A picture containing drawing, food

Description automatically generated

**ISO/IEC JTC 1/SC 29/WG 11**

**Coding of moving pictures and audio**

**Convenorship: UNI (Italy)**

**ISO/IEC JTC 1/SC 29/WG 11 N19001**

**Document type: Approved WG 11 document**

**Title: Working Draft 4 of Immersive Video**

**Status: Output document**

**Date of document: 2020-02-28**

**Source: Video**

**Expected action:**

**No. of pages: -**

**Email of convenor: leonardo@chiariglione.org**

**Committee URL: mpeg.chiariglione.org**

**Ed. notes:**

**WD4: (This version is a complete rewrite, in order to align with 3VC specification)**

* m52365: depth-map scaling for pixel-rate reduction.
* m52366: viewing-space handling for MIV and VPCC.
* m52556: 6DoF recommended viewport SEI message for MIV and V-PCC.
* m52429: auxilliary patches.
* m52876: adding coordinate axis system VUI to MIV WD4.

**WD3:**

* m49958: atlas groups
* m59049: entity\_id per patch
* m50815+m50948 (=m51439): depth-occupancy multiplexing
* errata: bug fixes
* m51487: viewing space

**WD2:**

* m49228: omaf v1 compatible flag
* m49230: full range and luma for depth
* m49343: IV sequence, IV access unit, and atlas list parameters syntax
* m49230: restrict to equirectangular and perspective projection (remove CMP)

**ISO/IEC JTC 1/SC 29/WG 11 N19001**

ISO/IEC JTC 1/SC 29/WG 11

Secretariat: JISC

**Information technology — Coded Representation of Immersive Media — Part 12: Immersive Video**

**Warning for WDs and CDs**

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

Recipients of this draft are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

© ISO/IEC 2019

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO’s member body in the country of the requester.

ISO copyright office

CP 401 • Ch. de Blandonnet 8

CH-1214 Vernier, Geneva

Phone: +41 22 749 01 11

Fax: +41 22 749 09 47

Email: copyright@iso.org

Website: www.iso.org

Published in Switzerland

Contents

[Foreword vii](#_Toc34409219)

[Introduction viii](#_Toc34409220)

[1 Scope 1](#_Toc34409221)

[2 Normative reference 1](#_Toc34409222)

[3 Terms and definitions 1](#_Toc34409223)

[4 Abbreviations 4](#_Toc34409224)

[5 Conventions 4](#_Toc34409225)

[6 Bitstream format, partitioning, and scanning processes 4](#_Toc34409226)

[6.1 Bitstream formats 4](#_Toc34409227)

[6.2 NAL bitstream formats 4](#_Toc34409228)

[6.3 Partitioning of atlas frames, tiles, and tile groups 4](#_Toc34409229)

[6.4 Scanning Processes 4](#_Toc34409230)

[6.5 Rendering 4](#_Toc34409231)

[6.6 Patches, atlases, block to patch map, view representations, and view representation pairs relationships 5](#_Toc34409232)

[6.7 Reference Architecture 5](#_Toc34409233)

[6.8 Sources and outputs 6](#_Toc34409234)

[7 Syntax and semantics 6](#_Toc34409235)

[7.1 Method of specifying syntax in tabular form 6](#_Toc34409236)

[7.2 Specification of syntax functions and descriptors 7](#_Toc34409237)

[7.3 Syntax in tabular form 7](#_Toc34409238)

[7.4 Semantics 15](#_Toc34409239)

[8 Decoding process 26](#_Toc34409240)

[8.1 General decoding process 26](#_Toc34409241)

[8.2 Occupancy video decoding process 27](#_Toc34409242)

[8.3 Geometry video decoding process 27](#_Toc34409243)

[8.4 Attribute(s) video decoding process 27](#_Toc34409244)

[8.5 Atlas data decoding process 28](#_Toc34409245)

[9 Reconstruction process 29](#_Toc34409246)

[10 Parsing process 29](#_Toc34409247)

[11 Sub-bitstream extraction process 29](#_Toc34409248)

[Annex A Profiles, tiers, and levels 30](#_Toc34409249)

[A.1 Overview of profiles, tiers, and levels 30](#_Toc34409250)

[A.2 V-PCC profile, tier and level structure 30](#_Toc34409251)

[A.3 V-PCC CodecGroup profile components 30](#_Toc34409252)

[A.4 V-PCC toolset profile components 30](#_Toc34409253)

[A.4.1 MIV Main toolset profile component 30](#_Toc34409254)

[A.5 Tiers and Levels 31](#_Toc34409255)

[A.5.1 MIV-specific level limits 31](#_Toc34409256)

[Annex B Post-decoding conversion to nominal video formats 34](#_Toc34409257)

[Annex C V-PCC Sample stream format 35](#_Toc34409258)

[Annex D NAL Sample stream format 36](#_Toc34409259)

[Annex E Atlas Hypothetical Reference Decoder 37](#_Toc34409260)

[Annex F Supplemental enhancement information 38](#_Toc34409261)

[F.1 General 38](#_Toc34409262)

[F.2 SEI payload syntax 38](#_Toc34409263)

[F.2.1 General SEI message syntax 38](#_Toc34409264)

[F.2.2 Viewing space SEI message syntax 39](#_Toc34409265)

[F.2.2.1 General 39](#_Toc34409266)

[F.2.2.2 Elementary shape 40](#_Toc34409267)

[F.2.2.3 Cuboid primitive 41](#_Toc34409268)

[F.2.2.4 Spheroid primitive 41](#_Toc34409269)

[F.2.2.5 Half space primitive 41](#_Toc34409270)

[F.2.3 Recommended viewport SEI message syntax 42](#_Toc34409271)

[F.2.4 Viewing space handling SEI payload syntax 42](#_Toc34409272)

[F.3 SEI payload semantics 42](#_Toc34409273)

[F.3.1 General SEI message semantics 42](#_Toc34409274)

[F.3.2 Viewing space SEI message semantics 42](#_Toc34409275)

[F.3.2.1 General 42](#_Toc34409276)

[F.3.2.2 Elementary shape 43](#_Toc34409277)

[F.3.2.3 Cuboid primitive 45](#_Toc34409278)

[F.3.2.4 Sheroid primitive 46](#_Toc34409279)

[F.3.2.5 Half space primitive 46](#_Toc34409280)

[F.3.3 Recommended viewport SEI message semantics 46](#_Toc34409281)

[F.3.4 Viewing space handling SEI payload semantics 47](#_Toc34409282)

[Annex G Hypothetical view rendering process (Informative) 50](#_Toc34409283)

[G.1 General 50](#_Toc34409284)

[G.2 Geometry video scaling process 51](#_Toc34409285)

[G.2.1 Nearest neighbour interpolation scaling process 52](#_Toc34409286)

[G.2.2 Sample neighbours enumeration process 52](#_Toc34409287)

[G.2.3 Foreground edge flag process 53](#_Toc34409288)

[G.2.4 Selective geometry erosion process 53](#_Toc34409289)

[G.2.5 Texture aligned geometry erosion process 53](#_Toc34409290)

[G.2.6 Geometry contour smoothening process 54](#_Toc34409291)

[G.3 Depth decoding process 55](#_Toc34409292)

[G.4 Reconstruction of reconstructed views process 56](#_Toc34409293)

[G.5 Projection of pixels of reconstructed views to viewport 56](#_Toc34409294)

[G.5.1 Group-based initialization process 56](#_Toc34409295)

[G.5.2 Group-based merging process 56](#_Toc34409296)

[G.5.3 General projection of pixels of reconstructed views to viewport process 57](#_Toc34409297)

[G.5.4 Local views coordinate to global coordinate point unprojection process 57](#_Toc34409298)

[G.5.4.1 ERP unprojection process 58](#_Toc34409299)

[G.5.4.2 Perspective unprojection process 58](#_Toc34409300)

[G.5.4.3 Orthographic unprojection process 59](#_Toc34409301)

[G.5.5 Pixel weighting recovery process 59](#_Toc34409302)

[Bibliography 61](#_Toc34409303)

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

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 ISO documents 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](https://www.iso.org/directives-and-policies.html)).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see [www.iso.org/patents](https://www.iso.org/iso-standards-and-patents.html)).

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](https://www.iso.org/foreword-supplementary-information.html).

This document was prepared by Subcommittee 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 website.

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](https://www.iso.org/members.html).

Introduction

This International Standard was developed to support compression of immersive video content, in which a real or virtual 3-D scene is captured by multiple real or virtual cameras. The use of this standard 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: Immersive Video**

# Scope

The document specifies the syntax, semantics and decoding processes for MPEG Immersive Video (MIV). 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). This document specifies the MIV extension of [3VC].

# Normative reference

The following normative references apply in addition to the normative references in [3VC] clause 2.

[Ed. (JB): Check all V-PCC section numbering and naming]

|  |  |
| --- | --- |
| [OMAF] | CD of ISO/IEC 23090-2 2nd edition OMAF, N18865, 11th October 2019 |
| [3VC] | draft DIS of ISO/IEC 23090-5 Video-based Point Cloud Compression, N18888, 11th October 2019 |
| [IEEE754] | IEEE Standard for Floating-Point Arithmetic (IEEE 754-2019) |

# Terms and definitions

The following definitions apply in addition to the definitions in [3VC] clause 3. These definitions are either not present in or replace definitions in [3VC] clause 3.

3D scene

visual content in the *global reference coordinate system*

associated atlas

The VPCC\_AD *V-PCC unit* with the same value of vuh\_atlas\_id as the current VPCC\_OVD, VPCC\_GVD, or VPCC\_AVD *V-PCC unit*

atlas

a collection of 2D bounding boxes, i.e. patches, projected into a rectangular frame that correspond to a 3-dimensional bounding box in 3D space, which may represent a subset of the *3D scene*

atlas data

data that is needed to perform the transformation of patches included in an atlas from 2D to 3D space

attribute

scalar or vector property optionally associated with each point in *3D scene* such as colour, reflectance, surface normal, time stamps, material ID, etc.

attribute component

2D array of *attribute* values output by the *planar projection* process

camera parameters

parameters determining the relation between the *3D scene* points and the pixels in the *planar projection*

field of view

the extent of the observable world in captured/recorded content or in a physical display device

geometry

set of Cartesian coordinates associated with a point of the *3D scene*

geometry component

2D array of *depth* values output by the *planar* *projection* process

global coordinates axes

coordinate axes that are associated with audio, video, and images representing the same acquisition position and intended to be rendered together

global reference coordinate system

a 3D coordinate system using *global coordinate axes*, in units of meters

hypothetical view renderer

a hypothetical *renderer* model that outputs a *viewport*

local coordinate axes

coordinate axes that are associated with audio, video, and images of a specific *view*, meaning that the *viewing position* is a tuple of zeros (the origin) and the *viewing orientation* is a tuple of zero angles (upright and forward)

local coordinate system

a 3D coordinate system using *local coordinate axes*, in units of meters

occupancy map

a 2D array corresponding to an atlas whose values indicate for each sample position in the *atlas* whether that position corresponds to a valid 3D point in the *3D scene*

patch

rectangular region within an *atlas* that corresponds to a rectangular region within a *view representation*

planar projection

inverse of the process by which the sample values of a projected *attribute* *component* picture of a *view representation* are mapped to a set of positions in a *3D scene* represented in the *global reference coordinate system* according to the corresponding sample values of the *geometry component* picture and *camera parameter*s *list*

renderer

an embodiment of a process to create a *viewport* from a *3D scene* representation corresponding to a *viewing orientation* and *viewing position*

source

a term used to describe the video material or some of its attributes before encoding

source view representation

a term used to describe *source* video material before encoding that corresponds to the format of a *view representation*, which may have been acquired by capture of a *3D scene* by a real camera or by *projection* by a virtual camera onto a surface using *camera parameters*

view parameters

defines the projection used to generate a *view representation* from a *3D scene*, including intrinsic and extrinsic parameters

view parameters list

a list of one or more *view parameters*

viewing orientation

triple of quat\_x, quat\_y, quat\_z characterizing the quaternion representation of the orientation that a user is consuming the audio-visual content; in case of image or video, characterizing the orientation of the *viewport*

viewing position

triple of x, y, z characterizing the position in the *global reference coordinate system* of a user who is consuming the audio-visual content; in case of image or video, characterizing the position of the *viewport*



view representation

2D sample arrays of *attribute component* and corresponding *geometry component* representing the projection of a *3D scene* onto a surface using *camera parameters*

viewing space

domain constraints for a good *viewport* rendering; the domain is defined in the 3D global space and related to the *viewing direction*; it defines a scale between 0 and 1 for every point in space for a given direction of the viewport, to be used by the application

viewport

projection of texture onto a planar surface of a field of view of an omnidirectional or 3D image or video suitable for display and viewing by the user with a particular *viewing orientation* and *viewing position*

V-PCC access unit

a set of *V-PCC units* that are associated with each other according to a specified classification rule, and are consecutive in decoding order, and contain all atlas *NAL units* and coded pictures in *video sub-streams* as well as other *V-PCC components* pertaining to one particular output time

IRAP V-PCC access unit

A *V-PCC access unit* for which all *coded atlas access units* are *IRAP coded atlas access units*, and all coded pictures in *video sub-streams* are of type IRAP

# Abbreviations

The following abbreviations apply in addition to the abbreviations in [3VC] clause 4.

|  |  |
| --- | --- |
| CVS | Coded Video Sequence |
| ERP | EquiRectangular Projection |
| FOV | Field of View |
| HEVC | High Efficiency Video Coding |
| HMD | Head-Mounted Display |
| OMAF  PSP | Omnidirectional MediA Format (specified in [OMAF])  Perspective Projection |

# Conventions

The specifications in [3VC] clause 5 and its subclauses apply.

# Bitstream format, partitioning, and scanning processes

## Bitstream formats

The specifications in [3VC] clause 6.1 apply.

## NAL bitstream formats

The specifications in [3VC] clause 6.2 apply.

## Partitioning of atlas frames, tiles, and tile groups

The specifications in clause 6.3 of [3VC] and its subclauses apply.

## Scanning Processes

The specifications in clause 6.4 of [3VC] and its subclauses apply.

## Rendering

Annex G provides an informative hypothetical view rendering process, which operates on the outputs defined in clause 6.8, for generating a view for display.

Other rendering processes may be performed using the outputs of clause 6.8, i.e., a 3D scene reconstructor can reconstruct a 3D video sequence represented in the source format, for use in display or other purposes, such as transcoding.

## Patches, atlases, block to patch map, view representations, and view representation pairs relationships

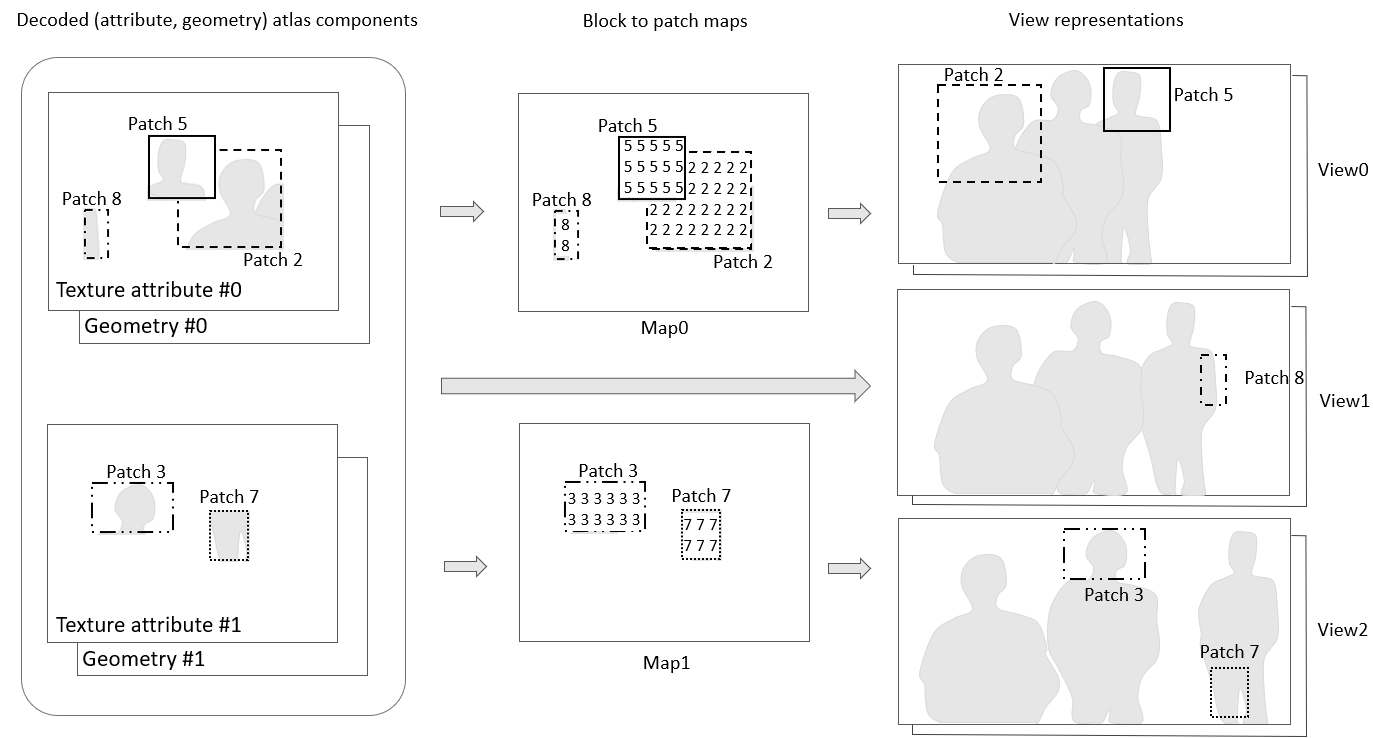


Figure 1: Example mapping of 5 patches in 2 atlases to 3 view representations

A patch is a rectangular region that is represented in both an atlas and a view representation. In this version of the specification, the size of all patches is the same in both the atlas representation and the view representation for the texture attribute. It may be equal or different for the geometry.

An atlas contains an aggregation of one or more patches from one or more view representations, with at least a corresponding geometry component and optionally a texture component. The block to patch map generator process specified in clause 8.5 outputs a map of values for each block of pixels, of the same size as the atlas divided by the size of the block. Each map value indicates the index of the patch to which all co-located samples in the atlas correspond when not equal to 0xFFFF. Map values equal to 0xFFFF indicates unused part of the atlas.

A view representation represents a field of view of a 3D scene for particular camera parameters, for the texture attribute and geometry components. View representations may be omnidirectional or perspective, and may use different projection formats, such as equirectangular projection as defined in [OMAF] or cube map projection through multiple perspective projections or orthographic as defined in [3VC]. In this version of the specification, the texture attribute and geometry components of a view representation use the same projection format.

Figure 1 shows an illustrative example, in which two atlases contain 5 patches, which are mapped to 3 view representations.

## Reference Architecture

The reference architecture is illustrated in Figure 2**.**

A CVS for each of the video-substreams for the geometry component and optionally the texture attribute component is input to a video decoder, which outputs a sequence of decoded picture pairs of synchronized decoded geometry pictures (A) and decoded texture attribute pictures (B). Geometry and texture attribute may have different resolutions.

The metadata is input to a metadata parser which outputs Atlas Data (C) which includes the information of the patch list and the camera parameters list, and the general information of the V-PCC parameter set (D).

The block to patch map generator, specified in clause 8.5, takes as inputs the Atlas Data (C), which includes the information of the patch list, and the general information of V-PCC parameter set (D) and outputs an block to patch map (E).

In the reference architecture, a view renderer take as inputs one or more pairs of decoded geometry component atlases (A) – possibly upscaled – and texture attribute component atlases (B) , the Atlas Data (C) , the block to patch map sequence (E), and the viewer position and orientation, and outputs a viewport.

The reference view renderer is not defined in this specification but a hypothetical view renderer, including the geometry scaler module, is described in Annex G.

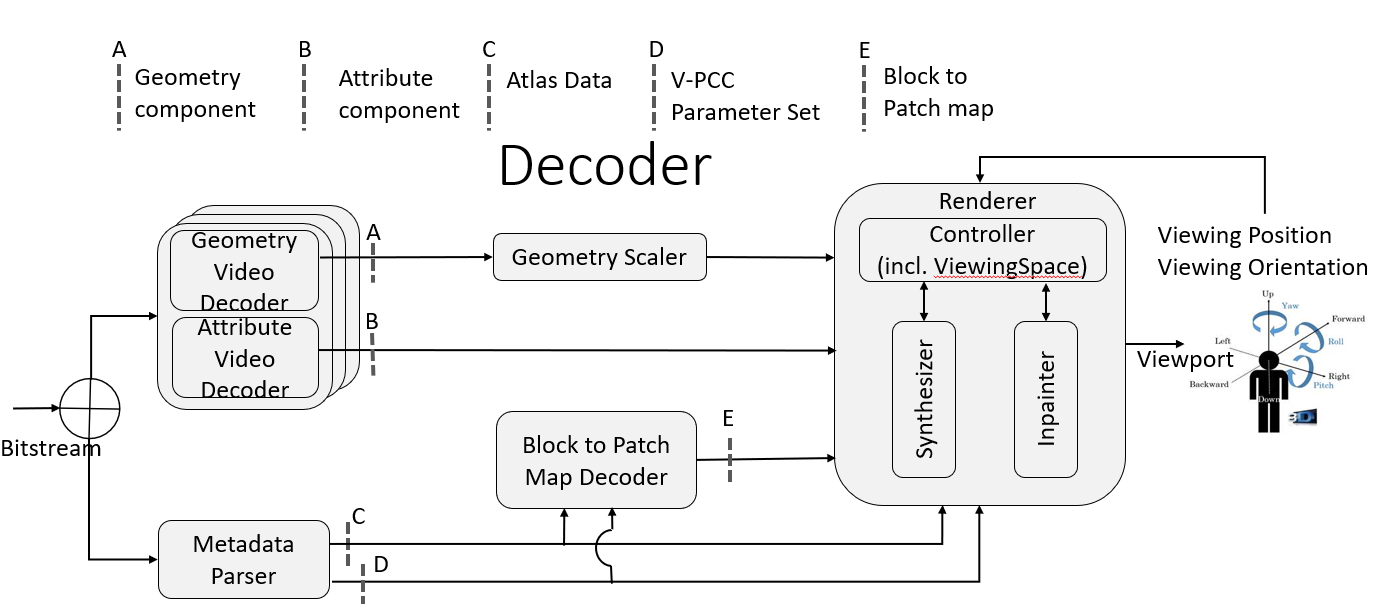


Figure 2: Reference architecture of the immersive video decoder

## Sources and outputs

The immersive video source that is represented by the bitstream is one or more independently coded sequence pairs of texture attribute and geometry pictures. Each of the sequence pairs represents a view of a 3D scene, which may have been captured by a real camera or generated by a virtual camera, with the texture attribute and geometry picture time aligned. The pictures in all views are time aligned, when present.

The outputs are a camera parameters list, and for each of one or more atlases the following: a sequence of decoded picture pairs with a texture attribute component and a geometry component, a sequence of atlas parameters, and a sequence of block to patch maps, as described in clause 8.1.

# Syntax and semantics

## Method of specifying syntax in tabular form

The specifications in [3VC] clause 7.1 apply.

## Specification of syntax functions and descriptors

The specifications in [3VC] clause 7.2 apply.

## Syntax in tabular form

[Ed. (ALL): There are limited syntax changes in here that are expected to be in the 3VC spec, but not included in current available 3VC draft.]

### General

The specifications in [3VC] clause 7.3.1 apply.

### V-PCC unit syntax

The specifications in [3VC] clause 7.3.2 apply.

### Byte alignment syntax

The specifications in [3VC] clause 7.3.3 apply.

### Sequence parameter set syntax

The specifications in [3VC] clause 7.3.4 and its subclauses apply.

#### General V-PCC parameter set syntax

|  |  |
| --- | --- |
| vpcc\_parameter\_set( ) { | **Descriptor** |
| profile\_tier\_level() |  |
| **vps\_vpcc\_parameter\_set\_id** | u(4) |
| **vps\_miv\_mode\_flag** [Ed. (ALL): Discuss if this flag is needed in the next meeting.] | u(1) |
| **vps\_reserved\_zero\_7bits** | u(7) |
| **vps\_atlas\_count\_minus1** | u(6) |
| for( j = 0; j < vps\_atlas\_count\_minus1 + 1; j++ ) { |  |
| **vps\_frame\_width**[ j ] | u(16) |
| **vps\_frame\_height**[ j ] | u(16) |
| **vps\_map\_count\_minus1**[ j ] | u(4) |
| if( vps\_map\_count\_minus1[ j ] > 0 ) |  |
| **vps\_multiple\_map\_streams\_present\_flag**[ j ] | u(1) |
| vps\_map\_absolute\_coding\_enabled\_flag[ j ][ 0 ] = 1 |  |
| for( i = 1; i <= vps\_map\_count\_minus1[ j ]; i++ ) { |  |
| if( vps\_multiple\_map\_streams\_present\_flag[ j ] ) |  |
| **vps\_map\_absolute\_coding\_enabled\_flag**[ j ][ i  ] | u(1) |
| else |  |
| vps\_map\_absolute\_coding\_enabled\_flag[ j ][ i  ] = 1 |  |
| if( vps\_map\_absolute\_coding\_enabled\_flag[ j ][ i  ]  = =  0 ) { |  |
| if( i > 0) |  |
| **vps\_map\_predictor\_index\_diff**[ j ][ i ] | ue(v) |
| else |  |
| vps\_map\_predictor\_index\_diff[ j ][ i ] = 0 |  |
| } |  |
| } |  |
| **vps\_auxiliary\_video\_present\_flag**[ j ] | u(1) |
| if ( !vps\_miv\_mode\_flag ) |  |
| occupancy\_information( j ) |  |
| geometry\_information( j ) |  |
| attribute\_information( j ) |  |
| } |  |
| **vps\_extension\_bit\_equal\_to\_one** | f(1) |
| **vps\_extension\_length\_minus1** | ue(v) |
| **vps\_miv\_extension\_flag** | u(1) |
| if( vps\_miv\_extension\_flag ) { |  |
| miv\_sequence\_params( ) |  |
| **vps\_miv\_sequence\_vui\_params\_present\_flag** | u(1) |
| if( vps\_miv\_sequence\_vui\_params\_present\_flag ) |  |
| miv\_vui\_parameters( ) |  |
| } |  |
| while( more\_data\_in\_miv\_extension( ) ) |  |
| **vps\_miv\_extension\_data\_flag** | u(1) |
| byte\_alignment( ) |  |
| } |  |

#### Profile, tier, and level syntax

The specifications in [3VC] clause 7.3.4.2 apply.

#### Occupancy information syntax

This clause does not apply to this version of the specification.

#### Geometry information syntax

The specifications in [3VC] clause 7.3.4.4 apply.

#### Attribute information syntax

The specifications in [3VC] clause 7.3.4.5 apply.

#### MIV sequence parameters (MSP)

|  |  |
| --- | --- |
| miv\_sequence\_params( ) { | **Descriptor** |
| **msp\_depth\_low\_quality\_flag** | u(1) |
| **msp\_geometry\_scale\_enabled\_flag** | u(1) |
| **msp\_num\_groups\_minus1** | ue(v) |
| **msp\_max\_entities\_minus1** | ue(v) |
| } |  |

[Ed. (MV): Adhoc group meeting on 10-2-2020 decided that there is no need for a special\_atlas flag]

#### MIV VUI parameters syntax

|  |  |
| --- | --- |
| miv\_vui\_params( ) { | **Descriptor** |
| coordinate\_axis\_system\_params( ) |  |
| } |  |

#### Coordinate axis system parameters syntax

|  |  |
| --- | --- |
| coordinate\_axis\_system\_params( ) { | **Descriptor** |
| **cas\_forward\_axis** | u(2) |
| **cas\_delta\_left\_axis\_minus1** | u(1) |
| **cas\_forward\_sign** | u(1) |
| **cas\_left\_sign** | u(1) |
| **cas\_up\_sign** | u(1) |
| } |  |

### NAL unit syntax

The specifications in [3VC] clause 7.3.5 and its subclauses apply.

### Atlas sequence, frame, and tile group parameter set syntax

The specifications in [3VC] clause 7.3.6 and its subclauses apply unless overridden by this specification.

#### Atlas sequence parameter set RBSP syntax

|  |  |
| --- | --- |
| atlas\_sequence\_parameter\_set\_rbsp( ) { | **Descriptor** |
| **asps\_atlas\_sequence\_parameter\_set\_id** | ue(v) |
| **asps\_frame\_width** | u(16) |
| **asps\_frame\_height** | u(16) |
| **asps\_log2\_patch\_packing\_block\_size** | u(3) |
| **asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4** | ue(v) |
| **asps\_max\_dec\_atlas\_frame\_buffering\_minus1** | ue(v) |
| **asps\_long\_term\_ref\_atlas\_frames\_flag** | u(1) |
| **asps\_num\_ref\_atlas\_frame\_lists\_in\_asps** | ue(v) |
| for( i = 0; i < asps\_num\_ref\_atlas\_frame\_lists\_in\_asps; i++ ) |  |
| ref\_list\_struct( i ) |  |
| **asps\_use\_eight\_orientations\_flag** | u(1) |
| **asps\_extended\_projection\_enabled\_flag** | u(1) |
| if( asps\_extended\_projection\_enabled\_flag ) |  |
| **asps\_max\_projections\_minus1** | ue(v) |
| **asps\_normal\_axis\_limits\_quantization\_enabled\_flag** | u(1) |
| **asps\_normal\_axis\_max\_delta\_value\_enabled\_flag** | u(1) |
| **asps\_remove\_duplicate\_point\_enabled\_flag** | u(1) |
| **asps\_pixel\_deinterleaving\_enabled\_flag** | u(1) |
| **asps\_patch\_precedence\_order\_flag** | u(1) |
| **asps\_patch\_size\_quantizer\_present\_flag** | u(1) |
| **asps\_raw\_patch\_enabled\_flag** | u(1) |
| **asps\_eom\_patch\_enabled\_flag** | u(1) |
| if (asps\_raw\_patch\_enabled\_flag || asps\_eom\_patch\_enabled\_flag ) |  |
| **asps\_auxiliary\_video\_enabled\_flag** | u(1) |
| **asps\_point\_local\_reconstruction\_enabled\_flag** | u(1) |
| **asps\_map\_count\_minus1** | u(4) |
| if( asps\_pixel\_deinterleaving\_enabled\_flag ) |  |
| for( j = 0; j < = vpcc\_map\_count\_minus1; j+ +) |  |
| **asps\_pixel\_deinterleaving\_map\_flag**[ j ] | u(1) |
| if( asps\_enhanced\_occupancy\_map\_for\_depth\_flag && asps\_map\_count\_minus1 = = 0 ) |  |
| **asps\_enhanced\_occupancy\_map\_fix\_bit\_count\_minus1** | u(4) |
| if( asps\_point\_local\_reconstruction\_enabled\_flag ) |  |
| asps\_point\_local\_reconstruction\_information( asps\_map\_count\_minus1 ) |  |
| if( asps\_pixel\_deinterleaving\_enabled\_flag || asps\_point\_local\_reconstruction\_enabled\_flag ) |  |
| **asps\_surface\_thickness\_minus1** | u(8) |
| **asps\_vui\_parameters\_present\_flag** | u(1) |
| if( asps\_vui\_parameters\_present\_flag ) |  |
| vui\_parameters( ) |  |
| **asps\_extension\_bit\_equal\_to\_one** | f(1) |
| **asps\_miv\_extension\_present\_flag** | u(1) |
| if( asps\_miv\_extension\_present\_flag ) |  |
| miv\_atlas\_sequence\_params( vuh\_atlas\_id ) |  |
| **asps\_extension2\_present\_flag** | u(1) |
| if( asps\_extension2\_present\_flag ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **asps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Point local reconstruction information syntax

This clause does not apply to this version of the specification.

#### Atlas frame parameter set RBSP syntax

The specifications in [3VC] clause 7.3.6.3 apply.

#### Atlas frame tile information syntax

The specifications in [3VC] clause 7.3.6.4 apply.

#### Atlas camera parameter set RBSP syntax

This clause does not apply to this version of the specification.

#### Supplemental enhancement information RBSP syntax

The specifications in [3VC] clause 7.3.6.6 apply.

#### Access unit delimiter RBSP syntax

The specifications in [3VC] clause 7.3.6.7 apply.

#### End of sequence RBSP syntax

The specifications in [3VC] clause 7.3.6.8 apply.

#### End of bitstream RBSP syntax

The specifications in [3VC] clause 7.3.6.9 apply.

#### Filler data RBSP syntax

The specifications in [3VC] clause 7.3.6.10 apply.

#### Atlas tile group layer RBSP syntax

The specifications in [3VC] clause 7.3.6.11 apply.

#### Atlas tile group header syntax

The specifications in [3VC] clause 7.3.6.12 apply.

#### Reference list structure syntax

The specifications in [3VC] clause 7.3.6.13 apply.

#### MIV atlas sequence parameters syntax

|  |  |
| --- | --- |
| miv\_atlas\_sequence\_params( vuh\_atlas\_id ) { | **Descriptor** |
| if( vuh\_atlas\_id == 0 ) |  |
| **masp\_omaf\_v1\_compatible\_flag** | u(1) |
| **masp\_group\_id** | u(v) |
| **masp\_auxiliary\_atlas\_flag** | u(1) |
| **masp\_depth\_occ\_threshold \_flag** | u(1) |
| if( msp\_geometry\_scale\_enabled\_flag == 1 ) { |  |
| **masp\_geometry\_frame\_scale\_factor\_x\_minus1** | ue(v) |
| **masp\_geometry\_frame\_scale\_factor\_y\_minus1** | ue(v) |
| } |  |
| } |  |

#### Frame order count RBSP syntax

### [Ed. (MV): The clause numbering has diverged from the current draft of the V3C specification]

|  |  |
| --- | --- |
| frame\_order\_count\_rbsp( ) { | **Descriptor** |
| **frm\_order\_cnt\_lsb** | u(v) |
| rbsp\_trailing\_bits |  |
| } |  |

### Atlas tile group data unit syntax

#### General atlas tile group data unit syntax

The specifications in [3VC] clause 7.3.7.1 apply.

#### Patch information data syntax

The specifications in [3VC] clause 7.3.7.2 apply.

#### Atlas patch data unit syntax

|  |  |
| --- | --- |
| patch\_data\_unit( patchIdx) { | **Descriptor** |
| **pdu\_2d\_pos\_x**[ patchIdx ] | ue(v) |
| **pdu\_2d\_pos\_y**[ patchIdx ] | ue(v) |
| **pdu\_2d\_size\_x\_minus1**[ patchIdx ] | ue(v) |
| **pdu\_2d\_size\_y\_minus1**[ patchIdx ] | ue(v) |
| **pdu\_view\_pos\_x**[ patchIdx ] /\* new semantics for pdu\_3d\_pos\_x \*/ | u(v) |
| **pdu\_view\_pos\_y**[ patchIdx ] /\* new semantics for pdu\_3d\_pos\_y \*/ | u(v) |
| **pdu\_depth\_start**[ patchIdx ] /\* new semantics for pdu\_3d\_pos\_min\_z \*/ | u(v) |
| if( asps\_normal\_axis\_max\_delta\_value\_enabled\_flag ) |  |
| **pdu\_depth\_end**[ patchIdx ] /\* new semantics for pdu\_3d\_pos\_delta\_max\_z \*/ | u(v) |
| **pdu\_view\_id**[ patchIdx ] /\* new semantics for pdu\_projection\_id \*/ | u(v) |
| **pdu\_orientation\_index**[ patchIdx ] | u(v) |
| if( afps\_lod\_mode\_enabled\_flag ) { |  |
| **pdu\_lod\_enabled\_flag**[ patchIndex ] | u(1) |
| if( pdu\_lod\_enabled\_flag[ patchIndex ] > 0 ) { |  |
| **pdu\_lod\_scale\_x\_minus1**[ patchIndex ] | ue(v) |
| **pdu\_lod\_scale\_y**[ patchIndex ] | ue(v) |
| } |  |
| } |  |
| if( asps\_point\_local\_reconstruction\_enabled\_flag ) |  |
| point\_local\_reconstruction\_data( patchIdx ) |  |
| if( vps\_miv\_extension\_flag ) { |  |
| if( msp\_max\_entities\_minus1 > 0 ) |  |
| **pdu\_entity\_id**[patchIdx ] | u(v) |
| if(masp\_depth\_occ\_threshold\_flag) |  |
| **pdu\_depth\_occ\_threshold**[ patchIdx ] | u(v) |
| } |  |
| } |  |

### Supplemental enhancement information message syntax

The specifications in [3VC] clause 7.3.8 apply.

### Atlas Adaptation parameters set RBSP syntax

[Ed. (ALL): MIV extension of a syntax structure expected to be in 3VC spec.]

|  |  |
| --- | --- |
| atlas\_adaptation\_parameter\_set\_rbsp( ) { | **Descriptor** |
| **aaps\_adaptation\_parameter\_set**\_**id** | ue(v) |
| **aaps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4** | ue(v) |
| **aaps**\_**camera\_params\_present\_flag** | u(1) |
| if( aaps\_camera\_params\_present\_flag ) |  |
| camera\_params( ) |  |
| **aaps\_extension\_bit\_equal\_to\_one** | f(1) |
| **aaps\_miv\_view\_params\_list\_present\_flag** | u(1) |
| if( aaps\_miv\_view\_params\_list\_present\_flag) { |  |
| **aaps\_miv\_view\_params\_list\_update\_mode** | u(2) |
| if( aaps\_miv\_view\_params\_list\_update\_mode == VPL\_INITLIST) |  |
| miv\_view\_params\_list( ) |  |
| else if( aaps\_miv\_view\_params\_list\_update\_mode == VPL\_UPD\_EXT) |  |
| miv\_view\_params\_update\_extrinsics( ) |  |
| else if( aaps\_miv\_view\_params\_list\_update\_mode == VPL\_UPD\_INT) |  |
| miv\_view\_params\_update\_intrinsics( ) |  |
| else if( aaps\_miv\_view\_params\_list\_update\_mode == VPL\_EXT\_INT) { |  |
| miv\_view\_params\_update\_extrinsics( ) |  |
| miv\_view\_params\_update\_intrinsics( ) |  |
| } |  |
| } |  |
| **aaps\_extension2\_flag** | u(1) |
| if( aaps\_extension2\_flag ) { |  |
| while ( more\_rbsp\_data() ) |  |
| **aaps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### MIV view parameters list syntax

|  |  |
| --- | --- |
| miv\_view\_params\_list( ) { | **Descriptor** |
| **mvp\_num\_views\_minus1** | u(16) |
| for( v = 0; v <= num\_views\_minus1; v++ ) |  |
| camera\_extrinsics( v ) |  |
| **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 , 0 ) |  |
| **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\_children( v ) |  |
| } |  |

#### 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\_views\_updates\_minus1; i++ ) { |  |
| **mvpue\_view\_idx**[ i ] | u(16) |
| camera\_extrinsics( mvpue\_view\_idx[ i ] ) |  |
| } |  |
| } |  |

#### 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 ], 1 ) |  |
| } |  |
| } |  |

#### Depth quantization syntax

|  |  |
| --- | --- |
| depth\_quantization( v ) { | **Descriptor** |
| **dq\_quantization\_law**[ v ] | u(8) |
| if( dq\_quantization\_law[ v ] == 0 ) { |  |
| **dq\_norm\_disp\_low**[ v ] | fl(32) |
| **dq\_norm\_disp\_high**[ v ] | fl(32) |
| } |  |
| **dq\_depth\_occ\_map\_threshold\_default**[ v ] | ue(v) |
| } |  |

#### Pruning children syntax

|  |  |
| --- | --- |
| pruning\_children( v ) { | **Descriptor** |
| **pc\_is\_leaf\_flag**[ v ] | u(1) |
| if ( pc\_is\_leaf\_flag[ v ] == 0 ) { |  |
| **pc\_num\_children\_minus1**[ v ] | u(v) |
| for ( i = 0; i <= pc\_num\_children\_minus1[ v ]; i++ ) |  |
| **pc\_child\_id**[ v ][ i ] | u(v) |
| } |  |
| } |  |

#### Camera intrinsics syntax

|  |  |
| --- | --- |
| camera\_intrinsics( v, mode ) { | **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\_center\_hor**[ v ] | fl(32) |
| **ci\_perspective\_center\_ver**[ v ] | fl(32) |
| } else if( ci\_cam\_type[v] == 2 ) { /\* orthographic \*/ |  |
| **ci\_ortho\_width**[ v ] | fl(32) |
| **ci\_ortho\_height**[ v ] | fl(32) |
| } |  |
| } |  |

#### 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 ] | fl(32) |
| **ce\_view\_quat\_y**[ v ] | fl(32) |
| **ce\_view\_quat\_z**[ v ] | fl(32) |
| } |  |

## Semantics

### General

The semantics in [3VC] clause 7.4.1 apply.

### V-PCC unit semantics

#### General V-PCC unit semantics

The semantics in [3VC] clause 7.4.2.1 apply.

#### V-PCC unit header semantics

The semantics in [3VC] clause 7.4.2.2 apply with the following modifications.

**vuh\_atlas\_id** specifies the index of the atlas that corresponds to the current V-PCC unit.

When vpcc\_unit\_type is equal to VPCC\_AD, the value of vuh\_atlas\_id shall be in the range of 0 to 63, inclusive.

Otherwise (vpcc\_unit\_type equal to VPCC\_GVD or VPCC\_AVD), the value of vuh\_atlas\_id shall be in the range of 0 to 62, inclusive.

#### V-PCC unit payload semantics

The semantics in [3VC] clause 7.4.2.3 apply.

#### Order of V-PCC units and association to coded information

The bitstream contains a series of V-PCC sequences. Each V-PCC sequence shall contain the following, in order:

* a VPCC\_VPS vpcc unit
* an IRAP V-PCC access unit
* zero or more non-IRAP V-PCC access units.

Each V-PCC access unit shall contain the following, in any order:

* zero or one VPCC\_AD vpcc unit with the value of vuh\_atlas\_id equal to 0x3F
* zero or more VPCC\_AD vpcc units, each with a unique value of vuh\_atlas\_id, in the range of 0 to 0x3E, inclusive
* for each V\_PCC GVD vpcc unit present in the bitstream, with vai set equal to the value of vuh\_atlas\_id of the GVD vpcc unit:
  + zero or one VPCC\_AD vpcc units with vuh\_atlas\_id equal to vai
  + zero or one VPCC\_AVD vpcc units with vuh\_atlas\_id equal to vai and ai\_attribute\_type\_id[ vai ][ 0 ] equal to ATTR\_TEXTURE

Each IRAP V-PCC access unit shall contain in a VPCC\_AD vpcc unit either one of the following, but not both:

1. an atlas adaptation parameter set RBSP with vuh\_atlas\_id = 0x3F
2. for each V\_PCC GVD vpcc unit present in the bitstream with vai set equal to the value of vuh\_atlas\_id of the GVD vpcc unit:
   * an atlas adaptation parameter set RBSP with vuh\_atlas\_id equal to vai

### Byte alignment semantics

The semantics in [3VC] clause 7.4.3 apply.

### V-PCC parameter set semantics

#### General V-PCC parameter set semantics

The semantics in [3VC] clause 7.4.4.1 apply.

#### Profile, tier, and level semantics

The semantics in [3VC] clause 7.4.4.2 apply.

#### Occupancy information semantics

This clause does not apply to this version of the specification.

#### Geometry information semantics

The semantics in [3VC] clause 7.4.4.4 apply, except for the following syntax element.

**gi\_geometry\_nominal\_2d\_bitdepth\_minus1**[ j ] plus 1 indicates the bits required for the pdu\_depth\_occ\_threshold[ patchIdx ] syntax element. gi\_geometry\_nominal\_2d\_bitdepth\_minus1 [ j ] shall be in the range of 0 to 31, inclusive.

#### Attribute information semantics

The semantics in [3VC] clause 7.4.4.5 apply.

#### MIV sequence parameters semantics

**msp\_depth\_low\_quality\_flag** equal to 1 indicates that the depth fidelity confidence in geometry video sub-streams is low. msp\_depth\_low\_quality\_flag equal to 0 indicates that the depth fidelity confidence is not low. When not present, the value of msp\_depth\_low\_quality\_flag is inferred to be equal to 0.

**msp\_geometry\_scale\_enabled\_flag** equal to 1 specifies that the geometry video sub-streams may have a different coded picture width and height than the atlas frame width and height, respectively, specified in the atlas sequence parameter set RBSP of the associated atlas. When msp\_geometry\_scale\_enabled\_flag is equal to 0, it is a requirement of bitstream conformance that the picture width and picture height of the geometry video stream be equal to the atlas frame width and height, respectively, specified in the atlas sequence parameter set RBSP of the associated atlas. When not present, the value of msp\_geometry\_scale\_enabled\_flag is inferred to be equal to 0.

**msp\_num\_groups\_minus1** specifies the maximum value of masp\_group\_id in the miv\_atlas\_sequence\_params( ) syntax structure. When not present, the value of msp\_num\_groups\_minus1 is inferred to be equal to 0.

**msp\_max\_entities\_minus1** specifies the maximum value of pdu\_entity\_id[ patchIdx ] in the patch\_data\_unit( ) syntax structure. When not present, the value of msp\_max\_entities\_minus1 is inferred to be equal to 0.

#### MIV VUI parameters semantics

[Ed. (JB): No syntax elements now.]

#### Coordinate axis system parameters semantics

The coordinate axis system params assign the orthogonal directions forward, left and up [OMAF] to the axis indices 0...2 corresponding to abstract dimensions x, y and z.

**cas\_forward\_axis** specifies the axis index of the forward direction. The variable CasForwardAxisIndex is derived as follows:

CasForwardAxisIndex = cas\_forward\_axis

**cas\_delta\_left\_axis\_minus1** + 1 specifies the axis index of the left direction as a delta (mod 3) on the axis index of the forward direction. The variable CasLeftAxisIndex is derived as follows:

CasLeftAxisIndex = (CasForwardAxisIndex + cas\_delta\_left\_axis\_minus1 + 1) % 3

The variable CasUpAxisIndex is derived from the CasForwardAxisIndex and CasLeftAxisIndex variables:

if( CasForwardAxisIndex != (CasLeftAxisIndex + 1) % 3)

CasUpAxisIndex = (CasLeftAxisIndex + 1) % 3

else

CasUpAxisIndex = (CasLeftAxisIndex + 2) % 3

**cas\_forward\_sign** specifies the forward direction in relation to the forward axis. cas\_forward\_sign equal to 1 specifies that the forward direction is equal to the forward axis direction. cas\_forward\_sign equal to 0 specifies that the forward direction is opposite to the forward axis direction.

The variable array CasForwardVector[ i ] is the forward direction as a unit vector with i the axis number and is derived as follows:

for( i = 0; i < 3; i++ )

CasForwardVector[ i ] = i == CasForwardAxisIndex ? (2 \* cas\_forward\_sign - 1) : 0

**cas\_left\_sign** specifies the left direction in relation to the left axis. cas\_left\_sign equal to 1 specifies that the left direction is equal to the left axis direction. cas\_left\_sign equal to 0 specifies that the left direction is opposite to the left axis direction.

The variable array CasLeftVector[ i ] is the left direction as a unit vector with i the axis number and is derived as follows:

for( i = 0; i < 3; i++ )

CasLeftVector[ i ] = i == CasLeftAxisIndex ? (2 \* cas\_left\_sign - 1) : 0

**cas\_up\_sign** specifies the up direction in relation to the up axis. cas\_up\_sign equal to 1 specifies that the up direction is equal to the up axis direction. cas\_up\_sign equal to 0 specifies that the up direction is opposite to the up axis direction.

The variable array CasUpVector[ i ] is the up direction as a unit vector with i the axis number and is derived as follows:

for( i = 0; i < 3; i++)

CasUpVector[ i ] = i == CasUpAxisIndex ? (2 \* cas\_up\_sign - 1) : 0

### NAL unit semantics

The specifications in [3VC] clause 7.4.5 and its sub-clauses apply.

### Atlas sequence, frame, and tile group parameter set semantics

#### Atlas sequence parameter set RBSP semantics

The specifications in [3VC] clause 7.4.6.1 apply, except as specified below.

An atlas sequence parameter set RBSP shall be available to the decoding process prior to it being referenced, included in the current V-PCC unit, or in a prior V-PCC unit with the same value of vuh\_atlas\_id, or in a prior V-PCC unit with vuh\_atlas\_id value equal to 0x3F, or provided through external means. [Ed. (JB): This paragraph can be removed once this constraint is added to the V-PCC spec clause 7.4.5.3.2 Order of ASPS and AFPS RBSPs and their activation]

**asps\_frame\_width** indicates the atlas frame width in terms of integer luma samples for the current atlas. It is a requirement of 3VC bitstream conformance that the value of asps\_frame\_width shall be less than or equal to the value of vps\_frame\_width[ vuh\_atlas\_id].

**asps\_frame\_height** indicates the atlas frame height in terms of integer luma samples for the current atlas. It is a requirement of 3VC bitstream conformance that the value of asps\_frame\_height shall be less than or equal to the value of vps\_frame\_height[ vuh\_atlas\_id ]

**asps\_log2\_patch\_packing\_block\_size** specifies the value of the variable PatchPackingBlockSize, that is used for the horizontal and vertical placement of the patches within the atlas, as follows:

PatchPackingBlockSize = 2( asps\_log2\_patch\_packing\_block\_size) (7‑1)

**asps\_max\_projections\_minus1** indicates the maximum value of pdu\_view\_id[ patchIdx ]. When not present, the value of asps\_max\_projections\_minus1 is inferred to be equal to 5.

The variables AspsFrameWidth[ ], AspsFrameHeight[ ], and AspsMaxProjections[ ] are derived as follows.

if (vuh\_atlas\_id == 0x3F) {

for (a = 0; a <= vps\_atlas\_count\_minus1; a++) {

AspsFrameWidth[ a ] = asps\_frame\_width

AspsFrameHeight[ a ] = asps\_frame\_height

AspsMaxProjections[ a ] = asps\_max\_projections\_minus1 + 1

AtlasPatchPackingBlockSize[ a ] = PatchPackingBlockSize

}

}

else {

AspsFrameWidth[ vuh\_atlas\_id ] = asps\_frame\_width

AspsFrameHeight[ vuh\_atlas\_id ] = asps\_frame\_height

AspsMaxProjections[ vuh\_atlas\_id ] = asps\_max\_projections\_minus1 + 1

AtlasPatchPackingBlockSize[ vuh\_atlas\_id ] = PatchPackingBlockSize

}

It is a requirement of bitstream conformance that for all values of vai in 0 .. atlas\_count\_minus1, the following applies

* the coded picture width and picture height of the texture attribute video sub-stream with vuh\_atlas\_id equal to vai, if present, be equal to AspsFrameWidth[vai] and AspsFrameHeight[vai], respectively.
* if geometry\_scale\_enabled\_flag equal to 0
  + the coded picture width and picture height of the geometry video sub-stream, with vuh\_atlas\_id equal to vai, be equal to AspsFrameWidth[ vai ] and AspsFrameHeight[ vai ], respectively.

#### Point local reconstruction information semantics

This clause does not apply to this version of the specification.

#### Atlas frame parameter set RBSP semantics

The specifications in [3VC] clause 7.4.6.3 apply.

#### Atlas frame tile information semantics

The specifications in [3VC] clause 7.4.6.4 apply.

#### Atlas camera parameter set RBSP semantics

This clause does not apply to this version of the specification.

#### Supplemental enhancement information RBSP semantics

The specifications in [3VC] clause 7.4.6.6 apply.

#### Access unit delimiter RBSP semantics

The specifications in [3VC] clause 7.4.6.7 apply.

#### End of sequence RBSP semantics

The specifications in [3VC] clause 7.4.6.8 apply.

#### End of bitstream RBSP semantics

The specifications in [3VC] clause 7.4.6.9 apply.

#### Filler data RBSP semantics

The specifications in [3VC] clause 7.4.6.10 apply.

#### Atlas tile group layer RBSP semantics

If vuh\_atlas\_id equal to 0x3F, the atlas tile group layer RBSP shall not be present.

#### Atlas tile group header semantics

The specifications in [3VC] clause 7.4.6.12 apply.

#### Reference list structure semantics

The specifications in [3VC] clause 7.4.6.13 apply.

#### MIV atlas sequence parameters semantics

**masp\_omaf\_v1\_compatible\_flag** specifies that the atlas texture component is compatible for carriage within [OMAF]. When masp\_omaf\_v1\_compatible\_flag is equal to 1, it is a requirement of bitstream conformance that at least one sub-set of patches in the atlas texture component conforms to a projection format specified in [OMAF]. When not present the value of masp\_omaf\_v1\_compatible\_flag is inferred to be 0. [Ed. (JB): Needs updating for V-PCC terminology.]

**masp\_group\_id**   specifies the group index of the atlas. The number of bits used for the representation of masp\_group\_id[ ia ] is Ceil( Log2( msp\_num\_groups\_minus1+1) ). The value of masp\_group\_id shall be in the range of 0 to msp\_num\_groups\_minus1. When not present, the value of masp\_group\_id is inferred to be equal to 0.

**masp\_auxiliary\_atlas\_flag**   equal to 1 indicates that the patches of the atlas are not to intended be used for view rendering. masp\_auxiliary\_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 masp\_auxiliary\_atlas\_flag is inferred to be equal to 0.

**masp\_depth\_occ\_threshold\_flag** equal to 1 specifies that the pdu\_depth\_occ\_threshold syntax element is present in the patch\_data\_unit( ) syntax structure. masp\_depth\_occ\_threshold\_flag equal to 0 specifies that the pdu\_depth\_occ\_threshold syntax elements is not present in the patch\_data\_unit( ) syntax structure. When not present, the value of masp\_depth\_occ\_threshold\_flag is inferred to be equal to 0.

**masp\_geometry\_frame\_scale\_factor\_x\_minus1** + 1 specifies the frame width of the geometry video data of the atlas in relation to the nominal atlas width. When not present, the value of masp\_geometry\_frame\_scale\_factor\_x\_minus1 is inferred to be equal to 0.

The variable MaspGeometryFrameScaleFactorX[ vuh\_atlas\_id ] is set equal to masp\_geometry\_frame\_scale\_factor\_x\_minus1 + 1. The variable MaspGeometryFrameWidth[ vuh\_atlas\_id ] is set equal to AspsFrameWidth[ vuh\_atlas\_id ] / MaspGeometryFrameScaleFactorX[ vuh\_atlas\_id ]. It is a requirement of bitstream conformance that AspsFrameWidth[ vuh\_atlas\_id ] is divisible by MaspGeometryFrameScaleFactorX[ vuh\_atlas\_id ].

**masp\_geometry\_frame\_scale\_factor\_y\_minus1** + 1 specifies the frame height of the geometry video data of the atlas in relation to the nominal atlas height. When not present, the value of masp\_geometry\_frame\_scale\_factor\_y\_minus1 is inferred to be equal to 0.

The variable MaspGeometryFrameScaleFactorY[ vuh\_atlas\_id ] is set equal to masp\_geometry\_frame\_scale\_factor\_y\_minus1 + 1. The variable MaspGeometryFrameHeight[ vuh\_atlas\_id ] is set equal to AspsFrameHeight[ vuh\_atlas\_id ] / MaspGeometryFrameScaleFactorY[ vuh\_atlas\_id ]. It is a requirement of bitstream conformance that AspsFrameHeight[ vuh\_atlas\_id ] is divisible by MaspGeometryFrameScaleFactorY[ vuh\_atlas\_id ].

#### Frame order count RBSP semantics

**frm\_order\_cnt\_lsb** specifies the frame order count modulo MaxAtlasFrmOrderCntLsb for the succeeding NAL units. The length of the frm\_order\_cnt\_lsb syntax element is equal to aaps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4 + 4 bits when vuh\_atlas\_id is equal 0x3F, and equal to asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4 + 4 bits when vuh\_atlas\_id is in the range 0..0x3E. The value of the frm\_order\_cnt\_lsb shall be in the range of 0 to MaxAtlasFrmOrderCntLsb − 1, inclusive. When frm\_order\_cnt\_lsb is not present, it shall be inferred to be equal to 0.

### Atlas tile group data unit semantics

#### General atlas tile group data unit semantics

The semantics in [3VC] clause 7.4.7.1 apply.

#### Patch information data semantics

The semantics in [3VC] clause 7.4.7.2 apply.

#### Atlas patch data unit semantics

The semantics in [3VC] clause 7.4.7.3 apply except for the following syntax elements.

**pdu\_view\_id**[ patchIdx ] specifies the view index associated with the patch with index equal to patchIdx. The number of bits used to represent pdu\_view\_id[ patchIdx ] is equal to Ceil( Log2( AspsMaxProjections[ vuh\_atlas\_id ] ) ). The value of pdu\_view\_id shall be in the range of 0 to mvp\_num\_views\_minus1, inclusive.

**pdu\_view\_pos\_x**[ patchIdx ] specifies the horizontal coordinate in luma samples, respectively, of the top-left corner of the patch with index equal to patchIdx in the view with index equal to pdu\_view\_id[ patchIdx ]. The value of pdu\_view\_pos\_x[ patchIdx ] shall be in the range of 0 to ci\_projection\_plane\_width\_minus1[ pdu\_view\_id[ patchIdx ] ], inclusive. The number of bits used to represent pdu\_3d\_pos\_x[ patchIdx ] is afps\_3d\_pos\_x\_bit\_count\_minus1 + 1.

NOTE: The number of bits is identical to V-PCC.

**pdu\_view\_pos\_y**[ patchIdx ] specifies the vertical coordinate in luma samples, respectively, of the top-left corner of the patch with index equal to patchIdx in the view with index equal to pdu\_view\_id[ patchIdx ]. The value of pdu\_view\_pos\_y[ patchIdx ] shall be in the range of 0 to ci\_projection\_plane\_height\_minus1[ pdu\_view\_id[ patchIdx ] ], inclusive. The number of bits used to represent pdu\_view\_pos\_y[ patchIdx ] is afps\_3d\_pos\_y\_bit\_count\_minus1 + 1.

NOTE: The number of bits is identical to V-PCC.

The variable depthBits is set equal to gi\_geometry\_3d\_coordinates\_bitdepth\_minus1 + 1– atgh\_pos\_min\_z\_quantizer + 2.

**pdu\_depth\_start**[ patchIdx ] is used to derive the start PduDepthStart[ patchIdx ] of the range of depth values for the patch with index equal to patchIdx. When not present, the value of pdu\_depth\_start[ patchIdx ] is inferred to be equal to 0. The variable PduDepthStart[ vuh\_atlas\_id ][ patchIdx ] is derived as follows:

PduDepthStart[ vuh\_atlas\_id ][ patchIdx ]=pdu\_depth\_start[ patchIdx ]<< atgh\_pos\_min\_z\_quantizer

The value of PduDepthStart[ p ] shall be in the range of 0 to 2gi\_geometry\_3d\_coordinates\_bitdepth\_minus1 + 1 – 1, inclusive.

**pdu\_depth\_end**[ patchIdx ] is used to derive the end PduDepthEnd[ patchIdx ] of the range of depth values for the patch with index equal to patchIdx. When not present, the value of pdu\_depth\_end[ patchIdx ] is inferred to be equal to an arbirary large value 0xFFFFFFFF. The variable PduDepthEnd[ patchIdx ] [ vuh\_atlas\_id ] is derived as follows:

PduDepthEnd[ vuh\_atlas\_id ][ patchIdx ] = pdu\_depth\_end[ patchIdx ] << atgh\_pos\_max\_z\_quantizer

The value of PduDepthEnd[ p ] shall be in the range of 0 to 2gi\_geometry\_3d\_coordinates\_bitdepth\_minus1 + 1 – 1, inclusive.

**pdu\_entity\_id**[ patchIdx ] specifies the entityID of the patch with index equal to patchIdx, within the view with index equal to pdu\_view\_id[ patchIdx ]. The number of bits used for the representation of pdu\_entity\_id[ patchIdx ] is Ceil( Log2( msp\_max\_entities\_minus1+1 ) ). The value of pdu\_entity\_id[ patchIdx ] shall be in range of 0 to msp\_max\_entities\_minus1, inclusive. When not present, the value of pdu\_entity\_id[ patchIdx ] is inferred to be equal to 0.

**pdu\_depth\_occ\_threshold**[ patchIdx ] specifies the threshold below which the occupancy value is defined to be unoccupied for the patch with index equal to patchIdx. Geometry and attribute values of unoccupied pixels are ignored by a MIV renderer. The number of bits used to represent pdu\_depth\_occ\_threshold[ patchIdx ] is equal to gi\_geometry\_nominal\_2d\_bitdepth\_minus1 + 1. When not present, the value of pdu\_depth\_occ\_threshold[ patchIdx ] is inferred to be equal to dq\_depth\_occ\_map\_threshold\_default[ pdu\_view\_id[ patchIdx ] ].

### Supplemental enhancement information message semantics

The semantics in [3VC] clause 7.4.8 and its subclauses apply.

### Atlas Adaptation parameters set RBSP semantics

[Ed. (ALL): MIV extension of a syntax structure expected to be in 3VC spec.]

The semantics of 3VC clause 7.4.9 apply with the following modifications and additions.

It is a requirement of bitstream conformance that if the value of vuh\_atlas\_id is in 0 .. 0x3E, there is no atlas adaptation parameter set RBSP present in the current access unit with vuh\_atlas\_id equal to 0x3F.

**aaps\_adaptation\_parameter\_set\_id** indicates the identifier of the APS RBSP.

**aaps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4** specifies the value of the variable MaxAtlasFrmOrderCntLsb that is used in the decoding process for frame order count as follows:

MaxAtlasFrmOrderCntLsb = 2( aaps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4 + 4 )

The value of aaps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4 shall be in the range of 0 to 12, inclusive. It is required for bitstream conformance that the value of MaxAtlasFrmOrderCntLsb be the same for all atlases in the bitstream

**aaps**\_**camera\_params\_present\_flag** equal to 1 specifies that the camera\_params( ) syntax structure is present. aaps\_camera\_params\_present\_flagequal to 0 specifies that the camera\_params( ) syntax structure is not present.

**aaps\_extension\_bit\_equal\_to\_one** shall be equal to 1.

**asps\_miv\_view\_params\_list\_present\_flag** equal to 1 specifies that view parameters are present in the syntax structure. ap\_miv\_view\_params\_list\_present\_flagequal to 0 specifies that view parameters are not present in the syntax structure. In an IRAP access unit the value of aaps\_miv\_view\_params\_list\_present\_flagshall be equal to 1.

**aaps\_miv\_view\_params\_list\_update\_mode** specifies the mode in which the view parameters list is updated, as specified in Table 1. In an IRAP access unit the value of aaps\_miv\_view\_params\_list\_present\_mode shall be equal to VPL\_INITLIST. In a non-IRAP access unit the value of aaps\_miv\_view\_params\_list\_present\_mode shall be one of VPL\_UPD\_EXT, VPL\_UPD\_INT, or VPL\_EXT\_INT.

Table 1: Updating modes for view parameters list update

|  |  |  |
| --- | --- | --- |
| **ap\_miv\_view\_params\_list\_update \_mode** | **Identifier** | **Description** |
| 0 | VPL\_INITLIST | a new initialized view parameters list is present |
| 1 | VPL\_UPD\_EXT | extrinsic parameters are updated for sub-set of existing views |
| 2 | VPL\_UPD\_INT | intrinsic parameters are updated for sub-set of existing views |
| 3 | VPL\_EXT\_INT | both extrinsic and intrinsic parameters are updated for sub-set of existing views |

#### MIV view parameters list semantics

**mvp\_num\_views\_minus1** plus 1 indicates the maximum number of views in an MIV view parameters list representing a 3D scene.

**mvp\_intrinsic\_params\_equal\_flag** equal to 1 specifies that the intrinsic parameters camera\_intrinsics( 0, 0 ) of the 0-th camera apply to all cameras in the camera parameters list. Intrinsic\_params\_equal\_flag equal to 0 specifies that the intrinsic parameters camera\_intrinsics( v, 0 ) are present for each camera in the camera parameters list.

**mvp\_depth\_quantization\_params\_equal\_flag** equal to 1 specifies that the depth quantization parameters depth\_quantization( 0 ) of the 0-th camera applies to all cameras in the camera parameters list. Depth\_quantization\_params\_equal\_flag equal to 0 specifies that the depth quantization parameters depth\_quantization( v ) are present for each camera in the camera parameters list.

**mvp\_pruning\_graph\_params\_present\_flag** equal to 1 specifies that pruning graph parameters are present. mvp\_pruning\_graph\_params\_present\_flag equal to 0 specifies that pruning graph parameters are not present.

#### MIV view parameters update extrinsics semantics

**mvpue\_num\_view\_updates\_minus1** plus 1 indicates the number of camera extrinsic parameters update entries present in the current miv\_view\_params\_update\_extrinsics syntax structure. The value of mvpue\_num\_view\_updates\_minus1shall be in the range of 0 to mvp\_num\_views\_minus1**,** inclusive**.**

**mvpue\_view\_idx**[ i ] specifies the index in the camera list of the i-th signaledd extrinsic parameters. The value of mvpue\_view\_idx[ i ] shall be in the range 0 to mvp\_num\_views\_minus1, inclusive.

#### MIV view parameters update intrinsics semantics

**mvpui\_num\_view\_updates\_minus1** plus 1 indicates the number of intrinsic parameters entries in the camera list present in the syntax structure. The value of mvpue\_num\_view\_updates\_minus1shall be in the range of 0 to mvp\_num\_views\_minus1**.**

**mvpui\_view\_idx**[ i ] specifies the index in the camera list of the i-th signalled intrinsic parameters. The value of mvpue\_view\_idx[ i ] shall be in the range 0 to mvp\_num\_views\_minus1 inclusive.

#### Depth quantization semantics

**dq\_quantization\_law**[ v ] indicates the type of depth quantization method of the v-th camera. quantization\_law[ v ] equal to 0 specifies a uniform quantization of the inverse of depth values. Values of quantization\_law[ v ] greater than 0 are reserved for future use by ISO/IEC.

**dq\_norm\_disp\_low**[ v ] and **dq\_norm\_disp\_high**[ v ] specify the minimum and maximum normalized disparity values, respectively, in meters-1 of the 3D scene captured by the v-th camera.

**dq\_depth\_occ\_map\_threshold\_default**[ v ] specifies the default occupancy threshold used in the occupancy value extraction process G.3.

#### Pruning children semantics

**pc\_is\_leaf\_flag**[ v ] equal to 1 indicates that v-th source view has no child in the pruning graph at encoder stage. pc\_is\_leaf\_flag[ v ] equal to 0 indicates that v-th source view has at least one child in the pruning graph at encoder stage.

**pc\_num\_children\_minus1**[ v ] plus 1 indicates the number of children of the v-th source view in the pruning graph at encoder stage. pc\_num\_children\_minus1[ v ] shall be in the range 0 to mvp\_num\_views\_minus1 exclusive.

**pc\_child\_id**[ v ][ i ] specifies the index of the i-th child view for the v-th source view in the pruning graph at encoder stage. pc\_child\_id[ v ][ i ] shall be in the range of 0 to mvp\_num\_views\_minus1 inclusive, but shall not be equal to v.

#### Camera intrinsics semantics

**ci\_cam\_type**[ v ] indicates the projection method of the v-th camera. 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 255 are reserved for future use by ISO/IEC. When not present and mode equal to 0, the value of ci\_cam\_type[ v ] is inferred to equal to ci\_cam\_type[ 0 ].

**ci\_projection\_plane\_width\_minus1**[ v ] + 1 and **ci\_projection\_plane\_height\_minus1**[ v ] + 1 specify the horizontal and vertical resolutions of the camera projection plane, respectively, expressed in coded luma samples. When not present and mode equal to 0, the values of ci\_projection\_plane\_width\_minus1[ v ] and ci\_projection\_plane\_height\_minus1[ v ] are inferred to be equal to ci\_projection\_plane\_width\_minus1[ 0 ] and ci\_projection\_plane\_height\_minus1[ 0 ], respectively.

**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 radians. ci\_erp\_phi\_min[ v ] and ci\_erp\_phi\_max[ v ] shall be in the range −π to π in the spherical coordinate system. When not present and mode equal to 0, the values of ci\_erp\_phi\_min[ v ] and ci\_erp\_phi\_max[ v ] are inferred to be equal to ci\_erp\_phi\_min[ 0 ] and ci\_erp\_phi\_max[ 0 ], respectively. It is a requirement of bitstream conformance that ci\_erp\_phi\_min[ v ]  < ci\_erp\_phi\_max[ v ].

**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 −π/2 to π /2 in the spherical coordinate system. When not present and mode equal to 0, the values of ci\_erp\_theta\_min[ v ] and ci\_erp\_theta\_max[ v ] are inferred to be equal to ci\_erp\_theta\_min[ 0 ] and ci\_erp\_theta\_max[ 0 ], respectively. It is a requirement of bitstream conformance that ci\_erp\_theta\_min[ v ] < ci\_erp\_theta\_max[ v ].

**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. When not present and mode equal to 0, the values of ci\_perspective\_focal\_hor[ v ] and ci\_perspective\_focal\_ver[ v ] are inferred to be equal to ci\_perspective\_focal\_hor[ 0 ] and ci\_perspective\_focal\_ver[ 0 ], respectively.

**ci\_perspective\_center\_hor**[ v ] and **ci\_perspective\_center\_ver**[ v ] are floating-point values that specify in luma sample positions the horizontal and vertical coordinates, respectively, of the principal point of a perspective projection (intersection of optical axis with image plane). When not present and mode equal to 0, the values of ci\_perspective\_center\_hor[ v ] and ci\_perspective\_center\_ver[ v ] are inferred to be equal to ci\_perspective\_center\_hor[ 0 ] and ci\_perspective\_center\_ver[ 0 ], respectively.

**ci\_ortho\_width**[ v ] and **ci\_ortho\_height**[ v ] are positive floating-point values that specify in meters the horizontal and vertical dimensions of the captured part of the 3D scene. When not present and mode equal to 0, the values of ci\_ortho\_width[ v ] and ci\_ortho\_height[ v ]are inferred to be equal to ci\_ortho\_width[ 0 ] and ci\_ortho\_height[ 0 ], respectively.

#### Camera extrinsics semantics

**ce\_view\_pos\_x**[ v ] specifies in meters the X coordinate, Tx, of the location of the v-th camera as floating point in the global reference coordinate system.

**ce\_view\_pos\_y**[ v ] specifies in meters the Y coordinate, Ty, of the location of the v-th camera as floating point in the global reference coordinate system.

**ce\_view\_pos\_z**[ v ] specifies in meters the Z coordinate, Tz, of the location of the v-th camera as floating point in the global reference coordinate system.

**ce\_view\_quat\_x**[ v ] specifies the x component, qX, for the rotation that is applied to convert the global coordinate axes to the local coordinate axes of the v-th camera using the quaternion representation. The value of ce\_view\_quat\_x[ v ] shall be a floating-point value in the range of -1 to 1, inclusive.

**ce\_view\_quat\_y**[ v ] specifies the y component, qY, for the rotation that is applied to convert the global coordinate axes to the local coordinate axes of the v-th camera using the quaternion representation. The value of ce\_view\_quat\_y[ v ] shall be a floating-point value in the range of -1 to 1, inclusive.

**ce\_view\_quat\_z**[ v ] specifies the z component, qZ, for the rotation that is applied to convert the global coordinate axes to the local coordinate axes of the v-th camera using the quaternion representation. The value of ce\_view\_quat\_z[ v ] shall be a floating-point value in the range of -1 to 1, inclusive.

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

qW = Sqrt( 1 – ( qX2 + qY2 + qZ2 ) )

# Decoding process

## General decoding process

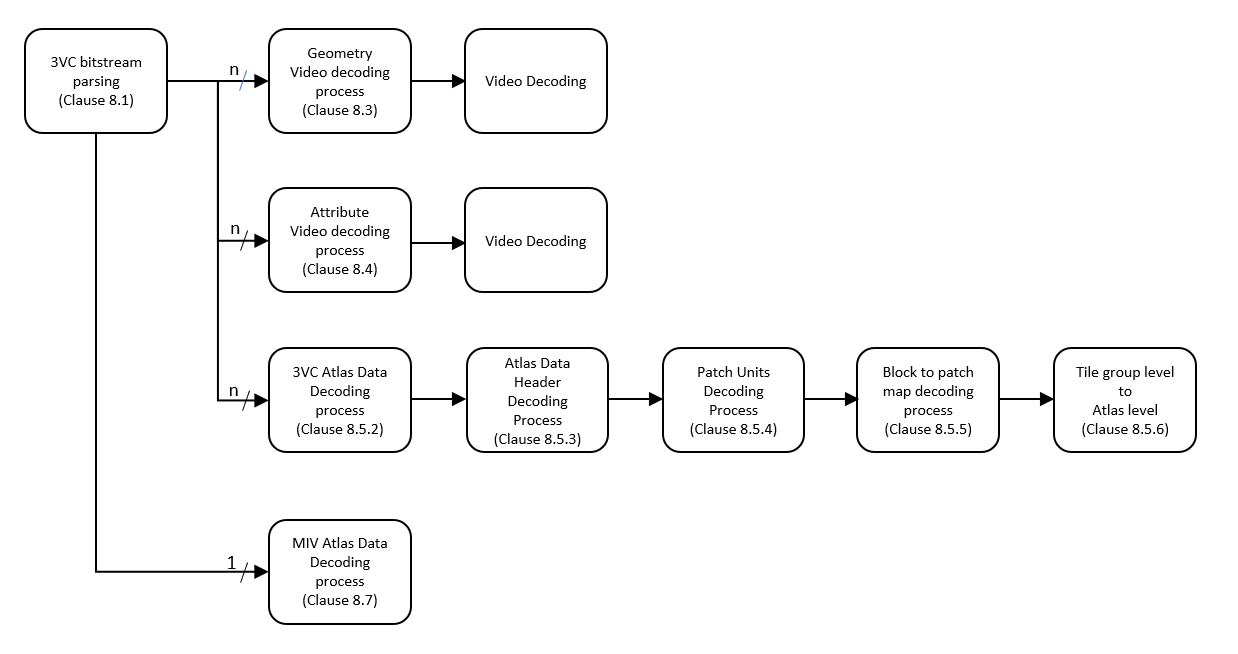


Figure 3: High-level mapping of the MIV decoding processes and their interactions.

Input to this process are the following:

* a 3VC bitstream,
* an array of flags indicating if each group is targeted for decode, TargetGroupFlag[ ], of length msp\_num\_groups\_minus1 + 1, determined via external means,
* an array of flags indicating if each entity is targeted for decode, TargetEntityFlag[ ], of length msp\_max\_entities\_minus1 + 1, determined via external means.

If msp\_num\_groups\_minus1 equal to 0, TargetGroupFlag[ 0 ]is set equal to 1.

If msp\_max\_entities\_minus1 equal to 0, TargetEntityFlag[ 0 ] is set equal to 1.

Output of this process is, for each access unit in a series of access units, a view parameter list containing intrinsic and extrinsic parameters, and for each atlas present in the access unit, each with a unique value of vuh\_atlas\_id, the following:

* one decoded geometry picture of size MaspGeometryFrameWidth[ vuh\_atlas\_id ] x  MaspGeometryFrameHeight[ vuh\_atlas\_id]
* zero or one texture attribute pictures of size AspsFrameWidth[ vuh\_atlas\_id ]  x AspsFrameHeight[ vuh\_atlas\_id ]
* one BlockToPatchMap of size (AspsFrameWidth[ vuh\_atlas\_id ] /  AtlasPatchPackingBlockSize[ vuh\_atlas\_id ]) x ( AspsFrameHeight[ vuh\_atlas\_id ]  / AtlasPatchPackingBlockSize[ vuh\_atlas\_id ] )

[Ed.(BK): I suggest to have a simple block diagram that shows the dependencies between the decoding processes. Both the geometry video decoding process and the attribute video decoding process invoke a video decoding process. With depth scaling the geometry video decoding process takes input from the block to patch map decoding process and the attribute video decoding process. If you agree, I can provide such a diagram.][Ed. (JB): Sounds like a good idea. Perhaps also good to create one for the rendering process]

The decoding process operates as follows:

For each 3VC sequence in the bitstream, the following is repeatedly invoked:

1. The geometry sub-bitstream is extracted from the 3VC sequence.
2. The geometry video decoding process, as specified in clause 8.3, is invoked for the geometry sub-bitstream.
3. If present, the texture attribute sub-bitstream is extracted from the 3VC sequence.
4. The attribute video decoding process, as specified in clause 8.4, is invoked for the texture attribute sub-bitstream, if present.
5. For each access unit in the 3VC sequence, the following is repeatedly invoked
   1. For each atlas present in the sequence, each with a unique value of vuh\_atlas\_id, the following is repeatedly invoked if TargetGroupFlag[ masp\_group\_id[ vuh\_atlas\_id ] ]> 0
      1. The sub-bitstream extraction process in clause 11 is invoked with targetAtlasId set equal to vuh\_atlas\_id, with the coded access unit of 3VC sequence as input and AtlasAccessUnitToDecode as ouput.
      2. The decoded geometry picture with the same value of vuh\_atlas\_id for the access unit is output.
      3. The decoded texture attribute picture with the same value of vuh\_atlas\_id for the access unit, if present, is output.
      4. If an atlas tile group layer RBSP is present in the current coded atlas frame, the atlas data decoding process, as specified in clause 8.5, is invoked for the current coded atlas frame, with output BlockToPatchMap[ ][ ].

Otherwise the atlas decoding process is invoked for the coded atlas data frame in the previous atlas tile group layer RBSP present in a coded atlas data frame of the sequence in a V-PCC unit with vuh\_atlas\_id equal to targetAtlasId , with output BlockToPatchMap[ ][ ].

## Occupancy video decoding process

This clause does not apply to this version of the specification.

## Geometry video decoding process

The specifications in [3VC] clause 8.3 apply.

## Attribute(s) video decoding process

The specifications in [3VC] clause 8.4 apply.

The variable AttrWidth[ 0 ][ 0 ][ orderIdx ] is set equal to AspsFrameWidth[ vuh\_atlas\_id ].

The variable AttrHeight[ 0 ][ 0 ][ orderIdx ] is set equal to AspsFrameHeight[ vuh\_atlas\_id ] .

The variable AiAttributeDimension[ vuh\_atlas\_id ][ 0 ] is set equal to ai\_attribute\_dimension\_minus1 [ vuh\_atlas\_id ][ 0 ] + 1.

## Atlas data decoding process

The specifications in [3VC] clause 8.5 apply with the following modifications to clause 8.5.5.



### Decoding process of the block to patch map

Inputs to this process are:

* the current atlas tile group address, tgAddress
* the total number of patches in the current atlas tile group, PfduTotalNumberOfPatches
* the Patch2dPosX, Patch2dPosY, Patch2dSizeX, Patch2dSizeY arrays
* the PatchPackingBlockSize, asps\_frame\_height, and asps\_frame\_width elements of the current active ASPS
* an array of flags indicating if each entity is targeted for decode, TargetEntityFlag[ ], of length msp\_max\_entities\_minus1 + 1

Outputs of this process are a two-dimensional array BlockToPatchMap and its width, BlockToPatchMapWidth, and height, BlockToPatchMapHeight, where:

BlockToPatchMapWidth = Ceil( TileGroupWidth[tgAddress ] ÷ PatchPackingBlockSize )

BlockToPatchMapHeight = Ceil( TileGroupHeight[tgAddress ] ÷ PatchPackingBlockSize )

All elements of BlockToPatchMap are first initialized to −1 as follows:

for( y = 0; y < BlockToPatchMapHeight; y++ )

for( x = 0; x < BlockToPatchMapWidth; x++ )

BlockToPatchMap[ y ][ x ] = −1

Then the BlockToPatchMap array is updated as follows:

for( patchIdx = 0; patchIdx < PfduTotalNumberOfPatches; patchIdx++ ) {

mode = atgdu\_patch\_mode[ patchIdx ]

if ((( atgh\_type == I\_TILE\_GRP ) && ( mode != I\_RAW ) && ( mode != I\_EOM )) ||  
 (( atgh\_type == P\_TILE\_GRP ) && (mode != P\_RAW) && ( mode != P\_EOM ) ||   
 ( atgh\_type == SKIP\_TILE\_GRP )) {

xOrg = Patch2dPosX[ patchIdx ] / PatchPackingBlockSize

yOrg = Patch2dPosY[ patchIdx ] / PatchPackingBlockSize

for( y = 0; y < Patch2dSizeX [ patchIdx ]/ PatchPackingBlockSize ; y++)

for( x = 0; x < Patch2dSizeY[ patchIdx ] / PatchPackingBlockSize ;x++) {

if(( asps\_patch\_precedence\_order\_flag == 0 )  ||  
 ( BlockToPatchMap[ yOrg + y ][ xOrg + x ] == −1)) {

if (msp\_max\_entities\_minus1== 0 || TargetEntityFlag[ pdu\_entity\_id[ patchIdx] ] )  
 BlockToPatchMap[ yOrg + y ][ xOrg + x ] = patchIdx

else

BlockToPatchMap[ yOrg + y ][ xOrg + x ] = -1

}

}

}

}

# Reconstruction process

This clause does not apply to this version of the specification.

# Parsing process

The specifications in [3VC] clause 10 and its subclauses apply.

# Sub-bitstream extraction process

Inputs to this process are a bitstream and a target atlas identifier atlasIdTarget.

Output of this process is a sub-bitstream.

It is a requirement of bitstream conformance for the input bitstream that any output sub-bitstream that is the output of the process specified in this clause shall be a conforming bitstream:

The output sub-bitstream is derived as follows:

– Remove all VPCC units with vuh\_atlas\_id not equal to atlasIdTarget or 0x3F.

1. Profiles, tiers, and levels
   1. Overview of profiles, tiers, and levels

The specifications in [3VC] Annex A.1 apply.

* 1. V-PCC profile, tier and level structure

The specifications in [3VC] Annex A.2 apply.

* 1. V-PCC CodecGroup profile components

The specifications in [3VC] Annex A.3 apply.

[Ed. (JB): May need to impose restriction on chroma\_format\_idc for video streams.]

* 1. V-PCC toolset profile components

**Table A-1** provides a list of the available toolset profile components for V-PCC.

**Table A-1 – Available toolset profile components**

|  |  |
| --- | --- |
| **Toolset** | **ptl\_profile\_pcc\_toolset\_idc** |
| Basic | 0 |
| Extended | 1 |
| MIV Main | 64 |

* + 1. MIV Main toolset profile component

V-PCC toolset profile components indicating the Main (ptl\_profile\_pcc\_toolset\_idc = 64) toolset profile component shall conform to the restrictions specified in Table A-2.

**Table A-2–Allowable values of syntax element values for MIV Main profile**

|  |  |
| --- | --- |
| **Syntax element** | MIV Main profile |
| vuh\_unit\_type | 1 (VPCC\_AD), 3 (VPCC\_GVD) |
| vps\_raw\_patch\_enabled\_flag | 0 |
| vps\_miv\_extension\_flag | 1 |
| ptl\_profile\_pcc\_toolset\_idc | 64 |
| ptl\_profile\_reconstruction\_idc | 64 |
| gi\_geometry\_MSB\_align\_flag | 0 |
| ai\_attribute\_MSB\_align\_flag | 0 |
| asps\_long\_term\_ref\_atlas\_frames\_flag | 0 |
| asps\_remove\_duplicate\_point\_enabled\_flag | 0 |
| asps\_pixel\_deinterleaving\_flag | 0 |
| asps\_patch\_precedence\_order\_flag | 0 |
| afps\_lod\_mode\_enabled\_flag | 0 |
| afps\_override\_eom\_for\_depth\_flag | 0 |
| afps\_raw\_3d\_pos\_bit\_count\_explicit\_mode\_flag | 0 |
| afti\_single\_tile\_in\_atlas\_frame\_flag | 1 |
| atgh\_type | I\_TILE\_GRP |
| atgdu\_patch\_mode | I\_INTRA |
| asps\_enhanced\_occupancy\_map\_for\_depth\_flag | 0 |
| vps\_map\_count\_minus1 | 0 |
| asps\_point\_local\_reconstruction\_enabled\_flag | 0 |
| ai\_attribute\_dimension\_minus1 | 2 |
| ai\_attribute\_dimension\_partitions\_minus1 | 0 |

* 1. Tiers and Levels

For purposes of comparison of tier capabilities, the tier with ptl\_tier\_flag equal to 0 is considered to be a lower tier than the tier with ptl\_tier\_flag equal to 1. Currently only a single tier, the Main tier, is specified for 3VC. This is indicated by setting the syntax element ptl\_tier\_flag equal to 0. It is a requirement for bitstream conformance to this current version of the Specification that ptl\_tier\_flag shall always be equal to 0.

For purposes of comparison of level capabilities, a particular level of a specific tier is considered to be a lower level than some other level of the same tier when the value of the ptl\_level\_idc of the particular level is less than that of the other level.

* + 1. MIV-specific level limits

Table A-3 specifies the general point cloud and VPS related limits for each level of each tier.

Table A-4 specifies the general atlas ASPS and tile related limits for each level of each tier.

Table A-5 specifies the general video bitstream related limits for each level of each tier.

A tier and level to which a bitstream conforms are indicated by the syntax elements ptl\_tier\_flag and ptl\_level\_idc. ptl\_level\_idc shall be set equal to a value of 30 times the level number specified in Table A-3, Table A-4, and Table A-5.

Table A-3 — General Point Cloud or VPS related level limits

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Level** |  | **Max atlas frame size**  **MaxAtlasSize [Ed. (JB): Do we want to constrain this?]** | **Max vps\_atlas\_count\_minus1** | **Max # of attributes**  **MaxNumAttributeCount**  **[Ed. (JB): Do we want to constraing this?]** |
| 1.0 |  | 2,228,224 | X | 1 |
| 2.0 |  | 2,228,224 | X | 3 |
| 3.0 |  | 8,912,896 | X | 1 |

[Ed. (All): Do the constraints on points per frame or per second make sense to be defined here? They also appear to be quite limiting given also the allowed atlas resolution and number of maps supported]

Table A-4 — General atlas ASPS and tile related level limits

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Level** | **Max # Patches**  **MaxNumPatches** | **Max # RAW points**  **MaxNumRawPoints** | **Max # EOM points**  **MaxNumEOMPoints** | **Max atlas bitrate**  **MaxAtlasBR** |
| 1.0 | 256 | 50000 | 50000 | 1,500,000 |
| 2.0 | 256 | 50000 | 50000 | 2,000,000 |
| 3.0 | ? | ? | ? |  |

Table A-5 — General video bitstream level limits

|  |  |  |  |
| --- | --- | --- | --- |
| **Level** | **Max aggregate sample rate**  **MaxAggregateSr (samples/sec)** | **Max # Mbit/s per video stream**  **MaxBitratePerStream** | **Max aggregate Mbit/s stream**  **MaxAggregateBitrate** |
| 1.0 | 133,693,440 | 50 | ? |
| 2.0 | 133,693,440 | 50 | ? |
| 3.X | 534,773,760 |  | 40 |

[Ed. (All): Do we specify samples per second rate and bitrate per stream or as an aggregate. See M50998]

[Ed. (All): Is the bitrate defined according to the High Tier of HEVC? It seems too high]

[Ed. (All): It was discussed whether we should consider having two level scenarios. A scenario controlled by individual bitstream decode capabilities and a second scenario controlled by aggregate bitstream capabilities. In the first, the limits are dictated for each sub-bitstream independently. For the second, the aggregate numbers are divided by the number of sub-bitstreams available. It might be desirable if both options are to be supported to either distinguish so through the level value or through an additional parameter in the profile/level syntax.]

1. Post-decoding conversion to nominal video formats

This clause does not apply to this version of the specification.

1. V-PCC Sample stream format

The specifications in [3VC] Annex C apply.

1. NAL Sample stream format

The specifications in [3VC] Annex D apply.

1. Atlas Hypothetical Reference Decoder

The specifications in [3VC] Annex E apply.

1. Supplemental enhancement information
   1. General

The specifications in [3VC] Annex F apply.

* 1. SEI payload syntax
     1. General SEI message syntax

|  |  |
| --- | --- |
| sei\_payload( payloadType, payloadSize ) { | **Descriptor** |
| if(( nal\_unit\_type  = =  NAL\_PREFIX\_NSEI )  | |  ( nal\_unit\_type  = =  NAL\_PREFIX\_ESEI )) { |  |
| if( payloadType  = =  0 ) |  |
| buffering\_period( payloadSize ) |  |
| else if( payloadType  = =  1 ) |  |
| atlas\_frame\_timing( payloadSize ) |  |
| else if( payloadType  = =  2 ) |  |
| filler\_payload( payloadSize ) |  |
| else if( payloadType  = =  3 ) |  |
| user\_data\_registered\_itu\_t\_t35( payloadSize ) |  |
| else if( payloadType  = =  4 ) |  |
| user\_data\_unregistered( payloadSize ) |  |
| else if( payloadType  = =  5 ) |  |
| recovery\_point( payloadSize ) |  |
| else if( payloadType  = =  6 ) |  |
| no\_display( payloadSize ) |  |
| else if( payloadType  = =  7 ) |  |
| time\_code( payloadSize ) |  |
| else if( payloadType  = =  8 ) |  |
| regional\_nesting( payloadSize ) |  |
| else if( payloadType  = =  9 ) |  |
| sei\_manifest( payloadSize ) |  |
| else if( payloadType  = =  10 ) |  |
| sei\_prefix\_indication( payloadSize ) |  |
| else if( payloadType  = =  11 ) |  |
| geometry\_transformation\_params( payloadSize ) |  |
| else if( payloadType  = =  12 ) |  |
| attribute\_transformation\_params( payloadSize ) |  |
| else if( payloadType  = =  13 ) |  |
| active\_sub\_bitstreams( payloadSize ) |  |
| else if( payloadType  = =  14 ) |  |
| component\_codec\_mapping( payloadSize ) |  |
| else if( payloadType  = =  15 ) |  |
| volumetric\_tiling\_info( payloadSize ) |  |
| else if( payloadType  = =  16 ) |  |
| presentation\_information( payloadSize ) |  |
| else if( payloadType  = =  17 ) |  |
| geometry\_smoothing( payloadSize ) |  |
| else if( payloadType  = =  18 ) |  |
| attribute\_smoothing( payloadSize ) |  |
| else if( payloadType  = =  64 ) |  |
| viewing\_space( payloadSize ) |  |
| else if( payloadType  = =  65 ) |  |
| rec\_viewport( payloadSize ) |  |
| else if( payloadType  = =  66 ) |  |
| viewing\_space\_handling( payloadSize ) |  |
| else if( payloadType  = =  67 ) |  |
| geometry\_upscaling\_parameters( payloadSize ) |  |
| else |  |
| reserved\_sei\_message( payloadSize ) |  |
| } |  |
| else { /\*( nal\_unit\_type  = =  NAL\_SUFFIX\_NSEI )  | |  ( nal\_unit\_type  = =  NAL\_SUFFIX\_ESEI )\*/ |  |
| if( payloadType  = =  2 ) |  |
| filler\_payload( payloadSize ) |  |
| else if( payloadType  = =  3 ) |  |
| user\_data\_registered\_itu\_t\_t35( payloadSize ) |  |
| else if( payloadType  = =  4 ) |  |
| user\_data\_unregistered( payloadSize ) |  |
| else |  |
| reserved\_sei\_message( payloadSize ) |  |
| } |  |
| if( more\_data\_in\_payload( ) ) { |  |
| if( payload\_extension\_present( ) ) |  |
| **sp\_reserved\_payload\_extension\_data** | u(v) |
| byte\_alignment( ) |  |
| } |  |
| } |  |

* + 1. Viewing space SEI message syntax
       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 ) |  |
| } |  |
| } |  |

* + - 1. Elementary shape

|  |  |
| --- | --- |
| elementary\_shape( e ) { | **Descriptor** |
| **es\_num\_primitive\_shapes\_minus\_1**[ 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 <= num\_primitive\_shapes\_minus1; s++ ) { |  |
| if( es\_camera\_inferred\_flag[ e ] == 1 ) |  |
| **es\_view\_idx**[ e ][ s ] | u(v) |
| **es\_primitive\_shape\_type**[ e ][ s ] | u(2) |
| if( primitive\_shape\_type[ e ][ s ] == 0) |  |
| cuboid\_primitive (e , s) |  |
| else if( primitive\_shape\_type[ e ][ s ] == 1 ) |  |
| spheroid\_primitive (e, s) |  |
| else if( primitive\_shape\_type[ e ][ s ]== 2 ) |  |
| halfspace\_primitive(e, s) |  |
| if( guard\_band\_present\_flag[ e ] ) |  |
| **es\_guard\_band\_size**[ e ][ s ] | fl(16) |
| if( primitive\_orientation\_present\_flag[ e ] ) { |  |
| if( es\_camera\_inferred\_flag[ e ] == 0 ) { |  |
| **es\_primitive\_shape\_quat\_x**[ e ][ s ] | fl(16) |
| **es\_primitive\_shape\_quat\_y**[ e ][ s ] | fl(16) |
| **es\_primitive\_shape\_quat\_z**[ e ][ s ] | fl(16) |
| } |  |
| } |  |
| if( viewing\_direction\_constraint\_present\_flag[ e ] ) { |  |
| if( guard\_band\_present\_flag[ e ]) |  |
| **es\_guard\_band\_direction\_size**[ e ][ s ] | fl(16) |
| if( camera\_inferred\_flag[ e ] == 0 ) { |  |
| **es\_primitive\_shape\_viewing\_direction\_quat\_x\_center**[ e ][ s ] | fl(16) |
| **es\_primitive\_shape\_viewing\_direction\_quat\_y\_center**[ e ][ s ] | fl(16) |
| **es\_primitive\_shape\_viewing\_direction\_quat\_z\_center**[ e ][ s ] | fl(16) |
| } |  |
| **es\_primitive\_shape\_viewing\_direction\_yaw\_range**[ e ][ s ] | fl(16) |
| **es\_primitive\_shape\_viewing\_direction\_pitch\_range**[ e ][ s ] | fl(16) |
| } |  |
| } |  |
| } |  |

* + - 1. Cuboid primitive

|  |  |
| --- | --- |
| cuboid\_primitive( e, s ) { |  |
| if( camera\_inferred\_flag[ e ] == 0 ) { |  |
| **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) |
| } |  |

* + - 1. Spheroid primitive

|  |  |
| --- | --- |
| spheroid\_primitive( e, s ) { | **Descriptor** |
| if( camera\_inferred\_flag[ e ] == 0 ) { |  |
| **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) |
| } |  |

* + - 1. 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) |
| } |  |

* + 1. Recommended viewport SEI message syntax

|  |  |
| --- | --- |
| rec\_viewport( payloadSize ) { | **Descriptor** |
| **rec\_viewport\_id** | u(10) |
| **rec\_viewport\_cancel\_flag** | u(1) |
| if( !rec\_viewport\_cancel\_flag ) { |  |
| **rec\_viewport\_persistence\_flag** | u(1) |
| **rec\_viewport\_center\_view\_flag** | u(1) |
| if( !rec\_viewport\_center\_flag ) |  |
| **rec\_viewport\_left\_view\_flag** | u(1) |
| **rec\_viewport\_pos\_x** | fl(32) |
| **rec\_viewport\_pos\_y** | fl(32) |
| **rec\_viewport\_pos\_z** | fl(32) |
| **rec\_viewport\_quat\_x** | fl(32) |
| **rec\_viewport\_quat\_y** | fl(32) |
| **rec\_viewport\_quat\_z** | fl(32) |
| **rec\_viewport\_hor\_range** | fl(32) |
| **rec\_viewport\_ver\_range** | fl(32) |
| } |  |
| } |  |

* + 1. 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) |
| } |  |
| } |  |

* + 1. 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) |
| } |  |
| } |  |

* 1. SEI payload semantics
     1. General SEI message semantics
     2. Viewing space SEI message semantics
        1. General

The viewing space indicates the portion of the space, possibility completed by viewing direction constraints, where the viewport can be rendered with high quality. It is based on the possibility given to the end device to compute a fading indice between 0 (no fading) and 1 (full fading) of inclusiveness inside this viewing space. The end device application can use this index to implement a fade out when the viewport leaves the viewing space.

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, substraction, 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 0specifies that the type of CSG operation to apply on the elementary shape e is additive. vs\_elementary\_shape\_operation [ e ]equal to 1specifies that the type of CSG operation to apply on the elementary shape e is substractive. vs\_elementary\_shape\_operation [ e ]equal to 2specifies that the type of CSG operation to apply on the elementary shape e is intersection. The operation consists of computing a signed distance of a point p related to shape S and combining that with the signed distance of the entire accumulated viewing space.

**vs\_num\_elementary\_shapes\_minus1** plus 1 indicates the number of elementary shapes to build the viewing space. When there is only one elementary shape, there is no interpolation in the case of the interpolation operation mode.

* + - 1. Elementary shape

**es\_num\_primitive\_shapes\_minus1**[ e ] plus 1 specifies the number of primitive shapes that is 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 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 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 of a point p related to a shape S center which has been shifted along the path. The shift value is a linear operation between regular distances and to the two closest successive primitive shapes and . The interpolated elementary shapes are combined additively into the viewing space.

**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 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 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 cameras with indices es\_view\_idx[ e ][  s ] in the miv\_view\_param\_list( ).

**es\_primitive\_shape\_type**[ e ][  s ]indicates the type of primitive shape s of the elementary shape e detailed below as in the following table.

Table F-1 : primitive\_shape\_type

|  |  |
| --- | --- |
| **es\_primitive\_type** | **Shape** |
| 0 | cuboid\_primitive |
| 1 | sphere\_primitive |
| 2 | halfspace\_primitive |
| 3 | Reserved for future use by ISO/IEC |

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 ] is a 16-bit floating-point value that specifies the width of the positional guard band for each primitive shape s of an elementary shape e. es\_guard\_band\_present 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 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, SGUARD).*

From these individual es\_guard\_band\_size[ e ][ s ] defined for each primitive shape s of an elementary shape e, a signed distance is computed for the elementary shape e. From these signed distance of each elementary shape, a global signed distance is computed for the whole viewing space. The index of positional fading within the global viewing space is then computed as shown in the following equation.

Equation F‑1 : *position*\_*fading index (p)= clamp((SD(p)+guard\_band\_size[e][s]) / guard\_band\_size[e][s], 0, 1)*

where p is the vector of coordinates of the viewport, S is the primitive shape s of the elementary shape e, SD(p, S) the signed distance of p to S and *guard\_band\_size* the global guard band size value.

**es\_primitive\_shape\_quat\_x**[ e ][ s ], **es\_primitive\_shape\_quat\_y**[ e ][ s ] and **es\_primitive\_shape\_quat\_z**[ e ][ s ] are 16-bit floating point value that gives respectively the x, y and z component of a rotation quaternion to apply on the primitive shape s of the elementary shape e. 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 )*. The value of these parameters shall be a floating-point value in the range of -1 to 1, inclusive.

**es\_guard\_band\_direction\_size**[ e ][ s ] is a 16-bit floating-point value that specifies the width of the directional guard band for each primitive shape s of an elementary shape e . es\_guard\_band\_present 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 ], is a 16-bit fixed floating point value giving the x quaternion component of suggested viewing directions center for the primitive shape s. The value of this parameter shall be a floating-point value in the range of -1 to 1, inclusive.

**es\_primitive\_shape\_viewing\_direction\_quat\_y\_center**[ e ][ s ] is a 16-bit fixed floating point value giving the y quaternion component of suggested viewing directions center for the primitive shape s of the elementary shape e. The value of this parameter shall be a floating-point value in the range of -1 to 1, inclusive.

**es\_primitive\_shape\_viewing\_direction\_quat\_z\_center**[ e ][ s ] is a 16-bit fixed floating point value giving the z quaternion component of suggested viewing directions center for the primitive shape s of the elementary shape e. The value of this parameter shall be a floating-point value in the range of -1 to 1, inclusive.

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

**es\_primitive\_shape\_viewing\_direction\_yaw\_range**[ e ][ s ] is a 16-bit fixed floating point value giving the yaw half range of suggested viewing directions for the s-th primitive shape.

**es\_primitive\_shape\_viewing\_direction\_pitch\_range**[ e ][ s ] is a 16-bit fixed floating point value giving the pitch half range of suggested viewing directions for the s-th primitive shape.

The viewing direction constraints (center, range and directional guard band) together define the viewing space constraints *V(p)*at point p.

When primitive\_shape\_operation equal 0 (operation on shapes based on CSG), these are interpolated for a given point *p* and all elementary shape *Si* and related signed distance *SD(p, Si)*as

Equation F‑2: *V(p) = ∑-SD(p, Si)Vi(p)/ ∑-SD(p, Si)*

When primitive\_shape\_operation equal 1 (operation on shapes based on interpolation), the above equation reduces to a linear interpolation between the two closest primitive shapes and taken in the order of the primitive\_shape list with the use of the regular distance *RD(p, Si)*.

Equation F-3: *V(p) = ( RD(p, Ss+1)Vs + RD(p, Ss)Vs+1  ) / ( RD(p, S)+ RD(p, Ss+1))*

*V(p)* gives the viewing direction center, range and directional guard band direction size at a given viewport position *p* and orientation *yaw* and *pitch*. The index of directional fading for yaw is then computed as shown in the following equation ( the equivalent equation for directional fading for pitch applies by replacing yaw by pitch ):

Equation F-4: *yaw fading index (p)= clamp((abs(yaw - viewing\_yaw\_center (p) )- viewing\_yaw\_range (p) + guard\_band\_direction\_size (p)) / guard\_band\_direction\_size (p), 0, 1)*

where *yaw* is the yaw value of the viewport quaternion, *viewing\_yaw\_center* is the yaw value of the direction center quaternion, *viewing\_yaw\_range* is the direction range in yaw, *guard\_band\_direction\_size* is the directional guard band size at that viewport position *p*.

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.

* + - 1. Cuboid primitive

**cp\_center\_x**[ e ][ s ], **cp\_center\_y**[ e ][  s ], **cp\_center\_min\_z**[ e ][  s ] are 16-bit floating-point values that specifies respectively the x, y, z co-ordinates 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 ] is a 16-bit floating-point value that specifies the size of the cuboid in x, y, z directions in the scene coordinate system.

The signed distance function for a cuboid primitive is

Equation F-5: *SDCUBOID(p, l, h) =* min(max(*dx*, max(*dy, dz*)), *0*) + |max(*d, 0*)|

where (*dx*, *dy*, *dz*) are the co-ordinates of the point as regards to the primitive shape center, *l* is the 3D vector *(center\_x – size\_x/2, center\_y – size\_y/2, center\_z – size\_z/2)*, *h* is *(center\_x + size\_x/2, center\_y + size\_y/2, center\_z + size\_z/2)*, and *d* is max(*l – p, p – h*). The max operations on vectors are to be applied per element.

* + - 1. Spheroid primitive

**sp\_center\_x**[ e ][  s ], **sp\_center\_y**[ e ][  s ], **sp\_center\_min\_z**[ e ][ s ] are 16-bit floating-point values that specifies respectively the x, y, z co-ordinates 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 ], are a 16-bit floating-point values that specifies the dimension x, y and z respectively of the spheroid in the scene coordinate system.

The signed distance function for a spheroid primitive is

Equation F-6: *SDSPHEROID(p, r) = |p/r| \* (|p/r| - 1) / |p/r2|*

where the 3D point *p* is as regards to the primitive center *center\_x, center\_y, center\_z, r* equals the (*radius\_x, radius\_y, radius\_z)* vector, and the division operation is applied per vector element.

* + - 1. Half space primitive

**hp\_normal\_x**[ e ][  s ], **hp\_normal\_y**[ e ][  s ], **hp\_normal\_z**[ e ][ s ] are 16-bit floating-point values that indicate the normal facing of the plane defining the half-space.

**hp\_distance**[ e ][ s ] is a 16-bit floating-point value that 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 is

Equation F-7: *SDHALFSPACE(p, n, d) =* dot(*p, n / |n|) – d*

where *n* is the normal vector given by (normal\_x, normal\_y, normal\_z) and *d* equals *distance*.

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

* + 1. Recommended viewport SEI message semantics

**rec\_viewport\_id** contains an identifying number that may be used to identify a recommended viewport.

**rec\_viewport\_cancel\_flag** equal to 1 indicates that the SEI message cancels the persistence of any previous recommended viewport SEI message in output order. rec\_viewport\_cancel\_flag equal to 0 indicates that recommended viewport information follows.

**rec\_viewport\_persistence\_flag** specifies the persistence of the recommended viewport SEI message for the current layer.

rec\_viewport\_persistence\_flag equal to 0 specifies that the recommended viewport SEI message applies to the current decoded picture only.

Let picA be the current picture. rec\_viewport\_persistence\_flag equal to 1 specifies that the recommended viewport SEI message persists for the current layer in output order until one or more of the following conditions are true:

* A new CLVS of the current layer begins.
* The bitstream ends.
* A picture picB in the current layer in an access unit containing an recommended viewport SEI message that is applicable to the current layer is output for which PicOrderCnt( picB ) is greater than PicOrderCnt( picA ), where PicOrderCnt( picB ) and PicOrderCnt( picA ) are the PicOrderCntVal values of picB and picA, respectively, immediately after the invocation of the decoding process for picture order count for picB.

**rec\_viewport\_center\_view\_flag** equal to 1 indicates that the viewport parameters signaled correspond to the center of the recommended viewport. rec\_viewport\_center\_view\_flagequal to 0 indicates that the viewport parameters signaled correspond to one of two stereo positions of the recommended viewport.

**rec\_viewport\_left\_view\_flag** equal to 1 indicates that the viewport parameters signaled correspond to the left stereo position of the recommended viewport. rec\_viewport\_left\_view\_flagequal to 0 indicates that the viewport parameters signaled correspond to the right stereo positions of the recommended viewport.

**rec\_viewport\_pos\_x** indicates a recommended viewport position in meters the x coordinate in the global reference coordinate system. [Ed. (JB): Does this need any change for the coordinate system?]

**rec\_viewport\_pos\_y** indicates a recommended viewport position in meters the y coordinate in the global reference coordinate system.

**rec\_viewport\_pos\_z** indicates a recommended viewport position in meters the z coordinate in the global reference coordinate system.

**rec\_viewport\_quat\_x, rec\_viewport\_quat\_y,** and **rec\_viewport\_quat\_z** indicate the x, y, and z components , respectively, of the rotation of the recommended viewport region using the quaternion representation. The values of rec\_viewport\_quat\_x, rec\_viewport\_quat\_y, and rec\_viewport\_quat\_z shall be a floating-point value in the range of −1 to 1, inclusive.

**rec\_viewport\_hor\_range** indicates the horizontal size of the recommended viewport region, in units of radians. The value of rec\_viewport\_hor\_range shall be in the range of 0 to 2π.

**rec\_viewport\_ver\_range**[ i ] indicates the vertical size of the recommended viewport region, in units of radians. The value of rec\_viewport\_ver\_range shall be in the range of 0 to π.

* + 1. 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 ] are specified in Table F- 2. 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 ] are specified in Table F- 3. 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 ] are specified in Table F- 3. 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- 2: 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\_UNSPCF\_63 | Unspecified (available for specification by other standards) |

Table F- 3: 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- 4: 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) |

* + 1. Geometry upscaling parameters SEI payload semantics

**gup\_type** is the type of geometry upscaling to which the provided parameters apply. This version of the standard defines the value 0 in accordance with the geometry video scaling process in clause (G.2). 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 (G.2.5) 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 (G.2.5) 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 (G.2.6) 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.

1. Hypothetical view rendering process (Informative)

This annex does not form an integral part of this Specification.

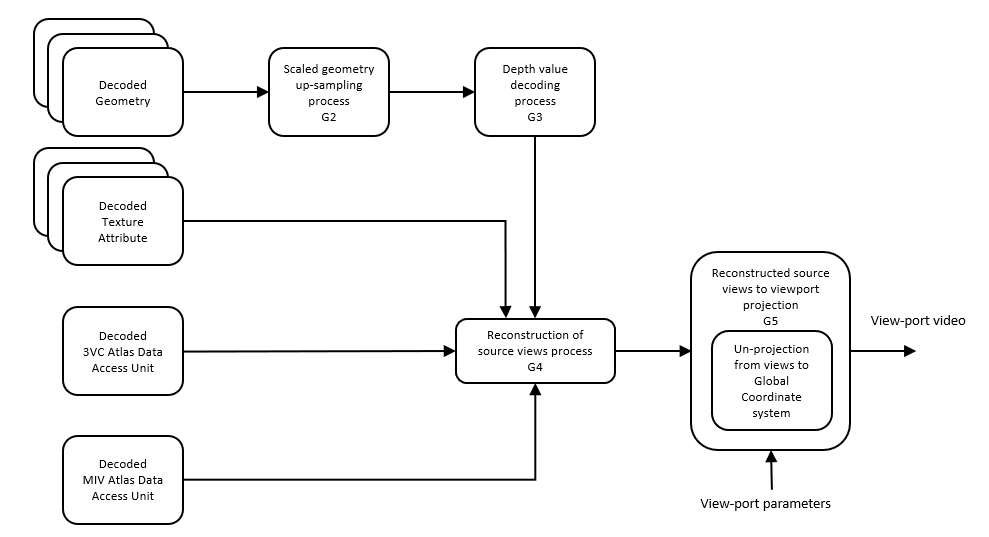


Figure G.1: Block diagram of hypothetical view renderer

* 1. General

Inputs to this process are the following for each access unit:

* the number of atlases, vps\_atlas\_count\_minus1
* for each of the atlases, a:
  + one decoded geometry picture of size MaspGeometryFrameWidth[ a ] x MaspGeometryFrameHeight[ a ]
  + one texture attribute picture of size AspsFrameWidth[ a ]  x AspsFrameHeight[ a ]
  + one BlockToPatchMap of size (AspsFrameWidth[ a ] /  AtlasPatchPackingBlockSize[ a ]) x ( AspsFrameHeight[ a ]  / AtlasPatchPackingBlockSize[ a ] )
  + masp\_auxiliary\_atlas\_flag[ a ]
* the maximum number of views, mvp\_num\_views\_minus1
* the intrinsic and extrinsic parameters per view
* parameters related to depth quantization from the atlas adaptation parameters set and Atlas tile group data unit [Ed. (JB): List specifically any other V-PCC parameters needed.]
* the number of atlas groups, num\_groups, and a group id value per atlas, group\_id[ a ]
* a viewing position (x, y, z)
* a viewing orientation (quat\_x, quat\_y, quat\_z)
* a viewport picture width, picW, and picture height, picH
* an array of flags indicating if each group is targeted for decode, TargetGroupFlag[ ], of length msp\_num\_groups\_minus1+ 1
* msp\_depth\_low\_quality\_flag

Outputs of this process are a sequence of viewport pictures of resolution picW by picH.

The rendering process operates as follows:

1. The geometry video scaling process as specified in in clause G.2 is invoked for each atlas.
2. The depth decoding process as specified in clause G.3 is invoked for each atlas, for each sample in the geometry decoded and upscaled picture of size AspsFrameFrameWidth[ a ] x AspsFrameFrameHeight[ a ] with a metric depth map and occupancy map dual output.
3. The reconstructed view process as specified in clause G.4 is invoked for each view.
4. The projection of pixels of reconstructed view to viewport process as specified in clause G.5 is invoked for reconstructed view v in 0 .. mvp\_num\_views\_minus1.
   1. Geometry video scaling process

The geometry video scaling process reconstructs a geometry frame at nominal atlas resolution. The assumption is that the encoder has downscaled a nominal resolution geometry frame using a max filter.

The inputs of this process are DecGeoFrame[ 0 ][ orderIdx ][ 0 ][ y ][ x ] and atlas parameters for the a-th atlas.

The output of the scaled geometry decoding process is GeoFrame[ mapIdx ][ orderIdx ][ compIdx ][ y ][ x ] where mapIdx equal to 0, and compIdx equal to 0.

If AspsFrameWidth[ a ] == MaspGeometryFrameWidth[ a ] and AspsFrameHeight[ a ] == MaspGeometryFrameHeight[ a ], then the following procedure applies:

for( y = 0; y < AspsFrameHeight[ a ]; y++ ) {

for( x = 0; x < AspsFrameWidth[ a ]; x++) {

GeoFrame[ 0 ][ orderIdx ][ 0 ][ y ][ x ] = DecGeoFrame[ 0 ][ orderIdx ][ 0 ][ y ][ x ]

}

}

Otherwise this process invokes the following sequence of sub processes to derive its output:

1. The nearest neighbour interpolation scaling process (section G.2.1) is invoked.
2. The texture aligned geometry erosion process (section G.2.5) is invoked.
3. The geometry contour smoothening process (section G.2.6) is invoked.

The sample neighbours enumeration process (section G.2.2), foreground edge flag process (section G.2.3), and selective geometry erosion process (section G.2.4) are used within the sub processes.

* + 1. Nearest neighbour interpolation scaling process

This process scales the geometry frame at decoded size DecGeoFrame[ 0 ][ orderIdx ][ 0 ][ y ][ x ] to a geometry frame at nominal atlas size using nearest neighbour interpolation. The output of this process ScaledGeoFrame[ y ][ x ] is derived as follows:

for( y = 0; y < AspsFrameHeight[ a ]; y++ ) {

for( x = 0; x < AspsFrameWidth[ a ]; x++) {

v = y / MaspGeometryFrameScaleFactorY[ a ]

u = x / MaspGeometryFrameScaleFactorX[ a ]

ScaledGeoFrame[ y ][ x ] = DecGeoFrame[ 0 ][ orderIdx ][ 0 ][ v ][ u ]

}

}

* + 1. Sample neighbours enumeration process

The sample neighbours enumeration process provides a list of neighbouring sample positions that are within the same patch and within the frame size.

Input to this process is:

* A sample position (x, y)
* BlockToPatchMap[ y ][ x ] and AtlasPatchPackingBlockSize[ a ] for atlas a
* Connectivity is a global parameter with value equal to 4 or 8 or 24 and specifies the maximum number of sample neighbours in a square kernel footprint of respectively 3x3or 5x5

Output of this process is a list of neighbouring sample positions that are within the same patch:

* NgX[ i ] is the x-sample position of the i'th neighbour
* NgY[ i ] is the y-sample position of the i'th neighbour
* NumNeighbours is the number of neighbours

The process is specified as follows:

n = AtlasPatchPackingBlockSize[ a ]

NumNeighbours = 0

if( Connectivity == 4 ) {

kx = [ -1, 0, 1, 0 ]

ky = [ 0,-1, 0,-1 ]

} else if ( Connectivity == 8 )

kx = [ 0, 1, 1, 1, 0,-1,-1,-1 ]

ky = [ -1,-1, 0, 1, 1, 1, 0,-1]

} else /\* Connectivity == 24 \*/

kx = [ 0, 1, 1, 1, 0,-1,-1,-1, 0, 1, 2, 2, 2, 2, 2, 1, 0,-1,-2,-2,-2,-2,-2,-1 ]

ky= [-1,-1, 0, 1, 1, 1, 0,-1,-2,-2,-2,-1, 0, 1, 2, 2, 2, 2, 2, 1, 0,-1,-2,-2 ]

}

for( i = 0; i < Connectivity; i++ ) {

if( 0 <= x + kx[i] && x + kx[i] < AspsFrameWidth[ a ] &&

0 <= y + ky[i] && y + ky]i] < AspsFrameHeight[ a ] &&

BlockToPatchMap[ y / n ][ x / n ] == BlockToPatchMap[ (y + ky[i]) / n ][ (x + kx[i]) / n ] } {

NgX[ NumNeighbours ] = x + kx[i]

NgY[ NumNeighbours ] = y + ky[i]

NumNeighbours++

}

}

* + 1. Foreground edge flag process

This process determines if a sample is a foreground edge. Because this process is used multiple times within the scaled geometry video scaling process the input frame has a generic name. Input to this process are:

* A sample position (x, y)
* InputFrame[ y ][ x ] is a geometry frame

Output of this process is ForegroundEdgeFlag[ y ][ x ].

This process invokes the sample neighbours enumeration process (section G.2.2). The process is defined by the following procedure, where NgX, NgY and NumNeighbours are computed according to G3.2 with Connectivity equal to 4:

ForegroundEdgFlag[ y ][ x ] = 0

for( i = 0; i < NumNeighbours; i++ ) {

if( GupDeltaThreshold <= InputFrame[ y ][ x ] - InputFrame[ NgY[ i ] ][ NgX[ i ] ] )

ForegroundEdgFlag[ y ][ x ] = 1

}

* + 1. Selective geometry erosion process

This process selectively erodes a geometry sample. Because this process is used multiple times within the scaled geometry video scaling process the input and output frame have a generic name.

Input to this process are:

* The sample position (x, y)
* InputFrame[ y ][ x ] is a geometry frame
* ErodeFlag[ y ][ x ] determines if the sample at (x, y) has to be eroded

Output of this process is OutputFrame[ y ][ x ].

This process invokes the sample neighbours enumeration process (section G.2.2). This process is defined by the following procedure, where NgX, NgY and NumNeighbours are computed according to G.2.2 with Connectivity equal to 8:

OutputFrame[ y ][ x ] = InputFrame[ y ][ x ]

if( ErodeFlag[ y ][ x ] ) {

for( i = 0; i < NumNeighbours; i++ ) {

OutputFrame[ y ][ x ] = Min( InputFrame[ y ][ x ], OutputFrame[ y ][ x ] )

}

}

* + 1. Texture aligned geometry erosion process

This process selectively erodes a geometry frame to align it with the texture attribute frame. Inputs to this process are:

* The sample position (x, y)
* ScaledGeoFrame[ y ][ x ]
* AttrFrame[ attrIdx ][ mapIdx ][ orderIdx ][ compIdx ][ y ][ x ] with attrIdx equal to 0 and mapIdx equal to 0

This process invokes the foreground edge frame process (section G.2.3) with ScaledGeoFrame[ y ][ x ] as InputFrame[ y ][ x ], providing ForegroundEdgeFlag[ y ][ x ] to this process.

Output of this process is TextureAlignedGeoFrame[ y ][ x ].

This process invokes the sample neighbours enumeration process (section G.2.2), and the selective geometry erosion process (section G.2.5) with ScaledGeoFrame[ ][ ] as InputFrame[ ][ ] and TextureAlignedGeoFrame[ ][ ] as OutputFrame[ ][ ]. The ErodeFlag[ y ][ x ] input to the selective geometry erosion process is derived using the following procedure, where NgX, NgY and NumNeighbours are computed according to G.2.2 with Connectivity equal to 24:

if( ForegroundEdgeFlag[ y ][ x ] == 0 )

ErodeFlag[ y ][ x ] = 0

else {

countForeground = 0

countBackground = 0

sadForeground = 0

sadBackground = 0

for( i = 0; i < NumNeighbours; i++ ) {

if( ForegroundEdgeFlag[ NgY[ i ] ][ NgX[ i ] ] == 0 ) {

for( c = 0; c <= AiAttributeDimension [ a ][ 0 ]; c++ )

sad += Abs(AttrFrame[ 0 ][ 0 ][ orderIdx ][ c ][ NgY[ i ] ][ NgX[ i ] ]

- AttrFrame[ 0 ][ 0 ][ orderIdx ][ c ][ y ][ x ])

if( GupDeltaThreshold   
 <= ScaledGeoFrame[ y ][ x ] - ScaledGeoFrame[ NgY[ i ] ][ NgX[ i ] ] )

countBackground += 1

sadBackground += sad

}

else if( GupDeltaThreshold

>= ScaledGeoFrame[ NgY[ i ] ][ NgX[ i ] ] - ScaledGeoFrame[ y ][ x ] )

countForeground += 1

sadForeground += sad

}

}

}

}

ErodeFlag[ y ][ x ] = sadForeground \* countBackground > GupErodeThreshold \* sadBackground \* countForeground ? 1 : 0

}

* + 1. Geometry contour smoothening process

This process smoothens the contours in a geometry frame to improve geometry edge stability. Inputs to this process are:

* The sample position (x, y)
* TextureAlignedGeoFrame[ y ][ x ]
* AttrFrame[ attrIdx ][ mapIdx ][ orderIdx ][ compIdx ][ y ][ x ] with attrIdx equal to 0 and mapIdx equal to 0

This process invokes the foreground edge frame process (section G.2.3) with TextureAlignedGeoFrame[ y ][ x ] as InputFrame[ y ][ x ], providing ForegroundEdgeFlag[ y ][ x ] to this process.

Output of this process is GeoFrame[ y ][ x ].

This process also invokes the sample neighbours enumeration process (section G.2.2), and the selective geometry erosion process (section G.2.4) with TextureAlignedGeoFrame[ ][ ] as InputFrame[ ][ ] and GeoFrame[ ][ ] as OutputFrame[ ][ ]. The ErodeFlag[ y ][ x ] input to the selective geometry erosion process is derived using the following procedure, where NgX, NgY and NumNeighbours are computed according to G.2.2 with Connectivity equal to 8:

if( ForegroundEdgeFlag[ y ][ x ] == 0 )

ErodeFlag[ y ][ x ] = 0

else {

countBackground = 0

for( i = 0; i < NumNeighbours; i++ ) {

if( ForegroundEdgeFlag[ NgY[ i ] ][ NgX[ i ] ] == 0 ) {

if( TextureAlignedGeoFrame[ orderIdx ][ NgY[ i ] ][ NgX[ i ] ]   
 <= TextureAlignedGeoFrame[ orderIdx ][ y ][ x ] - GupDeltaThreshold )

countBackground += 1

}

}

ErodeFlag[ y ][ x ] = countBackground > GupMaxCurvature ? 1 : 0

}

* 1. Depth decoding process

This process converts a sample of the decoded geometry frame upscaled at nominal atlas resolution to a floating-point depth value in meters.

Inputs to this process are:

* The sample position (x, y)
* GeoFrame[ 0 ][ orderIdx ][ 0 ][ y ][ x ]
* BlockToPatchMap[ y ][ x ] and AtlasPatchPackingBlockSize[ a ] for atlas a
* Parameters from the atlas adaptation parameters set and Atlas tile group data unit

Output of this process are:

* MetricDepth[ y ][ x ] , the metric depth value
* OccupancyValue[ y ][ x ], the occupancy map value

Regarding the metric depth generation, a normalized depth value DepthAtlasNormValue[ y ][ x ] is first derived as follows:

MaxDepthSampleValue = 1 << gi\_geometry\_nominal\_2d\_bitdepth\_minus1[ a ] – 1

bSz = AtlasPatchPackingBlockSize[ a ]

p = BlockToPatchMap[ y / bSz ][ x / bSz ]

ClampedDepthSample[ y ][ x ] = Clip(GeoFrame[ 0 ][ orderIdx ][ 0 ][ y ][ x ], PduDepthStart[ a ][ p ],

PduDepthEnd[ a ][ p ] )

DepthAtlasNormValue[ y ][ x ] = ClampedDepthSample[ y ][ x ] ÷ MaxDepthSampleValue

Then normalized disparity, NormDisp[ y ][ x ] , is derived as follows:

v = pdu\_view\_id[ a ][ p ]  
NormDisp[ y ][ x ] = dq\_norm\_disp\_low[ v ] + (dq\_norm\_disp\_high[ v ] – dq\_norm\_disp\_low[ v ])  
 \* DepthAtlasNormValue[ y ][ x ]

Finally, MetricDepth[ y ][ x ] is derived as follows:

MetricDepth[ y ][ x ] = 1.0 ÷ NormDisp[ y ][ x ]

Regarding the occupancy map value generation, the threshold value is first determined as follows:

If masp\_depth\_occ\_map\_threshold\_flag is equal to 0, the variable DepthOccupancyThreshold is set equal to dq\_depth\_occ\_map\_threshold\_default[ v ].

Otherwise (masp\_depth\_occ\_map\_threshold\_flag is equal to 1), the variable DepthOccupancyThreshold is set equal to pdu\_depth\_occ\_threshold[ p ].

The value of OccupancyValue[ y ][ x ] is then derived as follows:

If( DepthAtlasSample [ y ][ x ] < DepthOccupancyThreshold )

OccupancyValue[ y ][ x ] = 0

else

OccupancyValue[ y ][ x ] = 1

* 1. Reconstruction of reconstructed views process

Inputs to this process are:

* miv\_view\_params\_list parameters related to view v
* metric depth decoded pictures MetricDepth[ a ] of all atlases, as defined in clause G.3
* occupancy value map OccupancyValue[ a ] for all atlases, as defined in clause G.3
* decoded texture attribute pictures of all atlases
* atlas parameters for all atlases

Output of this process is the reconstructed view v of size (ci\_projection\_plane\_width\_minus1[ v ] + 1) x (ci\_projection\_plane\_width\_minus1[ v ] + 1) composed of:

* ReconstructedDepth[ v ]
* ReconstructedTextureAttribute[ v ]

The reconstructed depth for the view v ReconstructedDepth[ v ] is generated by first setting all samples to InvalidDepth (NaN) value. Then browsing all patches p of all atlases a and moving depth samples belonging to patch p in decoded picture MetricDepth[ a ] to their corresponding position in view v, if pdu\_view\_id[ p ] is equal to v and occupancy value is equal to 1.

The reconstructed texture attribute ReconstructedTextureAttribute[ v ] is produced by fetching the texture value at same coordinates.

* 1. Projection of pixels of reconstructed views to viewport
     1. Group-based initialization process

[Ed. (JB): To be provided.]

* + 1. Group-based merging process

[Ed. (JB): To be provided.]

* + 1. General projection of pixels of reconstructed views to viewport process

Inputs to this process are:

* ReconstructedDepth[ v ] and ReconstructedTextureAttribute[ v ] for view v as defined in clause G.4
* camera parameters list from the miv\_view\_params\_list
* target viewport size, position and orientation

Output to this process is:

* a viewport texture frame of size picW x picH

First, the reconstructed views are deprojected according to clause G.5.4.

Second, a weight of each input views with respect to target viewport, viewWeight[ v ], is generated by a function of the distance between view position and target viewport position.

The visibility pass first generates a depth map for the target viewport. Then the shading step computes the target viewport texture. Each not pruned pixel of the reconstructed views is blended into the target viewport with a contribution / weight taking into account its consistency with the visibility map and the weight of the view it belongs to.

For each sample in the viewport texture frame of size picW x picH, the pixel weighting recovery process of clause G.5.5 is invoked.

* + 1. Local views coordinate to global coordinate point unprojection process

Inputs to this process are:

* the depth component of reconstructed view v, ReconstructedDepth[ v ]
* camera parameters of view v

Output of this process is a global coordinate position map for v-th reconstructed view. GlobalCoordinatePositionMap[ v ][ i ][ j ] is a tuple of floating-point values (x, y, z) in the global coordinate system.

Not all positions (i, j) can be mapped to valid global coordinates. Invalid coordinates are set to InvalidCoordinate, where InvalidCoordinate = (NaN, NaN, NaN).

GlobalCoordinatePositionMap[ v ][ i ][ j ] is derived as follows.

pictureWidth = ci\_projection\_plane\_width\_minus1[ v ] + 1;

pictureHeight = ci\_projection\_plane\_height\_minus1[ v ] + 1;

for( i = 0; i < pictureWidth; i++ )  
 for( j = 0; j < pictureHeight; j++ )  
 GlobalCoordinatePositionMap[ v ][ i ][ j ] = InvalidCoordinate  
 if (ReconstructedDepth[ v ][ i ][ j ] != InvalidDepth )

if (cam\_type[ v ] == 0)  
 GlobalCoordinatePositionMap[ v ][ i ][ j ]  
 = LocalToGlobal(v, UnprojectERP( i, j, v, ReconstructedDepth[ v ][ i ][ j ] ))

if (cam\_type[ v ] == 1)  
 GlobalCoordinatePositionMap[ v ][ i ][ j ]  
 = LocalToGlobal(v, UnprojectPSP( i, j, v, ReconstructedDepth[ v ][ i ][ j ] ))

if (cam\_type[ v ] == 2)  
 GlobalCoordinatePositionMap[ v ][ i ][ j ]  
 = LocalToGlobal(v, UnprojectORT( i, j, v, ReconstructedDepth[ v ][ i ][ j ] ))

LocalToGlobal( v, (x, y, z) ) is derived as follow:

LocalToGlobal( v, ( x, y, z) ) = 3 first coordinates of ViewToScene[ v ] \* (x, y, z, 1)

Where ViewToScene is the complete transformation matrix from the view coordinate system to the scene coordinate system and is defined as follows:

The different functions UnprojectERP(), UnprojectPSP() and UnprojectORT() are described in the sub-paragraphs G.5.4.1, G.5.4.2 and G.5.4.3 respectively.

* + - 1. ERP unprojection process

Inputs to this process are:

* a depth r at column m and row n in view v
* view\_id v

Output of this process is a point in 3D scene UnprojectERP( m, n, v, r ) with m the image column, n the image row, v the view\_id, and r a floating-point depth value in units of meters. UnprojectERP( m, n, v, r ) is a tuple of floating-point values (x, y, z) in the global coordinate system and is derived as follows:

UnprojectERP( m, n, v, r ) = (  
 r \* Cosd(Theta( n, v )) \* Cosd(Phi( m, v )),  
 r \* Cosd(Theta( n, v )) \* Sind(Phi( m, v )),  
 r \* Sind(Theta( n, v )))

[Ed. (JB: Can any language from OMAF or the HEVC omnidirectional SEI messages be used here, although they are aimed as spheres?] [Ed.(BK): Fixed that by passing the depth value in meters in as an argument.] [Ed.(BK): It might be possible to re-use part of OMAF (not sure), but we should at least reference OMAF to state equivalance. I put some comments to indicate what parts are shared with OMAF.]

Hereby Phi( m, v ) and Theta( n, v ) map to spherical coordinates:

Phi( m, v ) = ci\_erp\_phi\_max[ v ] - (m + 0.5) \* (ci\_erp\_phi\_max[ v ] – ci\_erp\_phi\_min[ v ]) ÷

(projection\_plane\_width\_minus1 + 1)

Theta( m, v ) = ci\_erp\_theta\_max[ v ] - (m + 0.5) \* (ci\_erp\_theta\_max[ v ] – ci\_erp\_theta\_min[ v ]) ÷

(projection\_plane\_height\_minus1 + 1)

* + - 1. Perspective unprojection process

Inputs to this process are:

* a depth r at column m and row n in view v
* view\_id v

Output of this process is a point in 3D scene UnprojectPSP( m, n, v, r ) with m the image column, n the image row, v the view\_id, and r a floating-point depth value in units of meters. UnprojectPSP( m, n, v, r ) is a tuple of floating-point values (x, y, z) in the global coordinate system and is derived as follows:

UnprojectPSP( m, n, v, r ) = (  
 r,  
 -(r / ci\_perspective\_focal\_hor[ v ]) \* (m + 0.5 – ci\_perspective\_center\_hor[ v ]),  
 -(r / ci\_perspective\_focal\_ver[ v ]) \* (n + 0.5 – ci\_perspective\_center\_ver[ v ])) )

* + - 1. Orthographic unprojection process

Inputs to this process are:

* a depth r at column m and row n in view v
* view\_id v

Output of this process is a point in 3D scene UnprojectORT( m, n, v, r ) with m the image column, n the image row, v the view\_id, and r a floating-point depth value in units of meters. UnprojectORT( m, n, v, r ) is a tuple of floating-point values (x, y, z) in the global coordinate system and is derived as follows:

UnprojectORT( m, n, v, r ) = (  
 r,  
 - ci\_ortho\_width[ v ] /2 + (m + 0.5) \* ci\_ortho\_width[ v ] ÷ (projection\_plane\_width\_minus1 + 1)  
 - ci\_ortho\_height[ v ]/2 + (m + 0.5) \* ci\_ortho\_height[ v ] ÷ (projection\_plane\_height\_minus1 + 1)

* + 1. Pixel weighting recovery process

This process enables to apply a view weighting strategy in the renderer as if all reconstructed views would be plain fully unpruned views. Each pixel contribution in the visibility and shading pass is weighted by the contribution of its associated view. The process of recovering pixel weighting from un-pruned views is the following when pruning graph metadata are present in the bitstream (mvp\_pruning\_graph\_params\_present\_flag equal to 1).

Inputs to this process are:

* coordinates of a non-pruned pixel p = (y, x) in reconstructed view v
* the weights of input views with respect to target viewport: viewWeight[ v ]
* masp\_auxiliary\_atlas\_flag[ a ] of the atlas which pixel p belongs to

Output to this process is:

* the weight of contribution of pixel p = (y, x) in view v to the target viewport

The pixel is discarded if masp\_auxiliary\_atlas\_flag[ a ] is equal to 1.

If masp\_auxiliary\_atlas\_flag[a] is equal to 0, the weight of the contribution of pixel p of v-th view to the target viewport is derived as follows:

pixelWeight[ v ][ p ] = viewWeight[ v ] + computeChildrenWeight (v, Unproject(v, p))

computeChildrenWeight (v, P) {

{

w = 0

if ( pc\_is\_leaf\_flag[ v ] == 0 ) {

for( i = 0; i <= pc\_num\_children\_minus1[ v ] ; i++) {

vChild = pc\_child\_id[ v ][ i ]

pOnChild = Project (vChild, P)

if( IsInViewport(vChild, pOnChild) == 1) {

if ( IsOccupied( vChild, pOnChild) == 0 ) {

w += viewWeight[ vChild ] + computeChildrenWeight(vOnChild, P)

}

}

else {

w += computeChildrenWeight (vOnChild, P)

}

}

}

return w

}

UnProject(v, p) returns the 3D point P which projects onto pixel p in the v-th view.

Project(v, P) returns the pixel coordinates of the projection of 3D point P in the v-th view.

IsInViewport(v , p) returns true if pixel p is inside the viewport of the v-th view.

IsOccupied(v, p) returns false if the depth of pixel p in the v-th recovered pruned view is invalid.

Bibliography

1. Recommendation ITU-T H.222.0 (in force), *Information technology – Generic coding of moving pictures and associated audio information: Systems.*
2. ISO/IEC 13818-1(in force), *Information technology – Generic coding of moving pictures and associated audio information – Part 1: Systems.*
3. Recommendation ITU-T H.320 (in force), *Narrow-band visual telephone systems and terminal equipment*.
4. ISO/IEC 14496-10: *Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding*.
5. ISO/IEC 23008-2: *Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 2: High efficiency video coding*.
6. Registration authority for code-points in "MP4 Family" files: [https://mp4ra.org/#](https://mp4ra.org/)