A picture containing drawing, food

Description automatically generated

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

**Coding of moving pictures and audio**

**Convenorship: Japan (JISC)**

**Convenorship: UNI (Italy)**

**ISO/IEC JTC 1/SC 29/WG 11 N19482**

**Document type: Approved WG 11 document**

**Title: Committee Draft of MPEG Immersive Video**

**Status: Output document**

**Date of document : 2020-07-03**

**Source: Video**

**Expected action:**

**No. of pages: -**

**Email of active convenor: ostermann@tnt.uni-hannover.de**

**Committee URL: http://isotc.iso.org/livelink/livelink/open/jtc1sc29**

**Ed. notes:**

**CD:**

* m54492: vme\_embedded\_occupancy\_flag, geometry video absent, patches with constant depth
* m54362: integrated all parts of v(4) of the contribution except:
  + text that needs to be added in clause 8 of the CD and described in the contribution.
* m54766: integrated v2 of the contribution over the integration of m54362.
* m54274: integrated v(5) of the adopted contribution.
* issue #39: conditional signaling of mvp\_view\_enabled\_in\_atlas\_flag
* m54152: occupancy coding
* m54489: number of decoder instantiations
* m54491: packed independent regions SEI message
* PUC: Profiles under Consideration
* Editorial improvements
* issue #46: correct idx vs. id
* Added Figure 1: Overview of V3C bitstream with MIV extensions
* Figures 4 (Decoding process) and H.1 (Hypothetical reference view renderer) updated
* Rewriting of section H.6 (Reconstruction of reconstructed views process)
* Editorial improvements #2
* Integrating outcomes from 1st July 16th Ad hoc conf call on CD
* Integrating #7, #46, #56, etc.
* m54152-v2: occupancy coding
* Editorial improvements
* Editorial improvements #4: clarification of inputs and outputs, including some alignment with V3C spec
* Editorial improvements #5
* Fixes (clean H7, H8, issues #67, 68, 69, 70, 71,)

**WD5:**

* m53122: new structure of 23090-5 to reflect the clear separation between V-PCC and V3C
* m53348: decoded atlas data hash SEI
* m53042: views per atlas syntax and group rendering process removal
* entity: migration of target entities from decoding process to rendering
* m53124: VUI signalling in MIV
* m53917: solution to the access unit identification problem for MIV special atlas
* m53377: inverted pruning graph
* miscellaneous: editorial bug fixes and alignment of structure and naming with V3C (draft 86)
* m53507: the relation between 3VC sub bitstreams and proposed MIV restrictions on that.
* m52993+m52994+m5252998: AAPS and its carriage.
* m53569: handling discrepancies in how access-units are defined in 23090-12 and 23090-5.
* m53951: improvements to the specification of geometry scaling
* Editorial improvements
* Editorial v2 improvements

**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 N19482**

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 ix](#_Toc46245278)

[Introduction x](#_Toc46245279)

[1 Scope 1](#_Toc46245280)

[2 Normative reference 1](#_Toc46245281)

[3 Terms and definitions 1](#_Toc46245282)

[4 Abbreviations 3](#_Toc46245283)

[5 Conventions 4](#_Toc46245284)

[6 Bitstream format, partitioning, and scanning processes 4](#_Toc46245285)

[6.1 V3C bitstream formats 4](#_Toc46245286)

[6.2 NAL bitstream formats 4](#_Toc46245287)

[6.3 Partitioning of atlas frames into tiles 4](#_Toc46245288)

[6.4 Tile partition scanning processes 4](#_Toc46245289)

[6.5 Rendering 5](#_Toc46245290)

[6.6 Patches, atlases, block to patch map, view representations, and view representation pairs relationships 5](#_Toc46245291)

[6.7 Reference Architecture 6](#_Toc46245292)

[6.8 Sources and outputs 6](#_Toc46245293)

[7 Syntax and semantics 7](#_Toc46245294)

[7.1 Method of specifying syntax in tabular form 7](#_Toc46245295)

[7.2 Specification of syntax functions and descriptors 7](#_Toc46245296)

[7.3 Syntax in tabular form 7](#_Toc46245297)

[7.3.1 General 7](#_Toc46245298)

[7.3.2 V3C unit syntax 7](#_Toc46245299)

[7.3.3 Byte alignment syntax 8](#_Toc46245300)

[7.3.4 V3C parameter set syntax 8](#_Toc46245301)

[7.3.5 NAL unit syntax 10](#_Toc46245302)

[7.3.6 Raw byte sequence payloads, trailing bits, and byte alignment syntax 10](#_Toc46245303)

[7.3.7 Atlas tile data unit syntax 15](#_Toc46245304)

[7.3.8 Supplemental enhancement information message syntax 16](#_Toc46245305)

[7.4 Semantics 16](#_Toc46245306)

[7.4.1 General 16](#_Toc46245307)

[7.4.2 V3C unit semantics 16](#_Toc46245308)

[7.4.3 Byte alignment semantics 17](#_Toc46245309)

[7.4.4 V3C parameter set semantics 17](#_Toc46245310)

[7.4.5 NAL unit semantics 20](#_Toc46245311)

[7.4.6 Raw byte sequence payloads, trailing bits, and byte alignment semantics 21](#_Toc46245312)

[7.4.7 Atlas tile data unit semantics 29](#_Toc46245313)

[7.4.8 Supplemental enhancement information message semantics 30](#_Toc46245314)

[8 Decoding process 30](#_Toc46245315)

[8.1 General decoding process 30](#_Toc46245316)

[8.2 Atlas data decoding process 32](#_Toc46245317)

[8.3 Occupancy map video decoding process 32](#_Toc46245318)

[8.4 Geometry video decoding process 32](#_Toc46245319)

[8.5 Attribute(s) video decoding process 33](#_Toc46245320)

[8.6 Packed video decoding process 33](#_Toc46245321)

[9 Reconstruction process 33](#_Toc46245322)

[10 Parsing process 33](#_Toc46245323)

[11 Sub-bitstream extraction process 33](#_Toc46245324)

[Annex A (normative) Profiles, tiers, and levels 34](#_Toc46245325)

[A.1 Overview of profiles, tiers, and levels 34](#_Toc46245326)

[A.2 V3C profile, tier and level structure 34](#_Toc46245327)

[A.3 V3C CodecGroup profile components 34](#_Toc46245328)

[A.4 V3C toolset profile components 34](#_Toc46245329)

[A.4.1 MIV Main toolset profile component 34](#_Toc46245330)

[A.5 V3C Reconstruction profile components 35](#_Toc46245331)

[A.6 Tiers and Levels 35](#_Toc46245332)

[A.6.1 MIV-specific level limits 35](#_Toc46245333)

[A.7 Decoder instantiations 35](#_Toc46245334)

[Annex B (informative) Post-decoding conversion to nominal video formats 37](#_Toc46245335)

[B.1 General 37](#_Toc46245336)

[B.2 Nominal format conversion 37](#_Toc46245337)

[B.3 Conversion operations 37](#_Toc46245338)

[B.4 Unpacking process of a decoded packed video 37](#_Toc46245339)

[Annex C (informative) V3C sample stream format 40](#_Toc46245340)

[Annex D (normative) NAL sample stream format 41](#_Toc46245341)

[Annex E (normative) Atlas hypothetical reference decoder 42](#_Toc46245342)

[Annex F (normative) Supplemental enhancement information 43](#_Toc46245343)

[F.1 General 43](#_Toc46245344)

[F.2 SEI payload syntax 43](#_Toc46245345)

[F.2.1 Viewing space SEI payload syntax 43](#_Toc46245346)

[F.2.1.1 General 43](#_Toc46245347)

[F.2.1.2 Elementary shape 43](#_Toc46245348)

[F.2.1.3 Cuboid primitive 44](#_Toc46245349)

[F.2.1.4 Spheroid primitive 44](#_Toc46245350)

[F.2.1.5 Half space primitive 45](#_Toc46245351)

[F.2.2 Recommended viewport SEI payload syntax 45](#_Toc46245352)

[F.2.3 Viewing space handling SEI payload syntax 45](#_Toc46245353)

[F.2.4 Geometry upscaling parameters SEI payload syntax 46](#_Toc46245354)

[F.2.5 Packed independent regions SEI payload syntax 46](#_Toc46245355)

[F.3 SEI payload semantics 46](#_Toc46245356)

[F.3.1 Viewing space SEI payload semantics 46](#_Toc46245357)

[F.3.1.1 General 46](#_Toc46245358)

[F.3.1.2 Elementary shape 47](#_Toc46245359)

[F.3.1.3 Cuboid primitive 49](#_Toc46245360)

[F.3.1.4 Spheroid primitive 50](#_Toc46245361)

[F.3.1.5 Half space primitive 50](#_Toc46245362)

[F.3.2 Recommended viewport SEI payload semantics 50](#_Toc46245363)

[F.3.3 Viewing space handling SEI payload semantics 51](#_Toc46245364)

[F.3.4 Geometry upscaling parameters SEI payload semantics 53](#_Toc46245365)

[F.3.5 Packed independent regions SEI payload semantics 54](#_Toc46245366)

[Annex G (informative) Volumetric usability information 55](#_Toc46245367)

[Annex H (Informative) Hypothetical view rendering process 56](#_Toc46245368)

[H.1 General 56](#_Toc46245369)

[H.2 Entity filtering process 57](#_Toc46245370)

[H.3 Geometry video scaling process 58](#_Toc46245371)

[H.3.1 Nearest neighbour interpolation scaling process 59](#_Toc46245372)

[H.3.2 Sample neighbours enumeration process 59](#_Toc46245373)

[H.3.3 Foreground edge flag process 60](#_Toc46245374)

[H.3.4 Selective geometry erosion process 60](#_Toc46245375)

[H.3.5 Texture aligned geometry erosion process 60](#_Toc46245376)

[H.3.6 Geometry contour smoothening process 61](#_Toc46245377)

[H.4 Depth decoding process 62](#_Toc46245378)

[H.5 Depth Estimation Process 63](#_Toc46245379)

[H.6 Occupancy video reconstruction process 63](#_Toc46245380)

[H.7 Reconstruction of reconstructed views process 64](#_Toc46245381)

[H.8 Projection of pixels of reconstructed views to viewport 66](#_Toc46245382)

[H.8.1 General projection of pixels of reconstructed views to viewport process 66](#_Toc46245383)

[H.8.2 Local views coordinate to global coordinate point unprojection process 66](#_Toc46245384)

[H.8.2.1 ERP unprojection process 67](#_Toc46245385)

[H.8.2.2 Perspective unprojection process 68](#_Toc46245386)

[H.8.2.3 Orthographic unprojection process 68](#_Toc46245387)

[H.8.3 Pixel weighting recovery process 68](#_Toc46245388)

[Bibliography 70](#_Toc46245389)

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 3D 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 ISO/IEC 23090-5 [V3C].

# Normative reference

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. These references are in addition to the normative references in ISO/IEC 23090-5 clause 2. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

|  |  |
| --- | --- |
| [OMAF] | ISO/IEC 23090-2, Information technology — Coded Representation of Immersive Media — Part 2: Omnidirectional MediA Format (OMAF) 2nd Edition. |
| [V3C] | ISO/IEC 23090-5, Information technology — Coded Representation of Immersive Media — Part 5: Visual Volumetric Video-based Coding (V3C) and Video-based Point Cloud Compression (V-PCC), N19579, 4th July 2020. |

# Terms and definitions

The following definitions apply in addition to the definitions in [V3C] clause 3. These definitions are either not present in or replace definitions in [V3C] clause 3.

3D scene

visual content in the *global reference coordinate system*

associated atlas

the V3C\_AD *V3C unit* with the same value of vuh\_atlas\_id as the current V3C\_OVD, V3C\_GVD, or V3C\_AVD *V3C unit*

coded MIV sequence

a *coded V3C sequence* that is a *MIV IRAP access unit* followed by zero or more *MIV access units*

field of view

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

global coordinates axes

coordinate axes that are associated with video 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 video 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

MIV access unit

a *V3C composition unit* that is a set of all *sub-bitstream access units* that share the same decoding order count

MIV coded sub-bitstream sequence

a *coded sub-bitstream sequence* that is a *sub-bitstream IRAP access unit* followed by zero or more *V3C access units*

MIV IRAP access unit

a *V3C IRAP composition unit* that is a *MIV access unit* for which all *sub-bitstream access units* are *IRAP access units*

Projection plane

Picture result of a *planar projection* for a given component.

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,by projection by a virtual camera or constructed otherwise, onto a surface using *view parameters*

sub-bitstream access unit

a *sub-bitstream composition unit* that is a partition of a *sub-bitstream* that has a certain decoding order count

sub-bitstream IRAP access unit

a *sub-bitstream IRAP composition unit* that is a *sub-bitstream access unit* that forms an independent random-access point for the *sub-bitstream*

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

a unit quaternion representing the orientation that a user is consuming the 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 visual content; in case of image or video, characterizing the position of the *viewport*



view representation

2D sample arrays of *attribute frame* and corresponding *geometry frame* representing the projection of a *3D scene* onto a surface using *view 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*

IRAP V3C access unit

A *V3C access unit* for which all *coded atlas access units* are *IRAP coded atlas access units*, and all coded pictures in *video sub-bitstreams* are of type IRAP

# Abbreviations

The following abbreviations apply in addition to the abbreviations in [V3C] clause 4.

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

# Conventions

The specifications in [V3C] clause 5 and its subclauses apply.

# Bitstream format, partitioning, and scanning processes

## V3C bitstream formats

The specifications in [V3C] clause 6.1 apply.

An overview of the V3C bitstream structure with MIV extensions is represented in Figure 6‑1

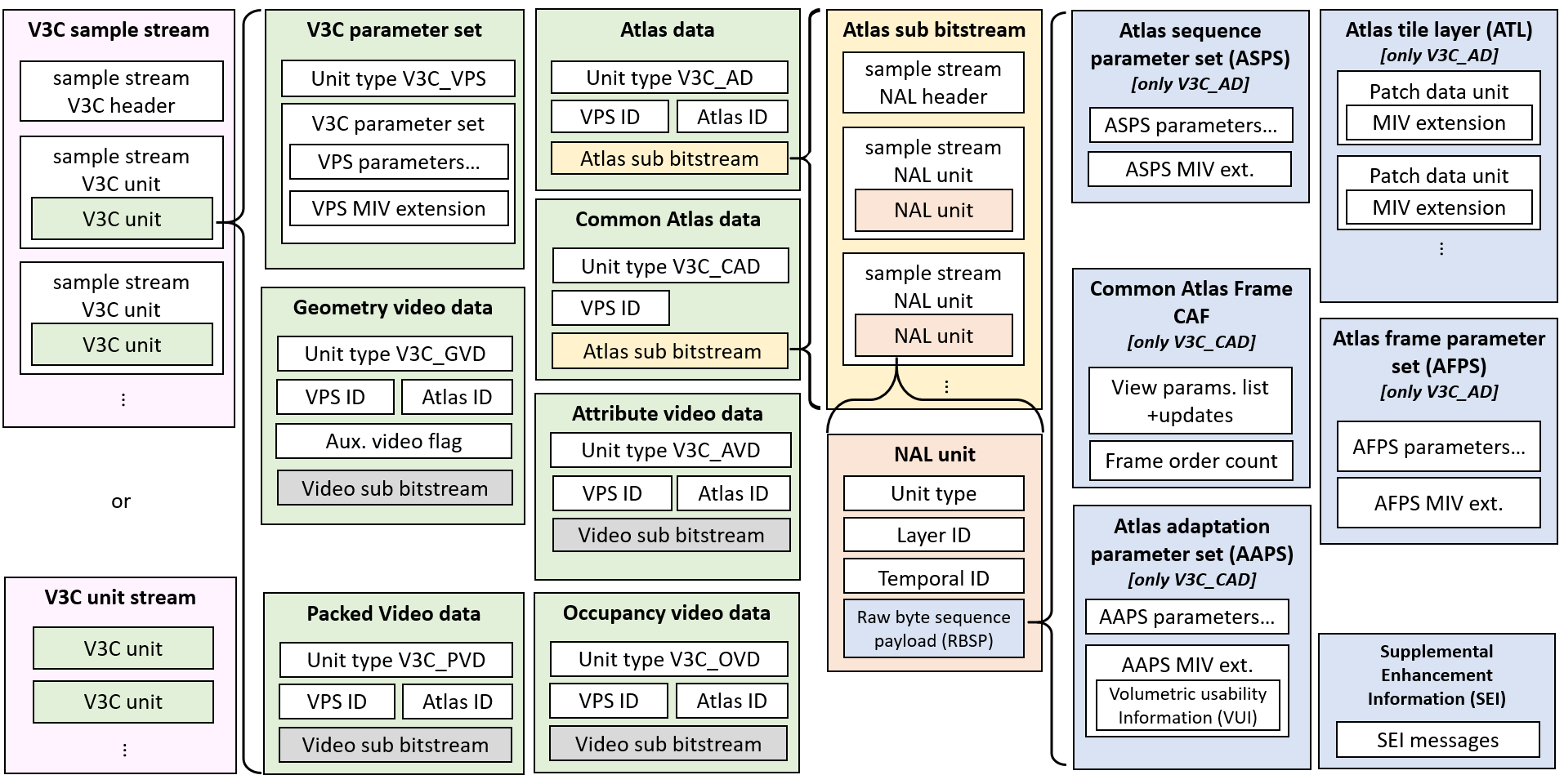


Figure 6‑1: Overview of V3C bitstream with MIV extensions

## NAL bitstream formats

The specifications in [V3C] clause 6.2 apply.

## Partitioning of atlas frames into tiles

The specifications in clause 6.3 of [V3C] and its subclauses apply.

## Tile partition scanning processes

The specifications in clause 6.4 of [V3C] apply.

## Rendering

Annex H 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 reconstruction process can reconstruct a coded MIV 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

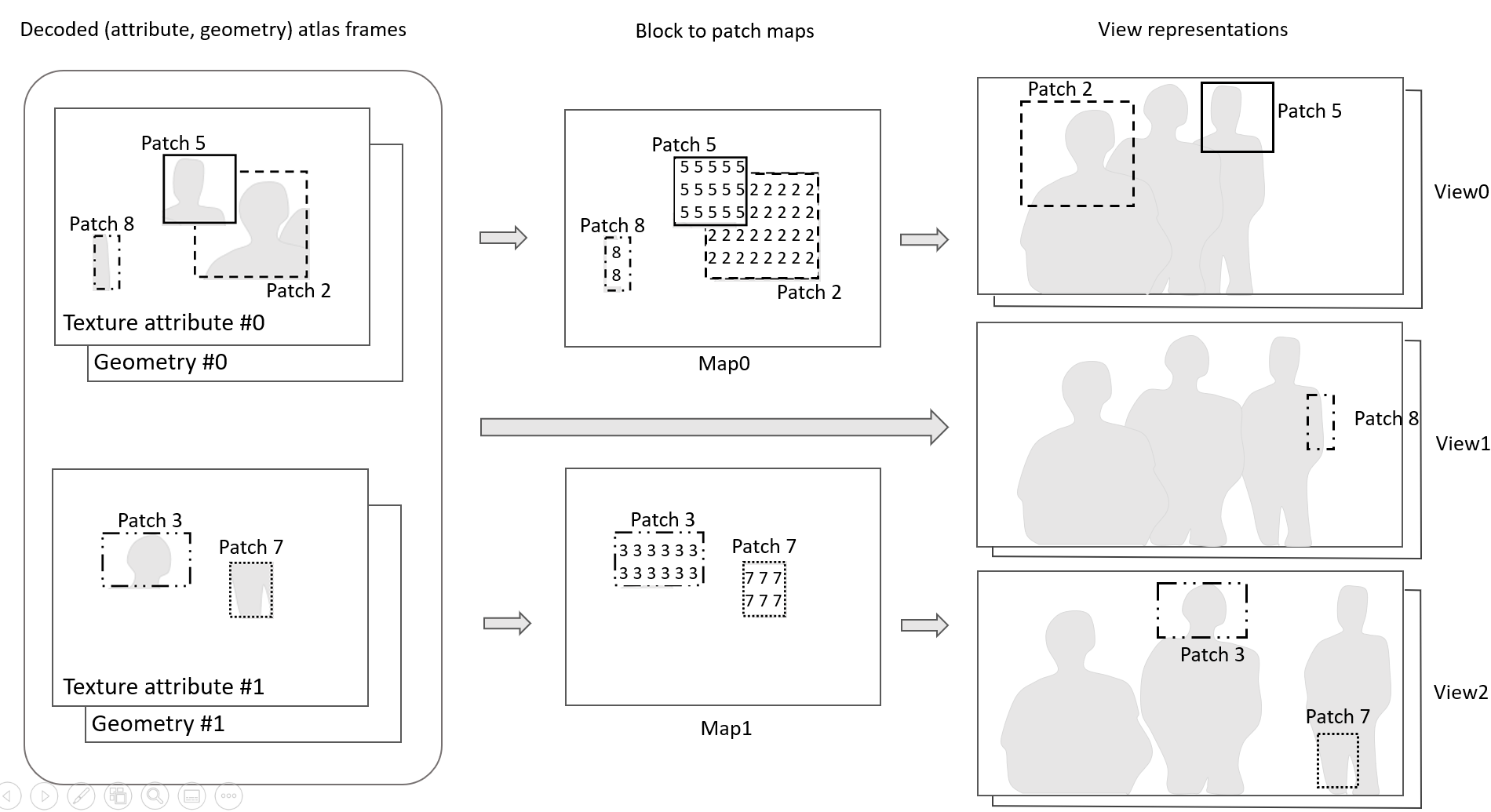


Figure 6‑2: Example mapping of 5 patches in 2 atlases to 3 view representations

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 a corresponding geometry frame and optionally a texture component. The block to patch map generator process specified in clause 8.2 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 -1. Map values equal to -1 indicates unused part of the atlas.

A view representation represents a field of view of a 3D scene for particular view parameters, for the texture attribute and geometry frames. 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 [V3C]. In this version of the specification, the texture attribute and geometry frames of a view representation use the same projection format.

Figure 6‑2. shows an illustrative example, in which two atlases contain 5 patches, which are mapped to 3 view representations.

## Reference Architecture

[(Editors): this clause needs updated text and a figure. Currently Figure 8-1 describes the conformance points. ]

The reference architecture is illustrated in ~~Figure 6-3~~.

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

The metadata is input to a metadata parser which outputs Atlas Data which includes the information of the patch list and the view parameters list, and the general information of the V3C parameter set.

The block to patch map generator, specified in clause 8.2, takes as inputs the Atlas Data, which includes the information of the patch list, and the general information of V3C parameter set and outputs an block to patch map .

In the reference architecture, a view renderer takes as inputs one or more pairs of decoded geometry frame atlases – possibly upscaled – and texture attribute frame atlases, the Atlas Data and Common Atlas Data, the block to patch map sequence, 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 H.

## Sources and outputs

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

The outputs are a view parameters list, and for each of one or more atlases a geometry frame and zero or more attribute frames, a sequence of parameter sets, 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 [V3C] clause 7.1 apply.

## Specification of syntax functions and descriptors

The specifications in [V3C] clause 7.2 apply.

## Syntax in tabular form

### General

The specifications in [V3C] clause 7.3.1 apply.

### V3C unit syntax

#### General V3C unit syntax

The specifications in [V3C] clause 7.3.2.1 apply.

#### V3C unit header syntax

[Ed (MV): This clause should be moved into the working draft of V3C version-2 when started.]

|  |  |
| --- | --- |
| v3c\_unit\_header( ) { | **Descriptor** |
| **vuh\_unit\_type** | u(5) |
| if( vuh\_unit\_type  ==  V3C\_AVD  | |  vuh\_unit\_type  ==  V3C\_GVD  | |  vuh\_unit\_type  ==  V3C\_OVD  | |  vuh\_unit\_type  ==  V3C\_AD | |   vuh\_unit\_type  ==  V3C\_CAD | |  vuh\_unit\_type  ==  V3C\_PVD ) |  |
| **vuh\_v3c\_parameter\_set\_id** | u(4) |
| if( vuh\_unit\_type  ==  V3C\_AVD  | |  vuh\_unit\_type  ==  V3C\_GVD  | |  vuh\_unit\_type  ==  V3C\_OVD  | |  vuh\_unit\_type  ==  V3C\_AD  | |  vuh\_unit\_type  ==  V3C\_PVD ) |  |
| **vuh\_atlas\_id** | u(6) |
| if( vuh\_unit\_type  = =  V3C\_AVD ) { |  |
| **vuh\_attribute\_index** | u(7) |
| **vuh\_attribute\_partition\_index** | u(5) |
| **vuh\_map\_index** | u(4) |
| **vuh\_auxiliary\_video\_flag** | u(1) |
| } else if( vuh\_unit\_type  ==  V3C\_GVD ) { |  |
| **vuh\_map\_index** | u(4) |
| **vuh\_auxiliary\_video\_flag** | u(1) |
| **vuh\_reserved\_zero\_12bits** | u(12) |
| } else if( vuh\_unit\_type  ==  V3C\_OVD  | |  vuh\_unit\_type  ==  V3C\_AD  | |    vuh\_unit\_type  ==  V3C\_PVD ) |  |
| **vuh\_reserved\_zero\_17bits** | u(17) |
| else if( vuh\_unit\_type  ==  V3C\_CAD ) |  |
| **vuh\_reserved\_zero\_23bits** | u(23) |
| else |  |
| **vuh\_reserved\_zero\_27bits** | u(27) |
| } |  |

#### V3C unit payload syntax

|  |  |
| --- | --- |
| v3c\_unit\_payload( numBytesInV3Cpayload ) { | **Descriptor** |
| if( vuh\_unit\_type  ==  V3C\_VPS ) |  |
| v3c\_parameter\_set( numBytesInV3Cpayload ) |  |
| else if( vuh\_unit\_type  ==  V3C\_AD || vuh\_unit\_type  = =  V3C\_CAD ) |  |
| atlas\_sub\_bitstream( numBytesInV3Cpayload ) |  |
| else if( vuh\_unit\_type  ==  V3C\_OVD  | |  vuh\_unit\_type  ==  V3C\_GVD  | |  vuh\_unit\_type  ==  V3C\_PVD  | |  vuh\_unit\_type  ==  V3C\_AVD) |  |
| video\_sub\_bitstream( numBytesInV3CPayload ) |  |
| } |  |

### Byte alignment syntax

The specifications in [V3C] clause 7.3.3 apply.

### V3C parameter set syntax

#### General V3C parameter set syntax

The specifications in [V3C] clause 7.3.4.1 apply.

#### Profile, tier, and level syntax

|  |  |
| --- | --- |
| profile\_tier\_level( ) { | **Descriptor** |
| **ptl\_tier\_flag** | u(1) |
| **ptl\_profile\_codec\_group\_idc** | u(7) |
| **ptl\_profile\_toolset\_idc** | u(8) |
| **ptl\_profile\_reconstruction\_idc** | u(8) |
| **ptl\_reserved\_zero\_16bits** | u(16) |
| **ptl\_max\_decodes\_idc** | u(4) |
| **ptl\_reserved\_0xfff\_12bits** | u(12) |
| **ptl\_level\_idc** | u(8) |
| **ptl\_num\_sub\_profiles** | u(6) |
| **ptl\_extended\_sub\_profile\_flag** | u(1) |
| for( i = 0; i  < ptl\_num\_sub\_profiles; i++ ) |  |
| **ptl\_sub\_profile\_idc**[ i ] | u(v) |
| **ptl\_tool\_constraints\_present\_flag** | u(1) |
| if( ptl\_toolset\_constraints\_present\_flag ) |  |
| profile\_toolset\_constraints\_information( ) |  |
| } |  |

#### Occupancy information syntax

The specifications in [V3C] clause 7.3.4.3 apply.

#### Geometry information syntax

The specifications in [V3C] clause 7.3.4.4 apply.

#### Attribute information syntax

The specifications in [V3C] clause 7.3.4.5 apply.

#### Profile toolset constraints information syntax

The specifications in [V3C] clause 7.3.4.6 apply.

#### V3C parameter set MIV extension syntax

|  |  |
| --- | --- |
| vps\_miv\_extension( ) { | **Descriptor** |
| **vme\_depth\_low\_quality\_flag** | u(1) |
| **vme\_geometry\_scale\_enabled\_flag** | u(1) |
| **vme\_num\_groups\_minus1** | ue(v) |
| **vme\_max\_entities\_minus1** | ue(v) |
| **vme\_embedded\_occupancy\_flag** | u(1) |
| if( !vme\_embedded\_occupancy\_flag ) |  |
| **vme\_occupancy\_scale\_enabled\_flag** | u(1) |
| for( k = 0 ;  k <= vps\_atlas\_count\_minus1; k++ ) { |  |
| **vme\_packed\_video\_present\_flag**[ k ] | u(1) |
| if ( vme\_packed\_video\_present\_flag[ k ] ) |  |
| packing\_information( vps\_atlas\_id[ k ] ) |  |
| } |  |
| } |  |

#### Packing information syntax syntax

|  |  |
| --- | --- |
| packing\_information( j ) { | **Descriptor** |
| **pi\_codec\_id**[ j ] | u(8) |
| **pi\_regions\_count\_minus1**[ j ] | ue(v) |
| for( i = 0;  i <= pi\_regions\_count\_minus1; i++ ) { |  |
| **pi\_region\_tile\_id**[ j ][ i ] | u(8) |
| **pi\_region\_type\_id\_minus2**[ j ][ i ] | u(2) |
| **pi\_region\_top\_left\_x**[ j ][ i ] | u(16) |
| **pi\_region\_top\_left\_y**[ j ][ i ] | u(16) |
| **pi\_region\_width\_minus1**[ j ][ i ] | u(16) |
| **pi\_region\_height\_minus1**[ j ][ i ] | u(16) |
| **pi\_region\_map\_index**[ j ][ i ] | u(4) |
| **pi\_region\_rotation\_flag**[ j ][ i ] | u(1) |
| if( pi\_region\_type\_id\_minus2[ j ][ i ] + 2  ==  V3C\_AVD ||  pi\_region\_type\_id\_minus2[ j ][ i ] + 2  ==  V3C\_GVD ) |  |
| **pi\_region\_auxiliary\_data\_flag**[ j ][ i ] | u(1) |
| if( pi\_region\_type\_id\_minus2[ j ][ i ] + 2  ==  V3C\_AVD ) { |  |
| **pi\_region\_attr\_type\_id**[ j ][ i ] | u(4) |
| **pi\_region\_attr\_partitions\_flag**[ j ][ i ] |  |
| if( pi\_region\_attr\_partitions\_flag[ j ][ i ] ) |  |
| **pi\_region\_attr\_partition\_index**[ j ][ i ] | u(5) |
| if( pi\_region\_attr\_partition\_index[ j ][ i ]  ==   0 ) |  |
| **pi\_region\_attr\_partitions\_minus1**[ j ][ i ] | u(6) |
| } |  |
| } |  |
| } |  |

### NAL unit syntax

The specifications in [V3C] clause 7.3.5 and its subclauses apply.

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

#### Atlas sequence parameter set RBSP syntax

##### General atlas sequence parameter set RBSP syntax

The specifications in [V3C] clause 7.3.6.1.1 apply.

##### Point local reconstruction information syntax

The specifications in [V3C] clause 7.3.6.1.2 do not apply.

##### Atlas sequence parameters MIV extension syntax

|  |  |
| --- | --- |
| asps\_miv\_extension( ) { | **Descriptor** |
| **asme\_group\_idx** | u(v) |
| **asme\_auxiliary\_atlas\_flag** | u(1) |
| if( vme\_embedded\_occupancy\_flag ) |  |
| **asme\_depth\_occ\_threshold\_flag** | u(1) |
| if( vme\_geometry\_scale\_enabled\_flag ) { |  |
| **asme\_geometry\_scale\_factor\_x\_minus1** | ue(v) |
| **asme\_geometry\_scale\_factor\_y\_minus1** | ue(v) |
| } |  |
| if( !vme\_embedded\_occupancy\_flag  &&  vme\_occupancy\_scale\_enabled\_flag ){ |  |
| **asme\_occupancy\_scale\_factor\_x\_minus1** | ue(v) |
| **asme\_occupancy\_scale\_factor\_y\_minus1** | ue(v) |
| } |  |
| **asme\_patch\_constant\_depth\_flag** | u(1) |
| } |  |

#### Atlas frame parameter set RBSP syntax

##### General atlas frame parameter set RBSP syntax

The specifications in [V3C] clause 7.3.6.2.1 apply.

##### Atlas frame tile information syntax

The specifications in [V3C] clause 7.3.6.2.2 apply.

#### Atlas adaptation parameter set RBSP syntax

##### General atlas adaptation parameter set RBSP syntax

The specifications in [V3C] clause 7.3.6.3.1 apply.

##### Atlas adaptation parameter MIV extension syntax

|  |  |
| --- | --- |
| aaps\_miv\_extension( ) { | **Descriptor** |
| **aame\_omaf\_v1\_compatible\_flag** | u(1) |
| **aame\_vui\_params\_present\_flag** | u(1) |
| if( aame\_vui\_params\_present\_flag ) |  |
| vui\_parameters( ) |  |
| } |  |

#### Supplemental enhancement information RBSP syntax

The specifications in [V3C] clause 7.3.6.4 apply.

#### Access unit delimiter RBSP syntax

The specifications in [V3C] clause 7.3.6.5 apply.

#### End of sequence RBSP syntax

The specifications in [V3C] clause 7.3.6.6 apply.

#### End of bitstream RBSP syntax

The specifications in [V3C] clause 7.3.6.7 apply.

#### Filler data RBSP syntax

The specifications in [V3C] clause 7.3.6.8 apply.

#### Atlas tile layer RBSP syntax

The specifications in [V3C] clause 7.3.6.9 apply.

#### RBSP trailing bit syntax

The specifications in [V3C] clause 7.3.6.10 apply.

#### Atlas tile header syntax

The specifications in [V3C] clause 7.3.6.11 apply.

#### Reference list structure syntax

The specifications in [V3C] clause 7.3.6.12 apply.

#### Common atlas frame RBSP syntax

##### General common atlas frame RBSP syntax

|  |  |
| --- | --- |
| common\_atlas\_frame\_rbsp( ) { | **Descriptor** |
| **caf\_atlas\_adaptation\_parameter\_set\_id** | ue(v) |
| **caf\_frm\_order\_cnt\_lsb** | u(v) |
| **caf\_miv\_view\_params\_list\_update\_mode** | u(3) |
| if( caf\_miv\_view\_params\_list\_update\_mode  ==  VPL\_INITLIST ) |  |
| miv\_view\_params\_list( ) |  |
| else if( caf\_miv\_view\_params\_list\_update\_mode  ==  VPL\_UPD\_EXT ) |  |
| miv\_view\_params\_update\_extrinsics( ) |  |
| else if( caf\_miv\_view\_params\_list\_update\_mode  ==  VPL\_UPD\_INT ) |  |
| miv\_view\_params\_update\_intrinsics( ) |  |
| else if( caf\_miv\_view\_params\_list\_update\_mode  ==  VPL\_UPD\_DQ ) |  |
| miv\_view\_params\_update\_depth\_quantization( ) |  |
| else if( caf\_miv\_view\_params\_list\_update\_mode  ==  VPL\_ALL ) { |  |
| miv\_view\_params\_update\_extrinsics( ) |  |
| miv\_view\_params\_update\_intrinsics( ) |  |
| miv\_view\_params\_update\_depth\_quantization( ) |  |
| } |  |
| **caf\_extension\_flag** | u(1) |
| if( caf\_extension\_flag ) |  |
| **caf\_extension\_8bits** | u(6) |
| if( caf\_extension\_8bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **caf\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

##### MIV view parameters list syntax

|  |  |
| --- | --- |
| miv\_view\_params\_list( ) { | **Descriptor** |
| **mvp\_num\_views\_minus1** | u(16) |
| **mvp\_view\_enabled\_present\_flag** | u(1) |
| if( mvp\_view\_enabled\_present\_flag ) { |  |
| for( a = 0;  a <=  vps\_atlas\_count\_minus1; a++ ) |  |
| for( v = 0; v<= mvp\_num\_views\_minus1; v++ ) { |  |
| **mvp\_view\_enabled\_in\_atlas\_flag**[ a ][ v ] | u(1) |
| if( mvp\_view\_enabled\_in\_atlas\_flag[ a ][ v ] ) |  |
| **mvp\_view\_complete\_in\_atlas\_flag** [ a ][ v ] | u(1) |
| } |  |
| } |  |
| **mvp\_explicit\_view\_id\_flag** | u(1) |
| if( mvp\_explicit\_view\_id\_flag ) |  |
| for( v = 0;  v <= mvp\_num\_views\_minus1; v++ ) |  |
| **mvp\_view\_id**[ v ] | u(16) |
| for( v = 0; v <= 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\_parents( 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 ) |  |
| } |  |
| } |  |

##### MIV view parameters update depth quantization syntax

|  |  |
| --- | --- |
| miv\_view\_params\_update\_depth\_quantization( ) { | **Descriptor** |
| **mvpudq\_num\_view\_updates\_minus1** | u(16) |
| for( i = 0; i <= mvpudq\_num\_view\_updates\_minus1; i++ ) { |  |
| **mvpudq\_view\_idx**[ i ] | u(16) |
| depth\_quantization( mvpudq\_view\_idx[ i ] ) |  |
| } |  |
| } |  |

##### 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) |
| } |  |

##### 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) |
| } |  |
| } |  |

##### 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) |
| } |  |
| if( vme\_embedded\_occupancy\_flag ) |  |
| **dq\_depth\_occ\_threshold\_default**[ v ] | ue(v) |
| } |  |

##### Pruning parents syntax

|  |  |
| --- | --- |
| pruning\_parents( v ) { | **Descriptor** |
| **pp\_is\_root\_flag**[ v ] | u(1) |
| if ( !pp\_is\_root\_flag[ v ] ) { |  |
| **pp\_num\_parents\_minus1**[ v ] | u(v) |
| for ( i = 0; i <= pp\_num\_parents\_minus1[ v ]; i++ ) |  |
| **pp\_parent\_id**[ v ][ i ] | u(v) |
| } |  |
| } |  |

### Atlas tile data unit syntax

#### General atlas tile data unit syntax

The specifications in [V3C] clause 7.3.7.1 apply.

#### Patch information data syntax

The specifications in [V3C] clause 7.3.7.2 apply.

#### Patch data unit syntax

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

#### Patch data unit MIV extension syntax

|  |  |
| --- | --- |
| pdu\_miv\_extension( tileId, p ) { | **Descriptor** |
| if( vme\_max\_entities\_minus1 > 0 ) |  |
| **pdu\_entity\_id**[ tileId ][ p ] | u(v) |
| if( asme\_depth\_occ\_threshold\_flag ) |  |
| **pdu\_depth\_occ\_threshold**[ tileId ][ p ] | u(v) |
| } |  |

### Supplemental enhancement information message syntax

The specifications in [V3C] clause 7.3.8 apply.

## Semantics

### General

The semantics in [V3C] clause 7.4.1 apply.

### V3C unit semantics

#### General V3C unit semantics

The semantics in [V3C] clause 7.4.2.1 apply.

#### V3C unit header semantics

[Ed (MV): This clause should be moved into the working draft of V3C version-2 when started.]

The semantics in [V3C] clause 7.4.2.2 apply with the following modifications.

Table 7‑1 V3C unit types

|  |  |  |  |
| --- | --- | --- | --- |
| **vuh\_unit\_type** | **Identifier** | **V3C unit type** | **Description** |
| 0 | V3C\_VPS | V3C parameter set | V3C level parameters |
| 1 | V3C\_AD | Atlas data | Atlas information |
| 2 | V3C\_OVD | Occupancy Video Data | Occupancy information |
| 3 | V3C\_GVD | Geometry Video Data | Geometry information |
| 4 | V3C\_AVD | Attribute Video Data | Attribute information |
| 5 | V3C\_PVD | Packed Video Data | Packing information |
| 6 | V3C\_CAD | Common Atlas Data | Information that is common for atlases in CVS. Specified in ISO/IEC 23090-12 |
| 7…31 | V3C\_RSV\_7 V3C\_RSV\_31 | Reserved | - |

#### V3C unit payload semantics

The semantics in [V3C] clause 7.4.2.3 apply.

#### Order of V3C units and association to coded information

A MIV bitstream consists of a series of coded MIV sequences. A MIV sequence consists of a series of V3C units. A VPS with vps\_v3c\_parameter\_set\_id equal to vuh\_v3c\_parameter\_set\_id shall be available within the MIV sequence or provided via external means before it is referenced in a V3C unit.

A MIV bitstream contains one or more sub-bitstreams. A sub-bitstream contains the V3C units with the same V3C unit header. Within a sub-bitstream, the V3C units are in decoding order. Decoding order may vary across different sub-bitstreams.

### Byte alignment semantics

The semantics in [V3C] clause 7.4.3 apply.

### V3C parameter set semantics

#### General V3C parameter set semantics

The semantics in [V3C] clause 7.4.4.1 apply.

#### Profile, tier, and level semantics

The semantics in [V3C] clause 7.4.4.2 apply with the following additions and modifications.

**ptl**\_**reserved\_zero\_16bits**, when present, shall be equal to 0 in bitstreams conforming to this version of this document. Other values for ptl\_reserved\_zero\_16bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ptl\_reserved\_zero\_16bits.

**ptl**\_**reserved\_0xfff\_12bits**, when present, shall be equal to 0xFFF in bitstreams conforming to this version of this document. Other values for ptl\_reserved\_0xfff\_12bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ptl\_reserved\_0xfff\_12bits.

**ptl\_max\_decodes\_idc** indicates on the number of sub-bitstreams requiring a video decoder instantiation to which the CVS conforms as specified in Annex A Bitstreams shall not contain values of ptl\_max\_decodes\_idc other than those specified in Annex A. Other values of ptl\_max\_decodes\_idc are reserved for future use by ISO/IEC.

#### Occupancy information semantics

The semantics in [V3C] clause 7.4.4.3 apply.

#### Geometry information semantics

The semantics in [V3C] 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[ p ] 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 [V3C] clause 7.4.4.5 apply.

#### Profile toolset constraints information semantics

The semantics in [V3C] clause 7.4.4.6 apply.

#### V3C parameter set MIV extension semantics

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

NOTE – Low depth fidelity indicates inconsistency in depth values between views.

**vme\_geometry\_scale\_enabled\_flag** equal to 1 specifies that the GVD 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 vme\_geometry\_scale\_enabled\_flag is equal to 0, it is a requirement for 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 vme\_geometry\_scale\_enabled\_flag is inferred to be equal to 0.

**vme\_num\_groups\_minus1** specifies the maximum value of asme\_group\_idx in the asps\_miv\_extension( ) syntax structure. When not present, the value of vme\_num\_groups\_minus1 is inferred to be equal to 0.

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

**vme\_embedded\_occupancy\_flag** equal to 1 specifies that occupancy information is derived from geometry as specified in clause H.5. vme\_embedded\_occupancy\_flag equal to 0 specifies that the occupancy information semantics of [V3C] clause 7.4.4.3 apply.

**vme\_occupancy\_scale\_enabled\_flag** equal to 1 specifies that an OVD sub-bitstream 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 vme\_occupancy\_scale\_enabled\_flag is equal to 0, it is a requirement of bitstream conformance that the coded picture width and height of all OVD sub-bitstreams 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 vme\_occupancy\_scale\_enabled\_flag is inferred to be equal to 0.

**vme\_packed\_video\_present\_flag**[ j ]equal to 0 indicates that the atlas with ID j does not have packed data. vps\_packed\_video\_present\_flag[ j ]equal to 1 indicates that the atlas with ID j has packed data. When vps\_packed\_video\_present\_flag[ j ]is not present, it is inferred to be equal to 0.

#### Packing information semantics

Packed video frame can be divided into one or more rectangular regions. One region shall be mapped to exactly one atlas tile. Rectangular regions of packed video frame are not allowed to overlap.

**pi\_codec\_id**[ j ] indicates the identifier of the codec used to compress the packed video data for the atlas with ID j. pi\_codec\_id shall be in the range of 0 to 255, inclusive. This codec may be identified through a component codec mapping SEI message or through means outside this document.

**pi\_regions\_count\_minus1**[ j ]plus 1 indicates the number of regions packed in one video frame for the atlas with ID j. pi\_regions\_count\_minus1 shall be in the range of 0 to 7, inclusive. When not present, the value of pi\_regions\_count\_minus1 is inferred to be equal to 0.

**pi\_region\_tile\_id**[ j ][ i ]indicates the atlas tile ID, of the atlas with ID j, corresponding to the region with index i.

**pi\_region\_type\_id\_minus2**[ j ][ i ]plus 2indicates the ID of the region with index i for the atlas with ID j. The value of pi\_region\_type\_id\_minus2[ j ][ i ]shall be in the range of 0 to 2, inclusive.

**pi\_region\_top\_left\_x**[ j ][ i ]specifies horizontal position of top left sample of the region with index i for the atlas with ID j in unit of luma samples in the packed video component frame. When not present, the value of pi\_region\_top\_left\_x[ j ][ i ] is inferred to be equal to 0.

**pi\_region\_top\_left\_y**[ j ][ i ]specifies vertical position of top left sample of the region with index i for the atlas with ID j in unit of luma samples in in the packed video component frame. When not present, the value of pi\_region\_top\_left\_y[ j ][ i ] is inferred to be equal to 0.

**pi\_region\_width\_minus1**[ j ][ i ] plus 1 specifies the width of the the region with index i for the atlas with ID j in units luma samples.

**pi\_region\_height\_minus1**[ j ][ i ] plus 1 specifies the height of the the region with index i for the atlas with ID j in units of luma samples.

**pi\_region\_map\_index**[ j ][ i ] specifies the map index of the i-th region of the atlas with ID j.

**pi\_region\_rotation\_flag**[ j ][ i ] equal to 0 indicates that no rotation on the region with index i for the atlas with ID j is performed. pi\_region\_rotation\_flag[ j ][ i ] equal to 1 indicates that the region with index i for the atlas with ID j is rotated 90 degrees.

**pi\_region\_auxiliary\_data\_flag**[ j ][ i ] equal to 1 indicates that the i-th region of the atlas with ID j contains RAW and/or EOM coded points only. pi\_region\_auxiliary\_data\_flag equal to 0 indicates that the i-th region of the atlas with ID j may contains RAW and/or EOM coded points.

**pi\_region\_attr\_type\_id**[ j ][ i ] indicates the attribute type of the i-th region of the atlas with ID j. Table 7.2 of [V3C] describes the list of supported attributes.

**pi\_region\_attr\_partitions\_flag**[ j ][ i ] equal to 1 indicates that the attribute with type pi\_region\_attr\_type\_id[ j ][ i ] contained in the i-th region of the atlas with ID j is part of an attribute with more than one partition. pi\_region\_attr\_partitions\_flag[ j ][ i ] equal to 0 indicates that the attribute with type pi\_region\_attr\_type\_id[ j ][ i ] has only one partition.

**pi\_region\_attr\_partition\_index**[ j ][ i ] indicates the attribute partition index of the i-th region of the atlas with ID j. When not present, the value of pi\_region\_attr\_partition\_index[ j ][ i ] is inferred to be equal to 0.

**pi\_attr\_partitions\_minus1** [ j ][ i ] plus 1 indicates the number of partitions for the attribute with type ID pi\_region\_attr\_type\_id[ j ][ i ] for the atlas with ID j. pi\_attr\_partitions\_minus1[ j ][ i ]shall be in the range of 0 to 63, inclusive.

### NAL unit semantics

The semantics in [V3C] clause 7.4.5 and its sub-clauses apply, except as specified below.

#### General NAL unit semantics

The specifications in [V3C] clause 7.4.5.1 apply.

#### NAL unit header semantics

[Ed (MV): This clause should be moved into the working draft of V3C version-2 when started.]

|  |  |  |  |
| --- | --- | --- | --- |
| 48 | NAL\_CAF | Common Atlas Frame  common\_atlas\_frame\_rbsp( ) | non-ACL |

#### Order of NAL units and association to coded atlas frames, access units, and coded atlas sequences

The semantics in [V3C] clause 7.4.5.3 and its sub-clauses apply, except as specified below.

##### General

The specifications in [V3C] clause 7.4.5.3.1 apply.

##### Order of AAPS, ASPS, and AFPS RBSPs and their activation

[Ed (MV): This clause should be moved into the working draft of V3C version-2 when started.]

The semantics in [V3C] clause 7.4.5.3.2 apply, except as specified below.

An AAPS RBSP, with that particular value of aaps\_atlas\_adaptation\_parameter\_set\_id, shall be available to the decoding process prior to its activation, included in at least one atlas access unit with TemporalId less than or equal to the TemporalId of the AAPS NAL unit or provided through external means, or through atlas common data, aapsAtlasId shall be equal to activatingAtlasId, and the AAPS NAL unit containing the AAPS RBSP shall have nal\_layer\_id equal to 0.

##### Order of access units and association to CASs

The specifications in [V3C] clause 7.4.5.3.3 apply.

##### Order of NAL units and coded atlas frames and their association to access units

The specifications in [V3C] clause 7.4.5.3.4 apply, except as specified below.

An access unit in a V3C\_CAD consists of one non-ACL NAL unit with nal\_unit\_type NAL\_CAF with nal\_layer\_id equal to 0, and zero or more non-ACL NAL units with nal\_layer\_id equal or greater than 0.

The first access unit in V3C\_CAD in the bitstream starts with the first NAL unit of the bitstream.

The order of non-ACL NAL units within an access unit in V3C\_CAD shall obey the following constraints:

– There shall be at most one common atlas frame NAL unit in any access unit in NAL\_CAF. If the Common atlas frame NAL unit is not the first NAL unit, then it shall be preceeded by an AAPS NAL unit.

– In V3C\_CAD, when any AAPS NAL units, prefix SEI NAL units, NAL units with nal\_unit\_type in the range of NAL\_RSV\_NACL\_49..NAL\_RSV\_NACL\_50, or NAL units with nal\_unit\_type in the range of NAL\_UNSPEC\_53..NAL\_UNSPEC\_57, nal\_unit\_type equal to NAL\_FDT, or in the range of NAL\_RSV\_NACL\_51..NAL\_RSV\_NACL\_52 or NAL\_UNSPEC\_58..NAL\_UNSPEC\_63 are present, they shall precede the first CAF NAL unit of the access unit.

– When any NAL units with nal\_unit\_type equal to NAL\_SUFFIX\_NSEI or NAL\_SUFFIX\_ESEI are are present in V3C\_CAD, they shall follow the last CAF NAL unit of the access unit.

– When an end of sequence NAL unit with nal\_layer\_id equal to 0 is present in V3C\_CAD, it shall be the last NAL unit among all NAL units with nal\_layer\_id equal to 0 in the access unit other than an end of an bitstream NAL unit (when present).

– When an end of an bitstream NAL unit is present in V3C\_CAD, it shall be the last NAL unit in the access unit.

##### Order of ACL NAL units and association to coded atlas frames

The specifications in [V3C] clause 7.4.5.3.5 apply.

### Raw byte sequence payloads, trailing bits, and byte alignment semantics

#### Atlas sequence parameter set RBSP semantics

##### General atlas sequence parameter set RBSP semantics

The semantics in [V3C] clause 7.4.6.1.1 apply, except as specified below.

**asps\_frame\_width** indicates the atlas frame width in terms of integer luma samples for the current atlas. It is a requirement of V3C 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 V3C 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\_max\_number\_projections\_minus1** indicates the maximum value of pdu\_view\_idx[ p ]. When not present, the value of asps\_max\_number\_projections\_minus1 is inferred to be equal to 5.

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

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

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

AspsMaxProjections[ vuh\_atlas\_id ]  =  asps\_max\_number\_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 vme\_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

The specifications in [V3C] clause 7.4.6.1.2 do not apply.

##### Atlas sequence parameters MIV extension semantics

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

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

**asme\_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. asme\_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 asme\_depth\_occ\_threshold\_flag is inferred to be equal to 0.

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

The variable AsmeGeometryFrameScaleFactorX[ vuh\_atlas\_id ] is set equal to asme\_geometry\_frame\_scale\_factor\_x\_minus1 + 1. The variable AsmeGeometryFrameWidth[ vuh\_atlas\_id ] is set equal to AspsFrameWidth[ vuh\_atlas\_id ] / AsmeGeometryFrameScaleFactorX[ vuh\_atlas\_id ]. It is a requirement of bitstream conformance that AspsFrameWidth[ vuh\_atlas\_id ] % AsmeGeometryFrameScaleFactorX[ vuh\_atlas\_id ] = 0.

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

The variable AsmeGeometryFrameScaleFactorY[ vuh\_atlas\_id ] is set equal to asme\_geometry\_frame\_scale\_factor\_y\_minus1 + 1. The variable AsmeGeometryFrameHeight[ vuh\_atlas\_id ] is set equal to AspsFrameHeight[ vuh\_atlas\_id ] / AsmeGeometryFrameScaleFactorY[ vuh\_atlas\_id ]. It is a requirement of bitstream conformance that AspsFrameHeight[ vuh\_atlas\_id ] % AsmeGeometryFrameScaleFactorY[ vuh\_atlas\_id ] = 0.

**asme\_occupancy\_scale\_factor\_x\_minus1** + 1 specifies a scale factor of the frame width of the occupancy video data of the atlas in relation to the nominal atlas width. When not present, the value of asme\_occupancy\_frame\_scale\_factor\_x\_minus1 is inferred to be equal to 0.

The variable AsmeOccupancyFrameScaleFactorX[ vuh\_atlas\_id ] is set equal to asme\_occupancy\_frame\_scale\_factor\_x\_minus1 + 1.

The variable AsmeOccupancyFrameWidth[ vuh\_atlas\_id ] is set equal to AspsFrameWidth[ vuh\_atlas\_id ] / AsmeOccupancyFrameScaleFactorX[ vuh\_atlas\_id ] + (AspsFrameWidth[ vuh\_atlas\_id ]/ AsmeOccupancyFrameScaleFactorX[ vuh\_atlas\_id ] ) % 2.

**asme\_occupancy\_scale\_factor\_y\_minus1** + 1 specifies a scale factor of the frame height of the occupancy video data of the atlas in relation to the nominal atlas height. When not present, the value of asme\_occupancy\_frame\_scale\_factor\_y\_minus1 is inferred to be equal to 0.

The variable AsmeOccupancyFrameScaleFactorY[ vuh\_atlas\_id ] is set equal to asme\_occupancy\_frame\_scale\_factor\_Y\_minus1 + 1.

The variable AsmeOccupancyFrameHeight[ vuh\_atlas\_id ] is set equal to AspsFrameHeight[ vuh\_atlas\_id ] / AsmeOccupancyFrameScaleFactorY[ vuh\_atlas\_id ] +  (AspsFrameHeight[ vuh\_atlas\_id ] / AsmeOccupancyFrameScaleFactorY[ vuh\_atlas\_id ] ) % 2.

**asme\_patch\_constant\_depth\_flag** equal to 1 indicates that the recommended depth for a patch be derived based on patch data unit parameters instead of the geometry video data. asme\_patch\_constant\_depth\_flag equal to 0 specifies that when vps\_geometry\_video\_present\_flag[ vuh\_atlas\_id ] is equal to 0, the depth is determined by external means.

#### Atlas frame parameter set RBSP semantics

##### General atlas frame parameter set RBSP semantics

The semantics in [V3C] clause 7.4.6.2.1 apply.

##### Atlas frame tile information semantics

The semantics in [V3C] clause 7.4.6.2.2 apply.

#### Atlas adaptation parameter set RBSP semantics

##### General atlas adaptation parameter set RBSP semantics

The semantics in [V3C] clause 7.4.6.3.1 apply with the following modification:

It is a requirement of bitstream conformance that the value of aaps\_log2\_max\_afoc\_present\_flag shall be equal to 1 for all IRAP access units.

##### Atlas adaptation parameter MIV extension semantics

**aame\_omaf\_v1\_compatible\_flag** specifies that the atlas texture frame of the atlas with ID equal to 0 is compatible for carriage within [OMAF]. When aame\_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 frame of the atlas with ID equal to 0 conforms to a projection format specified in [OMAF]. When not present the value of aame\_omaf\_v1\_compatible\_flag is inferred to be 0.

**aame\_vui\_params\_present\_flag** equal to 1 indicates that vui\_parameters( ) are present in the syntax structure. aame\_vui\_params\_present\_flag equal to 0 indicates that vui\_parameters( ) are not present in the syntax structure. It is a requirement of bitstream conformance that the value of aame\_vui\_params\_present\_flag shall be equal to 0 for all non-IRAP access units.

#### Supplemental enhancement information RBSP semantics

The semantics in [V3C] clause 7.4.6.4 apply.

#### Access unit delimiter RBSP semantics

The semantics in [V3C] clause 7.4.6.5 apply.

#### End of sequence RBSP semantics

The semantics in [V3C] clause 7.4.6.6 apply.

#### End of bitstream RBSP semantics

The semantics in [V3C] clause 7.4.6.7 apply.

#### Filler data RBSP semantics

The semantics in [V3C] clause 7.4.6.8 apply.

#### Atlas tile layer RBSP semantics

#### RBSP trailing bit semantics

The semantics in [V3C] clause 7.4.6.10 apply.

#### Atlas tile header semantics

The semantics in [V3C] clause 7.4.6.11 apply.

#### Reference list structure semantics

The semantics in [V3C] clause 7.4.6.12 apply.

#### Common atlas frame RBSP semantics

##### General common atlas frame RBSP semantics

**caf\_atlas\_adaptation\_parameter\_set\_id** identifies the atlas frame parameter set for reference by other syntax elements. The value of caf\_atlas\_adaptation\_parameter\_set\_idshall be in the range of 0 to 63, inclusive.

**caf\_frm\_order\_cnt\_lsb** specifies the atlas frame order count modulo MaxAtlasFrmOrderCntLsb for the current common atlas frame. The length of the caf\_frm\_order\_cnt\_lsb syntax element is equal to asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4 + 4 bits. The value of the caf\_frm\_order\_cnt\_lsb shall be in the range of 0 to MaxAtlasFrmOrderCntLsb − 1, inclusive.

**caf\_miv\_view\_params\_list\_update\_mode** specifies the mode in which the view parameters list is updated, as specified in Table 7‑9. When the current access unit is an IRAP access unit the value of aame\_miv\_view\_params\_list\_present\_mode shall be equal to 0. Otherwise, (the current access unit is a non-IRAP access unit), the value of aame\_miv\_view\_params\_list\_present\_mode shall be in the range of 1 .. 4. Bitstreams conforming to this version of this specification shall not contain values of caf\_miv\_view\_params\_list\_update\_ in the range of 5 .. 7.

[Ed. (JB): Make sure V3C spec clearly defines what is IRAP access units and non-IRAP access units are.]

Table 7‑2 – Updating modes for view parameters list update

|  |  |  |
| --- | --- | --- |
| **caf\_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\_UPD\_DQ | depth quantization parameters are updated for a sub-set of existing views |
| 4 | VPL\_ALL | extrinsic, intrinsic, and depth quantization parameters are updated for a sub-set of existing views |
| 5 .. 7 | Reserved | Reserved for future use by ISO/EC |

##### 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\_view\_enabled\_present\_flag** equal to 1 indicates that the signaling of enabled views and complete views is present in the syntax structure. mvp\_view\_enabled\_present\_flag equal to 0 indicates that the signaling of enabled views and complete views is not present in the syntax structure.

**mvp\_view\_enabled\_in\_atlas\_flag**[ a ][ v ] equal to 1 specifies that for any atlas\_tile\_data\_unit( ) in any V3C\_AD referring to the AAPS containing this syntax structure, for all p in 0 .. AtgduTotalNumberOfPatches − 1, inclusive, the value of pdu\_view\_idx[ p ] may be equal to v.mvp\_view\_enabled\_in\_atlas\_flag[ a ][ v ] equal to 0 specifies that the value of pdu\_view\_idx[ p ] shall not be equal to v.

**mvp\_view\_complete\_in\_atlas\_flag**[ a ][ v ] equal to 1 indicates that all values of ReconstructedDepth[ v ] output by the reconstruction of source view process for the a-th atlas specified in clause H.7 are valid values. **mvp\_view\_complete\_in\_atlas\_flag**[ a ][ v ] equal to 0 does not indicate a constraint.

**mvp\_explicit\_view\_id\_flag** equal to 1 specifies that mvp\_view\_id[ v ] is present in the syntax structure. mvp\_explicit\_view\_id\_flag equal to 0 specifies that mvp\_view\_id[ v ] is not present in the syntax structure.

**mvp\_view\_id**[ v ]indicates the view ID of the v-th signaled view parameters.

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

**mvp\_depth\_quantization\_params\_equal\_flag** equal to 1 specifies that the depth quantization parameters depth\_quantization( 0 ) of the 0-th view apply to all views in the view parameters list. mvp\_depth\_quantization\_params\_equal\_flag equal to 0 specifies that the depth quantization parameters depth\_quantization( v ) are present for each view in the view 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 view list of the i-th signaled 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 view 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 view list of the i-th signaled intrinsic parameters. The value of mvpue\_view\_idx[ i ] shall be in the range 0 to mvp\_num\_views\_minus1 inclusive.

##### MIV view parameters update depth quantization semantics

**mvpudq\_num\_view\_updates\_minus1** plus 1 indicates the number of depth quantization parameters entries in the view list present in the syntax structure. The value of mvpudq\_num\_view\_updates\_minus1shall be in the range of 0 to mvp\_num\_views\_minus1**.**

**mvpudq\_view\_idx**[ i ] specifies the index in the view list of the i-th signaled depth quantization parameters. The value of mvpudq\_view\_idx[ i ] shall be in the range 0 to mvp\_num\_views\_minus1 inclusive.

##### Camera extrinsics semantics

**ce\_view\_pos\_x**[ v ] specifies in meters the X coordinate, Tx, of the location of the camera of the v-th view 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 camera of the v-th view 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 camera of the v-th view 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 view 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 view 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 view 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 ) )

##### Camera intrinsics semantics

**ci\_cam\_type**[ v ] indicates the projection method of the v-th view. ci\_cam\_type[ v ] equal to 0 specifies ERP projection. ci\_cam\_type[ v ] equal to 1 specifies a perspective projection. ci\_cam\_type[ v ] equal to 2 specifies an orthographic projection. ci\_cam\_type values in range 3 to 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.

##### Depth quantization semantics

**dq\_quantization\_law**[ v ] indicates the type of depth quantization method of the v-th view. quantization\_law[ v ] equal to 0 specifies a uniform quantization of the inverse of depth values. Values of dq\_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 view.

**dq\_depth\_occ\_threshold\_default**[ v ] specifies the default occupancy threshold used in the occupancy value extraction process H.4. When not present the value of dq\_depth\_occ\_threshold\_default[ v ] is inferred to be 0.

##### Pruning parents semantics

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

**pp\_num\_parents\_minus1**[ v ] plus 1 specifies the number of parents of the v-th source view in the pruning graph at encoder stage. pp\_num\_parents\_minus1[ v ] shall be in the range 0 to mvp\_num\_views\_minus1 exclusive.

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

### Atlas tile data unit semantics

#### General atlas tile data unit semantics

The semantics in [V3C] clause 7.4.7.1 apply.

#### Patch information data semantics

The semantics in [V3C] clause 7.4.7.2 apply.

#### Patch data unit semantics

The semantics in [V3C] clause 7.4.7.3 apply except for the following syntax elements.

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

**pdu\_view\_pos\_x**[ tileId ][ p ] specifies the horizontal coordinate in luma samples, respectively, of the top-left corner of the patch with index equal to p in the view with index equal to pdu\_view\_idx[ tileId ][ p ], in the tile with id tileId. The value of pdu\_view\_pos\_x[ tileId ][ p ] shall be in the range of 0 to ci\_projection\_plane\_width\_minus1[ pdu\_view\_idx[ p ] ], inclusive. The number of bits used to represent pdu\_view\_pos\_x[ tileId ][ p ] is asps\_geometry\_3d\_bitdepth\_minus1  + 1.

**pdu\_view\_pos\_y**[ tileId ][ p ] specifies the vertical coordinate in luma samples, respectively, of the top-left corner of the patch with index equal to p in the view with index equal to pdu\_view\_idx[ tileId ][ p ], in the tile with id tileId. The value of pdu\_view\_pos\_y[ tileId ][ p ] shall be in the range of 0 to ci\_projection\_plane\_height\_minus1[ pdu\_view\_idx[ p ] ], inclusive. The number of bits used to represent pdu\_view\_pos\_y[ tileId ][ p ] is asps\_geometry\_3d\_bitdepth\_minus1  + 1.

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

PduDepthStart[ vuh\_atlas\_id ][ tileId ][  p ]=pdu\_depth\_start[ p ]<< ath\_pos\_min\_z\_quantizer

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

The number of bits used to represent pdu\_depth\_start[ tileId ][ p ] is equal to (asps\_geometry\_3d\_bitdepth\_minus1 – ath\_pos\_min\_z\_quantizer + 1).

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

PduDepthEnd[ vuh\_atlas\_id ][ tileId ][p ] = pdu\_depth\_end[ tileId ][p ] << ath\_pos\_delta\_max\_z\_quantizer

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

The number of bits used to represent pdu\_depth\_end[ tileId ][p ] is equal to (asps\_geometry\_3d\_bitdepth\_minus1 – ath\_pos\_min\_z\_quantizer + 1).

#### Patch data unit MIV extension semantics

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

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

### Supplemental enhancement information message semantics

The semantics in [V3C] clause 7.4.8 and its subclauses apply.

# Decoding process

## General decoding process

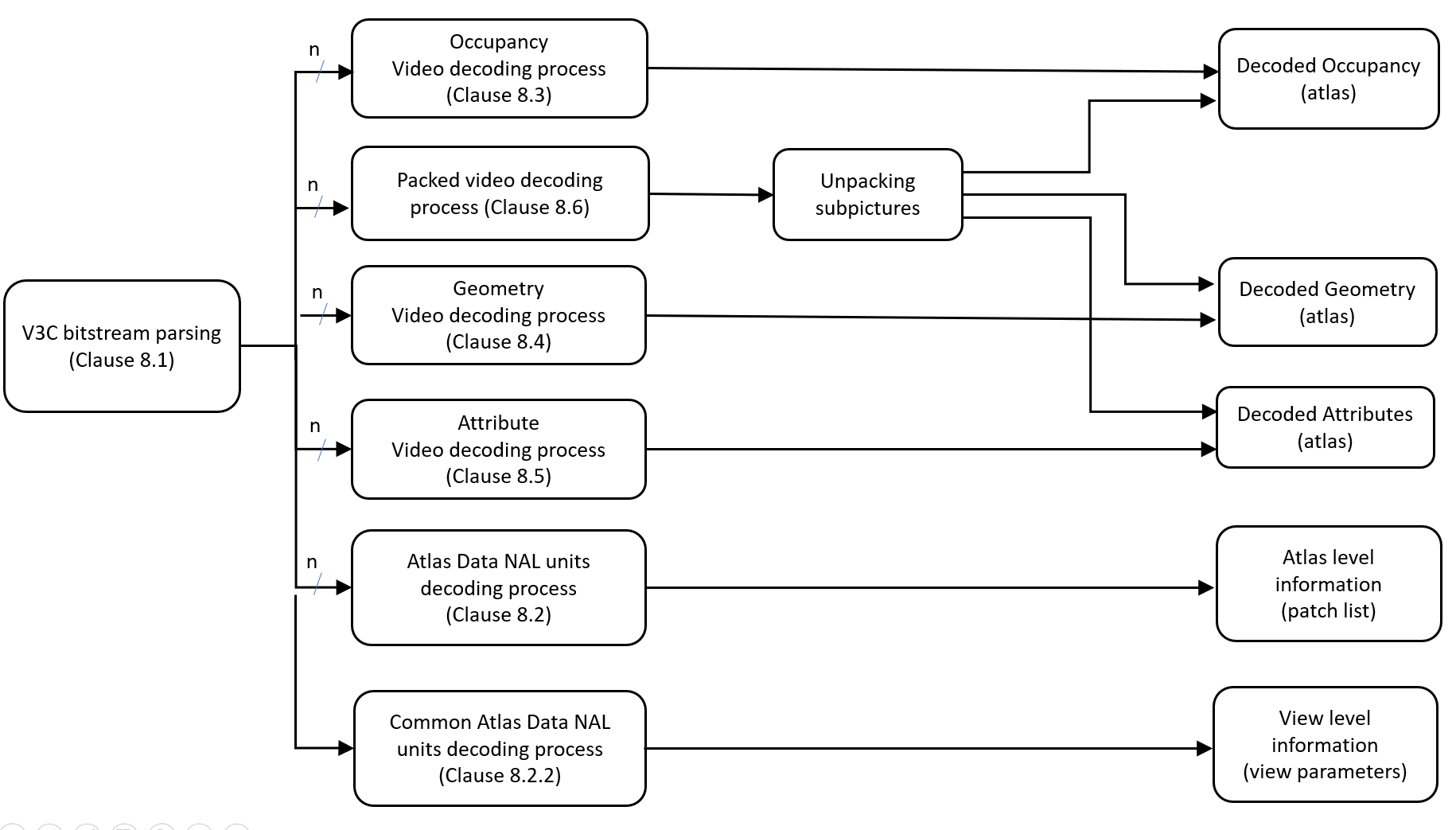


Figure 8‑1 High-level mapping of the MIV decoding processes and their interactions

The block diagram of the decoding process is illustrated in Figure 8‑1.

Input to this process are the following:

* a V3C bitstream,

Output of this process is, for each MIV access unit in a series of MIV access units with decoder output index equal to frameIdx, 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:

* zero or one decoded occupancy picture, decOccFrame[ frameIdx ][ 0 ][ ][ ], of size AsmeOccupancyFrameWidth[ vuh\_atlas\_id ] x AsmeOccupancyFrameHeight[ vuh\_atlas\_id ]
* zero or one decoded geometry picture, decGeoFrame[ 0 ][ frameIdx ][ 0 ][ ][ ], of size AsmeGeometryFrameWidth[ vuh\_atlas\_id ] x AsmeGeometryFrameHeight[ vuh\_atlas\_id ]
* zero or one texture attribute picture decAttrFrame[ 0 ][ 0 ][ 0 ][ frameIdx ][ compIdx ][  ][  ] of size AspsFrameWidth[ vuh\_atlas\_id ]  x  AspsFrameHeight[ vuh\_atlas\_id ], with compIdx in 0 .. 2. [Ed. (JB): this assumes 3 color components for the texture attribute picture. Should it be generalized?]
* one BlockToPatchMap[  ][  ] of size (AspsFrameWidth[ vuh\_atlas\_id ] / AtlasPatchPackingBlockSize[ vuh\_atlas\_id ])   x   ( AspsFrameHeight[ vuh\_atlas\_id ] / AtlasPatchPackingBlockSize[ vuh\_atlas\_id ] )
* Atlas parameters

The decoding process operates as follows:

For each V3C sequence in the bitstream, the following is repeatedly invoked:

1. If present, the occupancy sub-bitstream is extracted from the V3C sequence.
   1. The occupancy video decoding process, as specified in clause 8.3, is invoked for the occupancy sub-bitstream.
2. If present, the geometry sub-bitstream is extracted from the V3C sequence.
   1. The geometry video decoding process, as specified in clause 8.4, is invoked for the geometry sub-bitstream.
3. If present, the texture attribute sub-bitstream is extracted from the V3C sequence.
   1. The attribute video decoding process, as specified in clause 8.5, is invoked for the texture attribute sub-bitstream, if present.
4. If present, the packed sub-bitstream is extracted from the V3C sequence.
   1. The packed video decoding process, as specified in clause 8.6, is invoked for the packed sub-bitstream.
5. For each common atlas frame present in the V3C sequence, set the access unit decoding order count to caf\_frm\_order\_cnt\_lsb. Atlas tiles with ath\_atlas\_frm\_order\_cnt equal to the access unit decoding order count and sub-bitstreams coded pictures with frame order count equal to the access unit decoding order count are defined to be within the same access unit.
6. For each MIV access unit in the V3C sequence, the following is repeatedly invoked
   1. For each atlas present in the MIV access-unit, each with a unique value of vuh\_atlas\_id, the following is repeatedly invoked.
      1. The sub-bitstream extraction process in clause 11 is invoked with targetAtlasId set equal to vuh\_atlas\_id, MIV access unit as input.
      2. The decoded occupancy picture with the same value of vuh\_atlas\_id for the access unit, if present, is output, as variable decOccFrame[ frameIdx ][ 0 ][  ][  ]
      3. The decoded geometry picture with the same value of vuh\_atlas\_id for the access unit, if present, is output, as variable decGeoFrame[ 0 ][ frameIdx ][ 0 ][  ][  ].
      4. The decoded texture attribute picture with the same value of vuh\_atlas\_id for the access unit, if present, is output as variable decAttrFrame[ 0 ][ 0 ][ 0 ][ frameIdx ][ compIdx ][  ][  ], with compIdx in 0 ..2
      5. If an atlas tile layer RBSP is present in the current coded atlas frame, the atlas data decoding process, as specified in clause 8.2, 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 layer RBSP present in a coded atlas data frame of the sequence in a V3C unit with vuh\_atlas\_id equal to targetAtlasId, with output BlockToPatchMap[  ][  ].

## Atlas data decoding process

The specifications in [V3C] clause 8.2 apply.

## Occupancy map video decoding process

The specifications in [V3C] clause 8.3 apply with the following addition.

It is a requirement of bitstream conformance that decOccWidth[ frameIdx ] equal AsmeOccupancyFrameWidth[ vuh\_atlas\_id ] and decOccHeight[ frameIdx ] equal AsmeOccupancyFrameHeight[ vuh\_atlas\_id ].

## Geometry video decoding process

The specifications in [V3C] clause 8.4 apply with the following addition.

It is a requirement of bitstream conformance that decGeoWidth[ frameIdx ] equal AsmeGeometryFrameWidth[ vuh\_atlas\_id ] and decGeoHeight[ frameIdx ] equal AsmeGeometryFrameHeight[ vuh\_atlas\_id ].

## Attribute(s) video decoding process

The specifications in [V3C] clause 8.5 apply with the following addition.

It is a requirements of bitstream conformance that decAttrWidth[ 0 ][ 0 ][ 0 ][ frameIdx ] ] equal AspsFrameWidth[ vuh\_atlas\_id ] and decAttrHeight[ 0 ][ 0 ][ 0 ][ frameIdx ] equal AspsFrameHeight[ vuh\_atlas\_id ].

The variable AiAttributeDimension[ vuh\_atlas\_id ][ 0 ] is set equal to ai\_attribute\_dimension\_minus1 [ vuh\_atlas\_id ][ 0 ] + 1. [Ed. (JB): Check latest V3C spec for alignment.]

## Packed video decoding process

[Ed (MV): This clause should be moved into the working draft of V3C version-2 when started.]

The codec for the packed video component associated with the current atlas, with atlas index atlasIdx, is first determined using pi\_pack\_codec\_id[ atlasIdx ] and the codec mapping SEI message, if present. Then, the packed video decoding process, according to the corresponding codec specification, is invoked using the packed video sub-bitstreams present in the V3C bitstream as the input.

Outputs of this process are:

* the decoded packed video frames, decPackedFrame[ frameIdx ][ compIdx ][ y ][ x ], and
* for each decoded packed video frame, its corresponding coded representation parameters:
  + bit depth, decPackedBitDepth[ frameIdx ],
  + width, decPackedWidth[ frameIdx ],
  + height, decPackedHeight[ frameIdx ],
  + output order index, decPackedOutOrdIdx[ frameIdx ], and
  + composition time, decPackedCompTime[ frameIdx ],

where frameIdx is the decoder output index of the decoded packed video frames, compIdx corresponds to the colour component index and is in the range of 0 to 2, y is the row index in the decoded frame and is in the range of 0 to decPackedHeight[ frameIdx ] − 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to decPackedWidth[ frameIdx ] − 1, inclusive.

NOTE – Any existing video codec such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video codec can be used if included in pi\_packed\_codec\_id.

# Reconstruction process

The specifications in [V3C] clause 9 do not apply.

# Parsing process

The specifications in [V3C] clause 10 and its subclauses apply.

# Sub-bitstream extraction process

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

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 V3C units with vuh not equal to V3C\_CAD and vuh\_atlas\_id not equal to targetAtlasId.

1. (normative)  
   Profiles, tiers, and levels
   1. Overview of profiles, tiers, and levels

The specifications in [V3C] Annex A.1 apply.

* 1. V3C profile, tier and level structure

The specifications in [V3C] Annex A.2 apply.

* 1. V3C CodecGroup profile components

The specifications in [V3C] Annex A.3 apply.

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

* 1. V3C toolset profile components

The specifications in [V3C] Annex A.4 apply.

* + 1. MIV Main toolset profile component

V3C toolset profile components indicating the MIV Main (ptl\_profile\_toolset\_idc = 64) toolset profile component shall conform to the restrictions specified in Table A-1.

**Table A-1–Allowable values of syntax element values for MIV Main profile**

|  |  |
| --- | --- |
| **Syntax element** | MIV Main profile |
| vuh\_unit\_type | 0 (V3C\_VPS), 1 (V3C\_AD), 3 (V3C\_GVD), 4 (V3C\_AVD) |
| asps\_raw\_patch\_enabled\_flag | 0 |
| vps\_miv\_extension\_flag | 1 |
| ptl\_profile\_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\_pixel\_deinterleaving\_enabled\_flag | 0 |
| asps\_patch\_precedence\_order\_flag | 0 |
| afps\_lod\_mode\_enabled\_flag | 0 |
| afps\_raw\_3d\_pos\_bit\_count\_explicit\_mode\_flag | 0 |
| afti\_single\_tile\_in\_atlas\_frame\_flag | 1 |
| afti\_signalled\_tile\_id\_flag | 0 |
| ath\_type | I\_TILE |
| atdu\_patch\_mode | I\_INTRA |
| asps\_eom\_patch\_enabled \_flag | 0 |
| vps\_map\_count\_minus1 | 0 |
| aaps\_extension\_flag | 1 |
| aaps\_miv\_extension\_flag | 1 |
| asps\_point\_local\_reconstruction\_enabled\_flag | 0 |
| ai\_attribute\_dimension\_minus1 | 2 |
| ai\_attribute\_dimension\_partitions\_minus1 | 0 |
| vme\_embedded\_occupancy\_flag | 1 |
| vps\_occupancy\_video\_present\_flag[ j ] | 0 |
| vps\_geometry\_video\_present\_flag[  j ] | 1 |
| vps\_packing\_enabled\_flag[ j ] | 0 |
| asme\_patch\_constant\_depth\_flag | 0 |
| vps\_video\_geometry\_present\_flag[ j ] || asme\_patch\_constant\_depth\_flag | 1 |

* 1. V3C Reconstruction profile components

The specifications in [V3C] clause A.5 do not apply.

* 1. Tiers and Levels

The specifications in [V3C] clause A.6 apply with the following additions.

* + 1. MIV-specific level limits

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

* 1. Decoder instantiations

Conforming bitstreams shall obey the following constraint:

NumDecodes shall be less than or equal to MaxDecodes specified in Table A-7, for values of ptl\_max\_decodes\_idc in the range of 0 .. 9.

ptl\_max\_decodes\_idc equal to 15 does not indicate a constraint. ptl\_max\_decodes\_idc values in the range of 10..14 are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this version of this Specification.

Variable NumDecodes is derived as follows:

NumDecodes = 0  
for(j = 0; j < vps\_atlas\_count\_minus1 + 1; j++ ) {  
 NumDecodes  +=  vps\_auxiliary\_video\_present\_flag[ j ]  
 NumDecodes  +=  vps\_occupancy\_video\_present\_flag[ j ]  
 NumDecodes  +=  vps\_geometry\_video\_present\_flag[ j ] \*  (vps\_map\_count\_minus1[ j ] + 1)  
 if (vps\_attribute\_video\_present\_flag[ j ] )  
 NumDecodes  +=  ai\_attribute\_count[ j ] \* ( vps\_map\_count\_minus1[ j ] + 1 )  
 NumDecodes  −=  vme\_packed\_video\_present\_flag[ j ] ? pi\_regions\_count\_minus1[ j ] : 0  
}

**Table A-7 – Mapping of ptl\_max\_decodes\_idc to MaxDecodes**

|  |  |
| --- | --- |
| **ptl\_max\_decodes\_idc** | **MaxDecodes** |
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |
| 4 | 6 |
| 5 | 8 |
| 6 | 12 |
| 7 | 16 |
| 8 | 24 |
| 9 | 32 |
| 10..14 | Reserved |
| 15 | Unconstrained |

1. (informative)  
   Post-decoding conversion to nominal video formats
   1. General

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

* 1. Nominal format conversion

The specifications in [V3C] clause B.2 do not apply

* 1. Conversion operations

The specifications in [V3C] clause B.3 do not apply.

* 1. Unpacking process of a decoded packed video

[Ed. (MV): This clause should be moved into the working draft of V3C version-2 when started.]

[Ed. (JB): check the derivation of the scaling factors and the use of TileOffsetX & Y to determine the width and height]

For an atlas with id k and frame index frameIdx inputs to this process are:

* packed information syntax elements, as specified in clasue 7.3.4.8
* tile information TileWidth, TileHeightm TileOffsetX, TileOffsetY, as specified in clause 7.4.6.2.2
* the outputs of the packed video decoding process, as specified in clause 8.6

Outputs of this process are:

* the decoded occupancy frame decOccFrame[ frameIdx ][ compIdx ][ y ][ x ], if present, and its associated information
* bit depth, decOccBitDepth[ frameIdx ]
* width, decOccWidth[ frameIdx ],
* height, decOccHeight[ frameIdx ],
* output order index, decOccOutOrdIdx[ frameIdx ], and
* composition time, decOccCompTime[ frameIdx ]
* the decoded geometry frame decGeoFrame[ mapIdx ][ frameIdx ][ compIdx ][ y ][ x ], if present, and its associated information
* bitdepth, decGeoBitdepth[ mapIdx ][ frameIdx ],
* width, decGeoWidth[ mapIdx ][ frameIdx ],
* height, decGeoHeight[ mapIdx ][ frameIdx ],
* output order index, decGeoOutOrdIdx[ frameIdx ], and
* composition time, decGeoCompTime[ frameIdx ]
* the decoded auxiliary geometry frame decGeoAuxFrame[ frameIdx ][ compIdx ][ y ][ x ], if present, and its associated information
* bitdepth, decGeoAuxBitdepth[ frameIdx ],
* width, decGeoAuxWidth[ frameIdx ],
* height, decGeoAuxHeight[ frameIdx ],
* output order index decGeoAuxOutOrdIdx[ frameIdx ], and
* composition time, decGeoAuxCompTime[ frameIdx ]
* the decoded attribute frame decAttrFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ][ y ][ x ], if present, and its associated information
* bit depth, decAttrBitDepth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ],
* width, decAttrWidth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ],
* height, decAttrHeight[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ],
* output order index, decAttrOutOrdIdx[ frameIdx ], and
* composition time, decAttrCompTime[ frameIdx ]
* the decoded auxiliary attribute frame decAttrAuxFrame[ attrIdx ][ partIdx ][ frameIdx ][ compIdx ][ y ][ x ], if present, and its associated information
* bit depth, decAttrAuxBitDepth[ attrIdx ][ partIdx ][ frameIdx ],
* width, decAttrAuxWidth[ attrIdx ][ partIdx ][ frameIdx ],
* height, decAttrAuxHeight[ attrIdx ][ partIdx ][ frameIdx ],
* output order index, decAttrAuxOutOrdIdx[ frameIdx ], and
* composition time, decAttrAuxCompTime[ frameIdx ],

To unpacked the decoded packed video frame the following process apply.

decOccWidth[ frameIdx ] = 0   
decOccHeight[ frameIdx ] = 0  
decGeoWidth[ mapIdx ][ frameIdx ] = 0  
decGeoHeight[ mapIdx ][ frameIdx ] = 0  
decGeoAuxWidth[ frameIdx ] = 0  
decGeoAuxHeight[ frameIdx ] = 0  
decAttrWidth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = 0  
decAttrHeight[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = 0  
decAttrAuxWidth[ frameIdx ] = 0  
decAttrAuxHeight[ frameIdx ] = 0

for(i = 0; i <= pi\_num\_region\_minus1[ i ]; i++) {  
 mapIdx = pi\_region\_map\_index[ k ][ i ]  
 attrIdx = pi\_region\_attr\_type\_id[ k ][ i ]  
 partIdx = pi\_region\_attr\_partition\_index[ k ][ i ]  
 auxDataFlag = pi\_region\_auxiliary\_data\_flag[ k ][ i ]  
 regionOffsetX = pi\_region\_top\_left\_x[ k ][ i ]  
 regionOffsetY = pi\_region\_top\_left\_y[ k ][ i ]  
 regionWidth = pi\_region\_width\_minus1[ k ][ i ] + 1  
 regionHeight = pi\_region\_height\_minus1[ k ][ i ] + 1  
 tileID = pi\_region\_tile\_id[ k ][ i ]  
 tileWidth = TileWidth[ TileIdToIndex[ tileID ] ]  
 tileHeight = TileHeight[ TileIdToIndex[ tileID ] ]  
   
 if( pi\_region\_rotation\_flag[ k ][ i] == 0 ) {  
 scaleX = tileWidth/regionWidth   
 scaleY = tileHeight/regionHeight   
 } else {  
 scaleX = tileWidth/regionHeight   
 scaleY = tileHeight/regionWidth   
 }  
   
 offsetX = TileOffsetX[ TileIdToIndex[ tileID ] ]/scaleX   
 offsetY = TileOffsetY[ TileIdToIndex[ tileID ] ]/scaleY   
   
 if( pi\_region\_rotation\_flag[ k ][ i] == 0 ) {  
 tempWidth = offsetX + regionWidth   
 tempHeight = offsetY + regionHeight  
 } else {  
 tempWidth = offsetX + regionHeight  
 tempHeight = offsetY + regionWidth  
 }  
   
 if(pi\_region\_type\_id\_minus2[ k ][ i ] + 2 == V3C\_OVD) {  
 if(tempWidth > decOccWidth[ frameIdx ])  
 decOccWidth[ frameIdx ] = tempWidth   
 if(tempHeight > decOccHeight[ frameIdx ])  
 decOccHeight[ frameIdx ] = tempHeight   
 }else if (pi\_region\_type\_id\_minus2[ k ][ i ] +2 == V3C\_GVD && auxDataFlag == 0 ){  
 if(tempWidth > decGeoWidth[ mapIdx ][ frameIdx ])  
 decGeoWidth[ mapIdx ][ frameIdx ] = tempWidth   
 if(tempHeight > decGeoHeight[ mapIdx ][ frameIdx ])  
 decGeoHeight[ mapIdx ][ frameIdx ] = tempHeight   
 }else if (pi\_region\_type\_id\_minus2[ k ][ i ] +2 == V3C\_GVD && auxDataFlag == 1 ){  
 if(tempWidth > decGeoAuxWidth[ frameIdx ])  
 decGeoAuxWidth[ frameIdx ] = tempWidth   
 if(tempHeight > decGeoAuxHeight[ frameIdx ])  
 decGeoAuxHeight[ frameIdx ] = tempHeight   
 }else if (pi\_region\_type\_id\_minus2[ k ][ i ] +2 == V3C\_AVD && auxDataFlag == 0 ){  
 if(tempWidth > decAttrWidth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ])  
 decAttrWidth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = tempWidth   
 if(tempHeight > decAttrHeight[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] )  
 decAttrHeight[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = tempHeight   
 }else if (pi\_region\_type\_id\_minus2[ k ][ i ] +2 == V3C\_AVD && auxDataFlag == 1 ){  
 if(tempWidth > decAttrAuxWidth[ frameIdx ])  
 decAttrAuxWidth[ frameIdx ] = tempWidth   
 if(tempHeight > decAttrAuxHeight[ frameIdx ])  
 decAttrAuxHeight[ frameIdx ] = tempHeight   
 }  
  
 for (y = 0; y < regionHeight; y++) {  
 for(x = 0; x < regionWidth ; x++) {  
 for(compIdx = 0; compIdx < 3 ; compIdx ++) {  
 sample = decPackedFrame[ frameIdx ][ compIdx ][ regionOffsetY + y ][ regionOffsetX + x ]   
  
 if(pi\_region\_type\_id\_minus2[ k ][ i ] + 2 == V3C\_OVD && compIdx == 0 ) {  
 if(pi\_region\_rotation\_flag[ k ][ i] == 0) {  
 decOccFrame[ mapIdx ][ frameIdx ][ 0 ][ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decOccFrame[ mapIdx ][ frameIdx ][ 0 ][ offsetY + x ][ offsetX + y ] = sample  
 }  
  
 } else if (pi\_region\_type\_id\_minus2[ k ][ i ] +2 == V3C\_GVD &&   
 compIdx == 0 && auxDataFlag == 0 ){  
 if(pi\_region\_rotation\_flag[ k ][ i] == 0) {  
 decGeoFrame [ atlasIdx ][ mapIdx ][ frameIdx ][ 0 ][ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decGeoFrame[ atlasIdx ][ mapIdx ][ frameIdx ][ 0 ][ offsetY + x ][ offsetX + y ] = sample  
 }  
  
 } else if (pi\_region\_type\_id\_minus2[ k ][ i ] +2 == V3C\_GVD &&   
 compIdx == 0  && auxDataFlag == 1 ){  
 if(pi\_region\_rotation\_flag[ k ][ i] == 0) {  
 decGeoAuxFrame[ atlasIdx ][ mapIdx ][ frameIdx ][ 0 ][ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decGeoAuxFrame[ atlasIdx ][ mapIdx ][ frameIdx ][ 0 ][ offsetY + x ][ offsetX + y ] = sample  
 }  
  
 } else if (pi\_region\_type\_id\_minus2[ k ][ i ] + 2 == V3C\_AVD && auxDataFlag == 0 ) {  
 attrIdx = pi\_region\_attr\_type\_id[ k ][ i ]  
 decAttrType[ attrIdx ] = pi\_region\_attr\_type\_id[ k ][ i ]  
 partIdx = pi\_region\_attr\_partition\_index[ k ][ i ]  
 if(pi\_region\_rotation\_flag[ k ][ i] == 0) {  
 decAttrFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ][ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decAttrFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ][ offsetY + x ][ offsetX + y ] = sample  
 }  
 }  
  
 } else if (pi\_region\_type\_id\_minus2[ k ][ i ] + 2 == V3C\_AVD && auxDataFlag == 0 ) {  
 attrIdx = pi\_region\_attr\_type\_id[ k ][ i ]  
 decAttrAuxType[ attrIdx ] = pi\_region\_attr\_type\_id[ k ][ i ]  
 partIdx = pi\_region\_attr\_partition\_index[ k ][ i ]  
 if(pi\_region\_rotation\_flag[ k ][ i] == 0) {  
 decAttrAuxFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ][ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decAttrAuxFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ][ offsetY + x ][ offsetX + y ] = sample  
 }  
 }  
 }  
 }  
}

decOccBitdepth[ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decOccCompTime[ frameIdx ] = decPackedCompTime[ frameIdx ]   
decOccOutOrdIdx[ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]   
  
decGeoBitdepth[ mapIdx ][ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decGeoCompTime[ mapIdx ][ [ frameIdx ] = decPackedCompTime[ frameIdx ]   
decGeoOutOrdIdx[ mapIdx ][ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]   
  
decGeoAuxBitdepth[ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decGeoAuxCompTime[ frameIdx ] = decPackedCompTime[ frameIdx ]   
decGeoAuxOutOrdIdx[ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]   
  
decAttrBitdepth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decAttrCompTime[ frameIdx ] = decPackedCompTime[ frameIdx ]   
decAttrOutOrdIdx[ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]   
  
decAttrAuxBitdepth[ attrIdx ][ partIdx ][ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decAttrAuxCompTime[ frameIdx ] = decPackedCompTime[ frameIdx ]   
decAttrOutOrdIdx[ frameIdx ] = decPackedOutOrdIdx[ frameIdx ] 

1. (informative)  
   V3C sample stream format

The specifications in [V3C] Annex C apply.

1. (normative)  
   NAL sample stream format

The specifications in [V3C] Annex D apply.

1. (normative)  
   Atlas hypothetical reference decoder

The specifications in [V3C] Annex E apply.

1. (normative)  
   Supplemental enhancement information
   1. General

The specifications in clause F.1 of [V3C] Annex F apply.

* 1. SEI payload syntax

The specifications in clause F.2 of [V3C] Annex F and its subclauses apply, with the following additions:

* + 1. Viewing space SEI payload 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 <= es\_num\_primitive\_shapes\_minus1; s++ ) { |  |
| if( es\_camera\_inferred\_flag[ e ] ) |  |
| **es\_view\_idx**[ e ][ s ] | u(v) |
| **es\_primitive\_shape\_type**[ e ][ s ] | u(2) |
| if( es\_primitive\_shape\_type[ e ][ s ] == 0) |  |
| cuboid\_primitive (e , s) |  |
| else if( es\_primitive\_shape\_type[ e ][ s ] == 1 ) |  |
| spheroid\_primitive (e, s) |  |
| else if( es\_primitive\_shape\_type[ e ][ s ]== 2 ) |  |
| halfspace\_primitive(e, s) |  |
| if( es\_guard\_band\_present\_flag[ e ] ) |  |
| **es\_guard\_band\_size**[ e ][ s ] | fl(16) |
| if( es\_primitive\_orientation\_present\_flag[ e ] ) { |  |
| if( !es\_camera\_inferred\_flag[ e ] ) { |  |
| **es\_primitive\_shape\_quat\_x**[ e ][ s ] | fl(16) |
| **es\_primitive\_shape\_quat\_y**[ e ][ s ] | fl(16) |
| **es\_primitive\_shape\_quat\_z**[ e ][ s ] | fl(16) |
| } |  |
| } |  |
| if( es\_viewing\_direction\_constraint\_present\_flag[ e ] ) { |  |
| if( es\_guard\_band\_present\_flag[ e ]) |  |
| **es\_guard\_band\_direction\_size**[ e ][ s ] | fl(16) |
| if( !es\_camera\_inferred\_flag[ e ] ) { |  |
| **es\_primitive\_shape\_viewing\_direction\_quat\_x\_center**[ e ][ s ] | 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( !es\_camera\_inferred\_flag[ e ] ) { |  |
| **cp\_center\_x**[ e ][ s ] | fl(16) |
| **cp\_center\_y**[ e ][ s ] | fl(16) |
| **cp\_center\_z**[ e ][ s ] | fl(16) |
| } |  |
| **cp\_size\_x**[ e ][ s ] | fl(16) |
| **cp\_size\_y**[ e ][ s ] | fl(16) |
| **cp\_size\_z**[ e ][ s ] | fl(16) |
| } |  |

* + - 1. Spheroid primitive

|  |  |
| --- | --- |
| spheroid\_primitive( e, s ) { | **Descriptor** |
| if( !es\_camera\_inferred\_flag[ e ] ) { |  |
| **sp\_center\_x**[ e ][ s ] | fl(16) |
| **sp\_center\_y**[ e ][ s ] | fl(16) |
| **sp\_center\_z**[ e ][ s ] | fl(16) |
| } |  |
| **sp\_radius\_x**[ e ][ s ] | fl(16) |
| **sp\_radius\_y**[ e ][ s ] | fl(16) |
| **sp\_radius\_z**[ e ][ s ] | fl(16) |
| } |  |

* + - 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 payload 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. Packed independent regions SEI payload syntax

|  |  |
| --- | --- |
| packed\_independent\_regions( ) { | **Descriptor** |
| **pir\_num\_packed\_frames\_minus1** | u(5) |
| for( j= 0; j <= pir\_num\_packed\_frames\_minus1; j++ ) { |  |
| **pir\_packed\_frame\_id**[ j ] | u(5) |
| k = pir\_packed\_frame\_id[ j ] |  |
| **pir\_description\_type\_idc** [ k ] | u(2) |
| **pir\_num\_regions\_minus1**[ k ] | u(8) |
| for( i = 0; i <= pir\_num\_regions\_minus1[ k ]; i++ ) { |  |
| if( pir\_description\_type\_idc[ k ] = = 0 ) { |  |
| **pir\_top\_left\_tile\_idx**[ k ][ i ] | ue(v) |
| **pir\_bottom\_right\_tile\_idx**[ k ][ i ] | ue(v) |
| } |  |
| else if( pir\_description\_type\_idc = = 1 ) { |  |
| **pir\_subpic\_id**[ k ][ i ] | ue(v) |
| } |  |
| } |  |
| } |  |
| } |  |

* 1. SEI payload semantics

The specifications in clause F.3 of [V3C] Annex F and its subclauses apply, with the following additions:

* + 1. Viewing space SEI payload semantics
       1. General

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

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

**vs\_elementary\_shape\_operation** [ e ]equal to 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 subtractive. 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 views 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 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 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 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 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 floating point value expressed in degree 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 floating point value expressed in degree 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 payload 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 (H.3). All positive even values are reserved for future use by ISO/IEC. All odd values are unspecified (available for specification by other standards).

**gup\_erode\_threshold** specifies the threshold that is applied in the texture aligned geometry erosion process (H.3.5) 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 (H.3.5) 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 (H.3.6) 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. Packed independent regions SEI payload semantics

**pir\_num\_packed\_frames\_minus1** plus 1 specifies the number of packed frames for which independently decodable region information is signaled.

**pir\_packed\_frame\_id**[ j ] specifies the ID of the pack with index j. The value of vps\_packed\_frame\_id[ j ] shall be in the range of 0 to 15, inclusive.

**pir\_description\_type\_idc**[ k ] equal to 0 indicates that tile indices of the top left and bottom right tiles of an independently decodable rectangular region (such as a temporal motion-constrained tile set specified in ISO/IEC 23008-2) are signaled for the rectangles of the k-th packed frame. pir\_description\_type\_idc equal to 1 indicates that sub pictures IDs, as specified in ISO/IEC 23090-3, are signaled for the regions of the k-th pack.ed frame The value of pir\_description\_type\_idc[ k ] shall be in the range of 0 to 1. The values in the range of 2 .. 3 are reserved for future use by ISO/IEC MPEG.

**pir\_num\_regions\_minus1**[ k ] plus 1 specifies the number of rectangular regions of the packed frame with ID k for which independently decodable region information is signaled.

**pir\_top\_left\_tile\_idx**[ k ][ i ] and **pir\_bottom\_right\_tile\_idx**[ k ][ i ], when present, identify the tile indices of the top-left tile and the bottom-right tile of an independently decodable region (such as a temporal motion-constrained tile set specified in ISO/IEC 23008-2), respectively, in tile raster scan order, corresponding to the i-th region of the video sub-bitstream of the packed frame with ID k.

**pir\_subpic\_id**[ k ][ i ], when present, identifies the subpicture ID, as specified in ISO/IEC 23090-3, corresponding to the i-th rectangle of the video sub-bitstream of the pack with ID k.

1. (informative)  
   Volumetric usability information

The specifications in [V3C] Annex G apply with the following modifications.

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

In this version of this specification, the value of vui\_display\_box\_present\_flag and vui\_anchor\_point\_present\_flag shall be equal to zero.

1. (Informative)  
   Hypothetical view rendering process

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

A block diagram of the hypothetical view rendering process is shown in Figure H.1.

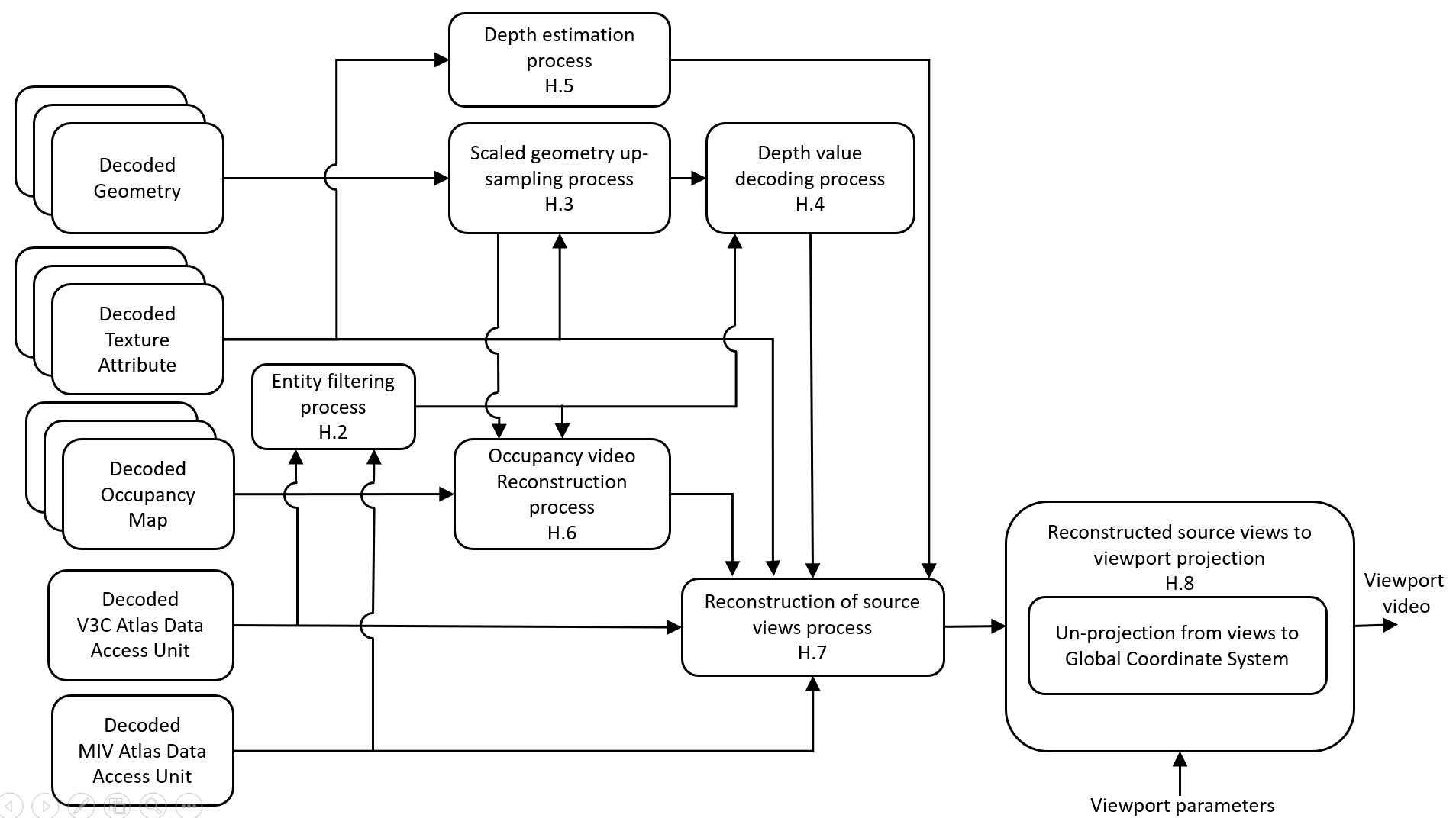


Figure H.1: Block diagram of hypothetical view renderer

* 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:
  + when vps\_occupancy\_video\_present\_flag[ a ] equal to 1, one decoded occupancy picture, decOccFrame[ frameIdx ][ 0 ][ ][ ], of size AsmeOccupancyFrameWidth[ a ] x AsmeOccupancyFrameHeight[ a ]
  + when vps\_geometry\_video\_present\_flag[ a ] equal to 1, one decoded geometry picture, decGeoFrame[ 0 ][ frameIdx ][ 0 ][ ][ ], of size AsmeGeometryFrameWidth[ a ] x AsmeGeometryFrameHeight[ a ]
  + one texture attribute picture decAttrFrame[ 0 ][ 0 ][ 0 ][ frameIdx ][ compIdx ][  ][  ] of size AspsFrameWidth[ a ] x AspsFrameHeight[ a ], with compIdx in 0 ..2.
  + one BlockToPatchMap[ ][ ] of size (AspsFrameWidth[ a ] /  AtlasPatchPackingBlockSize[ a ]) x ( AspsFrameHeight[ a ]  / AtlasPatchPackingBlockSize[ a ] )
  + asme\_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 data unit
* a viewing position (viewport\_pos\_x, viewport\_pos\_y, viewport\_pos\_z)
* a viewing orientation (viewport\_quat\_x, viewport\_quat\_y, viewport\_quat\_z)
* a viewport picture width, picW, and picture height, picH
* the max number of entities, vme\_max\_entities\_minus1
* an array of flags indicating if each entity is targeted for rendering, TargetEntityFlag[ ], of length vme\_max\_entities\_minus1 + 1, determined via external means. If vme\_max\_entities\_minus1 equal to 0, TargetEntityFlag[ 0 ] is set equal to 1
* vme\_depth\_low\_quality\_flag
* vme\_embedded\_occupancy\_flag, vme\_occupancy\_scaled\_enabled\_flag if present

Outputs of this process are for each access unit, viewports represented by the following

* ViewportDepth][ ][ ], of size picW x picH [Ed. (JB): Need to use these variable names in the sub-processes.]
* ViewportTextureAttribute[compIdx ][ ][ ], of size picW x picH, for compIdx in 0 ..2

The rendering process operates as follows for each access unit:

1. If vme\_max\_entities\_minus1 > 0. an entity filtering process as specified in clause H.2 is invoked for each atlas, a, with inputs BlockToPatchMap[ ][ ] of size BlockToPatchMapWidth x BlockToPatchMapHeight and output FilteredBlockToPatchMap[ ][ ] of size BlockToPatchMapWidth x BlockToPatchMapHeight . [Ed. (JB): Consider adding an atlas dimension to the variables.]
2. The geometry video scaling process as specified in in clause H.3 is invoked for each atlas, a, if present, with input decGeoFrame[ 0 ][ 0 ][ 0 ][ ][ ] of size AsmeGeometryFrameWidth[ a ] x AsmeGeometryFrameHeight[ a ] and ouptut GeoFrame[ 0 ][ 0 ][ 0 ][ ][ ] of size AspsFrameWidth[ a ] x AspsFrameHeight[ a ]
3. For each atlas, a, if vps\_geometry\_video\_present\_flag[ a ] equal to 1 or asme\_patch\_constant\_depth\_flag equal to 1, the depth decoding process as specified in clause H.4 is invoked for each atlas, for each sample (x, y) in the geometry decoded and upscaled picture of size AspsFrameWidth[ a ] x AspsFrameHeight[ a ], with input sample value GeoFrame[ 0 ][ 0 ][ 0 ][ y ][ x ], and output sample value, MetricDepth[ a ][ y ][ x ] within the metric depth map of size AspsFrameWidth[ a ] x AspsFrameHeight[ a ].
4. For each atlas, a, if vps\_geometry\_video\_present\_flag[ a ] equal to 0 and asme\_patch\_constant\_depth\_flag equal to 0, the depth estimation process as specified in clause ‎H.5 is invoked for each sample (x, y) of the metric depth map of size AspsFrameWidth[ a ] x AspsFrameHeight[ a ], with output sample value, MetricDepth[ a ][ y ][ x ].
5. The occupancy video reconstruction process as specified in clause H.6 is invoked for each atlas, for each sample in the output occupancy frame, OccupancyFrame of size AspsFrameWidth[ a ] x AspsFrameHeight[ a ].
6. The reconstructed view process as specified in clause H.7 is invoked for each view v in 0 .. mvp\_num\_views\_minus1.
7. The projection of pixels of reconstructed view to viewport process as specified in clause H.8 is invoked for each reconstructed view v in 0 .. mvp\_num\_views\_minus1. The final viewport texture ViewportTextureAttribute[ ][ ][ ] mentioned in H8. integrates over all views.
   1. Entity filtering process

The entity filtering process filters the BlockToPatchMap such that only patches related to targeted entities are kept and patches for not targeted entities are replaced by unoccupied value -1 to avoid being included in the rendering process.

Input to this process:

* the max number of entities, vme\_max\_entities\_minus1
* an array of flags indicating if each entity is targeted for rendering, TargetEntityFlag[ ], of length msp\_max\_entities\_minus1 + 1, determined via external means. If msp\_max\_entities\_minus1 equal to 0, TargetEntityFlag[ 0 ] is set equal to 1
* BlockToPatchMap[ y ][ x ] for atlas a, and its width, BlockToPatchMapWidth, and height, BlockToPatchMapHeight

Output to this process:

* Filtered BlockToPatchMap[a][ y ][ x ] for atlas a, of size BlockToPatchMapWidth x BlockToPatchMapHeight

The entity filtering process operates on BlockToPatchMap for atlas a as follows:

if( vme\_max\_entities\_minus1 > 0 ) {

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

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

patchId = BlockToPatchMap[y][x]

if( patchId != -1 && !TargetEntityFlag[pdu\_entity\_id[ tileId ][ patchId]] ) {

BlockToPatchMap[y][x] = -1

}

}

}

}

* 1. Geometry video scaling process

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

The inputs of this process are decGeoFrame[ 0 ][ 0 ][ 0 ][ ][ ] of size AsmeGeometryFrameWidth[ a ] x AsmeGeometryFrameHeight[ a ] and atlas parameters for the a-th atlas.

The output of this process is GeoFrame[ 0 ][ 0 ][ 0 ][ y ][ x ] of size AspsFrameWidth[ a ] x AspsFrameHeight[ a ] .

If AspsFrameWidth[ a ] == AsmeGeometryFrameWidth[ a ] and AspsFrameHeight[ a ] == AsmeGeometryFrameHeight[ 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 H.3.1) is invoked.
2. The texture aligned geometry erosion process (section H.3.5) is invoked.
3. The geometry contour smoothening process (section H.3.6) is invoked.

The sample neighbours enumeration process (section H.3.2), foreground edge flag process (section H.3.3), and selective geometry erosion process (section H.3.4) 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 ][ dec\_y ][ dec\_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 / AsmeGeometryFrameScaleFactorY[ a ]

u = x / AsmeGeometryFrameScaleFactorX[ 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 H.3.2).

The process is defined by the following procedure, where NgX, NgY and NumNeighbours are computed according to H.3.2 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 H.3.2). This process is defined by the following procedure, where NgX, NgY and NumNeighbours are computed according to H.3.2 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 ]
* decAttrFrame[ 0 ][ 0 ][ 0 ][ compIdx ][ y ][ x ] with compIdx in 0 .. 2

This process invokes the foreground edge frame process (section H.3.3) 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 H.3.2), and the selective geometry erosion process (section H.3.4) 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 H.3.2 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(decAttrFrame[ 0 ][ 0 ][ 0 ][ c ][ NgY[ i ] ][ NgX[ i ] ]

- decAttrFrame[ 0 ][ 0 ][ 0 ][ 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 ]
* decAttrFrame[ 0 ][ 0 ][ 0 ][ compIdx ][ y ][ x ] with compIdx in 0 ..2

This process invokes the foreground edge frame process (section H.3.3) 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 H.3.2), and the selective geometry erosion process (section H.3.4) 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 H.3.2 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)
* Sample value GeoFrame[ 0 ][ 0 ][ 0 ][ y ][ x ]
* BlockToPatchMap[ ][ ] and AtlasPatchPackingBlockSize[ a ] for atlas a
* Parameters from the atlas adaptation parameters set and Atlas tile data unit

Output of this process are:

* MetricDepth[ y ][ x ], the metric depth sample value

A normalized depth value DepthAtlasNormValue[ y ][ x ] is first derived as follows:

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

bSz = AtlasPatchPackingBlockSize[ a ]

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

if( p != -1 ) {

if ( !asme\_patch\_constant\_depth\_flag )

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

PduDepthStart[ a ][ p ], PduDepthEnd[ a ][ p ] )

else {

ClampedDepthSample[ y ][ x ] = PduDepthStart[ a ][ p ]

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

}

}

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

if( p != -1 ) {

v = pdu\_view\_idx[ 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[ a ][ y ][ x ] is derived as follows:

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

* 1. Depth Estimation Process

This process outputs metric depth value at the sample position in a geometry frame of atlas a at nominal atlas resolution.

Inputs to this process are:

* the sample position (x, y)
* decAttrFrame[ 0 ][ 0 ][ 0 ][ compIdx ][  y ][ x ] for all atlases and compIdx in 0 ..2
* the intrinsic and extrinsic parameters per view

Output of this process are:

* metric depth decoded pictures MetricDepth[ a ] [ y ][ x ] of atlas a

External means are used to derive the value of MetricDepth[ a ] [ y ][ x ] from the inputs.

* 1. Occupancy video reconstruction process

This process reconstructs an occupancy value at a given sample position in an occupancy frame at nominal atlas resolution.

Inputs to this process are:

* The sample position (x, y)
* If vme\_embedded\_occupancy\_flag equals 1, sample value GeoFrame[ 0 ][ orderIdx ][ 0 ][ y ][ x ] for the a-th atlas
* If vps\_occupancy\_video\_present\_flag[ a ] equals 1, decOccFrame[ 0 ][ orderIdx ][ 0 ][][ ] for the a-th atlas
* atlas parameters for the a-th atlas including AsmeOccupancyFrameScaleFactorX and AsmeOccupancyFrameScaleFactorY,
* BlockToPatchMap[][ ] and AtlasPatchPackingBlockSize[ a ] for atlas a
* Parameters from the atlas adaptation parameters set and Atlas tile data unit.

Output of this process are:

* OccupancyFrame [ a ][ y ][ x ]

The value of OccupancyFrame[ a ][ y ][ x ] is derived as follows:

bSz = AtlasPatchPackingBlockSize[ a ]

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

if (p == -1)

OccupancyFrame[ a ][ y ][ x ] = 0

else if( !vps\_occupancy\_video\_present\_flag[ a ] ){

if( vme\_embedded\_occupancy\_flag) {// occupancy is embedded in geometry

if( !asme\_depth\_occ\_threshold\_flag) {

v = pdu\_view\_idx [ p ]

DepthOccupancyThreshold = dq\_depth\_occ\_threshold\_default[ v ]

} else

DepthOccupancyThreshold = pdu\_depth\_occ\_threshold[ p ]

OccupancyFrame[ a ][ y ][ x ] =

( GeoFrame[ 0 ][ orderIdx ][ 0 ][ y ][ x ] < DepthOccupancyThreshold ) ? 0 : 1

} else {// no occupancy information is available

OccupancyFrame[ a ][ [ y ][ x ] = 1

}

} else // occupancy is signaled explicitly

OccupancyFrame[ a ][ y ][ x ] = decOccFrame[ 0 ][ orderIdx ][ 0 ]

[ y / AsmeOccupancyFrameScaleFactorY ][ x / AsmeOccupancyFrameScaleFactorX ]

* 1. Reconstruction of pruned views process

[Ed. (RD): the version beyond CD will point to the part 5 document once the H9.3.4 with the table below will be moved to main part 5, so this paragraph edition is provisional]

Inputs to this process are:

* miv\_view\_params\_list parameters related to view v
* metric depth decoded pictures MetricDepth[ a ] [ y ][ x ] of all atlases a, as defined in clause H.4
* occupancy value map OccupancyFrame[ a ][ y ][ x ] for all atlases a, as defined in clause H.6
* decAttrFrame[ 0 ][ 0 ][ 0 ][ compIdx ][ y ][ x ] with compIdx in 0 .. 2
* BlockToPatchMap[a][y][x ] for all atlas a
* atlas parameters for all atlases

Output of this process is the reconstructed pruned view v composed of:

* ReconstructedDepth[ v ][ ][ ] of size (ci\_projection\_plane\_width\_minus1[ v ] + 1) x (ci\_projection\_plane\_width\_minus1[ v ] + 1)
* ReconstructedTextureAttribute[ v ][compIdx ][ ][ ] of size (ci\_projection\_plane\_width\_minus1[ v ] + 1) x (ci\_projection\_plane\_width\_minus1[ v ] + 1) for compIdx in 0 ..2

The reconstructed depth for the view v ReconstructedDepth[ v ] is set to default value InvalidDepth (NaN).

for( yv=0; yv <= ci\_projection\_plane\_height\_minus1[ v ]; yv ++)

for( xv=0; xv <= ci\_projection\_plane\_width\_minus1[ v ]; xv ++)

ReconstructedDepth[v ][ yv ][ xv ]= InvalidDepth

The process uses the coordinates change from (x, y) in the atlas to (s, t) in the patch as follows:

mapping (x, y, p):

Inputs to this process are:

* coordinates (x, y) in atlas
* p the patch index

Outputs of this process are

* coordinates ( s, t) in the patch

The variable orientationIdx is set to AtlasPatchOrientationIndex[ p ].

The atlas coordinates ( s, t) are obtained as follows:

where the outputs of functions Rotation(x) and Offset(x) are matrices as specified in Table H‑17.

Table H‑17 Flexible patch orientation

|  |  |  |  |
| --- | --- | --- | --- |
| **x** | **Identifier** | **Rotation( x )** | **Offset( x )** |
| 0 | FPO\_NULL |  |  |
| 1 | FPO\_SWAP |  |  |
| 2 | FPO\_ROT90 |  |  |
| 3 | FPO\_ROT180 |  |  |
| 4 | FPO\_ROT270 |  |  |
| 5 | FPO\_MIRROR |  |  |
| 6 | FPO\_MROT90 |  |  |
| 7 | FPO\_MROT180 |  |  |

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

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

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

bSz = AtlasPatchPackingBlockSize[ a ]

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

if( pdu\_view\_id[ p ] == v ) {

if( OccupancyFrame[ a ][ y ][ x ] == 1) { \\ valid sample (x,y) of atlas a to be used for that view

(s, t ) = mapping(x, y, p)

xv = s + pdu\_view\_pos\_x [ tileId ][ p ]

yv = t + pdu\_view\_pos\_y [ tileId ][ p ]

ReconstructedDepth[ v ][ yv ][ xv ] = MetricDepth[ a ][ y ][ x ]

ReconstructedTextureAttribute [ v ][ compIdx ][ yv ][ xv ] = decAttrFrame [ a ][ y ][ x ] for compdIx = 0..2

}

}

}

* 1. Projection of pixels of reconstructed views to viewport

[(RD): some parts of H.8 will be precised further in next version ]

Inputs to this process are:

* ReconstructedDepth[ v ] and ReconstructedTextureAttribute[ v ] for view v as defined in clause H.6
* view parameters list from the miv\_view\_params\_list
* target viewport size, picW x picH
* target viewport position (viewport\_pos\_x, viewport\_pos\_y, viewport\_pos\_z) and orientation ( (quat\_x, viewport\_quat\_y, viewport\_quat\_z)

Output to this process is:

* a viewport texture frame ViewportTexture[ ][ ] of size picW x picH

First, the reconstructed views are deprojected according to clause H.8.2 by generating the global coordinate positions GlobalCoordinatePositionMap[ v ][ ][ ] of all valid samples.

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

This viewWeight[v] and GlobalCoordinatePositionMap[ v ][ ][ ] are inputs to the pixel weighting recovery process according to clause H8.3 which generates WeightPixel[ v ][ ][ ] for all samples of the view v

For each sample in the viewport texture frame ViewportTexture[ ][ ] of size picW x picH, the visibility pass first generates a depth map for the target viewport. Then the shading step computes the contribution to the target viewport texture.

Each not pruned pixel of the reconstructed views is blended through the use of its WeightPixel[ v ] 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.

This blending integrates over all views to generate ViewportTexture[ ][ ] for all the viewport samples.

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

Inputs to this process are:

* the depth frame of reconstructed view v, ReconstructedDepth[ v ][ ][ ] of size (ci\_projection\_plane\_width\_minus1[ v ] + 1) x (ci\_projection\_plane\_width\_minus1[ v ] + 1)
* view parameters of view v

Output of this process is

* a global coordinate position map for v-th reconstructed view GlobalCoordinatePositionMap[ v ][ ][ ] of size (ci\_projection\_plane\_width\_minus1[ v ] + 1) x (ci\_projection\_plane\_width\_minus1[ v ] + 1]. Each position 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 H.8.2.1, H.7.2.2 and H.7.2.3 respectively.

* + - 1. ERP unprojection function

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 )))

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 function

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 function

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:

* GlobalCoordinatePositionMap[ v ][ ][ ] of size (ci\_projection\_plane\_width\_minus1[ v ] + 1) x (ci\_projection\_plane\_width\_minus1[ v ] + 1], 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 ]
* asme\_auxiliary\_atlas\_flag[ a ] of the atlas which pixel p belongs to

Output to this process is:

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

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

If asme\_auxiliary\_atlas\_flag[ a ] is equal to 0, the variable arrays IsLeafFlag[ v ], NumChildrenMinus1[ v ] and ChildId[ v ][ i ] are computed as follows:

for ( v = 0; v <= mvp\_num\_views\_minus1, v++ ) {

IsLeafFlag[ v ] = 1

NumChildrenMinus1[ v ] = -1

}

for ( v = 0; v <= mvp\_num\_views\_minus1, v++ ) {

if (pp\_is\_root\_flag[ v ] == 0 ) {

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

j = pp\_parent\_idx[ v ][ i ]

IsLeafFlag[ j ] = 0

ChildId[ j ][ NumChildrenMinus1[ j ]++ ] = v

}

}

}

The weight of the contribution of pixel p of v-th view to the target viewport is then derived as follows:

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

computeChildrenWeight (v, P) {

{

w = 0

if ( IsLeafFlag[ v ] == 0 ) {

for( i = 0; i <= NumChildrenMinus1[ v ] ; i++) {

vChild = ChildId[ 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/)