

**ISO/IEC JTC 1/SC 29/WG 11 N18794**

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

**Coding of moving pictures and audio**

**Convenorship: UNI (Italy)**

**Document type:**

**Title: Working Draft 3 of Metadata for Immersive Video**

**Status:**

**Date of document:**

**Source: Video**

**Expected action:**

**No. of pages:**

**Email of convenor:**

**Editors’ integration notes:**

**WD3:**

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

**WD2:**

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

**INTERNATIONAL ORGANISATION FOR STANDARDISATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC1/SC29/WG11**

**CODING OF MOVING PICTURES AND AUDIO**

**ISO/IEC JTC1/SC29/WG11 MPEG2018/M18794**

**October 2019, Geneva, CH**

|  |  |
| --- | --- |
| **Source** | **Editors** |
| **Status** | **Draft** |
| **Title** | **Working Draft 3 of Immersive Video** |
| **Author** | **Jill Boyce, Renaud Doré, Vinod Kumar Malamal Vadakital** |

Contents

[Foreword vi](#_Toc23927668)

[1 Scope 1](#_Toc23927669)

[2 Normative reference 1](#_Toc23927670)

[3 Terms and definitions 1](#_Toc23927671)

[3.1 General 1](#_Toc23927672)

[3.2 Term list 2](#_Toc23927673)

[3.2.1 3D scene 2](#_Toc23927674)

[3.2.2 access unit 2](#_Toc23927675)

[3.2.3 atlas 2](#_Toc23927676)

[3.2.4 atlas component 2](#_Toc23927677)

[3.2.5 atlas list 2](#_Toc23927678)

[3.2.6 atlas patch occupancy map 2](#_Toc23927679)

[3.2.7 atlas sequence 2](#_Toc23927680)

[3.2.8 camera parameters 2](#_Toc23927681)

[3.2.9 camera parameters list 2](#_Toc23927682)

[3.2.10 component 2](#_Toc23927683)

[3.2.11 depth layer 3](#_Toc23927684)

[3.2.12 field of view 3](#_Toc23927685)

[3.2.13 global coordinate axes 3](#_Toc23927686)

[3.2.14 global reference coordinate system 3](#_Toc23927687)

[3.2.15 hypothetical reference renderer 3](#_Toc23927688)

[3.2.16 hypothetical 3D scene reconstructor 3](#_Toc23927689)

[3.2.17 immersive video sequence (IVS) 3](#_Toc23927690)

[3.2.18 immersive video access unit 3](#_Toc23927691)

[3.2.19 layer 3](#_Toc23927692)

[3.2.20 layer pair 3](#_Toc23927693)

[3.2.21 local coordinate axes 3](#_Toc23927694)

[3.2.22 local coordinate system 3](#_Toc23927695)

[3.2.23 omnidirectional media 4](#_Toc23927696)

[3.2.24 omnidirectional projection 4](#_Toc23927697)

[3.2.25 patch 4](#_Toc23927698)

[3.2.26 patch descriptor 4](#_Toc23927699)

[3.2.27 picture pair 4](#_Toc23927700)

[3.2.28 projection 4](#_Toc23927701)

[3.2.29 renderer 4](#_Toc23927702)

[3.2.30 spherical coordinates 4](#_Toc23927703)

[3.2.31 spherical coordinate system 4](#_Toc23927704)

[3.2.32 source 4](#_Toc23927705)

[3.2.33 source view representation 5](#_Toc23927706)

[3.2.34 texture layer 5](#_Toc23927707)

[3.2.35 viewing orientation 5](#_Toc23927708)

[3.2.36 viewing position 5](#_Toc23927709)

[3.2.37 view representation 5](#_Toc23927710)

[3.2.38 Viewing space 5](#_Toc23927711)

[3.2.39 viewport 5](#_Toc23927712)

[4 Abbreviations 5](#_Toc23927713)

[5 Conventions 6](#_Toc23927714)

[5.1 General 6](#_Toc23927715)

[5.2 Arithmetic operators 6](#_Toc23927716)

[5.3 Logical operators 6](#_Toc23927717)

[5.4 Relational operators 6](#_Toc23927718)

[5.5 Bit-wise operators 7](#_Toc23927719)

[5.6 Assignment operators 7](#_Toc23927720)

[5.7 Range notation 7](#_Toc23927721)

[5.8 Mathematical functions 7](#_Toc23927722)

[5.9 Order of operation precedence 8](#_Toc23927723)

[5.10 Variables, syntax elements, and tables 9](#_Toc23927724)

[5.11 Text description of logical operations 11](#_Toc23927725)

[5.12 Processes 12](#_Toc23927726)

[6 Formats and relationships 12](#_Toc23927727)

[6.1 Bitstream formats 12](#_Toc23927728)

[6.2 Sources and outputs 12](#_Toc23927729)

[6.3 Rendering 13](#_Toc23927730)

[6.4 Patches, atlases, atlas patch occupancy map, view representations, and view representation pairs relationships 13](#_Toc23927731)

[6.5 Reference Architecture 14](#_Toc23927732)

[7 Syntax and semantics 15](#_Toc23927733)

[7.1 Method of specifying syntax in tabular form 15](#_Toc23927734)

[7.2 Specification of syntax functions and descriptors 16](#_Toc23927735)

[7.3 Syntax in tabular form 19](#_Toc23927736)

[7.3.1 IV sequence parameters 19](#_Toc23927737)

[7.3.2 Camera parameters list syntax 20](#_Toc23927738)

[7.3.3 Atlas parameters list syntax 21](#_Toc23927739)

[7.3.4 IV access unit parameters syntax 21](#_Toc23927740)

[7.3.5 Viewing Space syntax 22](#_Toc23927741)

[7.4 Semantics 25](#_Toc23927742)

[7.4.1 General 25](#_Toc23927743)

[7.4.2 Order of NAL units and association to IVSes and IV access units 25](#_Toc23927744)

[7.4.3 IV sequence parameters semantics 25](#_Toc23927745)

[7.4.4 Camera parameters list semantics 26](#_Toc23927746)

[7.4.5 IV access units parameters semantics 27](#_Toc23927747)

[7.4.6 Atlas parameters list semantics 27](#_Toc23927748)

[7.4.7 Viewing Space 31](#_Toc23927749)

[8 Decoding process 35](#_Toc23927750)

[8.1 General decoding process 35](#_Toc23927751)

[8.1.1 General 35](#_Toc23927752)

[8.2 Atlas patch occupancy map generator process 36](#_Toc23927753)

[8.2.1 General atlas patch occupancy map generator process 36](#_Toc23927754)

[8.3 Viewport position to global coordinate point unprojection process 37](#_Toc23927755)

[8.3.1 General viewport position to global coordinate point unprojection process 37](#_Toc23927756)

[8.3.2 Depth decoding process 38](#_Toc23927757)

[8.3.3 ERP unprojection process 38](#_Toc23927758)

[8.3.4 Perspective unprojection process 38](#_Toc23927759)

[9 Parsing process 39](#_Toc23927760)

[9.1 General 39](#_Toc23927761)

[9.2 Parsing process for 0-th order Exp-Golomb codes 39](#_Toc23927762)

[9.2.1 General 39](#_Toc23927763)

[9.2.2 Mapping process for signed Exp-Golomb codes 40](#_Toc23927764)

[9.3 Parsing process for truncated unary codes 41](#_Toc23927765)

[9.4 Parsing process for truncated binary codes 41](#_Toc23927766)

[Annex A Profiles, tiers and levels 43](#_Toc23927767)

[A.1 Overview of profiles, tiers and levels 43](#_Toc23927768)

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).

**Information technology — Coded Representation of Immersive Media — Part 12: MPEG Immersive Video (MIV)**

# Scope

The document specifies MPEG Immersive Video (MIV). It provides support for playback of a scene with a range of viewing positions and orientations, with 6 Degrees of Freedom (6DoF).

Support for 360° video, also called omnidirectional video, has been standardized in Omnidirectional Media Format (OMAF) and in Supplemental Enhancement Information (SEI) messages defined for HEVC. These standards can be used for delivering a first degree of immersive visual content also named as 3DoF. However, rendering flat 360° video may generate visual discomfort when objects close to the viewer are rendered: the world around the viewer seems flattened. Interactive parallax provides viewers with cues to their visual system, resulting in an enhanced perception of volume around them. The interactive parallax feature of MIV provides viewers with visual content that closely mimics natural vision, but within a restricted range of viewer motion.

.

# Normative reference

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

|  |  |
| --- | --- |
| [HEVC] | ISO/IEC 23008-2, Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding |
| [IMM] | WD 5 of ISO/IEC 23090-7 Immersive Media Metadata, N18396, 29 March 2019 |
| [OMAF] | WD 4 of ISO/IEC 23090-2 OMAF, N18393, 29th March 2019 |
| [VPCC] | CD of ISO/IEC 23090-5 Video-based Point Cloud Compression, N18479 |
| [IEEE754] | IEEE Standard for Floating-Point Arithmetic (IEEE 754-2019) |

# Terms and definitions

For the purposes of this document, the following terms and definitions apply.

## General

For the purposes of this document, the following terms and definitions apply.

* The terminological databases for use in standardization maintained by ISO and IEC at the following addresses:
  + IEC Electropedia: available at <http://www.electropedia.org/>
  + ISO Online browsing platform: available at <http://www.iso.org/obp>
* The terms and definitions specified in ISO/IEC 14496-12 and ISO/IEC 23008-12 apply unless they are overridden by this clause.

NOTE: In particular, the terms coded image, coded image item, derived image, derived image item, image item, reconstructed image, and source image item are defined in ISO/IEC 23008-12.

* The terms and definitions specified by this document in the following subclauses. [Ed. (JB): Consider how to handle definitions that are duplicated here from Part 7.][Ed. (JB): For now, I suggest that we indicate when a definition is an exact copy or variant from another specification.]

## Term list

### 3D scene

visual content in the *global reference coordinate system*

### access unit

A set of *NAL units* that are associated with each other according to a specified classification rule, are consecutive in *decoding order* and contain at most one *coded picture* with any specific value of nuh\_layer\_id. [Ed. (JB): This definition is an exact copy from HEVC Annex F.]

### atlas

aggregation of *patches* from one or more *view representations* after a packing process, into a picture pair which contains a *texture component* *picture* and a corresponding *depth component* *picture*.

### atlas component

a *texture* or *depth* *component* of an *atlas*.

### atlas list

a list of one or more *atlases* which may be present within the same *IV* *access unit*.

### atlas patch occupancy map

a 2D array corresponding to an *atlas* whose values indicate for each sample position in the *atlas* which *patch* the sample corresponds to, or if the sample is invalid.

### atlas sequence

a sequence of one or more *atlases*, containing at most one *atlas* for each *access unit*.

### camera parameters

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

### camera parameters list

a list of one or more *camera parameters.*

### component

*depth* or *texture*.

### depth layer

a *layer* representing an *atlas depth component*.

### field of view

the extent of the observable world in captured/recorded content or in a physical display device. [Ed. (JB): This definition is an exact copy from IMM.]

### global coordinate axes

coordinate axes that are associated with audio, video, and images representing the same acquisition position and intended to be rendered together [Ed. (JB): This definition is an exact copy from IMM.]

### global reference coordinate system

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

### hypothetical reference renderer

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

### hypothetical 3D scene reconstructor

a hypothetical 3D scene reconstruction model that outputs a 3D video sequence.

### immersive video sequence (IVS)

a sequence of one or more immersive video *access units*

### immersive video access unit

a set of NAL units that are associated with each other according to a specified classification rule, are consecutive in decoding order, and contain exactly one *output representation*. [Need to define output representation.]

### layer

a set of VCL NAL units that all have a particular value of nuh\_layer\_id and the associated non-VCL NAL units, or one of a set of syntactical structures having a hierarchical relationship. [Ed. (JB): This definition is an exact copy from HEVC.]

### layer pair

a *texture layer* and a corresponding *depth layer* representing an atlas within the bitstream.

### local coordinate axes

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

### local coordinate system

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

### omnidirectional media

media such as image or video and its associated audio that enable rendering according to the user's *viewing orientation*, if consumed with a head-mounted device, or according to user's desired *viewport*, otherwise, as if the user was in the spot where and when the media was captured [Ed. (JB): This definition is an exact copy from IMM.]

### omnidirectional projection

inverse of the process by which the samples of a projected picture are mapped to a set of positions identified by a set of azimuth and elevation coordinates on a unit sphere. Projection example are ERP, or perspective. [Ed. (JB): This definition is renamed, but otherwise an exact copy of the definition of “projection” from IMM.]

### patch

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

### patch descriptor

a description of the *patch*, containing its size, location within an *atlas*, rotation within an *atlas*, and location within a *view*.

### picture pair

a *texture picture* and corresponding *depth picture*from the same *access unit*.

### projection

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

### renderer

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

### spherical coordinates

Alternative *local coordinate axes* expressed aslongitude (in degrees),latitude (in degrees) and radius (in meter).

### spherical coordinate system

Alternative representation of a *local coordinate system* using *spherical coordinates*, in units of degrees and meters.

### source

a term used to describe the video material or some of its attributes before encoding. [Ed. (JB): This definition is an exact copy from HEVC.]

### source view representation

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

### texture layer

a *layer*  representing an *atlas texture component*.

### viewing orientation

triple of yaw, pitch and roll characterizing the orientation that a user is consuming the audio-visual content; in case of image or video, characterizing the orientation of the *viewport.*

### viewing position

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

### view representation

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

### Viewing space

Domain constraints for a good viewport rendering. The domain is defined in the 3D global space and related to the viewing direction. It defines a indice between 0 and 1 for every point in space for a given direction of the viewport, to be used by the application.

### viewport

projection of texture onto a planar surface of a field of view of an omnidirectional or 3D image or video suitable for display and viewing by the user with a particular *viewing orientation* and *viewing position*. [Ed. (JB): Different definition for the same term in IMM.]

# Abbreviations

For the purposes of this International Standard, the following abbreviations apply:

|  |  |
| --- | --- |
| 2D | Two-Dimensional |
| CVS | Coded Video Sequence |
| ERP | EquiRectangular Projection |
| FOV | Field Of View |
| HEVC | High Efficiency Video Coding (specified in ISO/IEC 23008-2) |
| HMD | Head-Mounted Display |
| IRAP | Intra Random Access Point |
| OMAF  PSP | Omnidirectional MediA Format (specified in ISO/IEC 23090-2)  Perspective Projection |
| URN | Uniform Resource Name |
| VR | Virtual Reality |

# Conventions

## General

NOTE – The mathematical operators used in this Specification are similar to those used in the C programming language. However, the results of integer division and arithmetic shift operations are defined more precisely, and additional operations are defined, such as exponentiation and real-valued division. Numbering and counting conventions generally begin from 0.

## Arithmetic operators

The following arithmetic operators are defined as follows:

|  |  |
| --- | --- |
| + | Addition |
| − | Subtraction (as a two-argument operator) or negation (as a unary prefix operator) |
| \* | Multiplication, including matrix multiplication |
| xy | Exponentiation. Specifies x to the power of y. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation. |
| / | Integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1. |
| ÷ | Used to denote division in mathematical equations where no truncation or rounding is intended. |
|  | Used to denote division in mathematical equations where no truncation or rounding is intended. |
|  | The summation of f( i ) with i taking all integer values from x up to and including y. |
| x % y | Modulus. Remainder of x divided by y, defined only for integers x and y with x >= 0 and y > 0. |

## Logical operators

The following logical operators are defined as follows:

|  |  |
| --- | --- |
| x && y | Boolean logical "and" of x and y. |
| x  | |  y | Boolean logical "or" of x and y. |
| ! | Boolean logical "not". |
| x ? y : z | If x is TRUE or not equal to 0, evaluates to the value of y; otherwise, evaluates to the value of z. |

## Relational operators

The following relational operators are defined as follows:

|  |  |
| --- | --- |
| > | Greater than. |
| >= | Greater than or equal to. |
| < | Less than. |
| <= | Less than or equal to. |
| = = | Equal to. |
| != | Not equal to. |

## Bit-wise operators

The following bit-wise operators are defined as follows:

|  |  |
| --- | --- |
| & | Bit-wise "and". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0. |
| | | Bit-wise "or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0. |
| ^ | Bit-wise "exclusive or". When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0. |
| x >> y | Arithmetic right shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the MSBs as a result of the right shift have a value equal to the MSB of x prior to the shift operation. |
| x << y | Arithmetic left shift of a two's complement integer representation of x by y binary digits. This function is defined only for non-negative integer values of y. Bits shifted into the LSBs as a result of the left shift have a value equal to 0. |

## Assignment operators

The following arithmetic operators are defined as follows:

|  |  |  |
| --- | --- | --- |
| = | Assignment operator. | |
| Increment, i.e. x+ + is equivalent to x = x + 1; when used in an array index, evaluates to the value of the variable prior to the increment operation. | |
| − − | Decrement, i.e. x− − is equivalent to x = x − 1; when used in an array index, evaluates to the value of the variable prior to the decrement operation. | |
| += | Increment by amount specified, i.e. x += 3 is equivalent to x = x + 3, and x += (−3) is equivalent to x = x + (−3). | |
| −= | Decrement by amount specified, i.e. x −= 3 is equivalent to x = x − 3, and x −= (−3) is equivalent to x = x − (−3). | |

## Range notation

The following notation is used to specify a range of values:

|  |  |
| --- | --- |
| x = y..z | x takes on integer values starting from y to z, inclusive, with x, y, and z being integer numbers and z being greater than y. |

## Mathematical functions

The following mathematical functions are defined:

Abs( x ) = (5‑1)

Ceil( x ) the smallest integer greater than or equal to x. (5‑2)

Clamp( x, vmin, vmax ) = max ( min ( x, vmax), vmin ) (5-3)

Clip1Y( x ) = Clip3( 0, ( 1 << BitDepthY ) – 1, x ) (5‑34)

Clip1C( x ) = Clip3( 0, ( 1 << BitDepthC ) – 1, x ) (5‑45)

Clip3( x, y, z ) = (5‑56)

Cosd( x ) is the cosine function with x in degrees (5‑7)

Float( x ) provides a floating-point approximation of the integer value x (5‑7)

Floor( x ) the largest integer less than or equal to x. (5‑68)

Log2( x ) the base-2 logarithm of x. (5‑79)

Log10( x ) the base-10 logarithm of x. (5‑810)

Min( x, y ) = (5‑911)

Max( x, y ) = (5‑102)

Round( x ) = Sign( x ) \* Floor( Abs( x ) + 0.5 ) (5‑113)

Sign( x ) = (5‑124)

Sind( x ) is the sine function with x in degrees (5‑13)

Sqrt( x ) = (5‑146)

## Order of operation precedence

When order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules apply:

– Operations of a higher precedence are evaluated before any operation of a lower precedence.

– Operations of the same precedence are evaluated sequentially from left to right.

Table 5‑1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

NOTE – For those operators that are also used in the C programming language, the order of precedence used in this Specification is the same as used in the C programming language.

Table 5‑1 – Operation precedence from highest (at top of table) to lowest (at bottom of table)

|  |  |  |
| --- | --- | --- |
| **operations (with operands x, y, and z)** | | |
| "x++", "x− −" | |  |
| "!x", "−x" (as a unary prefix operator) | |  |
| xy | |  |
| "x \* y", "x / y", "x  y""", "x % y" |
| "x  y", "x − y" (as a two-argument operator), | |  |
| "x  <<  y", "x  >>  y" | |  |
| "x < y", "x  <=  y", "x > y", "x  >=  y" |
| "x  = =  y", "x  !=  y" | |  |
| "x & y" | |  |
| "x | y" | |  |
| "x  &&  y" | |  |
| "x  | |  y" | |  |
| "x ? y : z" | |  |
| "x..y" | |  |
| "x = y", "x  +=  y", "x  −=  y" | |  |

## Variables, syntax elements, and tables

Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lower case letters with underscore characters), and one descriptor for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e. not bold) type.

In some cases the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lower case and upper case letter and without any underscore characters. Variables starting with an upper case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an upper case letter may be used in the decoding process for later syntax structures without mentioning the originating syntax structure of the variable. Variables starting with a lower case letter are only used within the subclause in which they are derived.

In some cases, "mnemonic" names for syntax element values or variable values are used interchangeably with their numerical values. Sometimes "mnemonic" names are used without any associated numerical values. The association of values and names is specified in the text. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an upper case letter and may contain more upper case letters.

NOTE – The syntax is described in a manner that closely follows the C-language syntactic constructs.

Functions that specify properties of the current position in the bitstream are referred to as syntax functions. These functions are specified in subclause 7.2 and assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

Functions that are not syntax functions (including mathematical functions specified in subclause 5.8) are described by their names, which start with an upper case letter, contain a mixture of lower and upper case letters without any underscore character, and end with left and right parentheses including zero or more variable names (for definition) or values (for usage) separated by commas (if more than one variable).

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column (horizontal) index. The same indexing order is used when using square parentheses. Thus, an element of a matrix s at horizontal position x and vertical position y may be denoted either as s[ y ][ x ] or as syx. A single row of a matrix may be referred to as a list and denoted by omission of the column index. Thus, the row of a matrix s at horizontal position x may be referred to as the list s[ y ].

NOTE – In some video specifications a reverse indexing order may be used when using square parenthesis for depicting two dimensional arrays, i.e. the first element in the square parentheses is the column (horizontal) index and the second element in the square parentheses is the row (vertical) index.

A specification of values of the entries in rows and columns of an array may be denoted by { {...} {...} }, where each inner pair of brackets specifies the values of the elements within a row in increasing column order and the rows are ordered in increasing row order. Thus, setting a matrix s equal to { { 1 6 } { 4 9 } specifies that s[ 0 ][ 0 ] is set equal to 1, s[ 0 ][ 1 ] is set equal to 6, s[ 1 ][ 0 ] is set equal to 4, and s[ 1 ][ 1 ] is set equal to 9.

Binary notation is indicated by enclosing the string of bit values by single quote marks. For example, '01000001' represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Hexadecimal notation, indicated by prefixing the hexadecimal number by "0x", may be used instead of binary notation when the number of bits is an integer multiple of 4. For example, 0x41 represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Numerical values not enclosed in single quotes and not prefixed by "0x" are decimal values.

A value equal to 0 represents a FALSE condition in a test statement. The value TRUE is represented by any value different from zero.

## Text description of logical operations

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0 )  
 statement 0  
else if( condition 1 )  
 statement 1  
...  
else /\* informative remark on remaining condition \*/  
 statement n

may be described in the following manner:

... as follows / ... the following applies:

– If condition 0, statement 0

– Otherwise, if condition 1, statement 1

– ...

– Otherwise (informative remark on remaining condition), statement n

Each "If ... Otherwise, if ... Otherwise, ..." statement in the text is introduced with "... as follows" or "... the following applies" immediately followed by "If ... ". The last condition of the "If ... Otherwise, if ... Otherwise, ..." is always an "Otherwise, ...". Interleaved "If ... Otherwise, if ... Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0a && condition 0b )  
 statement 0  
else if( condition 1a | | condition 1b )  
 statement 1  
...  
else  
 statement n

may be described in the following manner:

... as follows / ... the following applies:

– If all of the following conditions are true, statement 0:

– condition 0a

– condition 0b

– Otherwise, if one or more of the following conditions are true, statement 1:

– condition 1a

– condition 1b

– ...

– Otherwise, statement n

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0 )  
 statement 0  
if( condition 1 )  
 statement 1

may be described in the following manner:

When condition 0, statement 0

When condition 1, statement 1

## Processes

Processes are used to describe the decoding of syntax elements. A process has a separate specification and invoking. All syntax elements and upper case variables that pertain to the current syntax structure and depending syntax structures are available in the process specification and invoking. A process specification may also have a lower case variable explicitly specified as input. Each process specification has explicitly specified an output. The output is a variable that can either be an upper case variable or a lower case variable.

When invoking a process, the assignment of variables is specified as follows:

– If the variables at the invoking and the process specification do not have the same name, the variables are explicitly assigned to lower case input or output variables of the process specification.

– Otherwise (the variables at the invoking and the process specification have the same name), assignment is implied.

In the specification of a process, a specific coding block may be referred to by the variable name having a value equal to the address of the specific coding block.

# Formats and relationships

## Bitstream formats

The bitstream contains:

* an HEVC bitstream containing one or more layer pairs
* IV sequence parameters, including:
  + camera parameters list
* IV access unit parameters, including:
  + atlas parameters list

## Sources and outputs

The immersive video source that is represented by the bitstream is one or more independently coded sequence pairs of texture and depth pictures. Each of the sequence pairs represents a view of a 3D scene, and may have been captured by a real camera or generated by a virtual camera. Texture pictures are in 4:2:0 chroma format, compatible with chroma\_format\_idc equal to 1 as described in ISO/IEC 23008-2. Depth pictures are in full range 4:2:0 chroma format, but only the luma component is used to represent the depth.

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

## Rendering

A hypothetical reference renderer can reconstruct a viewport at a specified viewing position and viewing orientation from the outputs defined in Clause 6.2.

A hypothetical 3D scene reconstructor can reconstruct a 3D video sequence from the outputsLayer pairs, layers, CVSes, decoded picture pairs, and atlas relationships

The bitstream contains one or more layer pairs, each layer pair having a texture layer and a depth layer. Each layer contains one or more consecutive CVSes in a unique single HEVC independent layer, with each CVS containing a sequence of coded pictures.

Each layer pair represents a sequence of atlases. An atlas is represented by a decoded picture pair in each access unit, with a texture component picture and a depth component picture. The size of an atlas is equal to the size of the decoded picture of the texture layer representing the atlas.

In this version of the specification, the depth decoded picture size shall be equal to the decoded picture size of the corresponding texture layer of the same layer pair. Decoded picture sizes may vary for different layer pairs in the same bitstream.

In this version of the specification, access units which contain an IRAP picture in any layer, shall contain an IRAP picture in all layers containing a coded picture.

## Patches, atlases, atlas patch occupancy map, view representations, and view representation pairs relationships

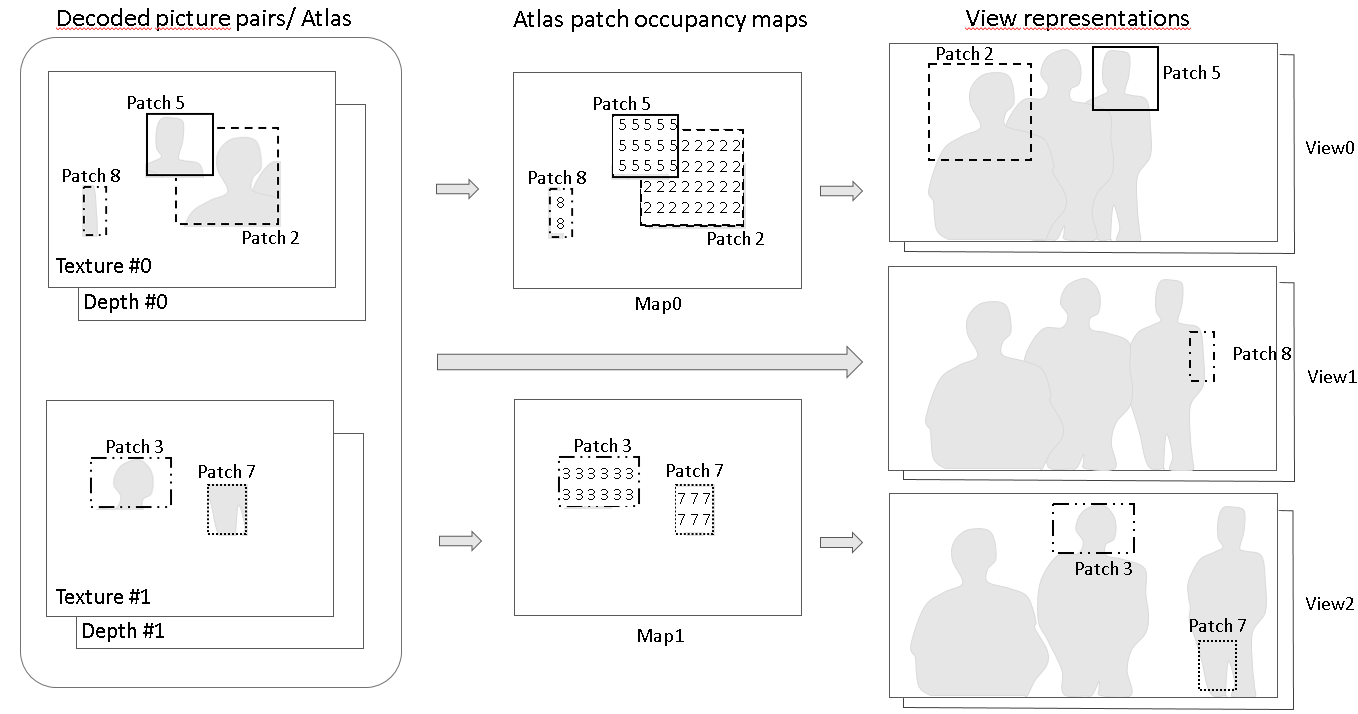


Figure 1: Example mapping of 5 patches in 2 atlases to 3 view representations

A patch is a rectangular region that is represented in both an atlas and a view representation. In this version of the specification, the size of a particular patch is the same in both the atlas representation and the view representation.

An atlas contains an aggregation of one or more patches from one or more view representations, with a corresponding texture component and depth component. The atlas patch occupancy map generator process specified in clause 8.2 outputs an atlas patch occupancy map. The atlas patch occupancy map is a 2D array of the same size as the atlas, with each value indicating the index of the patch to which the co-located sample in the atlas corresponds, if any, or otherwise indicates that the sample location has an invalid value.

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

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

## Reference Architecture

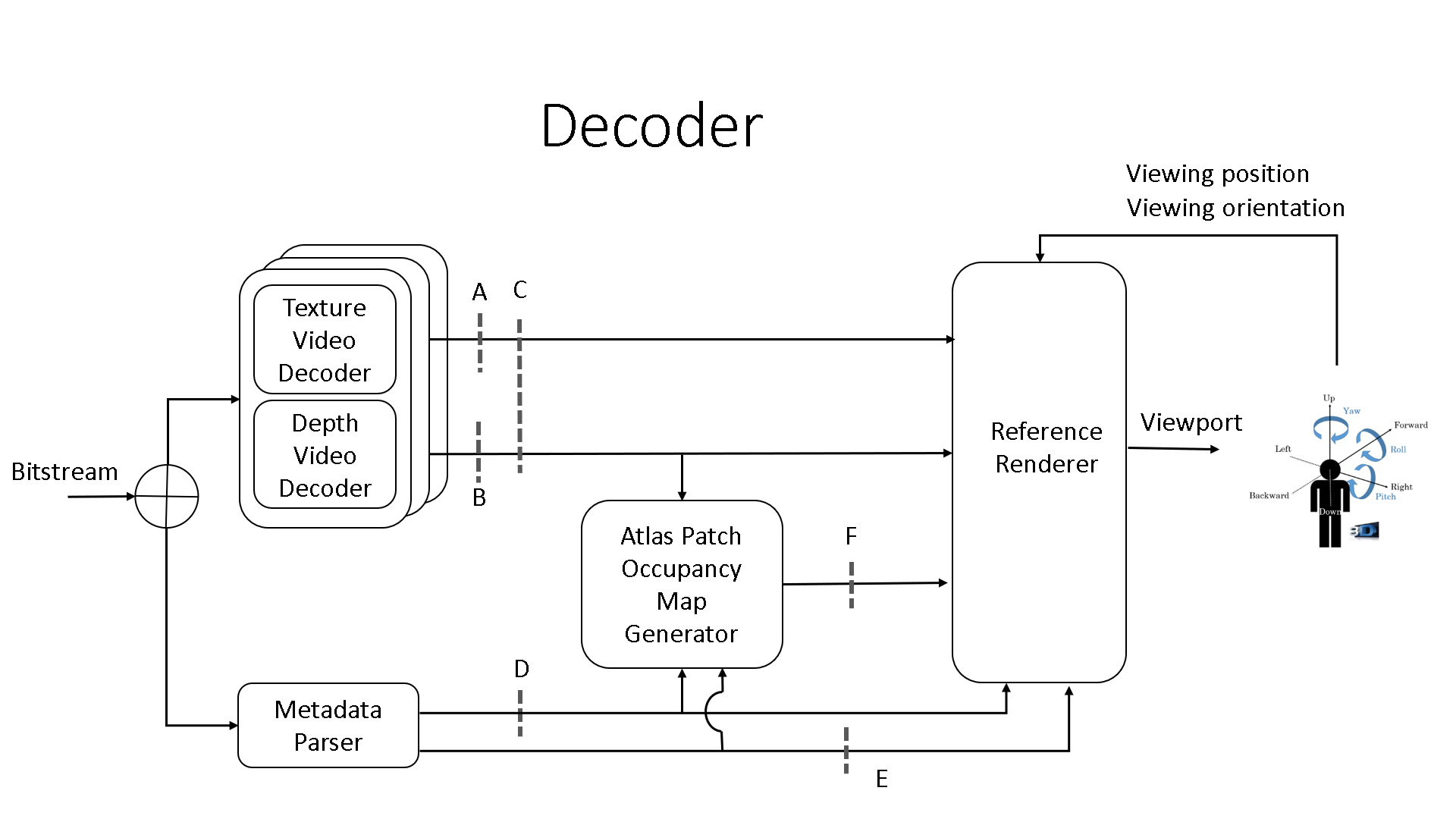
The reference architecture is illustrated in Figure 2. The contents of the bitstream are described in clause 6.1.

A CVS for each texture and depth layer of a layer pair is input to an HEVC decoder, which outputs a sequence of decoded picture pairs of synchronized decoded texture pictures (A) and decoded depth pictures (B). Each decoded picture pair represents an atlas (C).

The metadata is input to a metadata parser which outputs IV access unit parameters (D), which include an AU an atlas parameters list, and IV sequence parameters (E), which include a camera parameters list.

The atlas patch occupancy map generator, specified in Clause 8.2, takes as inputs the depth decoded picture (B),the IV access unit parameters (D), which include the atlas parameters list, and the IV sequence parametersand outputs an atlas patch occupancy map (F).

In the reference architecture, a hypothetical reference renderer take as inputs one or more decoded atlases (C), the IV access unit parameters (D), which includes the atlas parameters list, the IV sequence parameters (E), which includes the camera parameters list, the atlas patch occupancy map sequence (F), and the viewer position and orientation, and outputs a viewport. The reference renderer is not defined in this specification.



**Figure 2: Reference architecture of the immersive video decoder**

# Syntax and semantics

## Method of specifying syntax in tabular form

The syntax tables specify a superset of the syntax of all allowed bitstreams. Additional constraints on the syntax may be specified, either directly or indirectly, in other clauses.

NOTE – An actual decoder should implement some means for identifying entry points into the bitstream and some means to identify and handle non-conforming bitstreams. The methods for identifying and handling errors and other such situations are not specified in this Specification.

The following table lists examples of the syntax specification format. When **syntax\_element** appears, it specifies that a syntax element is parsed from the bitstream and the bitstream pointer is advanced to the next position beyond the syntax element in the bitstream parsing process.

|  |  |
| --- | --- |
|  | Descriptor |
| /\* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type, and quantity of syntax elements, as in the following two examples \*/ |  |
| **syntax\_element** | ue(v) |
| conditioning statement |  |
|  |  |
| /\* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. \*/ |  |
| { |  |
| Statement |  |
| Statement |  |
| ... |  |
| } |  |
|  |  |
| /\* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true \*/ |  |
| while( condition ) |  |
| Statement |  |
|  |  |
| /\* A "do ... while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true \*/ |  |
| Do |  |
| Statement |  |
| while( condition ) |  |
|  |  |
| /\* An "if ... else" structure specifies a test of whether a condition is true and, if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed \*/ |  |
| if( condition ) |  |
| primary statement |  |
| Else |  |
| alternative statement |  |
|  |  |
| /\* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. \*/ |  |
| for( initial statement; condition; subsequent statement ) |  |
| primary statement |  |

## Specification of syntax functions and descriptors

The functions presented here are used in the syntactical description. These functions are expressed in terms of the value of a bitstream pointer that indicates the position of the next bit to be read by the decoding process from the bitstream.

byte\_aligned( ) is specified as follows:

– If the current position in the bitstream is on a byte boundary, i.e. the next bit in the bitstream is the first bit in a byte, the return value of byte\_aligned( ) is equal to TRUE.

– Otherwise, the return value of byte\_aligned( ) is equal to FALSE.

more\_data\_in\_byte\_stream( ), which is used only in the byte stream NAL unit syntax structure specified in Annex B[Ed.(BK): Annex B is missing], is specified as follows:

– If more data follow in the byte stream, the return value of more\_data\_in\_byte\_stream( ) is equal to TRUE.

– Otherwise, the return value of more\_data\_in\_byte\_stream( ) is equal to FALSE.

more\_data\_in\_payload( ) is specified as follows:

– If byte\_aligned( ) is equal to TRUE and the current position in the sei\_payload( ) syntax structure is 8 \* payloadSize bits from the beginning of the sei\_payload( ) syntax structure, the return value of more\_data\_in\_payload( ) is equal to FALSE.

– Otherwise, the return value of more\_data\_in\_payload( ) is equal to TRUE.

more\_rbsp\_data( ) is specified as follows:

– If there is no more data in the raw byte sequence payload (RBSP), the return value of more\_rbsp\_data( ) is equal to FALSE.

– Otherwise, the RBSP data are searched for the last (least significant, right-most) bit equal to 1 that is present in the RBSP. Given the position of this bit, which is the first bit (rbsp\_stop\_one\_bit) of the rbsp\_trailing\_bits( ) syntax structure, the following applies:

– If there is more data in an RBSP before the rbsp\_trailing\_bits( ) syntax structure, the return value of more\_rbsp\_data( ) is equal to TRUE.

– Otherwise, the return value of more\_rbsp\_data( ) is equal to FALSE.

The method for enabling determination of whether there is more data in the RBSP is specified by the application (or in Annex B for applications that use the byte stream format).

more\_rbsp\_trailing\_data( ) is specified as follows:

– If there is more data in an RBSP, the return value of more\_rbsp\_trailing\_data( ) is equal to TRUE.

– Otherwise, the return value of more\_rbsp\_trailing\_data( ) is equal to FALSE.

next\_bits( n ) provides the next bits in the bitstream for comparison purposes, without advancing the bitstream pointer. Provides a look at the next n bits in the bitstream with n being its argument. When used within the byte stream format as specified in Annex B and fewer than n bits remain within the byte stream, next\_bits( n ) returns a value of 0.

payload\_extension\_present( ) is specified as follows:

– If the current position in the sei\_payload( ) syntax structure is not the position of the last (least significant, right-most) bit that is equal to 1 that is less than 8 \* payloadSize bits from the beginning of the syntax structure (i.e., the position of the payload\_bit\_equal\_to\_one syntax element), the return value of payload\_extension\_present( ) is equal to TRUE.

– Otherwise, the return value of payload\_extension\_present( ) is equal to FALSE.

The following descriptors specify the parsing process of each syntax element:

– b(8): byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function read\_bits( 8 ).

– f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ).

– se(v): signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2.

– i(n): signed integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ) interpreted as a two's complement integer representation with most significant bit written first.

– u(n): unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ) interpreted as a binary representation of an unsigned integer with most significant bit written first.

* fl(n): [IEEE754] binary*n* floating point value, representing a finite numbers.

– ue(v): unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 9.2.

## Syntax in tabular form

### IV sequence parameters

|  |  |
| --- | --- |
| iv\_sequence\_params( ) { | **Descriptor** |
| ivs\_profile\_tier\_level( ) |  |
| **depth\_params\_num\_bits\_minus8** | u(4) |
| view\_params\_list( ) |  |
| **depth\_low\_quality\_flag** | u(1) |
| **num\_groups\_minus1** | ue(v) |
| **max\_entities\_minus1** | ue(v) |
| **viewing\_space\_present\_flag** | u(1) |
| if(viewing\_space\_present\_flag) | u(1) |
| viewing\_space( ) |  |
| **ivs\_sp\_extension\_present\_flag** | u(1) |
| if( ivs\_sp\_extension\_present\_flag ) { |  |
| while( more\_data\_in\_payload( ) ) |  |
| **ivs\_sp\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) [Ed. (JB): Is this still needed?] |  |
| } |  |

#### IV profile, tier, level syntax

[Ed. (JB): To be provided.

|  |  |
| --- | --- |
| ivs\_profile\_tier\_level( ) { | Descriptor |
| } |  |

### Camera parameters list syntax

|  |  |
| --- | --- |
| view\_params\_list( ) { | **Descriptor** |
| **num\_views\_minus1** | u(16) |
| for( v = 0; v <= num\_views\_minus1; i++) { |  |
| **cam\_pos\_x**[ v ] | fl(32) |
| **cam\_pos\_y**[ v ] | fl(32) |
| **cam\_pos\_z**[ v ] | fl(32) |
| **cam\_yaw**[ v ] | fl(32) |
| **cam\_pitch**[ v ] | fl(32) |
| **cam\_roll**[ v ] | fl(32) |
| } |  |
| **intrinsic\_params\_equal\_flag** | u(1) |
| for( v = 0; v <= intrinsic\_params\_equal\_flag ? 0 : num\_ views\_minus1; v++ ) |  |
| camera\_intrinsics( v  ) |  |
| **depth\_quantization\_params\_equal\_flag** | u(1) |
| for( v = 0; v <= depth\_quantization\_equal\_flag ? 0 : num\_views\_minus1; v++ ) |  |
| depth\_quantization( v ) |  |
| } |  |

#### Camera intrinsics syntax

|  |  |
| --- | --- |
| camera\_intrinsics( v ) { | **Descriptor** |
| **cam\_type**[ v ] | u(8) |
| **projection\_plane\_width\_minus1**[ v ] | u(16) |
| **projection\_plane\_height\_minus1**[ v ] | u(16) |
| if( cam\_type[ v ] == 0 ) { |  |
| **erp\_phi\_min**[ v ] | fl(32) |
| **erp\_phi\_max**[ v ] | fl(32) |
| **erp\_theta\_min**[ v ] | fl(32) |
| **erp\_theta\_max**[ v ] | fl(32) |
| } else if(cam\_type[ v ] = = 1) { |  |
| **perspective\_focal\_hor**[ v ] | fl(32) |
| **perspective\_focal\_ver**[ v ] | fl(32) |
| **perspective\_center\_hor**[ v ] | fl(32) |
| **perspective\_center\_ver**[ v ] | fl(32) |
| } |  |
| } |  |

#### Depth quantization syntax

|  |  |
| --- | --- |
| depth\_quantization( v ) { | **Descriptor** |
| **quantization\_law**[ v ] | u(8) |
| if( quantization\_law[ v ] == 0 ) { |  |
| **norm\_disp\_low**[ v ] | fl(32) |
| **norm\_disp\_high**[ v ] | fl(32) |
| } |  |
| **depth\_occ\_map\_threshold\_default**[ v ] | u(v) |
| **depth\_start\_default\_present\_flag**[ v ] | u(1) |
| if( depth\_start\_default\_present\_flag[ v ] ) |  |
| **depth\_start\_default**[ v ] | u(v) |
| } |  |

### Atlas parameters list syntax

|  |  |
| --- | --- |
| atlas\_params\_list( ) { |  |
| **num\_atlases\_minus1** | ue(v) |
| **omaf\_v1\_compatible\_flag** | u(1) |
| for( i = 0; i <= num\_atlases\_minus1; i++ ) { |  |
| **atlas\_id**[ i ] | u(8) |
| if( num\_groups\_minus1 > 0 ) |  |
| **group\_id**[ i ] | u(v) |
| atlas\_params( atlas\_id[ i ] ) |  |
| } |  |
| } |  |

### IV access unit parameters syntax

|  |  |
| --- | --- |
| iv\_access\_unit\_params( ) { | **Descriptor** |
| **atlas\_params\_present\_flag** | u(1) |
| if(  atlas\_params\_present\_flag) |  |
| atlas\_params\_list( ) |  |
| **ivs\_aup\_extension\_present\_flag** | u(1) |
| if( ivs\_aup\_extension\_present\_flag ) { |  |
| while( more\_rbsp\_data( ) ) |  |
| **ivs\_aup\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Atlas parameters syntax

|  |  |
| --- | --- |
| atlas\_params( a ) { | **Descriptor** |
| **num\_patches\_minus1**[ a ] | u(16) |
| **atlas\_width\_minus1**[ a ] | u(16) |
| **atlas\_height\_minus1**[ a ] | u(16) |
| **depth\_occ\_params\_present\_flag**[ a ] | u(1) |
| for( p = 0; p <= num\_patches\_minus1; p++ ) { |  |
| **view\_id**[ a ][ p ] | u(v) |
| if( max\_entities\_minus1 > 0 ) |  |
| **entity\_id**[ a ][ p ] | u(v) |
| **patch\_width\_in\_view**[ a ][ p ] | u(v) |
| **patch\_height\_in\_view**[ a ][ p ] | u(v) |
| **patch\_pos\_in\_atlas\_x**[ a ][ p ] | u(v) |
| **patch\_pos\_in\_atlas\_y**[ a ][ p ] | u(v) |
| **patch\_pos\_in\_view\_x**[ a ][ p ] | u(v) |
| **patch\_pos\_in\_view\_y**[ a ][ p ] | u(v) |
| **patch\_rotation**[ a ][ p ] | u(3) |
| if( depth\_occ\_params\_present\_flag[ a ] ) |  |
| depth\_occupancy( a, p ) |  |
| } |  |
| } |  |

#### Depth occupancy syntax

|  |  |
| --- | --- |
| depth\_occupancy( a, p ) { |  |
| **depth\_occ\_map\_threshold\_present\_flag**[ a ][ p ] | u(1) |
| if( depth\_occ\_map\_threshold\_present\_flag[ a ][ p ] ) |  |
| **depth\_occ\_map\_threshold**[ a ][ p ] | u(v) |
| **depth\_start\_present\_flag**[ a ][ p ] | u(1) |
| if( depth\_start\_present\_flag[ a ][ p ] ) |  |
| **depth\_start**[ a ][ p ] | u(v) |
| } |  |

### Viewing Space syntax

|  |  |
| --- | --- |
| viewing\_space( ) { | **Descriptor** |
| **num\_elementary\_shapes\_minus1** | u(v) |
| for( e = 0; e <= num\_elementary\_shapes\_minus1; e++) { |  |
| **elementary\_shape\_operation**[ e ] | u(1) |
| elementary\_shape( e ) |  |
| } |  |
| } |  |

#### Elementary\_shape

|  |  |
| --- | --- |
| elementary\_shape( e ) { | **Descriptor** |
| **num\_primitive\_shapes\_minus\_1**[ e ] | u(8) |
| **primitive\_shape\_operation**[ e ] | u(1) |
| **guard\_band\_present\_flag**[ e ] | u(1) |
| **primitive\_orientation\_present\_flag**[ e ] | u(1) |
| **viewing\_direction\_constraint\_present\_flag**[ e ] | u(1) |
| for( s= 0; s <= num\_primitive\_shapes\_minus1; s++) { |  |
| **primitive\_shape\_type**[ e ][ s ] | u(2) |
| if(primitive\_shape\_type[ e ][ s ] == 0) |  |
| cuboid\_primitive (e , s) |  |
| else if(primitive\_shape\_type[ e ][ s ] == 1) |  |
| spheroid\_primitive (e, s) |  |
| else if(primitive\_shape\_type[ e ][ s ]== 2) |  |
| halfspace\_primitive(e, s) |  |
| if(guard\_band\_present\_flag[ e ] ) |  |
| **guard\_band\_size**[ e ][ s ] | fl(16) |
| if(primitive\_orientation\_present\_flag[ e ] ) { |  |
| **primitive\_shape\_yaw**[ e ][ s ] | fl(16) |
| **primitive\_shape\_pitch**[ e ][ s ] | fl(16) |
| **primitive\_shape\_roll**[ e ][ s ] | fl(16) |
| } |  |
| if( viewing\_direction\_constraint\_present\_flag[ e ] ) { |  |
| if( guard\_band\_present\_flag[ e ]) |  |
| **guard\_band\_direction\_size**[ e ][ s ] | fl(16) |
| **primitive\_shape\_viewing\_direction\_yaw\_center**[ e ][ s ] | fl(16) |
| **primitive\_shape\_viewing\_direction\_yaw\_range**[ e ][ s ] | fl(16) |
| **primitive\_shape\_viewing\_direction\_pitch\_center**[ e ][ s ] | fl(16) |
| **primitive\_shape\_viewing\_direction\_pitch\_range**[ e ][ s ] | fl(16) |
| } |  |
| } |  |
| } |  |

#### Cuboid primitive

|  |  |
| --- | --- |
| cuboid\_primitive( e, s ) { |  |
| **center\_x**[ e ][ s ] | fl(16) |
| **center\_y**[ e ][ s ] | fl(16) |
| **center\_z**[ e ][ s ] | fl(16) |
| **size\_x**[ e ][ s ] | fl(16) |
| **size\_y**[ e ][ s ] | fl(16) |
| **size\_z**[ e ][ s ] | fl(16) |
| } |  |

#### Sphere primitive

|  |  |
| --- | --- |
| sphere\_primitive( e, s ) { | **Descriptor** |
| **center\_x**[ e ][ s ] | fl(16) |
| **center\_y**[ e ][ s ] | fl(16) |
| **center\_z**[ e ][ s ] | fl(16) |
| **radius\_x**[ e ][ s ] | fl(16) |
| **radius\_y**[ e ][ s ] | fl(16) |
| **radius\_z**[ e ][ s ] | fl(16) |
| } |  |

#### Half\_space primitive

|  |  |
| --- | --- |
| halfspace\_primitive( e, s ) { | **Descriptor** |
| **normal\_x**[ e ] [ s ] | fl(16) |
| **normal\_y**[ e ] [ s ] | fl(16) |
| **normal\_z**[ e ] [ s ] | fl(16) |
| **distance**[ e ][ s ] | fl(16) |
| } |  |

## Semantics

### General

Semantics associated with the syntax structures and with the syntax elements within these structures are specified in this subclause. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this Specification.

### Order of NAL units and association to IVSes and IV access units

A bitstream conforming to this Specification consists of one or more IVSes. Each IVS consists of one or more IV access units .

Each IVS consists of an IV sequence parameters syntax structure, one or more IRAP-aligned multi-layer video codec CVSs, and one or more IV access unit parameters syntax structure.

An IRAP-aligned multi-layer video codec CVS consists of one or more layer groups, each layer group representing an atlas containing a texture component CVS and a depth component CVS, where each CVS is contained within a distinct layer. The IRAP-aligned multi-layer video codec CVS consists of one or more video codec access units, each video codec access unit containing one or more coded pictures, and when an IRAP picture is present in an access unit for any layer in the layer group, an IRAP picture shall be present in all layers in the layer group in that access unit.

An IV access unit consists of:

* an IV access unit parameter syntax structure
* zero or one atlas parameters list, which describes one or more atlases
* for each atlas
  + zero or one texture coded picture
  + zero or one depth coded picture

### IV sequence parameters semantics

**depth\_params\_num\_bits\_minus8** plus 8 specifies the number of bits used to represent the depth\_occ\_map\_threshold\_default[ v ] and depth\_start\_default[ v ] syntax elements in the depth\_quantization( v ) syntax structure, and the depth\_occ\_map\_threshold[ a ][ p ] and depth\_start[ a ][ p ] syntax elements in the depth\_occupancy( a, p ) syntax structure.

**depth\_low\_quality\_flag** equal to 1 indicates that the depth fidelity confidence is low. depth\_low\_quality\_flag equal to 0 indicates that the depth fidelity confidence is not low.

**num\_groups\_minus1** specifies the maximum value of group\_id[ i ] in the atlas\_params\_list( ) syntax structure. When not present, the value of num\_groups\_minus1 is inferred to be equal to 0.

**max\_entities\_minus1** specifies the maximum value of entity\_id[ a ][ i ] in the atlas\_params( ) syntax structure. When not present, the value of max\_entities\_minus1 is inferred to be equal to 0.

**viewing\_space\_present\_flag** equal to 1 specifies that the viewing\_space( ) syntax structure is present. viewing\_space\_present\_flag equal to 1 specifies that the viewing\_space( ) syntax structure is not present.

**ivs\_sp\_extension\_present\_flag** equal to equal to 0 specifies that no ivs\_sp\_extension\_data\_flag syntax elements are present in the syntax structure. ivs\_sp\_extension\_present\_flag equal to 1 specifies that there are ivs\_sp\_extension\_data\_flag syntax elements present syntax structure. Decoders conforming to a profile specified in Annex A shall ignore all data that follow the value 1 for ivs\_sp\_extension\_present\_flag.

**ivs\_sp\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in Annex A. Decoders conforming to a profile specified in Annex A shall ignore all ivs\_sp\_extension\_data\_flag syntax elements.

#### IV profile, tier, level semantics

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

### Camera parameters list semantics

**view\_params\_list**( ) specifies a list of cameras representing the 3D scene.

**num\_cameras\_minus1** plus 1 indicates the number of cameras in the camera list.

**cam\_pos\_x**[ i ] specifies in meters the X (back-to-front) coordinate of the location of the i-th camera as floating point in the global reference coordinate system.

**cam\_pos\_y**[ i ] specifies in meters the Y (lateral, left-to-right) coordinate of the location of the i-th camera as floating point in the global reference coordinate system.

**cam\_pos\_z**[ i ] specifies in meters the Z (down-to-up) coordinate of the location of the i-th camera as floating point in the global reference coordinate system.

**cam\_yaw**[ i ], **cam\_pitch**[ i ], and **cam\_roll**[ i ] specify the yaw, pitch, and roll angles respectively, of the rotation that is applied to convert the global coordinate axes to the local coordinate axes of the i-th camera, as floating-point in units of degrees, in the global reference coordinate system.

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

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

#### Camera intrinsics semantics

**cam\_type**[ v ] indicates the projection method of the v-th camera. cam\_type[ v ] equal to 0 specifies ERP projection. cam\_type[ v ] equal to 1 specifies a perspective projection. cam\_type values in range 2 to 255 are reserved for future use by ISO/IEC.

**projection\_plane\_width\_minus1**[ v ] + 1 and **projection\_plane\_height\_minus1**[ v ] + 1 specify the horizontal and vertical resolutions of the camera projection plane, respectively, expressed in coded luma samples.

**erp\_phi\_min**[ v ] and **erp\_phi\_max**[ v ] specify the longitude range (minimum and maximum values) for an ERP projection, as floating-point in units of degrees. erp\_phi\_min[ v ] and erp\_phi\_max[ v ] shall be in the range −180 to 180, inclusive, in the spherical coordinate system.

**erp\_theta\_min**[ v ] and **erp\_theta\_max**[ v ] specify the latitude range (minimum and maximum values) for an ERP projection, as floating-point in units of degrees. erp\_theta\_min[ v ] and erp\_theta\_max[ v ] shall be in the range −90 to 90, inclusive, in the spherical coordinate system.

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

**perspective\_center\_hor**[ v ] and **perspective\_center\_ver**[ v ] are floating-point values that specify in luma sample positions the horizontal and vertical coordinates, respectively, of the principal point of a perspective projection (intersection of optical axis with image plane).

NOTE: It is possible to have a perspective camera with a projection center outside of the viewport. [Ed.(BK): I have added this note only because I removed the restriction from the semantics.]

#### Depth quantization semantics

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

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

**depth\_occ\_map\_threshold\_default**[ v ] specifies the inference value of the depth\_occ\_map\_threshold[ a ][ i ] syntax element for the v-th view. The number of bits of the syntax element is depth\_params\_num\_bits\_minus8 + 8 bits.

**depth\_start\_default\_present\_flag** equal to 1 specifies that the depth\_start\_default[ v ] syntax element is present. depth\_start\_default\_present\_flag equal to 0 specifies that the depth\_start\_default[ v ] syntax element is not present. [Ed. (JB): Maybe rename and improve the semantics to clarify what happens when no depth start value is used at all, and reference the subclause of the corresponding decoding process.] [Ed.(BK): There are reasons to allow all combinations of depth\_start\_default\_present\_flag and depth\_start\_present[ v ]. Let’s use Section 8.3 for more explanation.]

**depth\_start\_default**[ v ] specifies the inference value of the depth\_start[ a ][ i ] syntax element for the v-th view. The number of bits of the syntax element is depth\_params\_num\_bits\_minus8 + 8 bits.

### IV access units parameters semantics

**atlas\_params\_present\_flag** equal to 1 indicates that the atlas\_params\_list( ) syntax structure is present. atlas\_params\_present\_flag equal to 0 indicates that the atlas\_params\_list( ) syntax structure is not present.

**ivs\_aup\_extension\_present\_flag** equal to equal to 0 specifies that no ivs\_aup\_extension\_data\_flag syntax elements are present in the syntax structure. ivs\_aup\_extension\_present\_flag equal to 1 specifies that there are ivs\_aup\_extension\_data\_flag syntax elements present syntax structure. Decoders conforming to a profile specified in Annex A shall ignore all data that follow the value 1 for ivs\_aup\_extension\_present\_flag.

**ivs\_aup\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in Annex A. Decoders conforming to a profile specified in Annex A shall ignore all ivs\_aup\_extension\_data\_flag syntax elements.

### Atlas parameters list semantics

**num\_atlases\_minus1** plus 1 indicates the number of atlases in the atlas list.

**omaf\_v1\_compatible\_flag** specifies that the layer containing the 0-th atlas texture component is compatible for carriage within ISO/IEC 23009-2 1st edition (2019). When 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 layer containing the 0-th atlas texture component conforms to a projection format specified in ISO/IEC 23009-2 1st edition (2019).

**atlas\_id**[ i ] indicates the index of the i-th atlas.

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

#### Atlas parameters semantics

**num\_patches\_minus1**[ a ]plus 1 indicates the number of patches in the a-th atlas.

**atlas\_width\_minus1**[ a ] plus 1 indicates the width of the a-th atlas in luma samples. It is a requirement of bitstream conformance that atlas\_width[ a ] equal pic\_width\_in\_luma\_samples in the active SPS for the texture atlas component layer of the a-th atlas.

**atlas\_height\_minus1**[ a ] plus 1 indicates the height of the a-th atlas in luma samples. It is a requirement of bitstream conformance that atlas\_height[ a ] equal pic\_height\_in\_luma\_samples in the active SPS for the texture atlas component layer of the a-th atlas.

**depth\_occ\_params\_present\_flag**[ a ] equal to 1 specifies that the depth\_occupancy(a, i) syntax structure is present. depth\_occ\_params\_present\_flag[ a ] equal to 1 specifies that the depth\_occupancy(a, i) syntax structure is not present.

**view\_id**[ a ][ p ] indicates the view representation pair index associated with the p-th patch of the a-th atlas.

**entity\_id**[ a ][ p ] specifies the entityID of the p-th patch of the a-th atlas, within the view\_id[ a ][ p ]-th view. The number of bits used for the representation of entity\_id[ a ][ p ] is Ceil(Log2(max\_entities\_minus1+1) ). The value of entity\_id[ a ][ p ] shall be in range of 0 .. max\_entities\_minus1. When not present, the value of entity\_id[ a ][ p ] is inferred to be equal to 0.

**patch\_width\_in\_view\_minus1**[ a ][ p ] + 1 and **patch\_height\_in\_view\_minus1**[ a ][ p ] + 1 specify the width and height in luma samples, respectively, of the p-th patch of the a-th atlas, within the view\_id[ a ][ p ]-th view. The number of bits used for the representation of patch\_width\_in\_view[ a ][ p ] and patch\_height\_in\_view[ a ][ p ] are Ceil(Log2( projection\_plane\_width\_minus1[view\_id[ a ][ p ] + 1) ) and Ceil(Log2( projection\_plane\_height\_minus1[view\_id[ a ][ p ] + 1) ) bits, respectively.

**patch\_pos\_in\_atlas\_x**[ a ][ p ] and **patch\_pos\_in\_atlas\_y**[ a ][ p ] specify the horizontal and vertical coordinates in luma samples, respectively, of the top-left corner of the p-th patch of the a-th atlas. The number of bits used for the representation of patch\_pos\_in\_atlas\_x[ a ][ p ] and patch\_pos\_in\_atlas\_y[ a ][ p ] are Ceil ( Log2( atlas\_width\_minus1[ a ] + 1 ) ) and Ceil( Log2( atlas\_height\_minus1[ a ] + 1 ) ), bits respectively.

**patch\_pos\_in\_view\_x** [ a ][ p ] and **patch\_pos\_in\_view\_y** [ a ][ p ] specify the horizontal and vertical coordinates in luma samples, respectively, of the top-left corner of the i-th patch in the view\_id[ a ][ p ]-th view. The number of bits used for the representation of patch\_pos\_in\_view\_x[ a ][ p ] and patch\_pos\_in\_view\_y[ a ][ i ] are Ceil( Log2( projection\_plane\_width\_minus1[ view\_id[ a ][ p ]] + 1) ) and Ceil Log2( projection\_plane\_height\_minus1[view\_id[ a ][ p ]] + 1) ) bits, respectively.

**patch\_rotation**[ a ][ p ] indicates rotation and mirror of the p-th patch in the a-th atlas relative to the orientation of the patch in the view\_id[ a ][ p ]-th view, as defined in.. Table 7‑1 and illustrated in Fig 3.

Table 7‑1 coding of patch\_rotation (3 bits)

|  |  |  |  |
| --- | --- | --- | --- |
| **patch\_rotation** | **Name** | **Mirror** | **Rotation** |
| 0 | NULL | none | 0° |
| 1 | SWAP | Y axis | 270° |
| 2 | ROT90 | none | 90° |
| 3 | ROT180 | none | 180° |
| 4 | ROT270 | none | 270° |
| 5 | MIRROR | Y axis | 180° |
| 6 | MROT90 | Y axis | 90° |
| 7 | MROT180 | Y axis | 0° |

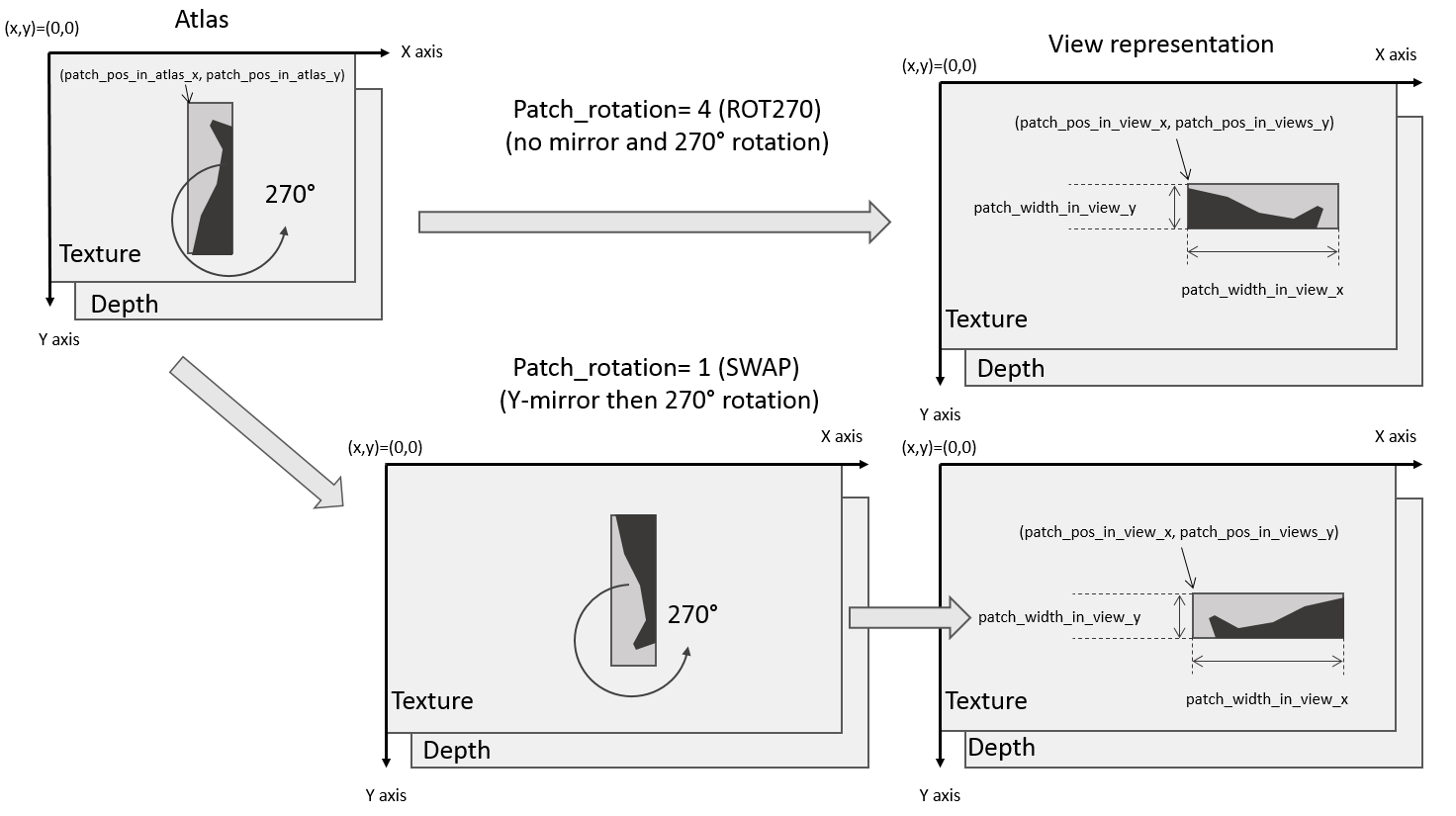


Figure 3: two patch rotation examples at the decoder with patch\_rotation= 4 (ROT270) and 1 (SWAP)

NOTE – The following process may be used to map sample positions of an atlas patch to corresponding sample positions in a view. A particular sample position in a view representation may have zero, one, or more than one valid corresponding atlas patch samples. It is outside the scope of this specification to determine how a renderer or 3D scene generator utilizes samples in a view representation. [Ed.(BK): I agree with that, but we have to define how to unproject points. Otherwise the camera parameters have no purpose. Please move this to Section 8 and let the outputs be RowInView and ColumnInView instead of ViewSample.]

Let p be the patch index and a be the atlas index. Let AtlasSample[ a ][ i ][ j ] be the sample value at the (i, j) position of a decoded picture of a component of the a-th atlas. Let ViewSample[ v ][ i ][ j ] be the sample value at the (i, j) position of the picture of the same component of the v-th view.

for( i = 0; i < patch\_width\_in\_view[a][p]; i++ ) {

for( j = 0; j < patch\_height\_in\_view[a][p];  j++ ) {

if( patch\_rotation[ a ][ p ] = = 0) { // NULL

xPos = patch\_pos\_in\_atlas\_x [ a ][ p ] + i

yPos = patch\_pos\_in\_atlas\_y [ a ][ p ] + j }

else if ( patch\_rotation[a][p] == 1) {// SWAP

xPos = patch\_pos\_in\_atlas\_x [ a ][ p ] + j

yPos = patch\_pos\_in\_atlas\_y [ a ][ p ] +  i

}

else if ( patch\_rotation[a][p] == 2) { // ROT90

xPos = patch\_pos\_in\_atlas\_x [ a ][ p ] + patch\_height\_in\_view[ a ][  p ]  -j-1

yPos = patch\_pos\_in\_atlas\_y [ a ][ p ] +  i

else if ( patch\_rotation[a][p] == 3) { // ROT180

xPos = patch\_pos\_in\_atlas\_x [ a ][ p ] + patch\_width\_in\_view[ a ][  p ]  -i-1

yPos = patch\_pos\_in\_atlas\_y [ a ][ p ] +  patch\_height\_in\_view[ a ][  p ] -j-1

}

else if ( patch\_rotation[a][p] == 4) { // ROT270

xPos = patch\_pos\_in\_atlas\_x [ a ][ p ] +  j

yPos = patch\_pos\_in\_atlas\_y [ a ][ p ] +  patch\_width\_in\_view[ a ][  p ] -i-1

}

else if ( patch\_rotation[a][p] == 5) { // MIRROR

xPos = patch\_pos\_in\_atlas\_x [ a ][ p ] + patch\_width\_in\_view[ a ][  p ] -i-1

yPos = patch\_pos\_in\_atlas\_y [ a ][ p ] +  j

}

else if ( patch\_rotation[a][p] == 6) { // MROT90

xPos = patch\_pos\_in\_atlas\_x [ a ][ p ] + patch\_height\_in\_view[ a ][  p ] -j-1

yPos = patch\_pos\_in\_atlas\_y [ a ][ p ] +  patch\_width\_in\_view[ a ][  p ] -i-1

}

else if ( patch\_rotation[a][p] == 7) { // MROT180

xPos = patch\_pos\_in\_atlas\_x [ a ][ p ] + i

yPos = patch\_pos\_in\_atlas\_y [ a ][ p ] +  patch\_height\_in\_view[ a ][  p ] -j-1

}

xViewPos = patch\_pos\_in\_view\_x [ a ][ p ] + i

yViewPos = patch\_pos\_in\_view\_y [ a ][ p ] + j

ViewSample[ view\_id[ a ][ p ] ][ xViewPos ][ yViewPos ] = AtlasSample[ a ][ xPos ][ yPos ]

ColumnInView[ a ][ p ][ i ][ j ] = xPos

RowInView[ a ][ p ][ i ][ j ] = yPos

}

}

**~~depth\_occupancy\_present\_flag~~**~~[ a ][ i ] equal to 1 specifies that a depth\_occupancy syntax element for patch [ a ][ i ] is present in the atlas\_params syntax structure. In this case, the patch assumes the values of depth\_occ\_map\_threshold and depth\_start signalled in the depth\_occupancy syntax structure following the flag. A depth\_occupancy\_present\_flag equal to 0 specifies that no depth\_occupancy syntax element is present for patch [ a ][ i ], and the patch will assume the values specifed for the relevant view, as indicated by view\_id[ a ][ i ], in the most recently received camera\_params\_list syntax structure. [Ed. (JB): bad~~

#### Depth and occupancy semantics

**depth\_occ\_map\_threshold\_present\_flag** equal to 1 specifies that the depth\_occ\_map\_threshold[ a ][ p ] syntax element is present. depth\_treshold\_present\_flag equal to 0 specifies that the depth\_occ\_map\_threshold[ a ][ p ] syntax element is not present

**depth\_occ\_map\_threshold**[ a ][ p ] specifies the threshold below which a decoded depth/occupancy level is defined to be “non-occupied” for the p-th patch of the a-th atlas, as defined in Section 8.3. The number of bits of the syntax element is depth\_params\_num\_bits\_minus8 + 8 bits. When not present, the value of depth\_occ\_map\_threshold[ a ][ p ] is inferred to be equal to depth\_occ\_map\_threshold\_default[ view\_id ].

**depth\_start\_present** equal to 1 specifies that the depth\_start syntax element is present. depth\_start\_presentequal to 0 specifies that the depth\_start syntax element is not present.

**depth\_start**[ a ][ p ] specifies the start of the valid range of depth values for the p-th patch of the a-th atlas, as described in Section 8.3. The number of bits of the syntax element is depth\_params\_num\_bits\_minus8 + 8 bits. When not present, the value of depth\_start[ a ][ p ] is inferred to be equal to depth\_start\_default\_present\_flag[ v ] ? depth\_start\_default[ view\_id[ a ][ p ] ] : 0.

### Viewing Space

[Ed. (JB): Move to an annex?]

#### Viewing\_space semantics

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

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

**elementary\_shape\_operation** [ e ]equal to 0specifies that the type of CSG operation to apply on the elementary shape e is additive. elementary\_shape\_operation [ e ]equal to 1specifies that the type of CSG operation to apply on the elementary shape e is substractive. The operation consists of computing a signed distance of a point p related to shape S and combining that with the signed distance of the entire accumulated viewing space. The order of the elementary shapes does not matter, but each elementary shape should be identified to be operated either additively or substractively.

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

#### Elementary shape semantics

**num\_primitive\_shapes\_minus1**[ e ] plus 1 specifies the number of primitive shapes that is used in the construction of the elementary shape e.

**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. primitive\_shape\_operation[ e ] equal to 1 specifies the interpolative mode, in which the the primitive shapes in the list are interpolated along a path defined by the ordered centroids of the primitive shape.

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

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

**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. primitive\_orientation\_present\_flag equal to 0 specifies that per-primitive orientation information is not present, and that the primitives are axis-aligned.

**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. viewing\_direction\_constraint\_present\_flag [ e ] equal to 0 specifies that per-primitive viewing direction constraints are not present.

**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 2: primitive\_shape\_type

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

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

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

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

Equation 7‑1 : *position*\_*fading index (p)= clamp((SD(p)+guard\_band\_size[e][s]) / guard\_band\_size[e][s], 0, 1)*

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

**primitive\_shape\_yaw**[ e ] [ s ], **primitive\_shape\_pitch**[ e ][ s ] and **primitive\_shape\_roll**[ e ] [ s ] are 16-bit floating point value that gives respectively the yaw, pitch and roll component of a rotation to apply on the primitive shape s of the elementary shape e. When the operation is based on CSG, the rotation is applied about the centroid of the primitive *S* before applying the corresponding distance function *SD(p, S )*. The values are expressed in degree.

**guard\_band\_direction\_size**[ e ] [ s ] is a 16-bit floating-point value that specifies the width of the directional guard band for each primitive shape s of an elementary shape e . guard\_band\_present equal to 0 implies that the guard band directional\_size is implicitly 0. This parameter is expressed in degree.

**primitive\_shape\_viewing\_direction\_yaw\_center**[ e ][ s ] is a 16-bit fixed floating point value giving the center of the yaw range of suggested viewing directions for the primitive shape s . The value is expressed in degree.

**primitive\_shape\_viewing\_direction\_pitch\_center**[ e ][ s ] is a 16-bit fixed floating point value giving the center of the pitch range of suggested viewing directions for the primitive shape s of the elementary shape e. The value is expressed in degree.

**primitive\_shape\_viewing\_direction\_roll\_center**[ e ][ s ] is a 16-bit fixed floating point value giving the center of the roll range of suggested viewing directions for the primitive shape s of the elementary shape e. The value is expressed in degree.

**primitive\_shape\_viewing\_direction\_yaw\_center**[ e ][ s ] points to the center of suggested viewing direction range.The value is expressed in degree.

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

**primitive\_shape\_viewing\_direction\_pitch\_center**[ e ][ s ] points to the center of suggested viewing direction pitch range.The value is expressed in degree.

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

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

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

Equation 7‑2: *V(p) = ∑-SD(p, Si)Vi(p)/ ∑-SD(p, Si)*

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

Equation 7‑3: *V(p) = ( RD(p, Ss+1)Vs + RD(p, Ss)Vs+1  ) / ( RD(p, S)+ RD(p, Ss+1))*

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

Equation 7‑4: *yaw fading index (p)= clamp((abs(yaw - viewing\_yaw\_center (p) )- viewing\_yaw\_range (p) + guard\_band\_direction\_size (p)) / guard\_band\_direction\_size (p), 0, 1)*

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

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

#### Cuboid primitive semantics

**center\_x**[ e ][ s ], **center\_y**[ e ][ s ], **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.

**size\_x**[ s ][ e ], **size\_y**[ s ][ e ], **size\_z**[ s ][ e ] is a 16-bit floating-point value that specifies the size of the cuboid in X, Y, Z directions in the scene coordinate system.

The signed distance function for a cuboid primitive is

Equation 7‑5: *SDCUBOID(p, l, h) =* min(max(*dx*, max(*dy, dz*)), *0*) + |max(*d, 0*)|

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

#### Spheroid\_primitive semantics

**center\_x**[ e ][ s ], **center\_y**[ e ][ s ], **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.

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

The signed distance function for a spheroid primitive is

Equation 7‑6: *SDSPHEROID(p, r) = |p/r| \* (|p/r| - 1) / |p/r2|*

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

#### Half space semantics

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

**distance**[ e ][ s ] is a 16-bit floating-point value that specifies the distance from the scene origin along the normal vector direction to the plane defining the half-space.

The signed distance function for a half-space primitive is

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

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

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

# Decoding process

## General decoding process

### General

Input to this process are the following:

* HEVC bitstream containing one or more layer pairs, each layer pair representing an atlas
* IV sequence parameters, including
  + camera parameters list
* IV access units parameters, including
  + sequence of atlas parameters lists

Outputs of this process are the following:

* one or more decoded picture sequence pairs representing an atlas, each pair containing
  + a texture picture sequence representing an atlas texture component
  + a depth picture sequence representing an atlas depth component
* one or more atlas patch occupancy map sequences

In this version of the specification, all layers in the HEVC bitstream shall have the same picture decoding order and alignment of CVSes. [Ed. (JB): Find a more proper way to describe this restriction.]

For each multi-layer aligned CVS in the HEVC bitstream, [Ed. (JB): Find a more proper way to describe this]

* Parse the IV sequence parameters
* For each IV access unit in the IV sequence
  + For each atlas in the IV access unit
    - The HEVC decoding process is invoked for the coded picture in the texture layer, with the texture decoded picture as output
    - The HEVC decoding process is invoked for the coded picture in the depth layer, with the depth decoded picture as output
    - The atlas patch occupancy map generator process in clause 8.2 is invoked, with the depth decoded picture and atlas parameters as inputs, and the atlas patch occupancy map as output

## Atlas patch occupancy map generator process

### General atlas patch occupancy map generator process

Inputs to this process are:

* a depth decoded picture of the a-th atlas
* IV sequence parameters
* atlas parameters for the a-th atlas

Output of this process is an atlas patch occupancy map for the a-th atlas, AtlasPatchOccupancyMap[ a ][ i ][ j ].

AtlasPatchOccupancyMap[ a ][ i ][ j ] is derived as follows.

Let DepthAtlasSample[ a ][ i ][ j ] be the sample value at the (i, j) position of the depth decoded picture of the a-th atlas. Let pictureWidth and pictureHeight, be the width and height, respectively, of the depth decoded picture of the a-th atlas.

The value of InvalidPatchId = 0xFFFF in a sample of AtlasPatchOccupancyMap[ a ][ i ][ j ] indicates an invalid value of a sample at the (i, j) position of the a-th atlas. Otherwise the value indicates the patch index of a valid sample value at the (i, j) position of the a-th atlas.

for( i = 0; i < pictureWidth; i++ )  
 for( j = 0; j < pictureHeight; j++ )  
 AtlasPatchOccupancyMap[ a ][ i ][ j ] = InvalidPatchId

for( p = 0; p <= num\_patches\_minus1[ a ]; p++)  
 if( (patch\_rotation[ a ][ p ]== 0) || (patch\_rotation[ a ][ p ]== 3) || ( patch\_rotation[ a ][ p ]== 5) || (patch\_rotation[ a ][ p ]== 7) )  
 patchWidthInAtlas = patch\_width\_in\_view[ a ] [ p ]  
 patchHeightInAtlas = patch\_height\_in\_view[ a ][ p ]   
 else  
 patchWidthInAtlas = patch\_height\_in\_view[ a ][ p ]   
 patchHeightInAtlas = patch\_width\_in\_view[ a ][ p ]

for( i = 0; i < patchWidthInAtlas; i++ )  
 for( j = 0; j < patchHeightInAtlas; j++ )   
 xPos = patch\_pos\_in\_atlas\_x[ a ][ p ] + i  
 yPos = patch\_pos\_in\_atlas\_y[ a ][ p ] + j  
 if( DepthAtlasSample[ a ][xPos ][ yPos ] >= depth\_occ\_map\_threshold[ a ][ p ] )   
 AtlasPatchOccupancyMap[ a ][ xPos ][ yPos ] = p

## Viewport position to global coordinate point unprojection process

### General viewport position to global coordinate point unprojection process

Inputs to this process are:

* a depth decoded picture of the a-th atlas
* IV sequence parameters
* atlas parameters for the a-th atlas

Output of this process is a global coordinate position map for the a-th atlas. GlobalCoordinatePositionMap[ a ][ i ][ j ] is a tuple of floating-point values (X, Y, Z) in the global coordinate system.

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

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

for( i = 0; i < pictureWidth; i++ )  
 for( j = 0; j < pictureHeight; j++ )  
 GlobalCoordinatePositionMap[ a ][ i ][ j ] = InvalidCoordinate  
 p = AtlasPatchOccupancyMap[ a ][ i ][ j ]  
 if (p != InvalidPatchId)  
 v = view\_id[ a ][ i ]  
 m = ColumnInView( a, p, i, j )  
 n = RowInView( a, p, i, j )  
 if (cam\_type[ v ] == 0)  
 GlobalCoordinatePositionMap[ a ][ i ][ j ]  
 = LocalToGlobal(v, UnprojectERP( m, n, v, Depth[ a ][ i ][ j ] ))  
 if (cam\_type[ v ] == 1)  
 GlobalCoordinatePositionMap[ a ][ i ][ j ]  
 = LocalToGlobal(v, UnprojectPSP( m, n, v, Depth[ a ][ i ][ j ] ))

Hereby RowInView( a, p, i, j ) and ColumnInView( a, p, i, j ) map the position ( i, j ) of patch p in atlas a to the position in view v = view\_id[ a ][ p ], as defined in Section 7.3.4.1, Depth[ a ][ i ][ j ] is defined in the depth decoding process (Section 8.3.4) and LocalToGlobal( v, (X, Y, Z) ) is derived as follows.

LocalToGlobal( v, (X,Y,Z) ) = EulerToRotMat( v ) \* (X, Y, Z) + CamPos[ v ]

Where EulerToRotMat[ v ] is a rotation matrix derived as follows:

EulerToRotMat[ v ] = RotZ[ v ] \* RotY[ v ] \* RotX[ v ]

With the following axis rotations.

### Depth decoding process

This process converts a DepthAtlasSample[ a ][ i ][ j ] to a floating-point depth value in meters Depth[ a ][ i ][ j ].

First a normalized depth value DepthAtlasNormValue[ a ][ i ][ j ] is derived as follows:

MaxDepthSampleValue = depth\_params\_num\_bits\_minus8 << 8 − 1

p = AtlasPatchOccupancyMap[ a ][ i ][ j ]

ClampedDepthSample[ a ][ i ][ j ] = Max( depth\_start[ a ][ p ], DepthAtlasSample[ a ][ i ][ j ] )

DepthAtlasNormValue[ a ][ i ][ j ] = ClampedDepthSample[ a ][ i ][ j ] ÷ MaxDepthSampleValue

Then normalized disparity, NormDisp[ a ][ i ][ j ], is derived as follows:.

v = view\_id[ a ][ p ]  
 NormDisp[ a ][ i ][ j ] = norm\_disp\_low[ v ] + (norm\_disp\_high[ v ] − norm\_disp\_low[ v ])  
 \* DepthAtlasNormValue[ a ][ i ][ j ]

Finally, Depth[ a ][ i ][ j ] is derived as follows:

Depth[ a ][ i ][ j ] = 1.0 ÷ NormDisp[ a ][ i ][ j ]

### ERP unprojection process

UnprojectERP( m, n, v, r ) with m the image column, n the image row, v the view\_id, and r a floating-point depth value in units of meters is defined as follows:

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

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

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

Phi( m, v ) = erp\_phi\_max[ v ] - (m + 0.5) \* (erp\_phi\_max[ v ] - erp\_phi\_min[ v ]) ÷ (projection\_plane\_width\_minus1 + 1)

Theta( m, v ) = erp\_theta\_max[ v ] - (m + 0.5) \* (erp\_theta\_max[ v ] - erp\_theta\_min[ v ]) ÷ (projection\_plane\_height\_minus1 + 1)

### Perspective unprojection process

UnprojectPSP( m, n, v, r ) with m the image column, n the image row, v the view\_id, and r a floating-point depth value in units of meters is derived as follows:

UnprojectPSP( m, n, v, r ) = (  
 Depth[ a ][ i ][ j ],  
 -(Depth[ a ][ i ][ j ] / perspective\_focal\_hor[ v ]) \* (m + 0.5 − perspective\_center\_hor[ v ]),  
 -(Depth[ a ][ i ][ j ] / perspective\_focal\_ver[ v ]) \* (n + 0.5 − perspective\_center\_ver[ v ])) )

# Parsing process

## General

Inputs to this process are bits from the bitstream

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v) or se(v) (see clause 9.2).

## Parsing process for 0-th order Exp-Golomb codes

### General

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v) or se(v).

Inputs to this process are bits from the bitstream.

Outputs of this process are syntax element values.

Syntax elements coded as ue(v) or se(v) are Exp-Golomb-coded. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first zero bit and counting the number of leading bits that are equal to 1. This process is specified as follows:

leadingZeroBits = −1  
for( b = 1; b; leadingZeroBits++ ) (9‑1)  
 b = read\_bits( 1 )

The variable codeNum is then assigned as follows:

codeNum = 2leadingZeroBits − 1 + read\_bits( leadingZeroBits ) (9‑2)

where the value returned from read\_bits( leadingZeroBits ) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

Table 9‑1 illustrates the structure of the Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits. The "prefix" bits are those bits that are parsed as specified above for the computation of leadingZeroBits and are shown as either 0 or 1 in the bit string column of Table 9‑1. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as xi in Table 9‑1, with i in the range of 0 to leadingZeroBits − 1, inclusive. Each xi is equal to either 0 or 1.

Table 9‑1 Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

|  |  |
| --- | --- |
| **Bit string form** | **Range of codeNum** |
| 0 | 0 |
| 1 0 x0 | 1..2 |
| 110 x1 x0 | 3..6 |
| 1110 x2 x1 x0 | 7..14 |
| 11110 x3 x2 x1 x0 | 15..30 |
| 111110 x4 x3 x2 x1 x0 | 31..62 |
| ... | ... |

Table 9‑2 illustrates explicitly the assignment of bit strings to codeNum values.

Table 9‑2 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v) (informative)

|  |  |
| --- | --- |
| **Bit string** | **codeNum** |
| 0 | 0 |
| 100 | 1 |
| 101 | 2 |
| 110 0 0 | 3 |
| 110 0 1 | 4 |
| 110 1 0 | 5 |
| 110 1 1 | 6 |
| 1110 0 0 0 | 7 |
| 1110 0 0 1 | 8 |
| 1110 0 1 0 | 9 |
| ... | ... |

Depending on the descriptor, the value of a syntax element is derived as follows:

– If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.

– Otherwise (the syntax element is coded as se(v)), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in clause 9.2.2 with codeNum as input.

### Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in clause 9.2.1.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. Table 9‑3 provides the assignment rule.

Table 9‑3 – Assignment of syntax element to codeNum for signed Exp-Golomb coded syntax elements se(v)

|  |  |
| --- | --- |
| **codeNum** | **syntax element value** |
| 0 | 0 |
| 1 | 1 |
| 2 | −1 |
| 3 | 2 |
| 4 | −2 |
| 5 | 3 |
| 6 | −3 |
| k | (−1)k + 1 Ceil( k ÷ 2 ) |

## Parsing process for truncated unary codes

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to tu(v).

Inputs to this process are bits from the RBSP and the maximum value maxVal.

Outputs of this process are syntax element values.

Syntax elements coded as tu(v) are truncated unary coded. The range of possible values for the syntax element is determined first. The range of this syntax element is 0 to maxVal inclusive, with maxVal being greater than or equal to 1. codeNum which is equal to the value of the syntax element is given by a process specified as follows:

codeNum = 0  
keepGoing = 1  
for(i = 0; i < maxVal && keepGoing; i++){  
 keepGoing = read\_bits( 1 ) (9‑3)  
 if( keepGoing )  
 codeNum ++  
}

## Parsing process for truncated binary codes

This process is invoked when the descriptor of a syntax element in the syntax tables in subclause 7.3 is equal to tb(v).

Inputs to this process are bits from the RBSP and the maximum value maxVal.

Outputs of this process are syntax element values.

Syntax elements coded as tb(v) are truncated binary coded. The range of possible values for the syntax element is determined first. The range of this syntax element is 0 to maxValinclusive, with maxVal being greater than or equal to 1. synVal which is equal to the value of the syntax element is given by a process specified as follows:

thVal = 1 << 8  
th = 8  
while( thVal <= maxVal ) {  
 th++  
 thVal <<= 1  
}  
th− −  
val = 1 << th (9‑4)  
b = n − val  
synVal = read\_bits( th )  
if( synVal >= val − b ) {  
 synVal <<= 1  
 synVal += read\_bits( 1 )  
 synVal −= val − b  
}

where the value returned from read\_bits( th ) is interpreted as a binary representation of an unsigned integer with most significant bit written first.

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

TBP.