

**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 60248
July 2022, Online

Title: [MIV] Extended geometry assistance SEI
Source: Adrian Dziembowski, Dawid Mieloch (PUT)
Jun Young Jeong, Gwangsoon Lee (ETRI)

Abstract

The document presents a proposal for a new SEI message: Extended Geometry Assistance (EGA). Currently, MIV includes the Geometry Assistance SEI (GA), which is useful for increasing the coding efficiency using the feature-driven DSDE approach (FD-DSDE). However, existing GA SEI has several limitations, which does not allow to further increase efficiency of the FD-DSDE. Proposed EGA SEI is more general and future-proof. The recommendation is to adopt proposed SEI in MIV.

1 Introduction

The syntax was split into general part and parts specifying different types of geometry assistance. Currently, there is only one, based on the current GA SEI, but EGA SEI allows for having several types of assistance.

In the general part, for each view v it is specified, if the assistance is available for that view (flag `ega_assistance_present_flag[v]`). If so, for each assistance type t it is specified if an assistance of type t is available for view v (flag `ega_assistance_type_present_flag[v][t]`).

Assistance type $= 0$ defines the block-based geometry features known from the GA SEI. However, several changes were made to make the syntax more general:

- `bbgf_qs`, `bbgf_log2_bw_minus2` are set per view, not globally, allowing different level of details for some views,
- `bbgf_max_number_of_splits` was added to allow recursive splitting of blocks (or disabling of block splitting if setting number of splits to 0).

Besides these changes, the syntax for the block-based geometry features simplifies to the GA SEI syntax (when the `bbgf_max_number_of_splits` is set to 1).

2 Syntax, Annex F of MIV spec.

F.2.7 Extended geometry assistance SEI payload syntax

F.2.7.1 General

	Descriptor
<code>extended_geometry_assistance(payloadSize) {</code>	
<code>ega_num_views_minus1</code>	<code>ue(v)</code>
<code>ega_num_available_assistance_types_minus1</code>	<code>u(4)</code>
<code>for(v = 0; v <= ega_num_views_minus1; v++) {</code>	
<code>ega_assistance_present_flag[v]</code>	<code>u(1)</code>
<code>if(ega_assistance_present_flag[v] == 1) {</code>	
<code>for(t = 0; t <= ega_num_assistance_types_minus1; t++) {</code>	

ega_assistance_type_present_flag [v][t]	u(1)
}	
if(ega_assistance_type_present_flag[v][0] == 1) {	
block_based_geometry_features(v)	
}	
if(ega_assistance_type_present_flag[v][1] == 1) {	
// something else, up to 16 types	
}	
}	
}	
}	

F.2.7.2 Block-based geometry features

	Descriptor
block_based_geometry_features(v) {	
bbgf_qs [v]	ue(v)
bbgf_log2_bw_minus2 [v]	ue(v)
bbgf_max_number_of_splits [v]	ue(v)
gasBw = 1 << (bbgf_log2_bw_minus2 + 2)	
bbgf_projection_plane_height_minus1 [v]	ue(v)
bbgf_projection_plane_width_minus1 [v]	ue(v)
for(l = 0; l < (bbgf_projection_plane_height_minus1[v] + gasBW) / gasBW; l++) {	
for(c = 0; c < (bbgf_projection_plane_width_minus1[v] + gasBW) / gasBW; c++) {	
recursiveSplitFunction(l, c, 0)	
}	
}	
}	
recursiveSplitFunction(sbl, sbc, lvl) {	
if (lvl < bbgf_max_number_of_splits) {	
bbgf_split_flag	u(1)
}	
if(lvl < bbgf_max_number_of_splits && bbgf_split_flag) {	
bbgf_quad_split_flag	u(1)
if(bbgf_quad_split_flag) {	
recursiveSplitFunction(sbl, sbc, lvl + 1)	
recursiveSplitFunction(sbl, sbc + 1, lvl + 1)	
recursiveSplitFunction(sbl + 1, sbc, lvl + 1)	
recursiveSplitFunction(sbl + 1, sbc + 1, lvl + 1)	
} else {	
bbgf_split_orientation_flag	u(1)
bbgf_split_symmetry_flag	u(1)
if(!bbgf_split_symmetry_flag) {	

bbgf_split_first_block_bigger	u(1)
}	
if(bbgf_split_orientation_flag) {	
recursiveSplitFunction(sbl, sbc, lvl + 1)	
recursiveSplitFunction(sbl, sbc + 1, lvl + 1)	
} else {	
recursiveSplitFunction(sbl, sbc, lvl + 1)	
recursiveSplitFunction(sbl + 1, sbc, lvl + 1)	
}	
}	
} else {	
bbgf_skip_flag	u(1)
if(!bbgf_skip_flag) {	
if (sbl == 0 && sbc == 0) { /*None*/	
<i>LTMinFlag</i> = 2	
<i>LTMaxFlag</i> = 2	
} else if(sbl == 0) { /*Left*/	
<i>LTMinFlag</i> = 0	
<i>LTMaxFlag</i> = 0	
} else if(sbc == 0) { /*Top*/	
<i>LTMinFlag</i> = 1	
<i>LTMaxFlag</i> = 1	
} else {	
bbgf_ltmin_flag	u(1)
bbgf_ltmax_flag	u(1)
<i>LTMinFlag</i> = bbgf_ltmin_flag	
<i>LTMaxFlag</i> = bbgf_ltmax_flag	
}	
bbgf_zmin_delta	se(v)
bbgf_zmax_delta	se(v)
}	
}	
}	

3 Semantics, Annex F of MIV spec.

F.3.9 Extended geometry assistance SEI payload semantics

The extended geometry assistance SEI message indicates suggested information associated with each view that can be used to reduce complexity of a depth estimation or refinement process occurring at decoder side.

F.3.9.1 General

ega_num_views_minus1 plus 1 specifies the number of views for which extended geometry assistance parameters are signalled.

ega_num_available_assistance_types_minus1 plus 1 specifies the number of available assistance types to be checked for each view.

ega_assistance_present_flag[v] equal to 1 indicates that the geometry assistance for view v is present in the syntax structure. **ega_assistance_present_flag**[v] equal to 0 indicates that the geometry assistance for view v is not present in the syntax structure.

ega_assistance_type_present_flag[v][t] equal to 1 indicates that the geometry assistance of type t for view v is present in the syntax structure. **ega_assistance_type_present_flag**[v][t] equal to 0 indicates that the geometry assistance of type t for view v is not present in the syntax structure. **ega_assistance_type_present_flag**[v][t]. t equal to 0 specifies the use of block-based geometry features. t values in range 1 to 15 are reserved for future use by ISO/IEC.









F.3.9.2 Block-based geometry features

This section contains semantics copied from F.3.8 (geometry assistance SEI), except for:

- Prefixes “gas_” changed to “bbgf_”,
- **Highlighted** parts.

A view is uniformly divided into square blocks, and each block can be further **recursively** divided **once** into smaller subblocks of square or rectangular shapes. The top left corner of the first block coincides with the top left corner of a view. One or more syntax elements are associated with each block. The syntax may indicate that the block is skipped, thereby suggesting that the depth of the current block does not need to be updated. If the syntax indicates that the current block is not skipped, the remaining syntax indicates the suggested minimum and maximum geometry values present in the block. A depth estimation process can take benefit of this information to reduce the search space of depth candidates, and to avoid producing geometry values that are outside of the suggested range. Table F-1 indicates the available split types (including no split) of a block, with the associated values of the block division syntax. Figure F-1 shows an example of the block subdivision of a view.

Table F-1: The different block split types and associated values of the syntax elements

								
bbgf_split_flag	0	1	1	1	1	1	1	1
bbgf_quad_split_flag	-	1	0	0	0	0	0	0
bbgf_split_orientation_flag	-	-	0	0	0	1	1	1
bbgf_split_symmetry_flag	-	-	1	0	0	1	0	0
bbgf_split_first_block_bigger	-	-	-	0	1	-	0	1

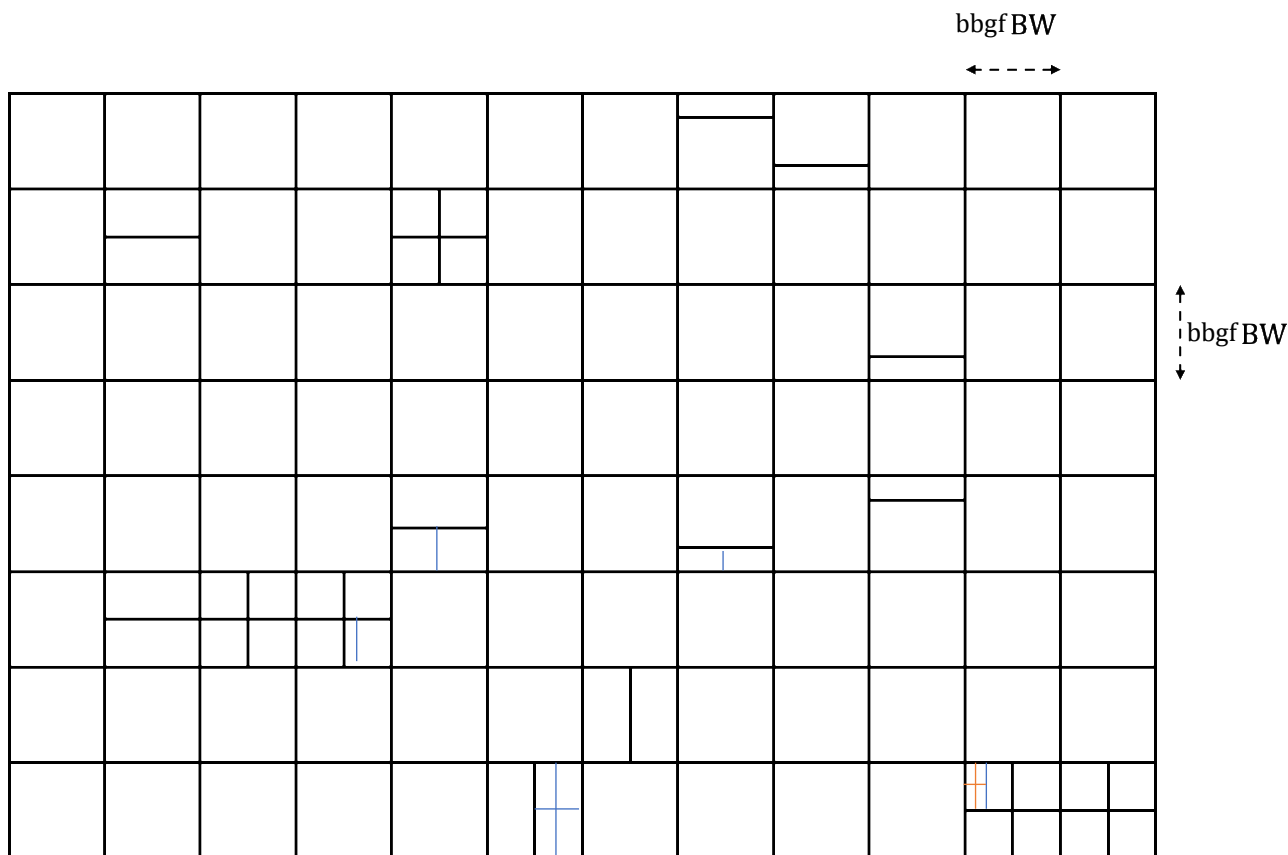


Figure F-1: Example of a partition of a view into all possible block divisions with `bbgf_max_number_of_splits [v]` set to 3

`bbgf_qs [v]` specifies the quantization step for the suggested geometry range boundaries for view v .

`bbgf_log2_bw_minus2 [v]` specifies the value of the variable `bbgfBW` for view v , as shown in Figure F-1, as follows:

$$bbgfBW = 1 \ll (bbgf_log2_bw_minus2 + 2) \quad (F-1)$$

`bbgf_max_number_of_splits [v]` specifies the maximum number of splits of each block for view v .

`bbgf_projection_plane_width_minus1 [v]` plus 1 and `bbgf_projection_plane_height_minus1 [v]` plus 1 specify the horizontal and vertical resolutions, respectively, of the camera projection planes, expressed in coded luma samples, for which geometry assistance parameters are signalled.

`bbgf_split_flag` equal to 1 indicates that the current block is split into smaller subblocks. `bbgf_split_flag` equal to 0 indicates that the current block is not split into smaller subblocks.

`bbgf_quad_split_flag` equal to 0 indicates that the current block is split into two rectangular subblocks. `bbgf_quad_split_flag` equal to 1 indicates that the current block is split into four square subblocks of identical sizes.

`bbgf_split_orientation_flag` equal to 0 indicates that the current block is split horizontally. `bbgf_split_orientation_flag` equal to 1 indicates that the current block is split vertically.

bbgf_split_symmetry_flag equal to 0 indicates that the area of the two subblocks differ, with the division occurring at a quarter of the block width from one end. **bbgf_split_symmetry_flag** equal to 1 indicates that the area of the two subblocks is equal.

bbgf_split_first_block_bigger equal to 1 indicates that the first subblock (top subblock if **bbgf_split_orientation_flag** is equal to 0, and left subblock if **bbgf_split_orientation_flag** is equal to 1) is bigger than the second subblock. **bbgf_split_first_block_bigger** equal to 0 indicates that the first subblock is smaller than the second subblock.

bbgf_skip_flag equal to 0 indicates that a **bbgf_zmin_delta** and a **bbgf_zmax_delta** syntax elements are present in the bitstream, and that a **bbgf_ltmin_flag** and a **bbgf_ltmax_flag** may be present. **bbgf_skip_flag** equal to 1 indicates that no other syntax elements are present in the bitstream for the current block, and it suggests that the geometry information in this block has not changed since the previous frame in display order.

bbgf_ltmin_flag equal to 0 indicates that the prediction of the current minimum geometry is to be taken from the left block, otherwise from the top block.

bbgf_ltmax_flag equal to 0 indicates that the prediction of the current maximum geometry is to be taken from the left block, otherwise from the top block.

bbgf_zmin_delta specifies the remainder to be added to the prediction to obtain the minimum geometry value suggested for the current block.

bbgf_zmax_delta specifies the remainder to be added to the prediction to obtain the maximum geometry value suggested for the current block

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 current block are derived by the following formulae:

$$ZMin = (LTMinFlag == 2 ? 0 : LTMinFlag == 1 ? ZMinTop : ZMinLeft) + bbgf_qs * bbgf_zmin_delta \quad (F-2)$$

$$ZMax = (LTMaxFlag == 2 ? 0 : LTMaxFlag == 1 ? ZMinTop : ZMinLeft) + bbgf_qs * bbgf_zmax_delta \quad (F-3)$$

4 SEI payload syntax, Annex F of V3C spec.

F.2.1 General SEI message syntax

sei_payload(payloadType, payloadSize) {	Descriptor
if((nal_unit_type == NAL_PREFIX_NSEI) (nal_unit_type == NAL_PREFIX_ESEI)) {	
if(payloadType == 0)	

buffering_period(payloadSize)	
else if(payloadType == 1)	
atlas_frame_timing(payloadSize)	
else if(payloadType == 2)	
filler_payload(payloadSize)	
else if(payloadType == 3)	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType == 4)	
user_data_unregistered(payloadSize)	
else if(payloadType == 5)	
recovery_point(payloadSize)	
else if(payloadType == 6)	
no_reconstruction(payloadSize)	
else if(payloadType == 7)	
time_code(payloadSize)	
else if(payloadType == 8)	
sei_manifest(payloadSize)	
else if(payloadType == 9)	
sei_prefix_indication(payloadSize)	
else if(payloadType == 10)	
active_sub_bitstreams(payloadSize)	
else if(payloadType == 11)	
component_codec_mapping(payloadSize)	
else if(payloadType == 12)	
scene_object_information(payloadSize)	
else if(payloadType == 13)	
object_label_information(payloadSize)	
else if(payloadType == 14)	
patch_information(payloadSize)	
else if(payloadType == 15)	
volumetric_rectangle_information(payloadSize)	
else if(payloadType == 16)	
atlas_object_association(payloadSize)	
else if(payloadType == 17)	
viewport_camera_parameters(payloadSize)	
else if(payloadType == 18)	
viewport_position(payloadSize)	
else if(payloadType == 20)	
packed_independent_regions(payloadSize)	
else if(payloadType == 64)	
attribute_transformation_params(payloadSize) /* Specified in Annex H */	
else if(payloadType == 65)	
occupancy_synthesis(payloadSize) /* Specified in Annex H */	
else if(payloadType == 66)	

geometry_smoothing(payloadSize) /* Specified in Annex H */	
else if(payloadType == 67)	
attribute_smoothing(payloadSize) /* Specified in Annex H */	
else if(payloadType == 128)	
viewing_space(payloadSize) /* Specified in ISO/IEC 23090-12 */	
else if(payloadType == 129)	
viewing_space_handling(payloadSize) /* Specified in ISO/IEC 2309-12 */	
else if(payloadType == 130)	
geometry_upscaling_parameters(payloadSize) /* Specified in ISO/IEC 23090-12 */	
else if(payloadType == 131)	
atlas_view_enabled(payloadSize) /* Specified in ISO/IEC 2309-12 */	
else if(payloadType == 132)	
omaf_v1_compatible(payloadSize) /* Specified in ISO/IEC 23090-12 */	
else if(payloadType == 133)	
geometry_assistance(payloadSize) /* Specified in ISO/IEC 23090-12 */	
else if(payloadType == 134)	
extended_geometry_assistance(payloadSize) /* Specified in ISO/IEC 23090-12 */	
else	
reserved_sei_message(payloadSize)	
}	
else {	
/*(nal_unit_type == NAL_SUFFIX_NSEI) (nal_unit_type == NAL_SUFFIX_ESEI)*/	
if(payloadType == 2)	
filler_payload(payloadSize)	
else if(payloadType == 3)	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType == 4)	
user_data_unregistered(payloadSize)	
else if(payloadType == 19)	
decoded_atlas_information_hash(payloadSize)	
else if(payloadType == 133)	
geometry_assistance(payloadSize) /* Specified in ISO/IEC 23090-12 */	
else if(payloadType == 134)	
extended_geometry_assistance(payloadSize) /* Specified in ISO/IEC 23090-12 */	
else	
reserved_sei_message(payloadSize)	
}	
if(more_data_in_payload()) {	
if(payload_extension_present())	
sp_reserved_payload_extension_data	u(v)
byte_alignment()	
}	

}	
---	--

5 Other changes, Annex F of MIV spec.

Table F-1 – The essential and non-essential SEI messages

SEI message	NAL Type	Conformance Type
Viewing space	NAL_PREFIX_NSEI	N/A
Viewing space handling	NAL_PREFIX_NSEI	N/A
Geometry upscaling parameters	NAL_PREFIX_NSEI	N/A
Atlas view enabled	NAL_PREFIX_NSEI	N/A
OMAF v1 compatible	NAL_PREFIX_NSEI	N/A
Geometry assistance	NAL_PREFIX_NSEI/ NAL_SUFFIX_NSEI	N/A
Extended geometry assistance	NAL_PREFIX_NSEI/ NAL_SUFFIX_NSEI	N/A

Table F-2 – Persistence scope of SEI messages

SEI message	Persistence scope
Viewing space	The remainder of the bitstream or until a new viewing space SEI message
Viewing space handling	The remainder of the bitstream or until a new view space handling SEI message
Geometry upscaling parameters	The remainder of the bitstream or until a new geometry upscaling parameters SEI message
Atlas view enabled	Specified by the semantics of the SEI message.
OMAF v1 compatible	The remainder of the sequence
Geometry assistance	The coded atlas access unit containing the SEI message
Extended geometry assistance	The coded atlas access unit containing the SEI message

6 Use cases

Existing:

- Typical geometry assistance (features sent for all views, no recursion, m56950),
- Recursive features (m57347),
- Features sent for a subset of views (m58047),
- Recursive features sent for a subset of views (m58334).

New:

- Different level of details for different views (quantization step, block width, number of splits set per view),
- Possibility for adding new schemes of feature extraction (not only block-based rectangular grid).

7 Recommendations

We recommend adopting proposed SEI in MIV.

8 Acknowledgement

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).