

**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 69425

July 2024, Sapporo

Title: Quadtree geometry features in EGA SEI
Source: PUT: Jakub Stankowski, Błażej Szydełko, Adrian Dziembowski,
Dawid Mieloch, ETRI: Jun Young Jeong, Gwangsoon Lee

1 Abstract

This document presents an **update of the** new approach to geometry features and a new type of Extended Geometry Assistance SEI – quadtree geometry features (QTGF, m68843). With the new assistance type, new geometry block encoding modes are proposed: rectangular arbitrary, wedgelet, bilateral, gradient, **and signaling skip**. 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 perform a crosscheck**, (2) to adopt the proposal **into future MIV2 Amd or start a new CE**.

V2 of the document contains link to posetraces repository (the only change):

<https://1drv.ms/f/s!Ascw-MEitOZ9g4ZL1pKqKa0MIMDfgA?e=yDKknX>

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 **six** new ones:

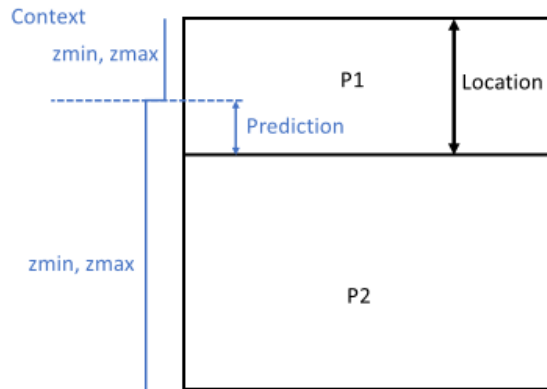
Name	Skip depth estimation	Skip feat. signalling	Split (NxN)	Uniform	Wedge	Bilateral	Gradient	Part 2NxN	Part 2NxN U	Part 2NxN D	Part 2NxN A	Part Nx2N	Part nLx2N	Part nRx2N	Part Ax2N
															
qtf_skip_flag (inter only)	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
qtf_exact_skip_flag (inter only)	1	0													
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):

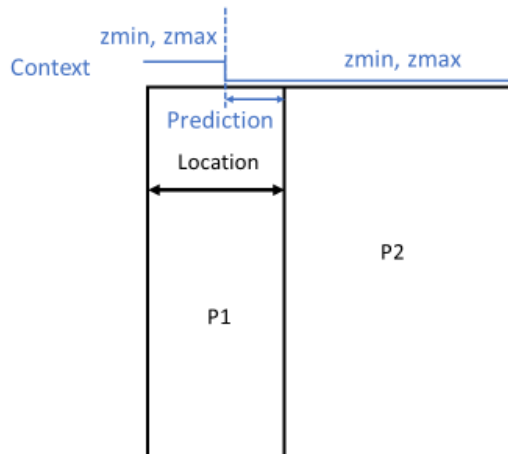
3.1.1 Horizontal arbitrary split mode – $2N \times A$

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 – $A \times 2N$

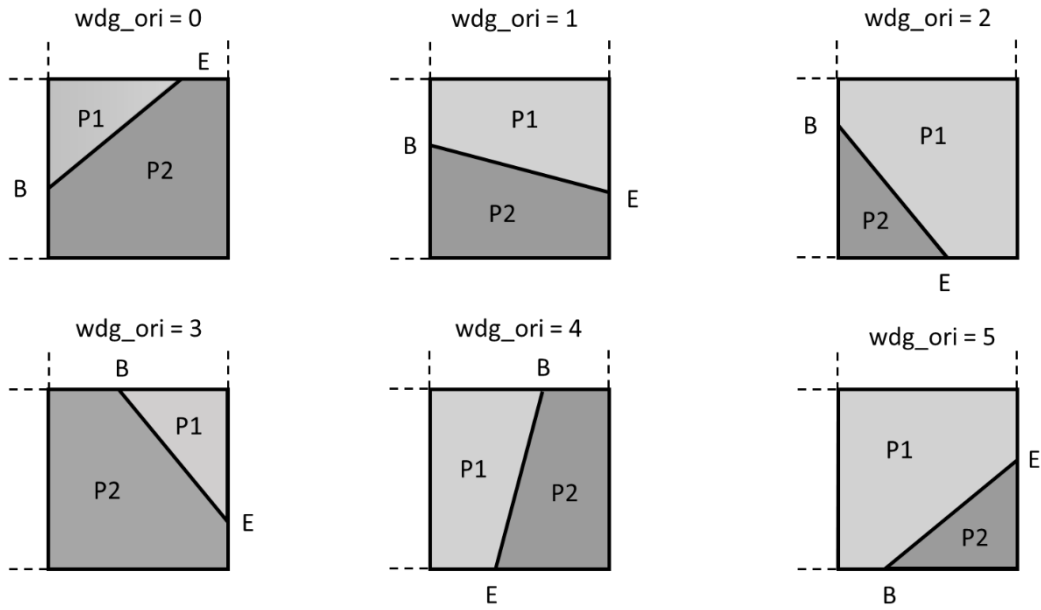
The position of the splitting line can be signaled by a prediction from edge in the top block, or sent as it is with reference to the left column of the block.



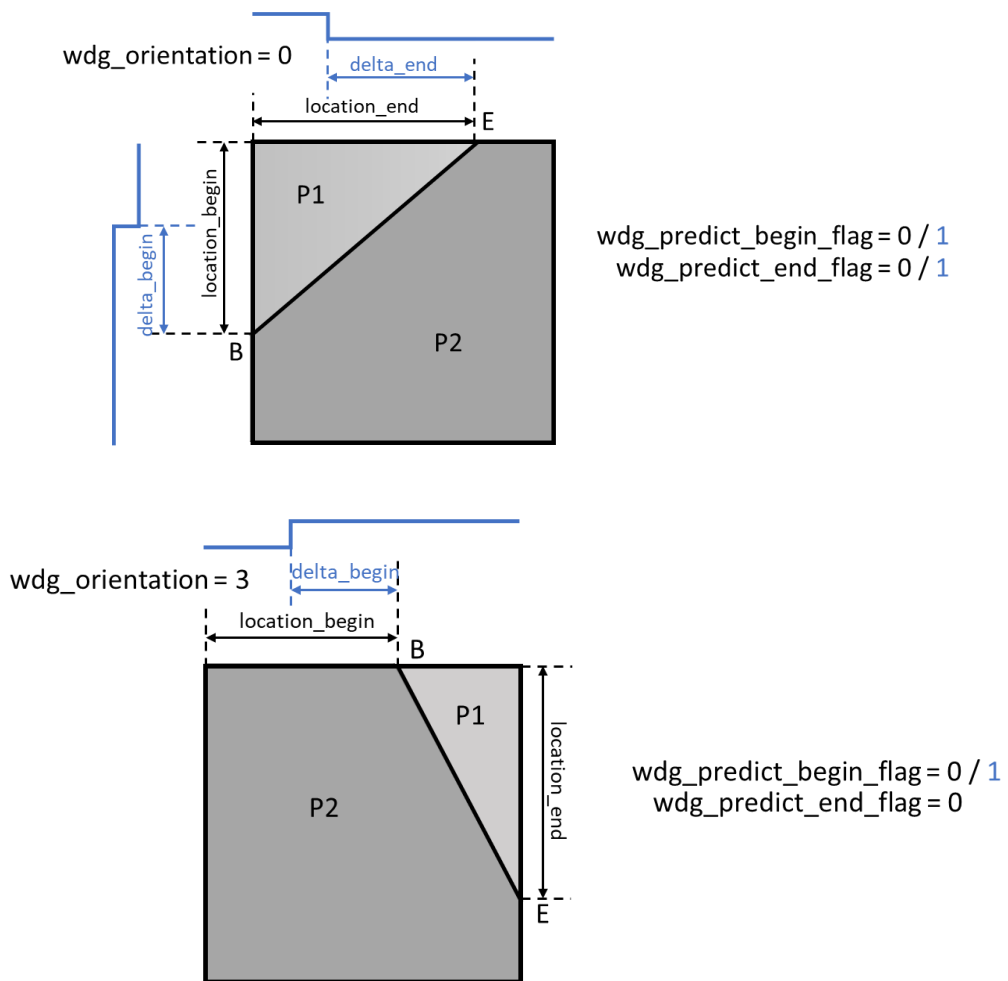
3.2 Non-uniform modes

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).

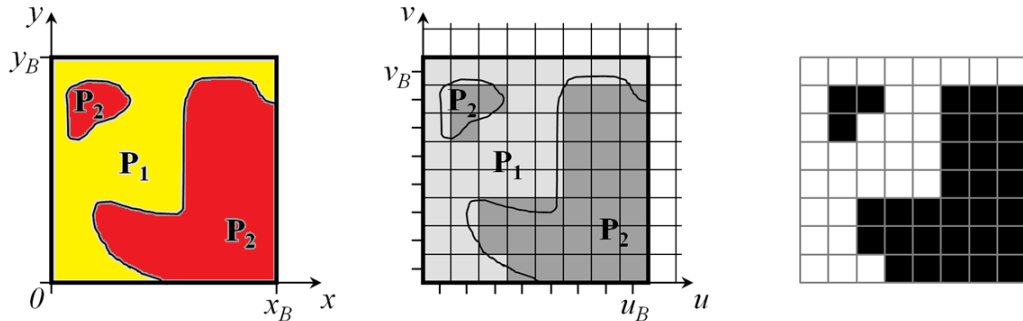


Values B and E may be predicted or signaled directly, depending on the orientation:



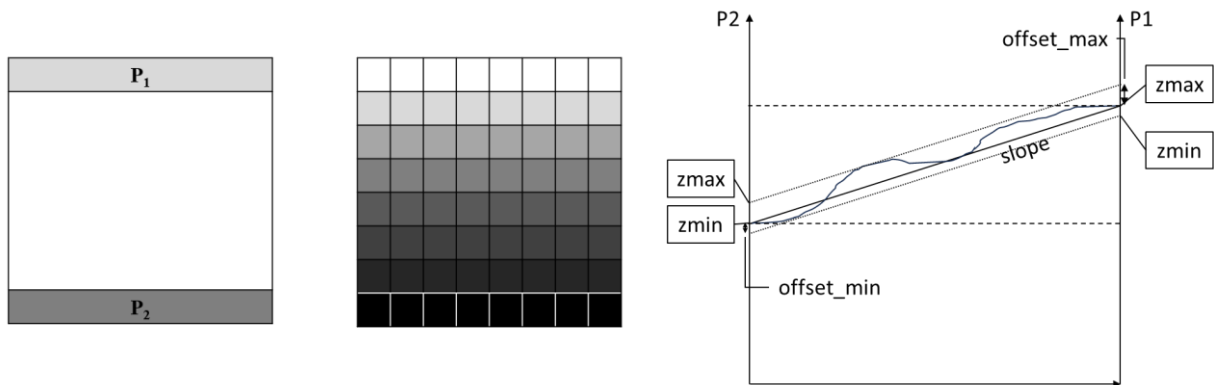
3.2.2 Bilateral split mode

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.



3.3 Two skip modes

3.3.1 Skip depth estimation mode

This mode works exactly the same as the skip mode in the BBGF features. In this mode, the depth estimator may skip the depth estimation for the block and copy all the depth values from the previous frame.

3.3.2 Skip features signaling mode

This mode signals, that the depth values within the block have changed (e.g., objects moved), but the depth range signaled in the previous frame persists.

4 Syntax & semantics

F.1.1.1 General extended geometry assistance SEI payload syntax

	Descriptor
extended_geometry_assistance(payloadSize) {	
ega_num_views_minus1	ue(v)
ega_num_available_assistance_types_minus1	u(4)
for(v = 0; v <= ega_num_views_minus1; v++) {	
ega_assistance_present_flag[v]	u(1)
if(ega_assistance_present_flag[v]) {	
for(t = 0; t <= ega_num_available_assistance_types_minus1; t++) {	
ega_assistance_type_present_flag[v][t]	u(1)
}	
if(ega_assistance_type_present_flag[v][0]) {	
block_based_geometry_features(v)	
} else if(ega_assistance_type_present_flag[v][1]) {	
quadtree_geometry_features(v)	
}	
}	
}	

F.1.1.2 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)
}	

$RootSize = 1 \ll (qtgf_log2_root_size_minus_def[v] + PredefRootSize)$	
$MaxQuadsplitDepth = qtgf_max_quadsplit_depth_minus_def[v] + PredefMaxQuadsplitDepth)$	
$MinUnitSize = 1 \ll (qtgf_log2_min_unit_size_minus_def[v] + PredefMinUnitSize)$	
$MinSymmetric = 1 \ll (qtgf_log2_min_symmetric_minus_def[v] + PredefMinSymmetric)$	
$MinAsymmetric = 1 \ll (qtgf_log2_min_asymmetric_minus_def[v] + PredefMinAsymmetric)$	
$MinArbitrary = 1 \ll (qtgf_log2_min_arbitrary_minus_def[v] + PredefMinArbitrary)$	
$MinWedge = 1 \ll (qtgf_log2_min_wedge_minus_def[v] + PredefMinWedge)$	
$MinBilateral = 1 \ll (qtgf_log2_min_bilateral_minus_def[v] + PredefMinBilateral)$	
$MinGradient = 1 \ll (qtgf_log2_min_gradient_minus_def[v] + PredefMinGradient)$	
if($MinBilateral \leq RootSize$) {	
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*. $BilateralThresholdStep = 1 \ll (qtgf_log2_bilateral_thr_step_minus1[v] + 1)$. 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
<i>PredefinedRootSize</i>	7
<i>PredefinedMaxQuadsplitDepth</i>	4
<i>PredefinedMinUnitSize</i>	3
<i>PredefinedMinSymmetric</i>	3
<i>PredefinedMinAsymmetric</i>	3
<i>PredefinedMinArbitrary</i>	3
<i>PredefinedMinWedge</i>	4
<i>PredefinedMinBilateral</i>	6
<i>PredefinedMinGradient</i>	4

F.1.1.3 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])	
qtf_exact_skip_flag [v][i][j][k]	u(1)
}	
if(!qtf_skip_flag[v][i][j][k]) {	
<i>CurrentBlockSize</i> = <i>RootSize</i> >> k	
if(k <= <i>MaxQuadsplitDepth</i> && <i>CurrentBlockSize</i> > <i>MinUnitSize</i>) {	
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 + <i>CurrentSize</i> >> 1, j, k + 1)	
quadtree_function (v, i, j + <i>CurrentSize</i> >> 1, k + 1)	
quadtree_function (v, i + <i>CurrentSize</i> >> 1, j + <i>CurrentSize</i> >> 1, k + 1)	
}else{	
if(<i>CurrentBlockSize</i> >= min(<i>MinSymmetric</i> , <i>MinAsymmetric</i> , <i>MinArbitrary</i>)) {	
qtf_partitioning_flag [v][i][j][k]	u(1)
}	
if(!qtf_partitioning_flag[v][i][j][k]) {	
if(<i>CurrentBlockSize</i> >= min(<i>MinWedge</i> , <i>MinBilateral</i> , <i>MinGradient</i>)) {	
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(<i>CurrentBlockSize</i> >= max(min(<i>MinWedge</i> , <i>MinBilateral</i>), <i>MinGradient</i>)) {	
qtf_pattern_flag [v][i][j][k]	u(1)

}	
if(qtf_pattern_flag[v][i][j][k]) {	
if(<i>CurrentBlockSize</i> >= max(<i>MinWedge</i> , <i>MinBilateral</i>)) {	
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(<i>CurrentBlockSize</i> >= max(min(<i>MinAsymmetric</i> , <i>MinArbitrary</i>), <i>MinSymmetric</i>)) {	
qtf_part_symmetric_flag [v][i][j][k]	u(1)
}	

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 $\text{qtf_wdg_orientation}[v][i][j][k] < 3$ 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 $\text{qtf_wdg_orientation}[v][i][j][k] \% 2 == 1$ 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, $\text{qtf_grd_dir_flag}[v][i][j][k]$ 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 $\text{qtf_grd_dir_flag}[v][i][j][k] == 0$ or from top to bottom if $\text{qtf_grd_dir_flag}[v][i][j][k] == 1$. $\text{qtf_grd_slope_sign}[v][i][j][k]$ 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 $\text{qtf_grd_dir_flag}[v][i][j][k] == 0$ or from top to bottom if $\text{qtf_grd_dir_flag}[v][i][j][k] == 1$.

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. $\text{qtf_part_direction_flag}[v][i][j][k]$ 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 subblocks of the block with indices i, j, k of the view with index v differ. $\text{qtf_part_symmetric_flag}[v][i][j][k]$ equal to 1 indicates that the area of the two subblocks 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 subblocks of the block with indices i, j, k of the view with index v equals to 3:1 or 1:3. $\text{qtf_part_arbitrary_flag}[v][i][j][k]$ equal to 1 indicates that the ratio between areas of the two subblocks 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.1.1.4 Quadtree geometry features: **delta z function**

delta_z_function (v, i, j, k) {	
if(i == 0 && j == 0) { /* none */	
<i>L</i> MinFlag = 2	
<i>L</i> MaxFlag = 2	
} else if(i == 0) { /* left */	
<i>L</i> MinFlag = 0	
<i>L</i> MaxFlag = 0	
} else if(j == 0) { /* above */	
<i>L</i> MinFlag = 1	
<i>L</i> MaxFlag = 1	
} else {	
dzf_ltmin_flag [v][i][j][k]	u(1)
dzf_ltmax_flag [v][i][j][k]	u(1)
<i>L</i> MinFlag = dzf_ltmin_flag [v][i][j][k]	
<i>L</i> MaxFlag = 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]) {	
<i>Delta</i> Min[k] = - dzf_z_delta_abs_minus1 - 1	
<i>Delta</i> Max[k] = - dzf_z_delta_abs_minus1 - 1	
} else {	
<i>Delta</i> Min[k] = dzf_z_delta_abs_minus1 + 1	
<i>Delta</i> Max[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)
<i>DeltaMin</i> [k] = dzf_c_zmin_delta	
<i>DeltaMax</i> [k] = dzf_c_zmax_delta	
}	
}else{	
<i>DeltaMin</i> [k] = 0	
<i>DeltaMax</i> [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 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:

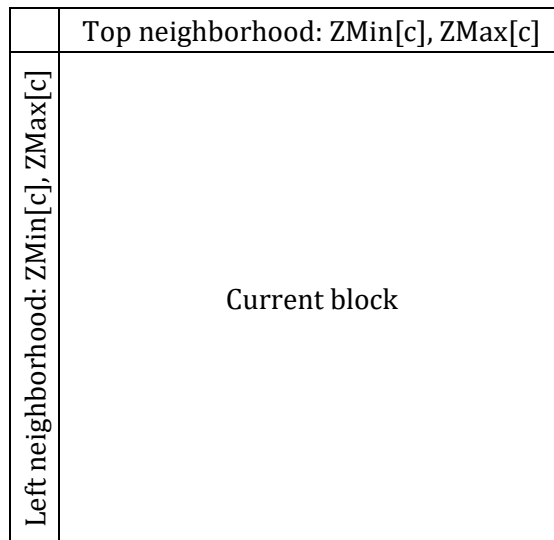
QuantStep = qtgf_quant_step_minus1[v] + 1

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.



```

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:

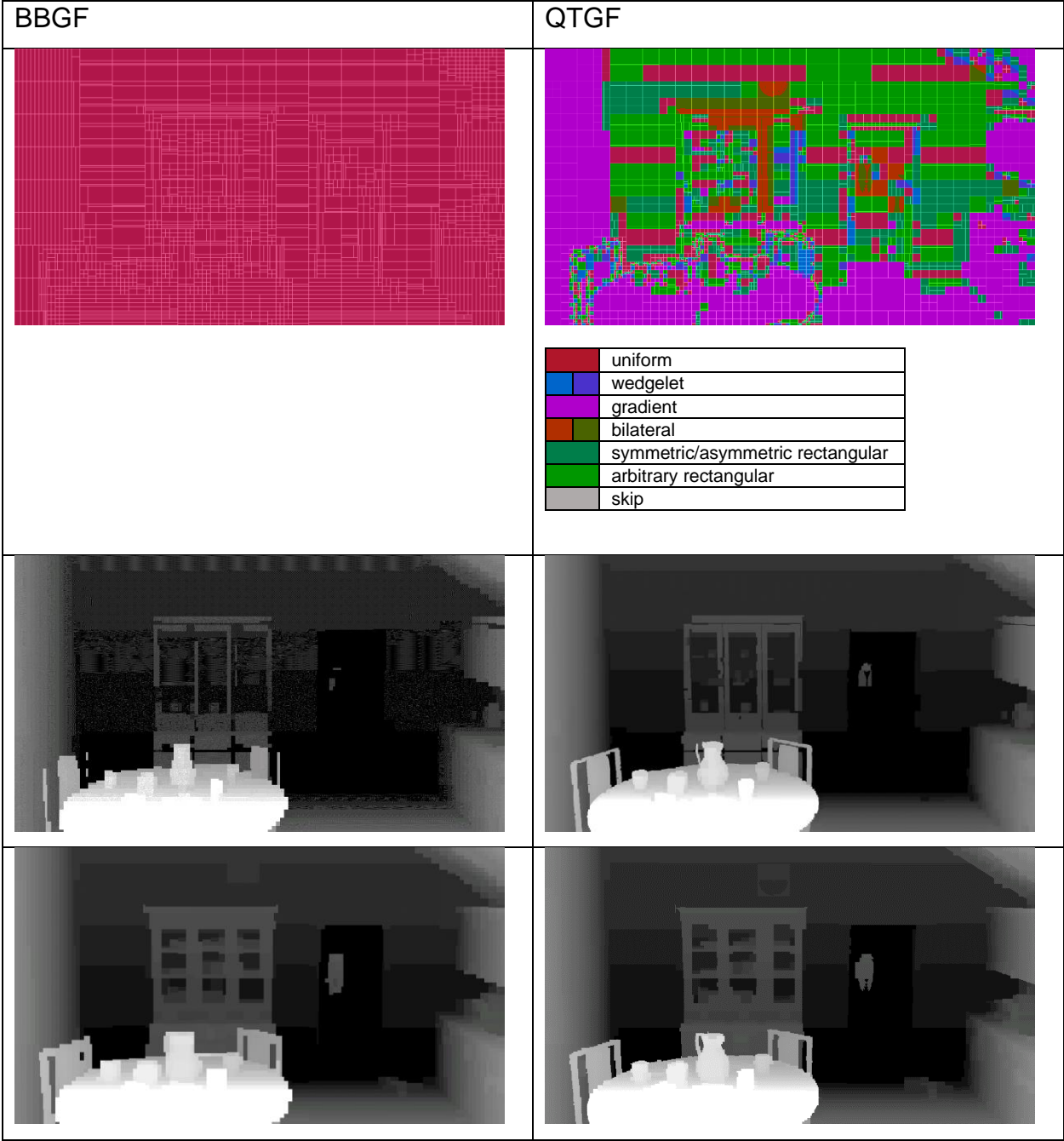
- Supports two types of geometry features:
 - EGA BBGF (block_based_geometry_features),
 - proposed QTGF (quadtree_geometry_features).
- Contains full Volume-Distortion Optimization (VD-opt)
 - equivalent of Rate-Distortion optimization (RD-opt) in regular video codec
 - Lagrange multiplier (λ) used 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 easily multiplexed into bitstream containing other V3C units.
- 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.

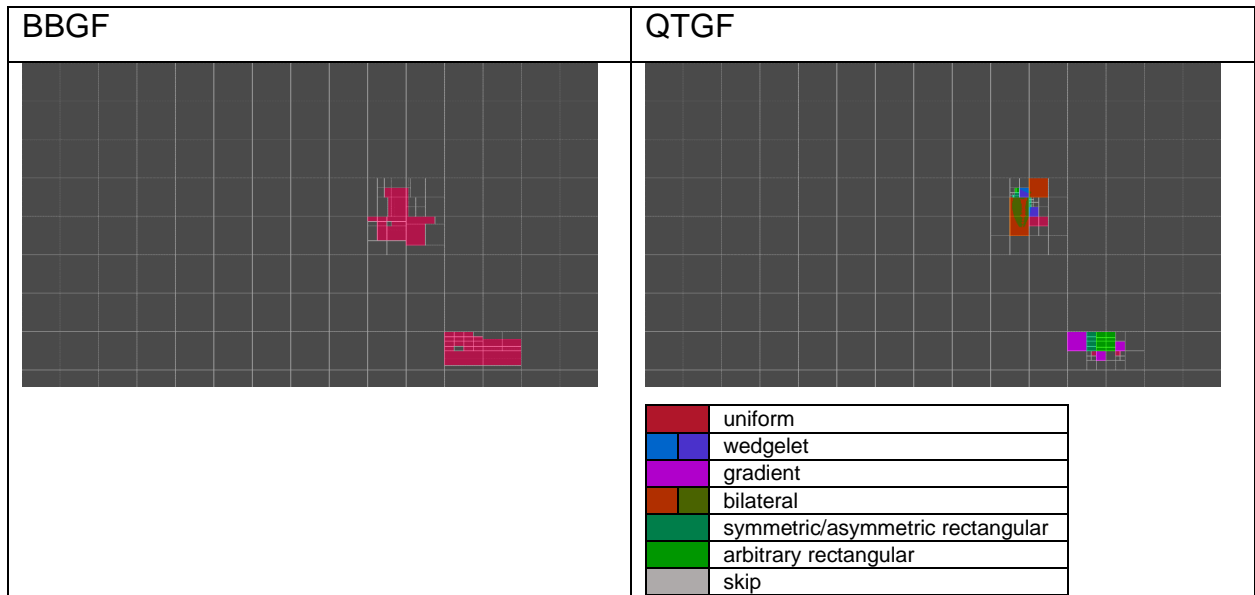
- Encoder and decoder are quite fast, and we still have few ideas how to make it even faster.
- DRFC uses the same underlying libraries as IV-PSNR [M68222] and QMIV [M68224] thus inheriting some optimizations and vectorized routines.
- Designed to allow parallel encoding and decoding at two independent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

6 Visualization

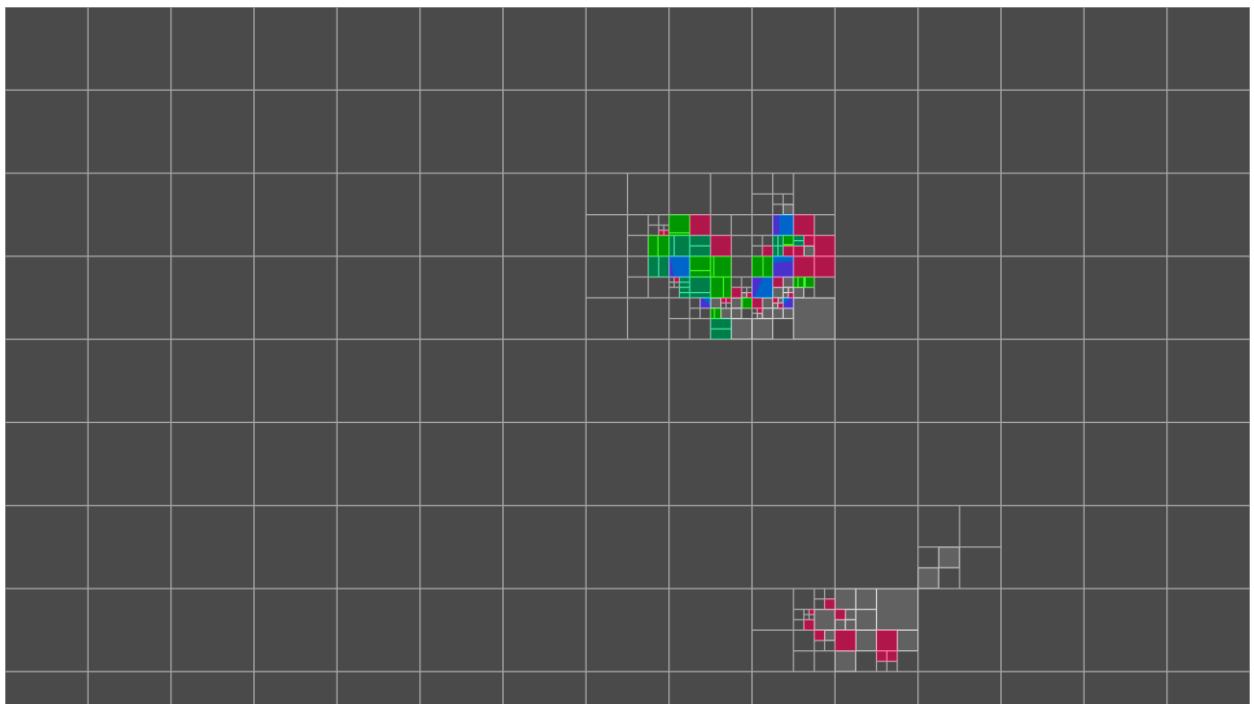
6.1 1st frame (selected modes, ZMin, ZMax)



6.2 2nd frame (grey: skip)



Skip features signaling (light grey) and skip depth estimation (dark grey):



7 Experimental results

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

1. Block-based geometry features (BBGF),
2. Quadtree geometry features without bilateral split mode (QTGF_noB),
3. Quadtree geometry features with bilateral split mode (QTGF_withB).

The features were generated and encoded using the DRFC software.

7.1 G65 anchor vs. BBGF

Mandatory content - Proposal vs. Low/High-bitrate Anchors

Sequence		BD-rate	BD-rate	BD-PSNR	BD-PSNR
		Y-PSNR	IV-PSNR	Y-PSNR	IV-PSNR
Kitchen	J01	11.3%	7.0%	-0.2%	-0.0%
Group	W01	---	---	34.7%	32.4%
Painter	D01	31.1%	18.9%	-0.1%	0.3%
Barn	D03	-23.2%	-12.9%	0.9%	0.7%
Fencing	L01	-60.6%	-35.9%	1.1%	0.7%
CBABasketball	L02	-17.1%	-43.7%	0.0%	0.8%
Average		---	---	6.1%	5.8%

Runtime ratio (%)

Atlas encoding	Video encoding	Decoding & Rendering
100.0%	100.0%	26.2%
100.0%	100.0%	40.9%
100.0%	100.0%	147.0%
100.0%	100.0%	182.5%
38.5%	100.0%	70.7%
37.3%	100.0%	50.7%
79.3%	100.0%	86.3%

Max delta Y-PSNR [dB]

MIV DSE	mXXXXX	Difference [%]
14.73	14.48	-1.7%
27.22	23.39	-14.1%
13.06	12.74	-2.4%
13.64	13.34	-2.2%
25.12	25.22	0.4%
27.43	27.32	-0.4%
20.20	19.42	-3.4%

Max delta IV-PSNR [dB]

MIV DSE	mXXXXX	Difference [%]
15.39	15.02	-2.4%
29.17	24.99	-14.3%
12.91	12.34	-4.4%
15.08	14.51	-3.8%
22.28	21.75	-2.4%
25.68	25.47	-0.8%
20.08	19.01	-4.7%

7.2 G65 anchor vs. QTGF_noB

Mandatory content - Proposal vs. Low/High-bitrate Anchors

Sequence		BD-rate	BD-rate	BD-PSNR	BD-PSNR
		Y-PSNR	IV-PSNR	Y-PSNR	IV-PSNR
Kitchen	J01	6.5%	-3.9%	-0.0%	0.7%
Group	W01	---	---	35.0%	33.0%
Painter	D01	-13.4%	-17.4%	1.2%	1.3%
Barn	D03	-40.1%	-27.2%	1.2%	0.9%
Fencing	L01	-73.6%	-42.7%	1.4%	0.9%
CBABasketball	L02	-44.6%	-61.8%	0.5%	1.2%
Average		---	---	6.5%	6.3%

Runtime ratio (%)

Atlas encoding	Video encoding	Decoding & Rendering
130.6%	100.0%	60.2%
183.5%	100.0%	37.6%
215.3%	100.0%	139.8%
112.9%	100.0%	169.2%
48.9%	100.0%	63.0%
46.7%	100.0%	62.1%
123.0%	100.0%	88.7%

Max delta Y-PSNR [dB]

MIV DSE	mXXXXX	Difference [%]
14.73	13.51	-8.3%
27.22	22.96	-15.6%
13.06	12.75	-2.3%
13.64	13.42	-1.6%
25.12	24.95	-0.7%
27.43	27.37	-0.2%
20.20	19.16	-4.8%

Max delta IV-PSNR [dB]

MIV DSE	mXXXXX	Difference [%]
15.39	13.60	-11.6%
29.17	24.55	-15.9%
12.91	12.54	-2.8%
15.08	14.64	-3.0%
22.28	21.64	-2.9%
25.68	25.48	-0.8%
20.08	18.74	-6.2%

7.3 G65 anchor vs. QTGF_B

Mandatory content - Proposal vs. Low/High-bitrate Anchors

Sequence		BD-rate	BD-rate	BD-PSNR	BD-PSNR
		Y-PSNR	IV-PSNR	Y-PSNR	IV-PSNR
Kitchen	J01	5.0%	-4.4%	0.0%	0.7%
Group	W01	---	---	35.0%	33.0%
Painter	D01	-13.8%	-17.7%	1.3%	1.3%
Barn	D03	-41.1%	-28.2%	1.2%	0.9%
Fencing	L01	-74.3%	-43.4%	1.4%	0.9%
CBABasketball	L02	-46.5%	-65.2%	0.6%	1.4%
Average		---	---	6.6%	6.4%

Runtime ratio (%)

Atlas encoding	Video encoding	Decoding & Rendering
100.0%	100.0%	60.0%
100.0%	100.0%	44.0%
100.0%	100.0%	140.9%
100.0%	100.0%	176.7%
100.0%	100.0%	64.5%
100.0%	100.0%	66.6%
100.0%	100.0%	92.1%

Max delta Y-PSNR [dB]

MIV DSE	mXXXXX	Difference [%]
14.73	13.51	-8.3%
27.22	22.96	-15.7%
13.06	12.72	-2.6%
13.64	13.38	-1.9%
25.12	24.93	-0.8%
27.43	27.35	-0.3%
20.20	19.14	-4.9%

Max delta IV-PSNR [dB]

MIV DSE	mXXXXX	Difference [%]
15.39	13.65	-11.3%
29.17	24.50	-16.0%
12.91	12.52	-3.0%
15.08	14.61	-3.2%
22.28	21.62	-2.9%
25.68	25.48	-0.8%
20.08	18.73	-6.2%

7.4 BBGF vs. QTGF_noB

Mandatory content - Proposal vs. Low/High-bitrate Anchors

Sequence		BD-rate	BD-rate	BD-PSNR	BD-PSNR
		Y-PSNR	IV-PSNR	Y-PSNR	IV-PSNR
Kitchen	J01	-4.1%	-11.0%	0.2%	0.7%
Group	W01	-11.3%	-13.4%	0.3%	0.6%
Painter	D01	-32.3%	-27.8%	1.4%	1.1%
Barn	D03	-16.3%	-13.0%	0.2%	0.2%
Fencing	L01	-25.3%	-10.1%	0.3%	0.1%
CBABasketball	L02	-42.2%	-37.5%	0.4%	0.5%
Average		-21.9%	-18.8%	0.5%	0.5%

Runtime ratio (%)

Atlas encoding	Video encoding	Decoding & Rendering
130.6%	100.0%	229.9%
183.5%	100.0%	91.9%
215.3%	100.0%	95.1%
112.9%	100.0%	92.7%
127.1%	100.0%	89.2%
125.3%	100.0%	122.4%
149.1%	100.0%	120.2%

Max delta Y-PSNR [dB]

MIV DSDE	mXXXXX	Difference [%]
14.48	13.51	-6.7%
23.39	22.96	-1.8%
12.74	12.75	0.1%
13.34	13.42	0.7%
25.22	24.95	-1.1%
27.32	27.37	0.2%
19.42	19.16	-1.5%

Max delta IV-PSNR [dB]

MIV DSDE	mXXXXX	Difference [%]
15.02	13.60	-9.5%
24.99	24.55	-1.8%
12.34	12.54	1.7%
14.51	14.64	0.9%
21.75	21.64	-0.5%
25.47	25.48	0.0%
19.01	18.74	-1.5%

7.5 QTGF_noB vs. QTGF_B

Mandatory content - Proposal vs. Low/High-bitrate Anchors

Sequence		BD-rate	BD-rate	BD-PSNR	BD-PSNR
		Y-PSNR	IV-PSNR	Y-PSNR	IV-PSNR
Kitchen	J01	-1.4%	-0.4%	0.1%	0.0%
Group	W01	-0.0%	-0.2%	0.0%	0.0%
Painter	D01	-0.4%	-0.2%	0.0%	0.0%
Barn	D03	-2.1%	-1.8%	0.0%	0.0%
Fencing	L01	-2.3%	-1.0%	0.0%	0.0%
CBABasketball	L02	-8.7%	-10.5%	0.1%	0.1%
Average		-2.5%	-2.4%	0.0%	0.0%

Runtime ratio (%)

Atlas encoding	Video encoding	Decoding & Rendering
76.6%	100.0%	99.6%
54.5%	100.0%	117.2%
46.5%	100.0%	100.8%
88.6%	100.0%	104.4%
204.5%	100.0%	102.3%
213.9%	100.0%	107.3%
114.1%	100.0%	105.3%

Max delta Y-PSNR [dB]

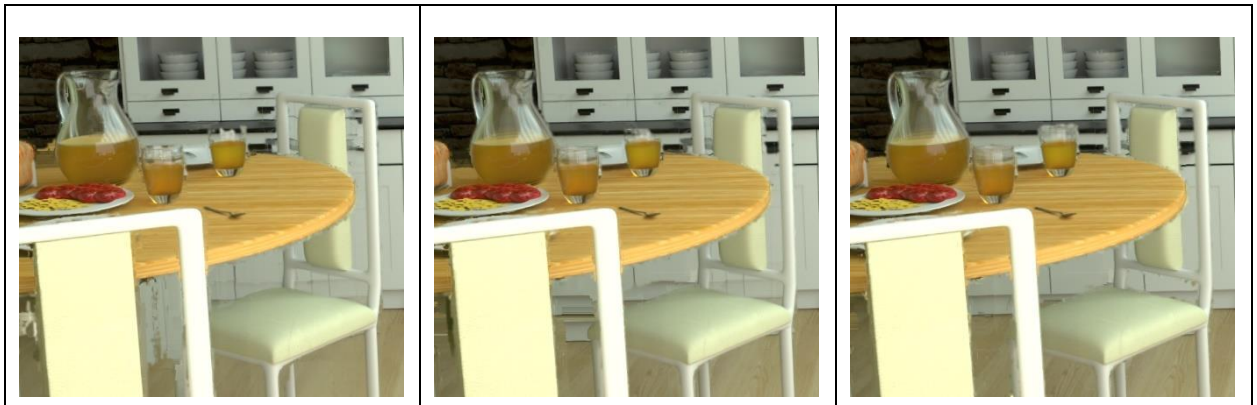
MIV DSDE	mXXXXX	Difference [%]
13.51	13.51	0.0%
22.96	22.96	0.0%
12.75	12.72	-0.2%
13.42	13.38	-0.3%
24.95	24.93	-0.1%
27.37	27.35	-0.1%
19.16	19.14	-0.1%

Max delta IV-PSNR [dB]

MIV DSDE	mXXXXX	Difference [%]
13.60	13.65	0.3%
24.55	24.50	-0.2%
12.54	12.52	-0.2%
14.64	14.61	-0.2%
21.64	21.62	-0.1%
25.48	25.48	0.0%
18.74	18.73	-0.1%

7.6 Subjective comparison (anchor vs. BBGF vs. QTGF)

J01, QP0, p01, f141



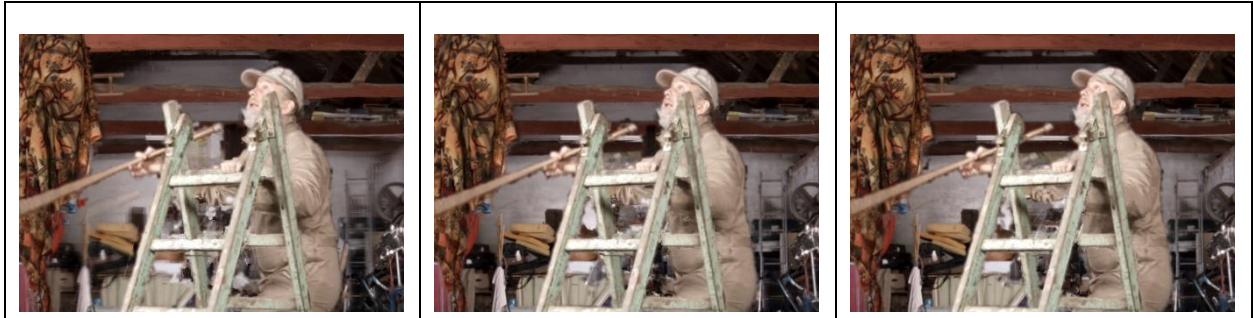
W01, QP0, p01, f35



D01, QP0, p01, f51



D03, QP0, p01, f70



L01, QP0, p01, f42



8 Recommendations

We recommend:

- performing the crosscheck (a volunteer is needed),
- adopting the proposal into the future MIV2 Amd or starting a new CE.

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).