1. **** **ISO/IEC JTC 1/SC 29/WG 4 N00004**

**ISO/IEC JTC 1/SC 29/WG 4**

**MPEG Video Coding Convenorship: CN**

**Document type:** Output document

**Title:** Potential Improvements of MIV

**Status:** Approved

**Date of document:** 2020-11-06

**Source:** ISO/IEC JTC 1/SC 29/WG 4

* **Expected action:** None
* **Action due date:** None

**No. of pages:** 73 (without cover pages)

**Email of Convenor:** yul@zju.edu.cn

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg4>

**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 4, MPEG VIDEO CODING**

**ISO/IEC JTC1/SC29/WG4 N00004**

**October 2020, Online**

|  |  |
| --- | --- |
| **Title** | **Potential Improvements of MIV** |
| **Source** | **WG 4, MPEG Video Coding** |
| **Status** | **Approved** |
| **Serial number** | **19677** |

Contents

[Foreword vii](#_Toc55548452)

[Introduction viii](#_Toc55548453)

[1 Scope 1](#_Toc55548454)

[2 Normative reference 1](#_Toc55548455)

[3 Terms and definitions 1](#_Toc55548456)

[4 Abbreviated terms 3](#_Toc55548457)

[5 Conventions 3](#_Toc55548458)

[5.1 Mathematical functions 3](#_Toc55548459)

[6 Overall V3C characteristics, decoding operations, and post-decoding processes 3](#_Toc55548460)

[7 Bitstream format, partitioning, and scanning processes 3](#_Toc55548461)

[7.1 General 3](#_Toc55548462)

[7.2 V3C bitstream formats 4](#_Toc55548463)

[7.3 NAL bitstream formats 4](#_Toc55548464)

[7.4 Partitioning of atlas frames into tiles 4](#_Toc55548465)

[7.5 Tile partition scanning processes 4](#_Toc55548466)

[7.6 Mapping of views to V3C components 4](#_Toc55548467)

[7.7 Sources and outputs 5](#_Toc55548469)

[8 Syntax and semantics 6](#_Toc55548470)

[8.1 Syntax in tabular form 6](#_Toc55548471)

[8.1.1 V3C MIV extension syntax 6](#_Toc55548472)

[8.2 Semantics 11](#_Toc55548473)

[8.2.1 V3C MIV extension semantics 11](#_Toc55548474)

[8.2.2 Order of V3C units and association to coded information 19](#_Toc55548475)

[9 Decoding process 20](#_Toc55548476)

[9.1 General decoding process 20](#_Toc55548477)

[9.2 Atlas data decoding process 22](#_Toc55548478)

[9.2.1 General atlas data decoding process 22](#_Toc55548479)

[9.2.2 Decoding process for a coded atlas frame 22](#_Toc55548480)

[9.2.3 Atlas NAL unit decoding process 22](#_Toc55548481)

[9.2.4 Atlas tile header decoding process 22](#_Toc55548482)

[9.2.5 Decoding process for patch data units 22](#_Toc55548483)

[9.2.6 Decoding process of the block to patch map 23](#_Toc55548484)

[9.2.7 Conversion of tile level patch information to atlas level patch information 23](#_Toc55548485)

[9.3 Occupancy video decoding process 24](#_Toc55548486)

[9.4 Geometry video decoding process 24](#_Toc55548487)

[9.5 Attribute video decoding process 24](#_Toc55548488)

[9.6 Packed video decoding process 24](#_Toc55548489)

[9.7 Common atlas data decoding process 24](#_Toc55548490)

[9.7.1 General common atlas data decoding process 24](#_Toc55548491)

[9.7.2 Decoding process for a coded common atlas frame 25](#_Toc55548492)

[9.7.3 Common atlas NAL unit decoding process 25](#_Toc55548493)

[9.7.4 Common atlas frame order count derivation process 25](#_Toc55548494)

[9.7.5 Common atlas frame MIV extension decoding process 25](#_Toc55548495)

[9.8 Sub-bitstream extraction process 31](#_Toc55548496)

[9.8.1 General 31](#_Toc55548497)

[9.8.2 V3C unit extraction 31](#_Toc55548498)

[9.8.3 NAL unit extraction process 31](#_Toc55548499)

[10 Pre-reconstruction process 32](#_Toc55548500)

[11 Reconstruction process 32](#_Toc55548501)

[12 Post-reconstruction process 32](#_Toc55548502)

[13 Adaptation process 32](#_Toc55548503)

[14 Parsing process 32](#_Toc55548504)

[Annex A (normative) Profiles, tiers, and levels 33](#_Toc55548505)

[A.1 Overview of profiles, tiers, and levels 33](#_Toc55548506)

[A.2 Profile, tier and level structure 33](#_Toc55548507)

[A.3 CodecGroup profile components 33](#_Toc55548508)

[A.4 Toolset profile components 33](#_Toc55548509)

[A.4.1 MIV Main, MIV Extended, and MIV Geometry Absent toolset profile component 33](#_Toc55548510)

[A.5 Reconstruction profile components 35](#_Toc55548511)

[A.6 Tiers and Levels 35](#_Toc55548512)

[A.7 Decoder instantiations 35](#_Toc55548513)

[Annex B (informative) Post-decoding conversion to nominal video formats 36](#_Toc55548514)

[B.1 General 36](#_Toc55548515)

[B.2 Nominal format conversion 36](#_Toc55548516)

[B.3 Conversion operations 36](#_Toc55548517)

[B.4 Unpacking process of a decoded packed video 36](#_Toc55548518)

[Annex C (informative) V3C sample stream format 37](#_Toc55548519)

[Annex D (normative) NAL sample stream format 38](#_Toc55548520)

[Annex E (normative) Atlas hypothetical reference decoder 39](#_Toc55548521)

[Annex F (normative) Supplemental enhancement information 40](#_Toc55548522)

[F.1 General 40](#_Toc55548523)

[F.2 SEI payload syntax 40](#_Toc55548524)

[F.2.1 Viewing space SEI payload syntax 40](#_Toc55548525)

[F.2.2 Viewing space handling SEI payload syntax 42](#_Toc55548526)

[F.2.3 Geometry upscaling parameters SEI payload syntax 42](#_Toc55548527)

[F.3 SEI payload semantics 42](#_Toc55548528)

[F.3.1 General SEI payload semantics 42](#_Toc55548529)

[F.3.2 Viewing space SEI payload semantics 43](#_Toc55548530)

[F.3.3 Viewing space handling SEI payload semantics 48](#_Toc55548531)

[F.3.4 Geometry upscaling parameters SEI payload semantics 49](#_Toc55548532)

[Annex G (informative) Volumetric usability information 51](#_Toc55548533)

[Annex H (Informative) Overview of rendering processes 52](#_Toc55548534)

[H.1 General 52](#_Toc55548535)

[H.2 Sample 3D reconstruction process 53](#_Toc55548536)

[H.2.1 Sample occupancy reconstruction process 54](#_Toc55548537)

[H.2.2 Depth clamping process 55](#_Toc55548538)

[H.2.3 Depth expansion process 55](#_Toc55548539)

[H.2.4 Coordinate unprojection process 56](#_Toc55548540)

[H.2.5 Reference frame switching process 57](#_Toc55548541)

[H.3 Entity filtering process 58](#_Toc55548542)

[H.4 Patch attribute offset process 59](#_Toc55548543)

[H.5 Geometry video scaling process 59](#_Toc55548544)

[H.5.1 Nearest neighbour interpolation scaling process 60](#_Toc55548545)

[H.5.2 Sample neighbours enumeration process 60](#_Toc55548546)

[H.5.3 Foreground edge flag process 61](#_Toc55548547)

[H.5.4 Selective geometry erosion process 61](#_Toc55548548)

[H.5.5 Texture aligned geometry erosion process 62](#_Toc55548549)

[H.5.6 Geometry contour smoothening process 63](#_Toc55548550)

[H.6 Pruned view reconstruction process 63](#_Toc55548551)

[H.7 Sample weighting recovery processes 65](#_Toc55548552)

[H.7.1 Pruning graph parsing process 65](#_Toc55548553)

[H.7.2 Sample weighting recovery process 66](#_Toc55548554)

[Bibliography 68](#_Toc55548555)

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 document enables storage and distribution of immersive video content over existing and future networks, for playback with 6 degrees of freedom of view position and orientation.

**Information technology — Coded Representation of Immersive Media — Part 12: 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 CD 23090-5(2E).

# Normative reference

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

ISO/IEC 23090-2(2E), *Information technology — Coded Representation of Immersive Media — Part 2: Omnidirectional MediA Format (OMAF)*

ISO/IEC CD 23090-5(2E), *Information technology — Coded Representation of Immersive Media — Part 5: Visual Volumetric Video-based Coding (V3C) and Video-based Point Cloud Compression (V-PCC)*

# Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC CD 23090-5(2E) and the following apply.

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

* ISO Online browsing platform: available at [https://www.iso.org/obp](https://www.iso.org/obp/ui)
* IEC Electropedia: available at <http://www.electropedia.org/>

coded MIV sequence

a *coded V3C sequence* that is an *MIV IRAP access unit* (3.6) followed by zero or more *MIV access units* (3.4)



field of view

FOV

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

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



MIV access unit

a *V3C composition unit* that is a set of all *sub-bitstream access units* (3.10) 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* (3.11) followed by zero or more *V3C access units*



MIV IRAP access unit

a *V3C IRAP composition unit* that is an *MIV access unit* (3.6) for which all *sub-bitstream access units* (3.10) are *sub-bitstream IRAP access units* (3.11)



renderer

an embodiment of a process to create a *viewport* (3.18) from a *volumetric frame* corresponding to a *viewing orientation* (3.15) and *viewing position* (3.16)



source

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

source view

a term used to describe *source* (3.8) video material before encoding that corresponds to the format of a *view* (3.12), which may have been acquired by capture of a 3D scene by a real or virtual camera



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* (3.10) that forms an independent random-access point for the *sub-bitstream*



view

2D rectangular arrays of *view samples* (3.15) consisting of *attribute* framesand corresponding *geometry* framerepresenting the projection of a *volumetric frame* onto a surface using *view parameters* (3.13)



view parameters

defines the projection used to generate a *view* (3.12) from a *volumetric frame*, including intrinsic and extrinsic parameters

view parameters list

a list of one or more *view parameters* (3.13)



view sample

position on the rectangular frame associated with a *view* (3.12)



viewing orientation

a unit quaternion representing the orientation that a user is consuming the visual content



viewing position

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

viewing space

domain constraints for an intended *viewport* (3.19) rendering; the domain is defined in the 3D global space and related to the *viewing orientation* (3.16); 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

a *view* (3.12) suitable for display and viewing

# Abbreviated terms

The following abbreviations apply in addition to the abbreviations in ISO/IEC CD 23090-5(2E) clause 4.

|  |  |
| --- | --- |
| ERP | EquiRectangular Projection |
| HMD | Head-Mounted Display |
| OMAF | Omnidirectional MediA Format |
| PSP | Perspective Projection |

# Conventions

The specifications in ISO/IEC CD 23090-5(2E) clause 5 and its subclauses apply with the following addition to subclause 5.8.

## Mathematical functions

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

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

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

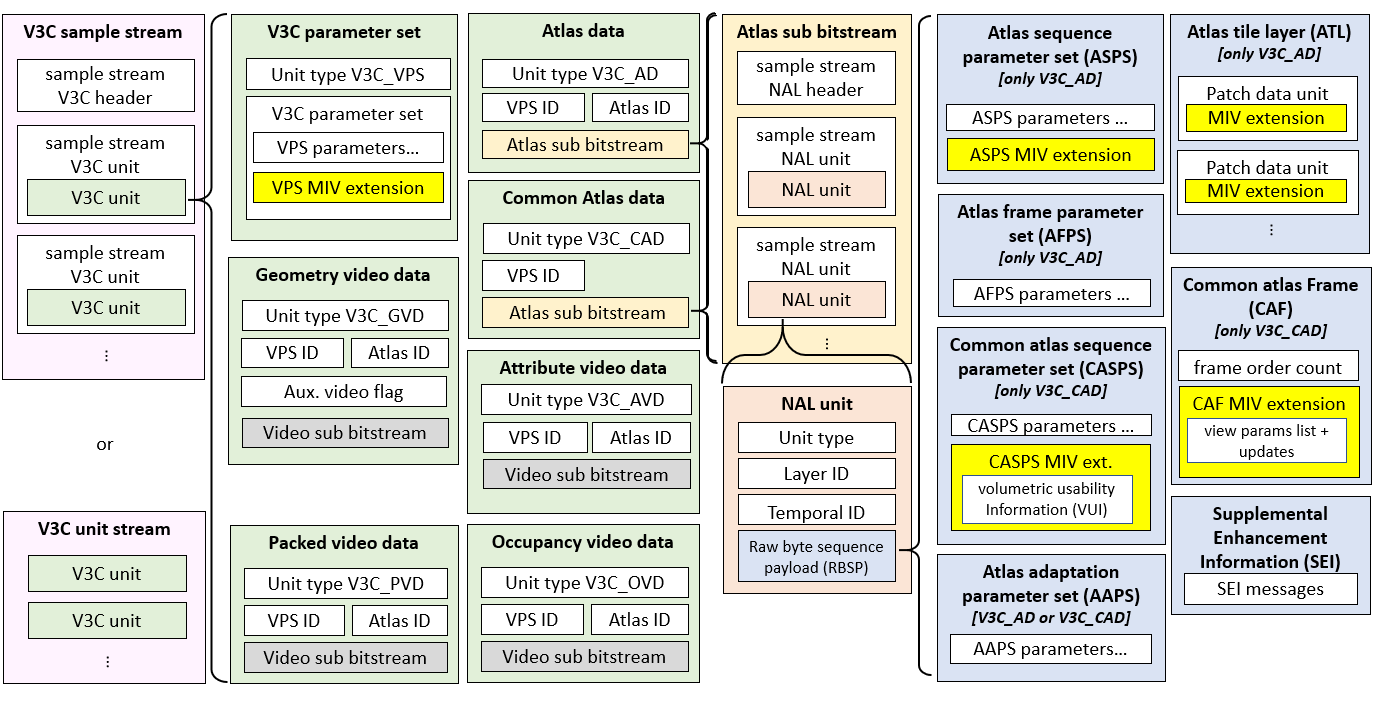
The specifications in the corresponding clause of ISO/IEC CD 23090-5(2E) apply.

# Bitstream format, partitioning, and scanning processes

## General

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

An overview of the V3C bitstream structure with MIV extensions is represented in **Figure 1**.



**Figure 1: Overview of V3C bitstream with MIV extensions**

[Ed. (LK) The corresponding figure in part 5 is placed in clause 8]

## V3C bitstream formats

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

## NAL bitstream formats

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

## Partitioning of atlas frames into tiles

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

## Tile partition scanning processes

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

## Mapping of views to V3C components

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

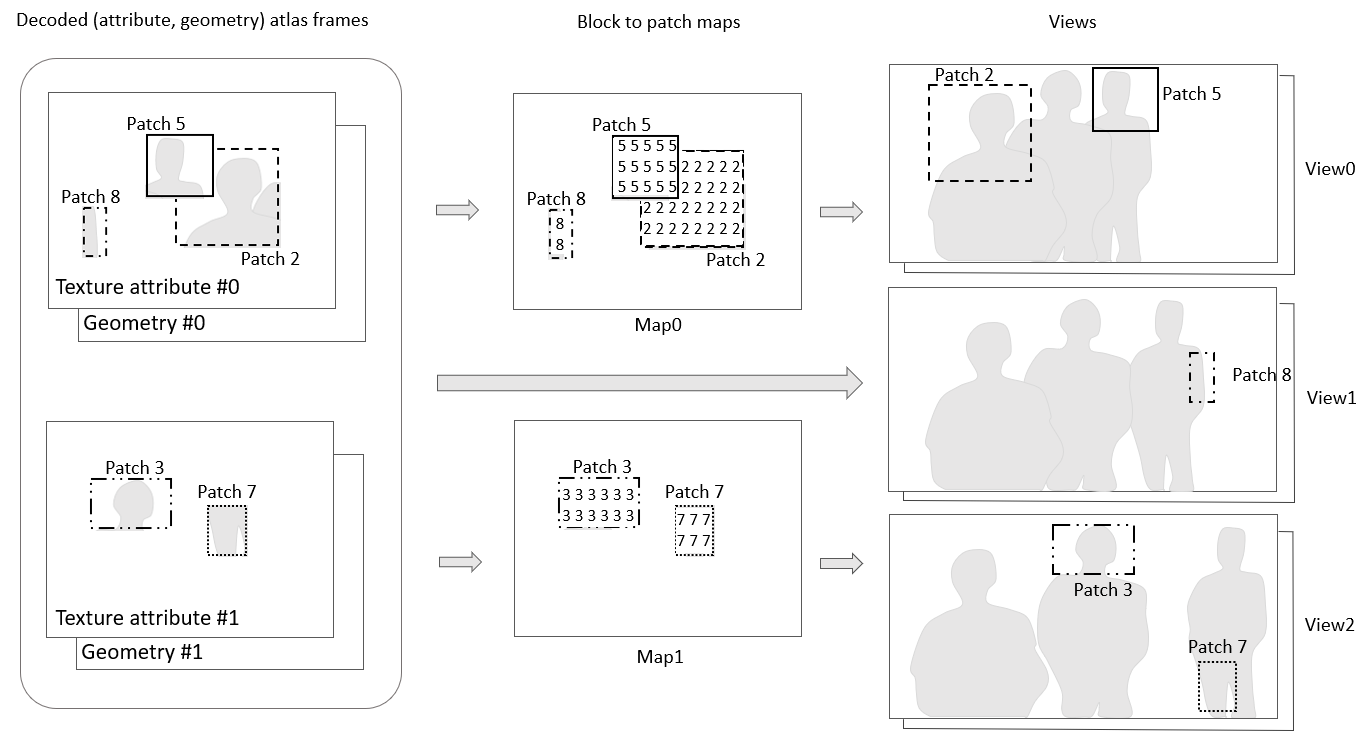
A view represents a field of view of a volumetric frame for particular view position and orientation. Each view, at a given time instance, is represented by two 2D frames, one providing texture information and one providing geometry information. Views may be omnidirectional or perspective, and may use different projection formats, such as equirectangular projection as defined in ISO/IEC 23090-2, cube map projection through multiple perspective projections, or orthographic as defined in ISO/IEC CD 23090-5(2E). In this version of the document, the texture attribute and geometry frames of a view use the same projection format.

The volumetric frame as well as all views each have an associated reference frame. Cartesian coordinates of 3D points can therefore be expressed according to the reference frame of the scene, that is the volumetric frame, or according to the reference frame of any view. The camera extrinsic parameters (position and orientation) of the views specify the relations between their reference frames, and thus enable reference frame switching for 3D points coordinates.

A rectangular region of a view is represented by a corresponding region, i.e., patch, in V3C attribute component and V3C geometry component. Additionally, V3C atlas component provides a patch information than among others includes a view ID information that indicates which view the patch originated from. V3C attribute and V3C geometry components are composed of frames that contains an aggregation of one or more patches from one or more views.

In this version of the document, the size of a patch in V3C attribute component is always the same as the corresponding rectangular region in the view texture component while the size of a patch in V3C geometry component may be different from the corresponding rectangular region in the view geometry component, i.e., scaling of a geometry is permitted.

Figure 2 shows an illustrative example, in which two atlases contain five patches, which are mapped to three views.



**Figure****2: Example mapping of 5 patches in 2 atlases to 3 views**

## Sources and outputs

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

The outputs of the decoding process are a view parameters list, and for each of one or more atlases, a sequence of the following: a texture attribute frame, a geometry frame, and a block to patch map, as described in subclause 9.1.

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

# Syntax and semantics

The specifications in ISO/IEC CD 23090-5(2E) clause 8 apply with the following additions.

## Syntax in tabular form

### V3C MIV extension syntax

#### 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\_entity\_id | ue(v) |
| vme\_embedded\_occupancy\_enabled\_flag | u(1) |
| if( !vme\_embedded\_occupancy\_enabled\_flag ) |  |
| vme\_occupancy\_scale\_enabled\_flag | u(1) |
| } |  |

[Ed. (LK, JB) Probably some of the syntax elements should move to casps\_miv\_extension. Need to consider one by one. A contribution would be needed. Related ballot comment #161]

#### Atlas sequence parameter set MIV extension syntax

|  |  |
| --- | --- |
| asps\_miv\_extension( ) { | **Descriptor** |
| **asme\_group\_idx** | u(v) |
| **asme\_auxiliary\_atlas\_flag** | u(1) |
| if( vme\_embedded\_occupancy\_enabled\_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\_enabled\_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) |
| **asme\_patch\_attribute\_offset\_enabled\_flag** | u(1) |
| if( asme\_patch\_attribute\_offset\_enabled\_flag ) |  |
| **asme\_patch\_attribute\_offset\_bit\_depth\_minus1** | ue(v) |
| } |  |

#### Common atlas sequence parameter set MIV extension syntax

|  |  |
| --- | --- |
| casps\_miv\_extension( ) { | **Descriptor** |
| **casme\_omaf\_v1\_compatible\_flag** | u(1) |
| **casme\_depth\_quantization\_params\_present\_flag** | u(1) |
| **casme\_vui\_params\_present\_flag** | u(1) |
| if( casme\_vui\_params\_present\_flag ) |  |
| vui\_parameters( ) |  |
| } |  |

#### Common atlas frame MIV extension syntax

|  |  |
| --- | --- |
| caf\_miv\_extension( ) { | **Descriptor** |
| if( nal\_unit\_type == NAL\_IRAP\_CAF ) { |  |
| miv\_view\_params\_list( ) |  |
| } else { |  |
| **came\_update\_extrinsics\_flag** | u(1) |
| **came\_update\_intrinsics\_flag** | u(1) |
| if( casme\_depth\_quantization\_params\_present\_flag ) |  |
| **came\_update\_depth\_quantization\_flag** | u(1) |
| if( came\_update\_extrinsics\_flag ) |  |
| miv\_view\_params\_update\_extrinsics( ) |  |
| if( came\_update\_intrinsics\_flag ) |  |
| miv\_view\_params\_update\_intrinsics( ) |  |
| if( came\_update\_depth\_quantization\_flag ) |  |
| miv\_view\_params\_update\_depth\_quantization( ) |  |
| } |  |
| } |  |

##### MIV view parameters list syntax

|  |  |
| --- | --- |
| miv\_view\_params\_list( ) { | **Descriptor** |
| **mvp\_num\_views\_minus1** | u(16) |
| **mvp\_explicit\_view\_id\_flag** | u(1) |
| if( mvp\_explicit\_view\_id\_flag ) { |  |
| for( v = 0; v <= mvp\_num\_views\_minus1; v++ ) { |  |
| **mvp\_view\_id**[ v ] | u(16) |
| ViewIDToIndex[ mvp\_view\_id[ v ] ] = v |  |
| ViewIndexToID[ v ] = mvp\_view\_id[ v ] |  |
| } |  |
| } else { |  |
| for( v = 0; v <= mvp\_num\_views\_minus1; v++ ) |  |
| mvp\_view\_id[ v ] = v |  |
| ViewIDToIndex[ mvp\_view\_id[ v ] ] = v |  |
| ViewIndexToID[ v ] = mvp\_view\_id[ v ] |  |
| } |  |
| } |  |
| **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++ ) { |  |
| atlasID = vps\_atlas\_id[ a ] |  |
| viewID = ViewIndexToID[ v ] |  |
| **mvp\_view\_enabled\_in\_atlas\_flag**[ atlasID ][ viewID ] | u(1) |
| if( mvp\_view\_enabled\_in\_atlas\_flag[ atlasID ][ viewID ] ) |  |
| **mvp\_view\_complete\_in\_atlas\_flag**[ atlasID ][ viewID ] | u(1) |
| } |  |
| } |  |
| for( v = 0; v <= mvp\_num\_views\_minus1; v++ ) { |  |
| viewID = ViewIndexToID[ v ] |  |
| camera\_extrinsics( viewID ) |  |
| } |  |
| **mvp\_intrinsic\_params\_equal\_flag** | u(1) |
| for( v = 0; v <= mvp\_intrinsic\_params\_equal\_flag ? 0 : mvp\_num\_views\_minus1; v++ ) { |  |
| viewID = ViewIndexToID[ v ] |  |
| camera\_intrinsics( viewID, 0 ) |  |
| } |  |
| if( came\_depth\_quantization\_params\_present\_flag ) { |  |
| **mvp\_depth\_quantization\_params\_equal\_flag** | u(1) |
| for( v = 0; v <= mvp\_depth\_quantization\_equal\_flag ? 0 : mvp\_num\_views\_minus1; v++ ) { |  |
| viewID = ViewIndexToID[ v ] |  |
| depth\_quantization( viewID ) |  |
| } |  |
| **mvp\_pruning\_graph\_params\_present\_flag** | u(1) |
| if ( mvp\_pruning\_graph\_params\_present\_flag ) |  |
| for( v = 0; v <= mvp\_num\_views\_minus1; v++ ) { |  |
| viewID = ViewIndexToID[ v ] |  |
| pruning\_parents( viewID ) |  |
| } |  |
| } |  |

##### 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\_id**[ i ] | u(16) |
| viewID = mvpue\_view\_id[ i ] |  |
| camera\_extrinsics( viewID ) |  |
| } |  |
| } |  |

##### 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\_id**[ i ] | u(16) |
| viewID = mvpui\_view\_id[ i ] |  |
| camera\_intrinsics( viewID, 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\_id**[ i ] | u(16) |
| viewID = mvpudq\_view\_id[ i ] |  |
| depth\_quantization( viewID ) |  |
| } |  |
| } |  |

##### Camera extrinsics syntax

|  |  |
| --- | --- |
| camera\_extrinsics( viewID ) { | **Descriptor** |
| **ce\_view\_pos\_x**[ viewID ] | fl(32) |
| **ce\_view\_pos\_y**[ viewID ] | fl(32) |
| **ce\_view\_pos\_z**[ viewID ] | fl(32) |
| **ce\_view\_quat\_x**[ viewID ] | i(16) |
| **ce\_view\_quat\_y**[ viewID ] | i(16) |
| **ce\_view\_quat\_z**[ viewID ] | i(16) |
| } |  |

##### Camera intrinsics syntax

|  |  |
| --- | --- |
| camera\_intrinsics( viewID, mode ) { | **Descriptor** |
| **ci\_cam\_type**[ viewID ] | u(8) |
| **ci\_projection\_plane\_width\_minus1**[ viewID ] | u(16) |
| **ci\_projection\_plane\_height\_minus1**[ viewID ] | u(16) |
| if( ci\_cam\_type[ viewID ] == 0 ) { /\* equirectangular \*/ |  |
| **ci\_erp\_phi\_min**[ viewID ] | fl(32) |
| **ci\_erp\_phi\_max**[ viewID ] | fl(32) |
| **ci\_erp\_theta\_min**[ viewID ] | fl(32) |
| **ci\_erp\_theta\_max**[ viewID ] | fl(32) |
| } else if( ci\_cam\_type[ viewID ] == 1 ) { /\* perspective \*/ |  |
| **ci\_perspective\_focal\_hor**[ viewID ] | fl(32) |
| **ci\_perspective\_focal\_ver**[ viewID ] | fl(32) |
| **ci\_perspective\_center\_hor**[ viewID ] | fl(32) |
| **ci\_perspective\_center\_ver**[ viewID ] | fl(32) |
| } else if( ci\_cam\_type[viewID] == 2 ) { /\* orthographic \*/ |  |
| **ci\_ortho\_width**[ viewID ] | fl(32) |
| **ci\_ortho\_height**[ viewID ] | fl(32) |
| } |  |
| } |  |

##### Depth quantization syntax

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

##### Pruning parents syntax

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

#### Patch data unit MIV extension syntax

|  |  |
| --- | --- |
| pdu\_miv\_extension( tileID, p ) { | **Descriptor** |
| if( vme\_max\_entity\_id > 0 ) |  |
| **pdu\_entity\_id**[ tileID ][ p ] | u(v) |
| if( asme\_depth\_occ\_threshold\_flag ) |  |
| **pdu\_depth\_occ\_threshold**[ tileID ][ p ] | u(v) |
| if( asme\_patch\_attribute\_offset\_enabled\_flag ) |  |
| for( c = 0; c < 3; c++ ) { |  |
| **pdu\_attribute\_offset**[ tileID ][ p ][ c ] | u(v) |
| } |  |

## Semantics

### V3C MIV extension semantics

#### V3C parameter set MIV extension semantics

**vme\_depth\_low\_quality\_flag** equal to 1 indicates that the depth fidelity confidence in geometry video sub-bitstreams 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 V3C sub-bitstream component corresponding to the geometry components, which are determined through either examining if vuh\_unit\_type is equal to V3C\_GVD or through external means if the V3C unit header is unavailable, may have a different coded picture width and height than the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When vme\_geometry\_scale\_enabled\_flag is equal to 0, it is a requirement for bitstream conformance that the picture width and picture height of the V3C sub-bitstream component corresponding to the geometry components be equal to the atlas frame width and height, respectively, specified in the active atlas sequence parameter set RBSP. When not present, the value of vme\_geometry\_scale\_enabled\_flag is inferred to be equal to 0.

**vme\_num\_groups\_minus1** specifies the maximum value that can be indicated for group index syntax element, asme\_group\_idx. When not present, the value of vme\_num\_groups\_minus1 is inferred to be equal to 0.

**vme\_max\_entity\_id** specifies the maximum value that can be indicated for the patch entity ID syntax element, pdu\_entity\_id[ i ][ j ], for a patch with index j in a tile with tile ID equal to i. When not present, the value of vme\_max\_entity\_id is inferred to be equal to 0.

**vme\_embedded\_occupancy\_enabled\_flag** equal to 1 specifies that the occupancy information is present in the V3C sub-bitstream component corresponding to the geometry component, which is determined through either examining if vuh\_unit\_type is equal to V3C\_GVD or through external means if the V3C unit header is unavailable. vme\_embedded\_occupancy\_enabled\_flag equal to 0 specifies that occupancy information is not present or present in the V3C sub-bitstream component corresponding to the occupancy component, which is determined through either examining if vuh\_unit\_type is equal to V3C\_OVD or through external means if the V3C unit header is unavailable.

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

#### Atlas sequence parameter set MIV extension semantics

Let the variable aspsAtlasID be either set equal to vuh\_atlas\_id of the ASPS RBSP or determined through external means if the V3C unit header is unavailable.

Let the variable numAtlases be either set equal to vps\_atlas\_count\_minus1 + 1 or determined through external means if the V3C VPS is unavailable. Let 1D arrays AspsFrameWidth and AspsFrameHeight of size numAtlases be derived as follows:

AspsFrameWidth[ aspsAtlasID ] = asps\_frame\_width(1)

AspsFrameHeight[ aspsAtlasID ] = asps\_frame\_height(2)

**asme\_group\_idx** specifies the group index of the atlas. The number of bits used for the representation of asme\_group\_idx 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 intended to 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.

[Ed. (LK) We should use more descriptive name. Also, auxiliary atlas may be confused with the auxiliary video. [Ed. (JB): Consider “ancillary” instead of “auxiliary”. ]

**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.

Let 1D arrays AsmeDepthOccThresholdFlag of size numAtlases be derived as follows:

AsmeDepthOccThresholdFlag[ aspsAtlasID ] = asme\_depth\_occ\_threshold\_flag(3)

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

Let 1D arrays AsmeGeometryFrameScaleFactorX and AsmeGeometryFrameWidth of size numAtlases be derived as follows:

AsmeGeometryFrameScaleFactorX[ aspsAtlasID ] =   
 asme\_geometry\_frame\_scale\_factor\_x\_minus1 + 1(4)

AsmeGeometryFrameWidth[ aspsAtlasID ] =   
 AspsFrameWidth[ aspsAtlasID ] / AsmeGeometryFrameScaleFactorX[ aspsAtlasID ](5)

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

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

Let 1D arrays AsmeGeometryFrameScaleFactorY and AsmeGeometryFrameHeight of size numAtlases be derived as follows:

AsmeGeometryFrameScaleFactorY[ aspsAtlasID ] =  
 asme\_geometry\_frame\_scale\_factor\_y\_minus1 + 1(6)

AsmeGeometryFrameHeight[ aspsAtlasID ]=  
 AspsFrameHeight[ aspsAtlasID ] / AsmeGeometryFrameScaleFactorY[ aspsAtlasID ] (7)

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

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

Let 1D arrays AsmeOccupancyFrameScaleFactorX and AsmeOccupancyFrameWidth of size numAtlases be derived as follows:

AsmeOccupancyFrameScaleFactorX[ aspsAtlasID ] =   
 asme\_occupancy\_frame\_scale\_factor\_x\_minus1 + 1(8)

tempDivX = AspsFrameWidth[ aspsAtlasID ] / AsmeOccupancyFrameScaleFactorX[ aspsAtlasID ]

AsmeOccupancyFrameWidth[ aspsAtlasID ] = tempDivX + tempDivX % 2(9)

[Ed. (LK) double check the equation above and check if there could be better name than tempDivX and tempDivY]

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

Let 1D arrays AsmeOccupancyFrameScaleFactorY and AsmeOccupancyFrameHeight of size numAtlases be derived as follows:

AsmeOccupancyFrameScaleFactorY[ aspsAtlasID ] =   
 asme\_occupancy\_frame\_scale\_factor\_Y\_minus1 + 1(10)

tempDivY = AspsFrameHeight[ aspsAtlasID ] / AsmeOccupancyFrameScaleFactorY[ aspsAtlasID ]

AsmeOccupancyFrameHeight[ aspsAtlasID ] = tempDivY + tempDivY %2(11)

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

Let 1D arrays AsmePatchConstantDepthFlag of size numAtlases be derived as follows:

AsmePatchConstantDepthFlag[ aspsAtlasID ] = asme\_patch\_constant\_depth\_flag(12)

**asme\_patch\_attribute\_offset\_enabled\_flag** equal to 1 indicates that the asps\_patch\_attribute\_offset\_bit\_depth\_minus1 syntax element is present in the syntax structure. asme\_patch\_attribute\_offset\_enabled\_flag equal to 1 indicates that the asps\_patch\_attribute\_offset\_bit\_depth\_minus1 syntax element is not present in the syntax structure.

**asme\_patch\_attribute\_offset\_bit\_depth\_minus1** plus 1 specifies the number of bits used to represent the pdu\_attribute\_offset[ tileID ][ p ][ c ] syntax element. asps\_patch\_attribute\_offset\_bit\_depth\_minus1 shall be in the range of 0 to ai\_attribute\_2d\_bit\_depth\_minus\_1, inclusive.

#### Common atlas sequence parameter set MIV extension semantics

**casme\_omaf\_v1\_compatible\_flag** specifies that the atlas texture frame of the atlas with ID equal to 0 is compatible for carriage within ISO/IEC 23090-2. When casme\_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 ISO/IEC 23090-2. When not present the value of casme\_omaf\_v1\_compatible\_flag is inferred to be equal to 0.

**casme\_depth\_quantization\_params\_present\_flag** equal to 1 indicates that the depth quantization parameters are present in the syntax structure. casme\_depth\_quantization\_params\_present\_flag equal to 0 indicates that the depth quantization parameters are not present in the syntax structure. When not present, the value of casme\_depth\_quantization\_params\_present\_flag is inferred to be equal to 1.

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

#### Common atlas frame MIV extension semantics

**came\_update\_extrinsics\_flag** equal to 1 indicates that the miv\_view\_params\_update\_extrinsics( ) syntax structure is present in this syntax structure. came\_update\_extrinsics\_flag equal to 0 indicates that the miv\_view\_params\_update\_extrinsics( ) syntax structure is not present in this syntax structure.

**came\_update\_intrinsics\_flag** equal to 1 indicates that the miv\_view\_params\_update\_intrinsics( ) syntax structure is present in this syntax structure. came\_update\_intrinsics\_flag equal to 0 indicates that the miv\_view\_params\_update\_intrinsics( ) syntax structure is not present in this syntax structure.

**came\_update\_depth\_quantization\_flag** equal to 1 indicates that the miv\_view\_params\_update\_depth\_quantization( ) syntax structure is present in this syntax structure. came\_update\_depth\_quantization\_flag equal to 0 indicates that the miv\_view\_params\_update\_depth\_quantization( ) syntax structure is not present in this syntax structure. When not present, the value of came\_update\_depth\_quantization\_flag is inferred to be equal to 0.

##### 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 volumetric frame.

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

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

The arrays ViewIDToIndex and ViewIndexToID provide a forward and inverse mapping, respectively, of the ID associated with each view and the order index of how each view was specified in the MIV view parameters list syntax.

**mvp\_view\_enabled\_present\_flag** equal to 1 indicates that mvp\_view\_enabled\_in\_atlas\_flag syntax element is present in miv\_view\_params\_list( ) syntax structure. mvp\_view\_enabled\_present\_flag equal to 0 indicates mvp\_view\_enabled\_in\_atlas\_flag syntax element is not present in miv\_view\_params\_list( ) syntax structure .

**mvp\_view\_enabled\_in\_atlas\_flag**[ atlasID ][ viewID ] equal to 1 indicates that patches in the atlas with atlas ID equal to atlasID may be associated with the view with view ID equal to viewID. mvp\_view\_enabled\_in\_atlas\_flag [ atlasID ][ viewID ] equal to 0 indicates that patches in the atlas with atlas ID equal to atlasID do not contain a representation of the view with view ID equal to viewID.

**mvp\_view\_complete\_in\_atlas\_flag**[ atlasID ][ viewID ] equal to 1 indicates that patches in the atlas with atlas ID equal to atlasID are associated with the complete representation of the view with view ID equal to viewID. mvp\_view\_complete\_in\_atlas\_flag[ atlasID ][ viewID ] equal to 0 does not indicate any constraints.

[ Ed. (LK) that is the old semantics of mvp\_view\_complete\_in\_atlas\_flag: equal to 1 indicates that all values of RecGeoFrame[ v ] output by the reconstruction of pruned view process for the a-th atlas specified in subclause H.6 are valid values.mvp\_view\_complete\_in\_atlas\_flag[ atlasID ][ viewID ] equal to 0 does not indicate a constraint.

**mvp\_intrinsic\_params\_equal\_flag** equal to 1 specifies that the intrinsic parameters are signalled only for the view with index 0 and each view with index in range of 1 to mvp\_num\_views\_minus1, inclusive, has the same intrinsic parameters as the view with index 0. mvp\_intrinsic\_params\_equal\_flag equal to 0 specifies that the intrinsic parameters are explicitly signalled for each view with index in range of 0 to mvp\_num\_views\_minus1, inclusive.

**mvp\_depth\_quantization\_params\_equal\_flag** equal to 1 specifies that the depth quantization parameters are signalled only for the view with index 0 and each view with index in range of 1 to mvp\_num\_views\_minus1, inclusive, has the same depth quantization parameters as the view with index 0. mvp\_depth\_quantization\_params\_equal\_flag equal to 0 specifies that the depth quantization parameters are explicitly signalled for each view with index in range of 0 to mvp\_num\_views\_minus1, inclusive.

**mvp\_pruning\_graph\_params\_present\_flag** equal to 1 specifies that the pruning\_parents( ) syntax structure is present in the miv\_view\_params\_list( ) syntax structure. mvp\_pruning\_graph\_params\_present\_flag equal to 0 specifies that the pruning\_parents( ) syntax structure is not present in the miv\_view\_params\_list( ) syntax structure.

##### MIV view parameters update extrinsics semantics

**mvpue\_num\_view\_updates\_minus1** plus 1 indicates the number of extrinsics 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**,** inclusive**.**

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

##### 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\_id**[ i ] specifies the ID of the view with index v. The value of mvpui\_view\_id[ i ] shall be in the range of 0 to 65 535, inclusive. It is a requirement of bitstream conformance to this version of this document that the value of mvpui\_view\_id[ j ] shall not be equal to mvpui\_view\_id[ k] for all j != k.

##### 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, inclusive**.**

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

##### Camera extrinsics semantics

**ce\_view\_pos\_x**[ viewID ] specifies in scene units the x-coordinate of the location of the view with view ID equal to viewID.

**ce\_view\_pos\_y**[ viewID ] specifies in scene units the y-coordinate of the location of the view with view ID equal to viewID.

**ce\_view\_pos\_z**[ viewID ] specifies in scene units the z-coordinate of the location of the view with view ID equal to viewID.

**ce\_view\_quat\_x**[ viewID ] specifies the x component, qX[ viewID ], for the rotation of the view with view ID equal to viewID using the quaternion representation. The value of ce\_view\_quat\_x[ viewID ] shall be in the range of − 214 to 214, inclusive. When ce\_view\_quat\_x[ viewID ] is not present, its value shall be inferred to be equal to 0. The value of qX[ viewID ] is computed as follows:

qX[ viewID ] = ce\_view\_quat\_x[ viewID ] ÷ 214(13)

**ce\_view\_quat\_y**[ viewID ] specifies the y component, qY[ viewID ], for the rotation of the view with view ID equal to viewID using the quaternion representation. The value of ce\_view\_quat\_y[ viewID ] shall be in the range of − 214 to 214, inclusive. When ce\_view\_quat\_y[ viewID ] is not present, its value shall be inferred to be equal to 0. The value of qY[ viewID ] is computed as follows:

qY[ viewID ] = ce\_view\_quat\_y[ viewID ] ÷ 214(14)

**ce\_view\_quat\_z**[ viewID ] specifies the z component, qZ[ viewID ], for the rotation of the view with view ID equal to viewID using the quaternion representation. The value of ce\_view\_quat\_z[ viewID ] shall be in the range of − 214 to 214, inclusive. When ce\_view\_quat\_z[ viewID ] is not present, its value shall be inferred to be equal to 0. The value of qZ[ viewID ] is computed as follows:

qZ[ viewID ] = ce\_view\_quat\_z[ viewID ] ÷ 214(15)

It is a requirement of bitstream conformance that:

qX[ viewID ]2 + qY[ viewID ]2 +qZ[ viewID ]2 <= 1(16)

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

qW[ viewID ] = Sqrt( 1 – ( qX[ viewID ]2 + qY[ viewID ]2 + qZ[ viewID ]2 ) )(17)

##### Camera intrinsics semantics

**ci\_cam\_type**[ viewID ] indicates the projection method of the view with view ID equal to viewID. ci\_cam\_type[ viewID ] equal to 0 specifies ERP projection. ci\_cam\_type[ viewID ] equal to 1 specifies a perspective projection. ci\_cam\_type[ viewID ] 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[ viewID ] is inferred to equal to ci\_cam\_type[ 0 ].

**ci\_projection\_plane\_width\_minus1**[ viewID ] plus 1 and **ci\_projection\_plane\_height\_minus1**[ viewID ] plus 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[ viewID ] and ci\_projection\_plane\_height\_minus1[ viewID ] are inferred to be equal to ci\_projection\_plane\_width\_minus1[ 0 ] and ci\_projection\_plane\_height\_minus1[ 0 ], respectively.

**ci\_erp\_phi\_min**[ viewID ] and **ci\_erp\_phi\_max**[ viewID ] specify the longitude range (minimum and maximum values) for an ERP projection, as floating-point in units of degrees. ci\_erp\_phi\_min[ viewID ] and ci\_erp\_phi\_max[ viewID ] shall be in the range −180 to 180 in the spherical coordinate system. When not present and mode equal to 0, the values of ci\_erp\_phi\_min[ viewID ] and ci\_erp\_phi\_max[ viewID ] 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[ viewID ] < ci\_erp\_phi\_max[ viewID ].

**ci\_erp\_theta\_min**[ viewID ] and **ci\_erp\_theta\_max**[ viewID ] specify the latitude range (minimum and maximum values) for an ERP projection, as floating-point in units of degrees. ci\_erp\_theta\_min[ viewID ] and ci\_erp\_theta\_max[ viewID ] shall be in the range −90 to 90 in the spherical coordinate system. When not present and mode equal to 0, the values of ci\_erp\_theta\_min[ viewID ] and ci\_erp\_theta\_max[ viewID ] 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[ viewID ] < ci\_erp\_theta\_max[ viewID ].

**ci\_perspective\_focal\_hor**[ viewID ] and **ci\_perspective\_focal\_ver**[ viewID ] 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[ viewID ] and ci\_perspective\_focal\_ver[ viewID ] are inferred to be equal to ci\_perspective\_focal\_hor[ 0 ] and ci\_perspective\_focal\_ver[ 0 ], respectively.

**ci\_perspective\_center\_hor**[ viewID ] and **ci\_perspective\_center\_ver**[ viewID ] 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[ viewID ] and ci\_perspective\_center\_ver[ viewID ] are inferred to be equal to ci\_perspective\_center\_hor[ 0 ] and ci\_perspective\_center\_ver[ 0 ], respectively.

**ci\_ortho\_width**[ viewID ] and **ci\_ortho\_height**[ viewID ] are positive floating-point values that specify in scene units (e.g. meters) the horizontal and vertical dimensions of the captured part of the volumetric frame. When not present and mode equal to 0, the values of ci\_ortho\_width[ viewID ] and ci\_ortho\_height[ viewID ] are inferred to be equal to ci\_ortho\_width[ 0 ] and ci\_ortho\_height[ 0 ], respectively.

##### Depth quantization semantics

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

**dq\_norm\_disp\_low**[ viewID ] and **dq\_norm\_disp\_high**[ viewID ] specify the minimum and maximum normalized disparity values, respectively, in scene units-1 (e.g. meters-1) of the view with view ID equal to viewID.

**dq\_depth\_occ\_threshold\_default**[ viewID ] specifies the default occupancy threshold used in the occupancy value extraction process of the view with view ID equal to viewID. When not present the value of dq\_depth\_occ\_threshold\_default[ viewID ] is inferred to be equal to 0.

##### Pruning parents semantics

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

**pp\_num\_parents\_minus1**[ viewID ] plus 1 specifies the number of parents of the view with view ID equal to viewID in the pruning graph at encoder stage. pp\_num\_parents\_minus1[ viewID ] shall be in the range 0 to mvp\_num\_views\_minus1, exclusive.

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

#### Patch data unit MIV extension semantics

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

**pdu\_depth\_occ\_threshold**[ tileID ][ p ] specifies the threshold below which the occupancy value is defined to be unoccupied for the patch with index equal to p, in the tile with ID equal to tileID. 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\_bit\_depth\_minus1 + 1. When not present, the value of pdu\_depth\_occ\_threshold[ tileID ][ p ] is inferred to be equal to dq\_depth\_occ\_threshold\_default[ pdu\_projection\_id[ tileID ][ p ] ].

[Ed. (LK) the marked sentence should be remove and pdu\_projection\_id[ tileID ][ p ] could be replaced by viewID. Ed. (JB) viewID doesn’t seem to be defined at this stage, so can’t just be replaced.]

**pdu\_attribute\_offset**[ tileID ][ p ][ c ] specifies an offset applied to the c-th component sample values of the attribute for the patch with index equal to p, in the tile with ID equal to tileID.. The number of bits used to represent pdu\_attribute\_offset[ tileID ][ p ][ c ] is equal to asps\_patch\_attribute\_offset\_bit\_depth\_minus1+ 1. When not present, the value of pdu\_attribute\_offset[ tileID ][ p ][ c ] is inferred to be equal to 0.

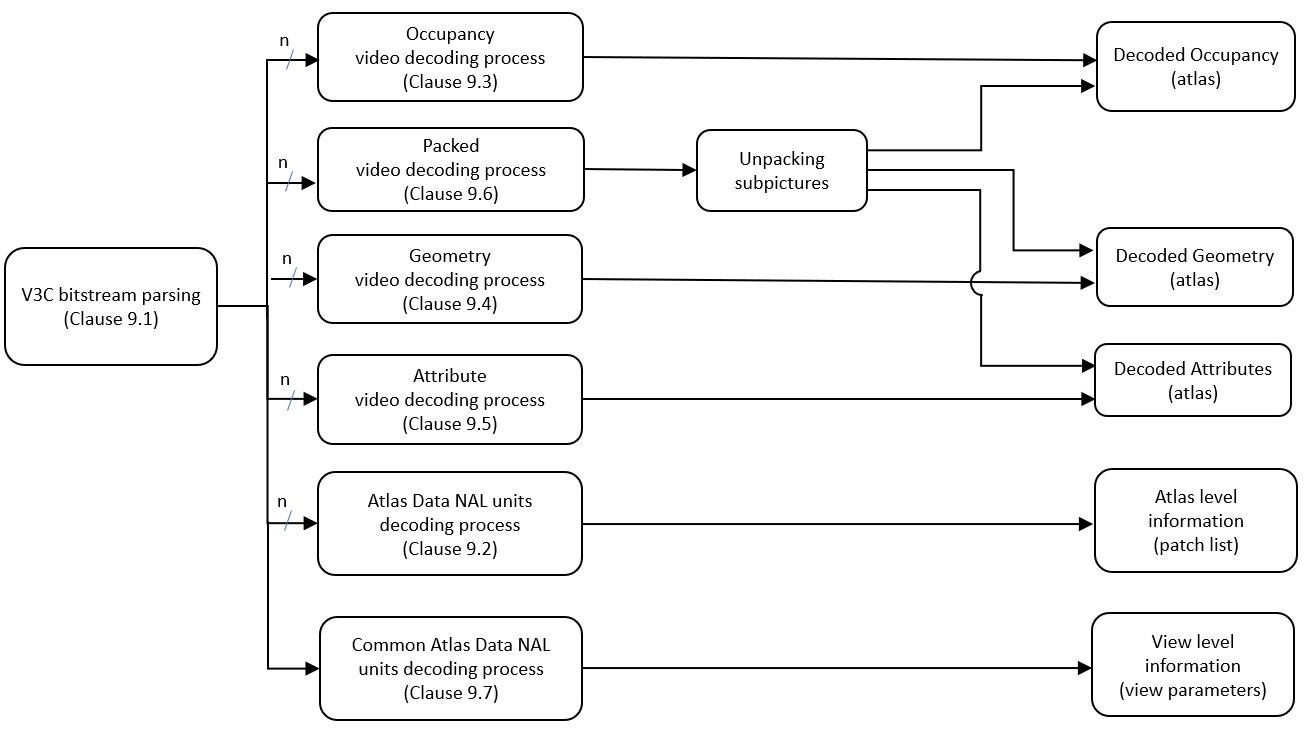
### Order of V3C units and association to coded information

An MIV bitstream consists of a series of coded MIV sequences. An 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.

An 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.

# Decoding process

## General decoding process



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

[Ed. (LK) This subclause should be inherited from part 5 and extended by the MIV specific information]

The block diagram of the decoding process is illustrated in **Figure 3**.

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 atlas ID decAtlasID equal to vuh\_atlas\_id or determined through external means if the V3C unit header is unavailable,, the following:

* zero or one decoded occupancy picture, the 2D array DecOccFrames[ frameIdx ][ 0 ], of size AsmeOccupancyFrameHeight[ decAtlasID ] × AsmeOccupancyFrameWidth[ decAtlasID ],
* zero or one decoded geometry picture, the 2D array DecGeoFrames[ 0 ][ frameIdx ][ 0 ], of size AsmeGeometryFrameHeight[ decAtlasID ] × AsmeGeometryFrameWidth[ decAtlasID ],
* zero or one texture attribute picture, the 3D array DecAttrFrames[ 0 ][ 0 ][ 0 ][ frameIdx ][ c ] of size AspsFrameHeight[ decAtlasID ] × AspsFrameWidth[ decAtlasID ], with c in 0 .. 2,
* a 2D array AtlasBlockToPatchMap of size ( AspsFrameHeight[ decAtlasID ] / PatchPackingBlockSize[ decAtlasID ] ) × ( AspsFrameWidth[ decAtlasID ] / PatchPackingBlockSize[ decAtlasID ]),
* 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 subclause 9.3, 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 subclause 9.4, 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 subclause 9.5, is invoked for the texture attribute sub-bitstream.
4. If present, the packed sub-bitstream is extracted from the V3C sequence.
   1. The packed video decoding process, as specified in subclause 9.6, is invoked for the packed sub-bitstream.
5. If present, the common atlas data sub-bitstream is extracted from the V3C sequence.
   1. The common atlas data decoding process, as specified in subclause 9.7, is invoked for the common atlas data sub-bitstream.
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 the following is repeatedly invoked:
      1. The sub-bitstream extraction process in 9.8 is invoked with targetAtlasId set equal to decAtlasID, 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 2D array DecOccFrames[ frameIdx ][ 0 ].
      3. The decoded geometry picture with the same value of vuh\_atlas\_id for the access unit, if present, is output, as 2D array DecGeoFrames[ 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 3D array DecAttrFrames[ 0 ][ 0 ][ 0 ][ frameIdx ][ c ], with c 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 subclause 9.2, is invoked for the current coded atlas frame, with the 2D array AtlasBlockToPatchMap as output.

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 decAtlasID equal to targetAtlasId, with the 2D array AtlasBlockToPatchMap as output.

## Atlas data decoding process

### General atlas data decoding process

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

### Decoding process for a coded atlas frame

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

### Atlas NAL unit decoding process

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

### Atlas tile header decoding process

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

### Decoding process for patch data units

#### General decoding process for patch data units

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply with the following modifications and additions:

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

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

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

These additional variables are initially set as follows:

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

#### Decoding process for patch data units coded in intra mode

In addition to the variables derived in the corresponding subclause of ISO/IEC CD 23090-5(2E) the variables TilePatchEntityID[ tileID ][ p ], TilePatchDepthOccThreshold[ tileID ][ p ], TilePatchAttributeOffset[ tileID ][ p ][ c ], for patch with index p in the atlas tile with ID tileID, are derived as follows:

If vme\_max\_entity\_id is not equal to 0, the following applies:

TilePatchEntityID[ tileID ][ p ] = pdu\_entity\_id[ tileID ][ p ] (18)

If asme\_depth\_occ\_threshold\_flag is equal to 1, the following applies:

TilePatchDepthOccThreshold[ tileID ][ p ] = pdu\_depth\_occ\_threshold[ tileID ][ p ] (19)

If asme\_patch\_attribute\_offset\_enabled\_flag is equal to 1, the following applies:

for( c = 0; c < 3; c++ )   
 TilePatchAttributeOffset[ tileID ][ p ][ c ] = pdu\_attribute\_offset[ tileId ][ p ][ c ] <<  
 ( ai\_attribute\_2d\_bit\_depth\_minus\_1 – asme\_pdu\_offset\_bit\_count\_minus1 ) –   
 ( 1 << ai\_attribute\_2d\_bit\_depth\_minus\_1 ) (20)

[Ed. (LK) MIV does not use subclasues 9.2.5.3 to 9.2.5.7, so probably we do not have to define anything here. Maybe some section should be “does not apply”]

[Ed. (BC) merge, inter, RAW and EOM clearly “do not apply” but perhaps we should keep “skip” applying?]

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

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

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

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

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

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

#### Decoding process for patch data units coded in RAW mode

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

#### Decoding process for patch data units coded in EOM mode

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

### Decoding process of the block to patch map

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

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

#### General

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

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

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

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

##### General

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

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

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

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

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply with the following modifications and additions:

The process ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch ) is defined as follows:

ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch ) {  
 blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )  
 if( AtlasPatchType[ atlasPatchIdx ] == PROJECTED ) {  
 AtlasTotalNumProjPatches += 1  
 AtlasPatchEntityID[ atlasPatchIdx ] = TilePatchEntityID[ tileID ][ p ]  
 AtlasPatchDepthOccThreshold[ atlasPatchIdx ] = TilePatchDepthOccThreshold[ tileID ][ p ]  
 for( c = 0; c < 3; c++ ) {  
 AtlasPatchAttributeOffset[ atlasPatchIdx ][ c ] =   
 TilePatchAttributeOffset[ tileID ][ p ][ c ]  
 }  
 }  
  
}

## Occupancy video decoding process

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

## Geometry video decoding process

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

## Attribute video decoding process

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

## Packed video decoding process

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

## Common atlas data decoding process

### General common atlas data decoding process

The specifications in ISO/IEC CD 23090-5(2E):9.7.1 apply with following additions.

Outputs of this process are:

* for each decoded common atlas frame, the corresponding upper-case variables from subclauses 9.7.5

### Decoding process for a coded common atlas frame

The specifications in ISO/IEC CD 23090-5(2E):9.7.2 apply with following additions.

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

1. The processes in subclause 9.7.5 specify the common atlas frame decoding processes:

* Initializing of view information is specified in subclause 9.7.5.2.
* Updating of extrinsic information is specified in subclause 9.7.5.3.
* Updating of intrinsic information is specified in subclause 9.7.5.4.
* Updating of depth quantization information is specified in subclause 9.7.5.5.

### Common atlas NAL unit decoding process

The specifications in ISO/IEC CD 23090-5(2E):9.7.3 apply

### Common atlas frame order count derivation process

The specifications in ISO/IEC CD 23090-5(2E):9.7.4 apply.

### Common atlas frame MIV extension decoding process

#### General decoding process for view parameters

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

More specifically the following arrays are derived:

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

ViewEnabledInAtlas[ atlasID ][ viewID ] specifies that atlas with atlas ID equal to atlas ID contains patches than may contain view with view ID equal to viewID.

ViewCompleteInAtlas[ atlasID ][ viewID ] specifies that atlas with atlas ID equal to atlas ID contains complete view with view ID equal to viewID.

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

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

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

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

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

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

ViewType[ viewID ] specifies the projection method of the view with view ID equal to viewID.

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

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

ViewErpPhiMin[ viewID ] specifies the minimum longitude range for an ERP projection, in units of degrees, of the view with view ID equal to viewID. ViewErpPhiMin[ viewID ] shall be in the range −180 to 180 in the spherical coordinate system

ViewErpPhiMax[ viewID ] specifies the maximum longitude range for an ERP projection, in units of degrees, of the view with view ID equal to viewID. ViewErpPhiMax[ viewID ] shall be in the range −180 to 180 in the spherical coordinate system

ViewErpThetaMin[ viewID ] specifies the minimum latitude range for an ERP projection, in units of degrees, of the view with view ID equal to viewID. ViewErpThetaMin[ viewID ] shall be in the range −90 to 90 in the spherical coordinate system.

ViewErpThetaMax[ viewID ] specifies the maximum latitude range for an ERP projection, in units of degrees, of the view with view ID equal to viewID. ViewErpThetaMax [ viewID ] shall be in the range −90 to 90 in the spherical coordinate system.

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

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

ViewPerspectiveCenterHor[ viewID ] specifies in luma sample positions the horizontal coordinates of the principal point of a perspective projection of the view with view ID equal to viewID.

ViewPerspectiveCenterVer[ viewID ] specifies in luma sample positions the vertical coordinates of the principal point of a perspective projection of the view with view ID equal to viewID.

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

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

ViewQuantizationLaw[ viewID ] specifies the type of depth quantization method of the view with view ID equal to viewID.

ViewNormDispMin[ viewID ] specifies the minimum normalized disparity value of the view with view ID equal to viewID.

ViewNormDispMax[ viewID ] specifies the maximum normalized disparity value of the view with view ID equal to viewID.

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

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

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

ViewParentIndex[ viewID ] [ i ] specifies index of the i-th parent view for the view with view ID equal to viewID in the pruning graph at encoder stage. ViewParentIndex [ viewID ] [ i ] shall be in the range of 0 to NumViews – 1, inclusive, and shall not be equal to ViewIDtoIndex[viewID].

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

NumViews = 0  
 for( v = 0; v <= 65 535 ; v++ ) {  
 for( a = 0; a < 64; a++) {  
 ViewEnabledInAtlas[ a ][ v ] = 0  
 ViewCompleteInAtlas[ a ][ v ] = 0  
 }  
 ViewPosX[ v ] = 0  
 ViewPosY[ v ] = 0  
 ViewPosZ[ v ] = 0  
 ViewQuatX[ v ] = 0  
 ViewQuatY[ v ] = 0  
 ViewQuatZ[ v ] = 0  
 ViewType[ v ] = 0  
 ViewProjectionPlaneWidth[ v ] = 0  
 ViewProjectionPlaneHeight[ v ] = 0  
 ViewErpPhiMin[ v ] = 0  
 ViewErpPhiMax[ v ] = 0  
 ViewErpThetaMin[ v ] = 0  
 ViewErpThetaMax[ v ] = 0  
 ViewPerspectiveFocalHor[ v ] = 0  
 ViewPerspectiveFocalVer[ v ] = 0  
 ViewPerspectiveCenterHor[ v ] = 0  
 ViewPerspectiveCenterVer[ v ] = 0  
 ViewOrthoWidth[ v ] = 0  
 ViewOrthoHeight[ v ] = 0  
 ViewQuantizationLaw[ v ] = 0  
 ViewNormDispMin[ v ] = 0  
 ViewNormDispMax[ v ] = 0  
 ViewOccThreshold[ v ] = 0  
 ViewRoot[ v ] = 0  
 ViewNumParents[ v ] = 0  
 for( i = 0; i < pp\_num\_parents\_minus1[ v ]; i++) {   
 ViewParentIndex[ v ][ i ] = 0  
 }  
 }

If nal\_unit\_type of common atlas frame equal to NAL\_IRAP\_CAF, then the process for decoding in subclause 9.7.5.2 is used and the outputs of that process are used as the output.

If nal\_unit\_type of common atlas frame equal to NAL\_CAF and came\_update\_extrinsics\_flag is 1, then the process for decoding in subclause 9.7.5.3 is used and the outputs of that process are used as the output.

If nal\_unit\_type of common atlas frame equal to NAL\_CAF and came\_update\_intrinsics\_flag is equal to 1, then the process for decoding in subclause 9.7.5.4 is used and the outputs of that process are used as the output.

If nal\_unit\_type of common atlas frame equal to NAL\_CAF and came\_update\_depth\_quantization\_flag is equal 1, then the process for decoding in subclause 9.7.5.5 is used and the outputs of that process are used as the output.

#### Initializing view information process

The following view related arrays are assigned given the parsed elements in common atlas frame:

NumViews = mvp\_num\_views\_minus1 + 1

If mvp\_view\_enabled\_present\_flag is equal to 1, the view in atlas related variables are set as follows:

for( a = 0; a <= vps\_atlas\_count\_minus1; a++ ){  
 for( v = 0; v < NumViews ; v++ ){  
 atlasID = vps\_atlas\_id[ a ]  
 v = ViewIndexToID[ v ]  
   
 ViewEnabledInAtlas[ atlasID ][ v ] =  
  mvp\_view\_enabled\_in\_atlas\_flag[ atlasID ][ v ]   
 ViewCompleteInAtlas[ atlasID ][ v ] =   
 mvp\_view\_complete\_in\_atlas\_flag[ atlasID ][ v ]

The extrinsic related arrays are set as follows:

for( v = 0; v < NumViews ; v++ ){  
 v = ViewIndexToID[ v ]  
   
 ViewPosX[ viewID ] = ce\_view\_pos\_x[ viewID ]  
 ViewPosY[ viewID ] = ce\_view\_pos\_y[ viewID ]  
 ViewPosZ[ viewID ] = ce\_view\_pos\_z[ viewID ]  
 ViewQuatX[ viewID ] = ce\_view\_quat\_x[ viewID ] ÷ 214  
 ViewQuatY[ viewID ] = ce\_view\_quat\_y[ viewID ] ÷ 214  
 ViewQuatZ[ viewID ] = ce\_view\_quat\_z[ viewID ] ÷ 214  
 }

If mvp\_intrinsic\_params\_equal\_flag equal to 1, the intrinsic related arrays are set as follows:

initViewID = ViewIndexToID[ 0 ]  
   
 ViewType[ initViewID ] = ci\_cam\_type[ initViewID ]  
 ViewProjectionPlaneWidth[ initViewID ] = ci\_projection\_plane\_width\_minus1[ initViewID ] + 1  
 ViewProjectionPlaneHeight[ initViewID ] = ci\_projection\_plane\_height\_minus1[ initViewID ] +1  
 ViewErpPhiMin[ initViewID ] = ci\_erp\_phi\_min[ initViewID ]  
 ViewErpPhiMax[ initViewID ] = ci\_erp\_phi\_max[ initViewID ]  
 ViewErpThetaMin[ initViewID ] = ci\_erp\_theta\_min[ initViewID ]  
 ViewErpThetaMax[ initViewID ] = ci\_erp\_theta\_max[ initViewID ]  
 ViewPerspectiveFocalHor[ initViewID ] = ci\_perspective\_focal\_hor[ initViewID ]  
 ViewPerspectiveFocalVer[ initViewID ] = ci\_perspective\_focal\_ver[ initViewID ]  
 ViewPerspectiveCenterHor[ initViewID ] = ci\_perspective\_center\_hor[ initViewID ]  
 ViewPerspectiveCenterVer[ initViewID ] = ci\_perspective\_center\_ver[ initViewID ]  
 ViewOrthoWidth[ initViewID ] = ci\_ortho\_width[ initViewID ]  
 ViewOrthoHeight[ initViewID ] = ci\_ortho\_height[ initViewID ]  
   
 for( v = 1; v < NumViews; v++ ){  
 viewID = ViewIndexToID[ v ]  
   
 ViewType[ viewID ] = ViewType[ initViewID ]  
 ViewProjectionPlaneWidth[ viewID ] = ViewProjectionPlaneWidth[ initViewID ]   
 ViewProjectionPlaneHeight[ viewID ] = ViewProjectionPlaneHeight[ initViewID ]   
 ViewErpPhiMin[ viewID ] = ViewErpPhiMin[ initViewID ]   
 ViewErpPhiMax[ viewID ] = ViewErpPhiMax[ initViewID ]   
 ViewErpThetaMin[ viewID ] = ViewErpThetaMin[ initViewID ]  
 ViewErpThetaMax[ viewID ] = ViewErpThetaMax[ initViewID ]  
 ViewPerspectiveFocalHor[ viewID ] = ViewPerspectiveFocalHor[ initViewID ]   
 ViewPerspectiveFocalVer[ viewID ] = ViewPerspectiveFocalVer[ initViewID ]   
 ViewPerspectiveCenterHor[ viewID ] = ViewPerspectiveCenterHor[ initViewID ]   
 ViewPerspectiveCenterVer[ viewID ] = ViewPerspectiveCenterVer[ initViewID ]   
 ViewOrthoWidth[ viewID ] = ViewOrthoWidth[ initViewID ]   
 ViewOrthoHeight[ viewID ] = ViewOrthoHeight[ initViewID ]   
 }

Otherwise, when mvp\_intrinsic\_params\_equal\_flag equal to 0, the intrinsic related arrays are set as follows:

for( v = 0; v < NumViews; v++ ){  
 viewID = ViewIndexToID[ v ]  
   
 ViewType[ viewID ] = ci\_cam\_type[ viewID ]  
 ViewProjectionPlaneWidth[ viewID ] = ci\_projection\_plane\_width\_minus1[ viewID ] + 1  
 ViewProjectionPlaneHeight[ viewID ] = ci\_projection\_plane\_height\_minus1[ viewID ] +1  
 ViewErpPhiMin[ initViewID ] = ci\_erp\_phi\_min[ viewID ]  
 ViewErpPhiMax[ initViewID ] = ci\_erp\_phi\_max[ viewID ]  
 ViewErpThetaMin[ initViewID ] = ci\_erp\_theta\_min[ viewID ]  
 ViewErpThetaMax[ initViewID ] = ci\_erp\_theta\_max[ viewID ]  
 ViewPerspectiveFocalHor[ initViewID ] = ci\_perspective\_focal\_hor[ viewID ]  
 ViewPerspectiveFocalVer[ initViewID ] = ci\_perspective\_focal\_ver[ viewID ]  
 ViewPerspectiveCenterHor[ initViewID ] = ci\_perspective\_center\_hor[ viewID ]  
 ViewPerspectiveCenterVer[ initViewID ] = ci\_perspective\_center\_ver[ viewID ]  
 ViewOrthoWidth[ initViewID ] = ci\_ortho\_width[ viewID ]  
 ViewOrthoHeight[ initViewID ] = ci\_ortho\_height[ viewID ]  
 }

If mvp\_depth\_quantization\_params\_equal\_flag equal to 1, the depth quantization related arrays are set as follows:

initViewID = ViewIndexToID[ 0 ]  
   
 ViewQuantizationLaw[ initViewID ] = dq\_quantization\_law[ initViewID ]  
 ViewNormDispMin[ initViewID ] = dq\_norm\_disp\_low[ initViewID ]  
 ViewNormDispMax[ initViewID ] = dq\_norm\_disp\_high[ initViewID ]  
 ViewOccThreshold[ initViewID ] = dq\_depth\_occ\_threshold\_default[ initViewID ]  
   
 for( v = 1; v < NumViews; v++ ){  
 viewID = ViewIndexToID[ v ]  
   
 ViewQuantizationLaw[ viewID ] = ViewQuantizationLaw[ initViewID ]   
 ViewNormDispMin[ viewID ] = ViewNormDispMin[ initViewID ]   
 ViewNormDispMax[ viewID ] = ViewNormDispMax[ initViewID ]   
 ViewOccThreshold[ viewID ] = ViewOccThreshold[ initViewID ]   
 }  
  
 Otherwise, when mvp\_depth\_quantization\_params\_equal\_flag equal to 0, the depth quantization related arrays are set as follows:

for( v = 1; v < NumViews ; v++ ){  
 viewID = ViewIndexToID[ v ]  
   
 ViewQuantizationLaw[ viewID ] = dq\_quantization\_law[ viewID ]  
 ViewNormDispMin[ viewID ] = dq\_norm\_disp\_low[ viewID ]  
 ViewNormDispMax[ viewID ] = dq\_norm\_disp\_high[ viewID ]  
 ViewOccThreshold[ viewID ] = dq\_depth\_occ\_threshold\_default[ viewID ]  
 }

If mvp\_pruning\_graph\_params\_present\_flag equal to 1, the pruning related arrays are set as follows:

for( v = 0; v < NumViews; v++ ){  
 viewID = ViewIndexToID[ v ]  
 ViewRoot[ viewID ] = pp\_is\_root\_flag[ viewID ]  
 ViewNumParents[ viewID ] = pp\_num\_parents\_minus1[ viewID ] + 1  
 for( i = 0; i < ViewNumParents[ viewID ] ; i++ ){  
 ViewParentIndex[ viewID ][ i ] = pp\_parent\_idx[ viewID ][ i ]  
 }  
 }

#### Updating extrinsic information process

The extrinsic related arrays are updated as follows:

for( i = 0; i < mvpue\_num\_views\_updates\_minus1 + 1; i++ ){  
 viewID = mvpue\_view\_idx[ i ]  
  
 ViewPosX[ viewID ] = ce\_view\_pos\_x[ viewID ]  
 ViewPosY[ viewID ] = ce\_view\_pos\_y[ viewID ]  
 ViewPosZ[ viewID ] = ce\_view\_pos\_z[ viewID ]  
 ViewQuatX[ viewID ] = ce\_view\_quat\_x[ viewID ] ÷ 214  
 ViewQuatY[ viewID ] = ce\_view\_quat\_y[ viewID ] ÷ 214  
 ViewQuatZ[ viewID ] = ce\_view\_quat\_z[ viewID ] ÷ 214  
 }

#### Updating intrinsic information process

The intrinsic related arrays are updated as follows:

for( i = 0; i < mvpui\_num\_view\_updates\_minus1 + 1; i++ ){  
 viewID = mvpui\_view\_id[ i ]  
  
 ViewType[ viewID ] = ci\_cam\_type[ viewID ]  
 ViewProjectionPlaneWidth[ viewID ] = ci\_projection\_plane\_width\_minus1[ viewID ] + 1  
 ViewProjectionPlaneHeight[ viewID ] = ci\_projection\_plane\_height\_minus1[ viewID ] +1  
 ViewErpPhiMin[ viewID ] = ci\_erp\_phi\_min[ viewID ]  
 ViewErpPhiMax[ viewID ] = ci\_erp\_phi\_max[ viewID ]  
 ViewErpThetaMin[ viewID ] = ci\_erp\_theta\_min[ viewID ]  
 ViewErpThetaMax[ viewID ] = ci\_erp\_theta\_max[ viewID ]  
 ViewPerspectiveFocalHor[ viewID ] = ci\_perspective\_focal\_hor[ viewID ]  
 ViewPerspectiveFocalVer[ viewID ] = ci\_perspective\_focal\_ver[ viewID ]  
 ViewPerspectiveCenterHor[ viewID ] = ci\_perspective\_center\_hor[ viewID ]  
 ViewPerspectiveCenterVer[ viewID ] = ci\_perspective\_center\_ver[ viewID ]  
 ViewOrthoWidth[ viewID ] = ci\_ortho\_width[ viewID ]  
 ViewOrthoHeight[ viewID ] = ci\_ortho\_height[ viewID ]  
 }

#### Updating depth quantization information process

The depth quantization related arrays are updated as follows:

for( i = 0; i < mvpudq\_num\_view\_updates\_minus1; i++ ){  
 viewID = mvpudq\_view\_id[ i ]  
  
 ViewQuantizationLaw[ viewID ] = dq\_quantization\_law[ viewID ]  
 ViewNormDispMin[ viewID ] = dq\_norm\_disp\_low[ viewID ]  
 ViewNormDispMax[ viewID ] = dq\_norm\_disp\_high[ viewID ]  
 ViewOccThreshold[ viewID ] = dq\_depth\_occ\_threshold\_default[ viewID ]  
 }

## Sub-bitstream extraction process

[Ed. LK: If there is anything MIV specific, then it should be as addition in subclauses 9.8.1-9.8.3

Old text:

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 subclause shall be a conforming bitstream:

The output sub-bitstream is derived as follows:

– Remove all V3C units with vuh\_unit\_type not equal to V3C\_CAD and vuh\_atlas\_id not equal to targetAtlasId.

]

### General

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

### V3C unit extraction

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

[Ed. (JB): The corresponding part 5 clause doesn’t handle V3C\_CAD, which should be fixed here or there.]

### NAL unit extraction process

The specifications in the corresponding subclause of ISO/IEC CD 23090-5(2E) apply.

# Pre-reconstruction process

The specifications in the corresponding clause of ISO/IEC CD 23090-5(2E) do not apply.

# Reconstruction process

The specifications in the corresponding clause of ISO/IEC CD 23090-5(2E) apply with following additions.

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

# Post-reconstruction process

The specifications in the corresponding clause of ISO/IEC CD 23090-5(2E) do not apply.

# Adaptation process

The specifications in the corresponding clause of ISO/IEC CD 23090-5(2E) do not apply.

# Parsing process

The specifications in the corresponding clause of ISO/IEC CD 23090-5(2E) and its subclauses apply.

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

The specifications in ISO/IEC CD 23090-5(2E) subclause A.1 apply.

* 1. Profile, tier and level structure

The specifications in ISO/IEC CD 23090-5(2E) subclause A.2 apply.

* 1. CodecGroup profile components

The specifications in ISO/IEC CD 23090-5(2E) subclause A.3 apply.

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

* 1. Toolset profile components
     1. MIV Main, MIV Extended, and MIV Geometry Absent toolset profile component

V3C toolset profile components indicating the MIV Main (ptl\_profile\_toolset\_idc = 64), MIV Extended (ptl\_profile\_toolset\_idc = 65), or MIV Geometry Absent (ptl\_profile\_toolset\_idc = 66) toolset profile component shall conform to the syntax element restrictions specified in **Table A-1**. If a syntax element is not mentioned in **Table A-1** it is not restricted through a toolset profile component.

[Ed. (BC) casps\_common\_atlas\_sequence\_parameter\_set\_id and caf\_common\_atlas\_sequence\_parameter\_ set\_id are u(4) in part 5, and not ue(v) as the other ones in the table, so no restriction for them here]

**Table A-1: Allowable values of syntax element values for the MIV toolset profile components**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Profile name** | | |
| **Syntax element** | **MIV Main** | **MIV Extended** | **MIV Geometry Absent** |
| vuh\_unit\_type | V3C\_VPS, V3C\_AD, V3C\_GVD, V3C\_AVD, or V3C\_CAD | V3C\_VPS, V3C\_AD, V3C\_OVD, V3C\_GVD, V3C\_AVD, V3C\_PVD, or V3C CAD | V3C\_VPS, V3C\_AD, V3C\_OVD, V3C\_GVD, V3C\_AVD, V3C\_PVD, or V3C CAD |
| ptl\_profile\_toolset\_idc | 64 | 65 | 66 |
| ptl\_profile\_reconstruction\_idc | 255 | 255 | 255 |
| vps\_miv\_extension\_present\_flag | 1 | 1 | 1 |
| vps\_packing\_information\_present\_flag | 0 | 0, 1 | 0, 1 |
| vps\_map\_count\_minus1[ atlasID ] | 0 | 0 | 0 |
| vps\_occupancy\_video\_present\_flag[ atlasID ] | 0 | 0, 1 | 0 |
| vps\_geometry\_video\_present\_flag[ atlasID ] | 1 | 0, 1 | 0 |
| vme\_embedded\_occupancy\_enabled\_flag | 1 | 0, 1 | 0 |
| gi\_geometry\_MSB\_align\_flag[ atlasID ] | 0 | 0 | 0 |
| ai\_attribute\_count[ atlasID ] | 0, 1 | 0, 1 | 0, 1 |
| ai\_attribute\_type\_id[ atlasID ][ 0 ] | ATTR\_TEXTURE | ATTR\_TEXTURE | ATTR\_TEXTURE |
| ai\_attribute\_dimension\_minus1[ atlasID ][ 0 ] | 2 | 2 | 2 |
| ai\_attribute\_dimension\_partitions\_minus1[ atlasID ][ 0 ] | 0 | 0 | 0 |
| ai\_attribute\_MSB\_align\_flag[ atlasID ][ 0 ] | 0 | 0 | 0 |
| asps\_long\_term\_ref\_atlas\_frames\_flag | 0 | 0 | 0 |
| asps\_pixel\_deinterleaving\_enabled\_flag | 0 | 0 | 0 |
| asps\_patch\_precedence\_order\_flag | 0 | 0 | 0 |
| asps\_raw\_patch\_enabled\_flag | 0 | 0 | 0 |
| asps\_eom\_patch\_enabled\_flag | 0 | 0 | 0 |
| asps\_plr\_enabled\_flag | 0 | 0 | 0 |
| asme\_patch\_constant\_depth\_flag | 0 | 0, 1 | 0, 1 |
| vps\_video\_geometry\_present\_flag[ atlasID ] || asme\_patch\_constant\_depth\_flag | 1 | 1 | 0, 1 |
| vps\_packed\_video\_present\_flag[ atlasID ] | 0 | 0, 1 | 0, 1 |
| afps\_lod\_mode\_enabled\_flag | 0 | 0 | 0 |
| afps\_raw\_3d\_pos\_bit\_count\_explicit\_mode\_flag | 0 | 0 | 0 |
| afti\_single\_tile\_in\_atlas\_frame\_flag | 1 | 0, 1 | 0, 1 |
| ath\_type | I\_TILE | I\_TILE | I\_TILE |
| atdu\_patch\_mode[ tileID ][ patchIdx ] | I\_INTRA | I\_INTRA | I\_INTRA |
| asps\_atlas\_sequence\_parameter\_set\_id | 0…63, inclusive | 0…63, inclusive | 0…63, inclusive |
| afps\_atlas\_frame\_parameter\_set\_id | 0…63, inclusive | 0…63, inclusive | 0…63, inclusive |
| afps\_atlas\_sequence\_parameter\_set\_id | 0…63, inclusive | 0…63, inclusive | 0…63, inclusive |
| aaps\_atlas\_adaptation\_parameter\_set\_id | 0…63, inclusive | 0…63, inclusive | 0…63, inclusive |
| ath\_atlas\_frame\_parameter\_set\_id | 0…63, inclusive | 0…63, inclusive | 0…63, inclusive |
| ath\_atlas\_adaptation\_parameter\_set\_id | 0…63, inclusive | 0…63, inclusive | 0…63, inclusive |

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

* For bitstreams conforming to the MIV Extended toolset profile component, in the decoding process of an occupancy video component, associated with the atlas with atlas ID DecAtlasID, DecOccWidth[ frameIdx ] shall be equal to AsmeOccupancyFrameWidth[ DecAtlasID ] and DecOccHeight[ frameIdx ] shall be equal to AsmeOccupancyFrameHeight[ DecAtlasID ].
* For bitstreams conforming to the MIV Main or the MIV Extended toolset profile component, in the decoding process of a geometry video component, associated with the atlas with atlas ID DecAtlasID, DecGeoWidth[ frameIdx ] shall be equal to AsmeGeometryFrameWidth[ DecAtlasID ] and DecGeoHeight[ frameIdx ] shall be equal to AsmeGeometryFrameHeight[ DecAtlasID ].
* For bitstreams conforming to the MIV Main, the MIV Extended, or the MIV Geometry Absent toolset profile component, in the decoding process of an attribute video component, associated with the atlas with atlas ID DecAtlasID, DecAttrWidth[ 0 ][ 0 ][ 0 ][ frameIdx ] shall be equal to AspsFrameWidth[ DecAtlasID ] and DecAttrHeight[ 0 ][ 0 ][ 0 ][ frameIdx ] shall be equal to AspsFrameHeight[ DecAtlasID ].
  1. Reconstruction profile components

The specifications in ISO/IEC CD 23090-5(2E) subclause A.5 apply.

* 1. Tiers and Levels

The specifications in ISO/IEC CD 23090-5(2E) subclause A.6 apply.

* 1. Decoder instantiations

The specifications in ISO/IEC CD 23090-5(2E) subclause A.7 apply.

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 ISO/IEC CD 23090-5(2E) subclause B.2 do not apply.

* 1. Conversion operations

The specifications in ISO/IEC CD 23090-5(2E) subclauseB.3 do not apply.

* 1. Unpacking process of a decoded packed video

The specifications in ISO/IEC CD 23090-5(2E)clause B.4 apply.

1. (informative)  
   V3C sample stream format

The specifications in ISO/IEC CD 23090-5(2E) Annex C apply.

1. (normative)  
   NAL sample stream format

The specifications in ISO/IEC CD 23090-5(2E) Annex D apply.

1. (normative)  
   Atlas hypothetical reference decoder

The specifications in ISO/IEC CD 23090-5(2E) Annex E apply.

1. (normative)  
   Supplemental enhancement information
   1. General

The specifications in ISO/IEC CD 23090-5(2E) Annex F.1 apply with the following additions:

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

|  |  |  |
| --- | --- | --- |
| **SEI message** | **NAL Type** | **Conformance Type** |
| Viewing space | NAL\_PREFIX\_NSEI | N/A |
| Viewing space handling | NAL\_PREFIX\_NSEI | N/A |
| Geometry upscaling parameters | NAL\_PREFIX\_NSEI | N/A |

* 1. SEI payload syntax

The specifications ISO/IEC CD 23090-5(2E) Annex F.2 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 ] | i(16) |
| **es\_primitive\_shape\_quat\_y**[ e ][ s ] | i(16) |
| **es\_primitive\_shape\_quat\_z**[ e ][ s ] | i(16) |
| } |  |
| } |  |
| if( es\_viewing\_direction\_constraint\_present\_flag[ e ] ) { |  |
| if( es\_guard\_band\_present\_flag[ e ] ) |  |
| **es\_guard\_band\_direction\_size**[ e ][ s ] | fl(16) |
| if( !es\_camera\_inferred\_flag[ e ] ) { |  |
| **es\_primitive\_shape\_viewing\_direction\_quat\_x\_center**[ e ][ s ] | i(16) |
| **es\_primitive\_shape\_viewing\_direction\_quat\_y\_center**[ e ][ s ] | i(16) |
| **es\_primitive\_shape\_viewing\_direction\_quat\_z\_center**[ e ][ s ] | i(16) |
| } |  |
| **es\_primitive\_shape\_viewing\_direction\_yaw\_range**[ e ][ s ] | fl(16) |
| **es\_primitive\_shape\_viewing\_direction\_pitch\_range**[ e ][ s ] | fl(16) |
| } |  |
| } |  |
| } |  |

* + - 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. Viewing space handling SEI payload syntax

|  |  |
| --- | --- |
| viewing\_space\_handling( payloadSize ) { | **Descriptor** |
| **vs\_handling\_options\_count** | ue(v) |
| for( h = 0; h <= vs\_handling\_options\_count; h++ ) { |  |
| **vs\_handling\_device\_class**[ h ] | u(6) |
| **vs\_handling\_application\_class**[ h ] | u(6) |
| **vs\_handling\_method**[ h ] | u(6) |
| } |  |
| } |  |

* + 1. Geometry upscaling parameters SEI payload syntax

|  |  |
| --- | --- |
| geometry\_upscaling\_parameters( payloadSize ) { | **Descriptor** |
| **gup\_type** | ue(v) |
| if( gup\_type == 0 ) { |  |
| **gup\_erode\_threshold** | fl(16) |
| **gup\_delta\_threshold** | ue(v) |
| **gup\_max\_curvature** | u(3) |
| } |  |
| } |  |

* 1. SEI payload semantics

The specifications in subclause F.3 of ISO/IEC CD 23090-5(2E) Annex F and its subclauses apply, with the following additions and modifications.

* + 1. General SEI payload semantics

The specifications in ISO/IEC CD 23090-5(2E) Annex F.3.1 apply with the following additions:

|  |  |
| --- | --- |
| Table F-2 – Persistence scope of SEI messages | |
| SEI message | Persistence scope |
| Viewing space | The remainder of the bitstream or until a new viewing space SEI message |
| Viewing space handling | The remainder of the bitstream or until a new view space handling SEI message |
| Geometry upscaling parameters | The remainder of the bitstream or until a new geometry upscaling parameters SEI message |

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

**vs\_num\_elementary\_shapes\_minus1** plus 1 indicates the number of elementary shapes to build the viewing space.

* + - 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 primitive shapes in the list are interpolated along a path defined by the ordered centroids of the primitive shape.

When es\_primitive\_shape\_operation[ e ] is equal to 1, the operation is based on interpolation along the segment path defined by the centers of the successive primitive shapes in the ordered list of the syntax structure. The operation is based on regular metric distance RD( p, S ) of a point p related to a shape S center which has been shifted along the path. The shift value is a linear operation between regular distances RD( p, Ss ) and RD( p, Ss+1 ) to the two closest successive primitive shapes Ss and Ss+1. The interpolated elementary shapes are combined additively into the viewing space.

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

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

**es\_primitive\_orientation\_present\_flag**[ e ] equal to 1 specifies that per-primitive orientation information is present for each primitive shape in the elementary shape e. es\_primitive\_orientation\_present\_flag[ e ] equal to 0 specifies that per-primitive orientation information is not present, and that the primitives are axis-aligned.

**es\_viewing\_direction\_constraint\_present\_flag**[ e ]equal to 1 specifies that viewing direction constraints are present for each primitive shape in the elementary shape e. es\_viewing\_direction\_constraint\_present\_flag[ e ] equal to 0 specifies that per-primitive viewing direction constraints are not present.

**es\_camera\_inferred\_flag**[ e ] equal to 1 specifies that the positions and orientations of the primitive shapes are those of the views with indices es\_view\_idx[ e ][ s ] in the miv\_view\_params\_list( ).

**es\_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-3: primitive shape types**

|  |  |
| --- | --- |
| **es\_primitive\_shape\_type**[ e ][ s ] | **Shape** |
| 0 | cuboid primitive |
| 1 | spheroid primitive |
| 2 | halfspace primitive |
| 3 | other primitive |

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

**es\_guard\_band\_size**[ e ][ s ] 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\_flag[ e ] equal to 0 implies that the guard band size is implicitly 0. This parameter is expressed in same unit as the position parameter of the primitive shape. It is based on the signed distance which can be computed for each primitive shape, whatever the es\_primitive\_shape\_operation[ e ] is (CSG or interpolation). The guard band can be effectively treated as a second signed distance SD( p, S ) + guard band size that can be carried through the same operations to result at a final guard band distance SD( p, SGUARD )*.*

**es\_primitive\_shape\_quat\_x**[ e ][ s ] specifies the x component, qX[ e ][ s ], for the rotation to apply on the primitive shape s of the elementary shape e using the quaternion representation. The value of es\_primitive\_shape\_quat\_x[ e ][ s ] shall be in the range of − 214 to 214, inclusive. When es\_primitive\_shape\_quat\_x[ e ][ s ] is not present, its value shall be inferred to be equal to 0. The value of qX[ e ][ s ] is computed as follows:

qX[ e ][ s ] = es\_primitive\_shape\_quat\_x[ e ][ s ] ÷ 214 (F‑1)

**es\_primitive\_shape\_quat\_y**[ e ][ s ] specifies the y component, qY[ e ][ s ], for the rotation to apply on the primitive shape s of the elementary shape e using the quaternion representation. The value of es\_primitive\_shape\_quat\_y[ e ][ s ] shall be in the range of − 214 to 214, inclusive. When es\_primitive\_shape\_quat\_y[ e ][ s ] is not present, its value shall be inferred to be equal to 0. The value of qY[ e ][ s ] is computed as follows:

qY[ e ][ s ] = es\_primitive\_shape\_quat\_y[ e ][ s ] ÷ 214 (F‑2)

**es\_primitive\_shape\_quat\_z**[ e ][ s ] specifies the y component, qZ[ e ][ s ], for the rotation to apply on the primitive shape s of the elementary shape e using the quaternion representation. The value of es\_primitive\_shape\_quat\_z[ e ][ s ] shall be in the range of − 214 to 214, inclusive. When es\_primitive\_shape\_quat\_z[ e ][ s ] is not present, its value shall be inferred to be equal to 0. The value of qZ[ e ][ s ] is computed as follows:

qZ[ e ][ s ] = es\_primitive\_shape\_quat\_z[ e ][ s ] ÷ 214(F‑3)

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

It is a requirement of bitstream conformance that:

qX[ e ][ s ]2 + qY[ e ][ s ]2 +qZ[ e ][ s ]2 <= 1(F‑4)

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

qW[ e ][ s ] = Sqrt( 1 – ( qX[ e ][ s ]2 + qY[ e ][ s ]2 + qZ[ e ][ s ]2 ) )(F‑5)

**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\_flag[ e ] equal to 0 implies that the guard band directional\_size is implicitly 0. This parameter is expressed in degree.

**es\_primitive\_shape\_viewing\_direction\_quat\_x\_center**[ e ][ s ] specifies the x component, qXc[ e ][ s ], for the suggested viewing directions center for the primitive shape s of the elementary shape e using the quaternion representation. The value of es\_primitive\_shape\_viewing\_direction\_quat\_x\_center[ e ][ s ] shall be in the range of − 214 to 214, inclusive. When es\_primitive\_shape\_viewing\_direction\_quat\_x\_center[ e ][ s ] is not present, its value shall be inferred to be equal to 0. The value of qXc[ e ][ s ] is computed as follows:

qXc[ e ][ s ] = es\_primitive\_shape\_viewing\_direction\_quat\_x\_center [ e ][ s ] ÷ 214(F‑6)

**es\_primitive\_shape\_viewing\_direction\_quat\_y\_center**[ e ][ s ] specifies the y component, qYc[ e ][ s ], for the suggested viewing directions center for the primitive shape s of the elementary shape e using the quaternion representation. The value of es\_primitive\_shape\_viewing\_direction\_quat\_y\_center[ e ][ s ] shall be in the range of − 214 to 214, inclusive. When es\_primitive\_shape\_viewing\_direction\_quat\_y\_center[ e ][ s ] is not present, its value shall be inferred to be equal to 0. The value of qYc[ e ][ s ] is computed as follows:

qYc[ e ][ s ] = es\_primitive\_shape\_viewing\_direction\_quat\_y\_center [ e ][ s ] ÷ 214(F‑7)

**es\_primitive\_shape\_viewing\_direction\_quat\_z\_center**[ e ][ s ] specifies the z component, qZc[ e ][ s ], for the suggested viewing directions center for the primitive shape s of the elementary shape e using the quaternion representation. The value of es\_primitive\_shape\_viewing\_direction\_quat\_z\_center[ e ][ s ] shall be in the range of − 214 to 214, inclusive. When es\_primitive\_shape\_viewing\_direction\_quat\_z\_center[ e ][ s ] is not present, its value shall be inferred to be equal to 0. The value of qZc[ e ][ s ] is computed as follows:

qZc[ e ][ s ] = es\_primitive\_shape\_viewing\_direction\_quat\_z\_center [ e ][ s ] ÷ 214(F‑8)

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

It is a requirement of bitstream conformance that:

qXc[ e ][ s ]2 + qYc[ e ][ s ]2 +qZc[ e ][ s ]2 <= 1(F‑9)

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

qWc[ e ][ s ] = Sqrt( 1 – ( qXc[ e ][ s ]2 + qYc[ e ][ s ]2 + qZc[ e ][ s ]2 ) )(F‑10)

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

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

V( p, E ) = ∑-SD( p, Si ) \* Vi( Si ) / ∑-SD( p, Si ) (F‑11)

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

V( p, E ) = ( RD( p, Ss+1 ) \* Vs + RD( p, Ss ) \* Vs+1 ) / ( RD( p, S ) + RD( p, Ss+1 ) )(F‑12)

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

V( p ) = ∑-SD( p, Ei ) \* V( p, Ei ) / ∑-SD( p, Ei ) (F‑13)

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

position\_fading\_index( p ) =   
Clip3( 0, 1, ( SD( p ) + guard\_band\_size( p )) / guard\_band\_size( p ))(F‑14)

where p is the vector of coordinates of the viewport, SD( p) the global signed distance of p and guard\_band\_size( p ) the global guard band size value at the position p from Equation (F‑13).

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):

yaw\_fading\_index( p ) = Clip3( 0, 1,   
( abs( yaw( p ) – viewing\_yaw\_center( p ) ) – viewing\_yaw\_range( p ) +   
guard\_band\_direction\_size( p ) ) / guard\_band\_direction\_size( p ) )(F‑15)

where yaw( p ) is the yaw value of the viewport quaternion, viewing\_yaw\_center( p ) is the yaw value of the direction center quaternion, viewing\_yaw\_range( p ) is the viewing direction range in yaw, guard\_band\_direction\_size( p ) is the directional guard band size at that viewport position p from Equation (F‑13).

The global fading index which is applied on the viewport RGB components is given by the multiplication of position\_fading\_index, yaw\_fading\_index and pitch\_fading\_index.

* + - 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 ] are 16-bit floating-point values 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

SDCUBOID( p , l, h ) = Min( Max( dx, Max( dy, dz ) ), 0 ) + | Max( d, 0 ) |(F‑16)

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

* + - 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 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

SDSPHEROID( p , r ) = | p / r | \* ( | p / r | – 1 ) / | p / r2 |(F‑17)

where the 3D point p is as regards to the primitive center ( sp\_center\_x, sp\_center\_y, sp\_center\_z ),r equals the ( sp\_radius\_x, sp\_radius\_y, sp\_radius\_z )vector, and the division 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

DHALFSPACE( p, n, d ) = dot( p, n / | n | ) – d(F‑18)

where n is the normal vector given by ( hp\_normal\_x, hp\_normal\_y, hp\_normal\_z ) and d equals hp\_distance.

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

* + 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 ] within those 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 ] within those 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 ] within those specified in **Table F-4**. A conformant bitstream shall not have duplicate values for vs\_handling\_application\_class[ h ] within the same viewing\_space( ) structure. When vs\_handling\_application\_class[ h ] == VHAC\_ALL, then it shall hold that h + 1 == vs\_handling\_options\_count.

**Table F-4: Viewing space handling device classes**

|  |  |  |
| --- | --- | --- |
| **Value** | **Name** | **Description** |
| 0 | VHDC\_ALL | Match against all devices |
| 1 | VHDC\_HMD | Head-mounted display with 6DoF positioning |
| 2 | VHDC\_PHONE | Mobile phone or tablet with screen rendering depending on IMU |
| 3 | VHDC\_ASD | Autostereoscopic (lightfield) display |
| 4..31 | VHDC\_RSRVD\_5..VHDC\_RSRVD\_31 | Reserved for future use by ISO/IEC |
| 32..63 | VHDC\_UNSPCF\_32..VHDC\_UNSPCF\_63 | Unspecified (available for specification by other standards) |

**Table F-5: Viewing space handling application classes**

|  |  |  |
| --- | --- | --- |
| **Value** | **Name** | **Description** |
| 0 | VHAC\_ALL | Match against all applications |
| 1 | VHAC\_AR | The coded immersive video is used to augment the physical world |
| 2 | VHAC\_VR | The coded immersive video is used as a virtual reality |
| 3 | VHAC\_WEB | The coded immersive video is embedded within a website |
| 4 | VHAC\_SD | The coded immersive video is used as an element within a larger scene description |
| 5..31 | VHAC\_RSRV\_5..VHAC\_RSRV\_31 | Reserved for future use by ISO/IEC |
| 32..63 | VHAC\_UNSPF\_32..VHAC\_UNSPF\_63 | Unspecified (available for specification by other standards) |

**Table F-6: Viewing space handling methods**

|  |  |  |
| --- | --- | --- |
| **Value** | **Name** | **Description** |
| 0 | VHM\_NULL | Default client behavior |
| 1 | VHM\_RENDER | Always render, even when outside of the viewing space. This may cause rendering artifacts. |
| 2 | VHM\_FADE | When moving towards the outside of the viewing space, the scene fades to a default color. |
| 3 | VHM\_EXTRAP | Extrapolate content in an abstract low-frequent way that prevents rendering artifacts but preserves the general color tone of the scene. |
| 4 | VHM\_RESET | The viewer position and/or orientation is reset when the viewer reaches the limit of the viewing zone |
| 5 | VHM\_STRETCH | The scene rotates and translates along with the viewer to prevent the viewer from reaching the limit of the viewing zone |
| 6 | VHM\_ROTATE | The scene rotates with the viewer to keep the viewer within the field of view |
| 7..31 | VHM\_RSRV\_5..VHM\_RSRV\_31 | Reserved for future use by ISO/IEC |
| 32..63 | VHM\_UNSPF\_32..VHM\_UNSPF\_63 | Unspecified (available for specification by other standards) |

* + 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 document defines the value 0 in accordance with the geometry video scaling process in subclause (H.4). All positive even values are reserved for future use by ISO/IEC. All odd values are unspecified (available for specification by other standards).

**gup\_erode\_threshold** specifies the threshold that is applied in the texture aligned geometry erosion process (H.5.5) 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.5.5) 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.5.6) 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. (informative)  
   Volumetric usability information

The specifications in ISO/IEC CD 23090-5(2E) 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\_info\_present\_flag and vui\_anchor\_point\_present\_flag shall be equal to zero.

1. (Informative)  
   Overview of rendering processes
   1. General

The purpose of Annex H is to illustrate application of a decoded MIV sequence. It provides informative hypothetical transcoding and rendering processes, which operate on the outputs defined in clause 8, clause 9, Annex A, and Annex B. An application may combine multiple processes. Annex H does not form an integral part of this document.

[Ed.(BK): Check that all syntax elements in the MIV specification are used directly or indirectly in Annex H. Try to use them in a **clear** and **concise** way. Write in such a way that changes to Clause 9 and Annex B do not cause a lot of rewriting, especially by properly defining inputs and outputs. The goal is to get to DIS quality in an acceptable amount of time and effort.]

[Ed.(BK): This annex does not assume MIV Main profile. When possible with little text, processes are generalized.]

* The sample 3D reconstruction process (H.2) specifies how to reconstruct an atlas sample to a 3D point in space. This process is a building block of 6DoF rendering and any transcoding that involves 3D format conversion. The process demonstrates the use of most view parameters including camera intrinsics, camera extrinsics and depth quantization parameters. When applied to all atlas samples in a volumetric frame, a point cloud is reconstructed. Such a representation is useful for rendering applications and immersive video analytics.
* The entity-filtering process (H.3) specifies how to filter a block to patch map by entity ID prior to transcoding or rendering, for instance to render only the foreground objects in a scene.
* The geometry video scaling process (H.4) specifies how to upscale a geometry frame with preservation of thin foreground edges.
* The pruned view reconstruction process (H.6) specifies how to reconstruct a pruned view from multiple atlases. This process enables multiview rendering as opposed to rendering per point or per patch. The process is also useful for transcoding applications that work with views instead of atlases.

[Ed. Let’s insert a couple of scientific references and add these to the bibliography for multivew rendering]

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

An application may combine all or a subset of the processes, for instance:

1. Scale geometry (H.5)
2. Apply patch attribute offset process (H.4)
3. Reconstruct pruned views (H.6)
4. Determine view blending weights based on a viewport pose
5. Recover sample weights (H.7)
6. Reconstruct 3D points (H.2)
7. Project to a viewport
8. Fetch texture from multiple views
9. Blend texture contributions.

Alternatively, an application operating on a bitstream conforming to the geometry absent profile may:

1. Select a subset of views based on a viewport pose,
2. Apply patch attribute offset process (H.4)
3. Reconstruct pruned views (H.6)
4. Determine view blending weights based on a viewport pose
5. Recover sample weights (H.7)
6. Reconstruct geometry of the subset of views using an estimator
7. Reproject the subset of views to a viewport (H.2)
8. Fetch texture from multiple views
9. Blend texture contributions.
   1. Sample 3D reconstruction process

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

Inputs to this process are:

* the variables u and v of subclause 9.2.5.1 of ISO/IEC CD 23090-5(2E),
* when decoded (or unpacked) occupancy video data is present:
  + the variable occSample, the decoded (and interpolated) occupancy value of the sample,
* when decoded (or unpacked) geometry video data is present:
  + the variable geoSample, the decoded (and interpolated) geometry value of the sample,
* the variable patchIdx, the atlas patch index of the sample,

Outputs of this process are:

* the variable sampleOccFlag, indicating occupancy,
* when sampleOccFlag is equal to 1:
  + the variables sampleX, sampleY and sampleZ, corresponding to the Cartesian coordinates (x, y, z) of the point in scene coordinates.

This process has the form of a sequence of processes:

1. Sample occupancy reconstruction process (H.2.1),
2. When sampleOccFlag is equal to 1:
   1. Depth value clamping (H.2.2),
   2. Depth expansion (H.2.3), applying depth quantization parameters,
   3. Coordinate unprojection (H.2.4), applying camera intrinsics,
   4. Change of reference frame (H.2.5), applying camera extrinsics,
   5. Change of coordinate system, applying VUI.

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

* + 1. Sample occupancy reconstruction process

This process reconstructs the occupancy of an atlas sample.

Inputs to this process are:

* the variable patchIdx, corresponding to the atlas patch index of the atlas sample,
* the variable atlasID, corresponding to the atlas ID of the atlas sample,
* the variable occSample, the decoded (and interpolated) occupancy value of the atlas sample,
* the variable geoSample, the decoded (and interpolated) geometry value of the atlas sample,

Output of this process is:

* the variable sampleOccFlag, indicating sample occupancy.

The occupancy video reconstruction process operates as follows:

sampleOccFlag = 0  
 if( 0 <= patchIdx ) {  
 if( vps\_occupancy\_video\_present\_flag[ atlasID ] )   
 sampleOccFlag = occSample  
 elseif( vme\_embedded\_occupancy\_enabled\_flag ) {  
 if( AsmeDepthOccThresholdFlag[ atlasID ] )  
 threshold = AtlasPatchDepthOccThreshold[ patchIdx ]  
 else {  
 viewID = AtlasPatchProjectionID[ patchIdx ]  
 threshold = ViewOccThreshold[ viewID ]  
 }  
 if( threshold <= geoSample )  
 sampleOccFlag = 1  
 } else   
 sampleOccFlag = 1  
 }

* + 1. Depth clamping process

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

The inputs to this process are:

* the variable patchIdx, corresponding to the atlas patch index of the atlas sample,
* the variable atlasID, corresponding to the atlas ID of the atlas sample,
* the variable geoSample, the decoded (and interpolated) geometry value of the sample, when present.

The output of this process is:

* the variable sampleD, the integer depth value.

This process operates as follows:

d1 = AtlasPatch3dOffsetD[ patchIdx ]  
 if( AsmePatchConstantDepthFlag[ atlasID ] )  
 sampleD = d1  
 else {  
 d2 = d1 + AtlasPatch3dRangeD[ patchIdx ]  
 sampleD = Clip3( d1, d2, geoSample )  
 }

NOTE – The application of the D range [d1, d2] is different between MIV and V-PCC. In MIV, the offset is not applied to the geometry sample, but only used to define the clipping range.

* + 1. Depth expansion process

This process expands the integer depth value of an occupant atlas sample to a floating-point depth value in scene coordinates (e.g. meters).

The integer depth values may be scaled to an implementation-defined bit depth and range 0 .. MaxSampleD. Otherwise, MaxSampleD is set equal to 2asps\_geometry\_2d\_bit\_depth\_minus1 + 1 – 1.

Inputs to this process are:

* the variable MaxSampleD, indicating the maximum depth sample value,
* the variable sampleD, the integer depth value, in the range 0 .. MaxSampleD,

Output of this process is:

* the variable sampleR, indicating the depth as a range value in scene units (e.g. meters) from the cardinal point of the camera to the Cartesian coordinates of the point that is associated with the atlas sample.

NOTE – The definition of range value depends on the projection type. For ERP the range is the ray length. For perspective projection and orthographic projection, the range value corresponds to the projection of the ray vector on the X axis.

The depth expansion process operates as follows:

n1 = ViewNormDispMin[ viewID ]  
 n2 = ViewNormDispMax[ viewID ]  
 normDisp = n1 + ( n2 – n1 ) \* ( SampleD ÷ MaxSampleD )  
 sampleR = 1.0 ÷ normDisp

* + 1. Coordinate unprojection process

This process unprojects the coordinates in projection plane (u, v, r) space to Cartesian coordinates (x, y, z) in scene units (e.g. meters). The Cartesian coordinates are in the reference frame of the view that is associated with the atlas sample.

Inputs to this process and sub-processes are:

* the variables u and v of subclause 9.2.5.1 of ISO/IEC CD 23090-5(2E),
* the variable sampleR, indicating the depth as a range value in scene units (e.g. meters) from the cardinal point of the camera to the Cartesian coordinates of the point that is associated with the atlas sample.,
* the variable patchIdx, the atlas patch index.

Output of this process is:

* the variables sampleX, sampleY and sampleZ, corresponding to the Cartesian coordinates (x, y, z) in scene units, and in the reference frame of the view that is associated with the atlas sample.

In the first step of the process, the projection plane coordinates (U, V) are normalized to [0, 1] range:

viewID = AtlasPatchProjectionID[ patchIdx ]  
 u0 = AtlasPatch3dOffsetU[ patchIdx ]  
 v0 = AtlasPatch3dOffsetV[ patchIdx ]  
 normU = ( u0 + u + 0.5 ) ÷ ViewProjectionPlaneWidth[ viewID ]  
 normV = ( v0 + v + 0.5 ) ÷ ViewProjectionPlaneHeight[ viewID ]

In the second step of the process, the projection type is determined:

projType = ViewType[ viewID ]

The final step of the process depends on the projection type:

* When projType is equal to 0, then the equirectangular unprojection process of H.2.4.1 is executed.
* When projType is equal to 1, then the perspective unprojection process of H.2.4.2 is executed.
* When projType is equal to 2, then the orthographic unprojection process of H.2.4.3 is executed.

Other values of projType are outside of the scope of this version of this document.

* + - 1. ERP unprojection function

The inputs and outputs of this process are defined by the coordinate unprojection process of subclause H.2.4.

This process operates as follows:

sampleX = sampleR \* Cos( theta \* π ÷ 180 ) \* Cos( phi \* π ÷ 180 )  
 sampleY = sampleR \* Cos( theta \* π ÷ 180 ) \* Sin( phi \* π ÷ 180 )  
 sampleZ = sampleR \* Sin( theta \* π ÷ 180 )

where phi and theta are spherical coordinates, derived as follows:

phi = normU \* ViewErpPhiMax[ viewID ] + ( 1 – normU ) \* ViewErpPhiMin[ viewID ]  
 theta = normV \* ViewErpThetaMax[ viewID ] + ( 1 – normV ) \* ViewErpThetaMin[ viewID ]

NOTE – phi decreases with increasing u, and theta decreases with increasing v.

[Ed. I have added this note, because a first reader may not notice that Min and Max are swapped]

* + - 1. Perspective unprojection function

The inputs and outputs of this process are defined by the coordinate unprojection process of subclause H.2.4.

This process operates as follows:

sampleX = sampleR  
 sampleY = –( sampleR / ViewPerspectiveFocalHor[ viewID ] ) \*  
 ( u0 + u + 0.5 – ViewPerspectiveCenterHor[ v ] )  
 sampleZ = –( sampleR / ViewPerspectiveFocalVer[ viewID ] ) \*  
 ( v0 + v + 0.5 – ViewPerspectiveCenterVer[ v ] )

* + - 1. Orthographic unprojection function

The inputs and outputs of this process are defined by the coordinate unprojection process of subclause H.2.4.

This process operates as follows:

sampleX = sampleR  
 sampleY = ViewOrthoWidth[ viewID ] \* ( normU – 0.5 )  
 sampleZ = ViewOrthoHeight[ viewID ] \* ( normV – 0.5 )

* + 1. Reference frame switching process

This process changes the reference frame from the view that is associated with the atlas sample, to the reference of the scene.

Inputs to this process are:

* the variables sampleX, sampleY and sampleZ, corresponding to the Cartesian coordinates (x, y, z) in scene units, and in the reference frame of the view that is associated with the atlas sample,

Outputs of this process are:

* the variables sceneX, sceneY and sceneZ, corresponding to the Cartesian coordinates (x, y, z) in scene units, and in the reference frame of the scene.

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

(H‑1)

where Tx is equal to ViewPosX[ viewID ], Ty is equal to ViewPosY[ viewID ], Tz is equal to ViewPosZ[ viewID ], qX is equal to ViewQuatX[ viewID ], qY is equal to ViewQuatY[ viewID ], and qZ is equal to ViewQuatZ[ viewID ].

The Cartesian coordinates in the reference frame of the view is transformed to Cartesian coordinates in the reference frame of the scene by applying the following equation:

(H‑2)

NOTE 1 – The matrix inverse of another viewToScene matrix viewToSceneB may be used to change the reference frame from the volumetric frame to another view B:

NOTE 2 – Two viewToScene matrices viewToSceneA and viewToSceneB may be combined into a single 4 × 4 matrix viewToSceneAB to change the frame of reference from view A to view B directly:

* 1. Entity filtering process

This process filters the block to patch map of an atlas such that only patches related to targeted entities are kept and patches for non-targeted entities are replaced by the value -1, indicating an unoccupied block in the block to patch map. The output TargetAtlasBlockToPatchMap is a drop-in replacement for AtlasBlockToPatchMap.

The inputs of this process are:

* a 1D array, TargetEntityFlags, indicating which entities are filtered in (1) or filtered out (0), where the dimension corresponds to the entity ID, and the array is determined via external means,

The output of this process is:

* the filtered 2D array TargetAtlasBlockToPatchMap of the same size as AtlasBlockToPatchMap.

The entity filtering process is defined as follows:

for( y = 0; y < AtlasBlockToPatchMapHeight; y++ )  
 for( x = 0; x < AtlasBlockToPatchMapWidth; x++ )  
 p = AtlasBlockToPatchMap[ y ][ x ]  
 e = AtlasPatchEntityID[ p ]  
 TargetAtlasBlockToPatchMap[ y ][ x ] =  
 ( p != -1 && TargetEntityFlag[ e ] ) ? p : -1  
 }  
 }

NOTE – When vme\_max\_entity\_id is equal to 0, then all patches have entity ID 0, and when TargetEntityFlag[ 0 ] is equal to 1, then TargetAtlasBlockToPatchMap will thus be equal to AtlasBlockToPatchMap. Otherwise, when TargetEntityFlag[ 0 ] is equal to 0, then all patches of this atlas will be filtered out.

* 1. Patch attribute offset process

The patch attribute offset process applies an offset value to the attribute component values of an atlas sample.

Inputs to this process are:

* the variable patchIdx, corresponding to the atlas patch index of the atlas sample,
* the 3D array attrSample, corresponding to the 3 component values of the sample attribute.

Output of this process is:

* the 3D array attrSample, corresponding to the 3 offsetted component values of the sample attribute.

The patch attribute offset process is defined as follows:

for(c = 0; c < 3; c++) {  
  attrSample = Clip3( 0, 1 << ai\_attribute\_2d\_bit\_depth\_minus\_1,  
 attrSample + AtlasPatchAttributeOffset[ patchIdx ][ c ] )

* 1. Geometry video scaling process

The geometry video scaling process reconstructs a geometry frame of an atlas 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, as indicated by AsmeGeometryFrameScaleFactorX nonequal to 1 or AsmeGeometryFrameScaleFactorY nonequal to 1.

The inputs of this process are:

* the 2D array DecGeoFrame of size AsmeGeometryFrameHeight × AsmeGeometryFrameWidth,
* the 3D array DecAttrFrame of size 3 × AspsFrameHeight × AspsFrameWidth.

The output of this process is:

* the 2D array GeoFrame of size AspsFrameHeight × AspsFrameWidth.

If AspsFrameWidth is equal to AsmeGeometryFrameWidth and AspsFrameHeight is equal to AsmeGeometryFrameHeight, then the following procedure applies:

for( y = 0; y < AspsFrameHeight; y++ ) {  
 for( x = 0; x < AspsFrameWidth; x++) {  
 GeoFrame[ y ][ x ] = DecGeoFrame[ y ][ x ]  
 }  
 }

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

* The nearest neighbour interpolation scaling process (subclause H.5.1) is invoked.
* The texture aligned geometry erosion process (subclause H.5.5) is invoked.
* The geometry contour smoothening process (subclause H.5.6) is invoked.

The sample neighbours enumeration process (subclause H.5.2), foreground edge flag process (subclause H.5.3), and selective geometry erosion process (subclause H.5.4) are used within the sub processes.

* + 1. Nearest neighbour interpolation scaling process

This process scales the geometry frame at decoded size, DecGeoFrame, to a geometry frame at nominal atlas size using nearest neighbour interpolation. The output of this process, the 2D array ScaledGeoFrame, is derived as follows:

for( y = 0; y < AspsFrameHeight; y++ ) {  
 for( x = 0; x < AspsFrameWidth; x++) {  
 v = y / AsmeGeometryFrameScaleFactorY  
 u = x / AsmeGeometryFrameScaleFactorX   
 ScaledGeoFrame[ y ][ x ] = DecGeoFrame[ 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.

Inputs to this process are:

* a sample position (x, y),
* 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 3×3 or 5×5.

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 operates as follows:

n = PatchPackingBlockSize   
 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 &&  
 0 <= y + ky[ i ] && y + ky[ i ] < AspsFrameHeight &&  
 AtlasBlockToPatchMap[ y / n ][ x / n ] ==  
 AtlasBlockToPatchMap[ ( 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:

* the sample position (x, y),
* the 2D array inputFrame of size sizeY × sizeX, representing a geometry frame,
* the 2D array foregroundEdgeFlag of size sizeY × sizeX

Output of this process is:

* the updated2D array foregroundEdgeFlag of size sizeY × sizeX

This process invokes the sample neighbours enumeration process (subclause H.5.2). The process is defined by the following procedure, where NgX, NgY and NumNeighbours are computed according to H.5.2 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),
* the 2D array inputFrame of size sizeY × sizeX, representing a geometry frame,
* the 2D array erodeFlag of size sizeY × sizeX determines if the sample at (x, y) has to be eroded.

Output of this process is:

* the 2D array outputFrame of size sizeY × sizeX.

This process invokes the sample neighbours enumeration process (subclause H.5.2). This process is defined by the following procedure, where NgX, NgY and NumNeighbours are computed according to H.5.2 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[ NgY[ i ] ][ NgX[ i ] ], 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),
* the 2D array scaledGeoFrame of size sizeY × sizeX

This process invokes the foreground edge flag process (subclause H.5.3) with ScaledGeoFrame as inputFrame, providing foregroundEdgeFlag[ y ][ x ] to this process.

Output of this process is:

* textureAlignedGeoFrame[ y ][ x ].

This process invokes the sample neighbours enumeration process (subclause H.5.2), and the selective geometry erosion process (subclause H.5.4) 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.5.2 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 ) {  
 sad = 0  
 for( c = 0; c <= 3; c++ )  
 sad += Abs( DecAttrFrame[ c ][ NgY[ i ] ][ NgX[ i ] ]  
 - DecAttrFrame[ 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),
* the 2D array textureAlignedGeoFrame of size sizeY × sizeX.

This process invokes the foreground edge flag process (subclause H.5.3) with TextureAlignedGeoFrame as InputFrame, providing ForegroundEdgeFlag[ y ][ x ] to this process.

Output of this process is:

* the 2D array GeoFrame of size sizeY × sizeX

This process also invokes the sample neighbours enumeration process (subclause H.5.2), and the selective geometry erosion process (subclause H.5.4) 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.5.2 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 [ NgY[ i ] ][ NgX[ i ] ]   
 <= TextureAlignedGeoFrame[ y ][ x ] - GupDeltaThreshold )  
 countBackground += 1  
 }  
 }  
 ErodeFlag[ y ][ x ] = countBackground > GupMaxCurvature ? 1 : 0   
 }

* 1. Pruned view reconstruction process

This process reconstructs a frame of a pruned view with view ID equal to viewID from a volumetric frame that has been converted to atlas frame resolution and common bit depths for each of the V3C components.

Inputs to this process are:

* the view ID viewID,
* for each atlas:
  + the variables AspsFrameHeight and AspsFrameWidth indicating the number of rows and columns of the atlas frame, respectively,
  + the 2D array AtlasBlockToPatchMap,
  + the variable PatchPackingBlockSize,
  + the variable asme\_auxiliary\_atlas\_flag,
  + when decoded (or unpacked) geometry video data is present,
    - the 2D array GeoFrame of size AspsFrameHeight × AspsFrameWidth,
  + when decoded (or unpacked) occupancy video data is present,
    - the 2D array OccFrame of size AspsFrameHeight × AspsFrameWidth with a sample value equal to 0 indicating an occupied sample, and any other value indicating an occupied sample,
  + when decoded (or unpacked) texture attribute video data is present,
    - the 3D array texFrame of size AspsFrameHeight × AspsFrameWidth,

For brevity, the following two local variables are defined:

h = ViewProjectionPlaneHeight[ viewID ]  
 w = ViewProjectionPlaneWidth[ viewID ]

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

* the 2D array recOccFrame, an occupancy frame, with dimensions indicating projection plane row index and projection plane column index respectively, of size h x w,
* the 2D array recGeoFrame, a geometry frame, with dimensions indicating projection plane row index and projection plane column index respectively, of size h × w,
* when texFrame is present,
  + the 3D array recTexFrame, with dimensions component index, projection plane row index, and projection plane column index respectively, of size 3 × h × w,
  + the variable texFramePresent is equal to 1, otherwise the value of TexFramePresent is inferred to be equal to 0.

NOTE – While decoded occupancy and geometry video data are optional inputs of this process, they are always provided as outputs of this process.

The initial values of the reconstructed occupancy and geometry frame are set as follows:

for( i= 0; i <= h ; i++ ) {  
 for( j= 0; j < h ; j++ ) {  
 recOccFrame[ i ][ j ]= 0  
 recGeoFrame[ i ][ j ]= 0  
 }  
 }

Initial values of recTexFrame are undefined, if present.

The following process is repeated for each atlas in no particular order:

if( !asme\_auxiliary\_atlas\_flag ) {  
 for( y = 0; y < AspsFrameHeight; y++ ) {  
 for( x = 0; x < AspsFrameWidth; x++) {  
 n = PatchPackingBlockSize  
 p = AtlasBlockToPatchMap[ y / n ][ x / n ]  
 if( 0 <= p && AtlasPatchProjectionID[ p ] == viewID && OccFrame[ y ][ x ] ) {

/\* derive (u, v) from (x, y) according to  
 equation (49) of ISO/IEC CD 23090-5(2E):2020 \*/  
 u0 = AtlasPatch3dOffsetU[ p ]  
 v0 = AtlasPatch3dOffsetV[ p ]

if ( asme\_patch\_constant\_depth\_flag )  
 recGeoFrame[ v0 + v ][ u0 + u ] = AtlasPatch3dOffsetD[ p ]  
 else  
 recGeoFrame[ v0 + v ][ u0 + u ] = GeoFrame[ y ][ x ]

if(texFramePresent )  
 for( c = 0; c < 3; c++ )  
 recTexFrame[ c ][ v0 + v ][ u0 + u ] = texFrame[ c ][ y ][ x ]  
 }  
 }  
 }  
 }

* 1. Sample weighting recovery processes

This process specifies how to use a pruning graph to recover a view blending weight for a sample within a pruned view. It is common for multiview renderers to have a weight per view that depends on the viewport position. This process recovers the sample weight when views are pruned.

* + 1. Pruning graph parsing process

This process converts the global variables output of the decoding process of the pruning graph information into variables for input to the sample weighting recovery process as specified in subclause H.7.2.

Outputs of this process are:

* for each of the views, with view index v in the range of 0 to NumViews - 1, inclusive:
  + the variables ViewLeaf[ v ] and ViewNumChildren[ v ],
  + the 1D array ViewChildIndex[ v ] of size ViewNumChildren[ v ].

The process operates as follows:

for ( v = 0; v < NumViews; v++ ) {  
 ViewLeaf[ v ] = 1  
 ViewNumChildren[ v ] = 0  
 }   
   
 for ( v = 0; v < NumViews; v++ ) {  
 viewID = ViewIndexToID[ v ]  
 if ( ViewRoot[ viewID ] == 0 ) {  
 for ( i = 0; i < ViewNumParents[ viewID ]; i++) {  
 j = ViewParentIndex[ viewID ][ i ]  
 ViewLeaf[ j ] = 0  
 ViewChildIndex[ j ][ ViewNumChildren[ j ]++ ] = v  
 }  
 }  
 }

* + 1. Sample weighting recovery process

This process computes the weight of the contribution to the target view of a sample from a reconstructed pruned view with index v.

Inputs to this process are:

* the position p = ( x, y ) of a sample with valid depth in a reconstructed view with index v,
* for each of the views with index k, in the range of 0 to NumViews - 1, inclusive:
  + the view weight with respect to target view, ViewWeight[ k ] determined by external means,
  + if mvp\_pruning\_graph\_params\_present\_flag is equal to 1, the pruning graph information composed of the variables ViewLeaf[ k ] and ViewNumChildren[ k ] and the 1D array ViewChildIndex[ k ] of size ViewNumChildren[ k ].

Output to this process is:

* the weight of the contribution of the sample to the target view, sampleWeight.

The pixel weighting recovery process operates as follows for pixel p with coordinates (x, y):

if ( mvp\_pruning\_graph\_params\_present\_flag == 0 )  
 sampleWeight = ViewWeight[ v ]  
 else  
 sampleWeight = ViewWeight[ v ] + ComputeChildrenWeight (v, Unproject(v, p))

The invoked function ComputeChildrenWeight(v, P) recursively operates as follows:

ComputeChildrenWeight(v, P) {  
 w = 0  
 if ( ViewLeaf[ v ] == 0 ) {  
 for( i = 0; i < ViewNumChildren[ v ] ; i++) {  
 vChild = ViewChildIndex[ v ][ i ]  
 pOnChild = Project( vChild, P )  
   
 if( IsInViewport( vChild, pOnChild ) == 1) {  
 if ( IsOccupied( vChild, pOnChild ) == 0 ) {  
 w += ViewWeight[ vChild ] + ComputeChildrenWeight( vChild, P )   
 }  
 }  
 else {  
 w += ComputeChildrenWeight (vChild, P)  
 }  
 }  
 }  
 return w  
 }

where:

* the function UnProject(v, p) returns the 3D point P which projects onto pixel p in the v-th view as specified in subclause H.2.4,
* the function Project(v, P) returns the pixel coordinates of the projection of 3D point P in the v-th view,
* the function IsInViewport(v , p) returns true if pixel p is inside the viewport of the v-th view,
* the functionIsOccupied(v, p) returns false if the depth of pixel p in the v-th reconstructed 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. Registration authority for code-points in "MP4 Family" files: [https://mp4ra.org/#](https://mp4ra.org/)