INTERNATIONAL STANDARD

ISO/IEC 23090-12

First edition
2023-08

Information technology — Coded representation of immersive media —

Part 12: MPEG immersive video

Technologies de l'information — Représentation codée de média immersifs —

Partie 12: Vidéo immersive MPEG

Reference number
ISO/IEC 23090-12:2023(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword ..... **v**
Introduction ..... **vi**
1 Scope **1**
2 Normative reference **1**
3 Terms and definitions **1**
4 Abbreviated terms **3**
5 Conventions **3**
6 Overall V3C characteristics, decoding operations, and post-decoding processes **4**
7 Bitstream format, partitioning, and scanning processes **4**
7.1 General 4
7.2 V3C bitstream formats 4
7.3 NAL bitstream formats 4
7.4 Partitioning of atlas frames into tiles 4
7.5 Tile partition scanning processes 4
7.6 Mapping of views to V3C components 4
7.7 Sources and outputs 5
8 Syntax and semantics **6**
8.1 Method of specifying syntax in tabular form 6
8.2 Specification of syntax functions and descriptors 6
8.3 Syntax in tabular form 6
8.3.1 General syntax 6
8.3.2 V3C unit syntax 6
8.3.3 Byte alignment syntax 6
8.3.4 V3C parameter set syntax 6
8.3.5 NAL unit syntax 6
8.3.6 Raw byte sequence payloads, trailing bits, and byte alignment syntax 7
8.3.7 Atlas tile data unit syntax 7
8.3.8 Supplemental enhancement information message syntax 7
8.3.9 V3C MIV extension syntax in tabular form 7
8.4 Semantics 12
8.4.1 General semantics 12
8.4.2 V3C MIV extension semantics 12
8.4.3 Order of V3C units and association to coded information 19
9 Decoding process **20**
9.1 General decoding process 20
9.2 Atlas data decoding process 20
9.2.1 General atlas data decoding process 20
9.2.2 Decoding process for a coded atlas frame 20
9.2.3 Atlas NAL unit decoding process 20
9.2.4 Atlas tile header decoding process 20
9.2.5 Decoding process for patch data units 20
9.2.6 Decoding process of the block to patch map 21
9.2.7 Conversion of tile level patch information to atlas level patch information 21
9.3 Occupancy video decoding process 22
9.4 Geometry video decoding process 22
9.5 Attribute video decoding process 22
9.6 Packed video decoding process 22
9.7 Common atlas data decoding process 22
9.7.1 General common atlas data decoding process 22
9.7.2 Decoding process for a coded common atlas frame 23
9.7.3 Common atlas NAL unit decoding process 23

9.7.4 Common atlas frame order count derivation process 23
9.7.5 Common atlas frame MIV extension decoding process 23
9.8 Sub-bitstream extraction process 28
9.8.1 General 28
9.8.2 V3C unit extraction 28
9.8.3 NAL unit extraction process 28
9.8.4 Group extraction process 28
10 Pre-reconstruction process 28
11 Reconstruction process 28
12 Post-reconstruction process 28
13 Adaptation process 28
14 Parsing process 28
Annex A (normative) **Profiles, tiers, and levels 29**
Annex B (informative) **Post-decoding conversion to nominal video formats 32**
Annex C (informative) **V3C sample stream format 34**
Annex D (normative) **NAL sample stream format 35**
Annex E (normative) **Atlas hypothetical reference decoder 36**
Annex F (normative) **Supplemental enhancement information 37**
Annex G (informative) **Volumetric usability information 53**
Annex H (Informative) **Overview of the rendering processes 54**
Bibliography 71

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 23090 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

This document was developed to support compression of immersive video content, in which a real or virtual 3D scene is captured by multiple real or virtual cameras. The use of this document enables storage and distribution of immersive video content over existing and future networks, for playback with 6 degrees of freedom of view position and orientation.

Information technology — Coded representation of immersive media —

Part 12: MPEG immersive video

1 Scope

This document specifies the syntax, semantics and decoding processes for MPEG immersive video (MIV), as an extension of ISO/IEC 23090-5. It provides support for playback of a three-dimensional (3D) scene within a limited range of viewing positions and orientations, with 6 Degrees of Freedom (6DoF).

2 Normative reference

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23090-5[1)], *Information technology — Coded Representation of Immersive Media — Part 5: Visual Volumetric Video-based Coding (V3C) and Video-based Point Cloud Compression (V-PCC)*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 23090-5 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
atlas sample
position on the rectangular frame associated with an *atlas*

3.2
coded MIV sequence
coded V3C sequence conforming to the constraints specified in this document

3.3
decoding process
process specified in this document that reads a *bitstream* and derives *patch data* and related information that can be used to render a *viewport* (3.22)

3.4
decoding order
order in which syntax elements are processed by the *decoding process* (3.3)

1) Under preparation. Stage at time of publication: ISO/IEC FDIS 23090-5:2023.

3.5
field of view
FOV
angular region of the observable world in captured/recorded content or in a physical display device

3.6
MIV access unit
V3C composition unit that is a set of all *sub-bitstream access units* (3.13) that share the same *decoding order* (3.4) count

3.7
MIV coded sub-bitstream sequence
sub-bitstream IRAP access unit (3.14) followed by zero or more *sub-bitstream access units* (3.13)

Note 1 to entry: A *MIV coded sub-bitstream sequence* is a *coded sub-bitstream sequence* conforming to the constraints specified in this document.

3.8
MIV IRAP access unit
MIV access unit (3.6) for which all *sub-bitstream access units* (3.13) are *sub-bitstream IRAP access units* (3.14)

Note 1 to entry: A *MIV IRAP access unit* is a *V3C IRAP composition unit* conforming to *the constraints specified in this document.*

3.9
multi-plane image
MPI
set of pairs of texture and transparency *attribute frames*, each associated with an implicit constant geometry frame

3.10
renderer
embodiment of a process to create a *viewport* (3.22) from a *volumetric frame* corresponding to a *viewing orientation* (3.19) and *viewing position* (3.20)

3.11
source
one or more video sequences, each containing *geometry* or an *attribute* such as texture and transparency information before encoding

3.12
source view
source (3.11) video material before encoding that corresponds to the format of a *view* (3.15), which may have been acquired by capture of a 3D scene by a real or virtual camera

3.13
sub-bitstream access unit
partition of a *sub-bitstream* that has a certain *decoding order* (3.4) count

Note 1 to entry: A *sub-bitstream access unit* is a *sub-bitstream composition unit*.

3.14
sub-bitstream IRAP access unit
sub-bitstream access unit (3.13) that forms an independent random-access point for the *sub-bitstream*

Note 1 to entry: A a sub-bitstream IRAP access unit is a sub-bitstream IRAP composition unit.

3.15
view
2D rectangular arrays of *view samples* (3.18) consisting of *attribute* frames and corresponding *geometry* frame representing the projection of a *volumetric frame* onto a surface using *view parameters* (3.16)

3.16
view parameters
parameters of the projection used to generate a *view* (3.15) from a *volumetric frame*, including intrinsic and extrinsic parameters

3.17
view parameters list
listing of one or more *view parameters* (3.16)

3.18
view sample
position on the rectangular frame associated with a *view* (3.15)

3.19
viewing orientation
unit quaternion representing the orientation of a user who is consuming the visual content

3.20
viewing position
triple of x, y, z characterizing the position in the Cartesian coordinates of a user who is consuming the visual content

3.21
viewing space
domain constraints for an intended *viewport* (3.22) rendering

Note 1 to entry: The domain is defined in the 3D global space and related to the *viewing orientation* (3.19); it defines a scale between 0 and 1 for every point in space for a given direction of the *viewport* (3.22), to be used by the application.

3.22
viewport
view (3.15) suitable for display and viewing

4 Abbreviated terms

For the purposes of this document, the abbreviated terms given in ISO/IEC 23090-5 and the following apply.

CSG	constructive solid geometry
ERP	equirectangular projection
HMD	head-mounted display
MIV	MPEG immersive video
OMAF	Omnidirectional media format

5 Conventions

The specifications in ISO/IEC 23090-5:—, Clause 5 and its subclauses apply with the following addition to subclause 5.8:

Cos(x) the trigonometric cosine function operating on an argument x in units of radians

Dot(x, y) dot product function, known also as scalar product function, operating on two vectors x and y

Norm(x) = Sqrt(Norm2(x, x))

Norm2(x) = Abs(Dot(x, x))

Sin(x) the trigonometric sine function operating on an argument x in units of radians

π the ratio of a circle's circumference to its diameter

6 Overall V3C characteristics, decoding operations, and post-decoding processes

The specifications in ISO/IEC 23090-5:—, Clause 6 apply.

7 Bitstream format, partitioning, and scanning processes

7.1 General

The specifications in ISO/IEC 23090-5:—, subclause 7.1 apply.

7.2 V3C bitstream formats

The specifications in ISO/IEC 23090-5:—, subclause 7.2 apply.

7.3 NAL bitstream formats

The specifications in ISO/IEC 23090-5:—, subclause 7.3 apply.

7.4 Partitioning of atlas frames into tiles

The specifications in ISO/IEC 23090-5:—, subclause 7.4 apply.

7.5 Tile partition scanning processes

The specifications in ISO/IEC 23090-5:—, subclause 7.5 apply.

7.6 Mapping of views to V3C components

This subclause describes the concept of views and its mapping to patches in V3C components.

A view represents a field of view of a volumetric frame for a particular view position and orientation. Each view, at a given time instance, may be represented by one 2D frame providing geometry information plus one 2D frame per attribute, providing attribute information, and occupancy information that may either be embedded within geometry or represented explicitly as a 2D frame. Each view may use the equirectangular, perspective, or orthographic projection format. The atlas components of a view use the same projection format.

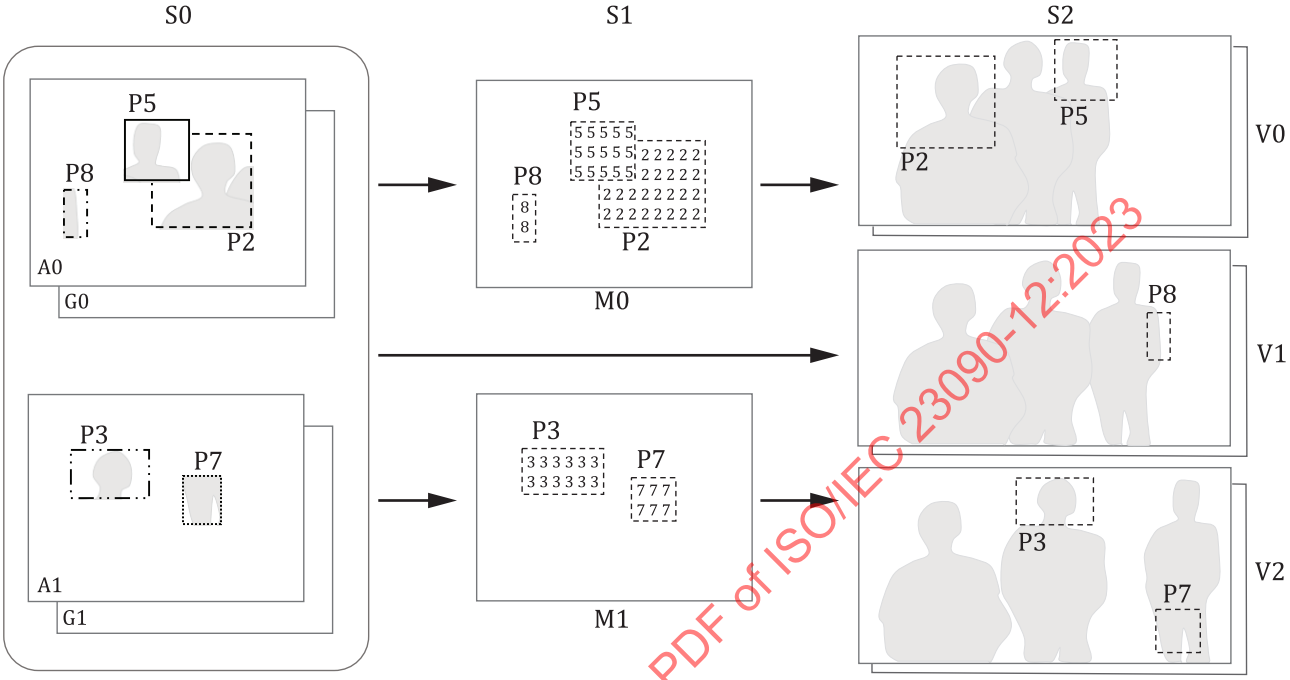
The volumetric frame and all views each have an associated reference frame. Cartesian coordinates of 3D points can therefore be expressed according to the reference frame of the scene, as represented by the volumetric frame, or according to the reference frame of any view. The camera extrinsic parameters (position and orientation) of the views specify the relations between their reference frames, enabling switching of the 3D coordinate system to represent a 3D point from one reference frame attached to a given view to another reference frame attached to another view.

A coded atlas contains information describing the patches within the atlas. For each patch, a view ID is signalled which identifies which view the patch originated from.

A patch represents a rectangular region of a view, with corresponding regions in all present atlas components: attribute(s), geometry, and occupancy. The size (width and height) of each patch in an atlas is signalled. In this version of the document, the size of a patch is always the same as the corresponding

rectangular region in the view texture attribute component, but scaling may optionally be applied to the geometry component or the occupancy component.

Figure 1 shows an illustrative example, in which two atlases contain five patches, which are mapped to three views, with a texture attribute component and a geometry component.



Key

A0-A1	decoded attribute frames for atlas 0 and 1
G0-G1	decoded geometry frames for atlas 0 and 1
M0-M1	maps for atlas 0 and 1
P0-P8	patches
S0	stage 0 where attribute and geometry frames are decoded for each atlas
S1	stage 1 where block to patch mapping is performed
S2	stage 2 where patches are mapped to views
V0-V2	reconstructed views

Figure 1 — Example mapping of 5 patches in 2 atlases to 3 views

7.7 Sources and outputs

The volumetric video source that is represented by the bitstream is a sequence of volumetric frames. Each volumetric frame is represented by one or more view frames, each of which may be represented by a geometry picture, an attribute picture for each attribute, and occupancy information, which may be conveyed in the geometry picture or represented separately.

The outputs of the decoding process are described in subclause 9.1.

The outputs of the non-normative rendering process of Annex H are a sequence of one or more views. The number of views and the associated view parameters may be selected by the application. For example, a single view may be output corresponding to a viewport suitable for display, or a set of views may be output which correspond to the source view parameters.

8 Syntax and semantics

8.1 Method of specifying syntax in tabular form

The specifications in ISO/IEC 23090-5:—, subclause 8.1 apply.

8.2 Specification of syntax functions and descriptors

The specifications in ISO/IEC 23090-5:—, subclause 8.2 apply.

8.3 Syntax in tabular form

8.3.1 General syntax

The specifications in ISO/IEC 23090-5:—, subclause 8.3 apply with the following addition.

An overview of the V3C bitstream structure with MIV extensions is represented in Figure 2.

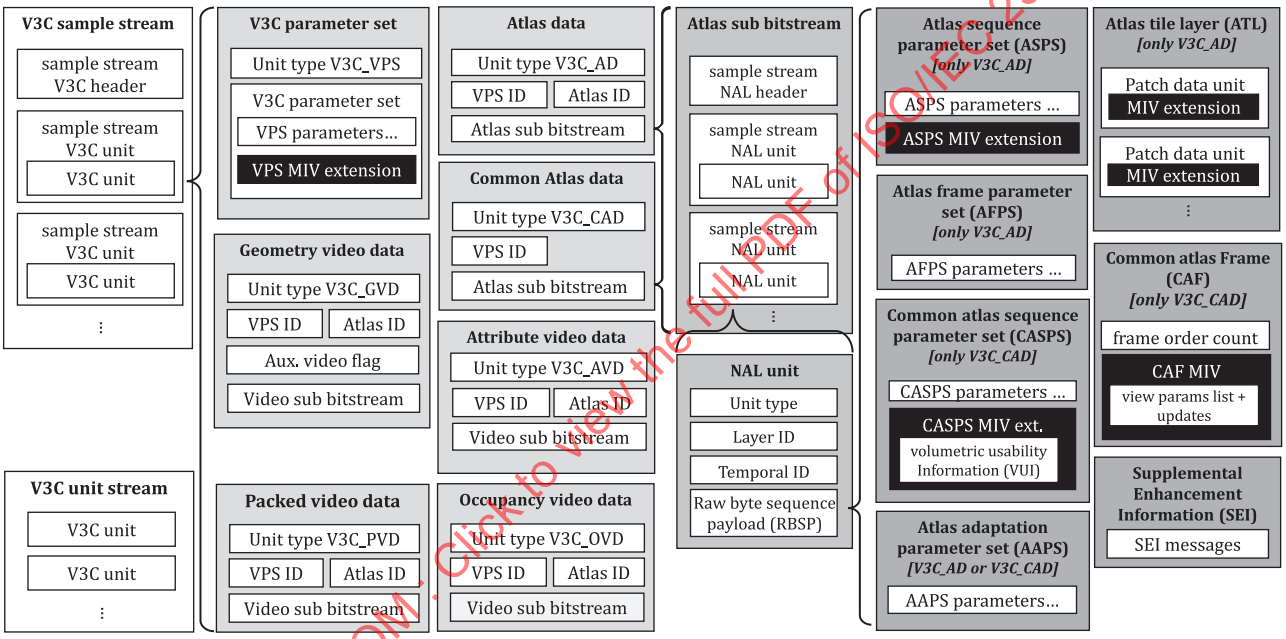


Figure 2 — Overview of V3C bitstream with MIV extensions

8.3.2 V3C unit syntax

The specifications in ISO/IEC 23090-5:—, subclause 8.3.2 apply.

8.3.3 Byte alignment syntax

The specifications in ISO/IEC 23090-5:—, subclause 8.3.3 apply.

8.3.4 V3C parameter set syntax

The specifications in ISO/IEC 23090-5:—, subclause 8.3.4 apply.

8.3.5 NAL unit syntax

The specifications in ISO/IEC 23090-5:—, subclause 8.3.5 apply.

8.3.6 Raw byte sequence payloads, trailing bits, and byte alignment syntax

The specifications in ISO/IEC 23090-5:—, subclause 8.3.6 apply.

8.3.7 Atlas tile data unit syntax

The specifications in ISO/IEC 23090-5:—, subclause 8.3.7 apply.

8.3.8 Supplemental enhancement information message syntax

The specifications in ISO/IEC 23090-5:—, subclause 8.3.8 apply.

8.3.9 V3C MIV extension syntax in tabular form

8.3.9.1 V3C parameter set MIV extension syntax

vps_miv_extension( ) {	Descriptor
vme_geometry_scale_enabled_flag	u(1)
vme_embedded_occupancy_enabled_flag	u(1)
if(!vme_embedded_occupancy_enabled_flag)	
vme_occupancy_scale_enabled_flag	u(1)
group_mapping( )	
}	

8.3.9.2 Group mapping syntax

group_mapping( ) {	Descriptor
gm_group_count	u(4)
if(gm_group_count > 0)	
for(k = 0; k <= vps_atlas_count_minus1; k++)	
j = vps_atlas_id[k]	
gm_group_id[j]	u(v)
}	

8.3.9.3 Atlas sequence parameter set MIV extension syntax

asps_miv_extension( ) {	Descriptor
asme_ancillary_atlas_flag	u(1)
asme_embedded_occupancy_enabled_flag	u(1)
if(asme_embedded_occupancy_enabled_flag)	
asme_depth_occ_threshold_flag	u(1)
asme_geometry_scale_enabled_flag	u(1)
if(asme_geometry_scale_enabled_flag) {	
asme_geometry_scale_factor_x_minus1	ue(v)
asme_geometry_scale_factor_y_minus1	ue(v)
}	
if(!asme_embedded_occupancy_enabled_flag)	
asme_occupancy_scale_enabled_flag	u(1)
if(!asme_embedded_occupancy_enabled_flag && asme_occupancy_scale_enabled_flag) {	

asme_occupancy_scale_factor_x_minus1	ue(v)
asme_occupancy_scale_factor_y_minus1	ue(v)
}	
asme_patch_constant_depth_flag	u(1)
asme_patch_attribute_offset_enabled_flag	u(1)
if(asme_patch_attribute_offset_enabled_flag)	
asme_patch_attribute_offset_bit_depth_minus1	ue(v)
asme_max_entity_id	ue(v)
asme_inpaint_enabled_flag	u(1)
}	

8.3.9.4 Atlas frame parameter set MIV extension syntax

afps_miv_extension( ) {	**Descriptor**
if(!afps_lod_mode_enabled_flag) {	
afme_inpaint_lod_enabled_flag	u(1)
if(afme_inpaint_lod_enabled_flag) {	
afme_inpaint_lod_scale_x_minus1	ue(v)
afme_inpaint_lod_scale_y_idc	ue(v)
}	
}	
}	

8.3.9.5 Common atlas sequence parameter set MIV extension syntax

casps_miv_extension( ) {	**Descriptor**
casme_depth_low_quality_flag	u(1)
casme_depth_quantization_params_present_flag	u(1)
casme_vui_params_present_flag	u(1)
if(casme_vui_params_present_flag)	
vui_parameters( )	
}	

8.3.9.6 Common atlas frame

8.3.9.6.1 MIV extension syntax

caf_miv_extension( ) {	**Descriptor**
if(nal_unit_type == NAL_CAF_IDR) {	
miv_view_params_list( )	
} else {	
came_update_extrinsics_flag	u(1)
came_update_intrinsics_flag	u(1)
if(casme_depth_quantization_params_present_flag)	
came_update_depth_quantization_flag	u(1)
if(came_update_extrinsics_flag)	

miv_view_params_update_extrinsics( )	
if(came_update_intrinsics_flag)	
miv_view_params_update_intrinsics( )	
if(came_update_depth_quantization_flag)	
miv_view_params_update_depth_quantization( )	
}	
}	

8.3.9.6.2 MIV view parameters list syntax

miv_view_params_list( ) {	Descriptor
mvp_num_views_minus1	u(16)
mvp_explicit_view_id_flag	u(1)
if(mvp_explicit_view_id_flag)	
for(v = 0; v <= mvp_num_views_minus1; v++)	
mvp_view_id[v]	u(16)
for(v = 0; v <= mvp_num_views_minus1; v++) {	
camera_extrinsics(v)	
mvp_inpaint_flag[v]	u(1)
}	
mvp_intrinsic_params_equal_flag	u(1)
for(v = 0; v <= mvp_intrinsic_params_equal_flag ? 0 : mvp_num_views_minus1; v++)	
camera_intrinsics(v)	
if(casme_depth_quantization_params_present_flag) {	
mvp_depth_quantization_params_equal_flag	u(1)
for(v = 0; v <= mvp_depth_quantization_equal_flag ? 0 : mvp_num_views_minus1; v++)	
depth_quantization(v)	
}	
mvp_pruning_graph_params_present_flag	u(1)
if (mvp_pruning_graph_params_present_flag)	
for(v = 0; v <= mvp_num_views_minus1; v++)	
pruning_parents(v)	
}	

8.3.9.6.3 MIV view parameters update extrinsics syntax

miv_view_params_update_extrinsics( ) {	Descriptor
mvpue_num_view_updates_minus1	u(16)
for(i = 0; i <= mvpue_num_view_updates_minus1; i++) {	
mvpue_view_idx[i]	u(16)
camera_extrinsics(mvpue_view_idx[i])	
}	
}	

8.3.9.6.4 MIV view parameters update intrinsics syntax

miv_view_params_update_intrinsics( ) {	Descriptor
mvpui_num_view_updates_minus1	u(16)
for(i = 0; i <= mvpui_num_view_updates_minus1; i++) {	
mvpui_view_idx[i]	u(16)
camera_intrinsics(mvpui_view_idx[i])	
}	
}	

8.3.9.6.5 MIV view parameters update depth quantization syntax

miv_view_params_update_depth_quantization( ) {	Descriptor
mvpudq_num_view_updates_minus1	u(16)
for(i = 0; i <= mvpudq_num_view_updates_minus1; i++) {	
mvpudq_view_idx[i]	u(16)
depth_quantization(mvpudq_view_idx[i])	
}	
}	

8.3.9.6.6 Camera extrinsics syntax

camera_extrinsics(v) {	Descriptor
ce_view_pos_x[v]	fl(32)
ce_view_pos_y[v]	fl(32)
ce_view_pos_z[v]	fl(32)
ce_view_quat_x[v]	i(32)
ce_view_quat_y[v]	i(32)
ce_view_quat_z[v]	i(32)
}	

8.3.9.6.7 Camera intrinsics syntax

camera_intrinsics(v) {	Descriptor
ci_cam_type[v]	u(8)
ci_projection_plane_width_minus1[v]	u(16)
ci_projection_plane_height_minus1[v]	u(16)
if(ci_cam_type[v] == 0) { /* equirectangular */	
ci_erp_phi_min[v]	fl(32)
ci_erp_phi_max[v]	fl(32)
ci_erp_theta_min[v]	fl(32)
ci_erp_theta_max[v]	fl(32)
} else if(ci_cam_type[v] == 1) { /* perspective */	
ci_perspective_focal_hor[v]	fl(32)
ci_perspective_focal_ver[v]	fl(32)
ci_perspective_principal_point_hor[v]	fl(32)

ci_perspective_principal_point_ver[v]	fl(32)
} else if(ci_cam_type[v] == 2) { /* orthographic */	
ci_ortho_width[v]	fl(32)
ci_ortho_height[v]	fl(32)
}	
}	

8.3.9.6.8 Depth quantization syntax

depth_quantization(v) {	**Descriptor**
dq_quantization_law[v]	ue(v)
if(dq_quantization_law[v] == 0) {	
dq_norm_disp_low[v]	fl(32)
dq_norm_disp_high[v]	fl(32)
}	
dq_depth_occ_threshold_default[v]	ue(v)
}	

8.3.9.6.9 Pruning parents syntax

pruning_parents(v) {	**Descriptor**
pp_is_root_flag[v]	u(1)
if(!pp_is_root_flag[v]) {	
pp_num_parents_minus1[v]	u(v)
for(i = 0; i <= pp_num_parents_minus1[v]; i++)	
pp_parent_idx[v][i]	u(v)
}	
}	

8.3.9.7 Patch data unit MIV extension syntax

pdu_miv_extension(tileID, p) {	**Descriptor**
if(asme_max_entity_id > 0)	
pdu_entity_id[tileID][p]	u(v)
if(asme_depth_occ_threshold_flag)	
pdu_depth_occ_threshold[tileID][p]	u(v)
if(asme_patch_attribute_offset_enabled_flag)	
for(c = 0; c < 3; c++) {	
pdu_attribute_offset[tileID][p][c]	u(v)
if(asme_inpaint_enabled_flag)	
pdu_inpaint_flag[tileID][p]	u(1)
}	

8.4 Semantics

8.4.1 General semantics

The specifications in ISO/IEC 23090-5:—, subclause 8.4 apply with the following addition.

It is a requirement of bitstream conformance that asps_geometry_3d_bit_depth_minus1 and asps_geometry_2d_bit_depth_minus1 shall be equal to gi_geometry_3d_coordinates_bit_depth_minus1 and gi_geometry_2d_bit_depth_minus1, respectively.

8.4.2 V3C MIV extension semantics

8.4.2.1 V3C parameter set MIV extension semantics

vme_geometry_scale_enabled_flag equal to 1 specifies that the V3C sub-bitstream component corresponding to the geometry components, which are determined through either examining if vuh_unit_type is equal to V3C_GVD or through external means if the V3C unit header is unavailable, may have a different coded picture width and height than the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When vme_geometry_scale_enabled_flag is equal to 0, it is a requirement of bitstream conformance that the picture width and picture height of the V3C sub-bitstream component corresponding to the geometry components be equal to the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When not present, the value of vme_geometry_scale_enabled_flag is inferred to be equal to 0.

vme_embedded_occupancy_enabled_flag equal to 1 specifies that the occupancy information is present in the V3C sub-bitstream component corresponding to the geometry component, which is determined through either examining if vuh_unit_type is equal to V3C_GVD or through external means if the V3C unit header is unavailable. vme_embedded_occupancy_enabled_flag equal to 0 specifies that occupancy information is not present or present in the V3C sub-bitstream component corresponding to the occupancy component, which is determined through either examining if vuh_unit_type is equal to V3C_OVD or V3C_PVD or through external means if the V3C unit header is unavailable.

vme_occupancy_scale_enabled_flag equal to 1 specifies that the V3C sub-bitstream component corresponding to the occupancy components, which are determined through either examining if vuh_unit_type is equal to V3C_OVD or V3C_PVD or through external means if the V3C unit header is unavailable, may have a different coded picture width and height than the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When vme_occupancy_scale_enabled_flag is equal to 0, it is a requirement of bitstream conformance that the coded picture width and height of all the V3C sub-bitstream components corresponding to the occupancy components be equal to the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When not present, the value of vme_occupancy_scale_enabled_flag is inferred to be equal to 0.

8.4.2.2 Group mapping semantics

gm_group_count specifies the total number of supported groups in the current bitstream. The value of gm_group_count shall be in the range of 0 to 15, inclusive.

gm_group_id[k] specifies the ID of the group for the atlas with ID equal to k. The number of bits used to represent gm_group_id[k] is equal to Ceil(Log2(gm_group_count)).

Atlases indicated by the same group ID should be used together for reconstruction. The constant MaxAtlasCount is set equal to 64. The 1D array *AtlasGroupId* of size MaxAtlasCount is derived as follows:

```
if( gm_group_count > 0 ) {
   for( k = 0; k <= vps_atlas_count_minus1; k++ ) {
      j = vps_atlas_id[ k ]
      AtlasGroupId[ j ] = gm_group_id[ j ]
   }
}
```

8.4.2.3 Atlas sequence parameter set MIV extension semantics

The variable *aspsAtlasID* is set equal to vuh_atlas_id of the ASPS RBSP or determined through external means if the V3C unit header is unavailable.

The 1D arrays *AspsFrameWidth* and *AspsFrameHeight* of size MaxAtlasCount are derived as follows:

```
AspsFrameWidth[ aspsAtlasID ] = asps_frame_width
AspsFrameHeight[ aspsAtlasID ] = asps_frame_height
```

asme_ancillary_atlas_flag equal to 1 indicates that the patches of the atlas are not intended to be used for view rendering. asme_ancillary_atlas_flag equal to 0 indicates that the patches of the atlas are intended to be used for view rendering. When not present, the value of asme_ancillary_atlas_flag is inferred to be equal to 0.

NOTE Patches belonging to atlases with this flag set to 1 may describe the unseen rear sides of objects, and be useful in application-specific collision detection, interactions, or scene relighting.

asme_embedded_occupancy_enabled_flag equal to 1 specifies that the occupancy information is present in the V3C sub-bitstream component corresponding to the geometry component, which is determined through either examining if vuh_unit_type is equal to V3C_GVD or through external means if the V3C unit header is unavailable. asme_embedded_occupancy_enabled_flag equal to 0 specifies that occupancy information is not present or present in the V3C sub-bitstream component corresponding to the occupancy component, which is determined through either examining if vuh_unit_type is equal to V3C_OVD or V3C_PVD or through external means if the V3C unit header is unavailable.

It is a requirement of bitstream conformance that the value of asme_embedded_occupancy_enabled_flag shall be equal to the value of vme_embedded_occupancy_enabled_flag when the V3C VPS is available.

asme_depth_occ_threshold_flag equal to 1 specifies that the pdu_depth_occ_threshold syntax element is present in the pdu_miv_extension( ) syntax structure. asme_depth_occ_threshold_flag equal to 0 specifies that the pdu_depth_occ_threshold syntax element is not present in the pdu_miv_extension( ) syntax structure. When not present, the value of asme_depth_occ_threshold_flag is inferred to be equal to 0.

The 1D array *AsmeDepthOccThresholdFlag* of size MaxAtlasCount is derived as follows:

```
AsmeDepthOccThresholdFlag[ aspsAtlasID ] = asme_depth_occ_threshold_flag
```

asme_geometry_scale_enabled_flag equal to 1 specifies that the V3C sub-bitstream component corresponding to the geometry components, which are determined through either examining if vuh_unit_type is equal to V3C_GVD or through external means if the V3C unit header is unavailable, may have a different coded picture width and height than the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When asme_geometry_scale_enabled_flag is equal to 0, it is a requirement of bitstream conformance that the picture width and picture height of the V3C sub-bitstream component corresponding to the geometry components be equal to the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When not present, the value of asme_geometry_scale_enabled_flag is inferred to be equal to 0.

It is a requirement of bitstream conformance that the value of asme_geometry_scale_enabled_flag shall be equal to the value of vme_geometry_scale_enabled_flag when the V3C VPS is available.

asme_geometry_scale_factor_x_minus1 plus 1 specifies a scale factor of the frame width of the geometry video data of the atlas in relation to the nominal atlas width. When not present, the value of asme_geometry_scale_factor_x_minus1 is inferred to be equal to 0.

The 1D arrays *AsmeGeometryScaleFactorX* and *AsmeGeometryFrameWidth* of size MaxAtlasCount are derived as follows:

```
AsmeGeometryScaleFactorX[ aspsAtlasID ] = asme_geometry_scale_factor_x_minus1 + 1

AsmeGeometryFrameWidth[ aspsAtlasID ] =
```

```
    AspsFrameWidth[ aspsAtlasID ] / AsmeGeometryScaleFactorX[ aspsAtlasID ]
```

It is a requirement of bitstream conformance that *AspsFrameWidth*[*aspsAtlasID*] % *AsmeGeometryScaleFactorX*[*aspsAtlasID*] shall be equal to 0.

asme_geometry_scale_factor_y_minus1 plus 1 specifies a scale factor of the frame height of the geometry video data of the atlas in relation to the nominal atlas height. When not present, the value of asme_geometry_ scale_factor_y_minus1 is inferred to be equal to 0.

The 1D arrays *AsmeGeometryScaleFactorY* and *AsmeGeometryFrameHeight* of size MaxAtlasCount are derived as follows:

```
AsmeGeometryScaleFactorY[ aspsAtlasID ] = asme_geometry_scale_factor_y_minus1 + 1

AsmeGeometryFrameHeight[ aspsAtlasID ]=
    AspsFrameHeight[ aspsAtlasID ] / AsmeGeometryScaleFactorY[ aspsAtlasID ]
```

It is a requirement of bitstream conformance that *AspsFrameHeight*[*aspsAtlasID*] % *AsmeGeometryScaleFactorY*[*aspsAtlasID*] shall be equal to 0.

asme_occupancy_scale_enabled_flag equal to 1 specifies that the V3C sub-bitstream component corresponding to the occupancy components, which are determined through either examining if vuh_unit_type is equal to V3C_OVD or V3C_PVD or through external means if the V3C unit header is unavailable, may have a different coded picture width and height than the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When asme_occupancy_scale_ enabled_flag is equal to 0, it is a requirement of bitstream conformance that the coded picture width and height of all the V3C sub-bitstream components corresponding to the occupancy components be equal to the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When not present, the value of asme_occupancy_scale_enabled_flag is inferred to be equal to 0.

asme_occupancy_scale_factor_x_minus1 plus 1 specifies a scale factor of the frame width of the occupancy video data of the atlas in relation to the nominal atlas width. When not present, the value of asme_occupancy_ scale_factor_x_minus1 is inferred to be equal to 0.

The 1D arrays *AsmeOccupancyScaleFactorX* and *AsmeOccupancyFrameWidth* of size MaxAtlasCount are derived as follows:

```
AsmeOccupancyScaleFactorX[ aspsAtlasID ] = asme_occupancy_ scale_factor_x_minus1 + 1

AsmeOccupancyFrameWidth[ aspsAtlasID ] =
    AspsFrameWidth[ aspsAtlasID ] / AsmeOccupancyScaleFactorX[ aspsAtlasID ]
```

asme_occupancy_scale_factor_y_minus1 plus 1 specifies a scale factor of the frame height of the occupancy video data of the atlas in relation to the nominal atlas height. When not present, the value of asme_occupancy_ scale_factor_y_minus1 is inferred to be equal to 0.

The 1D arrays *AsmeOccupancyScaleFactorY* and *AsmeOccupancyFrameHeight* of size MaxAtlasCount are derived as follows:

```
AsmeOccupancyScaleFactorY[ aspsAtlasID ] = asme_occupancy_scale_factor_y_minus1 + 1

AsmeOccupancyFrameHeight[ aspsAtlasID ] =
    AspsFrameHeight[ aspsAtlasID ] / AsmeOccupancyScaleFactorY[ aspsAtlasID ]
```

asme_patch_constant_depth_flag equal to 1 indicates that the recommended depth for a patch be derived based on patch data unit parameters instead of the geometry video data. asme_patch_constant_ depth_flag equal to 0 specifies that when vps_geometry_video_present_flag[*aspsAtlasID*] is equal to 0, the depth is determined by external means.

The 1D array *AsmePatchConstantDepthFlag* of size MaxAtlasCount is derived as follows:

```
AsmePatchConstantDepthFlag[ aspsAtlasID ] = asme_patch_constant_depth_flag
```

asme_patch_attribute_offset_enabled_flag equal to 1 indicates that the asme_patch_attribute_offset_bit_depth_minus1 syntax element is present in the syntax structure. asme_patch_attribute_offset_enabled_flag equal to 0 indicates that the asme_patch_attribute_offset_bit_depth_minus1 syntax element is not present in the syntax structure.

asme_patch_attribute_offset_bit_depth_minus1 plus 1 specifies the number of bits used to represent the pdu_attribute_offset[tileID][p][c] syntax element. asme_patch_attribute_offset_bit_depth_minus1 shall be in the range of 0 to ai_attribute_2d_bit_depth_minus1, inclusive.

asme_max_entity_id specifies the maximum value that can be indicated for the patch entity ID syntax element, pdu_entity_id[i][j], for a patch with index j in a tile with tile ID equal to i. When not present, the value of asme_max_entity_id is inferred to be equal to 0.

asme_inpaint_enabled_flag equal to 1 specifies that the pdu_inpaint_flag[tileID][p] syntax elements are present in the pdu_miv_extension( ) syntax structure. asme_inpaint_enabled_flag equal to 0 specifies that the pdu_inpaint_flag[tileID][p] syntax elements are not present in the pdu_miv_extension( ) syntax structure. When not present, the value of asme_inpaint_enabled_flag is inferred to be equal to 0.

8.4.2.4 Atlas frame parameter set MIV extension semantics

afme_inpaint_lod_enabled_flag equal to 1 specifies that the inpaint LOD parameters are present. If afme_inpaint_lod_enabled_flag is equal to 0, no inpaint LOD parameters are present. When not present, the value of afme_inpaint_lod_enabled_flag is inferred to be equal to 0.

afme_inpaint_lod_scale_x_minus1 specifies the inpaint LOD scaling factor to be applied to the local x coordinate of a point in any patch with index p of the current atlas tile, with tile ID equal to tileID, and with pdu_inpaint_flag[tileID][p] equal to 1, prior to its addition to the patch coordinate *TilePatch3dOffsetU*[tileID][p]. When not present, the value of afme_inpaint_lod_scale_x_minus1 is inferred to be equal to 0.

afme_inpaint_lod_scale_y_idc indicates the LOD scaling factor to be applied to the local y coordinate of a point in any patch with index p of the current atlas tile, with tile ID equal to tileID, and with pdu_inpaint_flag[tileID][p] equal to 1, prior to its addition to the patch coordinate *TilePatch3dOffsetV*[tileID][p]. When not present, the value of afme_inpaint_lod_scale_y_idc[tileID][p] is inferred to be equal to 0.

8.4.2.5 Common atlas sequence parameter set MIV extension semantics

casme_depth_low_quality_flag equal to 1 indicates that the depth fidelity confidence in geometry video sub-bitstreams is low. casme_depth_low_quality_flag equal to 0 indicates that the depth fidelity confidence is unknown. When not present, the value of casme_depth_low_quality_flag is inferred to be equal to 0.

NOTE Low depth fidelity indicates inconsistency in depth values between views. The encoder may set this flag to 1 for depth estimated on a natural content with an image-based depth estimation method. The encoder may set this flag to 0 for depth derived from a 3D model on a computer generated content.

casme_depth_quantization_params_present_flag equal to 1 indicates that the depth quantization parameters are present in the syntax structure. casme_depth_quantization_params_present_flag equal to 0 indicates that the depth quantization parameters are not present in the syntax structure. When not present, the value of casme_depth_quantization_params_present_flag is inferred to be equal to 1.

casme_vui_params_present_flag equal to 1 indicates that the vui_parameters( ) syntax structure is present in this syntax structure. casme_vui_params_present_flag equal to 0 indicates that the vui_parameters( ) syntax structure is not present in this syntax structure. It is a requirement of bitstream conformance that the value of casme_vui_params_present_flag shall be equal to 0 for all non-IRAP access units.

8.4.2.6 Common atlas frame MIV extension semantics

came_update_extrinsics_flag equal to 1 indicates that the miv_view_params_update_extrinsics( ) syntax structure is present in this syntax structure. came_update_extrinsics_flag equal to 0 indicates that the miv_view_params_update_extrinsics( ) syntax structure is not present in this syntax structure.

came_update_intrinsics_flag equal to 1 indicates that the miv_view_params_update_intrinsics( ) syntax structure is present in this syntax structure. came_update_intrinsics_flag equal to 0 indicates that the miv_view_params_update_intrinsics( ) syntax structure is not present in this syntax structure.

came_update_depth_quantization_flag equal to 1 indicates that the miv_view_params_update_depth_quantization( ) syntax structure is present in this syntax structure. came_update_depth_quantization_flag equal to 0 indicates that the miv_view_params_update_depth_quantization( ) syntax structure is not present in this syntax structure. When not present, the value of came_update_depth_quantization_flag is inferred to be equal to 0.

8.4.2.6.1 MIV view parameters list semantics

mvp_num_views_minus1 plus 1 specifies the maximum number of views in an MIV view parameters list representing a volumetric frame.

mvp_explicit_view_id_flag equal to 1 specifies that the syntax element mvp_view_id[v] is present in the miv_view_params_list( ) syntax structure. mvp_explicit_view_id_flag equal to 0 specifies that the syntax element mvp_view_id[v] is not present in the miv_view_params_list( ) syntax structure.

mvp_view_id[v] specifies the view ID of the view with index v. The value of mvp_view_id[v] shall be in the range of 0 to 65 535, inclusive. It is a requirement of bitstream conformance to this version of this document that the value of mvp_view_id[j] shall not be equal to mvp_view_id[k] for all j != k. When not present, the value of mvp_view_id[v] is inferred to be equal to v.

mvp_inpaint_flag[v] equal to 1 specifies that the view with index v is an inpaint view.

NOTE 1 A renderer can skip inpaint views.

NOTE 2 A renderer can use inpaint views to render viewport regions that would otherwise require inpainting.

mvp_intrinsic_params_equal_flag equal to 1 specifies that the intrinsic parameters are signalled only for the view with index 0 and each view with index in range of 1 to mvp_num_views_minus1, inclusive, has the same intrinsic parameters as the view with index 0. mvp_intrinsic_params_equal_flag equal to 0 specifies that the intrinsic parameters are explicitly signalled for each view with index in range of 0 to mvp_num_views_minus1, inclusive.

NOTE When mvp_intrinsic_params_equal_flag == 1, it is still allowed to send intrinsics updates for specific cameras.

mvp_depth_quantization_params_equal_flag equal to 1 specifies that the depth quantization parameters are signalled only for the view with index 0 and each view with index in range of 1 to mvp_num_views_minus1, inclusive, has the same depth quantization parameters as the view with index 0. mvp_depth_quantization_params_equal_flag equal to 0 specifies that the depth quantization parameters are explicitly signalled for each view with index in range of 0 to mvp_num_views_minus1, inclusive.

NOTE When mvp_depth_quantization_params_equal_flag == 1, it is still allowed to send depth quantization updates for specific cameras.

mvp_pruning_graph_params_present_flag equal to 1 specifies that the pruning_parents( ) syntax structure is present in the miv_view_params_list( ) syntax structure. mvp_pruning_graph_params_present_flag equal to 0 specifies that the pruning_parents( ) syntax structure is not present in the miv_view_params_list( ) syntax structure.

8.4.2.6.2 MIV view parameters update extrinsics semantics

mvpue_num_view_updates_minus1 plus 1 specifies the number of camera_extrinsics(v) syntax structures that are present within this syntax structure. The value of mvpue_num_view_updates_minus1 shall be in the range of 0 to mvp_num_views_minus1**,** inclusive**.**

mvpue_view_idx[i] specifies the view index for which updated camera extrinsic parameters will be signalled. The value of mvpue_view_idx[i] shall be in the range of 0 to mvp_num_views_minus1, inclusive. It is a requirement of bitstream conformance to this version of this document that the value of mvpue_view_idx[j] shall not be equal to mvpue_view_idx[k] for all j != k.

8.4.2.6.3 MIV view parameters update intrinsics semantics

mvpui_num_view_updates_minus1 plus 1 specifies the number of camera_intrinsics(v) syntax structures that are present within this syntax structure. The value of mvpue_num_view_updates_minus1 shall be in the range of 0 to mvp_num_views_minus1**.**

mvpui_view_idx[i] specifies the view index for which updated camera intrinsic parameters will be signalled. The value of mvpui_view_idx[i] shall be in the range of 0 to mvp_num_views_minus1, inclusive. It is a requirement of bitstream conformance to this version of this document that the value of mvpui_view_idx[j] shall not be equal to mvpui_view_idx[k] for all j != k.

8.4.2.6.4 MIV view parameters update depth quantization semantics

mvpudq_num_view_updates_minus1 plus 1 specifies the number of depth_quantization(v) syntax structures that are present within this syntax structure. The value of mvpudq_num_view_updates_minus1 shall be in the range of 0 to mvp_num_views_minus1, inclusive**.**

mvpudq_view_idx[i] specifies the view index for which updated depth quantization parameters will be signalled. The value of mvpudq_view_idx[i] shall be in the range of 0 to mvp_num_views_minus1, inclusive. It is a requirement of bitstream conformance to this version of this document that the value of mvpudq_view_idx[j] shall not be equal to mvpudq_view_idx[k] for all j != k.

8.4.2.6.5 Camera extrinsics semantics

ce_view_pos_x[v] specifies in scene units the x-coordinate of the location of the view with view index equal to v.

ce_view_pos_y[v] specifies in scene units the y-coordinate of the location of the view with view index equal to v.

ce_view_pos_z[v] specifies in scene units the z-coordinate of the location of the view with view index equal to v.

ce_view_quat_x[v] specifies the x component, *qX*[v], for the rotation of the view with view index equal to v using the quaternion representation. The value of ce_view_quat_x[v] shall be in the range of -2^{30} to 2^{30}, inclusive. When ce_view_quat_x[v] is not present, its value shall be inferred to be equal to 0. The value of *qX*[v] is computed as follows:

$$qX[\,v\,] = ce\_view\_quat\_x[\,v\,] \div 2^{30} \tag{1}$$

ce_view_quat_y[v] specifies the y component, *qY*[v], for the rotation of the view with view index equal to v using the quaternion representation. The value of ce_view_quat_y[v] shall be in the range of

-2^{30} to 2^{30}, inclusive. When ce_view_quat_y[v] is not present, its value shall be inferred to be equal to 0. The value of qY[v] is computed as follows:

$$qY[\,v\,] = \text{ce\_view\_quat\_y}[\,v\,] \div 2^{30} \tag{2}$$

ce_view_quat_z[v] specifies the z component, qZ[v], for the rotation of the view with view index equal to v using the quaternion representation. The value of ce_view_quat_z[v] shall be in the range of -2^{30} to 2^{30}, inclusive. When ce_view_quat_z[v] is not present, its value shall be inferred to be equal to 0. The value of qZ[v] is computed as follows:

$$qZ[\,v\,] = \text{ce\_view\_quat\_z}[\,v\,] \div 2^{30} \tag{3}$$

The fourth component, qW[v], of the quaternion is calculated as follows:

$$qW[\,v\,] = \text{Sqrt}(\,\text{Max}(\,0,\,1 - (\,qX[\,v\,]^2 + qY[\,v\,]^2 + qZ[\,v\,]^2\,)\,)\,) \tag{4}$$

8.4.2.6.6 Camera intrinsics semantics

ci_cam_type[v] specifies the projection method of the view with view index equal to v. ci_cam_type[v] equal to 0 specifies ERP projection. ci_cam_type[v] equal to 1 specifies a perspective projection. ci_cam_type[v] equal to 2 specifies an orthographic projection. ci_cam_type values in range 3 to 127 are reserved for future use by ISO/IEC. ci_cam_type values in range 128 to 255 are unspecified (available for specification by other standards).

ci_projection_plane_width_minus1[v] plus 1 and **ci_projection_plane_height_minus1**[v] plus 1 specify the horizontal and vertical resolutions of the camera projection planes, respectively, expressed in coded luma samples.

ci_erp_phi_min[v] and **ci_erp_phi_max**[v] specify the longitude range (minimum and maximum values) for an ERP projection, as floating-point in units of degrees. ci_erp_phi_min[v] and ci_erp_phi_max[v] shall be in the range –180 to 180.

ci_erp_theta_min[v] and **ci_erp_theta_max**[v] specify the latitude range (minimum and maximum values) for an ERP projection, as floating-point in units of degrees. ci_erp_theta_min[v] and ci_erp_theta_max[v] shall be in the range –90 to 90.

ci_perspective_focal_hor[v] and **ci_perspective_focal_ver**[v] are floating-point values that specify in luma sample position units the horizontal and vertical components, respectively, of the focal of a perspective projection.

ci_perspective_principal_point_hor[v] and **ci_perspective_principal_point_ver**[v] are floating-point values that specify in luma sample position units the horizontal and vertical coordinates, respectively, of the principal point of a perspective projection (intersection of optical axis with image plane).

ci_ortho_width[v] and **ci_ortho_height**[v] are positive floating-point values that specify in scene units (e.g. meters) the horizontal and vertical dimensions of the captured part of the volumetric frame.

8.4.2.6.7 Depth quantization semantics

dq_quantization_law[v] specifies the type of depth quantization method of the view with view index equal to v. dq_quantization_law[v] equal to 0 specifies a uniform quantization of the inverse of depth values. Even values of dq_quantization_law[v] greater than 0 are reserved for future use by ISO/IEC. Odd values of dq_quantization_law[v] are outside the scope of this document.

dq_norm_disp_low[v] and **dq_norm_disp_high**[v] specify the normalized disparity of the lowest and highest signalled geometry values, respectively, in scene units^{-1} (e.g. meters^{-1}) of the view with view index equal to v.

dq_depth_occ_threshold_default[v] specifies the default occupancy threshold used in the occupancy value extraction process of the view with view index equal to v. When not present the value of dq_depth_occ_threshold_default[v] is inferred to be equal to 0.

8.4.2.6.8 Pruning parents semantics

pp_is_root_flag[v] equal to 1 specifies that the view with view index equal to v has no parent in the pruning graph at the encoder stage. pp_is_root_flag[v] equal to 0 indicates that the view with view index equal to v has at least one parent in the pruning graph at the encoder stage.

pp_num_parents_minus1[v] plus 1 specifies the number of parents of the view with view index equal to v in the pruning graph at the encoder stage. pp_num_parents_minus1[v] shall be in the range 0 to mvp_num_views_minus1, exclusive. The number of bits used to represent the syntax element pp_num_parents_minus1[v] is Ceil(Log2(mvp_num_views_minus1)).

pp_parent_idx[v][i] specifies the index of the i-th parent view for the view with view index equal to v in the pruning graph at the encoder stage. pp_parent_idx[v][i] shall not be equal to v. The number of bits used to represent the syntax element pp_parent_idx[v] is Ceil(Log2(mvp_num_views_minus1 + 1)).

8.4.2.7 Patch data unit MIV extension semantics

pdu_entity_id[tileID][p] specifies the entity ID of the patch with index equal to p, in a tile with ID equal to tileID. The number of bits used for the representation of pdu_entity_id[tileID][p] is Ceil(Log2(asme_max_entity_id + 1)). The value of pdu_entity_id[tileID][p] shall be in range of 0 to asme_max_entity_id, inclusive. When not present, the value of pdu_entity_id[tileID][p] is inferred to be equal to 0.

pdu_depth_occ_threshold[tileID][p] specifies the threshold below which the occupancy value is defined to be unoccupied for the patch with index equal to p, in the tile with ID equal to tileID. The number of bits used to represent pdu_depth_occ_threshold[tileID][p] is equal to asps_geometry_2d_bit_depth_minus1 + 1. When not present, the value of pdu_depth_occ_threshold[tileID][p] is inferred to be equal to dq_depth_occ_threshold_default[pdu_projection_id[tileID][p]].

NOTE pdu_projection_id[tileID][p] corresponds to view ID of the patch with index equal to p, in the tile with ID equal to tileID.

pdu_attribute_offset[tileID][p][c] specifies an offset applied to the c-th component sample values of the attribute for the patch with index equal to p, in the tile with ID equal to tileID. The number of bits used to represent pdu_attribute_offset[tileID][p][c] is equal to asme_patch_attribute_offset_bit_depth_minus1 + 1. When not present, the value of pdu_attribute_offset[tileID][p][c] is inferred to be equal to 0.

pdu_inpaint_flag[tileID][p] equal to 1 specifies that the patch is an inpaint patch. When pdu_inpaint_flag[tileID][p] is equal to 0, then the patch is not an inpaint patch. When not present, the value of pdu_inpaint_flag[tileID][p] is inferred to be equal to 0.

NOTE A renderer can use inpaint patches to render viewport regions that would otherwise require inpainting.

8.4.3 Order of V3C units and association to coded information

The specifications in ISO/IEC 23090-5:—, subclause 8.4.2.5 apply with the following additions.

A bitstream consists of a series of coded MIV sequences. A VPS with vps_v3c_parameter_set_id equal to vuh_v3c_parameter_set_id shall be available within the coded MIV sequence or provided via external means before it is referenced in a V3C unit.

A bitstream contains one or more sub-bitstreams. A sub-bitstream contains the V3C units with the same V3C unit header. Within a sub-bitstream, the V3C units are in decoding order. All sub-bitstream

composition units within the same V3C composition unit shall have the same value of output order index.

9 Decoding process

9.1 General decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.1 apply.

9.2 Atlas data decoding process

9.2.1 General atlas data decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.2.1 apply.

9.2.2 Decoding process for a coded atlas frame

The specifications in ISO/IEC 23090-5:—, subclause 9.2.2 apply.

9.2.3 Atlas NAL unit decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.2.3 apply.

9.2.4 Atlas tile header decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.2.4 apply.

9.2.5 Decoding process for patch data units

9.2.5.1 General decoding process for patch data units

The specifications in ISO/IEC 23090-5:—, subclause 9.2.5.1 apply with the following modifications and additions:

TilePatchEntityID[tileID][p] specifies the patch entity ID for the current patch with patch index p, of the current tile with tile ID equal to tileID.

TilePatchDepthOccThreshold[tileID][p] specifies the threshold below which the occupancy value is defined to be unoccupied for the current patch with patch index p, of the current tile with tile ID equal to tileID.

TilePatchAttributeOffset[tileID][p][c] specifies an offset applied to the c-th component sample values of the attribute for the current patch with patch index p, of the current tile with tile ID equal to tileID, where c is in the range if 0 to 2, inclusive.

These additional variables are initially set as follows:

```
TilePatchEntityID[ tileID ][ p ] = 0
TilePatchDepthOccThreshold[ tileID ][ p ] = 0
TilePatchAttributeOffset[ tileID ][ p ][ c ] = 0
```

9.2.5.2 Decoding process for patch data units coded in intra mode

In addition to the variables derived in ISO/IEC 23090-5:—, subclause 9.2.5.2 the variables *TilePatchEntityID*[tileID][p], *TilePatchDepthOccThreshold*[tileID][p], *TilePatchAttributeOffset*[tileID][p][c], for patch with index p in the atlas tile with ID tileID, are derived as follows:

If asme_max_entity_id is not equal to 0, the following applies:

```
TilePatchEntityID[ tileID ][ p ] = pdu_entity_id[ tileID ][ p ]
```

If asme_depth_occ_threshold_flag is equal to 1, the following applies:

```
TilePatchDepthOccThreshold[ tileID ][ p ] = pdu_depth_occ_threshold[ tileID ][ p ]
```

If asme_patch_attribute_offset_enabled_flag is equal to 1, the following applies:

```
for( c = 0; c < 3; c++ )
   TilePatchAttributeOffset[ tileID ][ p ][ c ] =
      ( pdu_attribute_offset[ tileId ][ p ][ c ] <<
         ( ai_attribute_2d_bit_depth_minus1 -
            asme_patch_attribute_offset_bit_depth_minus1 ) ) -
         ( 1 << ai_attribute_2d_bit_depth_minus1 )
```

The derivation in ISO/IEC 23090-5:—, subclause 9.2.5.2 of *TilePatchLoDScaleX*[tileID][p] and *TilePatchLoDScaleY*[tileID][p] is extended as follows:

```
if( pdu_lod_enabled_flag[ tileID ][ p ] ) {
    scaleX = pdu_lod_scale_x_minus1[ tileID ][ p ] + 1
    scaleY = pdu_lod_scale_y_idc[ tileID ][ p ] + ( scaleX == 1 ? 2 : 1 )
} else if( pdu_inpaint_flag[ tileID ][ p ] ) {
    scaleX = afme_inpaint_lod_scale_x_minus1 + 1
    scaleY = afme_inpaint_lod_scale_y_idc + ( scaleX == 1 ? 2 : 1 )
} else {
    scaleX = 1
    scaleY = 1
}
TilePatchLoDScaleX[ tileID ][ p ] = scaleX
TilePatchLoDScaleY[ tileID ][ p ] = scaleY
```

9.2.5.3 Decoding process for patch data units coded in skip prediction mode

The specifications in ISO/IEC 23090-5:—, subclause 9.2.5.3 apply.

9.2.5.4 Decoding process for patch data units coded in merge prediction mode

The specifications in ISO/IEC 23090-5:—, subclause 9.2.5.4 apply.

9.2.5.5 Decoding process for patch data units coded in inter prediction mode

The specifications in ISO/IEC 23090-5:—, subclause 9.2.5.5 apply.

9.2.5.6 Decoding process for patch data units coded in RAW mode

The specifications in ISO/IEC 23090-5:—, subclause 9.2.5.6 apply.

9.2.5.7 Decoding process for patch data units coded in EOM mode

The specifications in ISO/IEC 23090-5:—, subclause 9.2.5.7 apply.

9.2.6 Decoding process of the block to patch map

The specifications in ISO/IEC 23090-5:—, subclause 9.2.6 apply.

9.2.7 Conversion of tile level patch information to atlas level patch information

9.2.7.1 General

The specifications in ISO/IEC 23090-5:—, subclause 9.2.7.1 apply.

9.2.7.2 Conversion of tile level blockToPatch information to atlas level blockToPatch information

The specifications in ISO/IEC 23090-5:—, subclause 9.2.7.2 apply.

9.2.7.3 Conversion of tile level patch information to atlas level patch information

9.2.7.3.1 General

The specifications in ISO/IEC 23090-5:—, subclause 9.2.7.3.1 apply.

9.2.7.3.2 Process of copying common patch parameters from a tile to an atlas representation

The specifications in ISO/IEC 23090-5:—, subclause 9.2.7.3.2 apply.

9.2.7.3.3 Process of copying application specific patch parameters from a tile to an atlas representation

The specifications in ISO/IEC 23090-5:—, subclause 9.2.7.3.3 apply with the following modifications and additions:

The process ApplicationTilePatchParamsToAtlas(atlasPatchIdx, tileID, p, offsetPatch) is amended to add the following at the end of the process:

```
AtlasPatchViewIndex[ atlasPatchIdx ] =
   ViewIDToIndex[ AtlasPatchProjectionID[ patchIdx ] ]

if( AtlasPatchType[ atlasPatchIdx ] == PROJECTED ) {
   AtlasPatchEntityID[ atlasPatchIdx ] = TilePatchEntityID[ tileID ][ p ]
     AtlasPatchDepthOccThreshold[ atlasPatchIdx ] =
        TilePatchDepthOccThreshold[ tileID ][ p ]
     for( c = 0; c < 3; c++ ) {
        AtlasPatchAttributeOffset[ atlasPatchIdx ][ c ] =
           TilePatchAttributeOffset[ tileID ][ p ][ c ]
      }
   }
}
```

9.3 Occupancy video decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.3 apply.

9.4 Geometry video decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.4 apply.

9.5 Attribute video decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.5 apply.

9.6 Packed video decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.6 apply.

9.7 Common atlas data decoding process

9.7.1 General common atlas data decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.7.1 apply with following additions.

Outputs of this process are:

— for each decoded common atlas frame, the corresponding upper-case variables from subclauses 9.7.5.1

9.7.2 Decoding process for a coded common atlas frame

The decoding processes specified in this subclause apply to each coded common atlas frame with nal_layer_id equal to 0, referred to as the current common atlas frame.

The decoding process for the current atlas frame takes as inputs the syntax elements and upper-case variables from Clause 8.

The decoding process operates as follows for the current common atlas frame:

1) The decoding of common atlas NAL units is specified in subclause 9.7.3.

2) Variables and functions relating to the common atlas frame order count are derived as specified in subclause 9.7.4.

3) The processes in subclause 9.7.5 specify the common atlas frame decoding processes:

   — Initializing of view information is specified in subclause 9.7.5.2.

   — Updating of extrinsic information is specified in subclause 9.7.5.3.

   — Updating of intrinsic information is specified in subclause 9.7.5.4.

   — Updating of depth quantization information is specified in subclause 9.7.5.5.

9.7.3 Common atlas NAL unit decoding process

The specifications in ISO/IEC 23090-5:—, subclause 9.7.3 apply.

9.7.4 Common atlas frame order count derivation process

The specifications in ISO/IEC 23090-5:—, subclause 9.7.4 apply.

9.7.5 Common atlas frame MIV extension decoding process

9.7.5.1 General decoding process for view parameters

Outputs of this process are several parameters associated with the current common atlas frame.

The following variables are derived:

NumViews indicates the number of views that are present in CVS.

The variable *ViewIDToIndex*[v] provides a mapping of the ID associated with each view and the index of how each view was specified in the MIV view parameters list syntax.

ViewInpaintFlag[v] specifies if the view with view index equal to v is an inpaint view.

ViewPosX[v] specifies in scene units the x-coordinate of the location of the view with view index equal to v.

ViewPosY[v] specifies in scene units the y-coordinate of the location of the view with view index equal to v.

ViewPosZ[v] specifies in scene units the z-coordinate of the location of the view with view index equal to v.

ViewQuatX[v] specifies the x component for the rotation of the view with view index equal to v using the quaternion representation.

ViewQuatY[v] specifies the y component for the rotation of the view with view index equal to v using the quaternion representation.

ViewQuatZ[v] specifies the z component for the rotation of the view with view index equal to v using the quaternion representation.

ViewQuatW[v] specifies the w component for the rotation of the view with view index equal to v using the quaternion representation.

ViewType[v] specifies the projection method of the view with view index equal to v.

ViewProjectionPlaneWidth[v] specifies the horizontal resolutions of projection plane, expressed in coded luma samples, of the view with view index equal to v.

ViewProjectionPlaneHeight[v] specifies the vertical resolutions of the projection plane, expressed in coded luma samples, of the view with view index equal to v.

ViewErpPhiMin[v] specifies the minimum longitude range for an ERP projection, in units of degrees, of the view with view index equal to v. *ViewErpPhiMin*[v] shall be in the range –180 to 180.

ViewErpPhiMax[v] specifies the maximum longitude range for an ERP projection, in units of degrees, of the view with view index equal to v. *ViewErpPhiMax*[v] shall be in the range –180 to 180.

ViewErpThetaMin[v] specifies the minimum latitude range for an ERP projection, in units of degrees, of the view with view index equal to v. *ViewErpThetaMin*[v] shall be in the range –90 to 90.

ViewErpThetaMax[v] specifies the maximum latitude range for an ERP projection, in units of degrees, of the view with view index equal to v. *ViewErpThetaMax*[v] shall be in the range –90 to 90.

ViewPerspectiveFocalHor[v] specifies in luma sample position units the horizontal components of the focal of a perspective projection of the view with view index equal to v.

ViewPerspectiveFocalVer[v] specifies in luma sample position units the vertical components of the focal of a perspective projection of the view with view index equal to v.

ViewPerspectivePrincipalPointHor[v] specifies in luma sample positions the horizontal coordinates of the principal point of a perspective projection of the view with view index equal to v.

ViewPerspectivePrincipalPointVer[v] specifies in luma sample positions the vertical coordinates of the principal point of a perspective projection of the view with view index equal to v.

ViewOrthoWidth[v] specifies in scene units the horizontal dimensions of the captured part of the volumetric frame by the view with view index equal to v.

ViewOrthoHeight[v] specifies in scene units the vertical dimensions of the captured part of the volumetric frame by the view with view index equal to v.

ViewQuantizationLaw[v] specifies the type of depth quantization method of the view with view index equal to v.

ViewNormDispLow[v] specifies the normalized disparity of the lowest signalled geometry value of the view with view index equal to v.

ViewNormDispHigh[v] specifies the normalized disparity of the highest signalled geometry value of the view with view index equal to v.

ViewOccThreshold[v] specifies the default occupancy threshold used in the occupancy value extraction process for the view with view index equal to v.

ViewRoot[v] specifies whether or not the view with view index equal to v has a parent in the pruning graph at the encoder stage.

ViewNumParents[v] specifies the number of parents of the view with view index equal to v in the pruning graph at the encoder stage. *ViewNumParents*[v] shall be in the range 0 to *NumViews* – 1.

ViewParentIdx[v][i] specifies the index of the i-th parent view for the view with view index equal to v in the pruning graph at the encoder stage. *ViewParentIdx*[v][i] shall not be equal to v.

At the start of each CVS, these arrays are initially set as follows:

```
NumViews = 0
for( v = 0; v <= 65 535; v++ ) {
   ViewInpaintFlag[ v ] = 0
   ViewPosX[ v ] = 0
   ViewPosY[ v ] = 0
   ViewPosZ[ v ] = 0
   ViewQuatX[ v ] = 0
   ViewQuatY[ v ] = 0
   ViewQuatZ[ v ] = 0
   ViewQuatW[ v ] = 0
   ViewType[ v ] = 0
   ViewProjectionPlaneWidth[ v ] = 0
   ViewProjectionPlaneHeight[ v ] = 0
   ViewErpPhiMin[ v ] = 0
   ViewErpPhiMax[ v ] = 0
   ViewErpThetaMin[ v ] = 0
   ViewErpThetaMax[ v ] = 0
   ViewPerspectiveFocalHor[ v ] = 0
   ViewPerspectiveFocalVer[ v ] = 0
   ViewPerspectivePrincipalPointHor[ v ] = 0
   ViewPerspectivePrincipalPointVer[ v ] = 0
   ViewOrthoWidth[ v ] = 0
   ViewOrthoHeight[ v ] = 0
   ViewQuantizationLaw[ v ] = 0
   ViewNormDispLow[ v ] = 0
   ViewNormDispHigh[ v ] = 0
   ViewOccThreshold[ v ] = 0
   ViewRoot[ v ] = 0
   ViewNumParents[ v ] = 0
   for( i = 0; i < pp_num_parents_minus1[ v ]; i++ ) {
      ViewParentIdx[ v ][ i ] = 0
   }
}
```

If the nal_unit_type of common atlas frame equal to NAL_CAF_IDR, then the process for decoding in subclause 9.7.5.2 is used and the outputs of that process are used as the output.

If the nal_unit_type of common atlas frame equal to NAL_CAF and came_update_extrinsics_flag is 1, then the process for decoding in subclause 9.7.5.3 is used and the outputs of that process are used as the output.

If the nal_unit_type of common atlas frame equal to NAL_CAF and came_update_intrinsics_flag is equal to 1, then the process for decoding in subclause 9.7.5.4 is used and the outputs of that process are used as the output.

If the nal_unit_type of common atlas frame equal to NAL_CAF and came_update_depth_quantization_flag is equal 1, then the process for decoding in subclause 9.7.5.5 is used and the outputs of that process are used as the output.

9.7.5.2 Initializing view information process

The view related variables are derived from the syntax elements in the common atlas frame as follows:

```
NumViews = mvp_num_views_minus1 + 1
```

The view ID and index related arrays are set as follows:

```
for( v = 0; v < NumViews; v++ )
    ViewIDToIndex[ mvp_view_id[ v ] ] = v
```

The inpaint flags are set as follows:

```
for( v = 0; v < NumViews; v++ )
    ViewInpaintFlag[ v ] = mvp_inpaint_flag[ v ]
```

The extrinsic related arrays are set as follows:

```
for( v = 0; v < NumViews; v++ ){
    ViewPosX[ v ] = ce_view_pos_x[ v ]
    ViewPosY[ v ] = ce_view_pos_y[ v ]
    ViewPosZ[ v ] = ce_view_pos_z[ v ]
    ViewQuatX[ v ] = qX[ v ]
    ViewQuatY[ v ] = qY[ v ]
    ViewQuatZ[ v ] = qZ[ v ]
    ViewQuatW[ v ] = qW[ v ]
}
```

If mvp_intrinsic_params_equal_flag equal to 1, the intrinsic related arrays are set as follows:

```
for( v = 0; v < NumViews; v++ ){
    ViewType[ v ] = ci_cam_type[ 0 ]
    ViewProjectionPlaneWidth[ v ] = ci_projection_plane_width_minus1[ 0 ] + 1
    ViewProjectionPlaneHeight[ v ] = ci_projection_plane_height_minus1[ 0 ] + 1
    ViewErpPhiMin[ v ] = ci_erp_phi_min[ 0 ]
    ViewErpPhiMax[ v ] = ci_erp_phi_max[ 0 ]
    ViewErpThetaMin[ v ] = ci_erp_theta_min[ 0 ]
    ViewErpThetaMax[ v ] = ci_erp_theta_max[ 0 ]
    ViewPerspectiveFocalHor[ v ] = ci_perspective_focal_hor[ 0 ]
    ViewPerspectiveFocalVer[ v ] = ci_perspective_focal_ver[ 0 ]
    ViewPerspectivePrincipalPointHor[ v ] = ci_perspective_principal_point_hor[ 0 ]
    ViewPerspectivePrincipalPointVer[ v ] = ci_perspective_principal_point_ver[ 0 ]
    ViewOrthoWidth[ v ] = ci_ortho_width[ 0 ]
    ViewOrthoHeight[ v ] = ci_ortho_height[ 0 ]
}
```

Otherwise, when mvp_intrinsic_params_equal_flag equal to 0, the intrinsic related arrays are set as follows:

```
for( v = 0; v < NumViews; v++ ){
    ViewType[ v ] = ci_cam_type[ v ]
    ViewProjectionPlaneWidth[ v ] = ci_projection_plane_width_minus1[ v ] + 1
    ViewProjectionPlaneHeight[ v ] = ci_projection_plane_height_minus1[ v ] + 1
    ViewErpPhiMin[ v ] = ci_erp_phi_min[ v ]
    ViewErpPhiMax[ v ] = ci_erp_phi_max[ v ]
    ViewErpThetaMin[ v ] = ci_erp_theta_min[ v ]
    ViewErpThetaMax[ v ] = ci_erp_theta_max[ v ]
    ViewPerspectiveFocalHor[ v ] = ci_perspective_focal_hor[ v ]
    ViewPerspectiveFocalVer[ v ] = ci_perspective_focal_ver[ v ]
    ViewPerspectivePrincipalPointHor[ v ] = ci_perspective_principal_point_hor[ v ]
    ViewPerspectivePrincipalPointVer[ v ] = ci_perspective_principal_point_ver[ v ]
    ViewOrthoWidth[ v ] = ci_ortho_width[ v ]
    ViewOrthoHeight[ v ] = ci_ortho_height[ v ]
}
```

If mvp_depth_quantization_params_equal_flag equal to 1, the depth quantization related arrays are set as follows:

```
for( v = 0; v < NumViews; v++ ){
    ViewQuantizationLaw[ v ] = dq_quantization_law[ 0 ]
    ViewNormDispLow[ v ] = dq_norm_disp_low[ 0 ]
    ViewNormDispHigh[ v ] = dq_norm_disp_high[ 0 ]
    ViewOccThreshold[ v ] = dq_depth_occ_threshold_default[ 0 ]
}
```

Otherwise, when mvp_depth_quantization_params_equal_flag equal to 0, the depth quantization related arrays are set as follows:

```
for( v = 0; v < NumViews; v++ ){
    ViewQuantizationLaw[ v ] = dq_quantization_law[ v ]
    ViewNormDispLow[ v ] = dq_norm_disp_low[ v ]
    ViewNormDispHigh[ v ] = dq_norm_disp_high[ v ]
    ViewOccThreshold[ v ] = dq_depth_occ_threshold_default[ v ]
}
```

If mvp_pruning_graph_params_present_flag equal to 1, the pruning related arrays are set as follows:

```
for( v = 0; v < NumViews; v++ ){
    ViewRoot[ v ] = pp_is_root_flag[ v ]
    ViewNumParents[ v ] = pp_num_parents_minus1[ v ] + 1
    for( i = 0; i < ViewNumParents[ v ]; i++ ){
       ViewParentIdx[ v ][ i ] = pp_parent_idx[ v ][ i ]
    }
}
```

9.7.5.3 Updating extrinsic information process

The extrinsic related arrays are updated as follows:

```
for( i = 0; i < mvpue_num_view_updates_minus1 + 1; i++ ){
   v = mvpue_view_idx[ i ]
   ViewPosX[ v ] = ce_view_pos_x[ v ]
   ViewPosY[ v ] = ce_view_pos_y[ v ]
   ViewPosZ[ v ] = ce_view_pos_z[ v ]
   ViewQuatX[ v ] = qX[ v ]
   ViewQuatY[ v ] = qY[ v ]
   ViewQuatZ[ v ] = qZ[ v ]
   ViewQuatW[ v ] = qW[ v ]
}
```

9.7.5.4 Updating intrinsic information process

The intrinsic related arrays are updated as follows:

```
for( i = 0; i < mvpui_num_view_updates_minus1 + 1; i++ ){
   v = mvpui_view_idx[ i ]
   ViewType[ v ] = ci_cam_type[ v ]
   ViewProjectionPlaneWidth[ v ] = ci_projection_plane_width_minus1[ v ] + 1
   ViewProjectionPlaneHeight[ v ] = ci_projection_plane_height_minus1[ v ] + 1
   ViewErpPhiMin[ v ] = ci_erp_phi_min[ v ]
   ViewErpPhiMax[ v ] = ci_erp_phi_max[ v ]
   ViewErpThetaMin[ v ] = ci_erp_theta_min[ v ]
   ViewErpThetaMax[ v ] = ci_erp_theta_max[ v ]
   ViewPerspectiveFocalHor[ v ] = ci_perspective_focal_hor[ v ]
   ViewPerspectiveFocalVer[ v ] = ci_perspective_focal_ver[ v ]
   ViewPerspectivePrincipalPointHor[ v ] = ci_perspective_principal_point_hor[ v ]
   ViewPerspectivePrincipalPointVer[ v ] = ci_perspective_principal_point_ver[ v ]
   ViewOrthoWidth[ v ] = ci_ortho_width[ v ]
   ViewOrthoHeight[ v ] = ci_ortho_height[ v ]
}
```

9.7.5.5 Updating depth quantization information process

The depth quantization related arrays are updated as follows:

```
for( i = 0; i < mvpudq_num_view_updates_minus1; i++ ){
   v = mvpudq_view_idx[ i ]
   ViewQuantizationLaw[ v ] = dq_quantization_law[ v ]
   ViewNormDispLow[ v ] = dq_norm_disp_low[ v ]
   ViewNormDispHigh[ v ] = dq_norm_disp_high[ v ]
   ViewOccThreshold[ v ] = dq_depth_occ_threshold_default[ v ]
}
```

9.8 Sub-bitstream extraction process

9.8.1 General

The specifications in ISO/IEC 23090-5:—, subclause 9.8.1 apply with the following additions.

Subclause 9.8.4 describes the extraction of V3C units of the same group.

9.8.2 V3C unit extraction

The specifications in ISO/IEC 23090-5:—, subclause 9.8.2 apply.

9.8.3 NAL unit extraction process

The specifications in ISO/IEC 23090-5:—, subclause 9.8.3 apply.

9.8.4 Group extraction process

This process extracts V3C units associated with the same group ID.

Inputs to this process are a V3C bitstream, such as a bitstream in the V3C sample stream format defined in Annex C and a target group identifier, *targetGroupId*.

The output of this process is a set of V3C units associated with *targetGroupId*.

The process is as follows:

```
for( a = 0; a < vps_atlas_count_minus1 + 1; a++ ) {
    if( AtlasGroupId[ a ] == targetGroupID )
      invoke subclause 9.8.2 with targetAtlasId equal to a
}
```

10 Pre-reconstruction process

The specifications in ISO/IEC 23090-5:—, Clause 10 do not apply.

11 Reconstruction process

The specifications in ISO/IEC 23090-5:—, Clause 11 apply with following additions.

This document does not define explicit reconstruction processing steps. However, Annex H provides an informative overview of rendering processes, which can be combined by an application to reconstruct a volumetric frame.

12 Post-reconstruction process

The specifications in ISO/IEC 23090-5:—, Clause 12 do not apply.

13 Adaptation process

The specifications in ISO/IEC 23090-5:—, Clause 13 do not apply.

14 Parsing process

The specifications in ISO/IEC 23090-5:—, Clause 14 and its subclauses apply.

Annex A
(normative)

Profiles, tiers, and levels

A.1 Overview of profiles, tiers, and levels

The specifications in ISO/IEC 23090-5:—, Annex A.1 apply.

A.2 Profile, tier and level structure

The specifications in ISO/IEC 23090-5:—, Annex A.2 apply.

A.3 CodecGroup profile components

The specifications in ISO/IEC 23090-5:—, Annex A.3 apply.

A.4 Toolset profile components

A.4.1 General constraints

The following restrictions shall apply to a bitstream conforming to any profile defined in this document.

For all atlases with atlas ID *DecAtlasID*, the following applies.

— When vps_occupancy_video_present_flag[*DecAtlasId*] equal to 1
  — *DecOccHeight* shall be equal to *AsmeOccupancyFrameHeight*[*DecAtlasID*]
  — *DecOccWidth* shall be equal to *AsmeOccupancyFrameWidth*[*DecAtlasID*]
  — *DecOccFullRange* shall be equal to 1
  — *DecOccTransferCharacteristics* shall be equal to 8
— When vps_geometry_video_present_flag[DecAtlasId] equal to 1
  — *DecGeoHeight* shall be equal to *AsmeGeometryFrameHeight*[*DecAtlasID*]
  — *DecGeoWidth* shall be equal to *AsmeGeometryFrameWidth*[*DecAtlasID*]
  — *DecGeoFullRange* shall be equal to 1
  — *DecGeoTransferCharacteristics* shall be equal to 8
— When vps_attribute_video_present_flag[*DecAtlasId*] equal to 1
  — *DecAttrHeight* shall be equal to *AspsFrameHeight*[*DecAtlasID*]
  — *DecAttrWidth* shall be equal to *AspsFrameWidth*[*DecAtlasID*]
— When vps_packed_video_present_flag[*DecAtlasId*] equal to 1
  — *DecPckFullRange* shall be equal to 1

— *DecPckTransferCharacteristics* shall be equal to 8

A.4.2 MIV Main, MIV Extended, MIV Extended Restricted Geometry and MIV Geometry Absent toolset profile components

V3C toolset profile components indicating the MIV Main (ptl_profile_toolset_idc = 64), MIV Extended (ptl_profile_toolset_idc = 65), MIV Extended Restricted Geometry (ptl_profile_toolset_idc = 65 and ptc_restricted_geometry_flag = 1) or MIV Geometry Absent (ptl_profile_toolset_idc = 66) toolset profile component shall conform to the syntax element restrictions specified in Table A.1 and additional restrictions described in this sub-clause.

Table A.1 — Allowable values of syntax element values for the MIV toolset profile components

	Profile name			
Syntax element	**MIV Main**	**MIV Extended**	**MIV Extended Restricted Geometry**	**MIV Geometry Absent**
vuh_unit_type	V3C_VPS, V3C_AD, V3C_GVD, V3C_AVD, or V3C_CAD	V3C_VPS, V3C_AD, V3C_OVD, V3C_GVD, V3C_AVD, V3C_PVD, or V3C CAD	V3C_VPS, V3C_AD, V3C_AVD, V3C_PVD, or V3C CAD	V3C_VPS, V3C_AD, V3C_AVD, V3C_PVD, or V3C CAD
ptl_profile_toolset_idc	64	65		66
ptl_profile_reconstruction_idc	255	255		255
ptc_restricted_geometry_flag	-	0	1	-
VpsMivExtensionPresentFlag	1	1	1	1
VpsPackingInformationPresentFlag	0	0, 1	0, 1	0, 1
vps_map_count_minus1[atlasID]	0	0	0	0
vps_occupancy_video_present_flag[atlasID]	0	0, 1	0	0
vps_geometry_video_present_flag[atlasID]	1	0, 1	0	0
vme_embedded_occupancy_enabled_flag	1	0, 1	0	0
gi_geometry_MSB_align_flag[atlasID]	0	0	0	0
ai_attribute_count[atlasID]	0, 1	0, 1, 2	2	0, 1
ai_attribute_type_id[atlasID][attrIdx]	ATTR_TEXTURE	ATTR_TEXTURE, ATTR_TRANSPARENCY	ATTR_TEXTURE, ATTR_TRANSPARENCY	ATTR_TEXTURE
ai_attribute_dimension_minus1[atlasID] [attrTextureIdx]	2	2	2	2
ai_attribute_dimension_minus1[atlasID] [attrTransparencyIdx]	-	0	0	-
ai_attribute_dimension_partitions_minus1[atlasID] [attrIdx]	0	0	0	0
ai_attribute_MSB_align_flag[atlasID][attrIdx]	0	0	0	0
asps_long_term_ref_atlas_frames_flag	0	0	0	0
asps_pixel_deinterleaving_enabled_flag	0	0	0	0
asps_patch_precedence_order_flag	0	0	0	0

Table A.1 *(continued)*

	Profile name			
Syntax element	**MIV Main**	**MIV Extended**	**MIV Extended Restricted Geometry**	**MIV Geometry Absent**
asps_raw_patch_enabled_flag	0	0	0	0
asps_eom_patch_enabled_flag	0	0	0	0
asps_plr_enabled_flag	0	0	0	0
asme_patch_constant_depth_flag	0	0, 1	1	0, 1
vps_geometry_video_present_flag[atlasID] || asme_patch_constant_depth_flag	-	1	1	0, 1
vps_packed_video_present_flag[atlasID]	0	0, 1	0, 1	0, 1
afps_lod_mode_enabled_flag	0	0	0	0
afps_raw_3d_pos_bit_count_explicit_mode_flag	0	0	0	0
afti_single_tile_in_atlas_frame_flag	1	0, 1	0, 1	0, 1
ath_type	I_TILE	I_TILE	I_TILE	I_TILE
atdu_patch_mode[tileID][patchIdx]	I_INTRA	I_INTRA	I_INTRA	I_INTRA
asps_atlas_sequence_parameter_set_id	0..63, inclusive	0..63, inclusive	0..63, inclusive	0..63, inclusive
afps_atlas_frame_parameter_set_id	0..63, inclusive	0..63, inclusive	0..63, inclusive	0..63, inclusive
afps_atlas_sequence_parameter_set_id	0..63, inclusive	0..63, inclusive	0..63, inclusive	0..63, inclusive
aaps_atlas_adaptation_parameter_set_id	0..63, inclusive	0..63, inclusive	0..63, inclusive	0..63, inclusive
ath_atlas_frame_parameter_set_id	0..63, inclusive	0..63, inclusive	0..63, inclusive	0..63, inclusive
ath_atlas_adaptation_parameter_set_id	0..63, inclusive	0..63, inclusive	0..63, inclusive	0..63, inclusive
dq_quantization_law[v]	0	0	0	0

The following restrictions shall apply to a bitstream or a collection of V3C sub-bitstreams conforming to the MIV Extended Restricted Geometry toolset profile component:

— For each atlas present, one texture attribute and one transparency attribute shall be present.

NOTE The MIV Extended Restricted Geometry profile corresponds to an MPI representation of the 3D scene and enables an alternative efficient rendering based on alpha blending commonly known as reverse painter's algorithm.

A.5 Reconstruction profile components

The specifications in ISO/IEC 23090-5:—, Annex A.5 apply.

A.6 Tiers and Levels

The specifications in ISO/IEC 23090-5:—, Annex A.6 apply.

A.7 Decoder instantiations

The specifications in ISO/IEC 23090-5:—, Annex A.7 apply.

Annex B
(informative)

Post-decoding conversion to nominal video formats

B.1 General

The video frames provided by the decoder may require additional processing steps before the reconstruction process. Such processing steps may include unpacking of the decoded video frames to separate geometry, attribute and/or occupancy frames, as described in Annex B.4.

B.2 Nominal format conversion

B.2.1 General

The specifications in ISO/IEC 23090-5:—, Annex B.2.1 apply.

B.2.2 Occupancy nominal format conversion

The specifications in ISO/IEC 23090-5:—, Annex B.2.1 apply.

B.2.3 Geometry nominal format conversion

The specifications in ISO/IEC 23090-5:—, Annex B.2.3 apply with the following additions.

The nominal output frame horizontal resolution is set to *AsmeGeometryFrameWidth*[j] and the vertical resolution is set to *AsmeGeometryFrameHeight*[j].

B.2.4 Auxiliary geometry nominal format conversion

The specifications in ISO/IEC 23090-5:—, Annex B.2.4 do not apply.

B.2.5 Attribute nominal format conversion

The specifications in ISO/IEC 23090-5:—, Annex B.2.5 apply.

B.2.6 Auxiliary attribute nominal format conversion

The specifications in ISO/IEC 23090-5:—, Annex B.2.6 do not apply.

B.3 Conversion operations

B.3.1 Map Extraction

The specifications in ISO/IEC 23090-5:—, Annex B.3.1 apply.

B.3.2 Bit depth conversion

The specifications in ISO/IEC 23090-5:—, Annex B.3.2 apply.

B.3.3 Resolution conversion

The specifications in ISO/IEC 23090-5:—, Annex B.3.3 apply.

B.3.4 Output composition time conversion

The specifications in ISO/IEC 23090-5:—, Annex B.3.4 do not apply.

B.3.5 Attribute dimension packing

The specifications in ISO/IEC 23090-5:—, Annex B.3.5 apply.

B.3.6 Chroma up-sampling

The specifications in ISO/IEC 23090-5:—, Annex B.3.6 apply.

B.3.7 Geometry map synthesis process

The specifications in ISO/IEC 23090-5:—, Annex B.3.7 apply.

B.3.8 Attribute map synthesis process

The specifications in ISO/IEC 23090-5:—, Annex B.3.8 apply.

B.3.9 Occupancy thresholding process

The specifications in ISO/IEC 23090-5:—, Annex B.3.9 apply.

B.4 Unpacking process of a decoded packed video

The specifications in ISO/IEC 23090-5:—, Annex B.4 apply.

Annex C
(informative)

V3C sample stream format

Annex C of ISO/IEC 23090-5:— applies.

Annex D
(normative)

NAL sample stream format

Annex D of ISO/IEC 23090-5:— applies.

Annex E
(normative)

Atlas hypothetical reference decoder

Annex E of ISO/IEC 23090-5:— applies.

Annex F
(normative)

Supplemental enhancement information

F.1 General

The specifications in ISO/IEC 23090-5:—, Annex F.1 apply with the following additions to Table F.1:

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

F.2 SEI payload syntax

The specifications in ISO/IEC 23090-5:—, Annex F.2 and its subclauses apply, with the following additions:

F.2.1 Viewing space SEI payload syntax

F.2.1.1 General

viewing_space(payloadSize) {	Descriptor
vs_num_elementary_shapes_minus1	u(v)
for(e = 0; e <= vs_num_elementary_shapes_minus1; e++) {	
vs_elementary_shape_operation[e]	u(1)
elementary_shape(e)	
}	
}	

F.2.1.2 Elementary shape

elementary_shape(e) {	Descriptor
es_num_primitive_shapes_minus1[e]	u(8)
es_primitive_shape_operation[e]	u(2)
es_guard_band_present_flag[e]	u(1)
es_primitive_orientation_present_flag[e]	u(1)
es_viewing_direction_constraint_present_flag[e]	u(1)
es_camera_inferred_flag[e]	u(1)

for(s = 0; s <= es_num_primitive_shapes_minus1; s++) {	
if(es_camera_inferred_flag[e])	
es_view_idx[e][s]	u(16)
es_primitive_shape_type[e][s]	u(2)
if(es_primitive_shape_type[e][s] == 0)	
cuboid_primitive(e, s)	
else if(es_primitive_shape_type[e][s] == 1)	
spheroid_primitive(e, s)	
else if(es_primitive_shape_type[e][s] == 2)	
halfspace_primitive(e, s)	
if(es_guard_band_present_flag[e])	
es_guard_band_size[e][s]	fl(16)
if(es_primitive_orientation_present_flag[e]) {	
if(!es_camera_inferred_flag[e]) {	
es_primitive_shape_quat_x[e][s]	i(16)
es_primitive_shape_quat_y[e][s]	i(16)
es_primitive_shape_quat_z[e][s]	i(16)
}	
}	
if(es_viewing_direction_constraint_present_flag[e]) {	
if(es_guard_band_present_flag[e])	
es_guard_band_direction_size[e][s]	fl(16)
if(!es_camera_inferred_flag[e]) {	
es_primitive_shape_viewing_direction_quat_x_center[e][s]	i(16)
es_primitive_shape_viewing_direction_quat_y_center[e][s]	i(16)
es_primitive_shape_viewing_direction_quat_z_center[e][s]	i(16)
}	
es_primitive_shape_viewing_direction_yaw_range[e][s]	fl(16)
es_primitive_shape_viewing_direction_pitch_range[e][s]	fl(16)
}	
}	
}	

F.2.1.3 Cuboid primitive

cuboid_primitive(e, s) {	
if(!es_camera_inferred_flag[e]) {	
cp_center_x[e][s]	fl(16)
cp_center_y[e][s]	fl(16)
cp_center_z[e][s]	fl(16)
}	
cp_size_x[e][s]	fl(16)
cp_size_y[e][s]	fl(16)
cp_size_z[e][s]	fl(16)
}	

F.2.1.4 Spheroid primitive

spheroid_primitive(e, s) {	Descriptor
if(!es_camera_inferred_flag[e]) {	
sp_center_x[e][s]	fl(16)
sp_center_y[e][s]	fl(16)
sp_center_z[e][s]	fl(16)
}	
sp_radius_x[e][s]	fl(16)
sp_radius_y[e][s]	fl(16)
sp_radius_z[e][s]	fl(16)
}	

F.2.1.5 Half space primitive

halfspace_primitive(e, s) {	Descriptor
hp_normal_x[e][s]	fl(16)
hp_normal_y[e][s]	fl(16)
hp_normal_z[e][s]	fl(16)
hp_distance[e][s]	fl(16)
}	

F.2.2 Viewing space handling SEI payload syntax

viewing_space_handling(payloadSize) {	Descriptor
vs_handling_options_count	ue(v)
for(h = 0; h < vs_handling_options_count; h++) {	
vs_handling_device_class[h]	u(6)
vs_handling_application_class[h]	u(6)
vs_handling_method[h]	u(6)
}	
}	

F.2.3 Geometry upscaling parameters SEI payload syntax

geometry_upscaling_parameters(payloadSize) {	Descriptor
gup_type	ue(v)
if(gup_type == 0) {	
gup_erode_threshold	fl(16)
gup_delta_threshold	ue(v)
gup_max_curvature	u(3)
}	
}	

F.2.4 Atlas view enabled SEI payload syntax

atlas_view_enabled(payloadSize) {	Descriptor
ave_cancel_flag	u(1)
if(!ave_cancel_flag) {	
ave_persistence_flag	u(1)
ave_atlas_count_minus1	u(6)
ave_num_views_minus1	u(16)
for(a = 0; a <= ave_atlas_count_minus1; a++) {	
ave_atlas_id[a]	u(v)
atlasID = ave_atlas_id[a]	
for(v = 0; v <= ave_num_views_minus1; v++) {	
ave_view_enabled_in_atlas_flag[atlasID][v]	u(1)
if(ave_view_enabled_in_atlas_flag[atlasID][v])	
ave_view_complete_in_atlas_flag[atlasID][v]	u(1)
}	
}	
}	
}	

F.2.5 OMAF v1 compatible SEI payload syntax

omaf_v1_compatible(payloadSize) {	Descriptor
ov1_num_atlases_minus1	u(6)
for(i = 0; i < ov1_num_atlases_minus1 + 1; i++) {	
ov1_atlas_id[i]	u(6)
}	
}	

F.2.6 Geometry assistance SEI payload syntax

geometry_assistance(payloadSize) {	Descriptor
gas_qs	ue(v)
gas_num_views_minus1	ue(v)
gas_log2_bw_minus2	ue(v)
for(v = 0; v <= gas_num_views_minus1; v++) {	
gas_projection_plane_height_minus1[v]	ue(v)
gas_projection_plane_width_minus1[v]	ue(v)
for(l = 0; l < (gas_projection_plane_height_minus1[v] + gasBW) /gasBW; l++) {	
for(c = 0; c < (gas_projection_plane_width_minus1[v] + gasBW) / gasBW; c++) {	
gas_split_flag	u(1)
subBlocksHorizontally = 1	
subBlocksVertically = 1	
if(gas_split_flag) {	
gas_quad_split_flag	u(1)
if(gas_quad_split_flag) {	

subBlocksHorizontally = 2	
subBlocksVertically = 2	
} else {	
gas_split_orientation_flag	u(1)
if(gas_split_orientation_flag) {	
subBlocksHorizontally = 2	
} else {	
subBlocksVertically = 2	
}	
gas_split_symmetry_flag	u(1)
if(!gas_split_symmetry_flag) {	
gas_split_first_block_bigger	u(1)
}	
}	
}	
for(sbl = 0; sbl < subBlocksVertically; sbl++) {	
for(sbc = 0; sbc < subBlocksHorizontally; sbc++) {	
gas_skip_flag	u(1)
if(!gas_skip_flag) {	
if(l == 0 && c == 0 && sbl == 0 && sbc == 0) { /*None*/	
LTMinFlag = 2	
LTMaxFlag = 2	
}	
else if(l == 0 && sbl == 0) { /*Left*/	
LTMinFlag = 0	
LTMaxFlag = 0	
}	
else if(c == 0 && sbc == 0) { /*Top*/	
LTMinFlag = 1	
LTMaxFlag = 1	
}	
else {	
gas_ltmin_flag	u(1)
gas_ltmax_flag	u(1)
LTMinFlag = gas_ltmin_flag	
LTMaxFlag = gas_ltmax_flag	
}	
gas_zmin_delta	se(v)
gas_zmax delta	se(v)
}	
}	
}	
}	
}	

}	
}	

F.3 SEI payload semantics

F.3.1 General

The specifications in ISO/IEC 23090-5:—, Annex F.3 and its subclauses apply, with the following additions and modifications.

F.3.2 General SEI payload semantics

The specifications in ISO/IEC 23090-5:—, Annex F.3.1 apply with the following additions to Table F.2:

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

F.3.3 Viewing space SEI payload semantics

F.3.3.1 General

The viewing space indicates the portion of the space, possibly completed by viewing direction constraints, where the viewport can be rendered with high quality. It is based on the possibility to give the end device the opportunity to handle viewing space exceedance. A viewing space inclusiveness factor can be computed where 0 indicates fully inside and 1 indicates fully outside. The end device application can use this factor to take a viewers' transient, from inside the viewing space to outside, into account.

The construction of the viewing space is based on a list of elementary shapes which are themselves based on a list of primitive shapes. The primitive shapes can be built into elementary shapes through CSG (Constructive Solid Geometry) operation or through interpolation operation, and these elementary shapes can be combined by CSG addition, subtraction, or intersection as defined by elementary_shape_operation, in the strict order of the list of elementary shapes.

vs_elementary_shape_operation[e] equal to 0 specifies that the type of CSG operation to apply on the elementary shape with index e is additive. vs_elementary_shape_operation[e] equal to 1 specifies that the type of CSG operation to apply on the elementary shape with index e is subtractive. vs_elementary_shape_operation[e] equal to 2 specifies that the type of CSG operation to apply on the elementary shape with index e is intersection. The operation consists of computing the contribution of the signed distance SD(p, E) of a point p related to that elementary shape E with index e to the global signed distance SD(p) in the entire global viewing space.

vs_num_elementary_shapes_minus1 plus 1 indicates the number of elementary shapes to build the viewing space.

F.3.3.2 Elementary shape

es_num_primitive_shapes_minus1[e] plus 1 specifies the number of primitive shapes that are used in the construction of the elementary shape e.

es_primitive_shape_operation[e] equal to 0 specifies the use of CSG mode for the primitive shapes which are simply added together to form the larger elementary shape e. es_primitive_shape_operation[e] equal to 1 specifies the interpolative mode, in which the primitive shapes in the list are interpolated along a path defined by the ordered centroids of the primitive shape.

When es_primitive_shape_operation[e] is equal to 1, the operation is based on interpolation along the segment path defined by the centers of the successive primitive shapes in the ordered list of the syntax structure. The operation is based on regular metric distance RD(p, S) of a point p related to a shape S center which has been shifted along the path, where the RD function is the geometric distance (L2) between two points. The shift value is a linear operation between regular distances RD(p, S_s) and RD(p, S_{s+1}) to the two closest successive primitive shapes S_s and S_{s+1}. The interpolated elementary shapes are combined additively into the viewing space.

The result of the operation on the primitives for the elementary shape E produces a signed distance SD(p, E) for all point p of the global space regarding this elementary shape E with index e.

es_guard_band_present_flag[e] equal to 1 specifies that a guard band information is present for each primitive shape in the elementary shape e. es_guard_band_present_flag[e] equal to 0 specifies that no information is present. The guard band is a frontier on the inside of the viewing volume which may trigger an action in the rendering client: for example, a scene may begin to fade or blur as the viewer enters the guard band distance, indicating proximity to the viewing volume boundary.

es_primitive_orientation_present_flag[e] equal to 1 specifies that per-primitive orientation information is present for each primitive shape in the elementary shape e. es_primitive_orientation_present_flag[e] equal to 0 specifies that per-primitive orientation information is not present, and that the primitives are axis-aligned.

es_viewing_direction_constraint_present_flag[e] equal to 1 specifies that viewing direction constraints are present for each primitive shape in the elementary shape e. es_viewing_direction_constraint_present_flag[e] equal to 0 specifies that per-primitive viewing direction constraints are not present.

es_camera_inferred_flag[e] equal to 1 specifies that the positions and orientations of the primitive shapes are those of the views with indices es_view_idx[e][s] in the miv_view_params_list( ).

es_view_idx[e][s] specifies the view index for for which the primitive shapes are defined. The value of es_view_idx[i] shall be in the range of 0 to mvp_num_views_minus1, inclusive.

es_primitive_shape_type[e][s] indicates the type of primitive shape s of the elementary shape e detailed below as in Table F.3.

Table F.3 — primitive shape types

es_primitive_shape_type[e][s]	Shape
0	cuboid primitive
1	spheroid primitive
2	halfspace primitive
3	other primitive

The value of 3 is typically reserved for shape which would be more complex and no more corresponding to a cardinal shape. This shape could be defined through a SEI message or through means outside this Specification.

es_guard_band_size[e][s] specifies the width of the positional guard band for each primitive shape s of an elementary shape e. es_guard_band_present_flag[e] equal to 0 implies that the guard band size is

implicitly 0. This parameter is expressed in same unit as the position parameter of the primitive shape. It is based on the signed distance which can be computed for each primitive shape, whatever the es_primitive_shape_operation[e] is (CSG or interpolation). The guard band can be effectively treated as a second signed distance SD(p, S) + guard band size that can be carried through the same operations to result at a final guard band distance SD(p, S_{GUARD}).

es_primitive_shape_quat_x[e][s] specifies the x component, *qX*[e][s], for the rotation to apply on the primitive shape s of the elementary shape e using the quaternion representation. The value of es_primitive_shape_quat_x[e][s] shall be in the range of -2^{14} to 2^{14}, inclusive. When es_primitive_shape_quat_x[e][s] is not present, its value shall be inferred to be equal to 0. The value of *qX*[e][s] is computed as follows:

$$qX[\,e\,][\,s\,] = es\_primitive\_shape\_quat\_x[\,e\,][\,s\,] \div 2^{14} \quad \text{(F.1)}$$

es_primitive_shape_quat_y[e][s] specifies the y component, *qY*[e][s], for the rotation to apply on the primitive shape s of the elementary shape e using the quaternion representation. The value of es_primitive_shape_quat_y[e][s] shall be in the range of -2^{14} to 2^{14}, inclusive. When es_primitive_shape_quat_y[e][s] is not present, its value shall be inferred to be equal to 0. The value of *qY*[e][s] is computed as follows:

$$qY[\,e\,][\,s\,] = es\_primitive\_shape\_quat\_y[\,e\,][\,s\,] \div 2^{14} \quad \text{(F.2)}$$

es_primitive_shape_quat_z[e][s] specifies the y component, *qZ*[e][s], for the rotation to apply on the primitive shape s of the elementary shape e using the quaternion representation. The value of es_primitive_shape_quat_z[e][s] shall be in the range of -2^{14} to 2^{14}, inclusive. When es_primitive_shape_quat_z[e][s] is not present, its value shall be inferred to be equal to 0. The value of *qZ*[e][s] is computed as follows:

$$qZ[\,e\,][\,s\,] = es\_primitive\_shape\_quat\_z[\,e\,][\,s\,] \div 2^{14} \quad \text{(F.3)}$$

When the operation is based on CSG, the rotation is applied about the centroid of the primitive S before applying the corresponding distance function SD(p, S).

It is a requirement of bitstream conformance that:

$$qX[\,e\,][\,s\,]^2 + qY[\,e\,][\,s\,]^2 + qZ[\,e\,][\,s\,]^2 <= 1 \quad \text{(F.4)}$$

The fourth component, qW, of the quaternion is calculated as follows:

$$qW[\,e\,][\,s\,] = Sqrt(\,1 - (\,qX[\,e\,][\,s\,]^2 + qY[\,e\,][\,s\,]^2 + qZ[\,e\,][\,s\,]^2\,)\,) \quad \text{(F.5)}$$

es_guard_band_direction_size[e][s] specifies the width of the directional guard band for each primitive shape s of an elementary shape e. es_guard_band_present_flag[e] equal to 0 implies that the guard band directional_size is implicitly 0. This parameter is expressed in degree.

es_primitive_shape_viewing_direction_quat_x_center[e][s] specifies the x component, *qXc*[e][s], for the suggested viewing directions center for the primitive shape s of the elementary shape e using the quaternion representation. The value of es_primitive_shape_viewing_direction_quat_x_center[e][s] shall be in the range of -2^{14} to 2^{14}, inclusive. When es_primitive_shape_viewing_direction_quat_x_center[e][s] is not present, its value shall be inferred to be equal to 0. The value of *qXc*[e][s] is computed as follows:

$$qXc[\,e\,][\,s\,] = es\_primitive\_shape\_viewing\_direction\_quat\_x\_center\,[\,e\,][\,s\,] \div 2^{14} \quad \text{(F.6)}$$

es_primitive_shape_viewing_direction_quat_y_center[e][s] specifies the y component, q*Yc*[e][s], for the suggested viewing directions center for the primitive shape s of the elementary shape e using the quaternion representation. The value of es_primitive_shape_viewing_direction_quat_y_center[e]

[s] shall be in the range of -2^{14} to 2^{14}, inclusive. When es_primitive_shape_viewing_direction_quat_y_center[e][s] is not present, its value shall be inferred to be equal to 0. The value of *qYc*[e][s] is computed as follows:

$$qYc[\,e\,][\,s\,] = \text{es\_primitive\_shape\_viewing\_direction\_quat\_y\_center}\,[\,e\,][\,s\,] \div 2^{14} \tag{F.7}$$

es_primitive_shape_viewing_direction_quat_z_center[e][s] specifies the z component, *qZc*[e][s], for the suggested viewing directions center for the primitive shape s of the elementary shape e using the quaternion representation. The value of es_primitive_shape_viewing_direction_quat_z_center[e][s] shall be in the range of -2^{14} to 2^{14}, inclusive. When es_primitive_shape_viewing_direction_quat_z_center[e][s] is not present, its value shall be inferred to be equal to 0. The value of *qZc*[e][s] is computed as follows:

$$qZc[\,e\,][\,s\,] = \text{es\_primitive\_shape\_viewing\_direction\_quat\_z\_center}\,[\,e\,][\,s\,] \div 2^{14} \tag{F.8}$$

The suggested viewing direction is obtained by applying the quaternion with previously mentioned components to the axis taken as forward axis for the views.

It is a requirement of bitstream conformance that:

$$qXc[\,e\,][\,s\,]^2 + qYc[\,e\,][\,s\,]^2 + qZc[\,e\,][\,s\,]^2 <= 1 \tag{F.9}$$

The fourth component, *qWc*, of the quaternion is calculated as follows:

$$qWc[\,e\,][\,s\,] = \text{Sqrt}(\,1 - (\,qXc[\,e\,][\,s\,]^2 + qYc[\,e\,][\,s\,]^2 + qZc[\,e\,][\,s\,]^2\,)\,) \tag{F.10}$$

es_primitive_shape_viewing_direction_yaw_range[e][s] specifies in degrees the yaw half range of suggested viewing directions for the s-th primitive shape.

es_primitive_shape_viewing_direction_pitch_range[e][s] specifies in degrees the pitch half range of suggested viewing directions for the s-th primitive shape.

The viewing constraints $V_i(S_i)$ of each primitive shapes S_i(guard band, viewing direction center, viewing direction range and directional guard band) together define the viewing space constraints V(p, E) at point p for the elementary shape E as follows.

When es_primitive_shape_operation[e] equal 0 (operation on shapes based on CSG), these are interpolated for a given point p and all primitive shapes S_i and related signed distances SD(p, S_i) of that elementary shape E as follows.

$$V(\,p,E\,) = \sum_{i=0}^{N} -SD(\,p,S_i\,) * V_i(\,S_i\,) \,/\, \sum_{i=0}^{N} -SD(\,p,S_i\,) \tag{F.11}$$

where N is equal to es_num_primitive_shape_minus1[e]

When es_primitive_shape_operation[e] equal 1 (operation on shapes based on interpolation), formula (F.11) reduces to a linear interpolation between the two closest primitive shapes S_s and S_{s+1} taken in the order of the primitive shape list with the use of the regular distance RD(p, S_i).

$$V(p, E) = (RD(p, S_{s+1}) * V_s + RD(p, S_s) * V_{s+1}) / (RD(p, S) + RD(p, S_{s+1})) \quad \text{(F.12)}$$

V(p) finally gives the viewing constraints guard band size, viewing direction center, viewing direction range and directional guard band size for a given point p from the viewing constraints V(p, E_i) of each elementary shape E_i as follows.

$$V(p) = \sum_{i=0}^{N} -SD(p, E_i) * V(p, E_i) / \sum_{i=0}^{N} -SD(p, E_i) \quad \text{(F.13)}$$

where N is equal to vs_num_elementary_shapes_minus1

The index of positional fading within the global viewing space is then computed as shown in the following formula.

position_fading_index(p) =
Clip3(0, 1, (SD(p) + guard_band_size(p)) / guard_band_size(p)) (F.14)

where p is the vector of coordinates of the viewport, SD(p) the global signed distance of p and guard_band_size(p) the global guard band size value at the position p from Formula (F.13).

The index of directional fading for yaw is then computed as shown in the following formula (the equivalent formula for directional fading for pitch applies by replacing yaw by pitch):

yaw_fading_index(p) = Clip3(0, 1,
(abs(yaw(p) – viewing_yaw_center(p)) – viewing_yaw_range(p) +
guard_band_direction_size(p)) / guard_band_direction_size(p)) (F.15)

where yaw(p) is the yaw value of the viewport quaternion, viewing_yaw_center(p) is the yaw value of the direction center quaternion, viewing_yaw_range(p) is the viewing direction range in yaw, guard_band_direction_size(p) is the directional guard band size at that viewport position p from Formula (F.13).

The global fading index which is applied on the viewport RGB components is given by the multiplication of position_fading_index, yaw_fading_index and pitch_fading_index.

F.3.3.3 Cuboid primitive

cp_center_x[e][s], **cp_center_y**[e][s], **cp_center_min_z**[e][s] specify the x, y, z coordinates, respectively, in the scene coordinate system of the center of the cuboid.

cp_size_x[s][e], **cp_size_y**[s][e], **cp_size_z**[s][e] specify the size of the cuboid in x, y, z directions, respectively, in the scene coordinate system.

The signed distance function for a cuboid primitive is

$$SD_{CUBOID}(p, l, h) = Min(Max(d_x, Max(d_y, d_z)), 0) + Norm(Max(d, 0)) \quad \text{(F.16)}$$

where (d_x, d_y, d_z) are the coordinates of the point as regards to the primitive shape center, l is the 3D vector (cp_center_x – cp_size_x / 2, cp_center_y – cp_size_y / 2, cp_center_z – size_z / 2), h is (cp_center_x + cp_size_x / 2, cp_center_y + cp_size_y / 2, cp_center_z + cp_size_z / 2), and d is Max(l – p, p – h). The max operations on vectors are to be applied per element.

F.3.3.4 Spheroid primitive

sp_center_x[e][s], **sp_center_y**[e][s], **sp_center_min_z**[e][s] specify the x, y, z coordinates, respectively, in the scene coordinate system of the center of the cuboid.

sp_radius_x[e][s], **sp_radius_y**[e][s], **sp_radius_z**[e][s], specify the dimension x, y and z, respectively, of the spheroid in the scene coordinate system.

The signed distance function for a spheroid primitive is

$$SD_{SPHEROID}(p, r) = Norm(p / r) * (Norm(p / r) - 1) / Norm(p / r^2) \quad (F.17)$$

where the 3D point p is as regards to the primitive center (sp_center_x, sp_center_y, sp_center_z), r equals the (sp_radius_x, sp_radius_y, sp_radius_z) vector, and the division and squaring operation is applied per vector element.

F.3.3.5 Half space primitive

hp_normal_x[e][s], **hp_normal_y**[e][s], **hp_normal_z**[e][s] indicate the normal facing of the plane defining the half-space.

hp_distance[e][s] specifies the distance from the scene origin along the normal vector direction to the plane defining the half-space.

The signed distance function for a half-space primitive for any point p in 3D space is

$$SD_{HALFSPACE}(p, n, d) = Dot(p, n / Norm(n)) - d \quad (F.18)$$

where n is the normal vector given by (hp_normal_x, hp_normal_y, hp_normal_z) and d equals hp_distance.

The centroid of a half-space primitive, if needed in calculations, shall be substituted with d * n.

F.3.4 Viewing space handling SEI payload semantics

When viewing space handling methods are present, the target device selects the first matching handling method. Matching is performed based on a device and application class. When none of the viewing space handling methods match with the target, no viewing space handling is provided. In that case the target device should choose an appropriate handling based on the viewing space information alone.

vs_handling_options_count specifies the number of viewing space handling options. When vs_handling_options_count is zero, no viewing space handling is provided. In that case the target device should choose an appropriate handling based on the viewing space information alone.

vs_handling_device_class[h] specifies the allowed values of vs_handling_device_class[h] within those specified in Table F.4. In all cases it is assumed that the device is capable (to some degree) of 6DoF viewer position tracking. In some cases, the viewer moves in respect to the display. A conformant bitstream shall not have duplicate values for vs_handling_device_class[h] within the same viewing_space( ) structure. When vs_handling_device_class[h] == VHDC_ALL, then it shall hold that h + 1 == vs_handling_options_count.

vs_handling_application_class[h] specifies the allowed values of vs_handling_application_class[h] within those specified in Table F.5. A conformant bitstream shall not have duplicate values for vs_handling_application_class[h] within the same viewing_space( ) structure. When vs_handling_application_class[h] == VHAC_ALL, then it shall hold that h + 1 == vs_handling_options_count.

vs_handling_method[h] specifies the allowed values of vs_handling_application_class[h] within those specified in Table F.6. A conformant bitstream shall not have duplicate values for vs_handling_application_class[h] within the same viewing_space( ) structure. When vs_handling_application_class[h] == VHAC_ALL, then it shall hold that h + 1 == vs_handling_options_count.

Table F.4 — Viewing space handling device classes

Value	Name	Description
0	VHDC_ALL	Match against all devices
1	VHDC_HMD	Head-mounted display with 6DoF positioning
2	VHDC_PHONE	Mobile phone or tablet with screen rendering depending on IMU
3	VHDC_ASD	Autostereoscopic (lightfield) display
4..31	VHDC_RSRVD_5..VHDC_RSRVD_31	Reserved for future use by ISO/IEC
32..63	VHDC_UNSPCF_32..VHDC_UNSP-CF_63	Unspecified (available for specification by other standards)

Table F.5 — Viewing space handling application classes

Value	Name	Description
0	VHAC_ALL	Match against all applications
1	VHAC_AR	The coded immersive video is used to augment the physical world
2	VHAC_VR	The coded immersive video is used as a virtual reality
3	VHAC_WEB	The coded immersive video is embedded within a website
4	VHAC_SD	The coded immersive video is used as an element within a larger scene description
5..31	VHAC_RSRV_5..VHAC_RSRV_31	Reserved for future use by ISO/IEC
32..63	VHAC_UNSPF_32..VHAC_UNSPF_63	Unspecified (available for specification by other standards)

Table F.6 — Viewing space handling methods

Value	Name	Description
0	VHM_NULL	Default client behavior
1	VHM_RENDER	Always render, even when outside of the viewing space. This may cause rendering artifacts.
2	VHM_FADE	When moving towards the outside of the viewing space, the scene fades to a default color.
3	VHM_EXTRAP	Extrapolate content in an abstract low-frequent way that prevents rendering artifacts but preserves the general color tone of the scene.
4	VHM_RESET	The viewer position and/or orientation is reset when the viewer reaches the limit of the viewing zone
5	VHM_STRETCH	The scene rotates and translates along with the viewer to prevent the viewer from reaching the limit of the viewing zone
6	VHM_ROTATE	The scene rotates with the viewer to keep the viewer within the field of view
7..31	VHM_RSRV_5..VHM_RSRV_31	Reserved for future use by ISO/IEC
32..63	VHM_UNSPF_32..VHM_UNSPF_63	Unspecified (available for specification by other standards)

F.3.5 Geometry upscaling parameters SEI payload semantics

gup_type is the type of geometry upscaling to which the provided parameters apply. This version of the document defines the value 0 in accordance with the geometry video scaling process in subclause (H.5). All positive even values are reserved for future use by ISO/IEC. All odd values are unspecified (available for specification by other standards).

gup_erode_threshold specifies the threshold that is applied in the texture aligned geometry erosion process (H.5.6) to determine if selective erosion is applied for a pixel or not. When not present, the value of gup_erode_threshold is inferred to be equal to 1.0.

The variable *GupErodeThreshold* is set equal to gup_erode_threshold.

gup_delta_threshold specifies the threshold that is applied in the texture aligned geometry erosion process (H.5.6) to determine the partial depth order between two samples. When not present, the value of gup_delta_threshold is inferred to be equal to 10.

The variable *GupDeltaThreshold* is set equal to gup_delta_threshold.

gup_max_curvature specifies the threshold that determines if the curvature correction of the geometry contour smoothing process (H.5.7) is applied to a geometry sample. When not present, the value of gup_max_curvature is inferred to be equal to 5.

The variable *GupMaxCurvature* is set equal to gup_max_curvature.

F.3.6 Atlas view enabled SEI payload semantics

The atlas view enabled SEI message provides indication of which views may be represented in atlases.

ave_cancel_flag equal to 1 cancels the persistence of the atlas view enabled SEI message.

ave_persistence_flag specifies the persistence of any previous atlas view enabled SEI message. ave_persistence_flag equal to 0 specifies that the atlas view enabled SEI message applies to the current atlas frame only. ave_persistence_flag equal to 1 specifies that the atlas view enabled SEI message applies to the current atlas frame and persists for all subsequence atlas frames in decoding order until one of the following conditions are true:

— A new sequence begins.

— The bitstream ends.

— An atlas frame containing an atlas view enabled SEI message is present

ave_atlas_count_minus1 plus 1 indicates the number of atlases for which atlas view enabling parameters will be signalled.

ave_num_views_minus1 plus 1 indicates the number of views for which enabling parameters are signalled.

ave_atlas_id[a] specifies the ID of the atlas with index a. The value of ave_atlas_id[a] shall be in the range of 0 to 63, inclusive. It is a requirement of bitstream conformance to this version of this document that the value of ave_atlas_id[k] shall not be equal to ave_atlas_id[j] for all j != k.

ave_view_enabled_in_atlas_flag[atlasID][v] equal to 1 indicates that patches in the atlas with atlas ID equal to atlasID may be associated with the view with index equal to v. ave_view_enabled_in_atlas_flag [atlasID][v] equal to 0 indicates that patches in the atlas with atlas ID equal to atlasID do not contain a representation of the view with index equal to v.

ave_view_complete_in_atlas_flag[atlasID][v] equal to 1 indicates that patches in the atlas with atlas ID equal to atlasID are associated with the complete representation of the view with index equal to v. ave_view_complete_in_atlas_flag[atlasID][v] equal to 0 does not indicate any constraints.

F.3.7 OMAF v1 compatible SEI payload semantics

This SEI message indicates that the texture frames of one or more atlases in the CVS are compatible for carriage within ISO/IEC 23090-2, i.e., that the metadata specified in ISO/IEC 23090-2 as constrained by the packed equirectangular or cubemap projected video scheme type ('ercm') specified in ISO/IEC 23090-2 may correctly specify the 360° video contained in the texture frames of one or more atlases in the CVS.

ov1_num_atlases_minus1 plus 1 indicates the number of atlases in the CVS for which the texture frames are compatible for carriage within ISO/IEC 23090-2. ov1_num_atlases_minus1 shall be equal to or less than vps_atlas_count_minus1.

ov1_atlas_id[i] specifies the atlas ID of the i-th atlas for which the texture frames are compatible for carriage within ISO/IEC 23090-2. The value of ov1_atlas_id[i] shall be in the range of 0 to 63, inclusive.

F.3.8 Geometry assistance SEI payload semantics

The geometry assistance SEI message indicates suggested information associated with each view that can be used to reduce complexity of a depth estimation process occurring at decoder side when a geometry component is not present.

A view is uniformly divided into square blocks, and each block can be further 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.7 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.7 — The different block split types and associated values of the syntax elements

								
gas_split_flag	0	1	1	1	1	1	1	1
gas_quad_split_flag	-	1	0	0	0	0	0	0
gas_split_orientation_flag	-	-	0	0	0	1	1	1
gas_split_symmetry_flag	-	-	1	0	0	1	0	0
gas_split_first_block_bigger	-	-	-	0	1	-	0	1

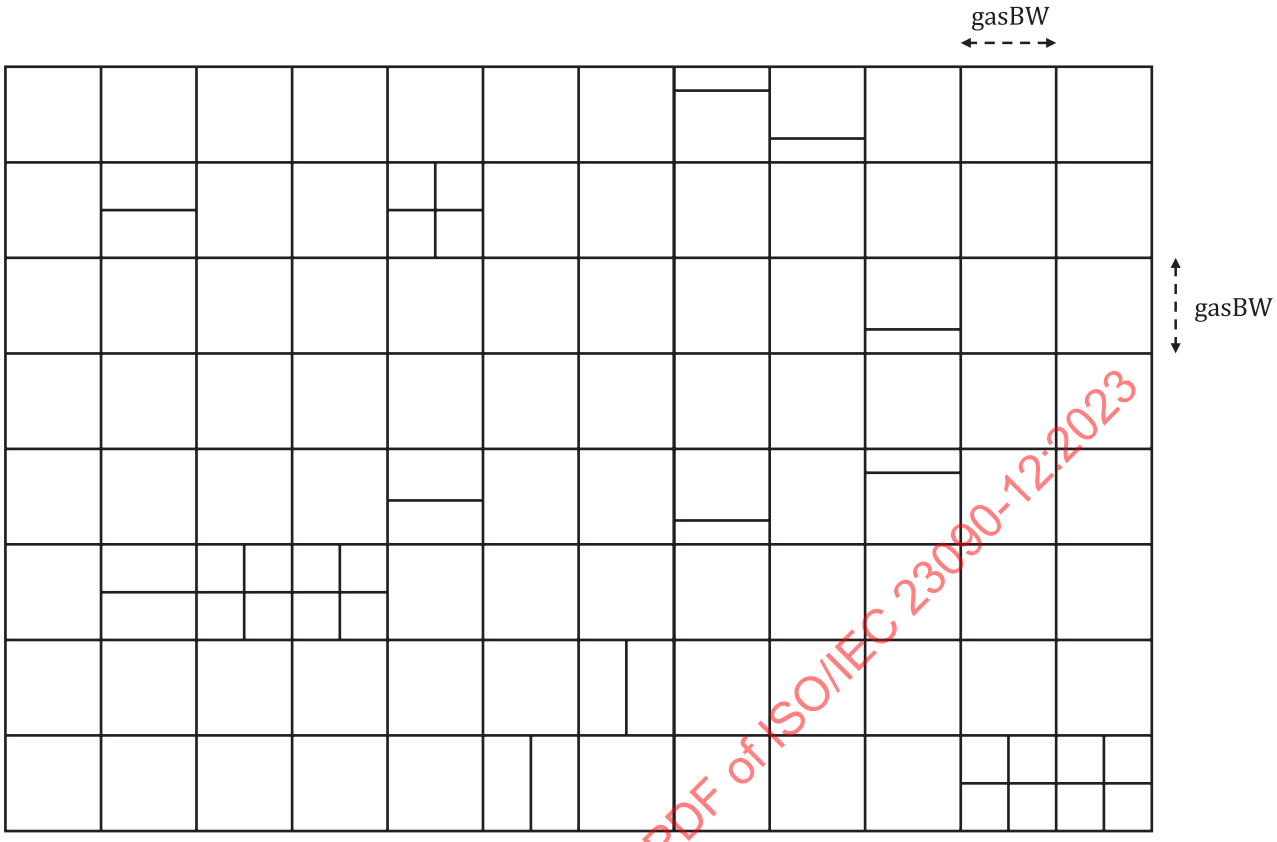


Figure F.1 — Example of a partition of a view into all possible block divisions

gas_qs specifies the quantization step for the suggested geometry range boundaries.

gas_num_views_minus1 plus 1 specifies the number of views for which geometry asssistance parameters are signalled.

gas_log2_bw_minus2 specifies the value of the variable *gasBW*, as shown in Figure F.1, as follows:

$$gasBW = 1 << (gas\_log2\_bw\_minus2 + 2) \quad \text{(F.19)}$$

gas_projection_plane_width_minus1[v] plus 1 and **gas_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.

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

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

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

gas_split_symmetry_flag equal to 0 indicates that the area of the two sublocks differ, with the division occuring at a quarter of the block width from one end. gas_split_symmetry_flag equal to 1 indicates that the area of the two sublocks is equal.

gas_split_first_block_bigger equal to 1 indicates that the first subblock (top subblock if gas_split_orientation_flag is equal to 0, and left subblock if gas_split_orientation_flag is equal to 1) is bigger than

the second subblock. gas_split_first_block_bigger equal to 0 indicates that the first subblock is smaller than the second subblock.

gas_skip_flag equal to 0 indicates that a gas_zmin_delta and a gas_zmax_delta syntax elements are present in the bitstream, and that a gas_ltmin_flag and a gas_ltmax_flag may be present. gas_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.

gas_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.

gas_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.

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

gas_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:

$$ZMin = (LTMinFlag == 2 \ ? \ 0 : LTMinFlag == 1 \ ? \ ZMinTop : ZMinLeft) + gas\_qs * gas\_zmin\_delta \quad (F.20)$$

$$ZMax = (LTMaxFlag == 2 \ ? \ 0 : LTMaxFlag == 1 \ ? \ ZMinTop : ZMinLeft) + gas\_qs * gas\_zmax\_delta \quad (F.21)$$

Annex G
(informative)

Volumetric usability information

The specifications in ISO/IEC 23090-5:—, Annex G apply with the following modifications.

When vui_parameters( ) are present in the CASPS, the syntax elements apply to all atlases present in the coded MIV sequence. If vui_parameters( ) present in both the CASPS and ASPS of a coded MIV sequence, then their contents shall be the same.

In this version of this specification, the value of vui_display_box_info_present_flag and vui_anchor_point_present_flag shall be equal to zero.

Annex H
(Informative)

Overview of the rendering processes

H.1 General

H.1.1 The purpose of Annex H is to provide informative hypothetical transcoding and rendering processes, which operate on the outputs defined in Clause 8, Clause 9, Annex A, and Annex B. An application can combine multiple processes.

— The sample 3D reconstruction process (H.2) specifies how to reconstruct an atlas sample to a 3D point in space. This process is a building block of 6DoF rendering and any transcoding that involves 3D format conversion. The process demonstrates the use of most view parameters including camera intrinsics, camera extrinsics and depth quantization parameters. When applied to all atlas samples in a volumetric frame, a point cloud is reconstructed. Such a representation is useful for rendering applications and immersive video analytics.

— The entity-filtering process (H.3) specifies how to filter a block to patch map by entity ID prior to transcoding or rendering, for instance to render only the foreground objects in a scene.

— The geometry video scaling process (H.5) specifies how to upscale a geometry frame with preservation of thin foreground edges.

— The pruned view reconstruction process (H.6) specifies how to reconstruct a pruned view from multiple atlases. This process enables multiview rendering as opposed to rendering per point or per patch. The process is also useful for transcoding applications that work with views instead of atlases.

— The sample weighting recovery process (H.7) specifies how to use a pruning graph to recover a view blending weight for a sample within a pruned view. It is common for multiview renderers to have a weight per view that depends on the viewport position. This process enables this type of solution when views are pruned.

— The MPI reconstruction process (H.9) specifies how to reconstruct a multi-plane image (MPI), from a decoded volumetric frame with ptc_restricted_geometry_flag equal to 1.

H.1.2 An application can combine all or a subset of the processes, for instance:

a) Scale geometry (H.5)

b) Apply patch attribute offset process (H.4)

c) Filter inpaint patches (H.8)

d) Reconstruct pruned views (H.6)

e) Determine view blending weights based on a viewport pose

f) Recover sample weights (H.7)

g) Reconstruct 3D points (H.2)

h) Project to a viewport

i) Fetch texture from multiple views

j) Blend texture contributions.

H.1.3 Alternatively, an application operating on a bitstream conforming to the geometry absent profile can:

a) Select a subset of views based on a viewport pose,

b) Apply patch attribute offset process (H.4)

c) Reconstruct pruned views (H.6)

d) Determine view blending weights based on a viewport pose

e) Recover sample weights (H.7)

f) Reconstruct geometry of the subset of views using an estimator

g) Reproject the subset of views to a viewport (H.2)

h) Fetch texture from multiple views

i) Blend texture contributions.

H.1.4 Alternatively, an application operating on a bitstream with ptc_restricted_geometry_flag equal to 1 can:

a) Reconstruct the MPI (H.9)

b) Project to a viewport and blend the different texture layers from the closest to the farthest, taking into account the associated transparency values (reverse painter's algorithm).

H.2 Sample 3D reconstruction process

H.2.1 General

This process specifies how to reconstruct an atlas sample to a 3D point in space. This process is a building block of 6DoF rendering and any transcoding that involves 3D format conversion. The process demonstrates the use of most view parameters including camera intrinsics, camera extrinsics and depth quantization parameters. When applied to all atlas samples, a point cloud is reconstructed. Such a representation is useful for rendering applications and immersive video analytics.

Inputs to this process are:

— the variables *u* and *v* calculated by Formula (52) of ISO/IEC 23090-5:—,

— when decoded (or unpacked) occupancy video data is present:

  — the variable *occSample*, the decoded (and interpolated) occupancy value of the sample,

— when decoded (or unpacked) geometry video data is present:

  — the variable *geoSample*, the decoded (and interpolated) geometry value of the sample,

— the variable *patchIdx*, the atlas patch index of the sample.

Outputs of this process are:

— the variable *sampleOccFlag*, indicating occupancy,

— when *sampleOccFlag* is equal to 1:

  — the variables *sampleX*, *sampleY* and *sampleZ*, corresponding to the Cartesian coordinates (x, y, z) of the point in scene coordinates.

This process has the form of a sequence of processes:

a) Sample occupancy reconstruction process (H.2.2).

b) When *sampleOccFlag* is equal to 1:

  1) Depth value clamping (H.2.3),

  2) Depth expansion (H.2.4), applying depth quantization parameters,

  3) Coordinate unprojection (H.2.5), applying camera intrinsics,

  4) Change of reference frame (H.2.6), applying camera extrinsics,

  5) Change of coordinate system, applying VUI.

  NOTE In the case of the geometry absent profile, the depth value clamping and depth decoding process may be replaced by a depth sample estimation process that is out of the scope of this version of this document.

H.2.2 Sample occupancy reconstruction process

This process reconstructs the occupancy of an atlas sample.

Inputs to this process are:

— the variable *patchIdx*, corresponding to the atlas patch index of the atlas sample,

— the variable *atlasID*, corresponding to the atlas ID of the atlas sample,

— the variable *occSample*, the decoded (and interpolated) occupancy value of the atlas sample, when present,

— the variable *geoSample*, the decoded (and interpolated) geometry value of the atlas sample, when present.

The output of this process is:

— the variable *sampleOccFlag*, indicating sample occupancy.

The occupancy video reconstruction process operates as follows:

```
   if( patchIdx != 0 )
      sampleOccFlag = 0
   else if( vps_occupancy_video_present_flag[ atlasID ] ||
            pin_occupancy_present_flag[ atlasID ] )
      sampleOccFlag = occSample
   else if( vme_embedded_occupancy_enabled_flag ) {
      if( AsmeDepthOccThresholdFlag[ atlasID ] )
        t = AtlasPatchDepthOccThreshold[ patchIdx ]
      else {
        v = AtlasPatchViewIndex[ patchIdx ]
        t = ViewOccThreshold[ v ]
      }
      sampleOccFlag = t <= geoSample ? 1 : 0
   } else
      sampleOccFlag = 1
}
```

H.2.3 Depth clamping process

This process clamps the depth value to be within the integer range of the atlas patch.