurrentSize* >> 1, k + 1)	
quadtree_function(v, i + *CurrentSize* >> 1, j + *CurrentSize* >> 1, k + 1)	

}else{	
if(*CurrentBlockSize* >= min(*MinSymmetric*, *MinAsymmetric*, *MinArbitrary*)) {	
qtf_partitioning_flag[v][i][j][k]	u(1)
}	
if(!qtf_partitioning_flag[v][i][j][k]) {	
if(*CurrentBlockSize* >= min(*MinWedge*, *MinBilateral*, *MinGradient*)) {	
qtf_uniform_flag[v][i][j][k]	u(1)
}	
if(qtf_uniform_flag[v][i][j][k]) {	
delta_z_function(v, i, j, 0)	
}else{	
if(*CurrentBlockSize* >= max(min(*MinWedge*, *MinBilateral*), *MinGradient*)) {	
qtf_pattern_flag[v][i][j][k]	u(1)
}	
if(qtf_pattern_flag[v][i][j][k]) {	
if(*CurrentBlockSize* >= max(*MinWedge*, *MinBilateral*)) {	
qtf_bilateral_flag[v][i][j][k]	u(1)
}	
if(qtf_bilateral_flag[v][i][j][k]) {	
qtf_bil_threshold[v][i][j][k]	u(v)
delta_z_function(v, i, j, 0)	
delta_z_function(v, i, j, 1)	
}else{	
qtf_wdg_orientation[v][i][j][k]	u(3)
if(qtf_wdg_orientation[v][i][j][k] < 5) {	
qtf_wdg_predict_begin_flag[v][i][j][k]	u(1)
}	
if(qtf_wdg_predict_begin_flag[v][i][j][k]) {	
qtf_wdg_delta_begin[v][i][j][k]	se(v)
}else{	
qtf_wdg_location_begin[v][i][j][k]	u(v)
}	
if(qtf_wdg_orientation[v][i][j][k] == 0) {	
qtf_wdg_predict_end_flag[v][i][j][k]	u(1)
}	
if(qtf_wdg_predict_end_flag[v][i][j][k]) {	
qtf_wdg_delta_end[v][i][j][k]	se(v)
}else{	
qtf_wdg_location_end[v][i][j][k]	u(v)
}	
delta_z_function(v, i, j, 0)	
delta_z_function(v, i, j, 1)	
}	

}else{	
qtf_grd_dir_flag[v][i][j][k]	u(1)
qtf_grd_slope_sign[v][i][j][k]	u(1)
qtf_grd_offset_min[v][i][j][k]	ue(v)
qtf_grd_offset_max[v][i][j][k]	ue(v)
delta_z_function(v, i, j, 0)	
}	
}	
}else{	
qtf_part_direction_flag[v][i][j][k]	u(1)
if(*CurrentBlockSize* >= max(min(*MinAsymmetric*, *MinArbitrary*), *MinSymmetric*)){	
qtf_part_symmetric_flag[v][i][j][k]	u(1)
}	
if(qtf_part_symmetric_flag[v][i][j][k]) {	
SplitOffset = *CurrentSize* >> 1	
}else{	
if(*CurrentBlockSize* >= max(*MinAsymmetric*, *MinArbitrary*)) {	
qtf_part_arbitrary_flag[v][i][j][k]	u(1)
}	
if(qtf_part_arbitrary_flag[v][i][j][k]) {	
qtf_arb_predict_flag[v][i][j][k]	u(1)
if(qtf_arb_predict_flag[v][i][j][k]) {	
qtf_arb_delta[v][i][j][k]	se(v)
SplitOffset = *PredictedLocation* + qtf_arb_delta[v][i][j][k]	
}else{	
qtf_arb_location[v][i][j][k]	u(v)
SplitOffset = qtf_arb_location[v][i][j][k]	
}	
}else{	
qtf_first_block_bigger[v][i][j][k]	u(1)
if(qtf_first_block_bigger[v][i][j][k]) {	
SplitOffset = (*CurrentSize* >> 1) + (*CurrentSize* >> 2)	
}else{	
SplitOffset = (*CurrentSize* >> 1) - (*CurrentSize* >> 2)	
}	
}	
}	
if(qtf_part_direction_flag) {	
delta_z_function(v, i, j, 0)	
delta_z_function(v, i + *SplitOffset*, j, 1)	
}else{	
delta_z_function(v, i, j, 0)	
delta_z_function(v, i, j + *SplitOffset*, 1)	

}	
}	
}	
}	
}	

qtf_skip_flag[v][i][j][k] equal to 1 indicates that no other syntax elements are present in the bitstream for the block with indices i, j, k of the view with index v, and it suggests that the geometry information in this block has not changed since the previous frame in display order.

qtf_quadsplit_flag[v][i][j][k] equal to 1 indicates that the block with indices i, j, k of the view with index v is split into four square subblocks. qtf_quadsplit_flag[v][i][j][k] equal to 0 indicates that the block with indices i, j, k of the view with index v is not split into four square subblocks.

qtf_partitioning_flag[v][i][j][k] equal to 1 indicates that the block with indices i, j, k of the view with index v is split into two rectangular subblocks. qtf_partitioning_flag[v][i][j][k] equal to 0 indicates that the block with indices i, j, k of the view with index v is not split into two rectangular subblocks.

qtf_uniform_flag[v][i][j][k] equal to 1 indicates that the block with indices i, j, k of the view with index v is not split into subblocks. qtf_uniform_flag[v][i][j][k] equal to 0 indicates that the block with indices i, j, k of the view with index v is split into subblocks.

qtf_pattern_flag[v][i][j][k] equal to 1 indicates that the block with indices i, j, k of the view with index v is split into two subblocks based on decoded texture analysis or using wedgelet splitting. qtf_pattern_flag[v][i][j][k] equal to 0 indicates that the block with indices i, j, k of the view with index v is neither split based on decoded texture analysis nor using wedgelet splitting.

qtf_bilateral_flag[v][i][j][k] equal to 1 indicates that the block with indices i, j, k of the view with index v is split into two subblocks based on decoded texture analysis. qtf_bilateral_flag[v][i][j][k] equal to 0 indicates that the block with indices i, j, k of the view with index v is not split based on decoded texture analysis.

qtf_bil_threshold[v][i][j][k] specifies the luma threshold within decoded texture, used for calculation of: *BilateralThreshold* = qtf_bil_threshold[v][i][j][k] * *BilateralThresholdStep*

qtf_wdg_orientation[v][i][j][k] specifies the orientation of the wedge of the block with indices i, j, k of the view with index v, as specified in Table W.

Table W – Wedge orientations

qtf_wdg_orientation value	Begin point position	End point position
0	Leftmost column	Top row
1	Leftmost column	Rightmost column
2	Leftmost column	Bottom row
3	Top row	Rightmost column
4	Top row	Bottom row
5	Bottom row	Rightmost column

qtf_wdg_predict_begin_flag[v][i][j][k] equal to 1 indicates that the location of the wedgelet begin point of the block with indices i, j, k of the view with index v is predicted from left or top block. Otherwise, the location is signaled directly.

qtf_wdg_delta_begin[v][i][j][k] specifies the location of the wedgelet begin point of the block with indices i, j, k of the view with index v, signaled as the offset from the value predicted from the left block if qtf_wdg_orientation[v][i][j][k] < 3 and the offset from the value predicted from the top block otherwise.

qtf_wdg_location_begin[v][i][j][k] specifies the location of the wedgelet begin point of the block with indices i, j, k of the view with index v, signaled as the offset from the top row of the block if qtf_wdg_orientation[v][i][j][k] < 3 and the offset from the leftmost column of the block otherwise.

qtf_wdg_predict_end_flag[v][i][j][k] equal to 1 indicates that the location of the wedgelet end point of the block with indices i, j, k of the view with index v is predicted from left or top block. Otherwise, the location is signaled directly.

qtf_wdg_delta_end[v][i][j][k] specifies the location of the wedgelet end point of the block with indices i, j, k of the view with index v, signaled as the offset from the value predicted from the top block.

qtf_wdg_location_end[v][i][j][k] specifies the location of the wedgelet end point of the block with indices i, j, k of the view with index v, signaled as the offset from the top row of the block if qtf_wdg_orientation[v][i][j][k] % 2 == 1 and the offset from the leftmost column of the block otherwise.

qtf_grd_dir_flag[v][i][j][k] equal to 0 indicates that the block with indices i, j, k of the view with index v contains horizontal gradient of depth, qtf_grd_dir_flag[v][i][j][k] equal to 1 indicates that the block with indices i, j, k of the view with index v contains vertical gradient of depth.

qtf_grd_slope_sign[v][i][j][k] equal to 1 indicates that the depth values in the block with indices i, j, k of the view with index v decrease from left to right if qtf_grd_dir_flag[v][i][j][k] == 0 or from top to bottom if qtf_grd_dir_flag[v][i][j][k] == 1. qtf_grd_slope_sign[v][i][j][k] equal to 0 indicates that the depth values in the block with indices i, j, k of the view with index v increase from left to right if qtf_grd_dir_flag[v][i][j][k] == 0 or from top to bottom if qtf_grd_dir_flag[v][i][j][k] == 1.

qtf_grd_offset_min[v][i][j][k] specifies the offset between the slope and the most outlier pixel below the slope for the block with indices i, j, k of the view with index v.

qtf_grd_offset_max[v][i][j][k] specifies the offset between the slope and the most outlier pixel above the slope for the block with indices i, j, k of the view with index v.

qtf_part_direction_flag[v][i][j][k] equal to 0 indicates that the block with indices i, j, k of the view with index v is split horizontally. qtf_part_direction_flag[v][i][j][k] equal to 1 indicates that the block with indices i, j, k of the view with index v is split vertically.

qtf_part_symmetric_flag[v][i][j][k] equal to 0 indicates that the area of the two sublocks of the block with indices i, j, k of the view with index v differ. qtf_part_symmetric_flag[v][i][j][k] equal to 1 indicates that the area of the two sublocks of the block with indices i, j, k of the view with index v is equal.

qtf_part_arbitrary_flag[v][i][j][k] equal to 0 indicates that the ratio between areas of the two sublocks of the block with indices i, j, k of the view with index v equals to 3:1 or 1:3. qtf_part_arbitrary_flag[v][i][j][k] equal to 1 indicates that the ratio between areas of the two sublocks of the block with indices i, j, k of the view with index v equals is not restricted.

qtf_arb_predict_flag[v][i][j][k] equal to 1 indicates that the location of the arbitrary split location of the block with indices i, j, k of the view with index v is predicted from left or top block. Otherwise, the location is signaled directly.

qtf_arb_delta[v][i][j][k] specifies the location of the arbitrary split of the block with indices i, j, k of the view with index v, signaled as the offset from the value predicted from the left block if qtf_part_direction_flag[v][i][j][k] == 0 and the offset from the value predicted from the top block otherwise.

qtf_arb_location[v][i][j][k] specifies the location of the arbitrary split of the block with indices i, j, k of the view with index v, signaled as the offset from the top row of the block if qtf_part_direction_flag[v][i][j][k] == 0 and the offset from the leftmost column of the block otherwise.

qtf_first_block_bigger[v][i][j][k] equal to 1 indicates that the first subblock of the block with indices i, j, k of the view with index v (left subblock if qtf_part_direction_flag[v][i][j][k] is equal to 0, and top subblock if qtf_part_direction_flag[v][i][j][k] is equal to 1) is three times bigger than the second subblock. qtf_first_block_bigger[v][i][j][k] equal to 0 indicates that the first subblock is three times smaller than the second subblock.

F.1.1.3 Quadtree geometry features: delta z function

delta_z_function(v, i, j, k) {	
if(i == 0 && j == 0) { /* none */	
LTMinFlag = 2	
LTMaxFlag = 2	
} else if(i == 0) { /* left */	
LTMinFlag = 0	
LTMaxFlag = 0	
} else if(j == 0) { /* above */	
LTMinFlag = 1	
LTMaxFlag = 1	
} else {	
dzf_ltmin_flag[v][i][j][k]	u(1)
dzf_ltmax_flag[v][i][j][k]	u(1)

LTMinFlag = dzf_ltmin_flag[v][i][j][k]	
LTMaxFlag = dzf_ltmax_flag[v][i][j][k]	
}	
dzf_any_nonzero_delta_flag[v][i][j][k]	u(1)
if(dzf_any_nonzero_delta_flag[v][i][j][k]) {	
dzf_same_minmax_delta_flag[v][i][j][k]	u(1)
if(dzf_same_minmax_delta_flag[v][i][j][k]) {	
dzf_z_delta_sign[v][i][j][k]	u(1)
dzf_z_delta_abs_minus1[v][i][j][k]	ue(v)
if(dzf_z_delta_sign[v][i][j][k]) {	
DeltaMin[k] = - dzf_z_delta_abs_minus1 - 1	
DeltaMax[k] = - dzf_z_delta_abs_minus1 - 1	
}else{	
DeltaMin[k] = dzf_z_delta_abs_minus1 + 1	
DeltaMax[k] = dzf_z_delta_abs_minus1 + 1	
}	
}else{	
dzf_c_zmin_delta[v][i][j][k]	se(v)
dzf_c_zmax_delta[v][i][j][k]	se(v)
DeltaMin[k] = dzf_c_zmin_delta	
DeltaMax[k] = dzf_c_zmax_delta	
}	
}else{	
DeltaMin[k] = 0	
DeltaMax[k] = 0	
}	
}	

dzf_ltmin_flag[v][i][j][k] equal to 0 indicates that the prediction of the minimum geometry of the block with indices i, j, k of the view with index v is to be taken from the left block, otherwise from the top block.

dzf_ltmax_flag[v][i][j][k] equal to 0 indicates that the prediction of the maximum geometry of the block with indices i, j, k of the view with index v is to be taken from the left block, otherwise from the top block.

dzf_any_nonzero_delta_flag[v][i][j][k] equal to 0 indicates that the there is no remainder to be added to the prediction for both minimum and maximum.

dzf_same_minmax_delta_flag[v][i][j][k] equal to 1 indicates that the remainder to be added to the prediction is the same for both minimum and maximum.

dzf_z_delta_sign[v][i][j][k] equal to 1 indicates that the sign the remainder to be added to the prediction is negative. dzf_z_delta_sign[v][i][j][k] equal to 0 indicates that the sign the remainder to be added to the prediction is positive.

dzf_z_delta_abs_minus1[v][i][j][k] plus 1 specifies the absolute remainder to be added/subtracted to the prediction to obtain the maximum/minimum geometry value suggested for the block with coordinates i, j, k of the view with index v.

dzf_c_zmin_delta[v][i][j][k] specifies the remainder to be added to the prediction to obtain the minimum geometry value suggested for the block with with coordinates i, j, k of the view with index v.

dzf_c_zmax_delta[v][i][j][k] specifies the remainder to be added to the prediction to obtain the maximum geometry value suggested for the block with with coordinates i, j, k of the view with index v.

Decoding process:

ZMin/ZMax:

Variables ZMinLeft and ZMaxLeft are set to the minimum and maximum geometry range of the left block, respectively, and if available.

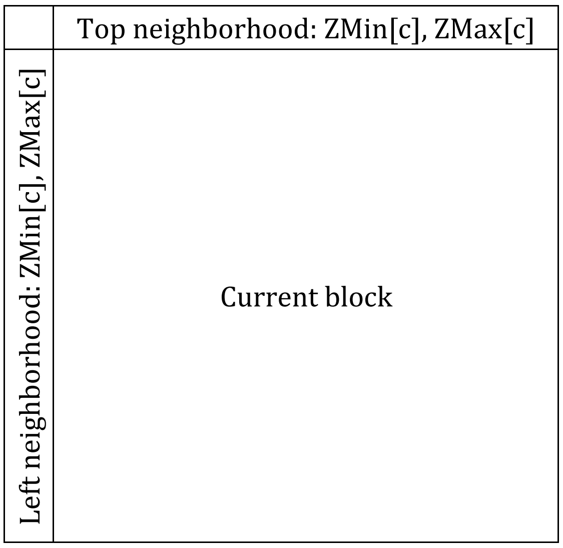
Variables ZMinTop and ZMaxTop are set to the minimum and maximum geometry range of the top block, respectively, and if available.

The suggested minimum geometry range ZMin and maximum geometry range ZMax of the block with coordinates i, j, k of the view with index v are derived by the following formulae:

QuantStep = qtgf_quant_step_minus1[v] + 1
ZMin[k] = (LTMinFlag == 2 ? 0 : LTMinFlag == 1 ? ZMinTop : ZMinLeft) + QuantStep * DeltaMin[k]
ZMax[k] = (LTMaxFlag == 2 ? 0 : LTMaxFlag == 1 ? ZMaxTop : ZMaxLeft) + QuantStep * DeltaMax[k]

PredictedLocation:

Variables ZMin[c] and ZMax[c] are set to the minimum and maximum geometry range of the left neighborhood or the top neighborhood, respectively, and if available.


Top neighborhood: ZMin[c], ZMax[c]
Left neighborhood: ZMin[c], ZMax[c]
Current block

```
PredictedLocation  = -1;
MaxDiff = 2;
for(c=1; c< CurrentSize; c++) {
  DiffMin = ZMin [c] - ZMin [c - 1];
  DiffMax = ZMax [c] - ZMax [c - 1];
  AbsDiff = abs(DiffMin) + abs(DiffMax);
  if(AbsDiff > MaxDiff) {
    MaxDiff = AbsDiff
    PredictedLocation  = c;
  }
}
if(PredictedLocation  == -1) {
   PredictedLocation = CurrentSize >> 1;
}
```

Bilateral:

```
for(y = i; y < i + CurrentSize; y++) {
  for(x = j; x < j + CurrentSize; x++) {
    if(Luma[y][x] < BilateralThreshold) {
      ZminPlane[y][x] = Zmin[0];
      ZmaxPlane[y][x] = Zmax[0];
    } else {
      ZminPlane[y][x] = Zmin[1];
      ZmaxPlane[y][x] = Zmax[1];
    }
  }
}
```

Wedge:

```
for(y = i; y < i + CurrentSize; y++) {
  for(x = j; x < j + CurrentSize; x++) {
    if(aboveLine()) {
      ZminPlane[y][x] = Zmin[0];
      ZmaxPlane[y][x] = Zmax[0];
    } else {
      ZminPlane[y][x] = Zmin[1];
      ZmaxPlane[y][x] = Zmax[1];
    }
  }
}
```

aboveLine() returns true if the picture element is above the dividing line BE; or – if the BE line is vertical – it returns true if the picture element is at the left of the dividing line BE. Otherwise, the function aboveLine() returns false.

Gradient:

```
SlopePrecisionBits  = 8;
SlopePrecisionValue = 1<< SlopePrecisionBits;
SlopePrecisionCorr  = 1 << (SlopePrecisionBits - 1);

Range    = Zmax[0] - Zmin[0];
SlopeDir = qtf_grd_slope_sign ? -1 : 1;
Log2Size = log2(CurrentSize)
EstSlope   = (Range * (1 << 8) + (1 << (Log2Size - 1))) >> Log2Size;
Slope      = EstSlope * SlopeDir;

for(i = 0; i < CurrentSize; i++) {
   ReferenceGradient[i] = (i * Slope + SlopePrecisionCorr) / SlopePrecisionValue;
}

StartMin = Slope > 0 ? BlkDpthMin : BlkDpthMin - ReferenceGradient [CurrentSize - 1];
StartMax = Slope < 0 ? BlkDpthMax : BlkDpthMax - ReferenceGradient [CurrentSize - 1];

for(i = 0; i < CurrentSize; i++) {
   GrdMin[i]  = max(BlkDpthMin, StartMin + ReferenceGradient[i] - qtf_grd_offset_min);
   GrdMax[i]  = min(BlkDpthMax, StartMax + ReferenceGradient[i] + qtf_grd_offset_max);
}

for(y = i; y < i + CurrentSize; y++) {
   for(x = j; x < j + CurrentSize; x++) {
      if(qtf_grd_dir_flag) {
         ZminPlane[y][x] = GrdMin[x - j];
         ZmaxPlane[y][x] = GrdMax[x - j];
      } else {
         ZminPlane[y][x] = GrdMin[y - i];
         ZmaxPlane[y][x] = GrdMax[y - i];
      }
   }
}
```

5 Software features

The proposed codec DRFC (Depth-range features codec) reassembles typical video codec, including following features:

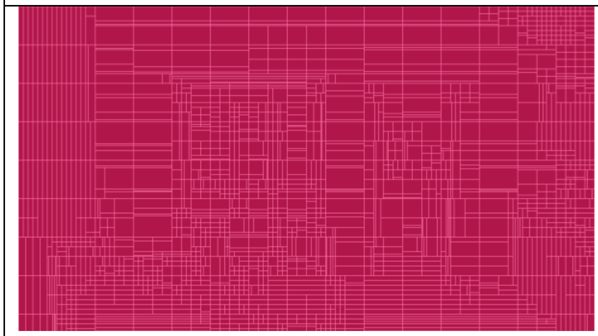
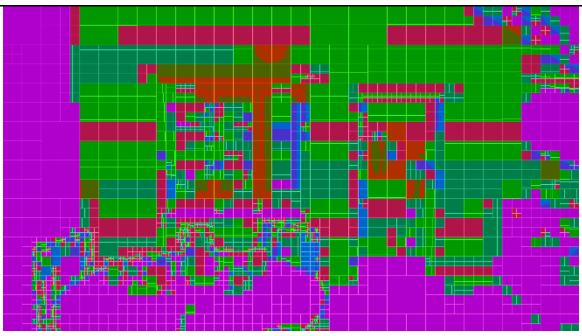
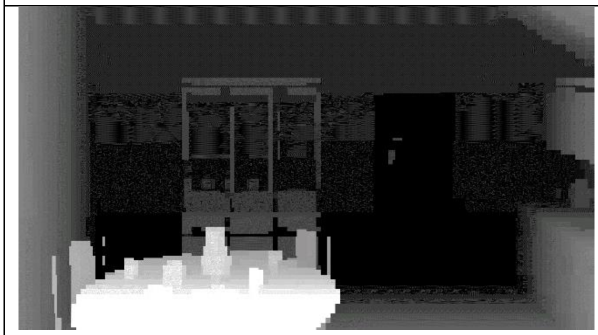
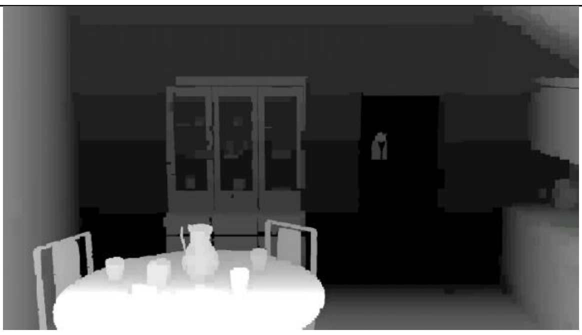
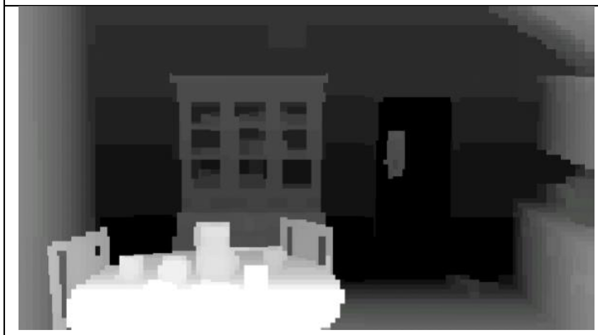
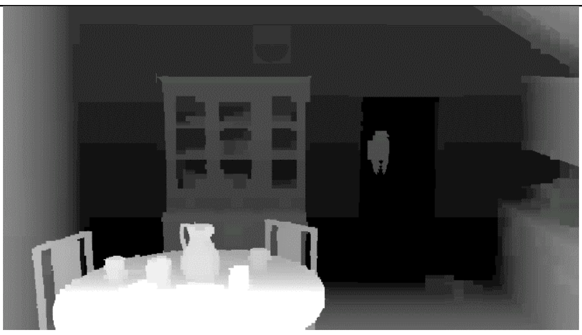
- Supports two types of geometry features:
    - EGA BBGF (block_based_geometry_features),
    - proposed QTGF (quadtree_geometry_features).
- Contains full Volume-Distortion Optimization (VD-opt)
    - equivalent of Rate-Distortion optimization (RD-opt) in regular video codec
    - Lagrange multiplier (λ) used to evaluate encoding mode cost
- Supports quantization of “z_delta” values.
- Generates (and decodes) bitstream consisting of SEI units containing EGA (extended_geometry_assistance) objects. Those SEI units can be easy multiplexed into bitstream containing other V3C units.
- Includes changes to EGA syntax proposed in [M68232].
- Generates detailed statistics related to “quality” (volume) and bitstream, including overhead introduced by encapsulation into SEI and NAL units.

- Includes extensive debugging/validation utilities, like:
    - encoder/decoder bitstream trace,
    - internal path to decode bitstream and validate decoded data (min, max, skip) against data reconstructed during encoding process,
    - validation of correctness of intermediate data (i.e. ZminPlane <= ZmaxPlane, left/top neighborhood Zmin[c] <= Zmax[c], etc.),
    - generation selected mode visualization.

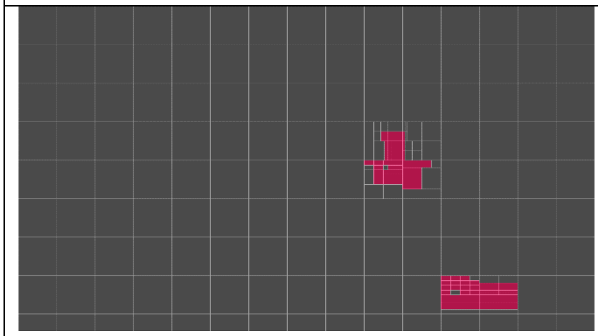
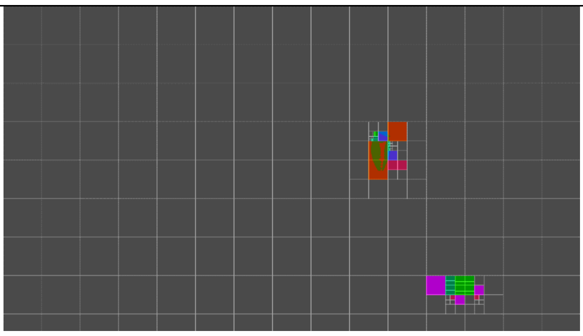
- Encoder and decoder are quite fast, and we still have few ideas how to make it even faster.
- DRFC uses the same underlying libraries as IV-PSNR [M68222] and QMIV [M68224] thus inheriting some optimizations and vectorized routines.
- Designed to allow parallel encoding and decoding at two independent levels (however, parallel implementation is not jet available):
    - fully parallelizable at view level (encoder and decoder),
    - mode search (encoder) at root block level.

6 Visualization

6.1 1st frame (selected modes, ZMin, ZMax)

BBGF	QTGF
	 🟥 uniform 🟦🟪 wedgelet 🟪 gradient 🟫🟩 bilateral 🟩 symmetric/asymmetric rectangular 🟩 arbitrary rectangular ⬜ skip
	
	

6.2 2nd frame (grey: skip)

BBGF	QTGF
	 🟥 uniform 🟦🟪 wedgelet 🟪 gradient 🟫🟩 bilateral 🟩 symmetric/asymmetric rectangular 🟩 arbitrary rectangular ⬜ skip

7 Experimental results

Experiments were conducted with compliance to CTC for G65 anchor for only mandatory content with two geometry features encoding scenarios:

1. Block-based geometry features (BBGF)
2. Quadtree geometry features (QTGF)

7.1 G65 anchor vs. BBGF

The experiment is still running. The results are expected before the MPEG meeting.

7.2 G65 anchor vs. QTGF

The experiment is still running. The results are expected before the MPEG meeting.

7.3 BBGF vs. QTGF

Seq		qp0	qp1	qp2	qp3	qp4	bbgf
J01	B	555.447	555.014	553.785	552.552	551.849	544.349
	V	0.119	0.120	0.120	0.121	0.122	0.138
W01	B	3218.135	3216.025	3218.820	3222.105	3221.899	3478.359
	V	22.041	22.043	22.050	22.058	22.104	23.092
D01	B	12828.285	12852.245	12859.871	12857.968	12860.246	22796.990
	V	112.089	111.892	111.881	111.852	111.906	124.631
D03	B	16013.582	16020.454	16022.762	16024.327	16029.974	17267.680
	V	184.705	184.623	184.605	184.596	184.554	179.328
L01	B	2778.356	2782.963	2783.344	2782.287	2784.672	5433.176
	V	23.756	23.630	23.612	23.619	23.647	29.186
L02	B	5876.218	5878.349	5878.666	5878.871	5878.695	8411.643
	V	36.583	36.557	36.555	36.558	36.541	35.846

Seq		qp0	qp1	qp2	qp3	qp4	bbgf
J01	B	102%	102%	102%	102%	101%	100%
	V	86%	87%	87%	87%	88%	100%
W01	B	93%	92%	93%	93%	93%	100%
	V	95%	95%	95%	96%	96%	100%
D01	B	56%	56%	56%	56%	56%	100%
	V	90%	90%	90%	90%	90%	100%
D03	B	93%	93%	93%	93%	93%	100%
	V	103%	103%	103%	103%	103%	100%
L01	B	51%	51%	51%	51%	51%	100%
	V	81%	81%	81%	81%	81%	100%
L02	B	70%	70%	70%	70%	70%	100%
	V	102%	102%	102%	102%	102%	100%
AVG	B	77%	77%	77%	77%	77%	100%
	V	93%	93%	93%	93%	93%	100%

B – bitrate [kbps]
V – volume [*10^9]:

$$V = \sum_{v=0}^{views} \sum_{h=0}^{H} \sum_{w=0}^{W} ZMax(v,h,w) - ZMin(v,h,w)$$

8 Recommendations

We recommend:

- adopting the proposal,
- including the DRFC software in the IVDE repository.

9 Acknowledgment

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00207, Immersive Media Research Laboratory).