ISO/IEC JTC 1/SC 29/WG 11 N18670

ISO/IEC JTC 1/SC 29/WG 11

Coding of moving pictures and audio

Document type:	**Approved WG 11 document**
Title:	**Text of ISO/IEC DIS 23090-5 Video-based Point Cloud Compression**
Status:	Approved
Date of document:	**2019-10-10**
Source:	**3DG**
Expected action:	
No. of pages:	**175**
Email of convenor:	leonardo@chiariglione.org
Committee URL:	**mpeg.chiariglione.org**

ISO/IEC 23090-5:2019(E)

ISO/IEC JTC 1/SC 29/WG 11

Secretariat: JISC

Information technology — Coded Representation of Immersive Media — Part 5: Video-based Point Cloud Compression

DIS stage

Contents

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).

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).

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.

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.

Introduction

Advances in 3D capturing and rendering technologies have unleashed a new wave of innovation in Virtual/Augmented/Mixed reality (VR/AR/MR) content creation and communication. Point clouds have arisen as one of the main representations for such applications. A point cloud frame consists of a set of individual 3D points. Each point, in addition to having a 3D position, i.e., spatial attribute, may also be associated with a number of other attributes such as colour, reflectance, surface normal, etc. A point cloud consists of a sequence of point cloud frames. The number of points, their positions, and their attributes may vary from one frame to another. Such representations require a large amount of data, which can be costly in terms of storage and transmission. Therefore, the ISO/IEC Moving Picture Experts Group (MPEG) developed a new Recommendation | International Standard, which aims at efficiently compressing point cloud representations.

Information technology — Coded Representation of Immersive Media — Part 5: Video-based Point Cloud Compression

1 Scope

This document specifies video-based point cloud compression.

2 Normative reference

There are no normative references in this document.

3 Terms and definitions

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

3.1
associated non-ACL NAL unit
a *non-ACL NAL unit* (when present) for an *ACL NAL unit* where the *ACL NAL unit* is the *associated ACL NAL unit* of the *non-ACL NAL unit*

3.2
associated ACL NAL unit
the preceding *ACL NAL unit* in *decoding order* for a *non-ACL NAL unit* with nal_unit_type equal to NAL_EOS, NAL_EOB, NAL_FD, or NAL_SUFFIX_SEI, or in the ranges of NAL_RSV_NACL_45..NAL_RSV_NACL_47 or NAL_UNSPEC_56..NAL_UNSPEC_63; or otherwise the next *ACL NAL unit* in *decoding order*

3.3
atlas
a collection of 2D bounding boxes, i.e. patches, projected into a rectangular frame that correspond to a 3-dimensional bounding box in 3D space, which may represent a subset of a point cloud

3.4
atlas frame
See atlas

3.5
atlas frame parameter set (AFPS)
a syntax structure containing syntax elements that apply to zero or more entire coded atlas frames as determined by the content of a syntax element found in the tile group header

3.6
atlas sequence
a collection of atlas frames

3.7
atlas sequence parameter set (ASPS)
a syntax structure containing syntax elements that apply to zero or more entire coded atlas sequences (CASs) as determined by the content of a syntax element found in the ASPS referred to by a syntax element found in each tile group header

3.8
atlas sub-bitstream
an extracted sub-bitstream from the V-PCC bitstream containing a part of an atlas *NAL bitstream*

3.9
attribute
scalar or vector property optionally associated with each point in a point cloud such as colour, reflectance, surface normal, time stamps, material ID, etc

3.10
attribute frame
a collection of attribute maps for a specific attribute that correspond to the same time instance

3.11
attribute map
a 2D rectangular array created through the aggregation of *patches* with *attribute* values for a specific attribute

3.12
bitstream
A sequence of bits of coded *V-PCC components*

3.13
byte
a sequence of 8 bits, within which, when written or read as a sequence of bit values, the left-most and right-most bits represent the most and least significant bits, respectively

3.14
byte-aligned
a position in a *bitstream* is byte-aligned when the position is an integer multiple of 8 bits from the position of the first bit in the *bitstream*, and a bit or *byte* or *syntax element* is said to be byte-aligned when the position at which it appears in a *bitstream* is byte-aligned

3.15
cartesian coordinates
three scalars (x, y, z) with finite precision and dynamic range that indicate the location of a point relative to a fixed reference point (the origin)

3.16
coded atlas
a coded representation of an *atlas*

3.17
coded atlas access unit
a set of atlas *nal units* that are associated with each other according to a specified classification rule, are consecutive in *decoding order*, and contain all atlas *nal_units* pertaining to one particular output time

3.18
coded atlas bitstream
sequence of bits that forms the representation of *atlas frames* and associated data forming one or more *CASs*

3.19
coded atlas frame
see *coded atlas*

3.20

coded atlas sequence (CAS)
a sequence of *coded atlas access units,* in decoding order, of an IRAP *coded atlas* AU, followed by zero or more *coded atlas* AUs that are not IRAP *coded atlas* AUs, including all subsequent *coded atlas* AUs up to but not including any subsequent *coded atlas* AU that is an IRAP *coded atlas* AU

3.21
coded point cloud frame
a collection of coded representations of an *atlas*, *occupancy map*, one or more *geometry maps*, and, for each available attribute, one or more *attribute maps*, pertaining to one particular time instance

3.22
coded point cloud sequence (CPCS)
sequence of V-PCC AUs that consists, in decoding order, of an IRAP V-PCC AU, followed by zero or more V-PCC AUs that are not IRAP V-PCC AUs, including all subsequent V-PCC AUs up to but not including any subsequent V-PCC AU that is an IRAP V-PCC AU

3.23
coded representation
a data element as represented in its coded form

3.24
coded V-PCC access unit
a coded representation of a V-PCC access unit

3.25
coded V-PCC component
a coded representation of a V-PCC component

3.26
collection of V-PCC sub-bitstream components
V-PCC sub-bitstream components that when decoded, they enable the reconstruction of a point cloud

3.27
flag
a variable or single-bit syntax element that can take one of the two possible values: 0 and 1

3.28
geometry
set of cartesian coordinates associated with a point cloud frame

3.29
geometry frame
a collection of geometry maps corresponding to the same time instance

3.30
geometry map
a 2D array created through the aggregation of the geometry information associated with each *patch*

3.31
instantaneous decoding refresh (IDR) coded atlas access unit
an access unit in which the coded atlas with nal_layer_id equal to 0 is an IDR or a GIDR coded atlas.

3.32
instantaneous decoding refresh (IDR) coded atlas
an IRAP coded atlas for which each ACL NAL unit has nal_unit_type equal to NAL_IDR_W_RADL, or NAL_IDR_N_LP, NAL_GIDR_W_RADL, or NAL_GIDR_N_LP.

NOTE – An IDR coded atlas does not refer to any atlases other than itself for inter prediction in its decoding process, and may be the first atlas in the bitstream in decoding order, or may appear later in the bitstream. Each IDR coded atlas is the first atlas of a CAS in decoding order. When an IDR coded atlas for which each ACL NAL unit has nal_unit_type equal to NAL_IDR_W_RADL or NAL_GIDR_W_RADL, it may have associated RADL coded atlases. When an IDR coded atlas for which each ACL NAL unit has nal_unit_type equal to NAL_IDR_N_LP or NAL_GIDR_N_LP, it does not have any associated leading coded atlases. An IDR coded atlas does not have associated RASL coded atlases.

3.33
instantaneous decoding refresh (IDR) V-PCC access unit
an access unit in which the coded atlas with nal_layer_id equal to 0 is an GIDR coded atlas.

3.34
instantaneous decoding refresh (IDR) V-PCC
an IRAP coded atlas for which each ACL NAL unit has nal_unit_type equal to NAL_GIDR_W_RADL or NAL_GIDR_N_LP.

NOTE – A GIDR coded atlas does not refer to any atlases other than itself for inter prediction in its decoding process, and may be the first atlas in the bitstream in decoding order, or may appear later in the bitstream. Each GIDR coded atlas is the first atlas of a CAS in decoding order. When an IDR coded atlas for which each ACL NAL unit has nal_unit_type equal to NAL_IDR_W_RADL, it may have associated RADL coded atlases. When an IDR coded atlas for which each ACL NAL unit has nal_unit_type equal to NAL_IDR_N_LP, it does not have any associated leading coded atlases. An IDR coded atlas does not have associated RASL coded atlases.

3.35
inter atlas tile group
an atlas tile group that is decoded using both intra or inter prediction methods

3.36
intra atlas tile group
an atlas tile group that is decoded using only intra and inter prediction methods

3.37
intra random access point (IRAP) coded atlas
A coded atlas for which each ACL NAL unit has nal_unit_type in the range of NAL_BLA_W_LP to NAL_IRAP_ACL_23, inclusive.

NOTE – An IRAP coded atlas does not refer to any coded atlases other than itself for inter prediction in its decoding process, and may be a BLA coded atlas, GBLA coded atlas, a CRA coded atlas, a GCRA coded atlas, an IDR coded atlas, or a GIDR coded atlas. The first coded atlas in the bitstream in decoding order must be an IRAP coded atlas. Provided the necessary parameter sets are available when they need to be activated, the IRAP coded atlas and all subsequent non-RASL coded atlas in decoding order can be correctly decoded without performing the decoding process of any coded atlases that precede the IRAP coded atlas in decoding order.

3.38
intra random access point (IRAP) coded atlas access unit
An access unit in which the coded atlas with nal_layer_id equal to 0 is an IRAP coded atlas.

intra random access point (IRAP) V-PCC unit
A V-PCC unit for which each ACL NAL unit has nal_unit_type in the range of NAL_GBLA_W_LP to NAL_GBLA_N_LP, or is in the range of NAL_GIDR_W_RADL to NAL_GIDR_N_LP, or is equal to NAL_GCRA, inclusive.

3.39
intra random access point (IRAP) V-PCC access unit
An access unit that corresponds to an IRAP V-PCC unit.

3.40
may
a term that is used to refer to behaviour that is allowed, but not necessarily required

NOTE: In some places where the optional nature of the described behaviour is intended to be emphasized, the phrase "may or may not" is used to provide emphasis.

3.41
must
a term that is used in expressing an observation about a requirement or an implication of a requirement that is specified elsewhere in this Specification (used exclusively in an informative context)

3.42
network abstraction layer (NAL) unit
syntax structure containing an indication of the type of data to follow and bytes containing that data in the form of an RBSP

3.43
network abstraction layer (NAL) unit stream
sequence of *NAL units*

3.44
occupancy map
a 2D array corresponding to an atlas whose values indicate for each sample position in the atlas whether that position corresponds to a valid 3D point in the point cloud representation

3.45
patch
rectangular region within an *atlas* that corresponds to a rectangular region within a *planar projection*

3.46
patch data
data that is needed to perform the transformation of patches included in an atlas from 2D to 3D space

3.47
planar projection
2D sample arrays of *attributes* and a corresponding *geometry* representing the projection of a *point cloud frame* onto a planar surface

3.48
point cloud
set of 3D points specified by their cartesian coordinates (x,y,z) and zero or more corresponding attributes at a particular time instance

3.49
point cloud frame
see *point cloud*

3.50
point cloud sequence
sequence of *point cloud frames*

3.51
prefix SEI message
an SEI message that is contained in a prefix SEI atlas NAL unit

3.52
prefix SEI atlas NAL unit
an SEI atlas NAL unit that has adg_unit_type equal to ADG_PREFIX_SEI

3.53

profile
a specified subset of the syntax of this Specification as well as of the syntax and usage of any associated specifications

3.54
profile component
a specified subset of the syntax of this Specification applying to either point cloud decoding or reconstruction

3.55
random access
the act of starting the decoding process for a *bitstream* at a point other than the beginning of the stream

3.56
raw byte sequence payload (RBSP)
a *syntax structure* containing an integer number of *bytes* that is encapsulated in a *NAL unit* and that is either empty or has the form of a *string of data bits* containing *syntax elements* followed by an *RBSP stop bit* and zero or more subsequent bits equal to 0

3.57
raw byte sequence payload (RBSP) stop bit
A bit equal to 1 present within a *raw byte sequence payload (RBSP)* after a *string of data bits*, for which the location of the end within an *RBSP* can be identified by searching from the end of the *RBSP* for the *RBSP stop bit*, which is the last non-zero bit in the *RBSP*

3.58
raw patch data

3.59
reserved
a term that may be used to specify that some values of a particular syntax element are for future use by ISO/IEC and shall not be used in bitstreams conforming to this version of this Specification, but may be used in bitstreams conforming to future extensions of this Specification by ISO/IEC

3.60
v-pcc parameter set (VPS)
a syntax structure containing syntax elements that apply to zero or more entire CPCSs as determined by the content of a syntax element found in the VPS referred to by a syntax element found in the V-PCC unit header

3.61
shall
a term used to express mandatory requirements for conformance to this Specification

NOTE: When used to express a mandatory constraint on the values of syntax elements or on the results obtained by operation of the specified decoding process, it is the responsibility of the encoder to ensure that the constraint is fulfilled.

3.62
should
a term used to refer to behaviour of an implementation that is encouraged to be followed under anticipated ordinary circumstances, but is not a mandatory requirement for conformance to this Specification

3.63
skip patch data

3.64

suffix SEI message
an SEI message that is contained in a suffix SEI NAL unit

3.65
suffix SEI NAL unit
an SEI atlas nal unit that has nal_unit_type equal to NAL_SUFFIX_SEI

3.66
supplemental enhancement information (SEI)
a NAL unit that has nal_unit_type equal to NAL_PREFIX_SEI or NAL_SUFFIX_SEI

3.67
syntax element
an element of data represented in the bitstream

3.68
syntax structure
zero or more syntax elements present together in the bitstream in a specified order

3.69
tile
a rectangular region of an *atlas*

3.70
tile group
a group of *tiles* ...

3.71
unspecified
a term that may be used to specify some values of a particular syntax element to indicate that the values have no specified meaning in this Specification and will not have a specified meaning in the future as an integral part of future versions of this Specification

3.72
video bitstream
a bitstream conforming to a video specification that may represent a V-PCC component as specified by this Specification

3.73
video data unit
syntax structure containing video data information, such as a NAL unit in the context of HEVC or AVC

3.74
video sub-bitstream
an extracted sub-bitstream from the V-PCC bitstream containing a part of a *video bitstream*

3.75
V-PCC access unit
a set of *V-PCC units* that are associated with each other according to a specified classification rule, and are consecutive in *decoding order*, and contain all atlas nal_units as well as other V-PCC components pertaining to one particular output time

3.76
V-PCC bitstream
sequence of bits that forms the representation of *coded point cloud frames* and associated data forming one or more *CPCSs*

3.77
V-PCC component
an *atlas*, *occupancy map*, *geometry,* or *attribute* of a particular type that is associated with a V-PCC point cloud representation

3.78
V-PCC component frame
a single V-PCC component pertaining to one particular output time

3.79
V-PCC sub-bitstream
a sub-bitstream of a V-PCC bitstream corresponding to a V-PCC component

3.80
V-PCC unit
a *syntax structure* containing an *V-PCC unit header* and a *V-PCC unit payload*

3.81
V-PCC unit header
a *syntax structure* containing an indication of the type of data to follow

3.82
V-PCC unit payload
a *syntax structure* containing the *bytes* containing V-PCC sub-bitstream data

4 Abbreviations

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

2D	Two-Dimensional
3D	Three-Dimensional
ACL	Atlas Coding Layer
AFPS	Atlas Frame Parameter Set
ASPS	Atlas Sequence Parameter Set
AU	Access Unit
CAS	Coded Atlas Sequence
CPCS	Coded Point Cloud Sequence
DAFB	Decoded Atlas Frame Buffer
EOM	Enhanced Occupancy Map
I	Intra
IDR	Instantaneous Decoding Refresh
IRAP	Intra Random Access Point
LSB	Least Significant Bit

MSB	Most Significant Bit
NAL	Network Abstraction Layer
P	Predictive
PCC	Point Cloud Compression
PLR	Point Local Reconstruction
RBSP	Raw Byte Sequence Payload
SEI	Supplemental Enhancement Information
SODB	String of Data Bits
VPS	V-PCC Parameter Set
V-PCC	Video-based PCC

5 Conventions

5.1 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.

5.2 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
x^y	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.
$\frac{x}{y}$	Used to denote division in mathematical equations where no truncation or rounding is intended.
$\sum_{i=x}^{y} f(i)$	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.

5.3 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.

5.4 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.

5.5 Bit-wise operators

The following bit-wise operators are defined as follows:

~	Bit-wise "not". 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 "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.

5.6 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).

5.7 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.

5.8 Mathematical functions

The following mathematical functions are defined:

$$\text{Abs}(x) = \begin{cases} x & ; \quad x >= 0 \\ -x & ; \quad x < 0 \end{cases} \tag{5-1}$$

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

$$\text{Clip1}_Y(x) = \text{Clip3}(0, (1 << \text{BitDepth}_Y) - 1, x) \tag{5-3}$$

$$\text{Clip1}_C(x) = \text{Clip3}(0, (1 << \text{BitDepth}_C) - 1, x) \tag{5-4}$$

$$\text{Clip3}(x, y, z) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases} \tag{5-5}$$

Floor(x) the largest integer less than or equal to x. (5-6)

Log2(x) the base-2 logarithm of x. (5-7)

Log10(x) the base-10 logarithm of x. (5-8)

$$\text{Min}(x, y) = \begin{cases} x & ; \quad x <= y \\ y & ; \quad x > y \end{cases} \tag{5-9}$$

$$\text{Max}(x, y) = \begin{cases} x & ; \quad x >= y \\ y & ; \quad x < y \end{cases} \tag{5-10}$$

$$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5) \tag{5-11}$$

$$\text{Sign}(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x = 0 \\ -1 & ; \quad x < 0 \end{cases} \tag{5-12}$$

Sqrt(x) the square root of x (5-13)

5.9 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)
x^y
"x * y", "x / y", "x ⍰ y"⍰⍰"$\frac{x}{y}$", "x % y"
"x + y", "x − y" (as a two-argument operator), "$\sum_{i=x}^{y} f(i)$"
"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"

5.10 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 clause 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 clause 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 s_{yx}. 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.

5.11 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

5.12 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.

6 Bitstream format, partitioning, and scanning processes

6.1 V-PCC bitstream formats

This clause specifies the relationship between the V-PCC unit stream format and the V-PCC sample stream, either of which are referred to as the V-PCC bitstream. This clause is not an essential component of this Specification and all V-PCC components including any associated V-PCC VPSs could be encapsulated using a different format depending on application.

The bitstream can be in one of two formats: the V-PCC unit stream format or the sample stream format. The V-PCC unit stream format is conceptually the more "basic" type. It consists of a sequence of syntax structures called V-PCC units. This sequence is ordered in decoding order. There are constraints imposed on the decoding order (and contents) of the V-PCC units in the V-PCC unit stream.

Note: The V-PCC unit stream format is commonly not intended to be used in any applications on its own since it requires additional information, i.e. sub-bitstream size information, for decoding its associated sub-bitstreams. One method of achieving this is through the use of the sample stream format.

The sample stream format can be constructed from the V-PCC unit stream format by ordering the V-PCC units in decoding order and prefixing each V-PCC unit with a heading that specifies the exact size, in bytes, of the V-PCC unit. A sample stream header is included at the beginning of the sample stream bitstream that specifies the precision, in bytes, of the signaled V-PCC unit size. The V-PCC unit stream format can be extracted from the sample stream format by traversing through the sample stream format, reading the size information and appropriately extracting each V-PCC unit. Methods of framing V-PCC units in a manner other than the use of the sample stream format are outside the scope of this Specification. The sample stream format is specified in Annex B.

6.2 NAL bitstream formats

This clause specifies the relationship between the network abstraction layer (NAL) unit stream and the NAL sample stream, either of which are referred to as the NAL bitstream.

The bitstream can be in one of two formats: the NAL unit stream format or the sample stream format. The NAL unit stream format is conceptually the more "basic" type. It consists of a sequence of syntax structures called NAL units. This sequence is ordered in decoding order, as described in clause 7.4.5.3.

There are constraints imposed on the decoding order (and contents) of the NAL units in the NAL unit stream.

Note: The NAL unit stream format is commonly not intended to be used in any applications on its own since it requires additional information, i.e. sub-bitstream size information, for decoding its associated sub-bitstreams. One method of achieving this is through the use of the NAL sample stream format.

The NAL sample stream format can be constructed from the NAL unit stream format by ordering the NAL units in decoding order and prefixing each NAL unit with a heading that specifies the exact size, in bytes, of the NAL unit. A sample stream header is included at the beginning of the sample stream bitstream that specifies the precision, in bytes, of the signaled NAL unit size. The NAL unit stream format can be extracted from the sample stream format by traversing through the sample stream format, reading the size information and appropriately extracting each NAL unit. Methods of framing NAL units in a manner other than the use of the sample stream format are outside the scope of this Specification. The sample stream format is specified in Annex C.

6.3 Partitioning of atlas frames, tiles, and tile groups

6.3.1 Partitioning of atlas frames into tiles and tile groups

This subclause specifies how an atlas frame is partitioned into tiles and tiles groups.

An atlas frame is divided into one or more tile rows and one or more tile columns. A tile is a rectangular region of an atlas frame.

A tile group contains a number of tiles of an atlas frame.

Only rectangular tile groups are supported. In this mode, a tile group contains a number of tiles of an atlas frame that collectively form a rectangular region of the atlas frame.

Figure 6-1 shows an example tile group partitioning of an atlas frame, where the atlas frame is divided into 24 tiles (6 tile columns and 4 tile rows) and 9 rectangular tile groups.

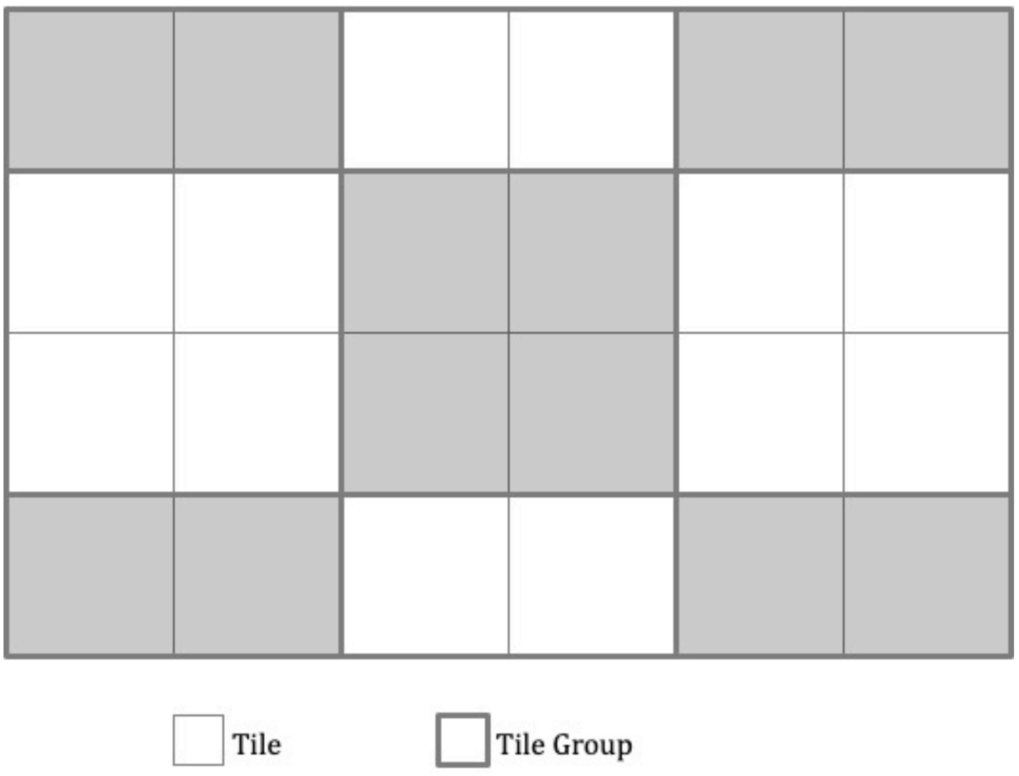

Figure 6-1 – An atlas frame that is partitioned into 24 tiles and 9 tile groups (informative)

6.4 Scanning Processes

6.4.1 Tile scanning process

The list ColWidth[i] for i ranging from 0 to afti_num_tile_columns_minus1, inclusive, specifying the width of the i-th tile column in units of 64 samples, is derived, and, when afti_uniform_tile_spacing_flag is equal to 1, the value of afti_num_tile_columns_minus1 is inferred, as follows:

```
if( afti_uniform_tile_spacing_flag ) {
    remainingWidthInCtbsY = ( asps_frame_width + 63 ) / 64
    i = 0
    while( remainingWidthInCtbsY > ( afti_tile_cols_width_minus1 + 1 ) ) {
        ColWidth[ i++ ] = afti_tile_cols_width_minus1 + 1
        remainingWidthInCtbsY −= ( afti_tile_cols_width_minus1 + 1 )
    }
    ColWidth[ i ] = remainingWidthInCtbsY
    afti_num_tile_columns_minus1 = i
} else {
    ColWidth[ afti_num_tile_columns_minus1 ] = ( asps_frame_width + 63 ) / 64        (6-1)
    for( i = 0; i < afti_num_tile_columns_minus1; i++ ) {
        ColWidth[ i ] = afti_tile_column_width_minus1[ i ] + 1
        ColWidth[ afti_num_tile_columns_minus1 ] −= ColWidth[ i ]
    }
}
```

The list RowHeight[j] for j ranging from 0 to afti_num_tile_rows_minus1, inclusive, specifying the height of the j-th tile row in units of 64 samples, is derived, and, when afti_uniform_tile_spacing_flag is equal to 1, the value of afti_num_tile_rows_minus1 is inferred, as follows:

```
if( afti_uniform_tile_spacing_flag ) {
    remainingHeightInCtbsY = ( asps_frame_height + 63 ) / 64
```

```
        i = 0
        while( remainingHeightInCtbsY > ( afti_tile_rows_height_minus1 + 1 ) ) {
            RowHeight[ i++ ] = afti_tile_rows_height_minus1 + 1
            remainingHeightInCtbsY  −=  ( afti_tile_rows_height_minus1 + 1 )
        }
        RowHeight[ i ] = remainingHeightInCtbsY
        afti_num_tile_rows_minus1 = i
    } else {
        RowHeight[ afti_num_tile_rows_minus1 ] = ( asps_frame_height + 63 ) / 64                (6-2)
        for( j = 0; j < afti_num_tile_rows_minus1; j++ ) {
            RowHeight[ j ] = afti_tile_row_height_minus1[ j ] + 1
            RowHeight[ afti_num_tile_rows_minus1 ]  −=  RowHeight[ j ]
        }
    }
```

The list TileColBd[i] for i ranging from 0 to afti_num_tile_columns_minus1 + 1, inclusive, specifying the location of the i-th tile column boundary in units of 64 samples, is derived as follows:

```
    for( TileColBd[ 0 ] = 0, i = 0; i  <=  afti_num_tile_columns_minus1; i++ )
        TileColBd[ i + 1 ] = TileColBd[ i ] + ColWidth[ i ]                                   (6-3)
```

The list TileRowBd[j] for j ranging from 0 to afti_num_tile_rows_minus1 + 1, inclusive, specifying the location of the j-th tile row boundary in units of 64 samples, is derived as follows:

```
    for( TileRowBd[ 0 ] = 0, j = 0; j  <=  afti_num_tile_rows_minus1; j++ )
        TileRowBd[ j + 1 ] = TileRowBd[ j ] + RowHeight[ j ]                                  (6-4)
```

7 Syntax and semantics

7.1 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.

	Descripto r
/* 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	

7.2 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_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 C for applications that use the sample 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.

more_data_in_vpcc_unit( ) is specified as follows:

– If more data follow in the current vpcc_unit, i.e. the decoded data up to now in the current vpcc_unit is less than numBytesInVPCCInit, the return value of more_data_in_vpcc_unit( ) is equal to TRUE.

– Otherwise, the return value of more_data_in_vpcc_unit( ) 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.

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.

afrm_layer_id(frameX) returns the value of the nal_layer_id of the ACL NAL units in the atlas frame frameX.

read_bits(n) reads the next n bits from the bitstream and advances the bitstream pointer by n bit positions. When n is equal to 0, read_bits(n) is specified to return a value equal to 0 and to not advance the bitstream pointer.

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).

– 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. In particular, the parsing process for this descriptor is specified as follows:

```
i(n) {
    value = read_bits( n )
    if( value < ( 1 << ( n – 1 )))
        return value
    else
        return ( value | ~(( 1 << (n – 1)) – 1 )
}
```

– 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 clause 10.2.

– st(v): null-terminated string encoded as UTF-8 characters as specified in ISO/IEC 10646. The parsing process is specified as follows: st(v) begins at a byte-aligned position in the bitstream and reads and returns a series of bytes from the bitstream, beginning at the current position and continuing up to but not including the next byte-aligned byte that is equal to 0x00, and advances the bitstream pointer by (stringLength + 1) * 8 bit positions, where stringLength is equal to the number of bytes returned.

NOTE The st(v) syntax descriptor is only used in this document when the current position in the bitstream is a byte-aligned position.

– 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 the most significant bit written first.

– 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 clause 10.2.

7.3 Syntax in tabular form

7.3.1 General

The V-PCC bitstream is composed of a set of V-PCC units as shown in Figure 7-1. Each V-PCC unit has a V-PCC unit header and a V-PCC unit payload. The V-PCC unit header describes the V-PCC unit type. Table 7-1 describes the supported V-PCC unit types.

The Attribute Video Data V-PCC unit header specifies also the attribute type and its index, allowing multiple instances of the same attribute type to be supported. Table 7-2 specifies the supported V-PCC attribute types.

The occupancy, geometry, and attribute Video Data unit payloads correspond to video data units (e.g., HEVC NAL unit) that could be decoded by an appropriate video decoder.

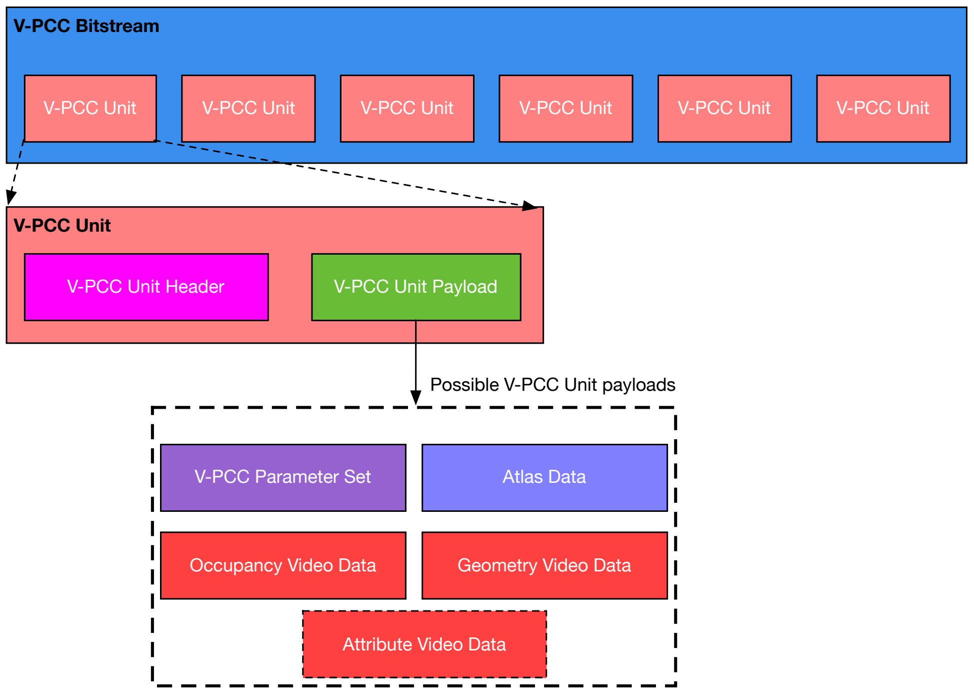

Figure 7-1 V-PCC Bitstream structure

7.3.2 V-PCC unit syntax

7.3.2.1 General V-PCC unit syntax

vpcc_unit(numBytesInVPCCUnit) {	**Descriptor**
vpcc_unit_header( )	
vpcc_unit_payload( )	
while(more_data_in_vpcc_unit)	
trailing_zero_8bits /* equal to 0x00 */	f(8)
}	

7.3.2.2 V-PCC unit header syntax

vpcc_unit_header( ) {	**Descriptor**
vuh_unit_type	u(5)

if(vuh_unit_type = = VPCC_AVD || vuh_unit_type = = VPCC_GVD ||
        vuh_unit_type = = VPCC_OVD || vuh_unit_type = = VPCC_AD) {	

vuh_vpcc_parameter_set_id	u(4)
vuh_atlas_id	u(6)
}	
if(vuh_unit_type = = VPCC_AVD) {	
vuh_attribute_index	u(7)
vuh_attribute_dimension_index	u(5)
vuh_map_index	u(4)
vuh_raw_video_flag	u(1)
} else if(vuh_unit_type = = VPCC_GVD) {	
vuh_map_index	u(4)
vuh_raw_video_flag	u(1)
vuh_reserved_zero_12bits	u(12)
} else if(vuh_unit_type = = VPCC_OVD || vuh_unit_type = = VPCC_AD)	
vuh_reserved_zero_17bits	u(17)
else	
vuh_reserved_zero_27bits	u(27)
}	

7.3.2.3 V-PCC unit payload syntax

vpcc_unit_payload( ) {	**Descriptor**
if(vuh_unit_type = = VPCC_VPS)	
vpcc_parameter_set( )	
else if(vuh_unit_type = = VPCC_AD)	
atlas_sub_bitstream( )	
else if(vuh_unit_type = = VPCC_OVD || vuh_unit_type = = VPCC_GVD || vuh_unit_type = = VPCC_AVD)	
video_sub_bitstream( )	
}	

7.3.3 Byte alignment syntax

byte_alignment( ) {	**Descriptor**
alignment_bit_equal_to_one /* equal to 1 */	f(1)
while(!byte_aligned( ))	
alignment_bit_equal_to_zero /* equal to 0 */	f(1)
}	

7.3.4 Sequence parameter set syntax

7.3.4.1 General V-PCC parameter set syntax

vpcc_parameter_set( ) {	**Descriptor**
profile_tier_level()	
vps_vpcc_parameter_set_id	u(4)
vps_atlas_count_minus1	u(6)
for(j = 0; j < vps_atlas_count_minus1 + 1; j++) {	
vps_frame_width[j]	u(16)
vps_frame_height[j]	u(16)

vps_map_count_minus1[j]	u(4)
if(vps_map_count_minus1[j] > 0)	
vps_multiple_map_streams_present_flag[j]	u(1)
vps_map_absolute_coding_enabled_flag[j][0] = 1	
for(i = 1; i <= vps_map_count_minus1[j]; i++) {	
if(vps_multiple_map_streams_present_flag[j])	
vps_map_absolute_coding_enabled_flag[j][i]	u(1)
else	
vps_map_absolute_coding_enabled_flag[j][i] = 1	
if(vps_map_absolute_coding_enabled_flag[j][i] = = 0) {	
if(i > 0)	
vps_map_predictor_index_diff[j][i]	ue(v)
else	
vps_map_predictor_index_diff[j][i] = 0	
}	
}	
vps_raw_patch_enabled_flag[j]	u(1)
if(vps_raw_patch_enabled_flag[j])	
vps_raw_separate_video_present_flag[j]	u(1)
occupancy_information(j)	
geometry_information(j)	
attribute_information(j)	
}	
vps_extension_present_flag	u(1)
if(vps_extension_present_flag) {	
vps_extension_length	ue(v)
for(j = 0; j < vps_extension_length + 1; j++) {	
vps_extension_data_byte	u(8)
}	
}	
byte_alignment( )	
}	

7.3.4.2 Profile, tier, and level syntax

profile_tier_level( ) {	**Descriptor**
ptl_tier_flag	u(1)
ptl_profile_codec_group_idc	u(7)
ptl_profile_pcc_toolset_idc	u(8)
ptl_profile_reconstruction _idc	u(8)
ptl_reserved_zero_32bits	u(32)
ptl_level_idc	u(8)
}	

7.3.4.3 Occupancy information syntax

occupancy_information(atlasId) {	**Descriptor**
oi_occupancy_codec_id[atlasId]	u(8)
oi_lossy_occupancy_map_compression_threshold[atlasId]	u(8)

oi_occupancy_nominal_2d_bitdepth_minus1[atlasId]	u(5)
oi_occupancy_MSB_align_flag[atlasId]	u(1)
}	

7.3.4.4 Geometry information syntax

geometry_information(atlasId) {	**Descriptor**
gi_geometry_codec_id[atlasId]	u(8)
gi_geometry_nominal_2d_bitdepth_minus1[atlasId]	u(5)
gi_geometry_MSB_align_flag[atlasId]	u(1)
gi_geometry_3d_coordinates_bitdepth_minus1[atlasId]	u(5)
if(vps_raw_separate_video_present_flag[atlasId])	
gi_raw_geometry_codec_id[atlasId]	u(8)
}	

7.3.4.5 Attribute information syntax

attribute_information(atlasId) {	**Descriptor**
ai_attribute_count[atlasId]	u(7)
for(i = 0; i < ai_attribute_count[atlasId]; i++) {	
ai_attribute_type_id[atlasId][i]	u(4)
ai_attribute_codec_id[atlasId][i]	u(8)
if(vps_raw_separate_video_present_flag[atlasId])	
ai_raw_attribute_codec_id[atlasId][i]	u(8)
for(j = 0; j < vps_map_count_minus1[atlasId]; j++)	
if (vps_map_absolute_coding_enabled_flag[atlasId][j] == 0)	
ai_attribute_map_absolute_coding_enabled_flag[atlasId][i][j]	u(1)
ai_attribute_dimension_minus1[atlasId][i]	u(6)
if(ai_attribute_dimension_minus1[atlasId][i] > 0) {	
ai_attribute_dimension_partitions_minus1[atlasId][i]	u(6)
remainingDimensions = ai_attribute_dimension_minus1[atlasId][i]	
k = ai_attribute_dimension_partitions_minus1[atlasId][i]	
for(j = 0; j < k; j++) {	
if(k – j == remainingDimensions)	
ai_attribute_partition_channels_minus1[atlasId][i][j] = 0	
else	
ai_attribute_partition_channels_minus1[atlasId][i][j]	ue(v)
remainingDimensions –= ai_attribute_partition_channels_minus1[atlasId][i][j] + 1	
}	
ai_attribute_partition_channels_minus1[atlasId][i][k] = remainingDimensions	
}	
ai_attribute_nominal_2d_bitdepth_minus1[atlasId][i]	u(5)
}	
if(ai_attribute_count[atlasId] > 0) {	
ai_attribute_MSB_align_flag[atlasId]	u(1)
}	
}	

7.3.5 NAL unit syntax

7.3.5.1 General NAL unit syntax

nal_unit(NumBytesInNalUnit) {	**Descriptor**
nal_unit_header( )	
NumBytesInRbsp = 0	
for(i = 2; i < NumBytesInNalUnit; i++)	
rbsp_byte[NumBytesInRbsp++]	b(8)
}	

7.3.5.2 NAL unit header syntax

nal_unit_header( ) {	**Descriptor**
nal_forbidden_zero_bit	f(1)
nal_unit_type	u(6)
nal_layer_id	u(6)
nal_temporaly_id_plus1	u(3)
}	

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

7.3.6.1 Atlas sequence parameter set RBSP

atlas_sequence_parameter_set_rbsp( ) {	**Descriptor**
asps_atlas_sequence_parameter_set_id	ue(v)
asps_frame_width	u(16)
asps_frame_height	u(16)
asps_log2_patch_packing_block_size	u(3)
asps_log2_max_atlas_frame_order_cnt_lsb_minus4	ue(v)
asps_max_dec_atlas_frame_buffering_minus1	ue(v)
asps_long_term_ref_atlas_frames_flag	u(1)
asps_num_ref_atlas_frame_lists_in_asps	ue(v)
for(i = 0; i < asps_num_ref_atlas_frame_lists_in_asps; i++)	
ref_list_struct(i)	
asps_use_eight_orientations_flag	u(1)
asps_45degree_projection_patch_present_flag	u(1)
asps_normal_axis_limits_quantization_enabled_flag	u(1)
asps_normal_axis_max_delta_value_enabled_flag	u(1)
asps_remove_duplicate_point_enabled_flag	u(1)
asps_pixel_deinterleaving_flag	u(1)
asps_patch_precedence_order_flag	u(1)
asps_patch_size_quantizer_present_flag	u(1)
asps_enhanced_occupancy_map_for_depth_flag	u(1)
asps_point_local_reconstruction_enabled_flag	u(1)
asps_map_count_minus1	u(4)
if(asps_enhanced_occupancy_map_for_depth_flag && asps_map_count_minus1 = = 0)	
asps_enhanced_occupancy_map_fix_bit_count_minus1	u(4)
if(asps_point_local_reconstruction_enabled_flag)	
asps_point_local_reconstruction_information(asps_map_count_minus1)	
if(asps_pixel_interleaving_flag || asps_point_local_reconstruction_enabled_flag)	

asps_surface_thickness_minus1	u(8)
asps_vui_parameters_present_flag	u(1)
if(asps_vui_parameters_present_flag)	
vui_parameters( )	
asps_extension_present_flag	u(1)
if(asps_extension_present_flag)	
while(more_rbsp_data( ))	
asps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.6.2 Point local reconstruction information syntax

asps_point_local_reconstruction_information(mapCountMinus1) {	**Descriptor**
for(i = 0; i < mapCountMinus1 + 1 ; i ++) {	
plri_point_local_reconstruction_map_enabled_flag[i]	u(1)
if(plri_point_local_reconstruction_map_enabled_flag[i]) {	
plri_number_of_modes_minus1[i]	u(4)
for(j = 0; j < plri_number_of_modes_minus1[i] + 1; j++) {	
plri_interpolate_flag[i][j]	u(1)
plri_filling_flag[i][j]	u(1)
plri_minimum_depth[i][j]	u(2)
plri_neighbour_minus1[i][j]	u(2)
}	
plri_block_threshold_per_patch_minus1[mapIdx]	u(6)
}	
}	
}	

7.3.6.3 Atlas frame parameter set RBSP syntax

atlas_frame_parameter_set_rbsp( ) {	**Descriptor**
afps_atlas_frame_parameter_set_id	ue(v)
afps_atlas_sequence_parameter_set_id	ue(v)
atlas_frame_tile_information( )	
afps_num_ref_idx_default_active_minus1	ue(v)
afps_additional_lt_afoc_lsb_len	ue(v)
afps_2d_pos_x_bit_count_minus1	u(4)
afps_2d_pos_y_bit_count_minus1	u(4)
afps_3d_pos_x_bit_count_minus1	u(5)
afps_3d_pos_y_bit_count_minus1	u(5)
afps_lod_bit_count	u(5)
afps_override_eom_for_depth_flag	u(1)
if(afps_override_eom_for_depth_flag) {	
afps_eom_number_of_patch_bit_count_minus1	u(4)
afps_eom_max_bit_count_minus1	u(4)
}	
afps_raw_3d_pos_bit_count_explicit_mode_flag	u(1)
afps_extension_present_flag	u(1)
if(afps_extension_present_flag)	

while(more_rbsp_data( ))	
afps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

7.3.6.4 Atlas frame tile information syntax

atlas_frame_tile_information( ) {	**Descriptor**
afti_single_tile_in_atlas_frame_flag	u(1)
if(!afti_single_tile_in_atlas_frame_flag) {	
afti_uniform_tile_spacing_flag	u(1)
if(afti_uniform_tile_spacing_flag) {	
afti_tile_cols_width_minus1	ue(v)
afti_tile_rows_height_minus1	ue(v)
} else {	
afti_num_tile_columns_minus1	ue(v)
afti_num_tile_rows_minus1	ue(v)
for(i = 0; i < afti_num_tile_columns_minus1; i++)	
afti_tile_column_width_minus1[i]	ue(v)
for(i = 0; i < afti_num_tile_rows_minus1; i++)	
afti_tile_row_height_minus1[i]	ue(v)
}	
afti_single_tile_per_tile_group_flag	u(1)
if(!afti_single_tile_per_tile_group_flag) {	
afti_num_tile_groups_in_atlas_frame_minus1	ue(v)
for(i = 0; i < afti_num_tile_groups_in_atlas_frame_minus1 + 1; i++) {	
if(i > 0)	
afti_top_left_tile_idx[i]	u(v)
afti_bottom_right_tile_idx_delta[i]	u(v)
}	
}	
afti_signalled_tile_group_ id_flag	u(1)
if(afti_signalled_tile_group_id_flag) {	
afti_signalled_tile_group_id_length_minus1	ue(v)
for(i = 0; i < afti_num_tile_groups_in_atlas_frame_minus1 + 1; i++)	
afti_tile_group_id[i]	u(v)
}	
}	
}	

7.3.6.5 Supplemental enhancement information RBSP syntax

sei_rbsp( ) {	**Descriptor**
do	
sei_message( )	
while(more_rbsp_data( ))	
rbsp_trailing_bits( )	
}	

7.3.6.6 Access unit delimiter RBSP syntax

access_unit_delimiter_rbsp( ) {	**Descriptor**
aframe_type	u(3)
rbsp_trailing_bits( )	
}	

7.3.6.7 End of sequence RBSP syntax

end_of_sequence_rbsp( ) {	**Descriptor**
}	

7.3.6.8 End of bitstream RBSP syntax

end_of_atlas_substream_rbsp( ) {	**Descriptor**
}	

7.3.6.9 Filler data RBSP syntax

filler_data_rbsp( ) {	**Descriptor**
while(next_bits(8) = = 0xFF)	
ff_byte /* equal to 0xFF */	f(8)
rbsp_trailing_bits( )	
}	

7.3.6.10 Atlas tile group layer RBSP syntax

atlas_tile_group_layer_rbsp( ) {	**Descriptor**
atlas_tile_group_header( )	
if(atgh_type != SKIP)	
atlas_tile_group_data_unit( )	
rbsp_trailing_bits( )	
}	

7.3.6.11 Atlas tile group header syntax

atlas_tile_group_header( ) {	**Descriptor**
atgh_atlas_frame_parameter_set_id	ue(v)
atgh_address	u(v)
atgh_type	ue(v)
atgh_atlas_frm_order_cnt_lsb	u(v)
if(asps_num_ref_atlas_frame_lists_in_asps > 0)	
atgh_ref_atlas_frame_list_sps_flag	u(1)
if(atgh_ref_atlas_frame_list_sps_flag == 0)	
ref_list_struct(asps_num_ref_atlas_frame_lists_in_asps)	
else if(asps_num_ref_atlas_frame_lists_in_asps > 1)	
atgh_ref_atlas_frame_list_idx	u(v)
for(j = 0; j < NumLtrAtlasFrmEntries; j++) {	
atgh_additional_afoc_lsb_present_flag[j]	u(1)
if(atgh_additional_afoc_lsb_present_flag[j])	

atgh_additional_afoc_lsb_val[j]	u(v)
}	
if(atgh_type ! = SKIP_TILE_GRP) {	
if(asps_normal_axis_limits_quantization_enabled_flag) {	
atgh_pos_min_z_quantizer	u(5)
if(asps_normal_axis_max_delta_value_enabled_flag)	
atgh_pos_delta_max_z_quantizer	u(5)
}	
if(asps_patch_size_quantizer_present_flag) {	
atgh_patch_size_x_info_quantizer	u(3)
atgh_patch_size_y_info_quantizer	u(3)
}	
if(afps_raw_3d_pos_bit_count_explicit_mode_flag)	
atgh_raw_3d_pos_axis_bit_count_minus1	u(v)
if(atgh_type = = P_TILE_GRP && num_ref_entries[RlsIdx] > 1) {	
atgh_num_ref_idx_active_override_flag	u(1)
if(atgh_num_ref_idx_active_override_flag)	
atgh_num_ref_idx_active_minus1	ue(v)
}	
}	
byte_alignment( )	
}	

7.3.6.12 Reference list structure syntax

ref_list_struct(rlsIdx) {	**Descriptor**
num_ref_entries[rlsIdx]	ue(v)
for(i = 0; i < num_ref_entries[rlsIdx]; i++) {	
if(asps_long_term_ref_atlas_frames_flag)	
st_ref_atlas_frame_flag[rlsIdx][i]	u(1)
if(st_ref_atlas_frame_flag[rlsIdx][i]) {	
abs_delta_afoc_st[rlsIdx][i]	ue(v)
if(abs_delta_afoc_st[rlsIdx][i] > 0)	
strpf_entry_sign_flag[rlsIdx][i]	u(1)
} else	
afoc_lsb_lt[rlsIdx][i]	u(v)
}	
}	

7.3.7 Atlas tile group data unit syntax

7.3.7.1 General atlas tile group data unit syntax

atlas_tile_group_data_unit( ) {	**Descriptor**
p = 0	
atgdu_patch_mode[p]	ue(v)
while(atgdu_patch_mode[p] !=I_END && atgdu_patch_mode[p] != P_END){	
patch_information_data(p, atgdu_patch_mode[p])	

p ++	
atgdu_patch_mode[p]	ue(v)
}	
AtgduTotalNumberOfPatches = p	
byte_alignment( )	
}	

7.3.7.2 Patch information data syntax

patch_information_data (patchIdx, patchMode) {	**Descriptor**
if(patchMode = = P_SKIP)	
skip_patch_data_unit(patchIdx)	
else if(patchMode = = P_MERGE)	
merge_patch_data_unit(patchIdx)	
else if(patchMode = = I_INTRA || patchMode = = P_INTRA)	
patch_data_unit(patchIdx)	
else if(patchMode = = P_INTER)	
inter_patch_data_unit(patchIdx)	
else if(patchMode = = I_RAW || patchMode = = P_RAW)	
raw_patch_data_unit(patchIdx)	
else if(patchMode = = I_EOM || patchMode = = P_EOM)	
eom_patch_data_unit(patchIdx)	
}	

7.3.7.3 Patch data unit syntax

patch_data_unit(patchIdx) {	**Descriptor**
pdu_2d_pos_x[patchIdx]	u(v)
pdu_2d_pos_y[patchIdx]	u(v)
pdu_2d_delta_size_x[patchIdx]	se(v)
pdu_2d_delta_size_y[patchIdx]	se(v)
pdu_3d_pos_x[patchIdx]	u(v)
pdu_3d_pos_y[patchIdx]	u(v)
pdu_3d_pos_min_z[patchIdx]	u(v)
if(asps_normal_axis_max_delta_value_enabled_flag)	
pdu_3d_pos_delta_max_z[patchIdx]	u(v)
pdu_projection_id[patchIdx]	u(v)
pdu_orientation_index[patchIdx]	u(v)
if(afps_lod_bit_count > 0)	
pdu_lod[patchIdx]	u(v)
if(asps_point_local_reconstruction_enabled_flag)	
point_local_reconstruction_data(patchIdx)	
}	

7.3.7.4 Skip patch data unit syntax

skip_patch_data_unit(frmIdx, patchIdx) {	**Descriptor**
}	

7.3.7.5 Merge patch data unit syntax

merge_patch_data_unit(frmIdx, patchIdx) {	**Descriptor**
if(atgh_num_ref_idx_active_minus1 > 0)	
mpdu_ref_index[patchIdx]	ue(v)
overridePlrFlag = 0	
mpdu_override_2d_params_flag[patchIdx]	u(1)
if(mpdu_override_2d_pos_flag[patchIdx]){	
mpdu_2d_pos_x[patchIdx]	se(v)
mpdu_2d_pos_y[patchIdx]	se(v)
mpdu_2d_delta_size_x[patchIdx]	se(v)
mpdu_2d_delta_size_y[patchIdx]	se(v)
if(asps_point_local_reconstruction_enabled_flag)	
overridePlrFlag = 1	
} else {	
mpdu_override_3d_params_flag[patchIdx]	u(1)
if(mpdu_override_3d_pos_flag[patchIdx]) {	
mpdu_3d_pos_x[patchIdx]	se(v)
mpdu_3d_pos_y[patchIdx]	se(v)
mpdu_3d_pos_min_z[patchIdx]	se(v)
if(asps_normal_axis_max_delta_value_enabled_flag)	
mpdu_3d_pos_delta_max_z[patchIdx]	se(v)
if(asps_point_local_reconstruction_enabled_flag) {	
mpdu_override_plr_flag[patchIdx]	u(1)
overridePlrFlag = mpdu_override_plr_flag[patchIdx]	
}	
}	
}	
if (overridePlrFlag && asps_point_local_reconstruction_ enabled_flag)	
point_local_reconstruction_data(patchIdx)	
}	

7.3.7.6 Inter patch data unit syntax

inter_patch_data_unit(patchIdx) {	**Descriptor**
if(atgh_num_ref_idx_active_minus1 > 0)	
ipdu_ref_index[patchIdx]	ue(v)
ipdu_patch_index[patchIdx]	se(v)
ipdu_2d_pos_x[patchIdx]	se(v)
ipdu_2d_pos_y[patchIdx]	se(v)
ipdu_2d_delta_size_x[patchIdx]	se(v)
ipdu_2d_delta_size_y[patchIdx]	se(v)
ipdu_3d_pos_x[patchIdx]	se(v)
ipdu_3d_pos_y[patchIdx]	se(v)
ipdu_3d_pos_min_z[patchIdx]	se(v)
if(asps_normal_axis_max_delta_value_enabled_flag)	
ipdu_3d_pos_delta_max_z[patchIdx]	se(v)
if(asps_point_local_reconstruction_enabled_flag)	
point_local_reconstruction_data(patchIdx)	
}	

7.3.7.7 Raw patch data unit syntax

raw_patch_data_unit(patchIdx) {	**Descriptor**
if(RawSeparateVideoPresentFlag)	
rpdu_patch_in_raw_video_flag[patchIdx]	u(1)
rpdu_2d_pos_x[patchIdx]	u(v)
rpdu_2d_pos_y[patchIdx]	u(v)
rpdu_2d_delta_size_x[patchIdx]	se(v)
rpdu_2d_delta_size_y[patchIdx]	se(v)
rpdu_3d_pos_x[patchIdx]	u(v)
rpdu_3d_pos_y[patchIdx]	u(v)
rpdu_3d_pos_z[patchIdx]	u(v)
rpdu_points[patchIdx]	ue(v)
}	

7.3.7.8 EOM patch data unit syntax

eom_patch_data_unit(patchIdx) {	**Descriptor**
epdu_2d_pos_x[patchIdx]	u(v)
epdu_2d_pos_y[patchIdx]	u(v)
epdu_2d_delta_size_x[patchIdx]	se(v)
epdu_2d_delta_size_y[patchIdx]	se(v)
epdu_patch_count_minus1[patchIdx]	u(v)
for(i = 0; i < epdu_patch_count_minus1[patchIdx] + 1; i++)	
epdu_points[patchIdx][i]	u(v)
}	

7.3.7.9 Point local reconstruction data syntax

point_local_reconstruction_data(patchIdx) {	**Descriptor**
for(i = 0; i < asps_map_count_minus1 + 1; i++) {	
if(plri_point_local_reconstruction_map_enabled_flag[i]) {	
if(BlockCount > plri_block_threshold_per_patch_minus1[i] + 1)	
plrd_level[i][patchIdx]	u(1)
else	
plrd_level[i][patchIdx] = 1	
if(plrd_level[i][patchIdx] == 0) {	
for(j = 0; j < BlockCount; j++) {	
plrd_present_block_flag[i][patchIdx][j]	u(1)
if(plrd_present_block_flag[i][patchIdx][j]) {	
plrd_block_mode_minus1[i][patchIdx][j]	u(v)
}	
}	
} else {	
plrd_present_flag[i][patchIdx]	u(1)
if(plrd_present_flag[i][patchIdx])	
plrd_mode_minus1[i][patchIdx]	u(v)
}	
}	
}	

}	

7.3.8 Supplemental enhancement information message syntax

sei_message( ) {	**Descriptor**
payloadType = 0	
do {	
sm_payload_type_byte	u(8)
payloadType += sm_payload_type_byte	
} while(sm_payload_type_byte = = 0xFF)	
payloadSize = 0	
do{	
sm_payload_size_byte	u(8)
payloadSize += sm_payload_size_byte	
} while(sm_payload_size_byte = = 0xFF)	
sei_payload(payloadType, payloadSize)	
}	

7.4 Semantics

7.4.1 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.

7.4.2 V-PCC unit semantics

7.4.2.1 General V-PCC unit semantics

trailing_zero_8bits is a byte equal to 0x00.

7.4.2.2 V-PCC unit header semantics

Table 7-1 V-PCC Unit Types

vuh_unit_type	**Identifier**	**V-PCC Unit Type**	**Description**
0	VPCC_VPS	V-PCC parameter set	V-PCC level parameters
1	VPCC_AD	Atlas data	Atlas information
2	VPCC_OVD	Occupancy Video Data	Occupancy information
3	VPCC_GVD	Geometry Video Data	Geometry information
4	VPCC_AVD	Attribute Video Data	Attribute information
5…31	VPCC_RSVD	Reserved	-

vuh_unit_type indicates the V-PCC unit type as specified in Table 7-1.

vuh_vpcc_parameter_set_id specifies the value of vps_vpcc_parameter_set_id for the active V-PCC VPS. The value of vuh_vpcc_parameter_set_id shall be in the range of 0 to 15, inclusive.

vuh_atlas_id specifies the index of the atlas that corresponds to the current V-PCC unit. The value of vuh_atlas_id shall be in the range of 0 to 63, inclusive.

vuh_attribute_index indicates the index of the attribute data carried in the Attribute Video Data unit. The value of vuh_attribute_index shall be in the range of 0 to (ai_attribute_count[vuh_atlas_id] – 1), inclusive.

vuh_attribute_dimension_index indicates the index of the attribute dimension group carried in the Attribute Video Data unit. The value of vuh_attribute_dimension_index shall be in the range of 0 to ai_attribute_dimension_partitions_minus1[vuh_atlas_id][vuh_attribute_index], inclusive.

vuh_map_index when present, indicates the map index of the current geometry or attribute stream. When not present, the map index of the current geometry or attribute stream is derived based on the type of the stream and the operations described in clauses 8.2 and 8.3 for geometry and attribute video streams respectively. The value of vuh_map_index, when present, shall be in the range of 0 to vps_map_count_minus1[vuh_atlas_id], inclusive.

vuh_raw_video_flag equal to 1 indicates that the associated geometry or attribute video data unit is a RAW coded points video only. vuh_raw_video_flag equal to 0 indicates that the associated geometry or attribute video data unit may contain RAW coded points. When vuh_raw_video_flag is not present, its value shall be inferred to be equal to 0.

vuh_reserved_zero_12bits, when present, shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for vuh_reserved_zero_12bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of vuh_reserved_zero_12bits.

vuh_reserved_zero_17bits, when present, shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for vuh_reserved_zero_17bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of vuh_reserved_zero_17bits.

vuh_reserved_zero_27bits, when present, shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for vuh_reserved_zero_27bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of vuh_reserved_zero_27bits.

7.4.2.3 V-PCC unit payload semantics

None.

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

A vpcc unit type with a value of vuh_unit_type equal to VPCC_VPS is expected to be the first VPCC unit type in a VPCC bitstream. All other VPCC unit types shall follow this unit type without any additional restrictions in their coding order.

7.4.3 Byte alignment semantics

alignment_bit_equal_to_one shall be equal to 1.

alignment_bit_equal_to_zero shall be equal to 0.

7.4.4 V-PCC parameter set semantics

7.4.4.1 General V-PCC parameter set semantics

vps_vpcc_parameter_set_id provides an identifier for the V-PCC VPS for reference by other syntax elements. The value of vps_vpcc_parameter_set_id shall be in the range of 0 to 15, inclusive.

vps_atlas_count_minus1 plus 1 indicates the total number of supported atlases in the current bitstream. The value of vps_atlas_count_minus1 shall be in the range of 0 to 63, inclusive.

vps_frame_width[j] indicates the V-PCC frame width in terms of integer luma samples for the atlas with index j. This frame width is the nominal width that is associated with all V-PCC components for the atlas with index j.

vps_frame_height[j] indicates the V-PCC frame height in terms of integer luma samples for the atlas with index j. This frame height is the nominal height that is associated with all V-PCC components for the atlas with index j.

vps_map_count_minus1[j] plus 1 indicates the number of maps used for encoding the geometry and attribute data for the atlas with index j. vps_map_count_minus1[j] shall be in the range of 0 to 15, inclusive.

vps_multiple_map_streams_present_flag[j] equal to 0 indicates that all geometry or attribute maps for the atlas with index j are placed in a single geometry or attribute video stream, respectively. vps_multiple_map_streams_present_flag[j] equal to 1 indicates that all geometry or attribute maps for the atlas with index j are placed in separate video streams. When vps_multiple_map_streams_present_flag[j] is not present, its value shall be inferred to be equal to 0.

vps_map_absolute_coding_enabled_flag[j][i] equal to 1 indicates that the geometry map with index i for the atlas with index j is coded without any form of map prediction. vps_map_absolute_coding_enabled_flag[j][i]equal to 0 indicates that the geometry map with index i for the atlas with index j is first predicted from another, earlier coded map, prior to coding. If vps_map_absolute_coding_enabled_flag[j][i] is not present, its value shall be inferred to be equal to 1.

vps_map_predictor_index_diff[j][i] is used to compute the predictor of the geometry map with index i for the atlas with index j when vps_map_absolute_coding_enabled_flag[j][i] is equal to 0. More specifically, the map predictor index for map i shall be computed as:

$$\text{VPCCMapPredictorIndex}(\ i\ ) = (i - 1) - \text{vps\_map\_predictor\_index\_diff}[\ j\ ][\ i\ ] \quad (7\text{-}1)$$

The value of vps_map_predictor_index_diff[j][i] shall be in the range from 0 to i - 1, inclusive. When vps_map_predictor_index_diff[j][i] is not present, its value shall be inferred to be equal to 0.

vps_raw_patch_enabled_flag[j] equal to 1 indicates that patches with RAW coded points for the atlas with index j may be present in the bitstream. vps_raw_patch_enabled_flag[j] equal to 0 indicates that patches with RAW coded points for the atlas with index j shall not be present in the bitstream.

vps_raw_separate_video_present_flag[j] equal to 1 indicates that RAW coded geometry and attribute information for the atlas with index j may be stored in a separate video stream. vps_raw_separate_video_present_flag[j] equal to 0 indicates that RAW coded geometry and attribute information for the atlas with index j shall not be stored in a separate video stream. When vps_raw_separate_video_present_flag[j] is not present, it is inferred to be equal to 0.

vps_extension_present_flag equal to 1 specifies that the syntax element vps_extension_length is present in vpcc_parameter_set syntax structure. vps_extension_present_flag equal to 0 specifies that syntax element vps_extension_length is not present. vps_extension_present_flag shall be equal to 0 in bitstreams conforming to this version of this Specification.

vps_extension_length specifies the length of the vpcc extension data in bytes, not including the bits used for signalling vps_extension_length itself. When not present, the value of vps_extension_length is inferred to be equal to 0.

vps_extension_data_byte may have any value.

7.4.4.2 Profile, tier, and level semantics

ptl_tier_flag specifies the tier context for the interpretation of ptl_level_idc as specified in Annex A.

ptl_profile_codec_group_idc indicates the codec group profile component to which the CPCS conforms as specified in Annex A. Bitstreams shall not contain values of ptl_profile_codec_group_idc other than those specified in Annex A. Other values of ptl_profile_codec_group_idc are reserved for future use by ISO/IEC.

ptl_profile_pcc_toolset_idc indicates the toolset combination profile component to which the CPCS conforms as specified in Annex A. Bitstreams shall not contain values of ptl_profile_pcc_toolset_idc other than those specified in Annex A. Other values of ptl_profile_pcc_toolset_idc are reserved for future use by ISO/IEC.

ptl_profile_reconstruction_idc indicates the reconstruction profile component to which the CPCS is recommended to conform to as specified in Annex A. This indication is only informative and decoders may select to use a different reconstruction profile than the one indicated in the bitstream. Bitstreams shall not contain values of ptl_profile_reconstruction_idc other than those specified in Annex A. Other values of ptl_profile_reconstruction_idc are reserved for future use by ISO/IEC.

ptl_reserved_zero_32bits, when present, shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for ptl_reserved_zero_32bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ptl_reserved_zero_32bits.

ptl_level_idc indicates a level to which the CPCS conforms as specified in Annex A Bitstreams shall not contain values of ptl_level_idc other than those specified in Annex A. Other values of ptl_level_idc are reserved for future use by ISO/IEC.

7.4.4.3 Occupancy information semantics

oi_occupancy_codec_id[j] indicates the identifier of the codec used to compress the occupancy map information for the atlas with index j. occupancy_codec_id shall be in the range of 0 to 255, inclusive. This codec may be identified through a component codec mapping SEI message or through means outside this Specification.

oi_lossy_occupancy_map_compression_threshold[j] indicates the threshold to be used to derive the binary occupancy map from the decoded occupancy map video for the atlas with index j. oi_lossy_occupancy_map_compression_threshold[j] shall be in the range of 0 to 255, inclusive.

oi_occupancy_nominal_2d_bitdepth_minus1[j] plus 1 indicates the nominal 2D bit depth to which the occupancy video for the atlas with index j shall be converted to. oi_occupancy_nominal_2d_bitdepth_minus1[j] shall be in the range of 0 to 31, inclusive.

oi_occupancy_MSB_align_flag[j] indicates how the decoded occupancy video samples associated with an atlas with index j are converted to samples at the nominal occupancy bitdepth, when the nominal occupancy bitdepth is less than the bitdepth of the decoded occupancy video.

Let bitDiff be the difference between the bitdepth of the decoded occupancy video and the nominal occupancy bitdepth. Let S_d and S_n be the decoded occupancy sample value and occupancy sample value at the nominal occupancy bitdepth, respectively. The following applies:

– If oi_occupancy_MSB_align_flag[j] is equal to 1,

S_n = (S_d >> bitDiff)

– Otherwise (oi_occupancy_MSB_align_flag[j] is equal to 0),

S_n = Min(S_d, (1 << (oi_occupancy_nominal_2d_bitdepth_minus1[j] + 1)) − 1).

7.4.4.4 Geometry information semantics

gi_geometry_codec_id[j] indicates the identifier of the codec used to compress the geometry video data for the atlas with index j. geometry_codec_id shall be in the range of 0 to 255, inclusive. This codec may be identified through a component codec mapping SEI message or through means outside this Specification.

gi_geometry_nominal_2d_bitdepth_minus1[j] plus 1 indicates the nominal 2D bit depth to which all geometry videos for the atlas with index j shall be converted to. gi_geometry_nominal_2d_bitdepth_minus1[j] shall be in the range of 0 to 31, inclusive.

gi_geometry_MSB_align_flag[j] indicates how the decoded geometry video samples associated with an atlas with index j are converted to samples at the nominal geometry bitdepth, when the nominal geometry bitdepth is less than the bitdepth of the decoded geometry video.

Let bitDiff be the difference between the bitdepth of the decoded geometry video and the nominal geometry bitdepth. Let S_d and S_n be the decoded geometry sample value and geometry sample value at the nominal geometry bitdepth, respectively. The following applies:

– If gi_geometry_MSB_align_flag[j] is equal to 1,

S_n = (S_d >> bitDiff)

– Otherwise (gi_geometry_MSB_align_flag[j] is equal to 0),

S_n = Min(S_d, (1 << (gi_geometry_nominal_2d_bitdepth_minus1[j] + 1)) − 1).

gi_geometry_3d_coordinates_bitdepth_minus1[j] plus 1 indicates the bit depth of the geometry coordinates of the reconstructed point cloud for the atlas with index j. gi_geometry_3d_coordinates_bitdepth_minus1[j] shall be in the range of 0 to 31, inclusive.

gi_raw_geometry_codec_id[j], when present, indicates the identifier of the codec used to compress the geometry video data for RAW coded points, when RAW coded points are encoded in a separate video stream for the atlas with index j. gi_raw_geometry_codec_id[j] shall be in the range of 0 to 255, inclusive. This codec may be identified through a component codec mapping SEI message or through means outside this Specification. When not present the value of gi_raw_geometry_codec_id[j] shall be set equal to gi_geometry_codec_id[j].

7.4.4.5 Attribute information semantics

ai_attribute_count[j] indicates the number of attributes associated with the point cloud for the atlas with index j. vpcc_attribute_count shall be in the range of 0 to 127, inclusive.

ai_attribute_type_id[j][i] indicates the attribute type of the Attribute Video Data unit with index i for the atlas with index j. Table 7-2 describes the list of supported attributes and their relationship with ai_attribute_type_id.

Table 7-2 V-PCC attribute types

ai_attribute_type_id[j][i]	**Identifier**	**Attribute type**
0	ATTR_TEXTURE	Texture
1	ATTR_MATERIAL_ID	Material ID
2	ATTR_TRANSPARENCY	Transparency
3	ATTR_REFLECTANCE	Reflectance
4	ATTR_NORMAL	Normals
5…14	ATTR_RESERVED	Reserved
15	ATTR_UNSPECIFIED	Unspecified

ATTR_TEXTURE indicates an attribute that contains texture information of a point cloud. For example, this may indicate an attribute that contains RGB (Red, Green, Blue) colour information.

ATTR_MATERIAL_ID indicates an attribute that contains supplemental information that indicates the material type of a point in a point cloud. For example, the material type could be used as a indicator for identifying an object or the characteristic of a point within a point cloud. The interpretation of the values of such attribute frame type is outside the scope of this Specification.

ATTR_TRANSPARENCY indicates an attribute that contains transparency information that is associated with each point in a point cloud.

ATTR_REFLECTANCE indicates an attribute that contains reflectance information that is associated with each point in a point cloud.

ATTR_NORMAL indicates an attribute that contains a unit vector information associated with each point in a point cloud. The unit vector specifies the perpendicular direction to a surface at a point (i.e. direction a point is facing). An attribute frame with this attribute type shall have ai_attribute_dimension_minus1 equal to 2. Each channel of an attribute frame with this attribute type shall contain one component of the unit vector (x, y, z), where the first component contains the x coordinate, the second component contains the y coordinate, and the third component contains the z coordinate.

ATTR_UNSPECIFIED indicates an attribute that contains values that have no specified meaning in this Recommendation | International Standard and will not have a specified meaning in the future as an integral part of this Recommendation | International Standard.

ai_attribute_codec_id[j][i] indicates the identifier of the codec used to compress the attribute video data with index i for the atlas with index j. ai_attribute_codec_id[j][i] shall be in the range of 0 to 255, inclusive. This codec may be identified through a component codec mapping SEI message or through means outside this Specification.

ai_raw_attribute_codec_id[j][i], when present, indicates the identifier of the codec used to compress the attribute video data for RAW coded points of attribute i, when RAW coded points are encoded in a separate video stream for the atlas with index j. ai_raw_attribute_codec_id[j][i] shall be in the range of 0 to 255, inclusive. This codec may be identified through a component codec mapping SEI message or through means outside this Specification. When not present the value of ai_raw_attribute_codec_id[j][i] shall be set equal to ai_attribute_codec_id[j][i].

ai_attribute_map_absolute_coding_enabled_flag[k][j][i] equal to 1 indicates that the attribute map with index i, for the attribute with index j, that corresponds to the atlas with index k, is coded without any form of map prediction. ai_attribute_map_absolute_coding_enabled_flag[k][j][i] equal to 0 indicates that the attribute map with index i, for attribute with index j, that corresponds to the atlas with index k is first predicted from another, earlier coded attribute map of the same attribute and atlas index, prior to coding. If ai_attribute_map_absolute_coding_enabled_flag[k][j][i] is not present, its value shall be inferred to be equal to 1.

ai_attribute_dimension_minus1[j][i] plus 1 indicates the total number of dimensions (i.e., number of channels) of the attribute with index i for the atlas with index j. ai_attribute_dimension_minus1[j][i] shall be in the range of 0 to 63, inclusive.

ai_attribute_dimension_partitions_minus1[j][i] plus 1 indicates the number of partition groups in which the attribute channels for attribute with index i for the atlas with index j should be grouped in. ai_attribute_dimension_partitions_minus1[j][i] shall be in the range of 0 to 63, inclusive.

ai_attribute_partition_channels_minus1[k][i][j] plus 1 indicates the number of channels assigned to the dimension partition group with index j of attribute with index i for atlas with index k. ai_attribute_partition_channels_minus1[k][i][j] shall be in the range of 0 to ai_attribute_dimension_minus1[k][i], inclusive, for all dimension partition groups.

ai_attribute_nominal_2d_bitdepth_minus1[j][i] plus 1 indicates the nominal 2D bit depth to which all the attribute videos with attribute index i for atlas with index j shall be converted to. ai_attribute_nominal_2d_bitdepth_minus1[j][i] shall be in the range of 0 to 31, inclusive.

ai_attribute_MSB_align_flag[j] indicates how the decoded attribute video samples associated with an atlas with index j are converted to samples at nominal attribute bitdepth, when the nominal attribute bitdepth is less than the bitdepth of the decoded attribute video.

Let bitDiff be the difference between the bitdepth of the decoded attribute video and the nominal attribute bitdepth. Let S_d and S_n be the decoded attribute sample value and attribute sample value at nominal bitdepth, respectively. The following applies:

– If ai_attribute_MSB_align_flag[j] is equal to 1,

$$S_n = (S_d >> \text{bitDiff})$$

– Otherwise (ai_attribute_MSB_align_flag[j] is equal to 0),

$$S_n = \text{Min}(S_d, (1 << (\text{oi\_occupancy\_nominal\_2d\_bitdepth\_minus1}[j] + 1)) - 1).$$

7.4.5 NAL unit semantics

7.4.5.1 General NAL unit semantics

NumBytesInNalUnit specifies the size of the NAL unit in bytes. This value is required for decoding of the NAL unit. Some form of demarcation of NAL unit boundaries is necessary to enable inference of NumBytesInNalUnit. One such demarcation method is specified in Annex C for the sample stream format. Other methods of demarcation may be specified outside of this Specification.

NOTE 1 – The atlas coding layer (ACL) is specified to efficiently represent the content of the patch data. The NAL is specified to format that data and provide header information in a manner appropriate for conveyance on a variety of communication channels or storage media. All data are contained in NAL units, each of which contains an integer number of bytes. A NAL unit specifies a generic format for use in both packet-oriented and bitstream systems. The format of NAL units for both packet-oriented transport and sample streams is identical except that in the sample stream format specified in Annex C each NAL unit can be preceded by an additional element that specifies the size of the NAL unit.

rbsp_byte[i] is the i-th byte of an RBSP. An RBSP is specified as an ordered sequence of bytes as follows:

The RBSP contains a string of data bits (SODB) as follows:

– If the SODB is empty (i.e., zero bits in length), the RBSP is also empty.
– Otherwise, the RBSP contains the SODB as follows:
  1) The first byte of the RBSP contains the first (most significant, left-most) eight bits of the SODB; the next byte of the RBSP contains the next eight bits of the SODB, etc., until fewer than eight bits of the SODB remain.
  2) The rbsp_trailing_bits( ) syntax structure is present after the SODB as follows:
     i) The first (most significant, left-most) bits of the final RBSP byte contain the remaining bits of the SODB (if any).
     ii) The next bit consists of a single bit equal to 1 (i.e., rbsp_stop_one_bit).
     iii) When the rbsp_stop_one_bit is not the last bit of a byte-aligned byte, one or more bits equal to 0 (i.e. instances of rbsp_alignment_zero_bit) are present to result in byte alignment.
  3) One or more cabac_zero_word 16-bit syntax elements equal to 0x0000 may be present in some RBSPs after the rbsp_trailing_bits( ) at the end of the RBSP.

Syntax structures having these RBSP properties are denoted in the syntax tables using an "_rbsp" suffix. These structures are carried within NAL units as the content of the rbsp_byte[i] data bytes. The association of the RBSP syntax structures to the NAL units is as specified in Table 7-3.

NOTE 2 – When the boundaries of the RBSP are known, the decoder can extract the SODB from the RBSP by concatenating the bits of the bytes of the RBSP and discarding the rbsp_stop_one_bit, which is the last (least significant, right-most) bit equal to 1, and discarding any following (less significant, farther to the right) bits that follow it, which are equal to 0. The data necessary for the decoding process is contained in the SODB part of the RBSP.

7.4.5.2 NAL unit header semantics

nal_forbidden_zero_bit shall be equal to 0.

nal_unit_type specifies the type of the RBSP data structure contained in the NAL unit as specified in Table 7-1.

NAL units that have nal_unit_type in the range of NAL_UNSPEC_48..NAL_UNSPEC_63, inclusive, for which semantics are not specified, shall not affect the decoding process specified in this Specification.

NOTE 1 – NAL unit types in the range of NAL_UNSPEC_48..NAL_UNSPEC_63 may be used as determined by the application. No decoding process for these values of nal_unit_type is specified in this Specification. Since different applications might use these NAL unit types for different purposes, particular care must be exercised in the design of encoders that generate NAL units with these nal_unit_type values, and in the design of decoders that interpret the content of NAL units with these nal_unit_type values. This Specification does not define any management for these values. These nal_unit_type values might only be suitable for use in contexts in which "collisions" of usage (i.e., different definitions of the meaning of the NAL unit content for the same nal_unit_type value) are unimportant, or not possible, or are managed – e.g., defined or managed in the controlling application or transport specification, or by controlling the environment in which bitstreams are distributed.

For purposes other than determining the amount of data in the decoding units of the bitstream (as specified in Annex D), decoders shall ignore (i.e. remove from the bitstream and discard) the contents of all NAL units that use reserved values of nal_unit_type.

NOTE 2 – This requirement allows future definition of compatible extensions to this Specification.

Table 7-3 – NAL unit type codes and NAL unit type classes

nal_unit_type	Name of nal_unit_type	Content of NAL unit and RBSP syntax structure	NAL unit type class
0	NAL_TRAIL	Coded tile group of a non-TSA, non STSA trailing atlas frame atlas_tile_group_layer_rbsp( )	ACL
1	NAL_TSA	Coded tile group of a TSA atlas frame atlas_tile_group_layer_rbsp( )	ACL
2	NAL_STSA	Coded tile group of a STSA atlas frame atlas_tile_group_layer_rbsp( )	ACL
3	NAL_RADL	Coded tile group of a RADL atlas frame atlas_tile_group_layer_rbsp( )	ACL
4	NAL_RASL	Coded tile group of a RASL atlas frame atlas_tile_group_layer_rbsp( )	ACL
5	NAL_SKIP	Coded tile group of a skipped atlas frame atlas_tile_group_layer_rbsp( )	ACL
6..9	NAL_RSV_ACL_6.. NAL_RSV_ACL_9	Reserved non-IRAP ACL NAL unit types	ACL
10 11 12	NAL_BLA_W_LP NAL_BLA_W_RADL NAL_BLA_N_LP	Coded tile group of a BLA atlas frame atlas_tile_group_layer_rbsp( )	ACL
13 14 15	NAL_GBLA_W_LP NAL_GBLA_W_RADL NAL_GBLA_N_LP	Coded tile group of a GBLA atlas frame atlas_tile_group_layer_rbsp( )	ACL
16 17	NAL_IDR_W_RADL NAL_IDR_N_LP	Coded tile group of an IDR atlas frame atlas_tile_group_layer_rbsp( )	ACL
18 19	NAL_GIDR_W_RADL NAL_GIDR_N_LP	Coded tile group of a GIDR atlas frame atlas_tile_group_layer_rbsp( )	ACL
20	NAL_CRA	Coded tile group of a CRA atlas frame atlas_tile_group_layer_rbsp( )	ACL
21	NAL_GCRA	Coded tile group of a GCRA atlas frame atlas_tile_group_layer_rbsp( )	ACL
22 23	NAL_IRAP_ACL_22 NAL_IRAP_ACL_23	Reserved IRAP ACL NAL unit types	ACL
24..31	NAL_RSV_ACL_24.. NAL_RSV_ACL_31	Reserved non-IRAP ACL NAL unit types	ACL
32	NAL_ASPS	Atlas sequence parameter set atlas_sequence_parameter_set_rbsp( )	non-ACL
33	NAL_AFPS	Atlas frame parameter set atlas_frame_parameter_set_rbsp( )	non-ACL
34	NAL_AUD	Access unit delimiter access_unit_delimiter_rbsp( )	non-ACL
35	NAL_VPCC_AUD	V-PCC access unit delimiter access_unit_delimiter_rbsp( )	non-ACL
36	NAL_EOS	End of sequence end_of_seq_rbsp( )	non-ACL

37	NAL_EOB	End of bitstream end_of_atlas_substream_rbsp( )	non-ACL
38	NAL_FD	Filler filler_data_rbsp( )	non-ACL
39 40	NAL_PREFIX_SEI NAL_SUFFIX_SEI	Supplemental enhancement information sei_rbsp( )	non-ACL
41..47	NAL_RSV_NACL_41.. NAL_RSV_NACL_47	Reserved non-ACL NAL unit types	non-ACL
48..63	NAL_UNSPEC_48.. NAL_UNSPEC_63	Unspecified non-ACL NAL unit types	non-ACL

NOTE 3 – A clean random access (CRA) and a global clean random access atlas frame may have associated random access skipped leading (RASL) or random access decodable leading (RADL) atlas frames present in the bitstream.

NOTE 4 – A broken link access (BLA) or a global broken link access (GBLA) atlas frame having nal_unit_type equal to NAL_BLA_W_LP and NAL_GBLA_W_LP respectively, may have associated RASL or RADL atlas frames present in the bitstream. A BLA or a GBLA atlas frame having nal_unit_type equal to NAL_BLA_W_RADL and NAL_GBLA_W_RADL respectively, does not have associated RASL atlas frames present in the bitstream, but may have associated RADL atlas frames in the bitstream. A BLA or a GBLA atlas frame having nal_unit_type equal to NAL_BLA_N_LP and NAL_GBLA_N_LP respectively, does not have associated leading atlas frames present in the bitstream.

NOTE 5 – An instantaneous decoding refresh (IDR) or a global instantaneous decoding refresh (GIDR) atlas frame having nal_unit_type equal to NAL_IDR_N_LP and NAL_GIDR_N_LP respectively, does not have associated leading atlas frames present in the bitstream. An IDR or a GIDR atlas frame having nal_unit_type equal to NAL_IDR_W_RADL and NAL_GIDR_W_RADL respectively, does not have associated RASL atlas frames present in the bitstream, but may have associated RADL atlas frames in the bitstream.

NOTE 6 – A sub-layer non-reference (SLNR) atlas frame is not included in any of RefPicSetStCurrBefore, RefPicSetStCurrAfter and RefPicSetLtCurr of any atlas frame with the same value of TemporalId, and may be discarded without affecting the decodability of other atlas frames with the same value of TemporalId.

All coded tile group NAL units of an access unit shall have the same value of nal_unit_type. An atlas frame or an access unit is also referred to as having a nal_unit_type equal to the nal_unit_type of the coded tile group NAL units of the atlas frame or the access unit.

If an atlas frame has nal_unit_type equal to NAL_TRAIL, NAL_TSA, NAL_STSA, NAL_RADL, and NAL_RASL, the atlas frame is an SLNR atlas frame. Otherwise, the atlas frame is a sub-layer reference atlas frame.

Each atlas frame, other than the first atlas frame in the bitstream in decoding order, is considered to be associated with the previous intra random access point (IRAP) atlas frame in decoding order.

When an atlas frame is a leading atlas frame, it shall be a RADL or RASL atlas frame.

When an atlas frame is a trailing atlas frame, it shall not be a RADL or RASL atlas frame.

When an atlas frame is a leading atlas frame, it shall precede, in decoding order, all trailing atlas frames that are associated with the same IRAP atlas frame.

No RASL atlas frames shall be present in the bitstream that are associated with a BLA atlas frame having nal_unit_type equal to NAL_BLA_W_RADL or NAL_BLA_N_LP.

No RASL atlas frames shall be present in the bitstream that are associated with a GBLA atlas frame having nal_unit_type equal to NAL_GBLA_W_RADL or NAL_GBLA_N_LP.

No RASL atlas frames shall be present in the bitstream that are associated with an IDR or a GIDR atlas frame.

No RADL atlas frames shall be present in the bitstream that are associated with a BLA atlas frame having nal_unit_type equal to NAL_BLA_N_LP, a GBLA atlas frame having nal_unit_type equal to NAL_GBLA_N_LP, an IDR atlas frame having nal_unit_type equal to NAL_IDR_N_LP, or a GIDR atlas frame having nal_unit_type equal to NAL_GIDR_N_LP.

NOTE 7 – It is possible to perform random access at the position of an IRAP access unit by discarding all access units before the IRAP access unit (and to correctly decode the IRAP atlas frame and all the subsequent non-RASL atlas frames in decoding order), provided each parameter set is available (either in the bitstream or by external means not specified in this Specification) when it needs to be activated.

Any RASL atlas frame associated with a CRA, GCRA, BLA, or GBLA atlas frame shall precede any RADL atlas frame associated with the CRA, GCRA, BLA, or GBLA atlas frame in output order.

Any RASL atlas frame associated with a CRA or GCRA atlas frame shall follow, in output order, any IRAP atlas frame that precedes the CRA or GCRA atlas frame in decoding order.

A nal_unit_type equal to NAL_VPCC_AUD that is associated with a particular atlas frame j means that all output V-PCC component frames that have the same output time order as the atlas frame j shall have the same decoding order as that of the atlas frame.

NOTE 8 – An application may specify that all frames in a V-PCC sequence are delimited using a nal_unit_type equal to NAL_VPCC_AUD, in which case all V-PCC sub-bitstreams including sub-bitstreams that correspond to different maps will be aligned in both decoding order and output time.

nal_layer_id specifies the identifier of the layer to which an ACL NAL unit belongs or the identifier of a layer to which a non-ACL NAL unit applies. The value of nal_layer_id shall be in the range of 0 to 62, inclusive. The value of 63 may be specified in the future by ISO/IEC. For purposes other than determining the amount of data in the decoding units of the bitstream, decoders shall ignore all data that follow the value 63 for nal_layer_id in a NAL unit, and decoders conforming to a profile specified in Annex A shall ignore (i.e., remove from the bitstream and discard) all NAL units with values of nal_layer_id not equal to 0.

NOTE 9 – The value of 63 for nal_layer_id may be used to indicate an extended layer identifier in a future extension of this Specification.

The value of nal_layer_id shall be the same for all ACL NAL units of a coded atlas frame. The value of nal_layer_id of a coded atlas frame is the value of the nal_layer_id of the ACL NAL units of the coded atlas frame.

When nal_unit_type is equal to NAL_EOB, the value of nal_layer_id shall be equal to 0.

nal_temporal_id_plus1 minus 1 specifies a temporal identifier for the NAL unit. The value of nal_temporal_id_plus1 shall not be equal to 0.

The variable TemporalId is specified as follows:

$$\text{TemporalId} = \text{nal\_temporal\_id\_plus1} - 1 \qquad (7\text{-}2)$$

When nal_unit_type is in the range of NAL_BLA_W_LP to NAL_RSV_IRAP_ACL23, inclusive, i.e., the coded tile group belongs to an IRAP atlas frame, TemporalId shall be equal to 0.

When nal_unit_type is equal to NAL_TSA, TemporalId shall not be equal to 0.

When nal_layer_id is equal to 0 and nal_unit_type is equal to NAL_STSA, TemporalId shall not be equal to 0.

The value of TemporalId shall be the same for all ACL NAL units of an access unit. The value of TemporalId of a coded atlas frame or an access unit is the value of the TemporalId of the ACL NAL units of the coded atlas frame or the access unit. The value of TemporalId of a sub-layer representation is the greatest value of TemporalId of all ACL NAL units in the sub-layer representation.

The value of TemporalId for non-ACL NAL units is constrained as follows:

– If nal_unit_type is equal to NAL_ASPS, TemporalId shall be equal to 0 and the TemporalId of the access unit containing the NAL unit shall be equal to 0.

– Otherwise if nal_unit_type is equal to NAL_EOS or NAL_EOB, TemporalId shall be equal to 0.

– Otherwise, if nal_unit_type is equal to NAL_AUD, NAL_VPCC_AUD, or NAL_FD, TemporalId shall be equal to the TemporalId of the access unit containing the NAL unit.

– Otherwise, TemporalId shall be greater than or equal to the TemporalId of the access unit containing the NAL unit.

NOTE 10 – When the NAL unit is a non-ACL NAL unit, the value of TemporalId is equal to the minimum value of the TemporalId values of all access units to which the non-ACL NAL unit applies. When nal_unit_type is equal to NAL_AFPS, TemporalId may be greater than or equal to the TemporalId of the containing access unit, as all atlas frame parameter sets (AFPSs) may be included in the beginning of a bitstream, wherein the first coded atlas frame has TemporalId equal to 0. When nal_unit_type is equal to NAL_PREFIX_SEI or NAL_SUFFIX_SEI, TemporalId may be greater than or equal to the TemporalId of the containing access unit, as an SEI NAL unit may contain information, e.g., in a buffering period SEI message or an atlas frame timing SEI message, that applies to a bitstream subset that includes access units for which the TemporalId values are greater than the TemporalId of the access unit containing the SEI NAL unit.

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

7.4.5.3.1 General

This clause specifies constraints on the order of NAL units in the bitstream.

Any order of NAL units in the bitstream obeying these constraints is referred to in the text as the decoding order of NAL units. Within a NAL unit, the syntax in clauses 7.3, specifies the decoding order of syntax elements. Decoders shall be capable of receiving NAL units and their syntax elements in decoding order.

7.4.5.3.2 Order of ASPS and AFPS RBSPs and their activation

This clause specifies the activation process of atlas sequence parameter sets (ASPSs) and atlas frame parameter sets (AFPSs).

NOTE 1 – The ASPS and AFPS mechanism decouples the transmission of infrequently changing information from the transmission of coded atlas data. ASPSs and AFPSs may, in some applications, be conveyed "out-of-band".

An AFPS RBSP includes parameters that can be referred to by the coded tile group NAL units of one or more coded atlas frames. Each AFPS RBSP is initially considered not active at the start of the operation of the decoding process. At most one AFPS RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular AFPS RBSP results in the deactivation of the previously-active AFPS RBSP.

When an AFPS RBSP (with a particular value of afps_atlas_frame_parameter_set_id) is not active and is referred to by a coded tile group NAL unit with nal_layer_id equal to 0 (using a value of atgh_atlas_frame_parameter_set_id equal to the afps_atlas_frame_parameter_set_id value), it is then activated. This AFPS RBSP is called the active AFPS RBSP until it is deactivated by the activation of another AFPS RBSP. An AFPS RBSP, with that particular value of afps_atlas_frame_parameter_set_id, shall be available to the decoding process prior to its activation, included in at least one access unit with TemporalId less than or equal to the TemporalId of the AFPS NAL unit or provided through external means, and the AFPS NAL unit containing the AFPS RBSP shall have nal_layer_id equal to 0.

Any AFPS NAL unit containing the value of afps_atlas_frame_parameter_set_id for the active AFPS RBSP for a coded atlas frame shall have the same content as that of the active AFPS RBSP for the coded atlas frame, unless it follows the last ACL NAL unit of the coded atlas frame and precedes the first ACL NAL unit of another coded atlas frame.

An ASPS RBSP includes parameters that can be referred to by one or more AFPS RBSPs. Each ASPS RBSP is initially considered not active at the start of the operation of the decoding process. At most one ASPS RBSP is considered active at any given moment during the operation of the decoding process, and the activation of any particular ASPS RBSP results in the deactivation of the previously-active ASPS RBSP.

When an ASPS RBSP (with a particular value of asps_atlas_sequence_parameter_set_id) is not already active and it is referred to by activation of a AFPS RBSP (in which afps_seq_parameter_set_id is equal to the asps_atlas_sequence_parameter_set_id value), it is activated. This ASPS RBSP is called the active ASPS RBSP until it is deactivated by the activation of another ASPS RBSP. An ASPS RBSP, with that particular value of asps_atlas_sequence_parameter_set_id, shall be available to the decoding process prior to its activation, included in at least one access unit with TemporalId equal to 0 or provided through external means, and the NAL unit containing the ASPS RBSP shall have nal_layer_id equal to 0. An activated ASPS RBSP shall remain active for the entire coded atlas sequence (CAS).

Any ASPS NAL unit with nal_layer_id equal to 0 containing the value of asps_atlas_sequence_parameter_set_id for the active ASPS RBSP for a CAS shall have the same content as that of the active ASPS RBSP for the CAS, unless it follows the last access unit of the CAS and precedes the first ACL NAL unit of another CAS.

All constraints that are expressed on the relationship between the values of the syntax elements and the values of variables derived from those syntax elements in ASPSs and AFPSs and other syntax elements are expressions of constraints that apply only to the active ASPS RBSP and the active AFPS RBSP. If any ASPS RBSP and AFPS RBSP is present that is never activated in the bitstream, its syntax elements shall have values that would conform to the specified constraints if it was activated by reference in an otherwise conforming bitstream.

During operation of the decoding process (see clause 8), the values of parameters of the active ASPS RBSP and the active AFPS RBSP are considered in effect. For interpretation of SEI messages, the values of the active ASPS RBSP and the active AFPS RBSP for the operation of the decoding process for the ACL NAL units of the coded atlas frame with nal_layer_id equal to 0 in the same access unit are considered in effect unless otherwise specified in the SEI message semantics.

7.4.5.3.3 Order of access units and association to CASs

An bitstream conforming to this Specification consists of one or more CASs.

A CAS consists of one or more access units. The order of NAL units and coded atlas frames and their association to access units is described in clause 7.4.5.3.4.

The first access unit of a CAS is an IRAP access unit with NoRaslOutputFlag equal to 1.

It is a requirement of bitstream conformance that, when present, the next access unit after an access unit that contains an end of sequence NAL unit or an end of an bitstream NAL unit shall be an IRAP access unit, which may be an IDR access unit, a GIDR access unit, a BLA access unit, a GBLA access unit, a CRA access unit, or a GCRA access unit.

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

This clause specifies the order of NAL units and coded atlas frames and their association to access units for CPCSs that conform to one or more of the profiles specified in Annex A and that are decoded using the decoding process specified in clauses 2 through 9.6.

An access unit consists of one coded picture with nal_layer_id equal to 0, zero or more ACL NAL units with nal_layer_id greater than 0 and zero or more non-ACL NAL units. The association of ACL NAL units to coded pictures is described in clause 7.4.5.3.5.

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

Let firstBlAFrmNalUnit be the first ACL NAL unit of a coded atlas frame with nal_layer_id equal to 0. The first of any of the following NAL units preceding firstBlAFrmNalUnit and succeeding the last ACL NAL unit preceding firstBlAFrmNalUnit, if any, specifies the start of a new access unit:

NOTE 1 – The last ACL NAL unit preceding firstBlAFrmNalUnit in decoding order may have nal_layer_id greater than 0.

– access or V-PCC access unit delimiter NAL unit with nal_layer_id equal to 0 (when present),

– ASPS NAL unit with nal_layer_id equal to 0 (when present),

– AFPS NAL unit with nal_layer_id equal to 0 (when present),

– Prefix SEI NAL unit with nal_layer_id equal to 0 (when present),

– NAL units with nal_unit_type in the range of NAL_RSV_NACL41..NAL_RSV_NACL44 with nal_layer_id equal to 0 (when present),

– NAL units with nal_unit_type in the range of NAL_UNSPEC_48..NAL_UNSPEC_55 with nal_layer_id equal to 0 (when present).

NOTE 2 – The first NAL unit preceding firstBlAFrmNalUnit and succeeding the last ACL NAL unit preceding firstBlAFrmNalUnit, if any, can only be one of the above-listed NAL units.

When there is none of the above NAL units preceding firstBlAFrmNalUnit and succeeding the last ACL NAL preceding firstBlAFrmNalUnit, if any, firstBlAFrmNalUnit starts a new access unit.

The order of the coded atlas frames and non-ACL NAL units within an access unit shall obey the following constraints:

– When an access unit delimiter or a V-PCC access unit delimiter NAL unit with nal_layer_id equal to 0 is present, it shall be the first NAL unit. There shall be at most one access unit delimiter or V-PCC access unit delimiter NAL unit with nal_layer_id equal to 0 in any access unit.

– When any ASPS NAL units, AFPS NAL units, prefix SEI NAL units, NAL units with nal_unit_type in the range of NAL_RSV_NACL_41..NAL_RSV_NACL_44, or NAL units with nal_unit_type in the range of NAL_UNSPEC_48..NAL_UNSPEC_55 are present, they shall not follow the last ACL NAL unit of the access unit.

– NAL units having nal_unit_type equal to NAL_FDT or NAL_SUFFIX_SEI or in the range of NAL_RSV_NACL_45..NAL_RSV_NACL_47 or NAL_UNSPEC_56..NAL_UNSPEC_63 shall not precede the first ACL NAL unit of the access unit.

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

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

NOTE 3 – Decoders conforming to profiles specified in Annex A do not use NAL units with nal_layer_id greater than 0, e.g., access or V-PCC access unit delimiter NAL units with nal_layer_id greater than 0, for access unit boundary detection, except for identification of a NAL unit as an ACL or non-ACL NAL unit.

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

This clause specifies the order of ACL NAL units and association to coded atlas frames.

Each ACL NAL unit is part of a coded atlas frame.

The order of the ACL NAL units within a coded atlas frame is constrained as follows:

– The first ACL NAL unit of the coded atlas frame shall have first_atgh_address equal to 0.

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

7.4.6.1 Atlas sequence parameter set RBSP semantics

asps_atlas_sequence_parameter_set_id provides an identifier for the atlas sequence parameter set for reference by other syntax elements. The value of asps_atlas_sequence_parameter_set_id shall be in the range of 0 to 15, inclusive.

asps_frame_width indicates the atlas frame width in terms of integer luma samples for the current atlas. It is a requirement of V-PCC bitstream conformance that the value of asps_frame_width shall be equal to the value of vps_frame_width[j], where j is the index of the current atlas.

asps_frame_height indicates the atlas frame height in terms of integer luma samples for the current atlas. It is a requirement of V-PCC bitstream conformance that the value of asps_frame_height shall be equal to the value of vps_frame_height[j], where j is the index of the current atlas.

During the reconstruction phase the decoded occupancy, geometry, and attribute videos shall be converted to the nominal width, height, and frame rate of the current atlas using appropriate scaling. Such scaling is outside the scope of this Specification.

asps_log2_patch_packing_block_size specifies the value of the variable PatchPackingBlockSize, that is used for the horizontal and vertical placement of the patches within the atlas, as follows:

$$\text{PatchPackingBlockSize} = 2^{(\text{ asps\_log2\_patch\_packing\_block\_size})} \quad (7\text{-}3)$$

The value of asps_log2_patch_packing_block_size shall be in the range of 0 to 7, inclusive.

asps_log2_max_atlas_frame_order_cnt_lsb_minus4 specifies the value of the variable MaxAtlasFrmOrderCntLsb that is used in the decoding process for the atlas frame order count as follows:

$$\text{MaxAtlasFrmOrderCntLsb} = 2^{(\text{ asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4} + 4\text{ })} \quad (7\text{-}4)$$

The value of asps_log2_max_atlas_frame_order_cnt_lsb_minus4 shall be in the range of 0 to 12, inclusive.

asps_max_dec_atlas_frame_buffering_minus1 plus 1 specifies the maximum required size of the decoded atlas frame buffer for the CAS in units of atlas frame storage buffers. The value of asps_max_dec_atlas_frame_buffering_minus1 shall be in the range of 0 to MaxDafbSize − 1, inclusive, where MaxDafbSize is as specified is Annex A.

asps_long_term_ref_atlas_frames_flag equal to 0 specifies that no long term reference atlas frame is used for inter prediction of any coded atlas frame in the CAS. asps_long_term_ref_atlas_frames_flag equal to 1 specifies that long term reference atlas frames may be used for inter prediction of one or more coded atlas frames in the CAS.

asps_num_ref_atlas_frame_lists_in_asps specifies the number of the ref_list_struct(rlsIdx) syntax structures included in the atlas sequence parameter set. The value of asps_num_ref_atlas_frame_lists_in_asps shall be in the range of 0 to 64, inclusive.

NOTE 1 – A decoder should allocate memory for a total number of ref_list_struct(rlsIdx) syntax structures equal to (num_ref_atlas_frame_lists_in_sps[i] + 1) since there may be one ref_list_struct(rlsIdx) syntax structure directly signalled in the atlas tile group headers of the current atlas tile group.

asps_use_eight_orientations_flag equal to 0 specifies that the patch orientation index for a patch with index j in a frame with index i, pdu_orientation_index[i][j], is in the range of 0 to 1, inclusive. asps_use_eight_orientations_flag equal to 1 specifies that the patch orientation index for a patch with index j in a frame with index i, pdu_orientation_index[i][j], is in the range of 0 to 7, inclusive.

asps_45degree_projection_patch_present_flag equal to 0 specifies that the patch projection information is not signalled for the current atlas tile group. asps_45degree_projection_present_flag equal to 1 specifies that the patch projection information is signalled for the current atlas tile group. When asps_45degree_projection_present_flag is not present, its value is inferred to be equal to 0.

asps_normal_axis_limits_quantization_enabled_flag equal to 1 specifies that quantization parameters shall be signalled and used for quantizing the normal axis related elements of a patch data unit, a merge patch data unit, or an inter patch data unit. If asps_normal_axis_limits_quantization_enabled_flag is equal to 0, then no quantization is applied on any normal axis related elements of a patch data unit, a merge patch data unit, or an inter patch data unit.

asps_normal_axis_max_delta_value_enabled_flag equal to 1 specifies that the maximum nominal shift value of the normal axis that may be present in the geometry information of a patch with index i in a frame with index j will be indicated in the bitstream for each patch data unit, a merge patch data unit, or an inter patch data unit. If asps_normal_axis_max_delta_value_enabled_flag is equal to 0 then the maximum nominal shift value of the normal axis that may be present in the geometry information of a patch with index i in a frame with index j shall not be be indicated in the bitstream for each patch data unit, a merge patch data unit, or an inter patch data unit.

asps_remove_duplicate_point_enabled_flag equal to 1 indicates that duplicated points shall not be reconstructed for the current atlas, where a duplicated point is a point with the same 2D and 3D geometry coordinates as another point from a lower index map. asps_remove_duplicate_point_enabled_flag equal to 0 indicates that all points shall be reconstructed.

asps_pixel_deinterleaving_flag equal to 1 indicates that the decoded geometry and attribute videos for the current atlas contain spatially interleaved pixels from two maps. asps_pixel_deinterleaving_flag equal to 0 indicates that the decoded geometry and attribute videos corresponding to the current atlas contain pixels from only a single map.

asps_patch_precedence_order_flag equal to 1 indicates that patch precedence for the current atlas is the same as the decoding order. asps_patch_precedence_order_flag equal to 0 indicates that patch precedence for the current atlas is the reverse of the decoding order.

asps_patch_size_quantizer_present_flag equal to 1 indicates that the patch size quantization parameters are present in an atlas tile group header. If asps_patch_size_quantizer_present_flag is equal to 0, then the patch size quantization parameters are not present.

asps_enhanced_occupancy_map_for_depth_flag equal to 1 indicates that the decoded occupancy map video for the current atlas contains information related to whether intermediate depth positions between two depth maps are occupied. asps_enhanced_occupancy_map_for_depth_flag equal to 0 indicates that the decoded occupancy map video does not contain information related to whether intermediate depth positions between two depth maps are occupied.

It is a requirement of bitstream conformance that if asps_enhanced_occupancy_map_for_depth_flag is equal to 1, oi_lossy_occupancy_map_compression_threshold shall be equal to 0.

asps_point_local_reconstruction_enabled_flag equal to 1 indicates that point local reconstruction mode information may be present in the bitstream for the current atlas. asps_point_local_reconstruction_enabled_flag equal to 0 indicates that no information related to the point local reconstruction mode is present in the bitstream for the current atlas.

It is a requirement of bitstream conformance that asps_pixel_deinterleaving_flag and asps_point_local_reconstruction_enabled_flag shall not be one simultaneously.

asps_map_count_minus1 plus 1 indicates the number of maps that may be used for encoding the geometry and attribute data for the current atlas. asps_map_count_minus1 shall be in the range of 0 to 15, inclusive. It is a requirement of bitstream conformance to this Specification that asps_map_count_minus1 is equal to vps_map_count_minus1[atlasId] , where atlasId is the index of the current atlas.

asps_enhanced_occupancy_map_fix_bit_count_minus1 plus 1 indicates the size in bits of the EOM codeword. asps_enhanced_occupancy_map_fix_bit_count shall be in the range of 0 to oi_occupancy_nominal_2d_bitdepth_minus1[atlasId] − 1, inclusive, where atlasId is the index of the current atlas.

asps_surface_thickness_minus1 plus 1 specifies the maximum absolute difference beween an explicitly coded depth value and interpolated depth value when asps_pixel_deinterleaving_flag or asps_point_local_reconstruction_flag is equal to 1.

asps_vui_parameters_present_flag equal to 1 specifies that the vui_parameters( ) syntax structure as specified in Annex E is present. asps_vui_parameters_present_flag equal to 0 specifies that the vui_parameters( ) syntax structure as specified in Annex E is not present.

asps_extension_present_flag equal to 0 specifies that no asps_extension_data_flag syntax elements are present in the syntax structure. asps_extension_present_flag equal to 1 specifies that there are asps_extension_data_flag syntax elements present in the syntax structure.

asps_extension_data_flag may have any value. When present, asps_extension_data_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. Values of asps_extension_data_flag not equal to 0 are reserved for future use by ISO/IEC.

7.4.6.2 Point local reconstruction information semantics

plri_point_local_reconstruction_map_enabled_flag[i] equal to 1 indicates that point local reconstruction mode information is present in the bitstream for the map of index i of the current atlas. plri_point_local_reconstruction_map_enabled_flag equal to 0 indicates that no information related to the point local reconstruction mode is present in the bitstream for the map of index i of the current atlas.

plri_number_of_modes_minus1[mapIdx] plus 1 indicates the number of reconstruction modes specified for the point local reconstruction process associated with the map of index mapIdx of the current atlas. The value of plri_number_of_modes_minus1[mapIdx] shall be in the range of 0 to 15, inclusive.

plri_interpolate_flag[mapIdx][j] equal to 1 indicates that point interpolation for the map of index mapIdx of the current atlas may be used for point local reconstruction mode j. plri_interpolate_flag[mapIdx][j] equal to 0 indicates that no point interpolation for the map of index mapIdx of the current atlas is used for point local reconstruction mode j.

plri_filling_flag[mapIdx][j] equal to 1 indicates that filling for the map of index mapIdx of the current atlas may be used for point local reconstruction mode j. plri_filling_flag[mapIdx][j] equal to 0 indicates that no filling for the map of index mapIdx of the current atlas is used for point local reconstruction mode j.

plri_minimum_depth[mapIdx][j] specifies the value of the minimum depth parameter of the point local reconstruction mode j for the map of index mapIdx of the current atlas. plri_minimum_depth[mapIdx][j] shall be in the range of 0 to 3, inclusive.

plri_neighbour_minus1[mapIdx][j] plus 1 specifies the size of the 2D neighbourhood for point local construction mode j for the map of index mapIdx of the current atlas. plri_neighbour_minus1[mapIdx][j] shall be in the range of 0 to 3, inclusive.

plri_block_threshold_per_patch_minus1[mapIdx] plus 1 specifies the value representative of the threshold defining the value of plrdLevel. plri_block_threshold_per_patch_minus1[mapIdx] shall be in the range of 0 to 63, inclusive.

7.4.6.3 Atlas frame parameter set RBSP semantics

afps_atlas_frame_parameter_set_id identifies the atlas frame parameter set for reference by other syntax elements. The value of afps_atlas_frame_parameter_set_id shall be in the range of 0 to 63, inclusive.

afps_atlas_sequence_parameter_set_id specifies the value of asps_atlas_sequence_parameter_set_id for the active atlas sequence parameter set. The value of afps_atlas_sequence_parameter_set_id shall be in the range of 0 to 15, inclusive.

afps_num_ref_idx_default_active_minus1 plus 1 specifies the inferred value of the variable NumRefIdxActive for the tile group with atgh_num_ref_idx_active_override_flag equal to 0. The value of afps_num_ref_idx_default_active_minus1 shall be in the range of 0 to 14, inclusive.

afps_additional_lt_afoc_lsb_len specifies the value of the variable MaxLtAtlasFrmOrderCntLsb that is used in the decoding process for reference atlas frame lists as follows:

$$\text{MaxLtAtlasFrmOrderCntLsb} = 2^{(\text{asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4} + 4 + \text{afps\_additional\_lt\_afoc\_lsb\_len})} \quad (7\text{-}5)$$

The value of afps_additional_lt_afoc_lsb_len shall be in the range of 0 to 32 − asps_log2_max_atlas_frame_order_cnt_lsb_minus4 − 4, inclusive. When asps_long_term_ref_atlas_frames_flag is equal to 0, the value of afps_additional_lt_afoc_lsb_len shall be equal to 0.

afps_2d_pos_x_bit_count_minus1 plus 1 specifies the number of bits in the fixed-length representation of pdu_2d_pos_x[j] of patch with index j in an atlas tile group that refers to afps_atlas_frame_parameter_set_id.

afps_2d_pos_y_bit_count_minus1 plus 1 specifies the number of bits in the fixed-length representation of pdu_2d_pos_y[j] of patch with index j in an atlas tile group that refers to afps_atlas_frame_parameter_set_id.

afps_3d_pos_x_bit_count_minus1 plus 1 specifies the number of bits in the fixed-length representation of pdu_3d_pos_x[j] of patch with index j in an atlas tile group that refers to afps_atlas_frame_parameter_set_id.

afps_3d_pos_y_bit_count_minus1 plus 1 specifies the number of bits in the fixed-length representation of pdu_3d_pos_y[j] of patch with index j in an atlas tile group that refers to afps_atlas_frame_parameter_set_id.

afps_lod_bit_count specifies the number of bits in the fixed-length representation of pdu_lod[j] of patch with index j in an atlas tile group that refers to afps_atlas_frame_parameter_set_id.

afps_override_eom_for_depth_flag equal to 1 indicates that the values of afps_eom_number_of_patch_bit_count_minus1 and afps_eom_max_bit_count_minus1 is explicitly present in the bitstream. afps_override_eom_for_depth_flag equal to 0 indicates that the values of afps_eom_number_of_patch_bit_count_minus1 and afps_eom_max_bit_count_minus1 are implicitly derived.

afps_eom_number_of_patch_bit_count_minus1 plus 1 specifies the number of bits used to represent the number of geometry patches associated with the current EOM attribute patch. afps_eom_number_of_patch_bit_count_minus1 shall be in the range of 0 to 15, inclusive. When not present, the value of afps_eom_number_of_patch_bit_count_minus1 is inferred to be equal to 7.

afps_eom_max_bit_count_minus1 plus 1 specifies the number of bits used to represent the number of EOM points per geometry patch associated with the current EOM attribute patch. afps_eom_max_bit_count_minus1 shall be in the range of 0 to 15, inclusive. When not present, the value of afps_eom_max_bit_count_minus1 is inferred to be equal to 7.

afps_raw_3d_pos_bit_count_explicit_mode_flag equal to 1 indicates that the bit count for rpdu_3d_pos_x, rpdu_3d_pos_y, and rpdu_3d_pos_z is explicitely coded in an atlas tile group header that refers to afps_atlas_frame_parameter_set_id. afps_raw_3d_pos_bit_count_explicit_mode_flag equal to 0 indicates that the bit count for rpdu_3d_pos_x, rpdu_3d_pos_y, and rpdu_3d_pos_z shall be set to gi_geometry_3d_coordinates_bitdepth_minus1 − gi_geometry_nominal_2d_bitdepth_minus1.

afps_extension_present_flag equal to 0 specifies that no afps_extension_data_flag syntax elements are present in the syntax structure. afps_extension_present_flag equal to 1 specifies that there are afps_extension_data_flag syntax elements present in the syntax structure.

afps_extension_data_flag may have any value. When present, afps_extension_data_flag shall be equal to 0 in bitstreams conforming to this version of this Specification. Values of afps_extension_data_flag not equal to 0 are reserved for future use by ISO/IEC.

7.4.6.4 Atlas frame tile information syntax

afti_single_tile_in_atlas_frame_flag equal to 1 specifies that there is only one tile in each atlas frame referring to the AFPS. afti_single_tile_in_atlas_frame_flag equal to 0 specifies that there is more than one tile in each atlas frame referring to the PPS.

afti_uniform_tile_spacing_flag equal to 1 specifies that tile column and row boundaries are distributed uniformly across the atlas frame and signalled using the syntax elements afti_tile_cols_width_minus1 and afti_tile_rows_height_minus1, respectively. afti_uniform_tile_spacing_flag equal to 0 specifies that tile column and row boundaries may or may not be distributed uniformly across the atlas frame and are signalled using the syntax elements afti_num_tile_columns_minus1 and afti_num_tile_rows_minus1 and a list of syntax element pairs afti_tile_column_width_minus1[i] and afti_tile_row_height_minus1[i]. When not present, the value of afti_uniform_tile_spacing_flag is inferred to be equal to 1.

afti_tile_cols_width_minus1 plus 1 specifies the width of the tile columns excluding the right-most tile column of the atlas frame in units of 64 samples when afti_uniform_tile_spacing_flag is equal to 1. The value of afti_tile_cols_width_minus1 shall be in the range of 0 to (asps_frame_width + 63) / 64 − 1, inclusive. When not present, the value of afti_tile_cols_width_minus1 is inferred to be equal to (asps_frame_width + 63) / 64 − 1**.**

afti_tile_rows_height_minus1 plus 1 specifies the height of the tile rows excluding the bottom tile row of the atlas frame in units of 64 samples when afti_uniform_tile_spacing_flag is equal to 1. The value of afti_tile_rows_height_minus1 shall be in the range of 0 to (asps_frame_height + 63) / 64 − 1, inclusive. When not present, the value of afti_tile_rows_height_minus1 is inferred to be equal to (asps_frame_height + 63) / 64 − 1.

afti_num_tile_columns_minus1 plus 1 specifies the number of tile columns partitioning the atlas frame when afti_uniform_tile_spacing_flag is equal to 0. The value of afti_num_tile_columns_minus1 shall be in the range of 0 to (asps_frame_width + 63) / 64 − 1, inclusive. If afti_single_tile_in_atlas_frame_flag is equal to 1, the value of afti_num_tile_columns_minus1 is inferred to be equal to 0. Otherwise, when afti_uniform_tile_spacing_flag is equal to 1, the value of afti_num_tile_columns_minus1 is inferred as specified in clause 6.4.1.

afti_num_tile_rows_minus1 plus 1 specifies the number of tile rows partitioning the atlas frame when pti_uniform_tile_spacing_flag is equal to 0. The value of pti_num_tile_rows_minus1 shall be in the range of 0 to (asps_frame_height + 63) / 64 − 1, inclusive. If afti_single_tile_in_pic_flag is equal to 1, the value of afti_num_tile_rows_minus1 is inferred to be equal to 0. Otherwise, when afti_uniform_tile_spacing_flag is equal to 1, the value of afti_num_tile_rows_minus1 is inferred as specified in clause 6.4.1.

The variable NumTilesInAtlasFrame is set equal to (afti_num_tile_columns_minus1 + 1) * (afti_num_tile_rows_minus1 + 1).

When afti_single_tile_in_atlas_frame_flag is equal to 0, NumTilesInAtlasFrame shall be greater than 1.

afti_tile_column_width_minus1[i] plus 1 specifies the width of the i-th tile column in units of 64 samples.

afti_tile_row_height_minus1[i] plus 1 specifies the height of the i-th tile row in units of 64 samples.

afti_single_tile_per_tile_group_flag equal to 1 specifies that each tile group that refers to this AFPS includes one tile. afti_single_tile_per_tile_group_flag equal to 0 specifies that a tile group that refers to this AFPS may include more than one tile. When not present, the value of afti_single_tile_per_tile_group_flag is inferred to be equal to 1.

afti_num_tile_groups_in_atlas_frame_minus1 plus 1 specifies the number of tile groups in each atlas frame referring to the AFPS. The value of afti_num_tile_groups_in_atlas_frame_minus1 shall be in the range of 0 to NumTilesInAtlasFrame − 1, inclusive. When not present and afti_single_tile_per_tile_group_flag is equal to 1, the value of afti_num_tile_groups_in_atlas_frame_minus1 is inferred to be equal to NumTilesInAtlasFrame − 1.

afti_top_left_tile_idx[i] specifies the tile index of the tile located at the top-left corner of the i-th tile group. The value of afti_top_left_tile_idx[i] shall not be equal to the value of afti_top_left_tile_idx[j] for any i not equal to j. When not present, the value of afti_top_left_tile_idx[i] is inferred to be equal to i. The length of the afti_top_left_tile_idx[i] syntax element is Ceil(Log2(NumTilesInAtlasFrame) bits.

afti_bottom_right_tile_idx_delta[i] specifies the difference between the tile index of the tile located at the bottom-right corner of the i-th tile group and afti_top_left_tile_idx[i]. When afti_single_tile_per_tile_group_flag is equal to 1, the value of afti_bottom_right_tile_idx_delta[i] is inferred to be equal to 0. The length of the afti_bottom_right_tile_idx_delta[i] syntax element is Ceil(Log2(NumTilesInAtlasFrame − afti_top_left_tile_idx[i])) bits.

The variables TopLeftTileColumn[i], TopLeftTileRow[i], BottomRightTileColumn[i], and BottomRightTileRow[i], which specify the corresponding tile column and row positions for the top left an bottom right tiles in a tile group are computed as follows:

```
    TopLeftTileColumn[ i ] = afti_top_left_tile_idx[ i ] % (afti_num_tile_rows_minus1 + 1)
    TopLeftTileRow[ i ] = afti_top_left_tile_idx[ i ] / (afti_num_tile_rows_minus1  + 1)
    botRightTileIdx = afti_top_left_tile_idx[ i ] + afti_bottom_right_tile_idx_delta[ i ]
    BottomRightTileColumn[ i ] = botRightTileIdx % (afti_num_tile_rows_minus1 + 1)
    BottomRightTileRow[ i ] ] = botRightTileIdx / (afti_num_tile_rows_minus1  + 1)
```

The variables NumTilesInTileGroups[i] and TilesToTileGroupMap[j], which specify the number of tiles in the i-th tile group and the mapping of tiles to tile groups, are derived as follows:

```
    NumTilesInTileGroups[ i ] = 0
    for( j = 0; j < NumTilesInAtlasFrame; j++) {
        if(  v  >=  TileColBd [ afti_top_left_tile_idx[ i ] ]  &&
                    TileColBd [ j ]  <=  TileColBd [ botRightTileIdx]  &&
                    TileRowBd[ j ]  >=  TileRowBd [ afti_top_left_tile_idx[ i ] ] &&          (7-6)
                    TileRowBd [ j ] <= TileRowBd [ botRightTileIdx] ) {
```

```
            NumTilesInTileGroups[ i ]++
            TilesToTileGroupMap[ j ] = i
        }
    }
```

The variables TileGroupWidth[i] and TileGroupHeight[i], which specify the width and height of a tile group respectively, are then computed as follows:

```
    TileGroupWidth[ i ] = 0
    TileGroupHeight[ i ] = 0
    for (j= TopLeftTileColumn[ i ]; j <= BottomRightTileColumn[i]; j++) {
        TileGroupWidth[ i ] += ColWidth[ j ] * 64
    }
    for (j= TopLeftTileRow[ i ]; j <= BottomRightTileRow[i]; j++) {
        TileGroupHeight[ i ] += ColHeight[ j ] * 64
    }
```

afti_signalled_tile_group_id_flag equal to 1 specifies that the tile group ID for each tile group is signalled. afti_signalled_tile_group_id_flag equal to 0 specifies that tile group IDs are not signalled.

afti_signalled_tile_group_id_length_minus1 plus 1 specifies the number of bits used to represent the syntax element afti_tile_group_id[i] when present, and the syntax element atgh_address in tile group headers. The value of afti_signalled_tile_group_id_length_minus1 shall be in the range of 0 to 15, inclusive. When not present, the value of afti_signalled_tile_group_id_length_minus1 is inferred to be equal to Ceil(Log2(afti_num_tile_groups_in_atlas_frame_minus1 + 1)) − 1.

afti_tile_group_id[i] specifies the tile group ID of the i-th tile group. The length of the afti_tile_group_id[i] syntax element is afti_signalled_tile_group_id_length_minus1 + 1 bits. When not present, the value of afti_tile_group_id[i] is inferred to be equal to i, for each i in the range of 0 to afti_num_tile_groups_in_atlas_frame_minus1, inclusive.

7.4.6.5 Supplemental enhancement information RBSP semantics

Supplemental enhancement information (SEI) contains information that is not necessary to decode the samples of coded atlas frames from ACL NAL units. An SEI RBSP contains one or more SEI messages.

7.4.6.6 Access unit delimiter RBSP semantics

The access unit delimiter may be used to indicate the type of tile groups present in the coded atlas frames in the access unit containing the access or V-PCC access unit delimiter NAL unit and to simplify the detection of the boundary between access units. There is no normative decoding process associated with the access unit delimiter.

aframe_type indicates that the atgh_type values for all tile groups of the coded atlas frame in the access unit containing the access unit delimiter or V-PCC access unit delimiter NAL unit are members of the set listed in Table 7-4 for the given value of aframe_type. The value of aframe_type shall be equal to 0, 1, 2, or 3 in bitstreams conforming to this version of this Specification. Other values of aframe_type are reserved for future use by ISO/IEC. Decoders conforming to this version of this Specification shall ignore reserved values of aframe_type.

Table 7-4 – Interpretation of aframe_type

aframe_type	atgh_type values that may be present in the coded atlas frame
0	I
1	P, I

2	SKIP, P, I
3	SKIP

7.4.6.7 End of sequence RBSP semantics

When present, the end of sequence RBSP specifies that the current access unit is the last access unit in the coded atlas sequence in decoding order and the next subsequent access unit in the bitstream in decoding order (if any) is an IRAP access unit. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty.

7.4.6.8 End of bitstream RBSP semantics

The end of bitstream RBSP indicates that no additional NAL units are present in the bitstream that are subsequent to the end of bitstream RBSP in decoding order. The syntax content of the SODB and RBSP for the end of bitstream RBSP are empty.

7.4.6.9 Filler data RBSP semantics

The filler data RBSP contains bytes whose value shall be equal to 0xFF. No normative decoding process is specified for a filler data RBSP.

ff_byte is a byte equal to 0xFF.

7.4.6.10 Atlas tile group layer rbsp semantics

None.

7.4.6.11 Atlas tile group header semantics

atgh_atlas_frame_parameter_set_id specifies the value of afps_atlas_frame_parameter_set_id for the active atlas frame parameter set for the current atlas tile group. The value of atgh_atlas_frame_parameter_set_id shall be in the range of 0 to 63, inclusive.

atgh_address specifies the tile group address of the tile group. When not present, the value of atgh_address is inferred to be equal to 0.

The following applies:

– The tile group address is the tile group ID of the tile group.

– The length of atgh_address is afti_signalled_tile_group_id_length_minus1 + 1 bits.

– If afti_signalled_tile_group_id_flag is equal to 0, the value of atgh_address shall be in the range of 0 to afti_num_tile_groups_in_atlas_frame_minus1, inclusive. Otherwise, the value of atgh_address shall be in the range of 0 to $2^{(\text{afti\_signalled\_tile\_group\_id\_length\_minus1} + 1)} - 1$, inclusive.

It is a requirement of bitstream conformance that the following constraints apply:

– The value of atgh_address shall not be equal to the value of atgh_address of any other coded atlas tile group unit of the same coded atlas frame.

– The tile groups of an atlas frame shall be in increasing order of their atgh_address values.

atgh_type specifies the coding type of the current atlas tile goup according to Table 7-5.

Table 7-5 – Name association to atgh_type

atgh_type	Name of atgh_type
0	P (Inter atlas tile group)
1	I (Intra atlas tile group)
2	SKIP (SKIP atlas tile group)

atgh_atlas_frm_order_cnt_lsb specifies the atlas frame order count modulo MaxAtlasFrmOrderCntLsb for the current atlas tile group. The length of the atgh_atlas_frm_order_cnt_lsb syntax element is equal to asps_log2_max_atlas_frame_order_cnt_lsb_minus4 + 4 bits. The value of the atgh_atlas_frm_order_cnt_lsb shall be in the range of 0 to MaxAtlasFrmOrderCntLsb − 1, inclusive. When atgh_atlas_frm_order_cnt_lsb is not present, it shall be inferred to be equal to 0.

atgh_ref_atlas_frame_list_sps_flag equal to 1 specifies that the reference atlas frame list of the current atlas tile group is derived based on one of the ref_list_struct(rlsIdx) syntax structures in the active ASPS. atgh_ref_atlas_frame_list_sps_flag equal to 0 specifies that the reference atlas frame list of the current atlas tile list is derived based on the ref_list_struct(rlsIdx) syntax structure that is directly included in the tile group header of the current atlas tile group. When asps_num_ref_atlas_frame_lists_in_asps is equal to 0, the value of atgh_ref_atlas_frame_list_sps_flag is inferred to be equal to 0.

atgh_ref_atlas_frame_list_idx specifies the index, into the list of the ref_list_struct(rlsIdx) syntax structures included in the active ASPS, of the ref_list_struct(rlsIdx) syntax structure that is used for derivation of the reference atlas frame list for the current atlas tile group. The syntax element atgh_ref_atlas_frame_list_idx is represented by Ceil(Log2(asps_num_ref_atlas_frame_lists_in_asps)) bits. When not present, the value of atgh_ref_atlas_frame_list_idx is inferred to be equal to 0. The value of atgh_ref_atlas_frame_list_idx shall be in the range of 0 to asps_num_ref_atlas_frame_lists_in_asps − 1, inclusive. When atgh_ref_atlas_frame_list_sps_flag is equal to 1 and asps_num_ref_atlas_frame_lists_in_asps is equal to 1, the value of atgh_ref_atlas_frame_list_idx is inferred to be equal to 0.

The variable RlsIdx for the current atlas tile group is derived as follows:

```
RlsIdx = asps_num_ref_atlas_frame_lists_in_asps ?
              atgh_ref_atlas_frame_list_idx : asps_num_ref_atlas_frame_lists_in_asps          (7-7)
```

atgh_additional_afoc_lsb_present_flag[j] equal to 1 specifies that atgh_additional_afoc_lsb_val[j] is present for the current atlas tile group. atgh_additional_afoc_lsb_present_flag[j] equal to 0 specifies that atgh_additional_afoc_lsb_val[j] is not present.

atgh_additional_afoc_lsb_val[j] specifies the value of FullAtlasFrmOrderCntLsbLt[RlsIdx][j] for the current atlas tile group as follows:

```
FullAtlasFrmOrderCntLsbLt[ RlsIdx ][ j ] =
         atgh_additional_afoc_lsb_val[ j ] * MaxAtlasFrmOrderCntLsb +
         afoc_lsb_lt[ RlsIdx ][ j ]                                                        (7-8)
```

The syntax element atgh_additional_afoc_lsb_val[j] is represented by afps_additional_lt_afoc_lsb_len bits. When not present, the value of atgh_additional_afoc_lsb_val[j] is inferred to be equal to 0.

atgh_pos_min_z_quantizer specifies the quantizer that is to be applied to the pdu_3d_pos_min_z[p] value of the patch p. If atgh_pos_min_z_quantizer is not present, its value shall be inferred to be equal to 0.

atgh_pos_delta_max_z_quantizer specifies the quantizer that is to be applied to the pdu_3d_pos_delta_max_z[p] value of the patch p. If atgh_pos_delta_max_z_quantizer is not present, its value shall be inferred to be equal to 0.

atgh_patch_size_x_info_quantizer specifies the value of the quantizer PatchSizeXQuantizer that is to be applied to the pdu_2d_delta_size_x value of the patch p. If atgh_patch_size_x_info_quantizer is not present, its value shall be inferred to be equal to asps_log2_patch_packing_block_size. The value of PatchSizeXQuantizer is computed as follows:

PatchSizeXQuantizer = 1 << atgh_patch_size_x_info_quantizer (7-9)

The value of atgh_patch_size_x_info_quantizer shall be in the range of 0 to asps_log2_patch_packing_block_size, inclusive.

atgh_patch_size_y_info_quantizer specifies the value of the quantizer PatchSizeYQuantizer that is to be applied to the pdu_2d_delta_size_y value of the patch p. If atgh_patch_size_y_info_quantizer is not present, its value shall be inferred to be equal to asps_log2_patch_packing_block_size. The value of PatchSizeYQuantizer is computed as follows:

PatchSizeYQuantizer = 1 << atgh_patch_size_y_info_quantizer (7-10)

The value of atgh_patch_size_y_info_quantizer shall be in the range of 0 to asps_log2_patch_packing_block_size, inclusive.

atgh_raw_3d_pos_axis_bit_count_minus1 plus 1 specifies the bit count of rpdu_3d_pos_x, rpdu_3d_pos_y, and rpdu_3d_pos_z. When not present, atgh_raw_3d_pos_axis_bit_count_minus1 is equal to gi_geometry_3d_coordinates_bitdepth_minus1 − gi_geometry_nominal_2d_bitdepth_minus1 − 1. The number of bits used to represent atgh_raw_3d_pos_axis_bit_count_minus1 is log2(gi_geometry_3d_coordinates_bitdepth_minus1 + 1).

atgh_num_ref_idx_active_override_flag equal to 1 specifies that the syntax element atgh_num_ref_idx_active_minus1 is present for the current atlas tile group. atgh_num_ref_idx_active_override_flag equal to 0 specifies that the syntax element atgh_num_ref_idx_active_minus1 is not present. If atgh_num_ref_idx_active_override_flag is not present, its value shall be inferred to be equal to 0.

atgh_num_ref_idx_active_minus1 is used for the derivation of the variable NumRefIdxActive as specified by Equation 7-11 for the current atlas tile group. The value of atgh_num_ref_idx_active_minus1 shall be in the range of 0 to 14, inclusive.

When the current atlas tile group is a P_TILE_GRP atlas tile group, atgh_num_ref_idx_active_override_flag is equal to 1, and atgh_num_ref_idx_active_minus1 is not present, atgh_num_ref_idx_active_minus1 is inferred to be equal to 0.

The variable NumRefIdxActive is derived as follows:

```
if( atgh_type  = =  P_TILE_GRP || atgh_type  = =  SKIP_TILE_GRP ) {
    if( atgh_num_ref_idx_active_override_flag = = 1 )
        NumRefIdxActive = atgh_num_ref_idx_active_minus1 + 1                    (7-11)
    else {
        if( num_ref_entries[ RlsIdx ]  >=  afps_num_ref_idx_default_active_minus1+ 1 )
            NumRefIdxActive = afps_num_ref_idx_default_active_minus1+ 1
        else
            NumRefIdxActive = num_ref_entries[ RlsIdx ]
    }
} else
```

```
        NumRefIdxActive = 0
    }
```

The value of NumRefIdxActive − 1 specifies the maximum reference index for reference the atlas frame list that may be used to decode the current atlas tile group. When the value of NumRefIdxActive is equal to 0, no reference index for the reference atlas frame list may be used to decode the current atlas tile group.

7.4.6.12 Reference list structure semantics

num_ref_entries[rlsIdx] specifies the number of entries in the ref_list_struct(rlsIdx) syntax structure. The value of num_ref_entries[rlsIdx] shall be in the range of 0 to asps_max_dec_atlas_frame_buffering_minus1 + 1, inclusive.

st_ref_atlas_frame_flag[rlsIdx][i] equal to 1 specifies that the i-th entry in the ref_list_struct(rlsIdx) syntax structure is a short term reference atlas frame entry. st_ref_atlas_frame_flag[rlsIdx][i] equal to 0 specifies that the i-th entry in the ref_list_struct(rlsIdx) syntax structure is a long term reference atlas frame entry. When not present, the value of st_ref_atlas_frame_flag[rlsIdx][i] is inferred to be equal to 1.

The variable NumLtrAtlasFrmEntries[rlsIdx] is derived as follows:

```
    NumLtrpfEntries[ rlsIdx ] = 0
    for( i = 0; i < num_ref_entries[ rlsIdx ]; i++ )
        if( !st_ref_atlas_frame_flag[ rlsIdx ][ i ] )                          (7-12)
            NumLtrAtlasFrmEntries[ rlsIdx ]++
```

abs_delta_afoc_st[rlsIdx][i], when the i-th entry is the first short term reference atlas frame entry in ref_list_struct(rlsIdx) syntax structure, specifies the absolute difference between the atlas frame order count values of the current atlas tile group and the atlas frame referred to by the i-th entry, or, when the i-th entry is a short term reference atlas frame entry but not the first short term reference atlas frame entry in the ref_list_struct(rlsIdx) syntax structure, specifies the absolute difference between the atlas frame order count values of the atlas frames referred to by the i-th entry and by the previous short term reference atlas frame entry in the ref_list_struct(rlsIdx) syntax structure.

The value of abs_delta_afoc_st[rlsIdx][i] shall be in the range of 0 to $2^{15} - 1$, inclusive.

strpf_entry_sign_flag[rlsIdx][i] equal to 1 specifies that i-th entry in the syntax structure ref_list_struct(rlsIdx) has a value greater than or equal to 0. strpf_entry_sign_flag[rlsIdx][i] equal to 0 specifies that the i-th entry in the syntax structure ref_list_struct(rlsIdx) has a value less than 0. When not present, the value of strpf_entry_sign_flag[rlsIdx][i] is inferred to be equal to 1.

The list DeltaAfocSt[rlsIdx][i] is derived as follows:

```
    for( i = 0; i < num_ref_entries[ rlsIdx ]; i++ )
        if( st_ref_atlas_frame_flag[ rlsIdx ][ i ] )
            DeltaAfocSt[ rlsIdx ][ i ] =
                (2 * strpf_entry_sign_flag[ rlsIdx ][ i ] − 1) * abs_delta_afoc_st[ rlsIdx ][ i ]     (7-13)
        else
            DeltaAfocSt[ rlsIdx ][ i ] = 0
```

afoc_lsb_lt[rlsIdx][i] specifies the value of the atlas frame order count modulo MaxAtlasFrmOrderCntLsb of the atlas frame referred to by the i-th entry in the ref_list_struct(rlsIdx) syntax structure. The length of the afoc_lsb_lt[rlsIdx][i] syntax element is asps_log2_max_atlas_frame_order_cnt_lsb_minus4 + 4 bits.

7.4.7 Atlas tile group data unit semantics

7.4.7.1 General atlas tile group data unit semantics

atgdu_patch_mode[p] indicates the patch mode for the patch with index p in the current atlas tile group. The permitted values for atgdu_patch_mode[p] are specified in Table 7-6 for atlas tile groups with atgh_type = I_TILE_GRP, Table 7-7 for atlas tile groups with atgh_type = P_TILE_GRP, and Table 7-8 for atlas tile groups with atgh_type = SKIP_TILE_GRP. A tile group with atgh_type = SKIP_TILE_GRP implies that the entire tile group information is copied directly from the tile group with the same atgh_address as that of the current tile group that corresponds to the first reference atlas frame in the RefAtlasFrmList.

Table 7-6 Patch mode types for I_TILE_GRP type atlas tile groups

atgdu_patch_mode	Identifier	Description
0	I_INTRA	Non-predicted Patch mode
1	I_RAW	RAW Point Patch mode
2	I_EOM	EOM Point Patch mode
3-13	I_RESERVED	Reserved modes
14	I_END	Patch termination mode

Table 7-7 Patch mode types for P_TILE_GRP type atlas tile groups

atgdu_patch_mode	Identifier	Description
0	P_SKIP	Patch Skip mode
1	P_MERGE	Patch Merge mode
2	P_INTER	Inter predicted Patch mode
3	P_INTRA	Non-predicted Patch mode
4	P_RAW	RAW Point Patch mode
5	P_EOM	EOM Point Patch mode
6-13	P_RESERVED	Reserved modes
14	P_END	Patch termination mode

Table 7-8 Patch mode types for SKIP type atlas tile groups

atgdu_patch_mode	Identifier	Description
0	P_SKIP	Patch Skip mode

7.4.7.2 Patch information data semantics

Let the variable RawSeparateVideoPresentFlag be set as follows:

RawSeparateVideoPresentFlag = vps_raw_separate_video_present_flag[atlasIdx]

where atlasIdx is the index of the current atlas.

7.4.7.3 Patch data unit semantics

pdu_2d_pos_x[p] specifies the x-coordinate of the top-left corner of the patch bounding box for patch p in the current atlas tile group, expressed as a multiple of PatchPackingBlockSize.

The value of pdu_2d_pos_x[p] shall be in the range of 0 to Min($2^{\text{afps\_2d\_pos\_x\_bit\_count\_minus1} + 1}$ − 1, asps_frame_width / PatchPackingBlockSize − 1), inclusive. The number of bits used to represent pdu_2d_pos_x[p] is afps_2d_pos_x_bit_count_minus1 + 1.

pdu_2d_pos_y[p] specifies the y-coordinate of the top-left corner of the patch bounding box for patch p in the current atlas tile group, expressed as a multiple of PatchPackingBlockSize.

The value of pdu_2d_pos_y[p] shall be in the range of 0 to Min($2^{\text{afps\_2d\_pos\_y\_bit\_count\_minus1} + 1}$ − 1, asps_frame_height / PatchPackingBlockSize − 1), inclusive. The number of bits used to represent pdu_2d_pos_y[p] is afps_2d_pos_y_bit_count_minus1 + 1.

pdu_2d_delta_size_x[p], when p is equal to 0, specifies the width value of the patch with index 0 in the current atlas tile group. When p is larger than 0, pdu_2d_delta_size_x[p] specifies the difference of the width values of the patch with index p and the patch with index (p – 1). It is a requirement of bitstream conformance that, when p is equal to 0, pdu_2d_delta_size_x[p] shall be greater than 0.

pdu_2d_delta_size_y[p], when p is equal to 0, specifies the height value of the patch with index 0 in the current atlas tile group. When p is larger than 0, pdu_2d_delta_size_y[p] specifies the difference of the height values of the patch with index p and the patch with index (p – 1). It is a requirement of bitstream conformance that, when p is equal to 0, pdu_2d_delta_size_y[p] shall be greater than 0.

pdu_3d_pos_x[p] specifies the shift to be applied to the reconstructed patch points in patch with index p of the current atlas tile group along the tangent axis. The value of pdu_3d_pos_x[p] shall be in the range of 0 to Min($2^{\text{afps\_3d\_pos\_x\_bit\_count\_minus1} + 1}$, $2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1} + 1}$) − 1, inclusive. The number of bits used to represent pdu_3d_pos_x[p] is afps_3d_pos_x_bit_count_minus1 + 1.

pdu_3d_pos_y[p] specifies the shift to be applied to the reconstructed patch points in patch with index p of the current atlas tile group along the bitangent axis. The value of pdu_3d_pos_y[p] shall be in the range of 0 to Min($2^{\text{afps\_3d\_pos\_y\_bit\_count\_minus1} + 1}$, $2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1} + 1}$) − 1, inclusive. The number of bits used to represent pdu_3d_pos_y[p] is afps_3d_pos_y_bit_count_minus1 + 1.

pdu_3d_pos_min_z[p] specifies the shift to be applied to the reconstructed patch points in patch with index p of the current atlas tile group along the normal axis, Pdu3dPosMinZ[p], as follows:

Pdu3dPosMinZ[p] = pdu_3d_pos_min_z[p] << atgh_pos_min_z_quantizer (7-14)

The value of Pdu3dPosMinZ[p] shall be in the range of 0 to $2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1} + 1}$ – 1, inclusive.

The number of bits used to represent pdu_3d_pos_min_z[p] is equal to (gi_geometry_3d_coordinates_bitdepth_minus1 – atgh_pos_min_z_quantizer + (pdu_projection_id[p] > 5) ? 2 : 1).

pdu_3d_pos_delta_max_z[p], if present, specifies the nominal maximum value of the shift expected to be present in the reconstructed bitdepth patch geometry samples, after conversion to their nominal representation, in patch with index p of the current atlas tile group along the normal axis, Pdu3dPosDeltaMaxZ[p], as follows:

Pdu3dPosDeltaMaxZ[p] = (pdu_3d_pos_delta_max_z[p] == 0 ? 0:
(pdu_3d_pos_delta_max_z[p] << atgh_pos_delta_max_z_quantizer) – 1 (7-15)

If pdu_3d_pos_delta_max_z[p] is not present the value of Pdu3dPosDeltaMaxZ[p] is assumed to be equal to $2^{\text{gi\_geometry\_nominal\_2d\_bitdepth\_minus1}+1} - 1$. When present, the value of Pdu3dPosDeltaMaxZ[p] shall be in the range of 0 to $2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1} + 1} - 1$, inclusive.

The number of bits used to represent pdu_3d_pos_delta_max_z[p] is equal to (gi_geometry_3d_coordinates_bitdepth_minus1 – atgh_pos_delta_max_z_quantizer + (pdu_projection_id [p] > 5) ? 2 : 1).

pdu_projection_id[p] specifies the values of the projection mode and of the index of the normal to the projection plane for the patch with index p of the current atlas tile group. The value of pdu_projection_id[p] shall be in range of 0 to (asps_45degree_projection_present_flag ? 17 : 5) , inclusive.

The variable PduProjectionPlane[p] is derived as follows:

PduProjectionPlane[p] =
asps_45degree_projection_present_flag ? (pdu_projection_id[p] >> 2) : pdu_projection_id[p]

Pdu45degreeProjectionRotationAxis[p] =
asps_45degree_projection_present_flag? pdu_projection_id[p] & 0x03 : 0

The number of bits used to represent pdu_projection_id[p] is (asps_45degree_projection_present_flag ? 5 : 3)

pdu_orientation_index[p] indicates the index to Table 9-1 of the patch orientation index for the patch with index p of the current atlas tile group. The number of bits used to represent pdu_orientation_index[p] is (asps_use_eight_orientations_flag ? 3 : 1)

pdu_lod[p] specifies the LOD scaling factor to be applied to the patch with index p of the current atlas tile group. The reconstructed point 3D positions for patch p in the current atlas tile group are to be scaled by $2^{\text{pdu\_lod}[\,p\,]}$ after their projection from 2D and before applying any further transformations. If pdu_lod[p] is not present, its value shall be inferred to be equal to 0. The number of bits used to represent pdu_lod[p] is afps_lod_bit_count.

7.4.7.4 Skip patch data unit semantics

None

7.4.7.5 Merge patch data unit semantics

mpdu_ref_index[p] specifies the atlas reference frame index, RefIdx , for the current patch. When mpdu_ref_index[p] is not present, it is inferred to be equal to 0.

mpdu_override_2d_params_flag[p] specifies whether the 2d parameters for the current patch are present in the bitstream.

mpdu_2d_pos_x[p] specifies the difference of the x-coordinate of the top-left corner of the patch bounding box of patch with index p in the current atlas tile group and of the x-coordinate of the top-left corner of the patch bounding box of the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that is associated with the reference RefIdx, expressed as a multiple of PatchPackingBlockSize. The value of mpdu_2d_pos_x[p] shall be in the range of (−asps_frame_width / PatchPackingBlockSize + 1) to (asps_frame_width / PatchPackingBlockSize − 1), inclusive. When mpdu_2d_pos_x[p] is not present, it is inferred to be equal to 0.

mpdu_2d_pos_y[p] specifies the difference of the y-coordinate of the top-left corner of the patch bounding box of patch with index p in the current atlas tile group and of the y-coordinate of the top-left corner of the patch bounding box of the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that is associated with the reference RefIdx, expressed as a multiple of PatchPackingBlockSize. The value of mpdu_2d_pos_y[p] shall be in the range of (−asps_frame_height / PatchPackingBlockSize + 1) to (asps_frame_height / PatchPackingBlockSize − 1), inclusive. When mpdu_2d_pos_x[p] is not present, it is inferred to be equal to 0.

mpdu_2d_delta_size_x[p] specifies the difference of the width values of the patch with index p in the current atlas tile group and the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that is associated with the reference RefIdx. When mpdu_2d_delta_size_x[p] is not present, it is inferred to be equal to 0.

mpdu_2d_delta_size_y[p] specifies the difference of the height values of the patch with index p in the current atlas tile group and the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that correponds to the reference RefIdx. When mpdu_2d_delta_size_y[p] is not present, it is inferred to be equal to 0.

mpdu_override_3d_params_flag[p] specifies whether the 3d parameters for the current patch are present in the bitstream. When mpdu_override_3d_params_flag[p] is not present, it is inferred to be equal to 0.

mpdu_3d_pos_x[p] specifies the difference between the shift to be applied to the reconstructed patch points along the tangent axis of patch with index p in the current atlas tile group and of the shift to be applied to the reconstructed patch points along the tangent axis of patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that corresponds to the reference RefIdx. The value of mpdu_3d_pos_x[p] shall be in the range of $(-2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1}+1} + 1)$ to $(2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1}+1} - 1)$, inclusive. When mpdu_3d_pos_x[p] is not present, it is inferred to be equal to 0.

mpdu_3d_pos_y[p] specifies the difference between the shift to be applied to the reconstructed patch points along the bitangent axis of patch with index p in the current atlas tile group and of the shift to be applied to the reconstructed patch points along the bitangent axis of the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that corresponds to RefIdx. The value of mpdu_3d_pos_y[p] shall be in the range of $(-2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1}+1} + 1)$ to $(2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1}+1} - 1)$, inclusive. When mpdu_3d_pos_y[p] is not present, it is inferred to be equal to 0.

mpdu_3d_pos_min_z[p] specifies the difference between the shift to be applied to the reconstructed patch points along the normal axis of patch with index p in the current atlas tile group and of the shift to be applied to the reconstructed patch points along the normal axis of patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that corresponds to RefIdx, Mpdu3dPosMinZ[p], as follows:

Mpdu3dPosMinZ[p] = mpdu_3d_pos_min_z[p] << atgh_pos_min_z_quantizer (7-16)

The value of mpdu_3d_pos_min_z[p] shall be in the range of $(-2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1}+1} + 1)$ to $(2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1}+1} - 1)$, inclusive. When mpdu_3d_ pos_min_z[p] is not present, it is inferred to be equal to 0.

mpdu_3d_pos_delta_max_z[p], if present, specifies the difference between the nominal maximum value of the shift expected to be present in the reconstructed bitdepth patch geometry samples, after conversion to their nominal representation, in patch with index p of the current atlas tile group along the normal axis and of the nominal maximum value of the shift expected to be presented in the reconstructed bitdepth patch geometry samples, after conversion to their nominal representation of the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that corresponds to RefIdx, Mpdu3dPosDeltaMaxZ[p], as follows:

```
Mpdu3dPosDeltaMaxZ[ p ] = (mpdu_3d_pos_delta_max_z[ p ] == 0 ? 0
        : (mpdu_3d_pos_delta_max_z[ p ] << atgh_pos_delta_max_z_quantizer ) – 1          (7-17)
```

If mpdu_3d_pos_delta_max_z[p] is not present the value of Mpdu3dPosDeltaMaxZ[p] is assumed to be equal to $2^{gi\_geometry\_nominal\_2d\_bitdepth\_minus1+1} - 1$. When present, the value of mpdu_3d_pos_delta_max_z[p] shall be in the range of of $(-2^{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1} + 1)$ to $(2^{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1} - 1)$, inclusive.

mpdu_override_plr_flag[patchIdx] specifies whether the point local reconstruction parameters may be overwritten by new parameters that are present in the bistream for the patch.

7.4.7.6 Inter patch data unit semantics

ipdu_ref_index[p] specifies the atlas reference frame index, RefIdx , for the current patch. When ipdu_ref_index[p] is not present, it is inferred to be equal to 0.

ipdu_patch_index[p] specifies the index, PredIdx, of the patch in the atlas tile group with the same address as the current tile group adress in the atlas frame that corresponds to index RefIdx in the current reference atlas frame list.

ipdu_2d_pos_x[p] specifies the difference of the x-coordinate of the top-left corner of the patch bounding box of patch with index p in the current atlas tile group and of the x-coordinate of the top-left corner of the patch bounding box of the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that is associated with the reference RefIdx, expressed as a multiple of PatchPackingBlockSize. The value of ipdu_2d_pos_x[p] shall be in the range of (−asps_frame_width / PatchPackingBlockSize + 1) to (asps_frame_width / PatchPackingBlockSize − 1), inclusive.

ipdu_2d_pos_y[p] specifies the difference of the y-coordinate of the top-left corner of the patch bounding box of patch with index p in the current atlas tile group and of the y-coordinate of the top-left corner of the patch bounding box of the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that is associated with the reference RefIdx, expressed as a multiple of PatchPackingBlockSize. The value of ipdu_2d_pos_y[p] shall be in the range of (−asps_frame_height / PatchPackingBlockSize + 1) to (asps_frame_height / PatchPackingBlockSize − 1), inclusive.

ipdu_2d_delta_size_x[p] specifies the difference of the width values of the patch with index p in the current atlas tile group and the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that is associated with the reference RefIdx.

ipdu_2d_delta_size_y[p] specifies the difference of the height values of the patch with index p in the current atlas tile group and the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that correponds to the reference RefIdx.

ipdu_3d_pos_x[p] specifies the difference between the shift to be applied to the reconstructed patch points along the tangent axis of patch with index p in the current atlas tile group and of the shift to be applied to the reconstructed patch points along the tangent axis of patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that corresponds to the reference RefIdx. The value of ipdu_3d_pos_x[p] shall be in the range of $\left(-2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1}} + 1\right)$ to $\left(2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1}} - 1\right)$, inclusive.

ipdu_3d_pos_y[p] specifies the difference between the shift to be applied to the reconstructed patch points along the bitangent axis of patch with index p in the current atlas tile group and of the shift to be applied to the reconstructed patch points along the bitangent axis of the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that corresponds to RefIdx. The value of ipdu_3d_pos_y[p] shall be in the range of $\left(-2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1}} + 1\right)$ to $\left(2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1}} - 1\right)$, inclusive.

ipdu_3d_pos_min_z[p] specifies the difference between the shift to be applied to the reconstructed patch points along the normal axis of patch with index p in the current atlas tile group and of the shift to be applied to the reconstructed patch points along the normal axis of patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that corresponds to RefIdx, Ipdu3dPosMinZ[p], as follows:

Ipdu3dPosMinZ[p] = ipdu_3d_pos_min_z[p] << atgh_pos_min_z_quantizer (7-18)

The value of ipdu_3d_pos_min_z[p] shall be in the range of $\left(-2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1}} + 1\right)$ to $\left(2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1}} - 1\right)$, inclusive.

ipdu_3d_pos_delta_max_z[p], if present, specifies the difference between the nominal maximum value of the shift expected to be present in the reconstructed bitdepth patch geometry samples, after conversion to their nominal representation, in patch with index p of the current atlas tile group along the normal axis and of the nominal maximum value of the shift expected to be presented in the reconstructed bitdepth patch geometry samples, after conversion to their nominal representation of the patch with index PredIdx in the atlas tile group with the same address as the current tile group in the atlas frame that corresponds to RefIdx, Ipdu3dPosDeltaMaxZ[p], as follows:

```
    Ipdu3dPosDeltaMaxZ[ p ] = (ipdu_3d_pos_delta_max_z[ p ] == 0 ? 0
            : ( ipdu_3d_pos_delta_max_z[ p ] << atgh_pos_delta_max_z_quantizer ) – 1          (7-19)
```

If ipdu_3d_pos_delta_max_z[p] is not present the value of Ipdu3dPosDeltaMaxZ[p] is assumed to be equal to $2^{\text{gi\_geometry\_nominal\_2d\_bitdepth\_minus1+1}} - 1$. When present, the value of ipdu_3d_pos_delta_max_z[p] shall be in the range of of $\left(-2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1}} + 1\right)$ to $\left(2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1+1}} - 1\right)$, inclusive.

7.4.7.7 Raw patch data unit semantics

rpdu_patch_in_raw_video_flag[patchIdx] specifies whether the geometry and attribute data associated with the raw coded patch with index patchIdx in the current atlas tile group are encoded in a separate video compared to those of the intra and inter coded patches. If rpdu_patch_in_raw_video_flag[patchIdx] is equal to 0, the geometry and attribute data associated with the raw coded patch p in the current atlas tile group are encoded in the same video as those of the intra and inter coded patches. If rpdu_patch_in_raw_video_flag[patchIdx] is equal to 1, the geometry and attribute data associated with the raw coded patch p in the current atlas tile group are encoded in a separate video from those of the intra and inter coded patches. If rpdu_patch_in_raw_video_flag[p] is not present, its value shall be inferred to be equal to 0.

rpdu_2d_pos_x[p] specifies the x-coordinate of the top-left corner of the patch bounding box size for raw coded patch p in the current atlas tile group, expressed as a multiple of PatchPackingBlockSize. The value of rpdu_2d_pos_x[p] shall be in the range of 0 to Min($2^{\text{afps\_2d\_pos\_x\_bit\_count\_minus1 + 1}} - 1$, asps_frame_width / PatchPackingBlockSize − 1), inclusive. The number of bits used to represent rpdu_2d_pos_x[p] is afps_2d_pos_x_bit_count_minus1 + 1.

rpdu_2d_pos_y[p] specifies the y-coordinate of the top-left corner of the patch bounding box size for raw coded patch p in the current atlas tile group, expressed as a multiple of PatchPackingBlockSize. The value of rpdu_2d_pos_y[p] shall be in the range of 0 to Min($2^{\text{afps\_2d\_pos\_y\_bit\_count\_minus1} + 1}$ − 1, asps_frame_height / PatchPackingBlockSize − 1), inclusive. The number of bits used to represent rpdu_2d_pos_y[p] is afps_2d_pos_y_bit_count_minus1 + 1.

rpdu_2d_delta_size_x[p], when p is equal to 0, specifies the width value of the raw coded patch with index 0 in the current atlas tile group. When p is larger than 0, rpdu_2d_delta_size_x[p] specifies the difference of the width values of the raw coded patch with index p and the patch with index (p − 1).

rpdu_2d_delta_size_y[p], when p is equal to 0, specifies the height value of the raw coded patch with index 0 in the current atlas tile group. When p is larger than 0, rpdu_2d_delta_size_y[p] specifies the difference of the height values of the raw coded patch with index p and the patch with index (p − 1).

rpdu_3d_pos_x[p] specifies the shift to be applied to the reconstructed raw patch points in the patch with index patchIdx of the current atlas tile group along the tangent axis. The number of bits used to represent rpdu_3d_pos_x[p] is (atgh_raw_3d_pos_axis_bit_count_minus1 + 1).

rpdu_3d_pos_y[p] specifies the shift to be applied to the reconstructed raw patch points in the patch with index patchIdx of the current atlas tile group along the bitangent axis. The number of bits used to represent rpdu_3d_pos_y[p] is (atgh_raw_3d_pos_axis_bit_count_minus1 + 1).

rpdu_3d_pos_z[p] specifies the shift to be applied to the reconstructed raw patch points in the patch with index patchIdx of the current atlas tile group along the normal axis. The number of bits used to represent rpdu_3d_pos_z[p] is (atgh_raw_3d_pos_axis_bit_count_minus1 + 1).

rpdu_points[p] specifies the number of raw points present in the raw coded patch p in the current atlas tile group. It is a requirement of bitstream conformance that the value of rpdu_points[p], when not present, shall not be equal to 0.

7.4.7.8 EOM patch data unit semantics

epdu_2d_pos_x[p] specifies the x-coordinate of the top-left corner of the patch bounding box size for EOM attribute coded patch p in the current atlas tile group, expressed as a multiple of PatchPackingBlockSize. The value of epdu_2d_pos_x[p] shall be in the range of 0 to Min($2^{\text{atgh\_2d\_pos\_x\_bit\_count\_minus1} + 1}$ − 1, sps_frame_width / PatchPackingBlockSize − 1), inclusive. The number of bits used to represent epdu_2d_pos_x[p] is afps_2d_pos_x_bit_count_minus1 + 1.

epdu_2d_pos_y[p] specifies the y-coordinate of the top-left corner of the patch bounding box size for EOM attribute coded patch p in the current atlas tile group, expressed as a multiple of PatchPackingBlockSize. The value of epdu_2d_pos_y[p] shall be in the range of 0 to Min($2^{\text{atgh\_2d\_pos\_y\_bit\_count\_minus1} + 1}$ − 1, sps_frame_height / PatchPackingBlockSize − 1), inclusive. The number of bits used to represent epdu_2d_pos_y[p] is afps_2d_pos_y_bit_count_minus1 + 1.

epdu_2d_delta_size_x[p], when p is equal to 0, specifies the width value of the EOM attribute coded patch with index 0 in the current atlas tile group. When p is larger than 0, epdu_2d_delta_size_x[p] specifies the difference of the width values of the EOM attribute coded patch with index p and the patch with index (p − 1).

epdu_2d_delta_size_y[p], when p is equal to 0, specifies the height value of the EOM attribute coded patch with index 0 in the current atlas tile group. When p is larger than 0, epdu_2d_delta_size_y[p] specifies the difference of the height values of the EOM attribute coded patch with index p and the patch with index (p − 1).

epdu_patch_count_minus1[p] plus 1 indicates the number of subsets of EOM points groups present in the EOM attribute coded patch, with index p, in the current atlas tile group. This number is representative of the number of geometry patches associated with EOM points present in the occupancy map. . epdu_patch_count_minus1[p] shall be in the range of 0 to $2^{\text{afps\_eom\_number\_of\_patch\_bit\_count\_minus1} + 1} - 1$, inclusive.

epdu_points[p] specifies the number of EOM attribute points present in in a subset of the EOM attribute coded patch. epdu_points[p] shall be in the range of 0 to $2^{\text{afps\_eom\_max\_bit\_count\_minus1} + 1} - 1$, inclusive.

7.4.7.9 Point local reconstruction data semantics

The variable BlockCount is derived as follows:

BlockCount= Patch2dSizeX[patchIdx] * Patch2dSizeY[patchIdx]

with the variables Patch2dSizeX[patchIdx] and Patch2dSizeY[patchIdx] derived as specified in clause 8.4.4.1.

plrd_level[mapIdx][patchIdx] equal to 0 indicates that point local reconstruction data information is present in the bitstream for each block of the map mapIndex of the patch patchIndex in the current atlas tile group. plrd_level[mapIdx][patchIdx] equal to 1 indicates that point local reconstruction data information is present in the bitstream for the entire map mapIndex of the patch patchIdx in the current atlas tile group. In this case, all blocks of the map mapIdx shall use the same point local reconstruction data. When plrd_level[mapIdx][patchIdx] is equal to 0, the syntax elements plrd_present_block_flag[mapIdx][patchIdx][i] and plrd_block_mode_minus1[mapIdx][patchIdx][i] are present in the bitstream. When plrd_level[mapIdx][patchIdx] is equal to 1, the syntax elements plrd_present_flag[mapIdx][patchIdx] and plrd_mode_minus1[mapIdx][patchIdx].

plrd_present_block_flag[mapIdx][patchIdx][i] equal to 1 indicates that a point local reconstruction mode information is present in the bitstream for block i of the map mapIdx of the patch patchIdx in the current atlas tile group. plrd_present_block_flag[mapIdx][patchIdx][i] equal to 0 indicates that a point local reconstruction mode information is not present in the bitstream for block i of the map mapIdx of the patch patchIdx in the current atlas tile group.

plrd_block_mode_minus1[mapIdx][patchIdx][i] plus 1 indicates the point local reconstruction mode for block i of the map mapIdx of the patch patchIdx in the current atlas tile group. plrd_block_mode_minus1[mapIdx][patchIdx][i] shall be in the range of 0 to plri_number_of_modes_minus1[mapIdx] plus 1, inclusive.

plrd_present_flag[mapIdx][patchIdx] equal to 1 indicates that a point local reconstruction mode information is present in the bitstream for the map mapIdx of the patch patchIdx in the current atlas tile group. plrd_present_flag[mapIdx][patchIdx] equal to 0 indicates that point local reconstruction mode information is not present in the bitstream for the map mapIdx of the patch patchIdx in the current atlas tile group.

plrd_mode_minus1[mapIdx][patchIdx] plus 1 indicates the point local resconstruction mode for all blocks of the map mapIdx of the patch patchIdx in the current atlas tile group. plrd_mode_minus1[mapIdx][patchIdx] shall be in the range 0 to plri_number_of_modes_minus1[mapIdx] plus 1, inclusive.

7.4.8 Supplemental enhancement information message semantics

Each SEI message consists of the variables specifying the type payloadType and size payloadSize of the SEI message payload. SEI message payloads are specified in Annex E. The derived SEI message payload size payloadSize is specified in bytes and shall be equal to the number of bytes in the SEI message payload.

sm_payload_type_byte is a byte of the payload type of an SEI message.

sm_payload_size_byte is a byte of the payload size of an SEI message.

8 Decoding process

8.1 General decoding process

Input to this process is a V-PCC bitstream or a collection of V-PCC sub-bitstream components.

Output of this process is a set of decoded video streams, corresponding to the occupancy, geometry, and attribute, if available, information of the point cloud sequence, as well as the decoded patch data information, which are needed to perform the 3D reconstruction process specified in clause 9. The decoding process may already include the 3D reconstruction process, as specified in clause 9. For such a case, the output of the decoding process is a sequence of decoded point cloud frames. A point cloud frame consists of a set of points with (x, y, z) coordinates. Optionally, it may also contain one or more set of attributes, as shown in Table 7-2, that shall be associated with each point.

The decoding process is specified such that all decoders that conform to a specified profile, tier, and level will produce numerically identical outputs when invoking the decoding process associated with that profile for a bitstream conforming to that profile, tier, and level. Any decoding process that produces identical outputs to those produced by the process described herein (with the correct output order or output timing, as specified) conforms to the decoding process requirements of this Specification.

NOTE 1 – For the purpose of best-effort decoding, a decoder that conforms to a particular profile at a given tier and level may additionally decode some bitstreams conforming to a different tier, level, or profile without necessarily using a decoding process that produces numerically identical decoded outputs to those produced by the process specified herein (without claiming conformance to the other profile, tier, and level).

The decoding process operates as follows:

1. If the input is a V-PCC bitstream then the V-PCC bitstream is parsed and demultiplexed into several sub-bitstreams, each one corresponding to a different set of information. In particular, the V-PCC bitstream is demultiplexed into occupancy map, geometry, and attributes, if present, video sub-bitstreams, and in an atlas sub-bitstream.
2. The demultiplexed sub-bitstreams in the previous process or the sub-bitstream if the input was a collection of V-PCC sub-bitstream components are then decoded as specified in the subsequent steps.
3. The occupancy map video decoding process as specified in clause 8.5 is invoked with the substream corresponding to the occupancy map information as the input and the decoded occupancy map video frames, as the output. Each decoded occupancy map video frame is represented as oFrame[y][x], where, y and x are the row and column indices, respectively. The variable y is in the range of 0 to asps_frame_height − 1, inclusive and the variable x is in the range of 0 to asps_frame_width − 1, inclusive.
4. The geometry video decoding process, as specified in clause 8.2, is invoked.
5. For every present attribute video sub-bitstreams in the bitstream, the attribute video decoding process as specified in clause 8.3, is invoked .
6. The atlas data group decoding process, as specified in clause 8.4, is invoked with the substream corresponding to the encoded atlas data group information as the input and the decoded atlas data group information as the output.

Optionally:

7. For each point cloud frame, the reconstruction process for a point cloud frame as specified in clause 9 is invoked with the array vps_map_absolute_coding_enabled_flag, all of the decoded occupancy, geometry, and attribute, if present, frames and the decoded atlas tile group

information corresponding to a time instance i as the input and a list of decoded points in the point cloud frame as the output.

8.2 Geometry video decoding process

The geometry video decoding process for the current atlas, with index atlasIdx, is performed as follows:

– If vps_map_count_minus1[atlasIdx] is equal to 0, a video decoding process is invoked using the geometry video bitstream and its associated codec specified by gi_geometry_codec_id[atlasIdx] as the input. Outputs of this process are the decoded and display/output ordered geometry video frames, GeoFrame[mapIdx][orderIdx][compIdx][y][x], and their associated bitdepth, GeoBitdepth[mapIdx][orderIdx], luma width, GeoWidth[mapIdx][orderIdx], and luma height, GeoHeight[mapIdx][orderIdx], where mapIdx corresponds to the map index and is equal to 0, orderIdx is the display order index of the decoded geometry frames, compIdx corresponds to the colour component index and is equal to 0, y is the row index in the decoded frame and is in the range of 0 to GeoHeight[mapIdx][orderIdx] − 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to GeoWidth[mapIdx][orderIdx] − 1, inclusive.

– Otherwise, the following applies:

If vps_multiple_map_streams_present_flag[atlasIdx] is equal to 0, a video decoding process is invoked using the geometry video bitstream and its associated codec specified by gi_geometry_codec_id[atlasIdx] as the input. Outputs of this process are the decoded and display/output ordered intermediate geometry video frames, tempGeoFrame[tempOrderIdx][compIdx][y][x], and their associated bitdepth, tempGeoBitdepth[tempOrderIdx], width, tempGeoWidth[tempOrderIdx], and height, tempGeoHeight[tempOrderIdx], where tempOrderIdx is the display order index of all the decoded geometry frames, compIdx corresponds to the colour component index and is equal to 0, y is the column index in the deoded frame and is in the range of 0 to tempGeoHeight[tempOrderIdx] − 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to tempGeoWidth[tempOrderIdx] − 1, inclusive. The decoded geometry video frames at display/output order orderIdx for each map are then derived as follows:

```
for (i = 0; i <= vps_map_count_minus1[ atlasIdx ]; i++ ){
    mappedIdx = orderIdx * ( vps_map_count_minus1[ atlasIdx ] + 1 ) + i
    GeoBitdepth[ i ][ orderIdx ] = tempGeoBitdepth[ mappedIdx ]
    GeoWidth[ i ][ orderIdx ] = tempGeoWidth[ mappedIdx ]
    GeoHeight[ i ][ orderIdx ] = tempGeoHeight[ mappedIdx ]
    GeoFrame[ i ][ orderIdx ][ 0 ][ y ][ x ]= tempGeoFrame[ mappedIdx ][ 0 ][ y ][ x ]

}
```

Otherwise (if vps_multiple_map_streams_present_flag[atlasIdx] is equal to 1), multiple video decoding processes are invoked, each using a geometry video bitstream with a different vuh_map_index associated with it and the associated codec specified by gi_geometry_codec_id[atlasIdx], as the input. Outputs of this process are the decoded and display/output ordered intermediate geometry video frames, tempGeoFrame[mapIdx][orderIdx][compIdx][y][x], and their associated bitdepth, tempGeoBitdepth[mapIdx][orderIdx], width, tempGeoWidth[mapIdx][orderIdx], and height, tempGeoHeight[mapIdx][orderIdx], where mapIdx corresponds to the map index of each frame and is in the range of 0 to vps_map_count_minus1[atlasIdx] , inclusive, orderIdx is the display order index of the decoded geometry frames for each map, compIdx corresponds to the colour component index and is equal to 0, y is the row index and is in the range of 0 to tempGeoHeight[mapIdx][orderIdx] − 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to tempGeoWidth[mapIdx][orderIdx] − 1, inclusive. The decoded geometry video frames at display/output order orderIdx for each map are then derived as follows:

```
for (i = 0; i <= vps_map_count_minus1[ atlasIdx ]; i++ ){
    GeoBitdepth[ i ][ orderIdx ] = tempGeoBitdepth[ i ][ orderIdx ]
    GeoWidth[ i ][ orderIdx ] = tempGeoWidth[ i ][ orderIdx ]
    GeoHeight[ i ][ orderIdx ] = tempGeoHeight[ i ][ orderIdx ]
    if( vps_map_absolute_coding_enabled_flag[ atlasIdx ][ i ] == 0) {
        j=  VPCCMapPredictorIndex( i )
        for (y = 0; y < GeoHeight[ i ][ orderIdx ] ; y++ ){
            for (x = 0; x < GeoWidth[ i ][ orderIdx ] ; x++ ){
                GeoFrame[ i ][ orderIdx ][ 0 ][ y ][ x ] =
                    Clip3(0, (1 << tempGeoBitdepth[ i ][ orderIdx ]) − 1,
                    tempGeoFrame[ i ][ orderIdx ][ 0 ][ y ][ x ] +
                    tempGeoFrame[ j ][ orderIdx ][ 0 ][ y ][ x ] )
            }
        }
    }
    else {
        for (y = 0; y < GeoHeight[ i ][ orderIdx ] ; y++ ){
            for (x = 0; x < GeoWidth[ i ][ orderIdx ] ; x++ ){
                GeoFrame[ i ][ orderIdx ][ 0 ][ y ][ x ] =
                    tempGeoFrame[ i ][ orderIdx ][ 0 ][ y ][ x ]
    }
}
```

NOTE – Any existing video codec such as AVC or HEVC or any future defined video codec may be used if included in gi_geometry_codec_id.

8.3 Attribute(s) video decoding process

The attribute decoding process for the current atlas atlasIdx is performed as follows:

– If ai_attribute_count[atlasIdx] is equal to 0, no attribute video frames are decoded and no attribute information is associated with the final, reconstructed point cloud.

– Otherwise (if ai_attribute_count[atlasIdx] is not equal to 0), the following applies:

If vps_map_count_minus1 is equal to 0, an ai_attribute_count[atlasIdx] number of video decoding processes are invoked each with a different vuh_attribute_index associated with it and the associated codec specified by ai_attribute_codec_id[atlasIdx][vuh_attribute_index] as the input. Outputs of this process are the decoded and display/output ordered attribute video frames, AttrFrame[attrIdx][mapIdx][orderIdx][compIdx][y][x], and their associated bitdepth, AttrBitdepth[attrIdx][mapIdx][orderIdx], width, AttrWidth[attrIdx][mapIdx][orderIdx], and height, AttrHeight[attrIdx][mapIdx][orderIdx], information, where attrIdx corresponds to the attribute index and is in the range of 0 to ai_attribute_count[atlasIdx] − 1, inclusive, mapIdx corresponds to the map index and is equal to 0, orderIdx is the display order index of the decoded attribute frames, compIdx corresponds to the attribute component index and is in the range of 0 to ai_attribute_dimension_minus1[attrIdx − 1], y is in the range of 0 to AttrHeight[attrIdx][mapIdx][orderIdx] − 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to AttrWidth[attrIdx][mapIdx][orderIdx] − 1, inclusive.

Otherwise (vps_map_count_minus1[atlasIdx] is not equal to 0), the following applies:

If vps_multiple_map_streams_present_flag[atlasIdx] is equal to 0, an ai_attribute_count[atlasIdx] number of video decoding processes are invoked, each with a different vuh_attribute_index associated with it and the associated codec specified by ai_attribute_codec_id[atlasIdx][vuh_attribute_index] as the input. Outputs of this process are the decoded and display/output ordered intermediate attribute video frames, tempAttrFrame[attrIdx][tempOrderIdx][compIdx][y][x], and their associated bitdepth, tempAttrBitdepth[attrIdx][tempOrderIdx], width, tempAttrWidth[attrIdx][tempOrderIdx], and height, tempAttrHeight[attrIdx][tempOrderIdx] information, where attrIdx corresponds to the attribute index and is in the range of 0 to ai_attribute_count[atlasIdx] − 1, inclusive, tempOrderIdx is the display order index of all the decoded attribute frames, compIdx corresponds to the attribute component index and is in the range of 0 to ai_attribute_dimension_minus1[atlasIdx][attrIdx − 1], y is in the range of 0 to tempAttrHeight[attrIdx][tempOrderIdx] − 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to tempAttrWidth[attrIdx][tempOrderIdx] − 1, inclusive. The decoded attribute video frames for attribute with index attrIdx, at display/output order orderIdx for each map are then derived as follows:

attributeDimension = ai_attribute_dimension_minus1[atlasIdx][attrIdx] + 1

```
for (i = 0; i <= vps_map_count_minus1[ atlasIdx ]; i++ ){
    mappedIdx = orderIdx * ( vps_map_count_minus1[ atlasIdx ] + 1 ) + i
    AttrBitdepth[ attrIdx ][ i ][ orderIdx ] = tempAttrBitdepth[ attrIdx ][ mappedIdx ]
    AttrWidth[ attrIdx ][ i ][ orderIdx ] = tempAttrWidth[ attrIdx ][ mappedIdx ]
    AttrHeight[ attrIdx ][ i ][ orderIdx ] = tempAttrHeight[ attrIdx ][ mappedIdx ]
    for (y = 0; y < AttrHeight[ attrIdx ][ i ][ orderIdx ] ; y++ ){
        for (x = 0; x < AttrWidth[ attrIdx ][ i ][ orderIdx ] ; x++ ){
            for(j = 0; j < attributeDimension; j++){
                AttrFrame[ attrIdx ][ i ][ orderIdx ][ j ][ y ][ x ]=
                        tempAttrFrame[ attrIdx ][ mappedIdx ][ j ][ y ][ x ]
            }
    }
}
```

Otherwise (vps_multiple_map_streams_present_flag[atlasIdx] is equal to 1), multiple video decoding processes are invoked, each using an attribute video bitstream with a different vuh_attribute_index and vuh_map_index associated with it and the associated codec specified by ai_attribute_codec_id[atlasIdx][vuh_attribute_index], as the input. Outputs of this process are the decoded and display/output ordered attribute video frames, AttrFrame[attrIdx][mapIdx][orderIdx][compIdx][y][x], and their associated bitdepth, AttrBitdepth[attrIdx][mapIdx][orderIdx], width, AttrWidth[attrIdx][mapIdx][orderIdx], and height, AttrHeight[attrIdx][mapIdx][orderIdx] information, where attrIdx corresponds to the attribute index and is in the range of 0 to ai_attribute_count[atlasIdx] – 1, mapIdx corresponds to the map index of each frame and is in the range of 0 to vps_map_count_minus1[atlasIdx] , inclusive, orderIdx is the display order index of the decoded attribute frames for each map, compIdx corresponds to the attribute component index and is in the range of 0 to ai_attribute_dimension_minus1[atlasIdx][attrIdx − 1], y is in the range of 0 to AttrHeight[attrIdx][mapIdx][orderIdx] – 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to AttrWidth[attrIdx][mapIdx][orderIdx] − 1, inclusive. The decoded attribute video frames at display/output order orderIdx for each map are then derived as follows:

```
for (i = 0; i <= vps_map_count_minus1[ atlasIdx ]; i++ ){
    AttrBitdepth[ attrIdx ][ i ][ orderIdx ] = tempAttrBitdepth[ attrIdx ][ i ][ orderIdx ]
    AttrWidth[ attrIdx ][ i ][ orderIdx ] = tempAttrWidth[ attrIdx ][ i ][ orderIdx ]
    AttrHeight[ attrIdx ][ i ][ orderIdx ] = tempAttrHeight[ attrIdx ][ i ][ orderIdx ]
    if( ai_attribute_map_absolute_coding_enabled_flag[ atlasIdx ][ attrIdx ][ i ] == 0) {
        j= VPCCMapPredictorIndex( i )
        for (y = 0; y < AttrHeight[ attrIdx ][ i ][ orderIdx ] ; y++ ){
            for (x = 0; x < AttrWidth[ attrIdx ][ i ][ orderIdx ] ; x++ ){
                AttrFrame[ attrIdx ][ i ][ orderIdx ][ 0 ][ y ][ x ] =
                    Clip3(0, (1 << tempAttrBitdepth[ attrIdx ][ i ][ orderIdx ]) – 1,
                    tempAttrFrame[ attrIdx ][ i ][ orderIdx ][ 0 ][ y ][ x ] +
                    tempAttrFrame[ attrIdx ][ j ][ orderIdx ][ 0 ][ y ][ x ] )
            }
        }
    }
    else {
        for (y = 0; y < AttrHeight[ attrIdx ][ i ][ orderIdx ] ; y++ ){
            for (x = 0; x < AttrWidth[ attrIdx ][ i ][ orderIdx ] ; x++ ){
                    AttrFrame[ attrIdx ][ i ][ orderIdx ][ 0 ][ y ][ x ] =
                        tempAttrFrame[ attrIdx ][ i ][ orderIdx ][ 0 ][ y ][ x ]
            }
        }
```

The attribute video decoding process also outputs the frame rate, AttrFrameRate[attrIdx], for each output attribute video with index attrIdx. The same frame rate shall be used for all maps.

NOTE 1 – Any existing video codec such as AVC or HEVC or any future defined video codec may be used if included in the attribute codec id.

NOTE 2 – The decoded attribute data are expected to correspond to geometry samples prior to any smoothing or any other process that may be performed after decoding.

8.4 Atlas data group decoding process

8.4.1 General atlas data group decoding process

Input to this process is the bitstream corresponding to the atlas data group unit of the V-PCC bitstream.

Outputs of this process are the decoded atlas frames, as well any associated geometry and/or attribute parameters contained in the atlas data group unit, ordered in increasing order according to their corresponding atlas frame order count number.

The decoding process operates as follows for the current atlas tile group, CurrPatchFrame, being decoded:

1. The decoding of atlas data group units is specified in clause 8.4.2.
2. The processes in clause 8.4.3 specify the following decoding processes using syntax elements in the atlas frame header layer and above:
   – Variables and functions relating to atlas frame order count are derived as specified in clause 8.4.3.1.
   – At the beginning of the decoding process for each atlas frame, the reference atlas frame list construction process specified in clause 8.4.3.2 is invoked for the derivation of the reference atlas frame list, RefAtlasFrmList.
   – The reference atlas frame marking process in clause 8.4.3.3 is invoked, wherein reference atlas frames may be marked as "unused for reference" or "used for long-term reference".
3. The processes in clause 8.4.4 specify the patch decoding processes according to the patch mode as follows:
   – Decoding of intra coded patches is specified in clause 8.4.4.2.
   – Decoding of inter coded patches is specified in clause 8.4.4.5.
   – Decoding of RAW coded patches is specified in clause 8.4.4.6.
   – Decoding of EOM coded patches is specified in clause 8.4.4.7.
4. The process in clause 8.4.5 specifies the creation of the block to patch map after the decoding of all patches within the current atlas tile group.
5. After the current atlas tile group have been decoded, it is marked as "used for short-term reference".

8.4.2 Atlas data group unit decoding process

Inputs to this process are atlas data group units of the current atlas tile group and their associated parameter set units.

Outputs of this process are the parsed syntax structures encapsulated within these atlas data group units.

The decoding process for each atlas data group unit extracts the syntax structure from the atlas data group unit and then parses the syntax structure.

8.4.3 Atlas data group header decoding process

8.4.3.1 Atlas frame order count derivation process

Output of this process is AtlasFrmOrderCntVal, the atlas frame order count of the current atlas tile group, with index AtlasFrmIndex.

Atlas frame order counts are used to identify and order atlas frames, as well as for decoder conformance checking.

Each coded atlas frame is associated with an atlas frame order count variable, denoted as AtlasFrmOrderCntVal.

When there is no available reference atlas frame in the reference atlas frame buffer, the variables prevAtlasFrmOrderCntLsb and prevAtlasFrmOrderCntMsb are derived as follows:

– Let prevAtlasFrm be the previous atlas frame in decoding order.

– The variable prevAtlasFrmOrderCntLsb is set equal to the atlas frame order count LSB value of prevAtlasFrm.

– The variable prevAtlasFrmOrderCntMsb is set equal to AtlasFrmOrderCntMsb of prevAtlasFrm.

The variable AtlasFrmOrderCntMsb of the current atlas tile group is derived as follows:

– If there is no available reference atlas frame in the reference atlas frame buffer, AtlasFrmOrderCntMsb is set equal to 0.

– Otherwise, AtlasFrmOrderCntMsb is derived as follows:

```
if( ( atgh_atlas_frm_order_cnt_lsb <  prevAtlasFrmOrderCntLsb  )  &&
        ( ( prevAtlasFrmOrderCntLsb  − atgh_atlas_frm_order_cnt_lsb)  >=
                              ( MaxAtlasFrmOrderCntLsb / 2 ) ) )
    AtlasFrmOrderCntMsb = prevAtlasFrmOrderCntMsb + MaxAtlasFrmOrderCntLsb(8-1)
else if( ( atgh_atlas_frm_order_cnt_lsb > prevAtlasFrmOrderCntLsb )  &&
        ( ( atgh_atlas_frm_order_cnt_lsb − prevAtlasFrmOrderCntLsb ) >
                              ( MaxAtlasFrmOrderCntLsb / 2 ) ) )
    AtlasFrmOrderCntMsb = prevAtlasFrmOrderCntMsb − MaxAtlasFrmOrderCntLsb
else
    AtlasFrmOrderCntMsb = prevAtlasFrmOrderCntMsb
```

AtlasFrmOrderCntVal is derived as follows:

AtlasFrmOrderCntVal = AtlasFrmOrderCntMsb + atgh_atlas_frm_order_cnt_lsb (8-2)

The value of AtlasFrmOrderCntVal shall be in the range of -2^{31} to $2^{31} - 1$, inclusive. In one CAS, the AtlasFrmOrderCntVal values for any two coded atlas frames shall not be the same.

At any moment during the decoding process, the values of AtlasFrmOrderCntVal & (MaxLtAtlasFrmOrderCntLsb − 1) for any two reference atlas frames in the DPB shall not be the same.

The function AtlasFrmOrderCnt(pFrmX) is specified as follows:

AtlasFrmOrderCnt(pFrmX) = AtlasFrmOrderCntVal of the atlas frame pFrmX (8-3)

The function DiffAtlasFrmOrderCnt(pFrmA, pFrmB) is specified as follows:

DiffAtlasFrmOrderCnt(pFrmA, pFrmB) =
AtlasFrmOrderCnt(pFrmA) – AtlasFrmOrderCnt(pFrmB) (8-4)

The bitstream shall not contain data that result in values of DiffAtlasFrmOrderCnt(pFrmA, pFrmB) used in the decoding process that are not in the range of -2^{15} to $2^{15} - 1$, inclusive.

NOTE 1 – Let X be the current atlas tile group and Y and Z be two other atlas frames in the same CAS, Y and Z are considered to be in the same output order direction from X when both DiffAtlasFrmOrderCnt(X, Y) and DiffAtlasFrmOrderCnt(X, Z) are positive or both are negative.

8.4.3.2 Reference atlas frame list construction process

This process is invoked at the beginning of the decoding process for each atlas frame.

Reference atlas frames are addressed through reference indices. A reference index is an index into a reference atlas frame list. When decoding an I atlas tile group, no reference atlas frame list is used in decoding of the atlas tile group data. When decoding a P_TILE_GRP atlas tile group, a single reference atlas frame list, RefAtlasFrmList, is used in decoding of the atlas frame data.

At the beginning of the decoding process for each atlas frame, the reference atlas frame list RefAtlasFrmList is derived. The reference atlas frame list is used in marking reference atlas frames as specified in clause 8.4.3.3 or in decoding of the atlas tile group data.

The reference atlas frame list RefAtlasFrmList is constructed as follows:

```
for( j = 0, afocBase = AtlasFrmOrderCntVal; j < num_ref_entries[ RlsIdx ]; j++) {          (8-5)
    if( st_ref_atlas_frame_flag[ RlsIdx ][ j ] ) {
        RefAtlasFrmAfocList[ j ] = afocBase − DeltaAfocSt [ RlsIdx ][ j ]
        if( reference pfA exists in the DAFB with AtlasFrmOrderCntVal equal to
                RefAtlasFrmAfocList[ j ] )
            RefAtlasFrmList[ j ] = pfA
        else
            RefAtlasFrmList[ j ] = "no reference atlas frame"
        afocBase = RefAtlasFrmAfocList[ j ]
    } else {
        if(reference pfA exists in the DAFB with
                AtlasFrmOrderCntVal & ( MaxLtAtlasFrmOrderCntLsb − 1 )
                equal to FullAtlasFrmOrderCntLsbLt[ RlsIdx ][ j ] )
            RefAtlasFrmList[ j ] = pfA
        else
            RefAtlasFrmList[ j ] = "no reference atlas frame"
    }
}
```

The first NumRefIdxActive[i] entries in RefPatchFrnList are referred to as the active entries in RefAtlasFrmList and the other entries in RefAtlasFrmList are referred to as the inactive entries in RefAtlasFrmList.

It is a requirement of bitstream conformance that the following constraints apply:

– num_ref_entries[RlsIdx] shall not be less than NumRefIdxActive[i].

- The atlas frame referred to by each active entry in RefAtlasFrmList shall be present in the DAFB.
- The atlas frame referred to by each entry in RefAtlasFrmList shall not be the current atlas frame.
- A short term reference atlas frame entry and a long term reference atlas frame entry in RefAtlasFrmList of an atlas frame shall not refer to the same atlas frame.
- There shall be no long term reference atlas frame entry in RefAtlasFrmList for which the difference between the AtlasFrmOrderCntVal of the current atlas tile group and the AtlasFrmOrderCntVal of the atlas frame referred to by the entry is greater than or equal to 2^{24}.
- Let setOfRefAtlasFrms be the set of unique atlas frames referred to by all entries in RefAtlasFrmList. The number of atlas frames in setOfRefAtlasFrms shall be less than or equal to asps_max_dec_atlas_frame_buffering_minus1.
- The atlas frames referred to by each active entry in RefAtlasFrmList shall have exactly the same tile group partitioning as the current atlas frame.
- The RefAtlasFrmList of all tile groups in the current atlas frame shall contain the same unique atlas frames, without, however, any restrictions in ordering of the reference atlas frames.

8.4.3.3 Reference atlas frame marking process

This process is invoked once per atlas frame, after decoding of an atlas frame header and the decoding process for reference atlas frame list construction for the atlas frame as specified in clause 8.4.3.2, but prior to the decoding of the atlas frame data. This process may result in one or more reference atlas frames in the DAFB being marked as "unused for reference" or "used for long-term reference".

A decoded atlas frame in the DAFB can be marked as "unused for reference", "used for short-term reference" or "used for long-term reference", but only one among these three at any given moment during the operation of the decoding process. Assigning one of these markings to an atlas frame implicitly removes another of these markings when applicable. When an atlas frame is referred to as being marked as "used for reference", this collectively refers to the atlas frame being marked as "used for short-term reference" or "used for long-term reference" (but not both).

Short term reference atlas frames are identified by their AtlasFrmOrderCntVal values. Long term reference atlas frames are identified by the Log2(MaxLtAtlasFrmOrderCntLsb) least significant bits of their AtlasFrmOrderCntVal values.

For all cases, the following applies:

- For each long term reference atlas frame entry in RefAtlasFrmList, when the referred atlas frame is a short term reference atlas frame, the atlas frame is marked as "used for long-term reference".
- Each reference atlas frame in the DAFB that is not referred to by any entry in RefAtlasFrmList is marked as "unused for reference".

8.4.4 Decoding process for patch units

8.4.4.1 Generic decoding process for patch units

Inputs to this process are the current patch index, p, and the current patch mode, atgdu_patch_mode[p].

Outputs of this process are several parameters associated with the current patch with patch index p, including its 2D and corresponding 3D location information, such as parameters Patch2dPosX[p], Patch2dPosY[p], Patch2dSizeX[p], Patch2dSizeY[p], Patch3dPosX[p], Patch3dPosY[p], Patch3dPosMinZ[p], PatchAxisZ[p], PatchProjectionMode[p], PatchOrientationIndex[p], PatchLod[p], and Patch45degreeProjectionRotationAxis[p] .

Patch2dPosX[p] specifies the x-coordinate of the top-left corner of the patch bounding box size for the current patch with patch index p. The value of Patch2dPosX[p] shall be in the range of 0 to Min(($2^{\text{afps\_2d\_pos\_x\_bit\_count\_minus1} + 1}$ – 1) * PatchPackingBlockSize, TileGroupWidth[atgh_address] − 1), inclusive.

Patch2dPosY[p] specifies the y-coordinate of the top-left corner of the patch bounding box size for the current patch with patch index p. The value of Patch2dPosY[p] shall be in the range of 0 to Min(($2^{\text{afps\_2d\_pos\_y\_bit\_count\_minus1} + 1}$ – 1) * PatchPackingBlockSize, TileGroupHeight[atgh_address] − 1), inclusive.

Patch2dSizeX[p] specifies the width of the bounding box of the current patch with patch index p. The value of Patch2dSizeX[p] shall be in the range of 0 to TileGroupWidth[atgh_address], inclusive. It is a requirement of bitstream conformance that Patch2dPosX[p] + Patch2dSizeX[p] <= TileGroupWidth[atgh_address].

Patch2dSizeY[p] specifies the height of the bounding box of the current patch with patch index p. The value of Patch2dSizeY[p] shall be in the range of 0 to TileGroupHeight[atgh_address] , inclusive. It is a requirement of bitstream conformance that Patch2dPosY[p] + Patch2dSizeY[p] <= TileGroupHeight[atgh_address].

Patch3dPosX[p] specifies the shift to be applied to the reconstructed patch points in the current patch with patch index p along the tangent axis. The value of Patch3dPosX[p] shall be in the range of 0 to Min($2^{\text{afps\_3d\_pos\_x\_bit\_count\_minus1} + 1}$, $2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1} + 1}$ – 1), inclusive.

Patch3dPosY[p] specifies the shift to be applied to the reconstructed patch points in the current patch with patch index p along the bitangent axis. The value of Patch3dPosY[p] shall be in the range of 0 to Min($2^{\text{afps\_3d\_pos\_y\_bit\_count\_minus1} + 1}$, $2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1} + 1}$ – 1), inclusive.

Patch3dPosMinZ[p] specifies the shift to be applied to the reconstructed patch points in the current patch with patch index p along the normal axis. The value of Patch3dPosMinZ[p] shall be in the range of 0 to Min($2^{\text{atgh\_3d\_pos\_normal\_axis\_bit\_count\_minus1} + 1}$, $2^{\text{gi\_geometry\_3d\_coordinates\_bitdepth\_minus1} + 1}$ – 1), inclusive.

PatchAxisZ[p] is the index of the normal to the projection plane for the current patch with patch index p. The value of PatchAxisZ[p] shall be in range of 0 to 2, inclusive.

PatchProjectionMode[p] specifies on which plane the current patch with patch index p is to be projected.

PatchOrientationIndex[p] specifies the patch orientation index to Table 9-1 for the current patch with patch index p.

PatchLod[p] specifies the LOD scaling factor to be applied to the current patch with patch index p. The reconstructed 3D positions for the current patch are to be scaled by $2^{\text{pdu\_lod[p]}}$ after their projection from 2D and before applying any further transformations.

Patch45degreeProjectionRotationAxis[p] specifies the patch projection rotation axis to be applied to the patch with index p.

If atgdu_patch_mode[p] is equal to I_INTRA or P_INTRA, then the process for decoding intra coded patches in clause 8.4.4.2 is used, with p as the input to that process and the outputs of that process are used as the output.

If atgdu_patch_mode[p] is equal to P_SKIP, then the process for decoding skip coded patches in clause 8.4.4.3 is used, with p as the input to that process and the outputs of that process are used as the output.

If atgdu_patch_mode[p] is equal to P_MERGE, then the process for decoding merge coded patches in clause 8.4.4.4 is used, with p as the input to that process and the outputs of that process are used as the output.

If atgdu_patch_mode[p] is equal to P_INTER, then the process for decoding inter coded patches in clause 8.4.4.5 is used, with p as the input to that process and the outputs of that process are used as the output.

If atgdu_patch_mode[p] is equal to I_RAW or P_RAW, then the process for decoding RAW coded patches in clause 8.4.4.6 is used, with p as the input to that process and the outputs of that process are used as output.

If atgdu_patch_mode[p] is equal to I_EOM or P_EOM, then the process for decoding EOM coded patches in clause 8.4.4.7 is used, with p as the input to that process and the outputs of that process are used as output.

8.4.4.2 Decoding process for patch units coded in intra mode

Input to this process is the current patch index, p.

The following patch related variables are first assigned given the parsed elements in the patch data unit:

Patch2dPosX[p] = pdu_2d_pos_x[p] * PatchPackingBlockSize (8-6)

Patch2dPosY[p] = pdu_2d_pos_y[p] * PatchPackingBlockSize (8-7)

Patch3dPosX[p] = pdu_3d_pos_x[p] (8-8)

Patch3dPosY[p]= pdu_3d_pos_y[p] (8-9)

Patch3dPosMinZ[p] = Pdu3dPosMinZ[p] (8-10)

PatchAxisZ[p] = PduProjectionPlane[p] % 3 (8-11)

PatchProjectionMode[p] = PduProjectionPlane[p] / 3 (8-12)

PatchOrientationIndex[p] = pdu_orientation_index[p] (8-13)

PatchLod[p] = 1 << pdu_lod[p] (8-14)

Patch45degreeProjectionRotationAxis[p] = Pdu45degreeProjectionRotationAxis[p] (8-15)

Then the variables Patch2dSizeX[p] and Patch2dSizeY[p] are derived as follows:

If p is equal to 0, then:

Patch2dSizeX[p] = pdu_2d_delta_size_x[p] * PatchSizeXQuantizer (8-16)

Patch2dSizeY[p] = pdu_2d_delta_size_y[p] * PatchSizeYQuantizer (8-17)

Otherwise, if (p > 0), then:

Patch2dSizeX[p] = Patch2dSizeX[p – 1] + pdu_2d_delta_size_x[p] * PatchSizeXQuantizer (8-18)

Patch2dSizeY[p] = Patch2dSizeY[p – 1] + pdu_2d_delta_size_y[p] * PatchSizeYQuantizer (8-19)

Finally, the variables PatchAxisX[p] and PatchAxisY[p] are derived as follows:

– If PatchAxisZ[p] is equal to zero, PatchAxisX[p] is assigned a value of 2 and PatchAxisY[p] is assigned a value of 1.

– Otherwise, if PatchAxisZ[p] is equal to 1, PatchAxisX[p] is assigned a value of 2 and PatchAxisY[p] is assigned a value of 0.

– Otherwise (PatchAxisZ[p] is equal to 2) PatchAxisX[p] is assigned a value of 0 and PatchAxisY[p] is assigned a value of 1

All these variables are set as the output of this process.

8.4.4.3 Decoding process for patch units coded in skip prediction mode

Input to this process is the current patch index, p.

First, the atlas frame index, refIdx, of the first atlas frame in the reference atlas frame list is derived based on the process described in clause 8.4.3.2. Then the tile group with the same tile address in the first reference atlas frame as the current tile group is selected to be used as the reference for the current tile group.

If p is equal to 0, then predictorIdx is set to 0.

Then the corresponding patch index, refPatchIdx, in the refIdx atlas frame reference for the current tile group is computed as:

RefPatchIdx = predictorIdx (8-20)

and predictorIdx for the next patch is set to RefPatchIdx + 1.

It is a requirement of bitstream conformance that the patch with index predictorIdx in the refIdx atlas frame reference is not a patch coded in a RAW or EOM prediction mode.

Using the variables refIdx and RefPatchIdx as inputs, the variables refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dPosX, refPatch3dPosY, refPatch3dPosMinZ, refPatchAxisZ, refPatchProjectionMode, refPatchAxisX, refPatchAxisY, refPatchOrientationIndex, and refPatchLod are derived using the process described in clause 8.4.4.5.1.

Then, the associated 2D and 3D patch parameters are derived and set as the output of this process as follows:

Patch2dPosX[p] = refPatch2dPosX (8-21)

Patch2dPosY[p] = refPatch2dPosY (8-22)

Patch2dSizeX[p] = refPatch2dSizeX (8-23)

Patch2dSizeY[p] = refPatch2dSizeY (8-24)

Patch3dPosX[p] = refPatch3dPosX (8-25)

Patch3dPosY[p] = refPatch3dPosY (8-26)

Patch3dPosMinZ[p] = refPatch3dPosMinZ (8-27)

PatchAxisZ[p] = refPatchAxisZ (8-28)

PatchProjectionMode[p] = refPatchProjectionMode (8-29)

PatchAxisX[p] = refPatchAxisX (8-30)

PatchAxisY[p] = refPatchAxisY (8-31)

PatchOrientationIndex[p] = refPatchOrientationIndex (8-32)

PatchLod[p] = refPatchLod (8-33)

Patch45degreeProjectionRotationAxis[p] = refPatch45degreeProjectionRotationAxis (8-34)

8.4.4.4 Decoding process for patch units coded in merge prediction mode

Input to this process is the current patch index, p.

First, the atlas frame index, refIdx, of the mpdu_ref_index[p] + 1 ordered atlas frame in the reference atlas frame list is derived based on the process described in clause 8.4.3.2. Then the tile group with the same tile address in this reference atlas frame as the current tile group is selected to be used as the reference for the current tile group.

If p is equal to 0, then predictorIdx is set to 0.

Then the corresponding patch index, refPatchIdx, in the refIdx atlas frame reference for the current tile group is computed as:

RefPatchIdx = predictorIdx (8-35)

and predictorIdx for the next patch is set to RefPatchIdx + 1.

It is a requirement of bitstream conformance that the patch with index predictorIdx in the refIdx atlas frame reference is not a patch coded in a RAW or EOM prediction mode.

Using the variables refIdx and RefPatchIdx as inputs, the variables refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dPosX, refPatch3dPosY, refPatch3dPosMinZ, refPatchAxisZ, refPatchProjectionMode, refPatchAxisX, refPatchAxisY, refPatchOrientationIndex, and refPatchLod are derived using the process described in clause 8.4.4.5.1.

Then, the associated 2D and 3D patch parameters are derived and set as the output of this process as follows:

Patch2dPosX[p] = refPatch2dPosX + mpdu_2d_pos_x[p] * PatchPackingBlockSize (8-36)

Patch2dPosY[p] = refPatch2dPosY + mpdu_2d_pos_y[p] * PatchPackingBlockSize (8-37)

Patch2dSizeX[p] = refPatch2dSizeX + mpdu_2d_delta_size_x[p] * PatchSizeXQuantizer (8-38)

Patch2dSizeY[p] = refPatch2dSizeY + mpdu_2d_delta_size_y[p] * PatchSizeYQuantizer (8-39)

Patch3dPosX[p] = refPatch3dPosX + mpdu_3d_pos_x[p] (8-40)

Patch3dPosY[p] = refPatch3dPosY + mpdu_3d_pos_y[p] (8-41)

Patch3dPosMinZ[p] = refPatch3dPosMinZ + Mpdu3dPosMinZ[p] (8-42)

PatchAxisZ[p] = refPatchAxisZ (8-43)

PatchProjectionMode[p] = refPatchProjectionMode (8-44)

PatchAxisX[p] = refPatchAxisX (8-45)

PatchAxisY[p] = refPatchAxisY (8-46)

PatchOrientationIndex[p] = refPatchOrientationIndex (8-47)

PatchLod[p] = refPatchLod (8-48)

Patch45degreeProjectionRotationAxis[p] = refPatch45degreeProjectionRotationAxis (8-49)

8.4.4.5 Decoding process for patch units coded in inter prediction mode

Input to this process is the current patch index, p.

First, the atlas frame index, refIdx, of the ipdu_ref_index[p] + 1 ordered atlas frame in the reference atlas frame list is derived based on the process described in clause 8.4.3.2. Then the tile group with the same tile address in this reference atlas frame as the current tile group is selected to be used as the reference for the current tile group.

If p is equal to 0, then predictorIdx is set to 0.

Then the corresponding patch index, refPatchIdx, in that atlas frame reference for the current tile group is computed as:

RefPatchIdx = predictorIdx + ipdu_patch_index[p] (8-50)

and predictorIdx for the next patch is set to RefPatchIdx + 1.

It is a requirement of bitstream conformance that the patch with index predictorIdx in the refIdx atlas frame reference is not a patch coded in a RAW or EOM prediction mode.

Using the variables refIdx and RefPatchIdx as inputs, the variables refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dPosX, refPatch3dPosY, refPatch3dPosMinZ, refPatchAxisZ, refPatchProjectionMode, refPatchAxisX, refPatchAxisY, refPatchOrientationIndex, and refPatchLod are derived using the process described in clause 8.4.4.5.1.

Then, the associated 2D and 3D patch parameters are derived and set as the output of this process as follows:

Patch2dPosX[p] = refPatch2dPosX + ipdu_2d_pos_x[p] * PatchPackingBlockSize (8-51)

Patch2dPosY[p] = refPatch2dPosY + ipdu_2d_pos_y[p] * PatchPackingBlockSize (8-52)

Patch2dSizeX[p] = refPatch2dSizeX + ipdu_2d_delta_size_x[p] * PatchSizeXQuantizer (8-53)

Patch2dSizeY[p] = refPatch2dSizeY + ipdu_2d_delta_size_y[p] * PatchSizeYQuantizer (8-54)

Patch3dPosX[p] = refPatch3dPosX + ipdu_3d_pos_x[p] (8-55)

Patch3dPosY[p] = refPatch3dPosY + ipdu_3d_pos_y[p] (8-56)

Patch3dPosMinZ[p] = refPatch3dPosMinZ + Ipdu3dPosMinZ[p] (8-57)

PatchAxisZ[p] = refPatchAxisZ (8-58)

PatchProjectionMode[p] = refPatchProjectionMode (8-59)

PatchAxisX[p] = refPatchAxisX (8-60)

PatchAxisY[p] = refPatchAxisY (8-61)

PatchOrientationIndex[p] = refPatchOrientationIndex (8-62)

PatchLod[p] = refPatchLod (8-63)

Patch45degreeProjectionRotationAxis[p] = refPatch45degreeProjectionRotationAxis (8-64)

8.4.4.5.1 Derivation of the inter reference patch parameters

Inputs to this process are the atlas frame reference index refIdx and patch reference index, refPatchIdx.

Outputs to this process are the variables refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dPosX, refPatch3dPosY, refPatch3dPosMinZ, refPatchAxisZ, refPatchProjectionMode, refPatchAxisX, refPatchAxisY, refPatchOrientationIndex, refPatchLod, and refPatch45degreeProjectionRotationAxis.

The following variables are derived, based on the patch parameters for each tile group that correspond to the same atlas tile group address in the patch reference associated with the index refIdx and set as the output of this process as follows:

refPatch2dPosX = Patch2dPosX[refPatchIdx] (8-65)

refPatch2dPosY = Patch2dPosY[RefPatchIdx] (8-66)

refPatch2dSizeX = Patch2dSizeX[RefPatchIdx] (8-67)

refPatch2dSizeY = Patch2dSizeY[RefPatchIdx] (8-68)

refPatch3dPosX = Patch3dPosX[RefPatchIdx] (8-69)

refPatch3dPosY = Patch3dPosY[RefPatchIdx] (8-70)

refPatch3dPosMinZ = Patch3dPosMinZ[RefPatchIdx] (8-71)

refPatchAxisZ = PatchAxisZ[RefPatchIdx] (8-72)

refPatchProjectionMode = PatchProjectionMode[RefPatchIdx] (8-73)

refPatchAxisX = PatchAxisX[RefPatchIdx] (8-74)

refPatchAxisY = PatchAxisY[RefPatchIdx] (8-75)

refPatchOrientationIndex = PatchOrientationIndex[RefPatchIdx] (8-76)

refPatchLod = PatchLod[RefPatchIdx] (8-77)

refPatch45degreeProjectionRotationAxis =
Patch45degreeProjectionRotationAxis[RefPatchIdx] (8-48)

8.4.4.6 Decoding process for patch units coded in raw mode

Input to this process is the current patch index, p.

Outputs to this process are the variables Patch3dPosX, Patch3PosY, Patch3dPosZ.

The following patch related variables are first assigned given the parsed elements in the patch data unit:

Patch2dPosX[p] = rpdu_2d_pos_x[p] * PatchPackingBlockSize (8-78)

Patch2dPosY[p] = rpdu_2d_pos_y[p] * PatchPackingBlockSize (8-79)

Then the variables Patch2dSizeX[p] and Patch2dSizeY[p] are derived as follows:

If ((p>0) && ((atgdu_patch_mode[p-1] == I_RAW) ||((atgdu_patch_mode[p-1] == P_RAW)))

Patch2dSizeX[p] = Patch2dSizeX[p – 1] + rpdu_2d_delta_size_x[p] * PatchSizeXQuantizer (8-80)

Patch2dSizeY[p] = Patch2dSizeY[p – 1] + rpdu_2d_delta_size_y[p] * PatchSizeYQuantizer (8-81)

Else:

Patch2dSizeX[p] = rpdu_2d_delta_size_x[p] * PatchSizeXQuantizer (8-82)

Patch2dSizeY[p] = rpdu_2d_delta_size_y[p] * PatchSizeYQuantizer (8-83)

PatchRawPoints[p] = rpdu_points[p] (8-84)

Then, the 3D patch variables Patch3dPosX[p], Patch3dPosY[p] and Patch3dPosZ[p] are derived and set as output of this process as follows:

if afps_raw_3d_pos_bit_count_explicit_mode_flag is equal to 1,

Patch3dPosX[p] = rpdu_3d_raw_pos_x[p] (8-85)

Patch3dPosY[p] = rpdu_3d_raw_pos_y[p] (8-86)

Patch3dPosZ[p] = rpdu_3d_raw_pos_z[p] (8-87)

Otherwise,

raw3dLevel = 1 << (gi_geometry_nominal_2d_bitdepth_minus1 + 1) (8-88)

Patch3dPosX[p] = rpdu_3d_raw_pos_x[p] * raw3dLevel (8-89)

Patch3dPosY[p] = rpdu_3d_raw_pos_y[p] * raw3dLevel (8-90)

Patch3dPosZ[p] = rpdu_3d_raw_pos_z[p] * raw3dLevel (8-91)

8.4.4.7 Decoding process for patch units coded in EOM mode

Input to this process is the current OEM patch index, p.

Outputs to this process are the variables EomPatch2dPosX, EomPatch2dPosY, EomPatch2dSizeX, EomPatch2dSizeY, and the vector EomPointOffset of size epdu_patch_count_minus1 + 1, containing the relative address of each first EOM points associated to each geometry patch.

The following patch related variables are first assigned given the parsed elements in the patch data unit:

EomPatch2dPosX[p] = epdu_2d_pos_x[p] * PatchPackingBlockSize (8-92)

EomPatch2dPosY[p] = epdu_2d_pos_y[p] * PatchPackingBlockSize (8-93)

Then the variables EomPatch2dSizeX[p] and EomPatch2dSizeY[p] are derived as follows:

If ((p>0) && ((atgdu_patch_mode[p-1] == I_EOM ||((atgdu_patch_mode[p-1] == P_EOM)))

EomPatch2dSizeX[p] =EomPatch2dSizeX[p – 1] +
epdu_2d_delta_size_x[p] * PatchSizeXQuantizer (8-94)

```
    EomPatch2dSizeY[ p ] = EomPatch2dSizeY[ p − 1] +
            epdu_2d_delta_size_y[ p ] * PatchSizeYQuantizer                    (8-95)
```

Else:

```
    EomPatch2dSizeX[ p ] = epdu_2d_delta_size_x[ p ] * PatchSizeXQuantizer     (8-96)

    EomPatch2dSizeY[ p ] = epdu_2d_delta_size_y[ p ] * PatchSizeYQuantizer     (8-97)
```

Then the vector EomPointOffset is derived as follows:

```
EomPointOffset[0] = 0

for( geoPatchIndex = 1 geoPatchIndex< epdu_patch_count_minus1[ p ]+1 ; geoPatchIndex ++ )

    EomPointOffset[ geoPatchIndex ] = EomPointOffset[ geoPatchIndex − 1 ]
                                 +  epdu_points[ geoPatchIndex − 1]
```

8.4.5 Decoding process of the block to patch map

Inputs to this process are:

– the current atlas tile group address, tgAddress

– the total number of patches in the current atlas tile group, PfduTotalNumberOfPatches

– the Patch2dPosX, Patch2dPosY, Patch2dSizeX, Patch2dSizeY arrays

– the PatchPackingBlockSize, asps_frame_height, and asps_frame_width elements of the current active ASPS

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

```
    BlockToPatchMapWidth = Ceil( TileGroupWidth[ tgAddress ] ÷ PatchPackingBlockSize )

    BlockToPatchMapHeight = Ceil( TileGroupHeight[ tgAddress ] ÷ PatchPackingBlockSize )
```

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

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

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

            BlockToPatchMap[ y ][ x ] = −1
```

Then the BlockToPatchMap array is updated as follows:

```
    for( patchIdx = 0; patchIdx < PfduTotalNumberOfPatches; patchIdx++ ) {
        mode = atgdu_patch_mode[ patchIdx ]
        if ((( atgh_type == I_TILE_GRP ) && ( mode != I_RAW ) && ( mode != I_EOM )) ||
            (( atgh_type == P_TILE_GRP ) && (mode != P_RAW) && ( mode != P_EOM ) ||
            ( atgh_type == SKIP_TILE_GRP ) ||( !rpdu_patch_in_raw_video_flag[ patchIdx ]) {
                xOrg = Patch2dPosX[ patchIdx ] / PatchPackingBlockSize
                yOrg = Patch2dPosY[ patchIdx ] / PatchPackingBlockSize
                for( y = 0; y < Patch2dSizeX [ patchIdx ]/ PatchPackingBlockSize ; y++)
                    for( x = 0; x < Patch2dSizeY[ patchIdx ] / PatchPackingBlockSize ;x++) {
                        if(( asps_patch_precedence_order_flag == 0 )  ||
                            ( BlockToPatchMap[ yOrg + y ][ xOrg + x ] == −1))
                                BlockToPatchMap[ yOrg + y ][ xOrg + x ] = patchIdx
                }
            }
        }
    }
```

8.4.6 Conversion of tile group level information to Atlas level information

In this clause, the patches from different tile groups are combined into a single list. Let the total number of such patches in the current Atlas, for the current point cloud frame be AtlasTotalNumberOfPatches. The array AtlasPatchMode[p] stores the patch mode for the patch with index p, p = 0..(AtlasTotalNumberOfPatches − 1), in the combined list of patches. The BlockToPatch arrays for the tile groups are also combined and each entry points to a patch from the combined list of patches. Patch2dPosX and Patch2dPosY are with respect to the origin of the current atlas instead of the current tile group. The arrays Patch2dSizeX, Patch2dSizeY, Patch3dPosX, Patch3dPosY, and Patch3dPosZ corresponding to the tile groups are also combined to form arrays at the atlas level.

8.5 Occupancy map video decoding process

A video decoding process is invoked using the occupancy video bitstream and its associated codec, specified by occ_occupancy_codec_id, as inputs. Outputs of this process are the frame rate of the occupancy map video, OccFrameRate, the decoded and display/output ordered occupancy video frames, OccFrame[orderIdx][compIdx][y][x], and their associated bitdepth, OccBitdepth[orderIdx], width, OccWidth[orderIdx], and height, OccHeight [orderIdx], where orderIdx is the display order index of the decoded occupancy map video frames, compIdx corresponds to the colour component index and is equal to 0, y is the row index in the decoded frame and is in the range of 0 to OccHeight[orderIdx] − 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to OccWidth[orderIdx] − 1, inclusive.

If asps_enhanced_occupancy_map_for_depth_flag is equal to 0, each decoded occupancy video frame with display order index of orderIdx, compIdx of 0, is thresholded as follows:

The variable lossyOccMapThreshold is derived as follows:

```
    lossyOccMapThreshold[ orderIdx ] =
        oi_lossy_occupancy_map_compression_threshold << ( OccBitdepth[ orderIdx ] – 8 ).

    for( y=0; y <  OccHeight[ orderIdx ]; y++ )
        for( x=0; x <  OccWidth[ orderIdx ]; x++ )
            if( OccFrame[ orderIdx ][ compIdx ][ y ][ x ] <=  lossyOccMapThreshold[ orderIdx ] )
                OccFrame[ orderIdx ][ compIdx ][ y ][ x ] = 0
            else
                OccFrame[ orderIdx ][ compIdx ][ y ][ x ] = 1
```

The decoded (possibly thresholded) occupancy map video frames shall be upscaled to match the nominal resolution of the decoded video frames, namely asps_frame_width pixels wide and asps_frame_height pixels high. The occupancy map upscaling process is defined in TBD.

NOTE – Any existing video codec such as AVC or HEVC or any future defined video codec may be used if included in occ_occupancy_codec_id.

9 Reconstruction process

9.1 General

Input to this process is a set of decoded video sub-bitstreams, corresponding to the occupancy, geometry, and if available, attribute information of the point cloud sequence, as well as the decoded atlas information. The process also takes as inputs the syntax elements and upper-case variables from clause 7 and 8. The decoded video sub-bitstreams and the decoded atlas information is referred to as a decoded point cloud frame.

The output of this process is a sequence of reconstructed, possibly smoothed, point cloud frames. A reconstructed point cloud frame consists of a set of points with (x, y, z) coordinates. Optionally, it may also contain one or more sets of attributes, as shown in Table 7-2, that shall be associated with each point.

For the sequence of decoded point cloud frames, the following ordered steps apply

– Chroma format, resolution, and frame rate conversions are performed, if required, as specified in clause 9.2.

– Point cloud reconstruction is performed as specified in clause 9.3.

– Point cloud smoothing is performed as specified in clause 9.6.

9.2 Chroma format, resolution, and frame rate conversion

Inputs to this process are:

– the decoded geometry frames, GeoFrame, and their associated bitdepth, GeoBitdepth, width, GeoWidth, height, GeoHeight, and frame rate GeoFrameRate

– the decoded attribute frames, AttrFrame, and their associated bitdepth, AttrBitdepth width, AttrWidth, height, AttrHeight, and frame rate AttrFrameRate

– the decoded occupancy frames, OccFrame, and their associated bitdepth, OccBitdepth, width, OccWidth, height, OccHeight, and frame rate OccFrameRate

– the BlockToPatchMap arrays and their associated width, BlockToPatchMapWidth, height, BlockToPatchMapHeight, and frame rate BlockToPatchMapRate

– the decoded atlas tile groups and their associated patch information, Patch2dPosX, Patch2dPosY, Patch2dSizeX, Patch2dSizeY, Patch3dPosX, Patch3dPosY, Patch3dPosMinZ, PatchAxisZ, PatchOrientationIndex, PatchLod, and PatchProjectionMode

– The atlas width, asps_frame_width, height, asps_frame_height, frame rate, vpcc_frame_rate, and geometry bitdepth, $2^{\text{gi\_geometry\_2d\_log2bitdepth}}$

Outputs of this process are:

– the upscaled to the atlas width, asps_frame_width, atlas height, asps_frame_height, nominal frame rate, vpcc_frame_rate, and geometry bitdepth, gi_geometry_2d_nominal_bitdepth_minus1, geometry frames, GeoFrameNR.

– the upscaled to the atlas width, asps_frame_width, height, asps_frame_height, and frame rate, vpcc_frame_rate, and attribute bitdepth, ai_attribute_nominal_2d_bitdepth_minus1, attribute frames, AttrFrameNR, occupancy frames, OccFrameNR, and block to patch map arrays, BlockToPatchMap.

– the upscaled to the atlas width, asps_frame_width, height, asps_frame_height, and frame rate, vpcc_frame_rate, atlas frames and their associated patch information, Patch2dPosX, Patch2dPosY, Patch2dSizeX, Patch2dSizeY, Patch3dPosX, Patch3dPosY, Patch3dPosMinZ, PatchAxisZ, PatchOrientationIndex, PatchLod, and PatchProjectionMod.

The upscaling process is outside the scope of this specification. At the end of this process, all such information should be in sync and time-aligned.

9.3 Point cloud reconstruction

For the current point cloud frame with index OrderIdx in display order, and the current atlas with index AtlasIdx, the inputs to this process are:

– GeoFrameNR[mapIdx][OrderIdx][0][y][x], the 0th component of the decoded geometry frame with index OrderIdx at nominal resolution (after upscaling if necessary), denoted by GFrame[mapIdx][y][x],

– AttrFrame[attrIdx][mapIdx][OrdIdx][compIdx][y][x], the decoded attribute frame with index OrdIdx at nominal resolution (after upscaling, if necessary), denoted by AFrame[attrIdx][mapIdx][compIdx][y][x], and

– OccFrame[OrderIdx][0][y][x], the decoded occupancy map frame with index OrderIdx at nominal resolution (after upscaling, if necessary), denoted by OFrame[y][x].

For the current point cloud frame with index OrderIdx in display order and atlas index AtlasIdx, the outputs of this process are:

– PointCnt, the number of points in the reconstructed point cloud frame.

– RecPcGeo[n][k], a container holding a list of coordinates of the points in the reconstructed point cloud frame, where n = 0 .. PointCnt − 1, k = 0 .. 2, and

– RecPcAttr[n][attrIdx][compIdx], a container holding a list of attributes corresponding to the points in the reconstructed point cloud frame, where n = 0 .. PointCnt − 1, attrIdx = 0 .. . ai_attribute_count − 1, and compIdx = 0 .. ai_attribute_dimension_minus1[attrIdx].

For the current point cloud frame with index OrderIdx in display order and the current atlas with index AtlasIdx, the following applies:

– A variable PointCnt is initialized to 0. Arrays RecPcGeo, RecPcAttr, and AttrPresent are initially empty.

– For p = 0 .. AtlasTotalNumberOfPatches − 1, the following applies:

  – If AtlasPatchMode[p] is I_RAW or P_RAW, clause 9.5 is invoked with GFrame, AFrame, and patch index p as inputs. The outputs are updated PointCnt, RecPcGeo, and RecPcAttr.

  – Otherwise, if AtlasPatchMode[p] is I_INTRA, P_SKIP, P_MERGE, P_INTRA, or P_INTER, the following applies:

    – For u = 0..Patch2dSizeX[p] − 1, v = 0..Patch2dSizeY[p] − 1, the following applies:

      – The conversion of patch coordinates to atlas coordinates as specified in clause 9.4.5 is invoked with patch coordinates (u, v), and patch index p, as inputs and atlas coordinates (x, y) as output.

      – The variables subBlkX and subBlkY are calculated as follows:

```
subBlkY = Y / PatchPackingBlockSize
subBlkX = X / PatchPackingBlockSize
```

   - If OFrame[y][x] is not equal to 0 and AtlasBlockToPatchMap[subBlkY][subBlkX] is equal to p, for mapIdx = 0 .. asps_map_count_minus1, clause 9.4.3 is invoked.
   - If asps_enhanced_occupancy_map_for_depth_flag is equal to 1, clause 9.4.4 is invoked with GFrame, OFrame, AFrame, patch index p, PatchProjectionMode[p], and PatchPackingBlockSize as inputs and the outputs are updated PointCnt, RecPcGeo, and RecPcAttr.

- The points for which no attributes are found in the sub-bitstream, are assigned attributes as follows:
  - For n = 0 .. (PointCnt − 1), if AttrPresent[n] is not equal to 1, the following applies:
    - A point from RecPcGeo that is nearest to RecPcGeo[n] and which satisfies the following conditions is determined. Let the index of the point in RecPcGeo be m.

      AttrPresent[m] = 1, and
      Points corresponding to indices m and n belong to the same patch.

    - The attributes corresponding to the point from RecPcGeo with index m are copied as follows:

```
for( a = 0; a < ai_attribute_count[ atlasIdx ]; a++ )
    for( c = 0; c < ai_attribute_dimension_minus1[ atlasIdx ][ a ]; c++ )
        RecPcAttr[ n ][ a ][ c ] =RecPcAttr[ m ][ a ][ c ]
```

9.4 Reconstruction of points for non-raw intra and inter coded patches

9.4.1 Reconstruction of points for non-raw intra and inter coded patches when asps_pixel_deinterleaving_flag is equal to 1

The neighbours of the pixel, which belong to the same patch, are identified and their minimum and maximum geometry values are derived as follows:

For u = 0..Patch2dSizeU[p] − 1, v = 0..Patch2dSizeV[p] − 1, the following applies:

- The conversion of patch coordinates to frame canvas coordinates as specified in clause 9.4.5 is invoked with patch coordinates (u, v) and patch index p as inputs and frame canvas coordinates (x, y) as output.
- The variables subBlkX and subBlkY are calculated as follows:

```
subBlkY = y / PatchPackingBlockSize
subBlkX = x / PatchPackingBlockSize
```

- If OFrame[y][x] is not equal to 0 and BlockToPatchMap[subBlkY][subBlkX] is equal to p, the following ordered steps apply

  1. The variables validNeighbourCount and depthNeighbours is initialized as follows:

```
validNeighbourCount = 0
for( k = 0; k < 4; k++ )
    depthNeighbours[ k ] = 0
```

2. The neighbours of the pixel, which belong to the same patch, are identified and their minimum and maximum geometry values are derived as follows:

```
minDepth = RecPcGeo[ PointCnt ][ PatchAxisZ[ p ] ]
maxDepth = RecPcGeo[ PointCnt ][ PatchAxisZ[ p ] ]

subBlkY = y / PatchPackingBlockSize
subBlkX = (x – 1) / PatchPackingBlockSize

if( ( OFrame[ x – 1 ][ y ] != 0 ) && (BlockToPatchMap[ subBlkY ][ subBlkX ] == p) &&
    ( x > 0 ) ) {
    if( PatchProjectionMode[ p ] = = 0)
        depthNeighbours[ 0 ] = GFrame[ 0 ][ x – 1 ][ y ] +
            Patch3dShiftMinNormalAxis[ p ]
    else
        depthNeighbours[ 0 ] = Patch3dShiftMinNormalAxis[ p ] – GFrame[ 0 ][ x – 1 ][ y ]

    validNeighbourCount++
    minDepth = Min( minDepth, depthNeighbour[ 0 ] )
    maxDepth = Max( maxDepth, depthNeighbour[ 0 ] )
}

subBlkY = y / PatchPackingBlockSize
subBlkX = (x + 1) / PatchPackingBlockSize

if( ( OFrame[ x + 1 ][ y ] != 0 ) && (BlockToPatchMap[ subBlkY ][ subBlkX ] == p) &&
    ( x < ( asps_frame_width – 1 ) ) {
    if( PatchProjectionMode[ p ] = = 0)
        depthNeighbours[ 1 ] = GFrame[ 0 ][ x + 1 ][ y ] + Patch3dShiftMinNormalAxis[ p ]
    else
        depthNeighbours[ 1 ] = Patch3dShiftMinNormalAxis[ p ] – GFrame[ 0 ][ x + 1 ][ y ]

    validNeighbourCount++
    minDepth = Min( minDepth, depthNeighbour[ 1 ] )
    maxDepth = Max( maxDepth, depthNeighbour[ 1 ] )
}

subBlkY = (y – 1) / PatchPackingBlockSize
subBlkX = x / PatchPackingBlockSize

if( ( OFrame[ x ][ y– 1 ] != 0 ) && (BlockToPatchMap[ subBlkY ][ subBlkX ] == p) &&
    ( y > 0) {
    if( PatchProjectionMode[ p ] = = 0)
        depthNeighbours[ 2 ] = GFrame[ 0 ][ x ][ y – 1 ] + Patch3dShiftMinNormalAxis[ p ]
    else
        depthNeighbours[ 2 ] = Patch3dShiftMinNormalAxis[ p ] – GFrame[ 0 ][ x ][ y – 1 ]

    validNeighbourCount++
    minDepth = Min( minDepth, depthNeighbour[ 2 ] )
    maxDepth = Max( maxDepth, depthNeighbour[ 2 ] )
}

subBlkY = (y + 1) / PatchPackingBlockSize
subBlkX = x / PatchPackingBlockSize
```

```
if( ( OFrame[ x ][ y+ 1 ] != 0 ) && (BlockToPatchMap[ subBlkY ][ subBlkX ] == p) &&
      ( y < ( asps_frame_height – 1 ) {
      if( PatchProjectionMode[ p ] = = 0)
            depthNeighbours[ 3 ] = GFrame[ 0 ][ x ][ y + 1 ] + Patch3dShiftMinNormalAxis[ p ]
      else
            depthNeighbours[ 3 ] = Patch3dShiftMinNormalAxis[ p ] – GFrame[ 0 ][ x ][ y + 1 ]

      validNeighbourCount++
      minDepth = Min( minDepth, depthNeighbour[ 3 ] )
      maxDepth = Max( maxDepth, depthNeighbour[ 3 ] )
}
```

The following applies:

– If(x + y) % 2) is equal to 1 the following applies:

```
depth1 = RecPcGeo[ PointCnt − 1 ][ PatchAxisZ[ p ] ]
for( k = 0; k < 3; k++ )
      pos[ k ] = RecPcGeo[ PointCnt − 1 ][ k ]
if( PatchProjectionMode[ p ] = = 0 )
      depth0 = Clip3(depth1 – asps_surface_thickness_minus1 – 1, depth1, MinDepth )
else
      depth0 = Clip3( depth1, depth1 + asp_surface_thickness_minus1 + 1, MaxDepth )
```

  – Clause 9.4.8 is invoked with (x, y), p, asps_map_count_minus1, depth0, and GFrame as inputs. The output is a variable, isDuplicate.

  – If isDuplicate is equal to 0, the following applies:

```
pos[ PatchAxisZ[ p ] ] = depth0
for( k = 0; k < 3; k++ )
      RecPcGeo[ PointCnt ][ k ] = pos[ k ]
AttrPresent[ PointCnt ] = 0
PointCnt += 1
```

– Otherwise ((x + y) % 2) is equal to 0) the following applies:

```
depth0 = RecPcGeo[ PointCnt − 1 ][ PatchAxisZ[ p ] ]
for( k = 0; k < 3; k++ )
      pos[ k ] = RecPcGeo[ PointCnt − 1 ][ k ]
averageDepth = 0
for( k = 0; k < 4; k++ )
      averageDepth += DepthNeighbours[ k ]
averageDepth = averageDepth / ValidNeighbourCount

if( PatchProjectionMode[ p ] = = 0 )
      depth1 = Clip3( depth0,  depth0 + asps_surface_thickness_minus1 + 1, averageDepth)
else
      depth1 = Clip3( depth0 − asps_surface_thickness_minus1 − 1, depth0, averageDepth)
```

  – Clause 9.4.8 is invoked with (x, y), p, asps_map_count_minus1, depth1, and GFrame as inputs. The output is a variable, isDuplicate.

  – If isDuplicate is equal to 0, the following applies:

```
pos[ PatchAxisZ[ p ] ] = depth1
for( k = 0; k < 3; k++ )
    RecPcGeo[ PointCnt ][ k ] = pos[ k ]
AttrPresent[ PointCnt ] = 0
PointCnt += 1
```

– The points between depth0 and depth1 are filled as follows:

```
maxDepth = Max( depth0, depth1 )
minDepth = Min( depth0, depth1 )
```

 – For s = 1..(maxDepth – minDepth – 1), clause 9.4.7 is invoked with (x, y), p, asps_map_count_minus1, (minDepth + s), and GFrame as inputs. The output is a variable, isDuplicate.

 – If isDuplicate is equal to 0, the following applies:

```
pos[ PatchAxisZ[ p ] ] = depth1
for( k = 0; k < 3; k++ )
    RecPcGeo[ PointCnt ][ k ] = pos[ k ]
AttrPresent[ PointCnt ] = 0
PointCnt += 1
```

9.4.2 Reconstruction of points for non-raw intra and inter coded patches when plr_point_local_reconstruction_map_enabled_flag is equal to 1

Inputs to this process are:

– PointCnt, the number of points in the reconstructed point cloud frame.

– GFrame[mapIdx][y][x], where mapIdx = 0..asps_map_count_minus1, y = 0 .. asps_frame_height – 1, and x = 0 .. asps_frame_width – 1.

– OFrame[y][x], where y = 0 .. asps_frame_height – 1 and x = 0 .. asps_frame_width – 1.

– AFrame[aIdx][mapIdx][cIdx][y][x], where mapIdx = 0 .. asps_map_count_minus1, y = 0 .. asps_frame_height – 1, x = 0 .. asps_frame_width – 1, aIdx = 0 .. ai_attribute_count – 1, and cIdx = 0 .. ai_attribute_dimension_minus1[aIdx].

– the index of the patch, p.

– PatchProjectionMode[p], the plane on which the current patch with patch index p is projected

– PatchPackingBlockSize, the size of the block used for the horizontal and vertical placement of the patches within the atlas.

– asps_point_local_reconstruction_information(mapIdx) decoded syntax elements of clause 7.3.6.2, where mapIdx = 0 .. asps_map_count_minus1.

– point_local_reconstruction_data(p) decoded syntax elements of clause 7.3.7.8.

Outputs to this process are:

– updated RecPcGeo[n][k], a container holding a list of coordinates of the points in the reconstructed point cloud frame, where n = 0 .. PointCnt – 1 and k = 0 .. 2.

– updated RecPcAttr[n][aIdx][cIdx], a container holding a list of attributes corresponding to the points in the reconstructed point cloud frame, where n = 0 .. PointCnt − 1, aIdx = 0 .. ai_attribute_count − 1, and cIdx = 0 .. ai_attribute_dimension_minus1[aIdx].

When asps_point_local_reconstruction_enabled_flag equal 1, the process described in this clause is used to reconstruct the point cloud. The point local reconstruction process is applied to each occupied pixel of the geometry frame. As an output, the point cloud is updated with new points.

The following processes apply for each map of index mapIdx with mapIdx = 0..asps_map_count_minus1.

Let log2BlockSize = log2(PatchPackingBlockSize).

When plr_point_local_reconstruction_map_enabled_flag[mapIdx] is equal to 1, the variable plrMode[bIdxY][bIdxX] is derived in a first step for each block of coordinates (bIdxX, bIdxY) for the map mapIdx of the patch p, as follows:

```
– If( plrd_level[ mapIdx ][ p ] = = 0 )
        1. The variable blockCnt is initialized as follows:
           blockCnt = 0
        2. For bIdxX = 0..(Patch2dSizeX[ p ] − 1)>> log2BlockSize   and,
            bIdxY = 0..(Patch2dSizeY[ p ] − 1)>> log2BlockSize, the following applies:
              plrMode[ bIdxY ][ bIdxX ] = plrd_block_mode_minus1[ mapIdx ][ p ][ blockCnt++ ] + 1
  Otherwise ( if( plrd_level[ mapIdx ][ p ] = = 1 ) )
        For bIdxX = 0..(Patch2dSizeX[ p ] − 1) >> log2BlockSize,
        bIdxY = 0..(Patch2dSizeY[ p ] − 1) >> log2BlockSize, the following applies:
              plrMode[ bIdxY ][ bIdxX ] = plrd_mode_minus1[ mapIdx ][ p ] + 1
```

Then, in a second step, the variables plrInterpolateFlag[bIdxY][bIdxX], plrFillingFlag[bIdxY][bIdxX], plrMinimumDepth[bIdxY][bIdxX] and plrNeighbour[bIdxY][bIdxX], are derived as follows:

```
For bIdxX = 0..(Patch2dSizeX[ p ] − 1) >> log2BlockSize,
    bIdxY = 0..(Patch2dSizeY[ p ] − 1) >> log2BlockSize, the following applies::
    plrInterpolateFlag[ bIdxY ][ bIdxX ] = plri_interpolate_flag[ mapIdx ][ plrMode[ bIdxY ][ bIdxX ] ]
    plrFillingFlag[ bIdxY ][ bIdxX ] = plri_filling_flag[ mapIdx ][ plrMode[ bIdxY ][ bIdxX ] ]
    plrMinimumDepth[ bIdxY ][ bIdxX ] = plri_minimum_depth[ mapIdx ][ plrMode[ bIdxY ][ bIdxX ] ]
    plrNeighbour[ bIdxY ][ bIdxX ] = plri_neighbour_minus1[ mapIdx ][ plrMode[ bIdxY ][ bIdxX ] ] + 1
```

The third step consists of the point local reconstruction process which applies as follows:

For u = 0..Patch2dSizeX[p] − 1, v = 0..Patch2dSizeY[p] − 1, the following applies:

– The conversion of patch coordinates to atlas coordinates as specified in clause 9.4.5 is invoked with patch coordinates (u, v) and patch index p as inputs and atlas coordinates (x, y) as output.

– If OFrame[y][x] is not equal to 0, the following ordered steps apply:

1. The variable deltaDepth is initialized as follows:

   deltaDepth = 0

2. If plrInterpolateFlag[y >> log2BlockSize][x >> log2BlockSize] is equal to 1, then

   – If PatchProjectionMode[p] is equal to 0 then,

     – Find the neighbouring pixel of coordinates (x_n, y_n) in the geometry frame belonging to the current patch with the largest geometry value such that GFrame[mapIdx][y_n][x_n] – GFrame[mapIdx][y][x] is equal or smaller than asps_surface_thickness_minus1 plus 1.

       The neighbourhood is defined as a square window of size 2 * plrNeighbour[y >> log2BlockSize][x >> log2BlockSize] + 1 centered around (x, y).

     – Set deltaDepth as follows:

       deltaDepth = GFrame[mapIdx][y_n][x_n] – GFrame[mapIdx][y][x] – 1

   – If PatchProjectionMode[p] is equal to 1 then,

     – Find the neighbouring pixel of coordinates (x_n, y_n) in the geometry frame belonging to the current patch with the largest geometry value such that GFrame[mapIdx][y_n][x_n] – GFrame[mapIx][y][x] is greater than or equal to asps_surface_thickness_minus1 + 1.

       The neighbourhood is defined as a square window of size 2 * plrNeighbour[y >> log2BlockSize][x >> log2BlockSize] + 1 centered around (x,y).

     – Set deltaDepth as follows:

       deltaDepth = GFrame[mapIdx][y_n][x_n] – GFrame[mapIdx][y][x] – 1

3. Update deltaDepth as follows:

   – If PatchProjectionMode[p] is equal to 0 then,

     deltaDepth = Max(deltaDepth,
     plrMinimumDepth[y >> log2BlockSize][x >> log2BlockSize])

   – If PatchProjectionMode[p] is equal to 1 then,

     deltaDepth = Min(deltaDepth,
     – (plrMinimumDepth[y >> log2BlockSize] [x>>log2BlockSize]))

4. If deltaDepth is different from 0, then

     – Add a point to reconstructed geometry point cloud with same tangent and bi-tangent coordinates as the current point, and normal coordinate equal to normal coordinate of the current point plus deltaDepth.

       – The variable plrDepth is set equal to GFrame[mapIdx][y][x] + deltaDepth

– The variable orgDepth is set equal to GFrame[mapIdx][y][x]

– Clause 9.4.8 is invoked with (x, y), p, asps_map_count_minus1, plrDepth, and GFrame as inputs. The output is a variable, isDuplicate.

– If isDuplicate is equal to 0, the following applies:

```
pos[ PatchAxisZ[ p ] ] = plrDepth
for( k = 0; k < 3; k++ )
   RecPcGeo[ PointCnt ][ k ] = pos[ k ]
AttrPresent[ PointCnt ] = 0
PointCnt += 1
```

– If plrFillingFlag[y >> log2BlockSize][x >> log2BlockSize] is equal to 1, then the points between orgDepth and plrDepth are filled as follows:

```
maxDepth = Max( orgDepth, plrDepth )
minDepth = Min( orgDepth, plrDepth )
```

– For s = 1..(maxDepth – minDepth – 1), clause 9.4.7 is invoked with (x, y), p, asps_map_count_minus1, (minDepth + s), and GFrame as inputs. The output is a variable, isDuplicate.

– If isDuplicate is equal to 0, the following applies:

```
pos[ PatchAxisZ[ p ] ] = plrDepth
for( k = 0; k < 3; k++ )
      RecPcGeo[ PointCnt ][ k ] = pos[ k ]
AttrPresent[ PointCnt ] = 0
PointCnt += 1
```

9.4.3 Reconstruction of points for non-raw intra and inter coded patches

The inputs to this process are:

– PointCnt, the number of points in the reconstructed point cloud frame,

– p, the patch index,

– mapIdx, the index of the map,

– (u, v), the coordinates within the patch,

– GFrame[mapIdx][y][x], where y = 0..asps_frame_height – 1, x = 0.. asps_frame_width – 1,

– OFrame[y][x], where y = 0..asps_frame_height – 1, x = 0..asps_frame_width – 1,

– AFrame[aIdx][mapIdx][cIdx][y][x], where aIdx = 0 .. ai_attribute_count – 1, cIdx = 0 .. ai_attribute_dimension_minus1[aIdx], y = 0 .. asps_frame_height – 1, and x = 0 .. asps_frame_width – 1,

– PatchProjectionMode[p], the plane on which the current patch with patch index p is projected,

– PatchPackingBlockSize, the size of the block used for the horizontal and vertical placement of the patches within the atlas.

The following applies:

– The conversion of patch coordinates to atlas coordinates as specified in clause 9.4.5 is invoked with patch coordinates (u, v), and patch index p, as inputs and atlas coordinates (x, y) as output.

– The 3D coordinates corresponding to the point with patch coordinates (u, v) are derived by invoking clause 9.4.6 with (u, v), (x, y), patch index p, mapIdx, and GFrame as inputs. The output is a vector, pos of size 3, containing the (x, y, z) coordinates of the point.

– If mapIdx is equal to zero, a variable, isDuplicate, is set to 0.

– Otherwise (mapIdx is greater than 0), clause 9.4.7 is invoked with atlas coordinates (x, y), patch index p, map index (mapIdx – 1), and GFrame as inputs. The output is the variable, isDuplicate.

– If isDuplicate is equal to 0 or mapIdx is equal to 0. the following applies:

  – The vector, pos, is copied over to a new entry in RecPcGeo.

```
for( n=0; n < 3; n++ )
    RecPcGeo[ PointCnt ][ n ] = pos[ n ]
```

  – For aIdx = 0 .. ai_attribute_count[AtlasIdx]– 1, attributes are assigned to the newly added point as follows:

```
attrDim = ai_attribute_dimension_minus1[ AtlasIdx ][ aIdx ] + 1
for( cIdx = 0; cIdx  <  attrDim; cIdx+ + )
    RecPcAttr[ PointCnt ][ aIdx ][ cIdx ] = AFrame[ aIdx ][ mapIdx ][ cIdx ][ y ][ x ]
AttrPresent[ PointCnt ] = 1
```

  – If Patch45degreeProjectionRotationAxis[p] is greater than 0, clause 9.4.7 is invoked.

  – The variable PointCnt is incremented by 1.

– If asps_pixel_deinterleaving_flag is equal to 1, clause 9.4.1 is invoked.

– Otherwise if plr_point_local_reconstruction_map_enabled_flag[mapIdx] is equal to 1, clause 9.4.2 is invoked.

9.4.4 Reconstruction of points when asps_enhanced_occupancy_map_for_depth_flag is 1

Inputs to this process are:

– GFrame[mapIdx][y][x], where mapIdx = 0..asps_map_count_minus1, y = 0..asps_frame_height – 1, and x = 0.. asps_frame_width – 1,

– OFrame[y][x], where y = 0..asps_frame_height – 1 and x = 0..asps_frame_width – 1,

– AFrame[aIdx][mapIdx][cIdx][y][x], where mapIdx = 0..asps_map_count_minus1, y = 0 .. asps_frame_height – 1, x = 0 .. asps_frame_width – 1, aIdx = 0 .. ai_attribute_count – 1, and cIdx = 0 .. ai_attribute_dimension_minus1[aIdx].

– p, the patch index,

– PatchProjectionMode[p], the plane on which the current patch with patch index p is projected, and

– PatchPackingBlockSize, the size of the block used for the horizontal and vertical placement of the patches within the atlas

Outputs to this process are:

- updated PointCnt,
- updated RecPcGeo[n][k], a container holding a list of coordinates of the points in the reconstructed point cloud frame, where n = 0 .. PointCnt − 1 and k = 0 .. 2, and
- updated RecPcAttr[n][aIdx][cIdx], a container holding a list of attributes corresponding to the points in the reconstructed point cloud frame, where n = 0 .. PointCnt − 1, aIdx = 0 .. ai_attribute_count − 1, and cIdx = 0 .. ai_attribute_dimension_minus1[aIdx].

This clause is invoked when asps_enhanced_occupancy_map_for_depth_flag is equal to 1.

The variable eomPointCnt is initialized to 0.

For u = 0..Patch2dSizeX[p] − 1, v = 0..Patch2dSizeY[p] − 1, the following applies:

- The conversion of patch coordinates to atlas coordinates as specified in clause 9.4.5 is invoked with patch coordinates (u, v), and patch index p, as inputs and atlas coordinates (x, y) as output.
- If OFrame[y][x] is not equal to 0 and (D1 − D0) is greater than 1, the following ordered steps apply:
  1. A variable D0 is set equal to GFrame[0][y][x].
  2. A variable D1 is derived as follows:
     - If asps_map_count_minus1 is equal to 0, a variable D1 is set equal to D0 + asps_enhanced_occupancy_map_fix_bit_count_minus1 + 1.
     - Otherwise (asps_map_count_minus1 is greater than 0) D1 is set equal to GFrame[1][y][x]
  3. If (D1 − D0) is greater than 1, the following applies:
     - A variable eomCode is derived as follows:

       eomCode = (1 << (D1 − D0 − 1)) − OFrame[y][x]
     - For b = 0 .. (D1 − D0 − 2), if (((eomCode >> b) & 0x01) = = 1), the following applies:
       - Clause 9.4.6 is invoked with the patch coordinates (u, v), the atlas coordinates (x, y), the patch index p, and depth (D0 + b + 1) and the output is an array pos of dimension 3.
       - The vector, pos, is copied over to a new entry in RecPcGeo.

         ```
         for( n=0; n < 3; n++ )
           RecPcGeo[ PointCnt ][ n ] = pos[ n ]
         ```
       - For aIdx = 0 .. ai_attribute_count[atlasIdx]− 1, attributes are assigned to the newly added point as follows:

         ```
           eomPos = EomPointOffset[p] + eomPointCnt
           inBlockPos = eomPos % ( PatchPackingBlockSize * PatchPackingBlockSize )
           nbBlock = eomPos / ( PatchPackingBlockSize * PatchPackingBlockSize )
         ```

```
        xBlock = nbBlock % ( EomPatch2dSizeX [ p ]/ PatchPackingBlockSize )
        yBlock = nbBlock / ( EomPatch2dSizeY [ p ]/ PatchPackingBlockSize )
        xx = xBlock * PatchPackingBlockSize + inBlockPos % PatchPackingBlockSize
        + EomPatch2dPosX[ p ]
        yy = yBlock * PatchPackingBlockSize + inBlockPos / PatchPackingBlockSize
        + EomPatch2dPosY[ p ]
       attrDim = ai_attribute_dimension_minus1[ atlasIdx ][ aIdx ] + 1
       for( cIdx = 0; cIdx  <  attrDim; cIdx+ + )
           RecPcAttr[ PointCnt ][ aIdx ][ cIdx ] = AFrame[ aIdx ][ 0 ][ cIdx ][ yy ][ xx ]
           AttrPresent[ PointCnt ] = 1
   –   eomPointCnt += 1
   –   PointCnt += 1
```

9.4.5 Conversion from patch coordinates to atlas coordinates

Inputs to this process are:

- Patch coordinates (u, v)
- frmIdx, the frame index
- p, the patch index

Outputs of this process are the atlas coordinates (x, y).

The variable orientationIdx is set to PatchOrientationIndex[p].

The atlas coordinates (x, y) are obtained as follows:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \text{Rotation(orientationIdx)} * \begin{bmatrix} u \\ v \end{bmatrix} + \text{Offset(orientationIdx)} + \begin{bmatrix} \text{Patch2dPosX[p]} \\ \text{Patch2dPosY[p]} \end{bmatrix}$$

where the outputs of functions Rotation(x) and Offset(x) are matrices as specified in Table 9-1.

Table 9-1 Flexible Patch Orientation

x	Identifier	Rotation(x)	Offset(x)
0	FPO_NULL	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$
1	FPO_SWAP	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$

2	FPO_ROT90	$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} \text{Patch2dSizeY[p]} - 1 \\ 0 \end{bmatrix}$
3	FPO_ROT180	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} \text{Patch2dSizeX[p]} - 1 \\ \text{Patch2dSizeY[p]} - 1 \end{bmatrix}$
4	FPO_ROT270	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \text{Patch2dSizeX[p]} - 1 \end{bmatrix}$
5	FPO_MIRROR	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \text{Patch2dSizeX[p]} - 1 \\ 0 \end{bmatrix}$
6	FPO_MROT90	$\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} \text{Patch2dSizeY[p]} - 1 \\ \text{Patch2dSizeX[p]} - 1 \end{bmatrix}$
7	FPO_MROT180	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \text{Patch2dSizeY[p]} - 1 \end{bmatrix}$

9.4.6 Reconstruct 3D point position

The inputs to this process are:

– patch coordinates (u, v)

– atlas coordinates (x, y)

– patch index p, and

– mapIdx, the map index

– depth value depthValue.

The output of this process is a vector, pos, of dimension 3.

The following steps apply:

– pos[PatchAxisX[p]] = (u + Patch3dPosX[p]) * PatchLod[p]

– pos[PatchAxisY[p]] = (v + Patch3dPosY[p]) * PatchLod[p]

– The coordinate corresponding to the normal axis is derived as follows:

  – If PatchProjectionMode[p] is equal to 0,

    pos[PatchAxisZ[p]] = (depthValue + Patch3dPosMinZ[p]) * PatchLod[p]

  – Otherwise (PatchProjectionMode[p] is equal to 1)

```
pos[ PatchAxisZ[ p ] ] = 0
tmpDepth = ( Patch3dPosMinZ[ p ] – depthValue )
if( tmpDepth > 0 )
    pos[PatchAxisZ[ p ] ] = tmpDepth * PatchLod[ p ]
```

9.4.7 Projection planes at 45 degrees

The 3D coordinates of the reconstruction point are adjusted based on the value of the variable Patch45degreeProjectionRotationAxis[p] as follows:

– The variable rotBias is initialized as follows:

```
rotBias = ( 1 << (gi_geometry_3d_coordinates_bitdepth_minus1 +1 ) – 1 ).
```

– The variables rotAxis1 and rotAxis2 are derived as follows.

```
rotAxis1 = ( 4 − Patch45degreeProjectionRotationAxis[ p ] ) % 3
rotAxis2 = ( 3 − Patch45degreeProjectionRotationAxis[ p ] ).
```

– The 3D coordinates of the reconstructed point are modified as follows:

```
temp1 = RecPcGeo[ PointCnt ][ rotAxis1 ]
temp2 = RecPcGeo[ PointCnt ][ rotAxis2 ]
RecPcGeo[ PointCnt ][ rotAxis1 ] = (temp1 – temp2 + rotBias ) / 2
RecPcGeo[ PointCnt ][ rotAxis2 ] = (temp1 + temp2 – rotBias ) / 2
```

9.4.8 Duplicate point check

Inputs to this process are:

– atlas coordinates (xC, yC)

– p, a patch index

– curIdx, a map index

– gFrame[mapIdx][y][x], decoded geometry frames at nominal resolution, where mapIdx = 0 .. asps_map_count_minus1, y = 0 .. asps_frame_height – 1, and x = 0 .. asps_frame_width – 1.

Outputs of this process is variable, isDuplicate, indicating whether the point with depth d is a duplicate of a point from the set of points with map indices in the range of 0 .. curIdx – 1.

The variable isDuplicate is assigned a value of 0.

If asps_remove_duplicate_point_enabled_flag is equal to 1, the following applies:

```
for( i=0; ( i < curIdx && !isDuplicate ) ; i++ )
    if( gFrame[ i ][ yC ][ xC ] = = gFrame[ curIdx ][ yC ][ xC ] )
        isDuplicate = 1
```

9.5 Reconstruction of point cloud from raw coded patches

Inputs to this process are:

– PointCnt, the number of points in the reconstructed point cloud frame.

– GFrame[0][y][x], where y = 0 .. asps_frame_height – 1, x = 0 .. asps_frame_width – 1.

– AFrame[aIdx][0][cIdx][y][x], where y = 0 .. asps_frame_height – 1, x = 0 .. asps_frame_width – 1, aIdx = 0 .. ai_attribute_count – 1, and cIdx = 0 .. ai_attribute_dimension_minus1[aIdx].

– p, the index of the patch

Output of this process are:

– updated RecPcGeo[n][k], a container holding a list of coordinates of the points in the reconstructed point cloud frame, where n = 0 .. PointCnt − 1 and k = 0 .. 2.

– updated RecPcAttr[n][aIdx][cIdx], a container holding a list of attributes corresponding to the points in the reconstructed point cloud frame, where n = 0 .. PointCnt − 1, aIdx = 0 .. . ai_attribute_count − 1, and cIdx = 0 .. ai_attribute_dimension_minus1[aIdx].

The geometry of the raw coded points is retrieved from the raw coded patch in the geometry frame and added to RecPcGeo, as specified in the ordered steps below:

1. The variables numRawPoints, xRawPos, yRawPos, zRawPos are initialized as follows:

   numRawPoints = 0

   xRawPos = PatchRawPoints[p]

   yRawPos = 2 * PatchRawPoints[p]

   zRawPos = 3 * PatchRawPoints[p]

2. For x = Patch2dPosX[p]..Patch2dPosX[p] + Patch2dSizeX[p] − 1,

   y = Patch2dPosY[p]..Patch2dPosY[p] + Patch2dSizeY[p] − 1, the following applies:

   – if(numRawPoints < xRawPos)

     RecPcGeo[PointCnt][0] = GFrame[0][y][x] + Patch3dPosX[p]

   – else if (numRawPoints < yRawPos)

     RecPcGeo[PointCnt][1] = GFrame[0][y][x] + Patch3dPosY[p]

   – else if (numRawPoints < zRawPos)

     RecPcGeo[PointCnt][2] = GFrame[0][y][x] + Patch3dPosZ[p]

   – The variables numRawPoints and PointCnt are incremented by 1

While the 3-tuple (x, y, z) coordinates of the raw coded points are stored in a single patch in the first component of the geometry frame, the attribute information for a raw coded point is stored across the different attribute dimensions in the attribute frame at the same location. Consequently, the area of the raw coded patch in the first component of the geometry frame is attribute dimension times the area of the raw coded patch in each of the dimension of the attribute frame. The attribute information of the raw coded points is retrieved from the different attribute dimensions of the attribute frame and added to RecPcAttr as specified below:

– The variable pointCntAttr is set equal to PointCnt − numRawPoints

– The variable numRawPoints is set equal to 0

– For j = 0.. vps_atlas_count_minus1

```
– For x = Patch2dPosX[ p ]..Patch2dPosX[ p ] + Patch2dSizeX[ p ] − 1,
  y = Patch2dPosY[ p ]..Patch2dPosY[ p ] + Patch2dSizeY[ p ] − 1, the following applies:
    – if( numRawPoints < PatchRawPoints[ p ] )
        1. For aIdx = 0..ai_attribute_count − 1
            – attrDim = ai_attribute_dimension_minus1[ j ][ aIdx ] + 1
            – for( cIdx = 0; cIdx < attrDim; cIdx+ + )
              RecPcAttr[ PointCnt ][ aIdx ][ cIdx ] = AFrame[ aIdx ][ 0 ][ cIdx ][ y ][ x ]
        2. The variable AttrPresent[ pointCntAttr ] is set to 1
        3. The variables numRawPoints and PointCntAttr are incremented by 1
```

9.6 Smoothing process

9.6.1 Geometry smoothing process

This process is invoked when gfp_geometry_smoothing_enabled_flag is equal to 1.

Inputs to this process are:

- the occupancy map frame, oFrame
- the patch index information from the active geometry_patch_parameter_set
- the geometry smoothing control parameters, GeometrySmoothingGridSize and gfp_geometry_smoothing_threshold
- recPCgSmIn, the container for holding the list of points in the reconstructed point cloud geometry without any attributes prior to the smoothing process.
- PointCnt, the number of points in the current reconstructed point cloud frame

Output to this process is:

- recPCgSmOut, the container for holding the list of points in the reconstructed point cloud geometry without any attributes after the geometry smoothing process.

First sub clause 9.6.3 is invoked to derive:

- number of center grid geometry positions, numCells.
- an array containing center grid of geometry positions, geomCenterGrid[i][k], for $0 \leq i <$ numCells - 1, $0 \leq k < 2$.
- a boolean array flag, cellDoSmoothing[x][y][z], for all x, y, z = 0 to numCells - 1, inclusive.
- an array, cellCount[x][y] [z], for all x, y, z = 0 to numCells - 1, inclusive

The points at the patch boundary are selected as specified in sub clause 9.6.4. The list of the selected points is included into an array recPCgOccEdges as follows:

```
for ( i = 0; i < PointCnt ; i++ ) {
    if (recPcgInBoundaryPointType[ i ] = = 1){
        addPointToPointCloud(recPCgOccEdges, recPCgSmIn[ i ][ 3 ])
```

```
            }
        }
```

For each recPcgPos[3] position in the recPgOccEdges, sub clause 9.6.5 is applied to derive:

– a 2x2x2 grid neighbourhood corresponding to the current position.
– a boolean value otherClusterPtCnt
– the top-left corner of 2x2x2 grid, s[i], i = 0 to 2, inclusive.

When the otherClusterPointCount variable is equal to 1, the centroid valCentroid[x][y][z][k] $0 \leq x, y, z < 1, 0 \leq k < 2$ and the number of points in the 2x2x2 grid are derived, as follows:

The variable cntPt is initialized to 0 and the array

```
for( k = 0; k < 3; k++ ) {
    for( dz = 0; dz < 2; dz++ ) {
        for( dy = 0; dy < 2; dy++ ) {
            for( dx = 0; dx < 2; dx++ ) {
                Ix = s[ 0 ] + dx
                Iy = s[ 1 ] + dy
                Iz = s[ 2 ] + dz
                valCentroid[ dx ][ dy ][ dz ][ k ] = cellCnt[ Ix ][ Iy ][ Iz ] > 0 ?
                            geomCenterGrid[ Ix ][ Iy ][ Iz ][ k ] : recPcgPos[ k ]
                cntPt += cellCnt[ Ix ][ Iy ][ Iz ]
            }
        }
    }
}
```

cntPt = cntPt / (GridSize * GridSize * GridSize * 8)

An 8-tap trilinear filter as described in 9.6.6 is applied to centroids, valCentroid[x][y][z][k], over the 2×2×2 neighborhood to derive the filtered point, filtOut[k], k = 0 to 2, inclusive.

Finally, depending on the distortion criteria shown below, filtOut[k] is added to the recPCgSmOut.

```
    if( otherClusterPointCount ) {
        dist2 = (recPcgPos[ 0 ] – filtOut[ 0 ])$^2$
                + (recPcgPos[ 1 ] – filtOut[ 1 ])$^2$
                + (recPcgPos[ 2 ] – filtOut[ 2 ])$^2$
        if(dist2 * cntPt >= Max( gfp_geometry_smoothing_threshold, cntPt ) * 2) {
            pointCloudToPointCloud(recPCgSmOut, centroid)
        }
    }
```

9.6.2 Attribute smoothing process

This process is invoked when afp_attribute_smoothing_enabled_flag[aIdx] is equal to 1, where aIdx is the attribute index.

Inputs to this process are:

- an occupancy map corresponding to the current point cloud frame at nominal resolution, oFrame,
- an attribute index, aIdx,
- number of components, numComps, corresponding to the attribute index aIdx,
- an attribute smoothing control parameter set, afp_attribute_smoothing_grid_size[aIdx], afp_attribute_smoothing_threshold[aIdx], afp_attribute_smoothing_threshold_local_entropy[aIdx], afp_attribute_smoothing_threshold_variation[aIdx], and afp_attribute_smoothing_threshold_difference[aIdx]
- an array containing reconstructed attribute values for attribute index aIdx, RecPcAttrSmIn[aIdx][i][j], $0 \leq i <$ PointCnt, $0 \leq j <$ numComps,
- an array containing reconstructed (possibly smoothed) reconstucted positions, RecPcGeomSmOut[i][j], $0 \leq i <$ PointCnt, $0 \leq j \leq 2$, and
- patch index information corresponding to each point in RecPcGeomSmOut

Output of this process is:

- an array containing reconstructed smoothed attribute values for attribute index aIdx, RecPcAttrSmOut[aIdx][i][j], 0 ≤ i < PointCnt, 0 ≤ j < numComps.

The variable GridSize is set to AttributeSmoothingGridSize[aIdx].

Number of cells, numCells, and the arrays of attrCenterGrid[i] and cellDoSmoothing[x][y][z], for i = 0 to numCells -1, inclusive, are derived as described in 9.6.3.2:

for i = 0 to (PointCnt – 1), inclusive, the following applies:

- otherClusterPtCnt is set equal to 0.

To determine the points at the patch boundary, clause 9.6.4 is invoked to produce an array recPCBoundary[idx], 0 ≤ idx < PointCnt, which identifies whether each point from RecPcGeomSmOut is a boundary point.

for i = 0 to (PointCnt – 1), inclusive, the following applies:

- If recPCBoundary[i] is equal to 1 the following steps are performed:
- Variables pointGeom[j], j in the range of 0 to 2, inclusive, and pointAttr[k], k in the range of 0 to (numComps – 1), inclusive, are defined as follows:

```
for( j = 0; j < 3 ; j++ )
   pointGeom[ j ] = RecPcGeomSmOut[ i ][ j ]
for( k = 0; k < numComps ; k++ )
   pointAttr[ k ] = RecPcAttrSmIn[ aIdx ][ i ][ k ]
```

sub clause 9.6.5 is applied to derive:

- a 2x2x2 grid neighbourhood corresponding to the current position, pointGeom[i]
- the top-left corner of 2x2x2 grid, s[i],
- the2x2x2 grid position associated with the current position, t[i], i = 0 to 2, inclusive
- a boolean value otherClusterPtCn

If otherClusterPtCnt is equal to 1, the following applies:

– The variation of attribute in the 2×2×2 cell neighbourhood of the current cell is determined as follows:

  The isOriginalCell array for the 2×2×2 cell neighbourhood is determined as follows:

```
for( dx = 0; dx < 2; dx++ ) {
    for( dy = 0; dy < 2; dy++ ) {
        for( dz = 0; dz < 2; dz++ ) {
            xIdx = s[ 0 ] + dx
            yIdx = s[ 1 ] + dy
            zIdx = s[ 2 ] + dz
            if( ( xIdx == t[0] ) && ( yIdx == t[1] ) && ( zIdx == t[2] ) ) {
                isOriginalCell[ dx ][ dy ][ dz ] = 1
                for( k=0; k < numComps; k++ ) {
                    currAttr[ k ] =  pointAttr[ k ]
                }
                if( abs( meanLuma[ xIdx ][ yIdx ][ zIdx ] − medianLuma[ xIdx ][ yIdx ][ zIdx ] )
                    <= afp_attribute_smoothing_threshold_variation[ aIdx ] ) {
                    lumaOrig = centroidLuma[ xIdx ][ yIdx ][ zIdx ]
                    for( k=0; k < numComps; k++ ) {
                        attrCentroid[ dx ][ dy ][ dx ][ k ] =
                            attrCenterGrid[ xIdx ][ yIdx ][ zIdx ][ k ]
                    }
                }
                else {
                    lumaOrig = 0.2126 * currAttr[ 0 ] + 0.7152 * currAttr[ 1 ] +
                                0.0722 * currAttr[ 2 ]
                    for( k=0; k < numComps; k++ ) {
                        attrCentroid[ dx ][ dy ][ dx ][ k ] = currAttr[ k ]
                    }
                }
            }
            else {
                isOriginalCell[ dx ][ dy ][ dz ] = 0
            }
        }
    }
}
```

– The variation in attribute values within a cell and difference between luma centroid values of the original cell and neighboring cells is determined as follows:

```
for( dx = 0; dx < 2; dx++ ) {
    for( dy = 0; dy < 2; dy++ ) {
        for( dz = 0; dz < 2; dz++ ) {
            xIdx = s[ 0 ] + dx
            yIdx = s[ 1 ] + dy
            zIdx = s[ 2 ] + dz
            if( isOriginalCell[ dx ][ dy ][ dz ] == 0 ) {
                if( cellCnt[ xIdx ][ yIdx ][ zIdx ] > 0 ) {
                    for( k=0; k < numComps; k++ ) {
                        attrCentroid[ dx ][ dy ][ dx ][ k ] =
                            attrCenterGrid[ xIdx ][ yIdx ][ zIdx ][ k ]
                    }
                    lumaN = centroidLuma[ xIdx ][ yIdx ][ zIdx ]
                    diff = abs( lumaOrig – lumaN )
                    var = meanLuma[ xIdx ][ yIdx ][ zIdx ]
                        − medianLuma[ xIdx ][ yIdx ][ zIdx ]
                    if( ( diff > afp_attribute_smoothing_threshold_difference[ aIdx ] ) ||
                        ( abs( var ) > afp_attribute_smoothing_threshold_variation[ aIdx ] ) ) {
                        for( k = 0; k < numComps; k++ ) {
                            attrCentroid[ dx ][ dy ][ dx ][ k ] = currAttr[ k ]
                        }
                    }
                }
                else {
                    for( k=0; k < numComps; k++ )
                        attrCentroid[ dx ][ dy ][ dx ][ k ] = currAttr[ k ]
                }
            }
        }
    }
}
```

– An 8-tap trilinear filter as described in clause 9.6.6 is applied to the attribute centroids, attrCentroid[x][y][z][k], where $0 \leq \mathrm{x}, \mathrm{y}, \mathrm{z} < 1$, $0 \leq \mathrm{k} < \mathrm{numComps} - 1$, in the 2×2×2 neighborhood as described in clause 9.6.5 is applied:
– If the following distortion criteria are satisfied, the output of the tri-linear filter will be added to recPCaSmOut.
  – The variable distToCentroid2 defined as abs(Ycell – Ycur) * 10 is greather than afp_attribute_smoothing_threshold_smoothing[aIdx], where Ycell and Ycur are the luma values of the centroid of points in the current cell (attrCentroid) and the current point (pointAttr), respectively.

9.6.3 Identification of the Center Grid

9.6.3.1 Identification of the Geometry Center Grid

Inputs to this process are:

– an array containing reconstructed positions, RecPcGeom[i], $0 \leq i < \mathrm{PointCnt}$.
– patch index information corresponding to each point in RecPcGeom[i].
– GridSize, indicating the smoothing grid size.

Output of this process are:

– a value, numCells, that specifies the number of cells in each direction (x, y, or z)
– an array containing reconstructed center grid geometry positions, geomCenterGrid[i][k], $0 \leq i < \mathrm{numCells}$ and k = 0 to 2, inclusive.
– Arrays containing boolean flag, cellDoSmoothing[x][y][z]) and cellCount[x][y]z] for all x, y, z = 0 to numCells - 1, inclusive.

The three-dimensional geometry coordinate space is divided into cells of size GridSize × GridSize × GridSize. The variable NumCells is derived as follows:

```
    for(k = 0; k < 3; k++) {
        maxRecPCg[ k ] = RecPcGeom[ 0 ][ k ]
        minRecPCg[k] = RecPcGeom[ 0 ][ k ]
    }
    for( i = 1; i < PointCnt ; i++ ) {
        for( k= 0; k < 3 ; k++ ) {
            maxRecPCg[ k ] = Max(maxRecPCg[ k ], RecPcGeom[ i ][ k ])
            minRecPCg[ k ] = Min(minRecPCg[ k ], RecPcGeom[ i ][ k ])
            bb[ k ] = maxRecPCg[ k ] – minRecPCg[ k ]
        }
    }
    numCells = ( Max( bb[ 0 ], Max( bb[ 1 ], bb[ 2 ] ) ) + GridSize − 1 ) / GridSize
```

The arrays cellDoSmoothing[x][y][z], cellCnt[x][y][z], cellPatchIdx[x][y][z], and geomCenterGrid[x][y][z][k] are initialized to 0 for all x, y, and z in the range of 0 to (numCells − 1), inclusive and k in the range of 0 to 2, inclusive.

for i = 0; to (PointCnt – 1) inclusive, the following is applied:

The variables xIdx, yIdx, and zIdx are derived as follows:

```
        xIdx = (RecPcGeom[ i] [ 0 ] / GridSize)

        yIdx = (RecPcGeom[ i ][ 1 ] / GridSize)

        zIdx = (RecPcGeom[ i ][ 2 ] / GridSize)
```

The arrays cellDoSmoothing[xIdx][yIdx][zIdx], cellCnt[xIdx][yIdx][zIdx], and cellPatchIdx[xIdx][yIdx][zIdx] are derived as follows:

if cellCnt[xIdx][yIdx][zIdx] is equal to 0, cellPatchIdx[xIdx][yIdx][zIdx] is set to the patch index of the patch that the current point belongs to.

Otherwise, if cellDoSmoothing[xIdx][yIdx][zIdx] is 0 and cellPatchIdx[xIdx][yIdx][zIdx] is not equal to the patch index that the current point RecPcGeom[i] belongs to, cellDoSmoothing[xIdx][yIdx][zIdx] is set to 1.

Apply the following steps:

```
          for( k = 0; k < 3 ; k++ )
              geomCenterGrid[ xIdx ][ yIdx][ zIdx ][ k ] += RecPcGeom[i][ k ];
           cellCnt[ xIdx ][ yIdx ][ zIdx ]++
```

for each xIdx, yIdx and zIdx where cellCnt[xIdx][yIdx][zIdx] is greater than 0, the following processes are performed for geometry.

```
    for( k = 0; k < 3 ; k++ )
        geomCenterGrid[ xIdx ][ yIdx ][ zIdx][ k] =
            geomCenterGrid[ xIdx ][ yIdx ][ zIdx ][ k ] / cellCnt[ xIdx][ yIdx ][ zIdx ]
```

9.6.3.2 Identification of the Attribute Center Grid

Inputs to this process are:

– an array containing reconstructed positions, RecPcGeom[i], with i in the range of 0 to PointCnt − 1, inclusive,
– aIdx, the attribute index,
– an array containing reconstructed attributes, RecPcAttr[aIdx][i], with i in the range of 0 to PointCnt − 1, inclusive, and
– numComps, indicating the number of attribute components.

Outputs of this process are:

– an array containing reconstructed center grid attribute values, attrCenterGrid[i][k], with i in the range of 0 to numCells − 1, inclusive, and k in the range of 0 to numComps − 1, inclusive,
– an array containing reconstructed center grid mean luma values, meanLuma[k], with k in the range of 0 to numComps − 1, inclusive, and
– an array containing reconstructed center grid luma median values, attrMean[i][k] with i in the range of 0 to numCells − 1, inclusive, and k in the range of 0 to numComps − 1, inclusive.

The elements of the arrays attrCenterGrid[x][y][z][m] and meanLuma[x][y][z] are initialized to 0 for all x, y, and z in the range of 0 to numCells − 1, inclusive and m in the range of 0 to numComps − 1, inclusive.

Clause 9.6.3.1 is applied to obtain the arrays cellDoSmoothing[x][y][z], cellCnt[x][y][z], and cellPatchIdx[x][y][z] for all x, y, and z in the range of 0 to numCells − 1, inclusive.

```
    For i = 0; to PointCnt – 1, inclusive, the following applies:

        The variables xIdx, yIdx, and zIdx are derived as follows:

        xIdx = (RecPcGeom[ i ][ 0 ] / GridSize)
        yIdx = (RecPcGeom[ i ][ 1 ] / GridSize)
        zIdx = (RecPcGeom[ i ][ 2 ] / GridSize)
If cellDoSmoothing[ xIdx ][ yIdx ][ zIdx ] is equal to 1, the following applies:

        for( k = 0; k < numComps ; k++ )
            attrCenterGrid[ xIdx ][ yIdx ][ zIdx ][ k ] += RecPcAttr[aIdx][i][ k ];
```

If cellCnt[x][y][z] is greater than 0 for x, y, and z in the range of 0 to numCells −1, inclusive, the mean and median luma values of attribute with index aIdx, for points belonging to that cell are calculated and assigned to arrays meanLuma[x][y][z] and medianLuma[x][y][z], respectively. Also, the luma value of attrCenterGrid is assigned to centroidLuma[x][y][z].

In case of a texture attribute in an RGB non-linear representation that uses the ITU-R BT.709 colour primaries, determine the luma value as follows:

```
    lumaValue = 0.2126 * RecPcAttr[ aIdx ][ i ][ 0 ]
            + 0.7152 * RecPcAttr[ aIdx ][ i ][ 1 ]
            + 0.0722 * RecPcAttr[ aIdx ][ i ][ 2 ];
    meanLuma[ xIdx ][ yIdx ][ zIdx ] += lumaValue
```

Processing of other attributes, e.g. texture attributes in other representations is not defined in this version of this Specification.

Derive attribute center grids.

```
    for( k = 0; k < numComps ; k++ )
        attrCenterGrid[ xIdx ][ yIdx ][zIdx ][ k] =
        attrCenterGrid[ xIdx ][ yIdx ][ zIdx ][ k ] ÷ cellCnt[ xIdx ][ yIdx ][ zIdx ]
    meanLuma[ xIdx ][ yIdx ][zIdx ] = meanLuma[ xIdx ][ yIdx ][ zIdx ] ÷ cellCnt[ xIdx ][ yIdx ][ zIdx ]
```

9.6.4 Identification of Boundary Points

Inputs to this process are:

– an occupancy map frame oFrame

– patch index information from the active geometry_patch_parameter_set

Output to this process is:

a container recPCBoundaryPointType for holding the list of the patch boundary type for the reconstructed point cloud.

This process is invoked from the geometry or the attribute smoothing process specified in clause 9.6.1 and 9.6.2 respectively.

A variable BoundaryPointType identifies if a point is located near a patch boundary. The BoundaryPointType may take values in the range of 0 to 1, inclusive. A BoundaryPointType value of 0 indicates that a point is not near a patch boundary. A two-dimensional (asps_frame_width) * (asps_frame_height) array BPTypes stores the BoundaryPointType values for each point in the projected point cloud frame.

A one-dimensional list recPCBoundaryPointType stores the BoundaryPointType for each point (x, y) in the projected point cloud frame for which the value of oFrame[y][x] is not equal to 0. BoundaryPointType values are assigned to BPTypes[y][x] and recPCBoundaryPointType as follows:

```
pointIndex = 0;
for( x = 0; x < asps_frame_width; x++ )
  for( y = 0; y < asps_frame_height; y++ ) {
      BPTypes[ y ][ x ] = 0
      recPCBoundaryPointType[ pointIndex ] = 0
      if( oFrame[ y ][ x ] ! = 0 )
          if( ( oFrame[ y ][ x − 1 ] = = 0 ) | | ( oFrame[ y ][ x + 1 ] = = 0 ) | |
              ( oFrame[ y − 1 ][ x ] = = 0 ) | | ( oFrame[ y + 1 ][ x ] = = 0 ) | |
              ( oFrame[ y − 1 ][ x − 1 ] = = 0 ) | | ( oFrame[ y − 1 ][ x + 1] = = 0 ) | |
              ( oFrame[ y + 1 ][ x − 1 ] = = 0 ) | | ( oFrame[ y + 1 ][ x + 1] = = 0 ) ) {
              recPCBoundaryPointType[ pointIndex ] = 1
              BPTypes[ y ][ x ] = 1
          }
      if( ( oFrame[ y ][ x ] ! = 0) && (recPCBoundaryPointType[ pointIndex ] ! = 1 ) )
          if( ( oFrame[ y − 2 ][ x − 2 ] = = 0 ) | | ( oFrame[ y − 2 ][ x − 1 ] = = 0 ) | |
              ( oFrame[ y − 2 ][ x ] = = 0 ) | | ( oFrame[ y − 2 ][ x + 1 ] = = 0 ) | |
              ( oFrame[ y − 2 ][ x + 2 ] = = 0 ) | | ( oFrame[ y − 1 ][ x − 2 ] = = 0 ) | |
              ( oFrame[ y  − 1 ][ x + 2 ] = = 0 ) | | ( oFrame[ y ][ x − 2 ] = = 0 ) | |
              ( oFrame[ y ][ x + 2 ] = = 0 ) | | ( oFrame[ y  + 1 ][ x − 2 ] = = 0 ) | |
              ( oFrame[ y + 1 ][ x + 2 ] = = 0 ) | | ( oFrame[ y + 2 ][ x − 2 ] = = 0 ) | |
              ( oFrame[ y + 2 ][ x − 1 ] = = 0 ) | | ( oFrame[ y + 2 ][ x ] = = 0 ) | |
              ( oFrame[ y + 2 ][ x + 1 ] = = 0 ) | | ( oFrame[ y + 2 ][ x + 2 ] = = 0 ) )
```

```
            recPCBoundaryPointType[ pointIndex ] = 1
        pointIndex++
    }
```

In the above assignment process, if (x – 2) or (x + 2) is outside the range of 0 to asps_frame_width – 1, inclusive, or (y – 2) or (y + 2) is outside the range of 0 to asps_frame_height – 1, inclusive, the values of BPTypes[y][x] and recPCBoundaryPointType[pointIndex] are assumed to be equal to 1.

9.6.5 Identification of 2x2x2 Cell for a Given Point Position

Inputs to this process are:

– current point position, pointGeom[k], k = 0 to 2, inclusive.
– array containing boolean flags, cellDoSmoothing[x][y][z]) for all x, y, and z = 0 to numCells - 1, inclusive.

Output of this process are:

– a boolean value, otherClusterPtCnt.
– an array containing the top-left corner of 2x2x2 grid, s[i], i = 0 to 2, inclusive.
– an array t[i], with i = 0 to 2, inclusive, which contains the 2x2x2 grid positions associated with the current position.

for each pointGeom[k]

```
    for( k = 0; k < 3 ; k++ ) {
        t[ k ] = ( pointGeom[ k ] / GridSize )
        s[ k ] = t[ k ] + ( ( ( pointGeom[ k ] % GridSize ) < ( GridSize / 2 ) ) ? –1 : 0 )
    }

    otherClusterPtCnt = 0
    for( dx = 0; dx < 2; dx++ ) {
        for( dy = 0; dy < 2; dy++ ) {
            for( dz = 0; dz < 2; dz++ ) {
                xIdx = s[ 0 ] + dx
                yIdx = s[ 1 ] + dy
                zIdx = s[ 2 ] + dz
                otherClusterPtCnt = otherClusterPtCnt | doSmoothing[ xIdx ][ yIdx ][ zIdx ]
            }
        }
    }
```

9.6.6 Trilinear Filtering

Inputs to this process are:

– a 3D point position, pointGeom
– the number of attribute components associated with the point that corresponds to the position pointGeom, numComps
– an array, valCentroid[x][y][z][k] for all values of x, y, and z, which are all in the range of 0 to 1, inclusive, and k = 0 to numComps - 1, inclusive.

Output of this process is:

– a 3D filtered value filtOut[k] for k = 0 to numComps – 1, inclusive.

First, the weights for trilinear filtering are calculated as follows:

```
for( k = 0; k < numComps ; k++ ) {
    w[k][1] = ( pointGeom[ k ] – s[ k ] * GridSize – (GridSize >> 1) )  << 1 + 1
    w[k][0] = ( (GridSize  << 1) – w[ k ][1] )
}
```

Then, filtering is performed as follows:

```
for( k = 0; k < numComps; ++) {
    filtOut[ k ] = 0
    for( dx = 0; dx < 2; dx++ ) {
        wx = w[ 0 ][ dx ]
        for( dy = 0; dy < 2; dy++ ) {
            wy = w[ 1 ][ dy ]
            for( dz = 0; dz < 2; dz++ ) {
                wz = w[ 2 ][ dz ]
                filtOut[ k ] += wx * wy * wz * valCentroid [dx ][ dy ][ dz ][ k ]
            }
        }
    }
    filtOut[ k ] = filtOut[ k ] ÷ ( 8 * GridSize * GridSize * GridSize )
}
```

10 Parsing process

10.1 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 10.2).

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

10.2.1 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++ )                                   (10-1)
    b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$\text{codeNum} = 2^{\text{leadingZeroBits}} - 1 + \text{read\_bits}(\ \text{leadingZeroBits}\ ) \qquad (10\text{-}2)$$

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 10-1 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 10-1. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as x_i in Table 10-1, with i in the range of 0 to leadingZeroBits − 1, inclusive. Each x_i is equal to either 0 or 1.

Table 10-1 Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges (informative)

Bit string form	Range of codeNum
0	0
1 0 x_0	1..2
110 x_1 x_0	3..6
1110 x_2 x_1 x_0	7..14
11110 x_3 x_2 x_1 x_0	15..30
111110 x_4 x_3 x_2 x_1 x_0	31..62
...	...

Table 10-2 illustrates explicitly the assignment of bit strings to codeNum values.

Table 10-2 – 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 10.2.2 with codeNum as input.

10.2.2 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in clause 10.2.1.

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 10-3 provides the assignment rule.

Table 10-3 – 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)

Annex A

Profiles, tiers, and levels

A.1 Overview of profiles, tiers, and levels

Profiles, tiers, and levels specify restrictions on the bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles, tiers, and levels may also be used to indicate interoperability points between individual decoder implementations.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

Each level of a tier specifies a set of limits on the values that may be taken by the syntax elements of this Specification. The same set of tier and level definitions is used with all profiles, but individual implementations may support a different tier and within a tier a different level for each supported profile. For any given profile, a level of a tier generally corresponds to a particular decoder processing load and memory capability.

Capabilities of V-PCC decoders conforming to this Specification are specified in terms of the ability to decode V-PCC streams conforming to the constraints of profiles, tiers and levels specified in this annex and other annexes. When expressing the capabilities of a decoder for a specified profile, the tier and level supported for that profile should also be expressed.

NOTE – The term "decode" in this context may or may not include the reconstruction in 3D space, i.e. "decode" may refer to only decoding the 2D video sub-bitstreams (i.e. geometry, occupancy, attribute(s)) and the associated atlas sub-bitstream, or it may include further reconstruction into 3D space. The respective steps included in the "decoding" are specified for each profile.

A.2 V-PCC profile, tier and level structure

V-PCC profiles follow a structured and flexible definition to allow for clearly identifying two distinct conformance points. These conformance points are illustrated in the decoding block diagram shown in Figure A-1.

The first conformance point, point A, covers the decoded attributes, geometry, and occupancy video sub-bitstreams, plus the decoded atlas sub-bitstream. It also covers the derived block to patch map information. It does not, however, cover the point cloud reconstruction process.

The second conformance point, point B, covers the point cloud reconstruction process.

A V-PCC profile consists of a combination of up to three profile components, identified from the syntax elements ptl_profile_codec_group_idc, ptl_profile_pcc_toolset_idc, and ptl_profile_reconstruction_idc, as shown in Figure A-2.

The first two profile components together describe conformance point A. More specifically, they describe the supported:

– video encoding specifications and their profiles (e.g., AVC Progressive High, HEVC Main/Main 10 etc), referred to as the **CodecGroup** profile component, and

– V-PCC specific tools (e.g., use of EOM and PLR etc), referred to as the **Toolset** profile component. The **Toolset** profile component describes the bitstream syntax structure.

The third profile component describes conformance point B, specifying the reconstruction tools supported or recommended to achieve conformance in 3D reconstruction. This is referred to as the **Reconstruction** profile component.

Together these profile components form the V-PCC profiles, following the naming convention **V-PCC CodecGroup Toolset Reconstruction**. For example, taking the first blocks of Figure A-2, the resulting V-PCC profile is named **V-PCC AVC Basic Rec0.**

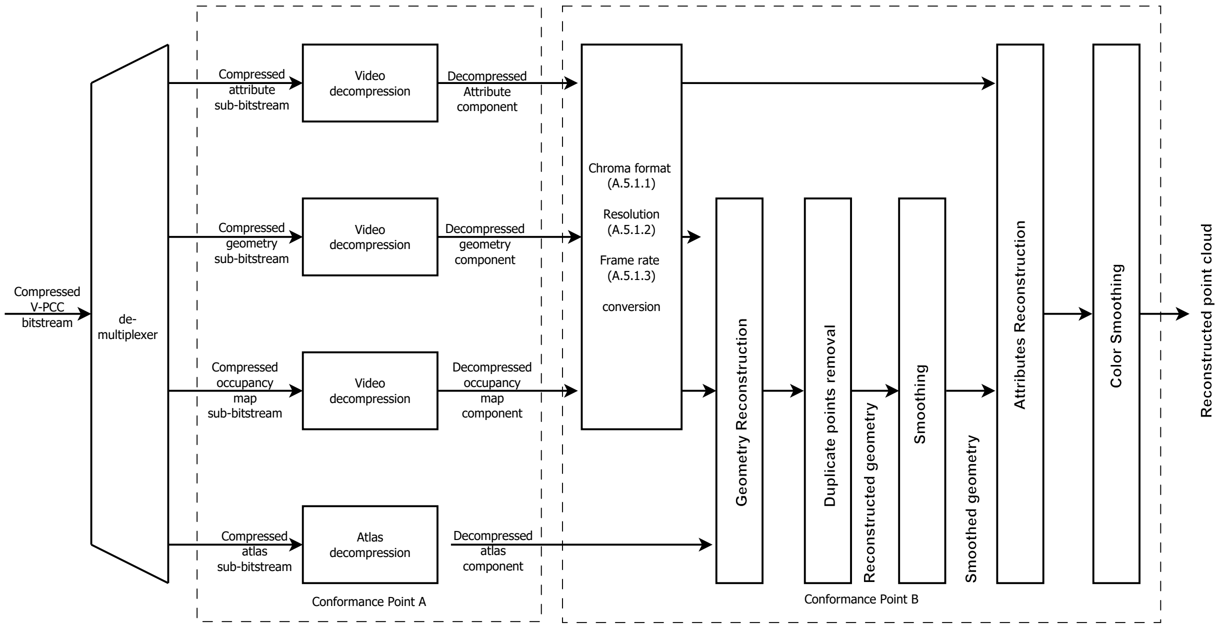
Compressed V-PCC bitstream
de-multiplexer
Compressed attribute sub-bitstream
Video decompression
Decompressed Attribute component
Compressed geometry sub-bitstream
Video decompression
Decompressed geometry component
Compressed occupancy map sub-bitstream
Video decompression
Decompressed occupancy map component
Compressed atlas sub-bitstream
Atlas decompression
Decompressed atlas component
Conformance Point A
Chroma format (A.5.1.1)
Resolution (A.5.1.2)
Frame rate (A.5.1.3)
conversion
Geometry Reconstruction
Duplicate points removal
Reconstructed geometry
Smoothing
Smoothed geometry
Attributes Reconstruction
Color Smoothing
Reconstructed point cloud
Conformance Point B

Figure A-1: V-PCC decoding block diagram with decoding conformance points A and B.

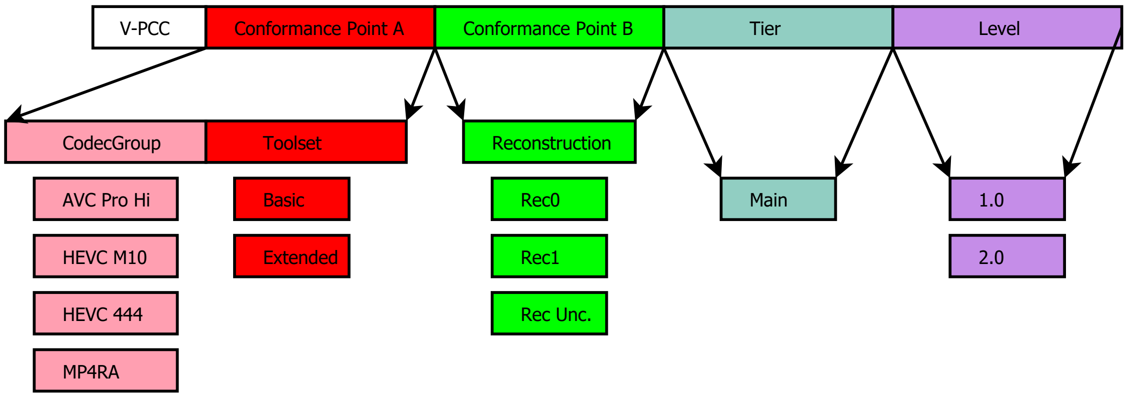

Figure A-2: The V-PCC profile, tier, and level structure.

Conformance can be specified for a single V-PCC bitstream, as specified in Annex B, or can be a collection of V-PCC sub-bitstream components that together specify a minimal collection of sub-bitstreams needed to reconstruct a point cloud. The minimal collection shall include the atlas, occupancy, and geometry map with index 0 sub-bitstreams. Additional geometry map sub-bitstreams should also be included. Attributes could also be included (with no specific order), but the included attributes shall be associated with a corresponding geometry map. An attribute map of index i that does not have an associated geometry map with the same index i should be discarded.

Any V-PCC bitstream, collection of V-PCC sub-bitstream components, or V-PCC decoder can claim conformance at two points. However, indicating conformance point A is mandatory, while for conformance point B it is optional, based on the selected profile. For example:

- **V-PCC AVC Basic,** is a valid profile to indicate decoding without any 3D reconstruction.
- **V-PCC AVC Basic Rec0,** is a valid profile to indicate decoding with elementary 3D reconstruction.
- **V-PCC AVC Rec0,** is NOT a valid profile as the Toolset information is missing.
- **V-PCC Basic Rec0,** is NOT a valid profile as the CodecGroup information is missing.

To indicate profile conformance of a bitstream at conformance point A, it is needed to signal the CodecGroup and Toolset profile components. To indicate profile conformance of a bitstream at conformance point B, it is needed to use all three profile components.

Decoders conforming to a V-PCC profile at conformance point A (identified by syntax elements ptl_profile_codec_group_idc and ptl_profile_pcc_toolset_idc) at a specific level (identified by a specific value of syntax element ptl_level_idc) of a specific tier (identified by a specific value of syntax element ptl_tier_flag) shall be capable of decoding all V-PCC bitstreams or collection of V-PCC sub-bitstream components, according to clause 8 (Decoding process), for which all of the following conditions apply:

– The V-PCC bitstream or the collection of V-PCC sub-bitstream components are indicated to conform to the supported CodecGroup and Toolset profile components.

– The V-PCC bitstream or the collection of V-PCC sub-bitstream components are indicated to conform to a level that is lower than or equal to the specified level.

– The V-PCC bitstream or the collection of V-PCC sub-bitstream components are indicated to conform to a tier that is lower than or equal to the specified tier.

Decoders conforming to a V-PCC profile at conformance point B (identified by syntax elements ptl_profile_codec_group_idc, ptl_profile_pcc_toolset_idc, and ptl_profile_reconstruction_idc) at a specific level (identified by a specific value of ptl_level_idc) of a specific tier (identified by a specific value of ptl_tier_flag) shall be capable of decoding all PCC bitstreams, according clause 8 (Decoding process) and clause 9 (Reconstruction process) for which all of the following conditions apply:

– The V-PCC bitstream or the collection of V-PCC sub-bitstream components are is indicated to conform to the supported CodecGroup, Toolset, and Reconstruction profile type.

– The V-PCC bitstream or the collection of V-PCC sub-bitstream components are indicated to conform to a level that is lower than or equal to the specified level.

– The V-PCC bitstream or the collection of V-PCC sub-bitstream components are indicated to conform to a tier that is lower than or equal to the specified tier.

A.3 V-PCC CodecGroup profile components

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

Table A-1 – Available CodecGroup profiles components

CodecGroup	ptl_profile_codec_group_idc
AVC Progressive High	0
HEVC Main10	1
HEVC444	2
MP4RA	3

Table A-2 provides a list of the supported codec functionalities for each CodecGroup profile component.

Table A-2 – CodecGroup profile component supported functionality

		AVC Progressive High			HEVC Main10			HEVC444			MP4RA		
	Capability	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes
Chroma format	Mono							✓	✓	✓	–	–	–
	4:2:0	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
	4:4:4									✓	–	–	–
Bitdepth	8 bit	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
	10 bit				✓	✓	✓	✓	✓	✓	–	–	–
Video coding specification	name	ISO/IEC 14496-10(AVC)			ISO/IEC 23008-2 (HEVC)						As defined by a component codec mapping SEI (E.2.13)		
	profile	Progressive High			Main 10			Main 4:4:4 10			–		

A tick mark in the above Table indicates that a particular feature is supported by the defined profile. A dash symbol, i.e. "–", indicates that support of a particular feature is specified by means outside this Specification.

All video sub-bitstreams of a V-PCC bitstream or a collection of V-PCC sub-bitstreams conform to the stream format indicated in the component codec mapping SEI (E.2.13).

The use of the MP4RA CodecGroup profile component is not meant as an interoperability point. Conformance to this profile can be specified by means outside this specification.

A.4 V-PCC toolset profile components

Table A-3 provides a list of the available toolset profile components for V-PCC.

Table A-3 – Available toolset profile components

Toolset	ptl_profile_pcc_toolset_idc
Basic	0
Extended	1

A.4.1 Basic and Extended toolset profile component

V-PCC toolset profile components indicating the Basic (ptl_profile_pcc_toolset_idc = 0) or the Extended (ptl_profile_pcc_toolset_idc = 1) toolset profile component shall conform to the restrictions specified in Table A-4. If a tool is not mentioned in Table A-4,, it is not restricted.

Table A-4– Required syntax element values for basic and extended toolset profile components

	Max allowed values

	ptl_profile_pcc_toolset_idc	
Syntax element	**0**	**1**
asps_enhanced_occupancy_map_for_depth_flag	0	–
vps_map_count_minus1	min(1, LevelMapCount – 1)	LevelMapCount – 1
vps_multiple_map_streams_present_flag (when vps_map_count_minus1 > 0)	when present, 1	–
vps_atlas_count_minus1	0	0
asps_point_local_reconstruction_enabled_flag	0	–
ai_attribute_dimension_minus1	2	–
ai_attribute_dimension_partitions_minus1	0	–
ai_attribute_partition_channels_minus1	–	2
asps_use_eight_orientations_flag	0	–
asps_45degree_projection_patch_present_flag	0	–

Furthermore, the following restrictions shall apply to a bitstream or a collection of V-PCC sub-bitstream components conforming to the Basic toolset profile component:

– All video sub-bitstreams, e.g. attribute, geometry, occupancy, shall have the same frame rate.

– All video sub-bitstreams shall have an intra random access pictures (IRAP) at IRAP points of an atlas sub-bitstream.

– All video bitstreams shall have the same prediction structure.

A.5 V-PCC reconstruction profile components

Table A-6 provides a list of the available reconstruction profile components for V-PCC.

Table A-5 – Available reconstruction profile components

reconstruction profile component	ptl_profile_reconstruction_idc
Rec0	0
Rec1	1
Rec Unconstrained	2

Table A-6 provides a list of the supported reconstruction functionalities for each reconstruction profile component.

Table A-6 – Supported reconstruction operations for each reconstruction profile component

Reconstruction Operation	Rec0	Rec1	Rec Unconstrained
chroma format conversion	see clause A.5.1.1	see clause A.5.1.1	

resolution upsampling	see clause A.5.1.2	see clause A.5.1.2	
Frame rate conversion	see clause A.5.1.3	see clause A.5.1.3	
Pixel deinterleaving	ignored	reconstructed, according to clause 9.4.1	unconstrained
PLR	ignored	reconstructed, according to clause 9.4.2	unconstrained
EOM	ignored	reconstructed, according to clause 9.4.4	unconstrained
Duplicate point removal	ignored	reconstructed, according to clause 9.4.8	unconstrained
RAW patches	ignored	reconstructed, according to clause 9.5	unconstrained
Smoothing	ignored	reconstructed, according to clause 9.6	unconstrained

A.5.1 V-PCC reconstruction decoder conformance

Decoders conforming to a V-PCC profile with a reconstruction profile component have to perform reconstruction operations. Conformance is assessed at conformance point A (decoded attribute, geometry, and occupancy bitstreams together with decoded patch metadata and decoded block to patch map) and conformance point B (reconstructed point cloud).

In order to check for decoder conformance at point B the following hypothetical reference operations are defined:

– Hypothetical reference chroma format conversion (A.5.1.1)

– Hypothetical reference resolution upsampling (A.5.1.2)

– Hypothetical reference frame rate conversion (A.5.1.3)

– Hypothetical reference attribute transfer process (A.5.1.4)

A decoder may wish to use these processes exactly as defined in their corresponding clauses. However, a decoder can still be considered as conforming to this specification if the resulting point cloud is perceived as similar in reconstruction as the point cloud constructed using these hypothetical reference processes. Similarity of these processes with the hypothetical reference processes is outside of the scope of this Specification.

For decoder conformance of the attribute values associated with each point in the point cloud, no further conversion to any other domain other than the domain used by the video representation associated with the respective attribute, shall be used. For example, if an attribute of type ATTR_TEXTURE with three components was coded using the HEVC specification and was associated with parameters colour_primaries equal to 1, transfer_characteristics equal to 1, and matrix_coeffs equal to 1, then the attribute is assumed to be in the ITU-R BT.709 non-constant luminance YCbCr representation.

A.5.1.1 Hypothetical reference chroma format conversion

When an attribute is of type ATTR_TEXTURE and has three components with a chroma format other than 4:4:4, it is converted to a 4:4:4 chroma format according to its specified chroma sample location. For conformance purposes, the 4 tap filter specified in Table A-7 shall be used:

Table A-7 – 4-phase chroma resampling filter

Phase p	Interpolation coefficients			
	f_c[p, 0]	f_c[p, 1]	f_c[p, 2]	f_c[p, 3]
0	0	32	0	0
1	−1	8	27	−2
2	−2	18	18	−2
3	−2	27	8	−1

A.5.1.2 Hypothetical reference resolution upscaling

For the occupancy map, pixel replication is used to upsample the occupancy map to a width of asps_nominal_width and a height of asps_nominal_height.

For all attributes except those of type ATTR_TEXTURE, pixel replication is used to upsample the attribute to a width of asps_nominal_width and a height of asps_nominal_height.

For attributes of type ATTR_TEXTURE, the upsampling filters specified in clause H.8.1.4.2.2, Table H.1 of the HEVC specification are used to upsample the attribute to a width of asps_nominal_width and a height of asps_nominal_height.

For geometry, bilinear interpolation is used to upsample the geometry to a width of asps_nominal_width and a height of asps_nominal_height.

A.5.1.3 Hypothetical reference frame rate conversion

Frame repetition using the nearest available frame from the past is used to perform frame rate conversion.

A.5.1.4 Hypothetical reference attribute transfer process

None.

A.6 Tiers and Levels

For purposes of comparison of tier capabilities, the tier with ptl_tier_flag equal to 0 is considered to be a lower tier than the tier with ptl_tier_flag equal to 1. Currently only a single tier, the Main tier, is specified for V-PCC. This is indicated by setting the syntax element ptl_tier_flag equal to 0. It is a requirement for bitstream conformance to this current version of the Specification that ptl_tier_flag shall always be equal to 0.
For purposes of comparison of level capabilities, a particular level of a specific tier is considered to be a lower level than some other level of the same tier when the value of the ptl_level_idc of the particular level is less than that of the other level.

Table A-8 specifies the general point cloud and VPS related limits for each level of each tier.

Table A-9 specifies the general atlas ASPS and tile related limits for each level of each tier.

Table A-10 specifies the general video bitstream related limits for each level of each tier.

A tier and level to which a bitstream conforms are indicated by the syntax elements ptl_tier_flag and ptl_level_idc. ptl_level_idc shall be set equal to a value of 30 times the level number specified in Table A-8, Table A-9, and Table A-10.

Table A-8 — General Point Cloud or VPS related level limits

Level	Max # of points per second MaxNumPointsPerSec	Max # of points per frame MaxNumPointsPerFrame	Max atlas frame size MaxAtlasSize	Max point cloud frame rate MaxPointCloudFrameRate	Max # of maps LevelMapCount	Max # of attributes MaxNumAttributeCount	Max # of attribute dimensions MaxNumAttributeDims
1.0	30,000,000	1,000,000	2,228,224	30	2	1	3
2.0	30,000,000	1,000,000	2,228,224	60	2	3	3

Table A-9 — General atlas ASPS and tile related level limits

Level	Max # Patches MaxNumPatches	Max # RAW patches MaxNumRawPatches	Max # RAW points MaxNumRawPoints	Max # EOM patches MaxNumEOMPatche	Max # EOM points MaxNumEOMPoints	Max # CAB for atlas MaxCAB	Max atlas bitrate MaxAtlasBR	Max # of tile columns MaxTileColumns	Max # of tile rows MaxTileRows
1.0	256	32	50000	32	50000		1,500,000	11	10
2.0	256	64	50000	32	50000		2,000,000	11	10

Table A-10 — General video bitstream level limits

Level	Max # luma samples per second per video stream MaxLumaSamplesPerSec	Max # Mbit/s per video stream MaxBitratePerStream	Max # CPB per video stream MaxCPBPerStream
1.0	133,693,440	50	20,000
2.0	133,693,440	50	20,000

Annex B

V-PCC Sample stream format

(This annex forms an integral part of this Recommendation | International Standard.)

B.1 General

This annex specifies syntax and semantics of a sample stream format specified for use by applications that deliver some or all of the V-PCC unit stream as an ordered stream of bytes or bits within which the locations of V-PCC unit boundaries need to be identifiable from patterns in the data, such as Rec. ITU-T H.222.0 | ISO/IEC 13818-1 systems or Recommendation ITU-T H.320 systems. For bit-oriented delivery, the bit order for the sample stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The sample stream format starts with a sample stream header and consists of a sequence of sample stream V-PCC unit syntax structures. Each sample stream V-PCC unit syntax structure contains one element, named ssvu_vpcc_unit_size, that specifies a size, followed by one vpcc_unit(ssvu_vpcc_unit_size) syntax structure. This vpcc_unit is essentially of size ssvu_vpcc_unit_size. The number of bits used to encode the element ssvu_vpcc_unit_size is specified in the sample stream header.

B.2 Sample stream V-PCC unit syntax and semantics

B.2.1 Sample stream V-PCC header syntax

sample_stream_vpcc_header() {	**Descriptor**
ssvh_unit_size_precision_bytes_minus1	u(3)
ssvh_reserved_zero_5bits	u(5)
}	

B.2.2 Sample stream V-PCC unit syntax

sample_stream_vpcc_unit() {	**Descriptor**
ssvu_vpcc_unit_size	u(v)
vpcc_unit(ssvu_vpcc_unit_size)	
}	

B.2.3 Sample stream V-PCC header semantics

The sample stream V-PCC header shall always be at the beginning of the sample stream V-PCC stream.

ssvh_unit_size_precision_bytes_minus1 plus 1 specifies the precision, in bytes, of the ssvu_vpcc_unit_size element in all sample stream V-PCC units. ssvh_unit_size_precision_bytes_minus1 shall be in the range of 0 to 7.

ssvh_reserved_zero_5bits shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for sshv_reserved_zero_5bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of sshv_reserved_zero_5bits.

B.2.4 Sample stream V-PCC unit semantics

The order of sample stream V-PCC units in the sample stream shall follow the decoding order of the V-PCC units contained in the sample stream V-PCC units. The content of each sample stream V-PCC unit is associated with the same access unit as the V-PCC unit contained in the sample stream V-PCC unit.

ssvu_vpcc_unit_size specifies the size, in bytes, of the subsequent vpcc_unit. The number of bits used to represent ssvu_vpcc_unit_size is equal to (ssvh_unit_size_precision_bytes_minus1 + 1) * 8.

Annex C

NAL Sample stream format

(This annex forms an integral part of this Recommendation | International Standard.)

C.1 General

This annex specifies syntax and semantics of a sample stream format specified for use by applications that deliver some or all of the atlas NAL unit stream as an ordered stream of bytes or bits within which the locations of atlas NAL unit boundaries need to be identifiable from patterns in the data, such as Rec. ITU-T H.222.0 | ISO/IEC 13818-1 systems or Recommendation ITU-T H.320 systems. For bit-oriented delivery, the bit order for the sample stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The sample stream format starts with a sample stream header and consists of a sequence of sample stream NAL unit syntax structures. Each sample stream NAL unit syntax structure contains one element, named ssnu_nal_unit_size, that specifies a size, followed by one nal_unit(ssnu_nal_unit_size) syntax structure. This nal_unit is essentially of size ssnu_nal_unit_size. The number of bits used to encode the element ssnu_nal_unit_size is specified in the sample stream header.

C.2 Sample stream NAL unit syntax and semantics

C.2.1 Sample stream NAL header syntax

sample_stream_nal_header() {	**Descriptor**
ssnh_unit_size_precision_bytes_minus1	u(3)
ssnh_reserved_zero_5bits	u(5)
}	

C.2.2 Sample stream NAL unit syntax

sample_stream_nal_unit() {	**Descriptor**
ssnu_nal_unit_size	u(v)
nal_unit(ssnu_nal_unit_size)	
}	

C.2.3 Sample stream NAL header semantics

The sample stream NAL header shall always be at the beginning of the NAL stream.

ssnh_unit_size_precision_bytes_minus1 plus 1 specifies the precision, in bytes, of the ssnu_nal_unit_size element in all sample stream NAL units. ssnh_unit_size_precision_bytes_minus1 shall be in the range of 0 to 7.

ssnh_reserved_zero_5bits shall be equal to 0 in bitstreams conforming to this version of this Specification. Other values for ssnh_reserved_zero_5bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ssnh_reserved_zero_5bits.

C.2.4 Sample stream NAL unit semantics

The order of sample stream NAL units in the sample stream shall follow the decoding order of the NAL units contained in the sample stream NAL units. The content of each sample stream NAL unit is associated with the same access unit as the NAL unit contained in the sample stream NAL unit.

ssnu_nal_unit_size specifies the size, in bytes, of the subsequent NAL_unit. The number of bits used to represent ssnu_nal_unit_size is equal to (ssnh_unit_size_precision_bytes_minus1 + 1) * 8.

Annex D

Atlas Hypothetical Reference Decoder

(This annex forms an integral part of this Recommendation | International Standard.)

D.1 General

This annex specifies the hypothetical reference decoder (HRD) and its use to check atlas bitstream and decoder conformance.

Two types of bitstreams or bitstream subsets are subject to HRD conformance checking for this Specification. The first type, called a Type I bitstream, is an ACL NAL unit stream containing only the ACL NAL units and NAL units with nal_unit_type equal to NAL_FD (filler data NAL units) for all access units in the bitstream. The second type, called a Type II bitstream, contains, in addition to the ACL NAL units and filler data NAL units for all access units in the atlas bitstream, additional non-ACL NAL units-other than filler data units.

Figure D-1 shows the types of bitstream conformance points checked by the atlas sub-bitstream HRD.

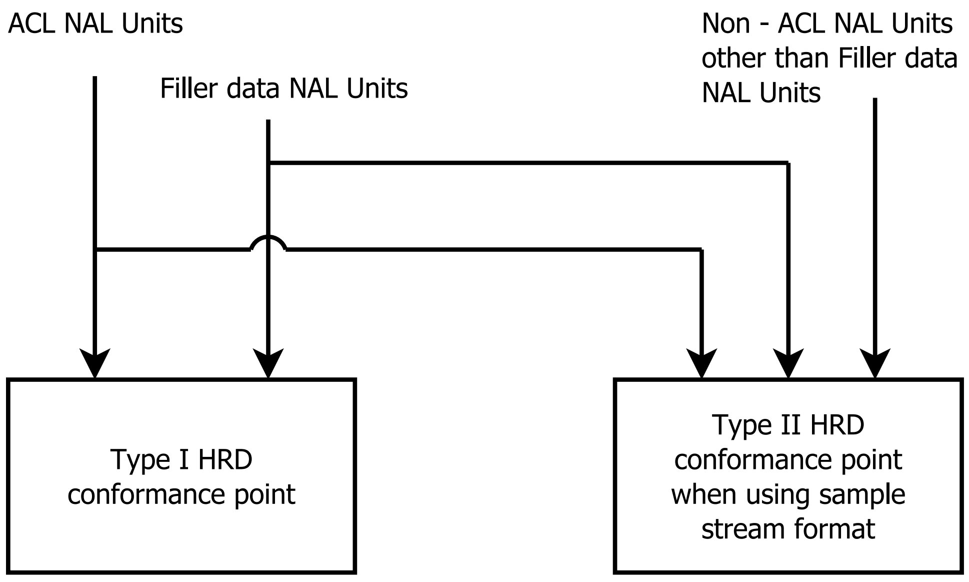
ACL NAL Units
Filler data NAL Units
Non - ACL NAL Units other than Filler data NAL Units
Type I HRD conformance point
Type II HRD conformance point when using sample stream format

Figure D-1 – Structure of Atlas NAL unit streams for HRD conformance checks

The syntax elements of non-ACL NAL units (or their default values for some of the syntax elements), required for the HRD, are specified in the semantic clauses of clause 7 and Annex E.

Two types of HRD parameter sets (NAL HRD parameters and non-ACL HRD parameters) are used. The HRD parameter sets are signalled through the hrd_parameters ( ) syntax structure, which is part of the VUI syntax structure.

Multiple tests may be needed for checking the conformance of an atlas sub-bitstream, which is referred to as the bitstream under test. For each test, the following steps apply in the order listed:

1. An operation point under test, denoted as TargetOp, is set by selecting a target highest TemporalId value OpTid.

2. The hrd_parameters( ) syntax structure in the VUI of the active ASPS (or provided through some external means not specified in this Specification) that applies to TargetOp that is selected. Within the selected hrd_parameters( ) syntax structure, if AtlasBitstreamToDecode is a Type I bitstream the variable NalHrdModeFlag is set equal to 0; otherwise (AtlasBitstreamToDecode is a Type II bitstream). When AtlasBitstreamToDecode is a Type II bitstream and NalHrdModeFlag is equal to 0, all non-ACL NAL units except filler data NAL units from the NAL unit stream (as specified in Annex C), when present, are discarded from AtlasBitstreamToDecode and the remaining atlas bitstream is assigned to AtlasBitstreamToDecode.

3. A coded atlas access unit associated with a buffering period SEI message (present in AtlasBitstreamToDecode or available through external means not specified in this Specification) applicable to TargetOp is selected as the HRD initialization point and referred to as atlas access unit 0.

4. For each atlas access unit in AtlasBitstreamToDecode starting from atlas access unit 0, the buffering period SEI message (present in AtlasBitstreamToDecode or available through external means not specified in this Specification) that is associated with the atlas access unit and applies to TargetOp is selected, the atlas frame timing SEI message (present in AtlasBitstreamToDecode or available through external means not specified in this Specification) that is associated with the atlas access unit and applies to TargetOp is selected.

5. A value of SchedSelIdx is selected. The selected SchedSelIdx shall be in the range of 0 to hrd_cab_cnt_minus1 inclusive, where hrd_cab_cnt_minus1 is found in the hrd_parameters( ) syntax structure as selected above.

6. When the coded atlas frame in access unit 0 has nal_unit_type equal to NAL_CRA/GCRA or NAL_BLA_W_LP/NAL_GBLA_W_LP, and bp_irap_cab_params_present_flag in the selected buffering period SEI message is equal to 1, either of the following applies for selection of the initial CAB removal delay and delay offset:

   – If NalHrdModeFlag is equal to 1, the default initial CAB removal delay and the delay offset represented by bp_nal_initial_cab_removal_delay[SchedSelIdx] and bp_nal_initial_cab_removal_offset[SchedSelIdx], respectively, in the selected buffering period SEI message are selected. Otherwise, the default initial CAB removal delay and delay offset represented by bp_acl_initial_cab_removal_delay[SchedSelIdx] and bp_acl_initial_cab_removal_offset[SchedSelIdx], respectively, in the selected buffering period SEI message are selected. The variable DefaultInitCabParamsFlag is set equal to 1.

   – If NalHrdModeFlag is equal to 1, the alternative initial CAB removal delay and delay offset represented by bp_nal_initial_alt_cab_removal_delay[SchedSelIdx] and bp_nal_initial_alt_cab_removal_offset[SchedSelIdx], respectively, in the selected buffering period SEI message are selected. Otherwise, the alternative initial CAB removal delay and delay offset represented by bp_acl_initial_alt_cab_removal_delay[SchedSelIdx] and bp_acl_initial_alt_cab_removal_offset[SchedSelIdx], respectively, in the selected buffering period SEI message are selected. The variable DefaultInitCafbParamsFlag is set equal to 0, and the RASL access units associated with access unit 0 are discarded from AtlasBitstreamToDecode and the remaining bitstream is assigned to AtlasBitstreamToDecode.

Each conformance test consists of a combination of one option in each of the above steps. When there is more than one option for a step, for any conformance test only one option is chosen. All possible combinations of all the steps form the entire set of conformance tests. For each operation point under test, the number of bitstream conformance tests to be performed is equal to n0 * n1 * (n2 * 2 + n3) * n4, where the values of n0, n1, n2, n3 and n4 are specified as follows:

– n0 is derived as follows:

  – If AtlasBitstreamToDecode is a Type I bitstream, n0 is equal to 1.

  – Otherwise (AtlasBitstreamToDecode is a Type II bitstream), n0 is equal to 2.

– n1 is equal to hrd_cab_cnt_minus1 + 1.

– n2 is the number of access units in AtlasBitstreamToDecode that each is associated with a buffering period SEI message applicable to TargetOp and for each of which both of the following conditions are true:

  – nal_unit_type is equal to NAL_GCRA/GBLA/ NAL_CRA/BLA-for the ACL NAL units.

  – The associated buffering period SEI message applicable to TargetOp has bp_irap_cab_params_present_flag equal to 1.

– n3 is the number of access units in AtlasBitstreamToDecode that each is associated with a buffering period SEI message applicable to TargetOp and for each of which one or both of the following conditions are true:

  – nal_unit_type is not equal to NAL_GCRA, NAL_GBLA, NAL_CRA, or NAL_BLA for the ACL NAL units.

  – The associated buffering period SEI message applicable to TargetOp has bp_irap_cab_params_present_flag equal to 0.

– n4 is equal to 2.

When AtlasBitstreamToDecode is a Type II patch bitstream, the following applies:

– If the hrd_parameters( ) syntax structure that immediately follows the condition "if(bp_acl_hrd_parameters_present_flag)" is selected, the test is conducted at the Type I conformance point shown in Figure D-1, and only ACL and filler data NAL units are counted for the input bit rate and CAB storage.

– Otherwise (the hrd_parameters( ) syntax structure that immediately follows the condition "if(nal_hrd_parameters_present_flag)" is selected), the test is conducted at the Type II conformance point shown in Figure D-1, and all bytes of the Type II bitstream, which is a NAL unit stream, are counted for the input bit rate and CAB storage.

NOTE 2 – NAL HRD parameters established by a value of SchedSelIdx for the Type II conformance point shown in Figure D-1 are sufficient to also establish ACL HRD conformance for the Type I conformance point shown in Figure D-1 for the same values of InitCabRemovalDelay[SchedSelIdx], BitRate[SchedSelIdx] and CabSize[SchedSelIdx] for the variable bit rate (VBR) case (hrd_cbr_flag[SchedSelIdx] equal to 0). This is because the data flow into the Type I conformance point is a subset of the data flow into the Type II conformance point and because, for the VBR case, the CAB is allowed to become empty and stay empty until the time a next atlas is scheduled to begin to arrive. For example, when decoding a CAS conforming to one or more of the profiles specified in Annex A using the decoding process specified in clause 8.4.

All ASPSs, and AFPSs referred to in the ACL NAL units and the corresponding buffering period and atlas timing information SEI messages shall be conveyed to the HRD, in a timely manner, either in the bitstream (by non-ACL NAL units), or by other means not specified in this Specification.

In Annex D and Annex E, the specification for "presence" of non-ACL NAL units that contain ASPSs, and AFPSs, buffering period SEI messages, atlas frame timing SEI messages is also satisfied when those NAL units (or just some of them) are conveyed to atlas frame decoders (or to the HRD) by other means not specified in this Specification. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

NOTE 3 – As an example, synchronization of such a non-ACL NAL unit, conveyed by means other than presence in the bitstream, with the NAL units that are present in the bitstream, can be achieved by indicating two points in the bitstream, between which the non-ACL NAL unit would have been present in the bitstream, had the atlas frame encoder decided to convey it in the bitstream.

When the content of such a non-ACL NAL unit is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the non-ACL NAL unit is not required to use the same syntax as specified in this Specification.

NOTE 4 – When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of this clause based solely on information contained in the bitstream. When the HRD information is not present in the bitstream, as is the case for all "stand-alone" Type I bitstreams, conformance can only be verified when the HRD data are supplied by some other means not specified in this Specification.

The HRD contains a coded atlas buffer (CAB), an instantaneous decoding process, a decoded atlas buffer (DAB), and output cropping as shown in Figure D-2.

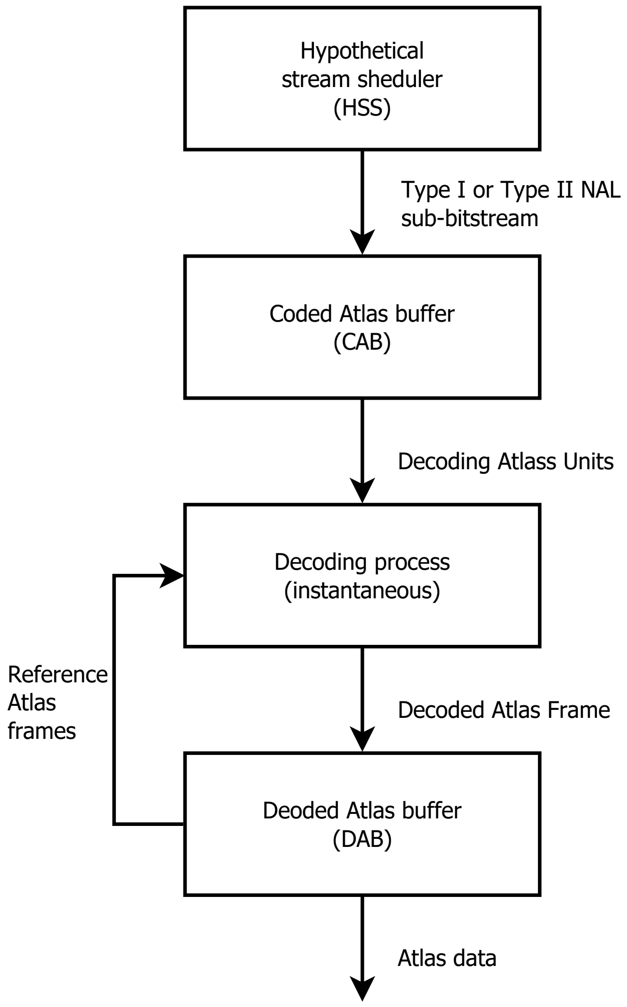
Hypothetical stream sheduler (HSS)
Type I or Type II NAL sub-bitstream
Coded Atlas buffer (CAB)
Decoding Atlass Units
Decoding process (instantaneous)
Reference Atlas frames
Decoded Atlas Frame
Deoded Atlas buffer (DAB)
Atlas data

Figure D-2 – Atlas Frame HRD buffer model

For each atlas bitstream conformance test, the CAB size (number of bits) is CabSize[SchedSelIdx] as specified in clause F.3.3, where SchedSelIdx and the HRD parameters are specified above in this clause. The DAB size (number of atlas frames storage buffers) is asps_max_dec_atlas_frame_buffering_minus1 + 1.

The following is specified for expressing the constraints in this annex:

– Each access unit is referred to as access unit n, where the number n identifies the particular access unit. Access unit 0 is selected per step 3 above. The value of n is incremented by 1 for each subsequent access unit in decoding order.

– Atlas frame n refers to the coded atlas frame or the decoded atlas frame of access unit n.

The HRD operates as follows:

– The HRD is initialized at decoding unit 0, with both the CAB and the DAB being set to be empty (the DAB fullness is set equal to 0).

NOTE 5 – After initialization, the HRD is not initialized again by subsequent buffering period SEI messages.

– Data associated with decoding units that flow into the CAB according to a specified arrival schedule are delivered by the hypothetical atlas stream scheduler (HSS).

– The data associated with each decoding unit are removed and decoded instantaneously by the instantaneous decoding process at the CAB removal time of the decoding unit.

– Each decoded atlas frame is placed in the DAB.

– A decoded atlas frame is removed from the DAB when it becomes no longer needed for inter prediction reference and no longer needed for output.

For each bitstream conformance test, the operation of the CAB is specified in clause D.2, the instantaneous decoder operation is specified in clauses 2 through 10, the operation of the DAB is specified in clause D.3

HSS and HRD information concerning the number of enumerated delivery schedules and their associated bit rates and buffer sizes is specified in clauses F.2.2 and F.3.2. The HRD is initialized as specified by the buffering period SEI message specified in clauses and . The removal timing of decoding units from the CAB and output timing of decoded atlases from the DAB is specified using information in atlas frame timing SEI messages (specified in clauses D.2.3 and D.3.3). All timing information relating to a specific decoding unit shall arrive prior to the CAB removal time of the decoding unit.

The requirements for atlas sub-bitstream conformance are specified in clause D.4 and the HRD is used to check conformance of atlas sub-bitstreams as specified above in this clause and to check conformance of decoders as specified in clause D.5.

NOTE 6 – While conformance is guaranteed under the assumption that all “atlas frame”-rates and clocks used to generate the atlas sub-bitstream match exactly the values signalled in the atlas sub-bitstream, in a real system each of these may vary from the signalled or specified value.

All the arithmetic in this annex is performed with real values, so that no rounding errors can propagate. For example, the number of bits in a CAB just prior to or after removal of a decoding unit is not necessarily an integer.

The variable ClockTick is derived as follows and is called a clock tick:

ClockTick = vui_num_units_in_tick ÷ vui_time_scale (D-1)

D.2 Operation of coded atlas frame buffer

D.2.1 General

The specifications in this clause apply independently to each set of coded atlas frame buffer (CAB) parameters that is present and to both the Type I and Type II conformance points shown in Figure D-1 and the set of CAB parameters is selected as specified in clause D.1.

D.2.2 Timing of decoding unit arrival

The decoding unit is considered as an access unit, for derivation of the initial and final CAB arrival times for access unit n.

The variables InitCabRemovalDelay[SchedSelIdx] and InitCabRemovalDelayOffset[SchedSelIdx] are derived as follows:

- If one or more of the following conditions are true, InitCabRemovalDelay[SchedSelIdx] and InitCabRemovalDelayOffset[SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements bp_nal_initial_alt_cab_removal_delay[SchedSelIdx] and bp_nal_initial_alt_cab_removal_offset[SchedSelIdx], respectively, when NalHrdModeFlag is equal to 1 or bp_acl_initial_alt_cab_removal_delay[SchedSelIdx] and bp_acl_initial_alt_cab_removal_offset[SchedSelIdx], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message syntax elements are selected as specified in clause D.1:
  - Access unit 0 is a NAL_CRA/GCRA/NAL_BLA/GBLA access unit, and the value of bp_irap_cab_params_present_flag of the buffering period SEI message is equal to 1 and one or more of the following conditions are true:
    - DefaultInitCabParamsFlag is equal to 0.
- Otherwise, InitCabRemovalDelay[SchedSelIdx] and InitCabRemovalDelayOffset[SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements bp_nal_initial_cab_removal_delay[SchedSelIdx] and bp_nal_initial_cab_removal_offset[SchedSelIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_acl_initial_cab_removal_delay[SchedSelIdx] and bp_acl_initial_cab_removal_offset[SchedSelIdx], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message syntax elements are selected as specified in clause D.1.

The time at which the first bit of decoding unit m begins to enter the CAB is referred to as the initial arrival time initArrivalTime[m].

The initial arrival time of decoding unit m is derived as follows:

- If the decoding unit is decoding unit 0 (i.e., when m is equal to 0), initArrivalTime[0] is set equal to 0.
- Otherwise (the decoding unit is decoding unit m with m > 0), the following applies:
  - If hrd_cbr_flag[SchedSelIdx] is equal to 1, the initial arrival time for decoding unit m is equal to the final arrival time (which is derived below) of decoding unit m − 1, i.e.,

initArrivalTime[m] = AuFinalArrivalTime[m − 1] (D-2)

- Otherwise (hrd_cbr_flag[SchedSelIdx] is equal to 0), the initial arrival time for decoding unit m is derived as follows:

initArrivalTime[m] = Max(AuFinalArrivalTime[m − 1], initArrivalEarliestTime[m]) (D-3)

where initArrivalEarliestTime[m] is derived as follows:

- The variable tmpNominalRemovalTime is derived as follows:

tmpNominalRemovalTime = AuNominalRemovalTime[m] (D-4) where AuNominalRemovalTime[m] is the nominal CAB removal time of access unit m respectively, as specified in clause D.2.3.

– If decoding unit m is not the first decoding unit of a subsequent buffering period, initArrivalEarliestTime[m] is derived as follows:

$$\text{initArrivalEarliestTime[m]} = \text{tmpNominalRemovalTime} - (\text{InitCabRemovalDelay[SchedSelIdx]} + \text{InitCabRemovalDelayOffset[SchedSelIdx]}) \div 90000 \quad \text{(D-5)}$$

– Otherwise (decoding unit m is the first decoding unit of a subsequent buffering period), initArrivalEarliestTime[m] is derived as follows:

$$\text{initArrivalEarliestTime[m]} = \text{tmpNominalRemovalTime} - (\text{InitCabRemovalDelay[SchedSelIdx]} \div 90000) \quad \text{(D-6)}$$

The final arrival time for decoding unit m is derived as follows:

$$\text{AuFinalArrivalTime[m]} = \text{initArrivalTime[m]} + \text{sizeInbits[m]} \div \text{BitRate[SchedSelIdx]} \quad \text{(D-7)}$$

where sizeInbits[m] is the size in bits of decoding unit m, counting the bits of the ACL NAL units and the filler data NAL units for the Type I conformance point or all bits of the Type II bitstream for the Type II conformance point, where the Type I and Type II conformance points are as shown in Figure D-1.

The values of SchedSelIdx, BitRate[SchedSelIdx] and CabSize[SchedSelIdx] are constrained as follows:

– If the content of the selected hrd_parameters( ) syntax structures for the access unit containing decoding unit m and the previous access unit differ, the HSS selects a value SchedSelIdx1 of SchedSelIdx from among the values of SchedSelIdx provided in the selected hrd_parameters( ) syntax structures for the access unit containing decoding unit m that results in a BitRate[SchedSelIdx1] or CabSize[SchedSelIdx1] for the access unit containing decoding unit m. The value of BitRate[SchedSelIdx1] or CabSize[SchedSelIdx1] may differ from the value of BitRate[SchedSelIdx0] or CabSize[SchedSelIdx0] for the value SchedSelIdx0 of SchedSelIdx that was in use for the previous access unit.

– Otherwise, the HSS continues to operate with the previous values of SchedSelIdx, BitRate[SchedSelIdx] and CabSize[SchedSelIdx].

When the HSS selects values of BitRate[SchedSelIdx] or CabSize[SchedSelIdx] that differ from those of the previous access unit, the following applies:

– The variable BitRate[SchedSelIdx] comes into effect at the initial CAB arrival time of the current access unit.

– The variable CabSize[SchedSelIdx] comes into effect as follows:

  – If the new value of CabSize[SchedSelIdx] is greater than the old CAB size, it comes into effect at the initial CAB arrival time of the current access unit.

  – Otherwise, the new value of CabSize[SchedSelIdx] comes into effect at the CAB removal time of the current access unit.

D.2.3 Timing of decoding unit removal and decoding of decoding unit

The variables InitCabRemovalDelay[SchedSelIdx], InitCabRemovalDelayOffset[SchedSelIdx], CabDelayOffset and DabDelayOffset are derived as follows:

– If one or more of the following conditions are true, CabDelayOffset is set equal to the value of the buffering period SEI message syntax element bp_cab_delay_offset, DabDelayOffset is set equal to the value of the buffering period SEI message syntax element bp_dab_delay_offset, and InitCabRemovalDelay[SchedSelIdx] and InitCabRemovalDelayOffset[SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements bp_nal_initial_alt_cab_removal_delay[SchedSelIdx] and bp_nal_initial_alt_cab_removal_offset[SchedSelIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_acl_initial_alt_cab_removal_delay[SchedSelIdx] and bp_acl_initial_alt_cab_removal_offset[SchedSelIdx], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message containing the syntax elements is selected as specified in clause D.1.

  – Access unit 0 is a BLA access unit for which the coded atlas has nal_unit_type equal to NAL_BLA_W_RADL or NAL_BLA_N_LP, and the value of bp_irap_cab_params_present_flag of the buffering period SEI message is equal to 1.

  – Access unit 0 is a BLA access unit for which the coded atlas has nal_unit_type equal to NAL_BLA_W_LP or is a NAL_CRA access unit, and the value of bp_irap_cab_params_present_flag of the buffering period SEI message is equal to 1, and one or more of the following conditions are true:

    – DefaultInitCabParamsFlag is equal to 0.

– Otherwise, InitCabRemovalDelay[SchedSelIdx] and InitCabRemovalDelayOffset[SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements bp_nal_initial_cab_removal_delay[SchedSelIdx] and bp_nal_initial_cab_removal_offset[SchedSelIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_acl_initial_cab_removal_delay[SchedSelIdx] and bp_acl_initial_cab_removal_offset[SchedSelIdx], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message containing the syntax elements is selected as specified in clause D.1, CabDelayOffset and DpbDelayOffset are both set equal to 0.

The nominal removal time of the access unit n from the CAB is specified as follows:

– If access unit n is the access unit with n equal to 0 (the access unit that initializes the HRD), the nominal removal time of the access unit from the CAB is specified by:

$$\text{AuNominalRemovalTime}[\ 0\ ] = \text{InitCabRemovalDelay}[\ \text{SchedSelIdx}\ ] \div 90000 \quad \text{(D-8)}$$

– Otherwise, the following applies:

– When access unit n is the first access unit of a buffering period that does not initialize the HRD, the following applies:

The nominal removal time of the access unit n from the CAB is specified by:

```
if( !concatenationFlag ) {
    baseTime = AuNominalRemovalTime[ firstAtlasInPrevBuffPeriod ]
    tmpCabRemovalDelay = AuCabRemovalDelayVal
}
else {
    baseTime = AuNominalRemovalTime[ prevNonDiscardablePic ]
    tmpCabRemovalDelay =Max( ( auCabRemovalDelayDeltaMinus1 + 1 ),            (D-9)
    Ceil( ( InitCabRemovalDelay[ SchedSelIdx ] ÷ 90000 +
    AuFinalArrivalTime[ n − 1 ] − AuNominalRemovalTime[ n − 1 ] ) ÷ ClockTick )
}
AuNominalRemovalTime( n ) = baseTime + ClockTick * ( tmpCabRemovalDelay −
            CabDelayOffset )
```

where AuNominalRemovalTime[firstAtlasInPrevBuffPeriod] is the nominal removal time of the first access unit of the previous buffering period, AuNominalRemovalTime[prevNonDiscardableAtlas] is the nominal removal time of the preceding atlas in decoding order with TemporalId equal to 0 that is not a RASL, RADL or sub-layer non-reference atlas frame, AuCabRemovalDelayVal is the value of AuCabRemovalDelayVal derived according to aft_cab_removal_delay_minus1 in the atlas timing SEI message, selected as specified in clause D.1, associated with access unit n, and concatenationFlag and auCabRemovalDelayDeltaMinus1 are the values of the syntax elements bp_concatenation_flag and bp_atlas_cab_removal_delay_delta_minus1, respectively, in the buffering period SEI message, selected as specified in clause D.1, associated with access unit n.

After the derivation of the nominal CAB removal time and before the derivation of the DAB output time of access unit n, the values of CabDelayOffset and DabDelayOffset are updated as follows:

– If one or more of the following conditions are true, CabDelayOffset is set equal to the value of the buffering period SEI message syntax element bp_cab_delay_offset, and DabDelayOffset is set equal to the value of the buffering period SEI message syntax element bp_dab_delay_offset, where the buffering period SEI message containing the syntax elements is selected as specified in clause D.1:

– Access unit n is a BLA access unit for which the coded atlas has nal_unit_type equal to BLA_W_RADL or BLA_N_LP, and the value of bp_irap_cab_params_present_flag of the buffering period SEI message is equal to 1.

– Access unit n is a BLA access unit for which the coded atlas has nal_unit_type equal to BLA_W_LP or is a CRA access unit, and the value of bp_irap_cab_params_present_flag of the buffering period SEI message is equal to 1, and UseAltCabParamsFlag for access unit n is equal to 1.

– Otherwise, CabDelayOffset and DpbDelayOffset are both set equal to 0.

– When access unit n is not the first access unit of a buffering period, the nominal removal time of the access unit n from the CAB is specified by:

$$\text{AuNominalRemovalTime}[\ n\ ] = \text{AuNominalRemovalTime}\ [\ \text{firstAtlasInCurrBuffPeriod}\ ] + \text{ClockTick} * (\ \text{AuCabRemovalDelayVal} - \text{CabDelayOffset}\ ) \quad \text{(D-10)}$$

where AuNominalRemovalTime[firstAtlasInCurrBuffPeriod] is the nominal removal time of the first access unit of the current buffering period, and AuCabRemovalDelayVal is the value of AuCabRemovalDelayVal derived according to aft_cab_removal_delay_minus1 in the atlas timing SEI message, selected as specified in clause D.1, associated with access unit n.

CAB removal time of access unit n, the access unit is instantaneously decoded.

D.3 Operation of the decoded atlas frame buffer

D.3.1 General

The specifications in this clause apply independently to each set of decoded atlas frame buffer (DAB) parameters selected as specified in clause D.1.

The decoded atlas frame buffer contains atlas frame storage buffers. Each of the atlas frame storage buffers may contain a decoded atlas frame that is marked as "used for reference" or is held for future output. The processes specified in clauses D.3.2 and D.3.3 are sequentially applied as specified below.

D.3.2 Removal of atlas frames from the DAB before decoding of the current atlas frame

The removal of atlas frames from the DAB before decoding of the current atlas frame (but after parsing the tile group header of the first tile group of the current atlas frame) happens instantaneously at the CAB removal time of the first decoding unit of access unit n (containing the current atlas frame) and proceeds as follows:

- The decoding process for reference atlas list (RAL) as specified in clause 8.4.3.2 is invoked.
- When the current atlas is an IRAP atlas frame with NoRaslOutputFlag equal to 1 that is not atlas frame 0, the following ordered steps are applied:

1. The variable NoOutputOfPriorAtlasFramesFlag is derived for the decoder under test as follows:
   - If the current atlas frame is a CRA atlas frame, NoOutputOfPriorAtlasFramesFlag is set equal to 1.
   - Otherwise, NoOutputOfPriorAtlasFramesFlag is set equal to 0.
2. The value of NoOutputOfPriorAtlasFramesFlag derived for the decoder under test is applied for the HRD, such that when the value of NoOutputOfPriorAtlasFramesFlag is equal to 1, all atlas storage buffers in the DAB are emptied without output of the atlas frames they contain, and the DAB fullness is set equal to 0.
   - When both of the following conditions are true for any atlass k in the DAB, all such atlas frames k in the DAB are removed from the DAB:
   - atlas frame k is marked as "unused for reference".
   - atlas frame k has AtlasFrameOutputFlag equal to 0 or its DAB output time is less than or equal to the CAB removal time of the first decoding unit (denoted as decoding unit m) of the current atlas frame n; i.e., DabOutputTime[k] is less than or equal to DuCabRemovalTime[m].
   - For each atlas frame that is removed from the DAB, the DAB fullness is decremented by one.

D.3.3 Atlas frame output

The processes specified in this clause happen instantaneously at the CAB removal time of access unit n, AuCabRemovalTime[n].

When atlas n has AtlasFrameOutputFlag equal to 1, its DAB output time DabOutputTime[n] is derived as follows, where the variable firstAtlasFrameInBufferingPeriodFlag is equal to 1 if access unit n is the first access unit of a buffering period and 0 otherwise:

```
DabOutputTime[ n ] = AuCabRemovalTime[ n ] + ClockTick * atlasFrameDafbOutputDelay
if( firstAtlasFrameInBufferingPeriodFlag)                                        (D-11)
    DabOutputTime[ n ]  −=  ClockTick * DabDelayOffset
```

where atlasFrameDafbOutputDelay is the value of aft_dab_output_delay in the atlas frame timing SEI message associated with access unit n, when present, in the decoding unit information SEI messages associated with access unit n.

The output of the current atlas is specified as follows:

– If AtlasFrameOutputFlag is equal to 1 and DabOutputTime[n] is equal to AuCabRemovalTime[n], the current atlas frame is output.

– Otherwise, if AtlasFrameOutputFlag is equal to 0, the current atlas frame is not output, but will be stored in the DAB as specified in clause D.3.4

– Otherwise (AtlasFrameOutputFlag is equal to 1 and DabOutputTime[n] is greater than AuCabRemovalTime[n]), the current atlas frame is output later and will be stored in the DAB (as specified in clause D.3.4) and is output at time DabOutputTime[n] unless indicated not to be output at a time that precedes DabOutputTime[n].

When atlas frame n is an atlas frame that is output and is not the last atlas frame of the bitstream that is output, the value of the variable DabOutputInterval[n] is derived as follows:

DabOutputInterval[n] = DabOutputTime[nextAtlasFrameInOutputOrder] −
DabOutputTime[n] (D-12)

where nextAtlasFrameInOutputOrder is the atlas frame that follows atlas frame n in output order and has AtlasFrameOutputFlag equal to 1.

D.3.4 Current decoded atlas frame marking and storage

The current decoded atlas frame is stored in the DAB in an empty atlas frame storage buffer, the DAB fullness is incremented by one. When asps_long_term_ref_atlas_frames_flag is equal to 1, this atlas frame is marked as "used for long-term reference". After all the tile groups of the current atlas frame have been decoded, this atlas frame is marked as "used for short-term reference".

NOTE – Unless more memory than required by the level limit is available for storage of decoded atlas frames, decoders should start storing decoded parts of the current atlas frames into the DAB when the first tile group is decoded and continue storing more decoded samples as the decoding process proceeds.

D.3.5 Removal of atlas frames from the DAB after decoding of the current atlas frame

Immediately after decoding of the current atlas frame, at the CAB removal time of the last decoding unit of access unit n (containing the current atlas frame), the current decoded atlas frame is removed from the DAB, and the DAB fullness is decremented by one.

D.4 Bitstream conformance

A bitstream of coded data conforming to this Specification shall fulfil all requirements specified in this clause.

The bitstream shall be constructed according to the syntax, semantics and constraints specified in this Specification outside of this annex.

The first coded atlas in a bitstream shall be an IRAP atlas, i.e., an IDR atlas frame, or a CRA atlas frame.

The patch bitstream is tested by the HRD for conformance as specified in clause D.1.

For each current atlas frame, let the variables maxAtlasFrameOrderCnt and minAtlasFrameOrderCnt be set equal to the maximum and the minimum, respectively, of the AtlasFrameOrderCntVal values of the following atlas frames:

– The current atlas frame.

– The previous atlas frame in decoding order.

– The short-term reference atlas frames in the reference atlas list (RAL) of the current atlas frame.

– All atlas frames n that have AtlasFrameOutputFlag equal to 1, AuCabRemovalTime[n] less than AuCabRemovalTime[currAtlasFrame] and DabOutputTime[n] greater than or equal to AuCabRemovalTime[currAtlasFrame], where currAtlasFrame is the current atlas frame.

All of the following conditions shall be fulfilled for each of the bitstream conformance tests:

1. For each atlas access unit n, with n greater than 0, associated with a buffering period SEI message, let the variable deltaTime90k[n] be specified as follows:

   $$\text{deltaTime90k[n]} = 90000 * (\text{ AuNominalRemovalTime[n]} - \text{AuFinalArrivalTime[n − 1] }) \quad \text{(D-13)}$$

   The value of InitCabRemovalDelay[SchedSelIdx] is constrained as follows:

   – If hrd_cbr_flag[SchedSelIdx] is equal to 0, the following condition shall be true:

   $$\text{InitCabRemovalDelay[SchedSelIdx]} <= \text{Ceil(deltaTime90k[n])} \quad \text{(D-14)}$$

   – Otherwise (hrd_cbr_flag[SchedSelIdx] is equal to 1), the following condition shall be true:

   $$\text{Floor(deltaTime90k[n])} <= \text{InitCabRemovalDelay[SchedSelIdx]} <= \text{Ceil(deltaTime90k[n])} \quad \text{(D-15)}$$

   NOTE 1 – The exact number of bits in the CAB at the removal time of each atlas frame may depend on which buffering period SEI message is selected to initialize the HRD. Encoders must take this into account to ensure that all specified constraints must be obeyed regardless of which buffering period SEI message is selected to initialize the HRD, as the HRD may be initialized at any one of the buffering period SEI messages.

2. A CAB overflow is specified as the condition in which the total number of bits in the CAB is greater than the CAB size. The CAB shall never overflow.

3. When low_delay_hrd_flag[HighestTid] is equal to 0, the CAB shall never underflow. A CAB underflow is specified as follows:

   – A CAB underflow is specified as the condition in which the nominal CAB removal time of access unit n AuNominalRemovalTime[n] is less than the final CAB arrival time of access unit n AuFinalArrivalTime[n] for at least one value of n.

4. The nominal removal times of atlases from the CAB (starting from the second atlas in decoding order) shall satisfy the constraints on AuNominalRemovalTime[n] and AuCabRemovalTime[n] expressed in Annex A.

5. For each current atlas frame, after invocation of the process for removal of atlass from the DAB as specified in clause D.3.2, the number of decoded atlas frames in the DAB, including all atlas frames n that are marked as "used for reference", or that have AtlasFrameOutputFlag equal to 1 and AuCabRemovalTime[n] less than AuCabRemovalTime[currAtlasFrame], where currAtlasFrame is the current atlas frame, shall be less than or equal to asps_max_dec_atlas_frame_buffering_minus1.

6. All reference atlas frames shall be present in the DAB when needed for prediction. Each atlas frame that has AtlasFrameOutputFlag equal to 1 shall be present in the DAB at its DAB output time unless it is removed from the DAB before its output time by one of the processes specified in clause D.3.2.

7. The value of DabOutputInterval[n] as given by Equation D-12, which is the difference between the output time of an atlas frame and that of the first atlas frame following it in output order and having AtlasFrameOutputFlag equal to 1, shall satisfy the constraint expressed in Annex A for the profile, tier and level specified in the bitstream using the decoding process specified in clauses 2 through 10.
8. For any two atlass m and n in the same CAS, when DabOutputTime[m] is greater than DabOutputTime[n], the AtlasFrameOrderCntVal of atlas frame m shall be greater than the AtlasOrderCntVal of atlas n.

NOTE 2 – All atlass of an earlier CAS in decoding order that are output are output before any atlass of a later CAS in decoding order. Within any particular CAS, the atlass that are output are output in increasing AtlasOrderOrderCntVal order.

D.5 Decoder conformance

D.5.1 General

A decoder conforming to this Specification shall fulfil all requirements specified in this clause.

A decoder claiming conformance to a specific profile, tier and level shall be able to successfully decode all atlas frame bitstreams that conform to the atlas frame bitstream conformance requirements specified in clause D.4, in the manner specified in Annex A, provided that all ASPSs, and AFPSs referred to in the ACL NAL units and appropriate buffering period and atlas frame timing SEI messages are conveyed to the decoder, in a timely manner, either in the bitstream (by non-ACL NAL units), or by external means not specified in this Specification.

When an atlas frame bitstream contains syntax elements that have values that are specified as reserved and it is specified that decoders shall ignore values of the syntax elements or NAL units containing the syntax elements having the reserved values, and the atlas frame bitstream is otherwise conforming to this Specification, a conforming decoder shall decode the atlas frame bitstream in the same manner as it would decode a conforming atlas frame bitstream and shall ignore the syntax elements or the NAL units containing the syntax elements having the reserved values as specified.

There are two types of conformance that can be claimed by an atlas frame decoder: output timing conformance and output order conformance.

To check conformance of an atlas frame decoder, test bitstreams conforming to the claimed profile, tier and level, as specified in clause D.4 are delivered by a hypothetical atlas frame stream scheduler (HSS) both to the HRD and to the atlas frame decoder under test (DUT). All decoded atlas frames output by the HRD shall also be output by the DUT, each decoded atlas frame output by the DUT shall be an atlas frame with AtlasFrameOutputFlag equal to 1, and, for each such decoded atlas frame output by the DUT, the values of all atlas units that are output shall be equal to the values of the atlas units produced by the specified atlas frame decoding process.

For output timing atlas frame decoder conformance, the HSS operates as described above, with delivery schedules selected only from the subset of values of SchedSelIdx for which the bit rate and CAB size are restricted as specified in Annex A for the specified profile, tier and level or with "interpolated" delivery schedules as specified below for which the bit rate and CAB size are restricted as specified in Annex A. The same delivery schedule is used for both the HRD and the DUT.

When the HRD parameters and the buffering period SEI messages are present with hrd_cab_cnt_minus1[HighestTid] greater than 0, the atlas frame decoder shall be capable of decoding the atlas frame bitstream as delivered from the HSS operating using an "interpolated" delivery schedule specified as having peak bit rate r, CAB size c(r) and initial CAB removal delay (f(r) ÷ r) as follows:

$$\alpha = (r - \text{BitRate}[\ \text{SchedSelIdx} - 1\ ]) \div (\text{BitRate}[\ \text{SchedSelIdx}\ ] - \text{BitRate}[\ \text{SchedSelIdx} - 1\ ]), \qquad \text{(D-16)}$$

$$c(r) = \alpha * \text{CabSize}[\text{SchedSelIdx}] + (1 - \alpha) * \text{CabSize}[\text{SchedSelIdx} - 1] \tag{D-17}$$

$$f(r) = \alpha * \text{InitCabRemovalDelay}[\text{SchedSelIdx}] * \text{BitRate}[\text{SchedSelIdx}] + (1 - \alpha) * \text{InitCabRemovalDelay}[\text{SchedSelIdx} - 1] * \text{BitRate}[\text{SchedSelIdx} - 1] \tag{D-18}$$

for any SchedSelIdx > 0 and r such that BitRate[SchedSelIdx − 1] <= r <= BitRate[SchedSelIdx] such that r and c(r) are within the limits as specified in Annex A for the maximum bit rate and buffer size for the specified profile, tier and level.

NOTE 1 – InitCabRemovalDelay[SchedSelIdx] can be different from one buffering period to another and have to be re-calculated.

For output timing atlas frame decoder conformance, an HRD as described above is used and the timing (relative to the delivery time of the first bit) of atlas frame output is the same for both the HRD and the DUT up to a fixed delay.

For output order atlas frame decoder conformance, the following applies:

– The HSS delivers the bitstream AtlasBitstreamToDecode to the DUT "by demand" from the DUT, meaning that the HSS delivers bits (in decoding order) only when the DUT requires more bits to proceed with its processing.

  NOTE 2 – This means that for this test, the coded atlas frame buffer of the DUT could be as small as the size of the largest decoding unit.

– A modified HRD as described below is used, and the HSS delivers the bitstream to the HRD by one of the schedules specified in the atlas frame bitstream AtlasBitstreamToDecode such that the bit rate and CAB size are restricted as specified in Annex A. The order of atlas frame output shall be the same for both the HRD and the DUT.

– The HRD CAB size is given by CabSize[SchedSelIdx] as specified in clause F.3.3, where SchedSelIdx and the HRD parameters are selected as specified in clause D.1. The DAB size is given by asps_max_dec_atlas_frame_buffering_minus1 + 1. Removal time from the CAB for the HRD is the final bit arrival time and decoding is immediate. The operation of the DAB of this HRD is as described in clauses D.5.2 through D.5.2.3.

D.5.2 Operation of the output order DAB

D.5.2.1 General

The decoded atlas frame buffer contains atlas frame storage buffers. Each of the atlas frame storage buffers contains a decoded atlas frame that is marked as "used for reference" or is held for future output. The process for output and removal of atlas frames from the DAB before decoding of the current atlas frame as specified in clause D.5.2.2 is invoked, the invocation of the process for current decoded atlas frame marking and storage as specified in clause D.5.2.3, further followed by the invocation of the process for removal of atlas frame from the DAB after decoding of the current atlas frame as specified in clause D.5.2.4, and finally followed by the invocation of the process for additional bumping as specified in clause D.5.2.3. The "bumping" process is specified in clause D.5.2.4 and is invoked as specified in clauses and D.5.2.3.

D.5.2.2 Output and removal of atlas frames from the DAB

The output and removal of atlas frames from the DAB before the decoding of the current atlas frame (but after parsing the tile group header of the first tile group of the current atlas frame) happens instantaneously when the first decoding unit of the access unit containing the current atlas frame is removed from the CAB and proceeds as follows:

- The decoding process for RAL as specified in clause 8.4.3.2 is invoked.
- If the current atlas is an IRAP atlas frame with NoRaslOutputFlag equal to 1 that is not atlas frame 0, the following ordered steps are applied:
  1. The variable NoOutputOfPriorAtlasFramesFlag is derived for the atlas frame decoder under test as follows:
     - If the current atlas frame is a CRA atlas frame, NoOutputOfPriorAtlasFramesFlag is set equal to 1.
     - Otherwise, NoOutputOfPriorAtlasFramesFlag is set equal to 0.
  2. The value of NoOutputOfPriorAtlasFramesFlag derived for the decoder under test is applied for the HRD as follows:
     - If NoOutputOfPriorAtlasFramesFlag is equal to 1, all atlas storage buffers in the DAB are emptied without output of the atlas frames they contain and the DAB fullness is set equal to 0.
     - Otherwise (NoOutputOfPriorAtlasFramesFlag is equal to 0), all atlas frame storage buffers containing atlas frames that are marked as "not needed for output" and "unused for reference" are emptied (without output) and all non-empty atlas frames storage buffers in the DAB are emptied by repeatedly invoking the "bumping" process specified in clause D.5.2.4 and the DAB fullness is set equal to 0.
- Otherwise (the current atlas frame is not an IRAP atlas frame with NoRaslOutputFlag equal to 1), all atlas frames storage buffers containing atlas frames which are marked as "not needed for output" and "unused for reference" are emptied (without output). For each atlas frame storage buffer that is emptied, the DAB fullness is decremented by one. When one or more of the following conditions are true, the "bumping" process specified in clause D.5.2.4 is invoked repeatedly while further decrementing the DAB fullness by one for each additional atlas frame storage buffer that is emptied, until none of the following conditions is true:
  - The number of atlas frames in the DAB is greater than or equal to asps_max_dec_atlas_frame_buffering_minus1 + 1.

D.5.2.3 Additional bumping

The processes specified in this clause happen instantaneously when the last decoding unit of access unit n containing the current atlas frames is removed from the CAB.

When the current atlas frames has AtlasFrameOutputFlag equal to 1, for each atlas frames in the DAB that is marked as "needed for output" and follows the current atlas frames in output order, the associated variable AtlasFrameLatencyCount is set equal to AtlasFrameLatencyCount + 1.

The following applies:

- If the current decoded atlas has AtlasFrameOutputFlag equal to 1, it is marked as "needed for output" and its associated variable AtlasFrameLatencyCount is set equal to 0.
- Otherwise (the current decoded atlas framehas AtlasFrameOutputFlag equal to 0), it is marked as "not needed for output".

When one or more of the following conditions are true, the "bumping" process specified in clause D.5.2.4 is invoked repeatedly until none of the following conditions are true:

- The number of atlas frames in the DAB that are marked as "needed for output" is greater than asps_max_dec_atlas_frame_buffering_minus1.

D.5.2.4 "Bumping" process

The "bumping" process consists of the following ordered steps:

1. The atlas frame that is first for output is selected as the one having the smallest value of atlasFrameOrderCntVal of all atlas frames in the DAB marked as "needed for output".

NOTE – For any two atlas frames atlasFrameA and atlasFrameB that belong to the same CAS and are output by the "bumping process", when atlasFrameA is output earlier than atlasFrameB, the value of AtlasFrameOrderCntVal of atlasFrameA is less than the value of AtlasFrameOrderCntVal of atlasFrameB.

Annex E

Supplemental enhancement information

(This annex forms an integral part of this Recommendation | International Standard.)

E.1 General

This annex specifies syntax and semantics for SEI message payloads.

SEI messages assist in processes related to decoding, reconstruction, display, or other purposes. However, SEI messages are not required by the decoding process. Conforming decoders are not required to process this information for output order conformance to this Specification (see Annex C for the specification of conformance). Some SEI message information is required to check bitstream conformance and for output timing decoder conformance.

In clause C.5.2 including its subclauses, specification for presence of SEI messages is also satisfied when those messages (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified in this Specification. When present in the bitstream, SEI messages shall obey the syntax and semantics specified in clause 7.3.8 and this annex. When the content of an SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

E.2 SEI payload syntax

E.2.1 General SEI message syntax

sei_payload(payloadType, payloadSize) {	**Descriptor**
if(psd_unit_type = = PSD_PREFIX_SEI) {	
if(payloadType = = 0)	
buffering_period(payloadSize)	
else if(payloadType = = 1)	
atlas_frame_timing(payloadSize)	
else if(payloadType = = 2)	
filler_payload(payloadSize)	
else if(payloadType = = 3)	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType = = 4)	
user_data_unregistered(payloadSize)	
else if(payloadType = = 5)	
recovery_point(payloadSize)	
else if(payloadType = = 6)	
no_display(payloadSize)	
else if(payloadType = = 7)	
time_code(payloadSize)	
else if(payloadType = = 8)	
regional_nesting(payloadSize)	

else if(payloadType = = 9)	
sei_manifest(payloadSize)	
else if(payloadType = = 10)	
sei_prefix_indication(payloadSize)	
else if(payloadType = = 11)	
geometry_transformation_params(payloadSize)	
else if(payloadType = = 12)	
attribute_transformation_params(payloadSize)	
else if(payloadType = = 13)	
active_substreams(payloadSize)	
else if(payloadType = = 14)	
component_codec_mapping(payloadSize)	
else if(payloadType = = 15)	
volumetric_tiling_info(payloadSize)	
else	
reserved_sei_message(payloadSize)	
}	
else { /* psd_unit_type = = PSD_SUFFIX_SEI */	
if(payloadType = = 2)	
filler_payload(payloadSize)	
else if(payloadType = = 3)	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType = = 4)	
user_data_unregistered(payloadSize)	
else	
reserved_sei_message(payloadSize)	
}	
if(more_data_in_payload( )) {	
if(payload_extension_present( ))	
sp_reserved_payload_extension_data	u(v)
byte_alignment( )	
}	
}	

E.2.2 Filler payload SEI message syntax

filler_payload(payloadSize) {	**Descriptor**
for(k = 0; k < payloadSize; k++)	
ff_byte /* equal to 0xFF */	f(8)
}	

E.2.3 User data registered by Recommendation ITU-T T.35 SEI message syntax

user_data_registered_itu_t_t35(payloadSize) {	**Descriptor**
itu_t_t35_country_code	b(8)
if(itu_t_t35_country_code != 0xFF)	
i = 1	
else {	
itu_t_t35_country_code_extension_byte	b(8)
i = 2	
}	
do {	
itu_t_t35_payload_byte	b(8)
i++	
} while(i < payloadSize)	
}	

E.2.4 User data unregistered SEI message syntax

user_data_unregistered(payloadSize) {	**Descriptor**
uuid_iso_iec_11578	u(128)
for(i = 16; i < payloadSize; i++)	
user_data_payload_byte	b(8)
}	

E.2.5 Recovery point SEI message syntax

recovery_point(payloadSize) {	**Descriptor**
recovery_afoc_cnt	se(v)
exact_match_flag	u(1)
broken_link_flag	u(1)
}	

E.2.6 No display SEI message syntax

no_display(payloadSize) {	**Descriptor**
}	

E.2.7 Reserved SEI message syntax

reserved_sei_message(payloadSize) {	**Descriptor**
for(i = 0; i < payloadSize; i++)	
reserved_sei_message_payload_byte	b(8)
}	

E.2.8 SEI manifest SEI message syntax

sei_manifest(payloadSize) {	**Descriptor**
manifest_num_sei_msg_types	u(16)
for(i = 0; i < manifest_num_sei_msg_types; i++) {	
manifest_sei_payload_type[i]	u(16)
manifest_sei_description[i]	u(8)
}	
}	

E.2.9 SEI prefix indication SEI message syntax

sei_prefix_indication(payloadSize) {	**Descriptor**
prefix_sei_payload_type	u(16)
num_sei_prefix_indications_minus1	u(8)
for(i = 0; i <= num_sei_prefix_indications_minus1; i++) {	
num_bits_in_prefix_indication_minus1[i]	u(16)
for(j = 0; j <= num_bits_in_prefix_indication_minus1[i]; j++)	
sei_prefix_data_bit[i][j]	u(1)
while(!byte_aligned( ))	
byte_alignment_bit_equal_to_one /* equal to 1 */	f(1)
}	
}	

E.2.10 Geometry transformation parameters SEI message syntax

geometry_transformation_params(payloadSize) {	**Descriptor**
gtp_cancel_flag	u(1)
if(!gtp_cancel_flag) {	
gtp_smoothing_enabled_flag	u(1)
gtp_scale_enabled_flag	u(1)
gtp_offset_enabled_flag	u(1)
gtp_rotation_enabled_flag	u(1)
gtp_point_size_info_enabled_flag	u(1)
gtp_point_shape_info_enabled_flag	u(1)
if (gtp_smoothing_enabled_flag) {	
gtp_smoothing_grid_size_minus2	u(7)
gtp_smoothing_threshold	u(8)
}	
if(gtp_scale_enabled_flag)	
for(d = 0; d < 3; d++)	
gtp_geometry_scale_on_axis[d]	u(32)
if(gtp_offset_enabled_flag)	
for(d = 0; d < 3; d++)	
gtp_geometry_offset_on_axis[d]	i(32)
if(gtp_rotation_enabled_flag) {	
gtp_rotation_x	i(16)

gtp_rotation_y	i(16)
gtp_rotation_z	i(16)
}	
if(gtp_point_size_info_enabled_flag)	
gtp_point_size_info_minus1	u(16)
if(gtp_point_shape_info_enabled_flag)	
gtp_point_shape_info	u(4)
}	
}	

E.2.11 Attribute transformation parameters SEI message syntax

attribute_transformation_params(payloadSize) {	**Descriptor**
atp_cancel_flag	
if(!atp_cancel_flag) {	
atp_num_attribute_updates	ue(v)
for(j = 0; j < atp_num_attribute_updates; j++) {	
atp_attribute_idx[j]	u(8)
atp_dimension_minus1[atp_attribute_idx[j]]	u(8)
for(i = 0; i < atp_dimension_minus1[atp_attribute_idx[j]]; i++) {	
atp_smoothing_params_enabled_flag[atp_attribute_idx[j]][i]	u(1)
atp_scale_params_enabled_flag[atp_attribute_idx[j]][i]	u(1)
atp_offset_params_enabled_flag[atp_attribute_idx[j]] [i]	u(1)
if(atp_smoothing_params_enabled_flag[atp_attribute_idx[j]] [i]) {	
atp_smoothing_grid_size_minus2[atp_attribute_idx[j]] [i]	u(8)
atp_smoothing_threshold[atp_attribute_idx[j]] [i]	u(8)
atp_smoothing_local_entropy_threshold[atp_attribute_idx[j]] [i]	u(3)
atp_smoothing_threshold_variation[atp_attribute_idx[j]] [i]	u(8)
atp_smoothing_threshold_difference[atp_attribute_idx[j]] [i]	u(8)
}	
if(atp_scale_params_enabled_flag**[**atp_attribute_idx[j]][i])	
atp_attribute_scale[atp_attribute_idx[j]] [i]	u(32)
if(atp_offset_params_enabled_flag[atp_attribute_idx[j]][i])	
atp_attribute_offset[atp_attribute_idx[j]] [i]	i(32)
}	
}	
}	
}	

E.2.12 Active substreams SEI message syntax

active_substreams(payloadSize) {	**Descriptor**
active_attributes_changes_flag	u(1)
active_maps_changes_flag	u(1)
raw_points_substreams_active_flag	u(1)
if (active_attributes_changes_flag) {	
all_attributes_active_flag	u(1)
if (!all_attributes_active_flag) {	
active_attribute_count_minus1	u(7)
for (i = 0; i <= active_attribute_count_minus1; i++)	
active_attribute_idx[i]	u(7)
}	
}	
if (active_maps_changes_flag) {	
all_maps_active_flag	u(1)
if (!all_maps_active_flag) {	
active_map_count_minus1	u(4)
for (i = 0; i <= active_map_count_minus1)	
active_map_idx[i]	u(4)
}	
}	
}	

E.2.13 Component codec mapping SEI message syntax

component_codec_mapping(payloadSize) {	**Descriptor**
ccm_codec_mappings_count_minus1	u(8)
for (i=0; i <= codec_mappings_count_minus1; i++) {	
ccm_codec_id	u(8)
ccm_codec_4cc[ccm_codec_id]	st(v)
}	
}	

E.2.14 Volumetric Tiling SEI message syntax

E.2.14.1 General

volumetric_tiling_info(payloadSize) {	**Descriptor**
vti_cancel_flag	u(1)
if (!vti_cancel_flag) {	
vti_object_label_present_flag	u(1)
vti_3d_bounding_box_present_flag	u(1)
vti_object_priority_present_flag	u(1)
vti_object_hidden_present_flag	u(1)
vti_object_collision_shape_present_flag	u(1)
vti_object_dependency_present_flag	u(1)
if(vti_object_label_present_flag)	
volumetric_tiling_info_labels( )	
if (vti_3d_bounding_box_present_flag) {	
vti_bounding_box_scale_log2	u(5)
vti_3d_bounding_box_scale_log2	u(5)
vti_3d_bounding_box_precision_minus8	u(5)
}	
volumetric_tiling_info_objects(vti_object_label_present_flag, vti_3d_bounding_box_present_flag, vti_object_priority_present_flag, vti_object_hidden_present_flag, vti_object_collision_shape_present_flag, vti_object_dependency_present_flag)	
}	
}	

E.2.14.2 Volumetric Tiling Info Labels

volumetric_tiling_info_labels( ) {	**Descriptor**
vti_object_label_language_present_flag	u(1)
if(vti_object_label_language_present_flag) {	
while(!byte_aligned( ))	
vti_bit_equal_to_zero /* equal to 0 */	f(1)
vti_object_label_language	st(v)
}	
vti_num_object_label_updates	ue(v)
for(i = 0; i < vti_num_object_label_updates ; i++) {	
vti_label_idx[i]	ue(v)
vti_label_cancel_flag	u(1)
LabelAssigned[vti_label_idx[i]] = !vti_label_cancel_flag	
if (!vti_label_cancel_flag) {	
while(!byte_aligned( ))	
vti_bit_equal_to_zero /* equal to 0 */	f(1)
vti_label[vti_label_idx[i]]	st(v)
}	
}	
}	

E.2.14.3 Volumetric Tiling Info Objects

volumetric_tiling_info_objects(vtiObjectLabelPresentFlag, vti3dBoundingBoxPresentFlag, vtiObjectPriorityPresentFlag, vtiObjectHiddenPresentFlag, , vtiObjectCollisionShapePresentFlag, vtiObjectDependencyPresentFlag) {	
vti_num_object_updates	ue(v)
for(i = 0; i <= vti_num_object_updates; i++) {	
vti_object_idx[i]	ue(v)
vti_object_cancel_flag[vti_object_idx[i]]	u(1)
ObjectTracked[vti_object_idx[i]] = ! vti_object_cancel_flag[vti_object_idx[i]]	
if (!vti_object_cancel_flag[vti_object_idx[i]]) {	
vti_bounding_box_update_flag[vti_object_idx[i]]	u(1)
if(vti_bounding_box_update_flag[vti_object_idx[i]]) {	
vti_bounding_box_top[vti_object_idx[i]]	u(v)
vti_bounding_box_left[vti_object_idx[i]]	u(v)
vti_bounding_box_width[vti_object_idx[i]]	u(v)
vti_bounding_box_height[vti_object_idx[i]]	u(v)
}	
if(vti3dBoundingBoxPresentFlag) {	
vti_3d_bounding_box_update_flag[vti_object_idx[i]]	u(1)
if(vti_3d_bounding_box_update_flag[vti_object_idx[i]]) {	
vti_3d_bounding_box_x[vti_object_idx[i]]	u(v)
vti_3d_bounding_box_y[vti_object_idx[i]]	u(v)
vti_3d_bounding_box_z[vti_object_idx[i]]	u(v)
vti_3d_bounding_box_delta_x[vti_object_idx[i]]	u(v)
vti_3d_bounding_box_delta_y[vti_object_idx[i]]	u(v)
vti_3d_bounding_box_delta_z[vti_object_idx[i]]	u(v)
}	
}	
if(vtiObjectPriorityPresentFlag) {	
vti_object_priority_update_flag[vti_object_idx[i]]	u(1)
if(vti_object_priority_update_flag[vti_object_idx[i]])	
vti_object_priority_value[vti_object_idx[i]]	u(4)
}	
if(vtiObjectHiddenPresentFlag)	
vti_object_hidden_flag[vti_object_idx[i]]	u(1)
if(vtiObjectLabelPresentFlag) {	
vti_object_label_update_flag[vti_object_idx[i]]	u(1)
if(vti_object_label_update_flag[vti_object_idx[i]])	
vti_object_label_idx[vti_object_idx[i]]	ue(v)
}	
if(vtiObjectCollisionShapePresentFlag) {	
vti_object_collision_shape_update_flag[vti_object_idx[i]]	u(1)
if (vti_object_collision_shape_update_flag[vti_object_idx[i]])	
vti_object_collision_shape_id[vti_object_idx[i]]	u(16)
}	
if(vtiObjectDependencyPresentFlag) {	
vti_object_dependency_update_flag[vti_object_idx[i]]	u(1)
if (vti_object_dependency_update_flag[vti_object_idx[i]]) {	

vti_object_num_dependencies[vti_object_idx[i]]	u(4)
for(j = 0; j < vti_object_num_dependencies[vti_object_idx[i]]; j++)	
vti_object_dependency_idx[vti_object_idx[i]] [j]	u(8)
}	
}	
}	
}	
}	

E.2.15 Buffering period SEI message syntax

buffering_period(payloadSize) {	**Descriptor**
bp_atlas_sequence_parameter_set_id	ue(v)
bp_irap_cab_params_present_flag	u(1)
if(bp_irap_cab_params_present_flag) {	
bp_cab_delay_offset	u(v)
bp_dab_delay_offset	u(v)
}	
bp_concatenation_flag	u(1)
bp_atlas_cab_removal_delay_delta_minus1	u(v)
bp_max_sub_layers_minus1 = 0	u(3)
for(i = 0; i <= bp_max_sub_layers_minus1; i++) {	
if(NalHrdBpPresentFlag) {	
for(j = 0; j < hrd_cab_cnt_minus1[i] + 1; j++) {	
bp_nal_initial_cab_removal_delay[i][j]	u(v)
bp_nal_initial_cab_removal_offset[i][j]	u(v)
if(bp_irap_cab_params_present_flag) {	
bp_nal_initial_alt_cab_removal_delay[i]	u(v)
bp_nal_initial_alt_cab_removal_offset[i]	u(v)
}	
}	
if(AclHrdBpPresentFlag) {	
for(j = 0; j < hrd_cab_cnt_minus1[i] + 1; j++) {	
bp_acl_initial_cab_removal_delay[i][j]	u(v)
bp_acl_initial_cab_removal_offset[i][j]	u(v)
if(bp_irap_cab_params_present_flag) {	
bp_acl_initial_alt_cab_removal_delay[i]	u(v)
bp_acl_initial_alt_cab_removal_offset[i]	u(v)
}	
}	
}	
}	
}	

E.2.16 Atlas frame timing SEI message syntax

atlas_frame_timing(payloadSize) {	Descriptor
if(CabDabDelaysPresentFlag) {	
aft_cab_removal_delay_minus1	u(v)
aft_dab_output_delay	u(v)
}	
}	

E.3 SEI payload semantics

E.3.1 General SEI payload semantics

sp_reserved_payload_extension_data shall not be present in bitstreams conforming to this version of this Specification. However, decoders conforming to this version of this Specification shall ignore the presence and value of sp_reserved_payload_extension_data. When present, the length, in bits, of sp_reserved_payload_extension_data is equal to 8 * payloadSize − nEarlierBits − nPayloadZeroBits − 1, where nEarlierBits is the number of bits in the sei_payload( ) syntax structure that precede the sp_reserved_payload_extension_data syntax element and nPayloadZeroBits is the number of payload_bit_equal_to_zero syntax elements at the end of the sei_payload( ) syntax structure.

NOTE 1 – SEI messages with the same value of payloadType are conceptually the same SEI message regardless of whether they are contained in prefix or suffix SEI atlas data group units.

The semantics and persistence scope for each SEI message are specified in the semantics specification for each particular SEI message.

NOTE 2 – Persistence information for SEI messages is informatively summarized in Table E-11.

Table E-11 — Persistence scope of SEI messages (informative)

SEI message	Persistence scope
Buffering period	The remainder of the bitstream
Atlas frame timing	The access unit containing the SEI message
Filler payload	The access unit containing the SEI message
User data registered by Rec. ITU-T T.35	Unspecified
User data unregistered	Unspecified
Recovery point	Specified by the syntax of the SEI message
Decoded point cloud hash	The access unit containing the SEI message
No display	The access unit containing the SEI message
Time code	The access unit containing the SEI message
Regional nesting	Depending on the region-nested SEI messages; each region-nested SEI message has the same persistence scope as if the SEI message was non-region-nested
Geometry transformation parameters	
Attribute transformation parameters	

The values of some SEI message syntax elements, including regional_nesting_id, are split into two sets of value ranges, where the first set is specified as "may be used as determined by the application", and the second set is specified as "reserved for future use by ISO/IEC". Applications should be cautious of potential “collisions” of the interpretation for values of these syntax elements belonging to the first set of value ranges. Since different applications might use these IDs having values in the first set of value ranges for different purposes, particular care should be exercised in the design of encoders that generate SEI messages with these IDs having values in the first set of value ranges, and in the design of decoders that interpret SEI messages with these IDs. This Specification does not define any management for these values. These IDs having values in the first set of value ranges might only be suitable for use in contexts in which "collisions" of usage (i.e., different definitions of the syntax and semantics of an SEI message with one of these IDs having the same value in the first set of value ranges) are unimportant, or not possible, or are managed – e.g., defined or managed in the controlling application or transport specification, or by controlling the environment in which bitstreams are distributed.

In the following subclauses of this annex, the following applies:

– The current SEI message refers to the particular SEI message.

– The current access unit refers to the access unit containing the current SEI message.

In the following subclauses of this annex, when a particular SEI message applies to a set of one or more maps (instead of a set of operation points), i.e., when the payloadType value is not equal to one of 0 (buffering period) and 1 (atlas frame timing), the following applies:

– The semantics apply independently to each unit the particular SEI message applies.

In the following subclauses of this annex, when a particular SEI message applies to a set of one or more operation points (instead of a set of one or more maps), i.e., when the payloadType value is equal to 0 (buffering period) or 1 (atlas frame timing), the following applies:

– The semantics apply independently to each particular operation point of the set of operation points to which the particular SEI message applies.

– The current operation point refers to the particular operation point.

– The terms "access unit" and "CPCS" apply to the bitstream BitstreamToDecode that is the sub-bitstream of the particular operation point.

E.3.2 Filler payload SEI message semantics

This SEI message contains a series of payloadSize bytes of value 0xFF, which can be discarded.

ff_byte shall be a byte having the value 0xFF.

E.3.3 User data registered by Recommendation ITU-T T.35 SEI message semantics

This SEI message contains user data registered as specified in Recommendation ITU-T T.35, the contents of which are not specified in this Specification.

itu_t_t35_country_code shall be a byte having a value specified as a country code by Annex A of Recommendation ITU-T T.35.

itu_t_t35_country_code_extension_byte shall be a byte having a value specified as a country code by Annex B of Recommendation ITU-T T.35.

itu_t_t35_payload_byte shall be a byte containing data registered as specified in Recommendation ITU-T T.35.

The ITU-T T.35 terminal provider code and terminal provider oriented code shall be contained in the first one or more bytes of the itu_t_t35_payload_byte, in the format specified by the Administration that issued the terminal provider code. Any remaining itu_t_t35_payload_byte data shall be data having syntax and semantics as specified by the entity identified by the ITU-T T.35 country code and terminal provider code.

E.3.4 User data unregistered SEI message semantics

This SEI message contains unregistered user data identified by a universal unique identifier (UUID), the contents of which are not specified in this Specification.

uuid_iso_iec_11578 shall have a value specified as a UUID according to the procedures of Annex A of ISO/IEC 11578:1996.

user_data_payload_byte shall be a byte containing data having syntax and semantics as specified by the UUID generator.

E.3.5 Recovery point SEI message semantics

The recovery point SEI message assists a decoder in determining when the decoding process will produce acceptable point cloud frames for reconstruction and display after the decoder initiates random access or after the encoder indicates a broken link in the CPCS. When the decoding process is started with the access unit in decoding order associated with the recovery point SEI message, all decoded point cloud frames at or subsequent to the recovery point in output order specified in this SEI message are indicated to be correct or approximately correct in content. Decoded point cloud frames produced by random access at or before the point cloud frame associated with the recovery point SEI message need not be correct in content until the indicated recovery point, and the operation of the decoding process starting at the point cloud frame associated with the recovery point SEI message may contain references to point cloud frames and related video data that might be unavailable for prediction.

In addition, by use of the broken_link_flag, the recovery point SEI message can indicate to the decoder the location of some point cloud frames in the bitstream that can result in serious visual artefacts when reconstructed and displayed, because of potentially missing references.

NOTE 1 – The broken_link_flag can be used by encoders to indicate the location of a point after which the decoding process for the decoding of some point cloud frames may cause references to information that, though available for use in the decoding process, is not the information that was used for reference when the bitstream was originally encoded (e.g., due to a splicing operation performed during the generation of the bitstream).

When random access is performed to start decoding from the access unit associated with the recovery point SEI message, the decoder operates as if the associated point cloud frame was the first point cloud frame in the bitstream in decoding order, and the variables prevAtlasFrmOrderCntLsb and prevAtlasFrmOrderCntMsb used in derivation of AtlasFrmOrderCntValare both set equal to 0.

NOTE 2 – When HRD information is present in the bitstream, a buffering period SEI message should be associated with the access unit associated with the recovery point SEI message in order to establish initialization of the HRD buffer model after a random access.

recovery_afoc_cnt specifies the recovery point of decoded point cloud frames in output order. If there is a point cloud frame pcfA that follows the current point cloud frame (i.e., the point cloud frame associated with the current SEI message) in decoding order in the CPCS and that has AtlasFrmOrderCntValequal to the AtlasFrmOrderCntValof the current point cloud frame plus the value of recovery_afoc_cnt, the point cloud frame pcfA is referred to as the recovery point point cloud frame. Otherwise, the first point cloud frame in output order that has AtlasFrmOrderCntValgreater than the AtlasFrmOrderCntValof the current point cloud frame plus the value of recovery_afoc_cnt is referred to as the recovery point point cloud frame. The recovery point point cloud frame shall not precede the current point cloud frame in decoding order. All decoded point cloud frames in output order are indicated to be correct or approximately correct in content starting at the output order position of the recovery point point cloud frame. The value of recovery_afoc_cnt shall be in the range of −MaxAtlasFrmOrderCntLsb / 2 to MaxAtlasFrmOrderCntLsb / 2 − 1, inclusive.

exact_match_flag indicates whether decoded point cloud frames at and subsequent to the specified recovery point in output order derived by starting the decoding process at the access unit associated with the recovery point SEI message will be an exact match to the point cloud frames that would be produced by starting the decoding process at the location of a previous IRAP access unit, if any, in the bitstream. The value 0 indicates that the match may not be exact and the value 1 indicates that the match will be exact. When exact_match_flag is equal to 1, it is a requirement of bitstream conformance that the decoded point cloud frames at and subsequent to the specified recovery point in output order derived by starting the decoding process at the access unit associated with the recovery point SEI message shall be an exact match to the point cloud frames that would be produced by starting the decoding process at the location of a previous IRAP access unit, if any, in the bitstream.

When exact_match_flag is equal to 0, the quality of the approximation at the recovery point is chosen by the encoding process and is not specified in this Specification.

broken_link_flag indicates the presence or absence of a broken link in the atlas data group unit stream at the location of the recovery point SEI message and is assigned further semantics as follows:

– If broken_link_flag is equal to 1, point cloud frames produced by starting the decoding process at the location of a previous IRAP access unit may contain undesirable visual artefacts to the extent that decoded point cloud frames at and subsequent to the access unit associated with the recovery point SEI message in decoding order should not be reconstructed and displayed until the specified recovery point in output order.

– Otherwise (broken_link_flag is equal to 0), no indication is given regarding any potential presence of visual artefacts.

Regardless of the value of the broken_link_flag, point cloud frames subsequent to the specified recovery point in output order are specified to be correct or approximately correct in content.

E.3.6 No display SEI message semantics

The no display SEI message indicates that the current point cloud frame should not be displayed.

E.3.7 Reserved SEI message semantics

The reserved SEI message consists of data reserved for future backward-compatible use by ISO/IEC. It is a requirement of bitstream conformance that bitstreams shall not contain reserved SEI messages until and unless the use of such messages has been specified by ISO/IEC. Decoders shall ignore reserved SEI messages.

E.3.8 SEI manifest SEI message semantics

The SEI manifest SEI message conveys information on SEI messages that are indicated as expected (i.e., likely) to be present or not present. Such information may include:

1) The indication that certain types of SEI messages are expected (i.e., likely) to be present (although not guaranteed to be present) in the CPCS.

2) For each type of SEI message that is indicated as expected (i.e., likely) to be present in the CPCS, the degree of expressed necessity of interpretation of the SEI messages of this type.

   The degree of necessity of interpretation of an SEI message type may be indicated as "necessary", "unnecessary", or "undetermined".

   An SEI message is indicated by the encoder (i.e., the content producer) as being "necessary" when the information conveyed by the SEI message is considered as necessary for interpretation by the decoder or receiving system in order to properly process the content and enable an adequate user experience; it does not mean that the bitstream is required to contain the SEI message in order to be a conforming bitstream. It is at the discretion of the encoder to determine which SEI messages are to be considered as necessary in a particular CPCS. However, it is suggested that some SEI messages should typically be considered as necessary.

3) The indication that certain types of SEI messages are expected (i.e., likely) not to be present (although not guaranteed not to be present) in the CPCS.

The content of an SEI manifest SEI message may, for example, be used by transport-layer or systems-layer processing elements to determine whether the CPCS is suitable for delivery to a receiving and decoding system, based on whether the receiving system can properly process the CPCS to enable an adequate user experience or whether the CPCS satisfies the application needs.

When an SEI manifest SEI message is present in any access unit of a CPCS, an SEI manifest SEI message shall be present in the first access unit of the CPCS. The SEI manifest SEI message persists in decoding order from the current access unit until the end of the CPCS. When there are multiple SEI manifest SEI messages present in a CPCS, they shall have the same content.

manifest_num_sei_msg_types specifies the number of types of SEI messages for which information is provided in the SEI manifest SEI message.

manifest_sei_payload_type[i] indicates the payloadType value of the i-th type of SEI message for which information is provided in the SEI manifest SEI message. The values of manifest_sei_payload_type[m] and manifest_sei_payload_type[n] shall not be identical when m is not equal to n.

manifest_sei_description[i] provides information on SEI messages with payloadType equal to manifest_sei_payload_type[i] as specified in Table E-12.

Table E-12 — manifest_sei_description[i] values

Value	Description
0	Indicates that there is no SEI message with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CPCS.
1	Indicates that there are SEI messages with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CPCS, and these SEI messages are considered as necessary.
2	Indicates that there are SEI messages with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CPCS, and these SEI messages are considered as unnecessary.
3	Indicates that there are SEI messages with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CPCS, and the necessity of these SEI messages is undetermined.
4-255	Reserved

The value of manifest_sei_description[i] shall be in the range of 0 to 3, inclusive, in bitstreams conforming to this version of this Specification. Other values for manifest_sei_description[i] are reserved for future use by ISO/IEC. Decoders shall allow the value of manifest_sei_description[i] greater than or equal to 4 to appear in the syntax and shall ignore all information for payloadType equal to manifest_sei_payload_type[i] signalled in the SEI manifest SEI message and shall ignore all SEI prefix indication SEI messages with prefix_sei_payload_type equal to manifest_sei_payload_type[i] when manifest_sei_description[i] is greater than or equal to 4.

E.3.9 SEI prefix indication SEI message semantics

The SEI prefix indication SEI message carries one or more SEI prefix indications for SEI messages of a particular value of payloadType. Each SEI prefix indication is a bit string that follows the SEI payload syntax of that value of payloadType and contains a number of complete syntax elements starting from the first syntax element in the SEI payload.

Each SEI prefix indication for an SEI message of a particular value of payloadType indicates that one or more SEI messages of this value of payloadType are expected (i.e., likely) to be present in the CPCS and to start with the provided bit string. A starting bit string would typically contain only a true subset of an SEI payload of the type of SEI message indicated by the payloadType, may contain a complete SEI payload, and shall not contain more than a complete SEI payload. It is not prohibited for SEI messages of the indicated value of payloadType to be present that do not start with any of the indicated bit strings.

These SEI prefix indications should provide sufficient information for indicating what type of processing is needed or what type of content is included. The former (type of processing) indicates decoder-side processing capability, e.g., whether some type of post-filtering process is needed. The latter (type of content) indicates, for example, whether the bitstream contains subtitle captions in a particular language.

The content of an SEI prefix indication SEI message may, for example, be used by transport-layer or systems-layer processing elements to determine whether the CPCS is suitable for delivery to a receiving and decoding system, based on whether the receiving system can properly process the CPCS to enable an adequate user experience or whether the CPCS satisfies the application needs (as determined in some manner by external means outside the scope of this Specification).

In one example, for user data registered SEI messages that are used to carry captioning information, an SEI prefix indication should include up to at least the language code; and for user data unregistered SEI messages extended for private use, an SEI prefix indication should include up to at least the UUID.

When an SEI prefix indication SEI message is present in any access unit of a CPCS, an SEI prefix indication SEI message shall be present in the first access unit of the CPCS. The SEI prefix indication SEI message persists in decoding order from the current access unit until the end of the CPCS. When there are multiple SEI prefix indication SEI messages present in a CPCS for a particular value of payloadType, they shall have the same content.

prefix_sei_payload_type indicates the payloadType value of the SEI messages for which one or more SEI prefix indications are provided in the SEI prefix indication SEI message. When an SEI manifest SEI message is also present for the CPCS, the value of prefix_sei_payload_type shall be equal to one of the manifest_sei_payload_type[m] values for which manifest_sei_description[m] is equal to 1 to 3, inclusive, as indicated by an SEI manifest SEI message that applies to the CPCS.

num_sei_prefix_indications_minus1 plus 1 specifies the number of SEI prefix indications.

num_bits_in_prefix_indication_minus1[i] plus 1 specifies the number of bits in the i-th SEI prefix indication.

sei_prefix_data_bit[i][j] specifies the j-th bit of the i-th SEI prefix indication.

The bits sei_prefix_data_bit[i][j] for j ranging from 0 to num_bits_in_prefix_indication_minus1[i], inclusive, follow the syntax of the SEI payload with payloadType equal to prefix_sei_payload_type, and contain a number of complete syntax elements starting from the first syntax element in the SEI payload syntax, and may or may not contain all the syntax elements in the SEI payload syntax. The last bit of these bits (i.e., the bit sei_prefix_data_bit[i][num_bits_in_prefix_indication_minus1[i]]) shall be the last bit of a syntax element in the SEI payload syntax, unless it is a bit within an itu_t_t35_payload_byte or user_data_payload_byte.

NOTE – The exception for itu_t_t35_payload_byte and user_data_payload_byte is provided because these syntax elements may contain externally-specified syntax elements, and the determination of the boundaries of such externally-specified syntax elements is a matter outside the scope of this Specification.

byte_alignment_bit_equal_to_one shall be equal to 1.

E.3.10 Geometry transformation parameters SEI message semantics

This SEI message provides information that relate to geometry processing and interpretation during reconstruction.

The persistence scope for this SEI message is the remainder of the bitstream (i.e., the signaled geometry transformation parameters SEI message persists until the end of the stream) or when a new geometry transformation parameters SEI message is encountered.

gtp_cancel_flag equal to 1 indicates that the geometry transformation parameters SEI message cancels the persistence of any previous geometry transformation parameters SEI message in output order that applies to the current map. gtp_cancel_flag equal to 0 indicates that geometry transformation parameters information follows.

gtp_smoothing_enabled_flag equal to 1 specifies that the geometry smoothing is applied to the decoded geometry in the reconstruction process. gtp_smoothing_enabled_flag equal to 0 specifies the geometry smoothing is not applied in the reconstruction process. When not present, the value of gtp_smoothing_enabled_flag is inferred to be equal to 0.

gtp_scale_enabled_flag equal to 1 indicates that geometry scale parameters are present. gtp_scale_enabled_flag equal to 0 indicates that geometry scale parameters are not present. When gtp_scale_enabled_flag is not present, it shall be inferred to be equal to 0.

gtp_offset_enabled_flag equal to 1 indicates that geometry offset parameters are present. gtp_offset_enabled_flag equal to 0 indicates that geometry offset parameters are not present. When gtp_offset_enabled_flag is not present, it shall be inferred to be equal to 0.

gtp_rotation_enabled_flag equal to 1 indicates that geometry rotation parameters are present. gtp_rotation_enabled_flag equal to 0 indicates that geometry rotation parameters are not present. When gtp_rotation_enabled_flag is not present, it shall be inferred to be equal to 0.

gtp_point_size_info_enabled_flag equal to 1 indicates that geometry point size information is present. gtp_point_size_info_enabled_flag equal to 0 indicates that geometry point size information is not present. When gtp_point_size_info_enabled_flag is not present, it shall be inferred to be equal to 0.

gtp_point_shape_info_enabled_flag equal to 1 indicates that geometry point shape information is present. gtp_point_shape_info_enabled_flag equal to 0 indicates that geometry point shape information is not present. When gtp_point_shape_info_enabled_flag is not present, it shall be inferred to be equal to 0.

gtp_smoothing_grid_size_minus2 specifies the value of the variable GeometrySmoothingGridSize used for the geometry smoothing. The value of gtp_geometry_smoothing_grid_size shall be in the range of 0 to 126, inclusive. When not present, the value of gtp_smoothing_grid_size_minus2 is inferred to be equal to 0. The value of GeometrySmoothingGridSize is computed as follows:

GeometrySmoothingGridSize = gtp_smoothing_grid_size_minus2 + 2

gtp_smoothing_threshold indicates the smoothing threshold. The value of gtp_smoothing_threshold shall be in the range of 0 to 255, inclusive. When not present, the value of gtp_smoothing_threshold shall be inferred to be equal to 0.

gtp_geometry_scale_on_axis[d] indicates the value of the scale along the d axis. The value of gtp_geometry_scale_on_axis[d], shall be in the range of 0 to $2^{32} - 1$, inclusive, where d is in the range of 0 to 2, inclusive. The values of d equal to 0, 1, and 2 correspond to the X, Y, and Z axis, respectively. When gtp_geometry_scale_on_axis[d] is not present, it shall be inferred to be equal to 1.

gtp_geometry_offset_on_axis[d] indicates the value of the offset along the d axis. The value of gtp_geometry_offset_on_axis[d] shall be in the range of -2^{31} to $2^{31} - 1$, inclusive, where d is in the range of 0 to 2, inclusive. The values of d equal to 0, 1, and 2 correspond to the X, Y, and Z axis, respectively. When gtp_geometry_offset_on_axis[d] is not present, it shall be inferred to be equal to 0.

gtp_rotation_x specifies the x component, RotationX, for the geometry rotation of the current point cloud image using the quaternion representation. The value of gtp_rotation_x shall be in the range of -2^{15} to $2^{15} - 1$, inclusive. When gtp_rotation_x is not present, its value shall be inferred to be equal to 0. The value of RotationX is computed as follows:

RotationX = gtp_rotation_x ÷ 2^{15}

gtp_rotation_y specifies the y component, RotationY, for the geometry rotation of the current point cloud image using the quaternion representation. The value of gtp_rotation_y shall be in the range of -2^{15} to 2^{15}

$- 1$, inclusive. When gtp_rotation_y is not present, its value shall be inferred to be equal to 0. The value of RotationY is computed as follows:

$$\text{RotationY} = \text{gtp\_rotation\_y} \div 2^{15}$$

gtp_rotation_z specifies the z component, RotationZ, for the geometry rotation of the current point cloud image using the quaternion representation. The value of gtp_rotation_z shall be in the range of -2^{15} to $2^{15} - 1$, inclusive. When gtp_rotation_z is not present, its value shall be inferred to be equal to 0. The value of RotationZ is computed as follows:

$$\text{RotationZ} = \text{gtp\_rotation\_z} \div 2^{15}$$

The fourth component, RotationW, for the geometry rotation of the current point cloud image using the quaternion representation is calculated as follows:

$$\text{RotationW} = \text{Sqrt}(\ 1 - (\ \text{RotationX}^2 + \text{RotationY}^2 + \text{RotationZ}^2\ )\ )$$

gtp_point_size_info_minus1 plus 1 indicates the geometry point size to be used for rendering. The value of gtp_point_size_info_minus1 shall be in the range of 0 to 65535, inclusive. When gtp_point_size_info_minus1 is not present, it shall be inferred to be equal to 0.

gtp_point_shape_info indicates the geometry point shape to be used for rendering. The value of gtp_point_shape_info shall be in the range of 0 to 15, inclusive. When gtp_point_shape_info is not present, it shall be inferred to be equal to 0.

E.3.11 Attribute transformation parameters SEI message semantics

This SEI message provides information that relate to attribute processing and interpretation during reconstruction.

The persistence scope for this SEI message is the remainder of the bitstream (i.e., the signaled attribute transformation parameters SEI message persists until the end of the stream) or when a new attribute transformation parameters SEI message is encountered.

atp_cancel_flag equal to 1 indicates that the attribute transformation parameters SEI message cancels the persistence of any previous attribute transformation parameters SEI message in output order that applies to the current map. atp_cancel_flag equal to 0 indicates that attribute transformation parameters information follows.

atp_num_attribute_updates indicates the number of attributes that is to be updated by the current SEI message.

atp_attribute_idx[j] indicates the attribute index of the j-th attribute that is to be updated by the current SEI message.

atp_dimension_minus1[j] plus 1 indicates the number of dimensions that are expected to be associated with the attribute with index j in the current bitstream.

atp_smoothing_params_enabled_flag[j][i] equal to 1 indicates that attribute smoothing parameters are present in the current atlas tile group attribute parameter set. atp_smoothing_params_enabled_flag[j][i] equal to 0 indicates that attribute smoothing parameters are not present in the current atlas tile group attribute parameter set. When atp_smoothing_params_enabled_flag[j][i] is not present, it shall be inferred to be equal to 0.

atp_scale_params_enabled_flag[j][i] equal to 1 indicates that attribute scale parameters are present in the current atlas tile group attribute parameter set. atp_scale_params_enabled_flag equal to 0 indicates that attribute scale parameters are not present in the current atlas tile group attribute parameter set. When atp_scale_params_enabled_flag[j][i] is not present, it shall be inferred to be equal to 0.

atp_offset_params_enabled_flag[j][i] equal to 1 indicates that attribute offset parameters are present in the current atlas tile group attribute parameter set. atp_offset_params_enabled_flag equal to 0 indicates that attribute offset parameters are not present in the current atlas tile group attribute parameter set. When atp_offset_params_enabled_flag[j][i] is not present, it shall be inferred to be equal to 0.

atp_smoothing_grid_size_minus2[k][j][i] specifies the value of the variable AttributeSmoothingGridSize[k][j][i] used for the attribute smoothing. The value of atp_smoothing_grid_size_minus2[k][j][i] shall be in the range of 0 to 126, inclusive. When not present, the value of atp_smoothing_grid_size_minus2[k][j][i] is inferred to be equal to 0. The value of AttributeSmoothingGridSize[k][j][i] is computed as follows:

AttributeSmoothingGridSize[k][j][i] = atp_smoothing_grid_size_minus2[k][j][i] + 2

atp_smoothing_threshold[k][j][i] indicates the attribute smoothing threshold. The value of atp_smoothing_threshold[k][j][i] shall be in the range of 0 to 255, inclusive. When atp_smoothing_threshold[k][j][i] is not present, it shall be inferred to be equal to 0.

atp_smoothing_local_entropy_threshold[k][j][i] indicates the local entropy threshold in the neighbourhood of a boundary point. The value of atp_smoothing_local_entropy_threshold[k][j][i] shall be in the range of 0 to 7, inclusive. When atp_smoothing_local_entropy_threshold[k][j][i] is not present, it shall be inferred to be equal to 0.

atp_smoothing_threshold_attribute_variation[k][j][i] indicates the threshold of attribute variation for the attribute smoothing. The value of atp_smoothing_threshold_attribute_variation[k][j][i] shall be in the range of 0 to 255, inclusive. When atp_smoothing_threshold_attribute_variation[k][j][i] is not present, it shall be inferred to be equal to 255.

atp_smoothing_threshold_attribute_difference[k][j][i] indicates the threshold of attribute difference for the attribute smoothing. The value of atp_smoothing_threshold_attribute_difference[k][j][i] shall be in the range of 0 to 255, inclusive. When atp_smoothing_threshold_attribute_difference[k][j][i] is not present, it shall be inferred to be equal to 255.

atp_attribute_scale[j][i] indicates the value of the scale to be applied to the values of the i-th dimension of an attribute with index j. The value of atp_attribute_scale[j][i] shall be in the range of 0 to $2^{32} - 1$, inclusive**.** When atp_attribute_scale[j][i] is not present, it shall be inferred to be equal to 0.

atp_attribute_offset[j][i] indicates the value of the offset to be added to the values of the i-th dimension of an attribute with index j. The value of atp_attribute_offset[j][i] shall be in the range of -2^{31} to $2^{31} - 1$, inclusive**.** When atp_attribute_offset[j][i] is not present, it shall be inferred to be equal to 0.

E.3.12 Active substreams SEI message semantics

This SEI message informs the V-PCC decoder which attributes and/or maps are not available and can therefore be skipped in the decoding process. Similarly, it can also inform the decoder that raw points are not available. This is done by signaling changes to active attributes/maps and whether raw points are active or not. An active substreams SEI message references a specific V-PCC sequence parameter set. When only a sub-set of the attributes/maps shall be active, the SEI message contains the attribute/map indices for these attributes/maps. A V-PCC decoder shall consider any other attributes/maps in the referenced VPS that are not listed in the SEI message as inactive and skip them in the decoding and/or rendering process.

The persistence scope for this SEI message is the remainder of the bitstream (i.e., the signaled active attributes persist until the end of the stream) or when a new active attributes SEI message is encountered. This SEI message should not be ignored by the decoder.

The semantics of the fields of the active substreams SEI message are as follows:

active_attributes_changes_flag indicates whether there are activation changes for any of the attributes of the V-PCC stream. Value 1 indicates an activation change for at least one attribute. Value 0 indicates no changes.

active_maps_changes_flag indicates whether there are activation changes for any of the V-PCC maps. Value 1 indicates an activation change for at least one map. Value 0 indicates no changes.

raw_points_substreams_active_flag indicates whether the raw points substreams for the geometry and all attributes are active or not. Value 1 indicates that such substreams are active.

all_attributes_active_flag indicates whether all the attributes signaled in the referenced VPS shall be active. Value 1 indicates that all attributes shall be active. Value 0 indicates that only a sub-set of the attributes shall be active.

active_attribute_count_minus1 plus 1 indicates the number of active attributes signaled in the active substreams SEI message.

active_attribute_idx[i] indicates the attribute index in the V-PCC VPS for the active attribute at index i in the associated SEI message.

all_maps_active_flag indicates whether all the maps signaled in the referenced VPS shall be active. Value 1 indicates that all maps shall be active. Value 0 indicates that only a sub-set of the maps shall be active.

active_map_count_minus1 plus 1 indicates the number of active maps signaled in the active substreams SEI message.

active_map_idx[i] indicates the map index in the V-PCC VPS for the active map at index i in the associated SEI message.

E.3.13 Component codec mapping SEI message semantics

This SEI message informs the V-PCC decoder of the codec mapping for the codec ids of the component substreams signaled in the V-PCC VPS. Each component substream codec id is mapped to a specific codec index in a codec lookup table. The codec ids for the component substreams in the V-PCC VPS shall be unique. The component codec mapping SEI message shall be used to signal the initial codec mapping to the decoder at the beginning of the V-PCC bitstream as well as signalling updated mappings when the codec of one or more of the V-PCC components substreams changes. A V-PCC decoder receiving a component codec mapping SEI message should instantiate new video decoders for the respective component substreams signaled in the message.

The persistence scope for this SEI message is the remainder of the bitstream (i.e., the codec changes for the signaled components persist until the end of the stream) or until a new component codec change SEI message is encountered. Only the codec mapping for codec ids specified in the SEI message shall be updated. Previously defined mappings for other codec ids from an earlier SEI message shall persist if not modified. This SEI message shall not be ignored by the decoder.

When a component codec mapping SEI message is present in any access unit of a CPCS, a component codec mapping SEI message shall be present in the first access unit of the CPCS. The component codec mapping SEI message persists in decoding order from the current access unit until the end of the CPCS.

When there are multiple component codec mapping SEI messages present in a CPCS, they shall have the same content.

The semantics of the fields of the component codec mapping SEI message are as follows:

ccm_codec_mappings_count_minus1 plus 1 indicates the number of codec mappings that are listed in this SEI message.

ccm_codec_id is the codec id that is to be mapped to a particular 4CC codec. This codec id may be associated with one or more of the substreams in a V-PCC bitstream, as specified within the active V-PCC VPS.

ccm_codec_4cc[j] is the four-character code (4CC) for the codec mapped to the codec id of value j. The codec code shall be an MP4RA registered code.

E.3.14 Volumetric tiling information SEI message semantics

This SEI message informs a V-PCC decoder aboid different characteristics of a decoded point cloud, including correspondence of areas within a 2D atlas and the 3D space, relationship and labeling of areas and association with objects.

The persistence scope for this SEI message is the remainder of the bitstream or until a new volumetric tiling SEI message is encountered. Only the corresponding parameters specified in the SEI message shall be updated. Previously defined parameters from an earlier SEI message shall persist if not modified and if the value of vti_cancel_flag is not equal to 1.

vti_cancel_flag equal to 1 indicates that the volumetric tiling information SEI message cancels the persistence of any previous volumetric tiling information SEI message in output order. vti_cancel_flag equal to 0 indicates that volumetric tiling information follows.

vti_object_label_present_flag equal to 1 indicates that object label information is present in the current volumetric tiling information SEI message. vti_object_label_present_flag equal to 0 indicates that object label information is not present.

vti_3d_bounding_box_present_flag equal to 1 indicates that 3D bounding box information is present in the current volumetric tiling information SEI message. vti_3d_bounding_box_present_flag equal to 0 indicates that 3D bounding box information is not present.

vti_object_priority_present_flag equal to 1 indicates that object priority information is present in the current volumetric tiling information SEI message. vti_object_priority_present_flag equal to 0 indicates that object priority information is not present.

vti_object_hidden_present_flag equal to 1 indicates that hidden object information is present in the current volumetric tiling information SEI message. vti_object_hidden_present_flag equal to 0 indicates that hidden object information is not present.

vti_object_collision_shape_present_flag equal to 1 indicates that object collision information is present in the current volumetric tiling information SEI message. vti_object_collision_shape_present_flag equal to 0 indicates that object collision shape information is not present.

vti_object_dependency_present_flag equal to 1 indicates that object dependency information is present in the current volumetric tiling information SEI message. vti_object_dependency_present_flag equal to 0 indicates that object dependency information is not present.

vti_object_label_language_present_flag equal to 1 indicates that object label language information is present in the current volumetric tiling information SEI message. vti_object_label_language_present_flag equal to 0 indicates that object label language information is not present.

vti_bit_equal_to_zero shall be equal to 0.

vti_object_label_language contains a language tag as specified by IETF RFC 5646 followed by a null termination byte equal to 0x00. The length of the vti_object_label_language syntax element shall be less than or equal to 255 bytes, not including the null termination byte.

vti_num_object_label_updates indicates the number of object labels that are to be updated by the current SEI.

vti_label_idx[i] indicates the label index of the i-th label to be updated.

vti_label_cancel_flag equal to 1 indicates that the label with index equal to vti_label_idx[i] shall be canceled and set equal to an empty string. vti_label_cancel_flag equal to 0 indicates that the label with index equal to vti_label_idx[i] shall be updated with information that follows this element.

vti_bit_equal_to_zero shall be equal to 0

vti_label[i] indicates the label of the i-th label. The length of the vti_label[i] syntax element shall be less than or equal to 255 bytes, not including the null termination byte.

vti_bounding_box_scale_log2 indicates the scale to be applied to the 2D bounding box parameters that may be specified for an object.

vti_3d_bounding_box_scale_log2 indicates the scale to be applied to the 3D bounding box parameters that may be specified for an object.

vti_3d_bounding_box_precision_minus8 plus 8 indicates the precision of the 3D bounding box parameters that may be specified for an object.

vti_num_object_updates indicates the number of objects that are to be updated by the current SEI.

vti_object_idx[i] indicates the object index of the i-th object to be updated.

vti_object_cancel_flag[i] equal to 1 indicates that the object with index equal to i shall be canceled and that the variable ObjectTracked[i] shal be set to 0. Its 2D and 3D bounding box parameters shall also be set equal to 0. vti_object_cancel_flag equal to 0 indicates that the object with index equal to vti_object_idx[i] shall be updated with information that follows this element and that that the variable ObjectTracked[i] shal be set to 1.

vti_bounding_box_update_flag[i]equal to 1 indicates that 2D bounding box information is present for object with object index i. vti_bounding_box_update_flag[i] equal to 0 indicates that 2D bounding box information is not present.

vti_bounding_box_top[i] indicates the vertical coordinate value of the top-left position of the bounding box of an object with index i within the current atlas frame.

vti_bounding_box_left[i] indicates the horizontal coordinate value of the top-left position of the bounding box of an object with index i within the current atlas frame.

vti_bounding_box_width[i] indicates the width of the bounding box of an object with index i.

vti_bounding_box_height[i] indicates the height of the bounding box of an object with index i.

vti_3d_bounding_box_update_flag[i]equal to 1 indicates that 3D bounding box information is present for object with object index i. vti_3d_bounding_box_update_flag[i] equal to 0 indicates that 3D bounding box information is not present.

vti_3d_bounding_box_x[i] indicates the x coordinate value of the origin position of the 3D bounding box of an object with index i.

vti_3d_bounding_box_y[i] indicates the y coordinate value of the origin position of the 3D bounding box of an object with index i.

vti_3d_bounding_box_z[i] indicates the z coordinate value of the origin position of the 3D bounding box of an object with index i.

vti_3d_bounding_box_delta_x[i] indicates the size of the bounding box on the x axis of an object with index i.

vti_3d_bounding_box_delta_y[i] indicates the size of the bounding box on the y axis of an object with index i.

vti_3d_bounding_box_delta_z[i] indicates the size of the bounding box on the z axis of an object with index i.

vti_object_priority_update_flag[i] equal to 1 indicates that object priority update information is present for object with object index i. vti_object_priority_update_flag[i] equal to 0 indicates that object priority information is not present.

vti_object_priority_value[i] indicates the priority of an object with index i. The lower the priority value, the higher the priority.

vti_object_hidden_flag[i] equal to 1 indicates that the object with index i shall be hidden. vti_object_hidden_flag[i] equal to 0 indicates that the object with index i shall become present.

vti_object_label_update_flag equal to 1 indicates that object label update information is present for object with object index i. vti_object_label_update_flag[i] equal to 0 indicates that object label update information is not present.

vti_object_label_idx[i] indicates the label index of an object with index i.

vti_object_collision_shape_update_flag[i] equal to 1 indicates that object collision shape update information is present for object with object index i. vti_object_collision_shape_update_flag[i] equal to 0 indicates that object collision shape update information is not present.

vti_object_collision_shape_id[i] indicates the collision shape id of an object with index i.

vti_object_dependency_update_flag[i] equal to 1 indicates that object dependency update information is present for object with object index i. vti_object_dependency_update_flag[i] equal to 0 indicates that object dependency update information is not present.

vti_object_num_dependencies[i] indicates the number of dependencies of object with index i.

vti_object_dependency_idx[i][j] indicates the index of the j-th object that has a dependency with object with object index i.

E.3.15 Buffering period SEI message semantics

A buffering period SEI message provides initial CAB removal delay and initial CAB removal delay offset information for initialization of the HRD at the position of the associated access unit in decoding order.

The following applies for the buffering period SEI message syntax and semantics:

– The syntax elements hrd_initial_cab_removal_delay_length_minus1, hrd_au_cab_removal_delay_length_minus1, hrd_dab_output_delay_length_minus1, and the variables NalHrdBpPresentFlag and AclHrdBpPresentFlag are found in or derived from syntax elements found in the hrd_parameters( ) syntax structure that is applicable to at least one of the operation points to which the buffering period SEI message applies.

– The variables CabSize[i], BitRate[i] and CabCnt are derived from syntax elements found in the sub_layer_hrd_parameters( ) syntax structure that is applicable to at least one of the operation points to which the buffering period SEI message applies.

– Any two operation points that the buffering period SEI message applies to having different OpTid values tIdA and tIdB indicate that the values of hrd_cab_cnt_minus1[tIdA] and hrd_cab_cnt_minus1[tIdB] coded in the hrd_parameters( ) syntax structure(s) applicable to the two operation points are identical.

– Any two operation points that the buffering period SEI message applies to having different OpLayerIdList values layerIdListA and layerIdListB indicate that the values of nal_hrd_parameters_present_flag and acl_hrd_parameters_present_flag, respectively, for the two hrd_parameters( ) syntax structures applicable to the two operation points are identical.

– The bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with any of the operation points to which the buffering period SEI message applies.

The presence of buffering period SEI messages for an operation point is specified as follows:

– If NalHrdBpPresentFlag is equal to 1 or AclHrdBpPresentFlag is equal to 1, the following applies for each access unit in the CAS:
    – If the access unit is an IRAP access unit, a buffering period SEI message applicable to the operation point shall be associated with the access unit.
    – Otherwise, if both of the following conditions apply, a buffering period SEI message applicable to the operation point may or may not be present for the access unit:
        – The atlas frame has TemporalId equal to 0.
        – The atlas frame is not a RASL, RADL or sub-layer non-reference atlas frame.
    – Otherwise, the access unit shall not be associated with a buffering period SEI message applicable to the operation point.
– Otherwise (NalHrdBpPresentFlag and AclHrdBpPresentFlag are both equal to 0), no access unit in the CAS shall be associated with a buffering period SEI message applicable to the operation point.

NOTE 1 – For some applications, frequent presence of buffering period SEI messages may be desirable (e.g., for random access at an IRAP atlas frame or a non-IRAP atlas frame or for bitstream splicing).

bp_seq_parameter_set_id indicates and shall be equal to the asps_seq_parameter_set_id for the ASPS that is active for the coded atlas frame associated with the buffering period SEI message. The value of bp_seq_parameter_set_id shall be equal to the value of afps_seq_parameter_set_id in the AFPS referenced by the atgh_atlas_frame_parameter_set_id of the atlas tile group headers of the coded atlas associated with the buffering period SEI message. The value of bp_seq_parameter_set_id shall be in the range of 0 to 15, inclusive.

bp_irap_cab_params_present_flag equal to 1 specifies the presence of the bp_acl_initial_alt_cab_removal_delay[i] and bp_acl_initial_alt_cab_removal_offset[i] syntax elements. When not present, the value of bp_irap_cab_params_present_flag is inferred to be equal to 0. When the associated atlas is neither a CRA atlas nor a BLA atlas, the value of bp_irap_cab_params_present_flag shall be equal to 0.

bp_cab_delay_offset specifies an offset to be used in the derivation of the nominal CAB removal times of access units following, in decoding order, the CRA or BLA access unit associated with the buffering period SEI message when the RASL access units associated with the CRA or BLA access unit are not present. The syntax element has a length in bits given by hrd_au_cab_removal_delay_length_minus1 + 1. When not present, the value of bp_cab_delay_offset is inferred to be equal to 0.

bp_dab_delay_offset specifies an offset to be used in the derivation of the DAB output times of the CRA or BLA access unit associated with the buffering period SEI message when the RASL access units associated with the CRA or BLA access unit are not present. The syntax element has a length in bits given by hrd_dab_output_delay_length_minus1 + 1. When not present, the value of bp_dab_delay_offset is inferred to be equal to 0.

When the current atlas is not the first atlas in the bitstream in decoding order, let prevNonDiscardableAtlas be the preceding atlas in decoding order with TemporalId equal to 0 that is not a RASL, RADL or sub-layer non-reference atlas frame.

bp_concatenation_flag indicates, when the current atlas is not the first atlas in the bitstream in decoding order, whether the nominal CAB removal time of the current atlas is determined relative to the nominal CAB removal time of the preceding atlas with a buffering period SEI message or relative to the nominal CAB removal time of the atlas prevNonDiscardablePic.

bp_atlas_cab_removal_delay_delta_minus1 plus 1, when the current atlas is not the first atlas in the bitstream in decoding order, specifies a CAB removal delay increment value relative to the nominal CAB removal time of the atlas prevNonDiscardablePic. This syntax element has a length in bits given by hrd_au_cab_removal_delay_length_minus1 + 1.

When the current atlas contains a buffering period SEI message and bp_concatenation_flag is equal to 0 and the current atlas is not the first atlas in the bitstream in decoding order, it is a requirement of bitstream conformance that the following constraint applies:

- If the atlas prevNonDiscardableAtlas is not associated with a buffering period SEI message, the aft_cab_removal_delay_minus1 of the current atlas shall be equal to the aft_cab_removal_delay_minus1 of prevNonDiscardableAtlas plus bp_atlas_cab_removal_delay_delta_minus1 + 1.
- Otherwise, aft_cab_removal_delay_minus1 shall be equal to bp_atlas_cab_removal_delay_delta_minus1.

NOTE 2 – When the current atlas contains a buffering period SEI message and bp_concatenation_flag is equal to 1, the aft_cab_removal_delay_minus1 for the current atlas is not used. The above-specified constraint can, under some circumstances, make it possible to splice bitstreams (that use suitably-designed referencing structures) by simply changing the value of bp_concatenation_flag from 0 to 1 in the buffering period SEI message for an IRAP atlas at the splicing point. When bp_concatenation_flag is equal to 0, the above-specified constraint enables the decoder to check whether the constraint is satisfied as a way to detect the loss of the atlas prevNonDiscardableAtlas.

bp_max_sub_layers_minus1 plus 1 specifies the maximum number of temporal sub-layers. The value of bp_max_sub_layers_minus1 shall be equal to 0.

bp_nal_initial_cab_removal_delay[i] and **bp_nal_initial_alt_cab_removal_delay**[i] specify the default and the alternative initial CAB removal delays, respectively, for the i-th CAB when the NAL HRD parameters are in use. The syntax elements have a length in bits given by hrd_initial_cab_removal_delay_length_minus1 + 1, and are in units of a 90 kHz clock. The values of the syntax elements shall not be equal to 0 and shall be less than or equal to 90000 * (CabSize[i] ÷ BitRate[i]), the time-equivalent of the CAB size in 90 kHz clock units.

bp_nal_initial_cab_removal_offset[i] and **bp_nal_initial_alt_cab_removal_offset**[i] specify the default and the alternative initial CAB removal offsets, respectively, for the i-th CAB when the NAL HRD parameters are in use. The syntax elements have a length in bits given by hrd_initial_cab_removal_delay_length_minus1 + 1 and are in units of a 90 kHz clock.

Over the entire CAS, the sum of bp_nal_initial_cab_removal_delay[i] and bp_nal_initial_cab_removal_offset[i] shall be constant for each value of i, and the sum of bp_nal_initial_alt_cab_removal_delay[i] and bp_nal_initial_alt_cab_removal_offset[i] shall be constant for each value of i.

bp_acl_initial_cab_removal_delay[i] and **bp_acl_initial_alt_cab_removal_delay**[i] specify the default and the alternative initial CAB removal delays, respectively, for the i-th CAB when the VCL HRD parameters are in use. The syntax elements have a length in bits given by hrd_initial_cab_removal_delay_length_minus1 + 1, and are in units of a 90 kHz clock. The values of the syntax elements shall not be equal to 0 and shall be less than or equal to 90000 * (CabSize[i] ÷ BitRate[i]), the time-equivalent of the CAB size in 90 kHz clock units.

bp_acl_initial_cab_removal_offset[i] and **bp_acl_initial_alt_cab_removal_offset**[i] specify the default and the alternative initial CAB removal offsets, respectively, for the i-th CAB when the VCL HRD parameters are in use. The syntax elements have a length in bits given by hrd_initial_cab_removal_delay_length_minus1 + 1 and are in units of a 90 kHz clock.

Over the entire CAS, the sum of bp_acl_initial_cab_removal_delay[i] and bp_acl_initial_cab_removal_offset[i] shall be constant for each value of i, and the sum of bp_acl_initial_alt_cab_removal_delay[i] and bp_acl_initial_alt_cab_removal_offset[i] shall be constant for each value of i.

NOTE 3 – Encoders are recommended not to include irap_cab_params_present_flag equal to 1 in buffering period SEI messages associated with a CRA or BLA atlas for which at least one of its associated RASL atlass follows one or more of its associated RADL atlass in decoding order.

E.3.16 Atlas frame timing SEI message semantics

The atlas frame timing SEI message provides CAB removal delay and DAB output delay information for the access unit associated with the SEI message.

The following applies for the atlas timing SEI message syntax and semantics:

– The syntax elements and variable hrd_au_cab_removal_delay_length_minus1, hrd_dab_output_delay_length_minus1 and CabDabDelaysPresentFlag are found in or derived from syntax elements found in the hrd_parameters( ) syntax structure that is applicable to at least one of the operation points to which the atlas frame timing SEI message applies.

– The bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with any of the operation points to which the atlas timing SEI message applies.

NOTE 4 – The syntax of the atlas timing SEI message is dependent on the content of the hrd_parameters( ) syntax structures applicable to the operation points to which the atlas timing SEI message applies. These hrd_parameters( ) syntax structures are in the ASPS that are active for the coded atlas associated with the atlas timing SEI message. When the atlas timing SEI message is associated with an IRAP access unit with NoRaslOutputFlag equal to 1, unless it is preceded by a buffering period SEI message within the same access unit, the activation of the ASPS (and, for IRAP atlass with NoRaslOutputFlag equal to 1 that are not the first atlas in the bitstream in decoding order, the determination that the coded atlas is an IRAP NoRaslOutputFlag equal to 1) does not occur until the decoding of the first coded tile group NAL unit of the coded atlas. Since the coded tile group NAL unit of the coded atlas follows the atlas timing SEI message in NAL unit order, there may be cases in which it is necessary for a decoder to store the RBSP containing the atlas timing SEI message until determining the active ASPS for the coded atlas, and then perform the parsing of the atlas timing SEI message.

aft_cab_removal_delay_minus1 plus 1 specifies the number clock ticks between the nominal CAB removal time of the access unit associated with the atlas timing SEI message and the preceding access unit in decoding order that contained a buffering period SEI message. This value is also used to calculate an earliest possible time of arrival of access unit atlas data into the CAB for the HSS. The syntax element is a fixed length code whose length in bits is given by hrd_au_cab_removal_delay_length_minus1 + 1.

NOTE 5 – The value of hrd_au_cab_removal_delay_length_minus1 that determines the length (in bits) of the syntax element aft_cab_removal_delay_minus1 is the value of hrd_au_cab_removal_delay_length_minus1 coded in the ASPS that is active for the coded atlas frame associated with the atlas frame timing SEI message, although aft_cab_removal_delay_minus1 specifies a number of clock ticks relative to the removal time of the preceding atlas access unit containing a buffering period SEI message, which may be an access unit of a different CAS.

The variable AuCabRemovalDelayMsb of the current atlas frame is derived as follows:

– If the current atlas frame is associated with a buffering period SEI message that is applicable to at least one of the operation points to which the atlas frame timing SEI message applies, AuCabRemovalDelayMsb is set equal to 0.
– Otherwise, the following applies:
  – Let maxCabRemovalDelay be equal to $2^{\text{hrd\_au\_cab\_removal\_delay\_length\_minus1 + 1}}$.
  – Let prevAuCabRemovalDelayMinus1 and prevAuCabRemovalDelayMsb be set equal to aft_cab_removal_delay_minus1 and AuCabRemovalDelayMsb, respectively, of the previous atlas frame in decoding order that has TemporalId equal to 0, that is not a RASL, RADL or sub-layer non-reference atlas frame, and that is within the same buffering period as the current atlas frame.
  – AuCabRemovalDelayMsb is derived as follows:

```
if( aft_cab_removal_delay_minus1 <= prevAuCabRemovalDelayMinus1 )
    AuCabRemovalDelayMsb = prevAuCabRemovalDelayMsb +
        maxCabRemovalDelay                                        (E-1)
else
    AuCabRemovalDelayMsb = prevAuCabRemovalDelayMsb
```

The variable AuCabRemovalDelayVal is derived as follows:

```
AuCabRemovalDelayVal = AuCabRemovalDelayMsb +
    aft_cab_removal_delay_minus1 + 1                              (E-2)
```

The value of AuCabRemovalDelayVal shall be in the range of 1 to 2^{32}, inclusive. Within one buffering period, the AuCabRemovalDelayVal values for any two access units shall not be the same.

aft_dab_output_delay is used to compute the DAB output time of the atlas. It specifies how many clock ticks to wait after removal of the access unit from the CAB before the decoded atlas is output from the DAB.

NOTE 6 – A atlas is not removed from the DAB at its output time when it is still marked as "used for short-term reference" or "used for long-term reference".

The length of the syntax element aft_dab_output_delay is given in bits by hrd_dab_output_delay_length_minus1 + 1. When asps_max_dec_atlas_buffering_minus1[minTid] is equal to 0, where minTid is the minimum of the OpTid values of all operation points the atlas timing SEI message applies to, aft_dab_output_delay shall be equal to 0.

The output time derived from the aft_dab_output_delay of any atlas that is output from an output timing conforming decoder shall precede the output time derived from the aft_dab_output_delay of all atlases in any subsequent CAS in decoding order.

The atlas output order established by the values of this syntax element shall be the same order as established by the values of AtlasOrderCntVal.

For atlases that are not output by the "bumping" process because they precede, in decoding order, an IRAP atlas with NoRaslOutputFlag inferred to be equal to 1, the output times derived from aft_dab_output_delay shall be increasing with increasing value of AtlasOrderCntVal relative to all atlases within the same CAS.

Annex F

Volumetric usability information

(This annex forms an integral part of this Recommendation | International Standard.)

F.1 General

This annex specifies syntax and semantics of the VUI parameters of the ASPSs.

VUI parameters are not required for constructing the atlas by the decoding process. Conforming decoders are not required to process this information for output order conformance to this document (see Annex D for the specification of output order conformance). Some VUI parameters are required to check bitstream conformance and for output timing decoder conformance.

In Annex F, specification for presence of VUI parameters is also satisfied when those parameters (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified in this Specification. When present in the bitstream, VUI parameters shall follow the syntax and semantics specified in this annex. When the content of VUI parameters is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the VUI parameters is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

F.2 VUI syntax

F.2.1 VUI parameters syntax

vui_parameters( ) {	**Descriptor**
vui_timing_info_present_flag	u(1)
if(vui_timing_info_present_flag) {	
vui_num_units_in_tick	u(32)
vui_time_scale	u(32)
vui_poc_proportional_to_timing_flag	u(1)
if(vui_poc_proportional_to_timing_flag)	
vui_num_ticks_poc_diff_one_minus1	ue(v)
vui_hrd_parameters_present_flag	u(1)
if(vui_hrd_parameters_present_flag)	
hrd_parameters( )	
}	
}	

F.2.2 HRD parameters syntax

hrd_parameters( ) {	**Descriptor**
hrd_nal_parameters_present_flag	u(1)
hrd_acl_parameters_present_flag	u(1)
if(hrd_nal_parameters_present_flag || hrd_acl_parameters_present_flag){	
hrd_bit_rate_scale	u(4)
hrd_cab_size_scale	u(4)
hrd_initial_cab_removal_delay_length_minus1	u(5)
hrd_au_cab_removal_delay_length_minus1	u(5)

hrd_dab_output_delay_length_minus1	u(5)
}	
maxNumSubLayersMinus1 = 0	
for(i = 0; i <= maxNumSubLayersMinus1; i++) {	
hrd_fixed_atlas_rate_general_flag[i]	u(1)
if(!hrd_fixed_atlas_rate_general_flag[i])	
hrd_fixed_atlas_rate_within_cas_flag[i]	u(1)
if(hrd_fixed_atlas_rate_within_cas_flag[i])	
hrd_elemental_duration_in_tc_minus1[i]	ue(v)
else	
hrd_low_delay_flag[i]	u(1)
if(!hrd_low_delay_flag[i])	
hrd_cab_cnt_minus1[i]	ue(v)
if(hrd_nal_parameters_present_flag)	
hrd_sub_layer_parameters(0, i)	
if(hrd_acl_parameters_present_flag)	
hrd_sub_layer_parameters(1, i)	
}	
}	

F.2.3 Sub-layer HRD parameters syntax

hrd_sub_layer_parameters(type, subLayerId) {	**Descriptor**
for(i = 0; i <= CabCnt; i++) {	
hrd_bit_rate_value_minus1[type][i]	ue(v)
hrd_cab_size_value_minus1[type][i]	ue(v)
hrd_cbr_flag[type][i]	u(1)
}	
}	

F.3 VUI semantics

F.3.1 VUI parameters semantics

vui_timing_info_present_flag equal to 1 specifies that vui_num_units_in_tick, vui_time_scale, vui_poc_proportional_to_timing_flag, and vui_hrd_parameters_present_flag are present in the vui_parameters( ) syntax structure. vui_timing_info_present_flag equal to 0 specifies that vui_num_units_in_tick, vui_time_scale, vui_poc_proportional_to_timing_flag, and vui_hrd_parameters_present_flag are not present in the vui_parameters( ) syntax structure.

vui_num_units_in_tick is the number of time units of a clock operating at the frequency vui_time_scale Hz that corresponds to one increment (called a clock tick) of a clock tick counter. vui_num_units_in_tick shall be greater than 0. A clock tick, in units of seconds, is equal to the quotient of vui_num_units_in_tick divided by vui_time_scale. For example, when the frame rate of an atlas signal is 25 Hz, vui_time_scale may be equal to 27 000 000 and vui_num_units_in_tick may be equal to 1 080 000 and consequently a clock tick may be equal to 0.04 seconds.

vui_time_scale is the number of time units that pass in one second. For example, a time coordinate system that measures time using a 27 MHz clock has a vui_time_scale of 27 000 000. The value of vui_time_scale shall be greater than 0.

vui_poc_proportional_to_timing_flag equal to 1 indicates that the atlas frame order count value for each atlas in the CAS that is not the first atlas in the CAS, in decoding order, is proportional to the output time of the atlas relative to the output time of the first atlas in the CAS. vui_poc_proportional_to_timing_flag equal to 0 indicates that the atlas frame order count value for each atlas in the CAS that is not the first atlas in the CAS, in decoding order, may or may not be proportional to the output time of the atlas relative to the output time of the first atlas in the CAS.

vui_num_ticks_poc_diff_one_minus1 plus 1 specifies the number of clock ticks corresponding to a difference of atlas frame order count values equal to 1. The value of vui_num_ticks_poc_diff_one_minus1 shall be in the range of 0 to $2^{32} - 2$, inclusive.

vui_hrd_parameters_present_flag equal to 1 specifies that the syntax structure hrd_parameters( ) is present in the vui_parameters( ) syntax structure. vui_hrd_parameters_present_flag equal to 0 specifies that the syntax structure hrd_parameters( ) is not present in the vui_parameters( ) syntax structure.

F.3.2 HRD parameters semantics

The hrd_parameters( ) syntax structure provides HRD parameters used in the HRD operations for a layer set. It is a requirement for bitstream conformance to this version of the Specification that only one layer shall be supported, i.e. nal_layer_id and nal_temporal_id shall be equal to 0. When the hrd_parameters( ) syntax structure is included in an ASPS, the layer set to which the hrd_parameters( ) syntax structure applies is the layer set for which the associated layer identifier list contains all nal_layer_id values present in the CAS.

For interpretation of the following semantics, the bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with the layer set to which the hrd_parameters( ) syntax structure applies.

hrd_nal_parameters_present_flag equal to 1 specifies that NAL HRD parameters (pertaining to Type II bitstream conformance) are present in the hrd_parameters( ) syntax structure. hrd_nal_parameters_present_flag equal to 0 specifies that NAL HRD parameters are not present in the hrd_parameters( ) syntax structure.

NOTE 1 – When hrd_nal_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the NAL HRD parameters and all buffering period and atlas timing SEI messages, by some means not specified in this Specification.

The variable NalHrdBpPresentFlag is derived as follows:

– If one or more of the following conditions are true, then the value of NalHrdBpPresentFlag is set equal to 1:

  – hrd_nal_parameters_present_flag is present in the bitstream and is equal to 1.

  – The need for presence of buffering periods for NAL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Specification.

– Otherwise, the value of NalHrdBpPresentFlag is set equal to 0.

hrd_acl_parameters_present_flag equal to 1 specifies that ACL HRD parameters (pertaining to all bitstream conformance types) are present in the hrd_parameters( ) syntax structure. hrd_acl_parameters_present_flag equal to 0 specifies that ACL HRD parameters are not present in the hrd_parameters( ) syntax structure.

NOTE 2 – When hrd_acl_parameters_present_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the ACL HRD parameters and all buffering period and atlas timing SEI messages, by some means not specified in this Specification.

The variable AclHrdBpPresentFlag is derived as follows:

– If one or more of the following conditions are true, then the value of AclHrdBpPresentFlag is set equal to 1:

  – hrd_acl_parameters_present_flag is present in the bitstream and is equal to 1.

  – The need for presence of buffering periods for ACL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this Specification.

– Otherwise, the value of AclHrdBpPresentFlag is set equal to 0.

The variable CabDabDelaysPresentFlag is derived as follows:

– If one or more of the following conditions are true, then the value of CabDabDelaysPresentFlag is set equal to 1:

  – hrd_nal_parameters_present_flag is present in the bitstream and is equal to 1.

  – hrd_acl_parameters_present_flag is present in the bitstream and is equal to 1.

  – The need for presence of CAB and DAB output delays to be present in the bitstream in atlas timing SEI messages is determined by the application, by some means not specified in this Specification.

– Otherwise, the value of CabDabDelaysPresentFlag is set equal to 0.

hrd_initial_cab_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the bp_nal_initial_cab_removal_delay[i], bp_nal_initial_cab_removal_offset[i], bp_acl_initial_cab_removal_delay[i], and bp_acl_initial_cab_removal_offset[i] syntax elements of the buffering period SEI message. When the hrd_initial_cab_removal_delay_length_minus1 syntax element is not present, it is inferred to be equal to 23.

hrd_au_cab_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the bp_cab_delay_offset syntax element in the buffering period SEI message and the aft_cab_removal_delay_minus1 syntax element in the atlas timing SEI message. When the hrd_au_cab_removal_delay_length_minus1 syntax element is not present, it is inferred to be equal to 23.

hrd_dab_output_delay_length_minus1 plus 1 specifies the length, in bits, of the bp_dab_delay_offset syntax element in the buffering period SEI message and the aft_dab_output_delay syntax element in the atlas timing SEI message. When the hrd_dab_output_delay_length_minus1 syntax element is not present, it is inferred to be equal to 23.

hrd_fixed_atlas_rate_general_flag[i] equal to 1 indicates that, when HighestTid is equal to i, the temporal distance between the HRD output times of consecutive atlases in output order is constrained as specified below. hrd_fixed_atlas_rate_general_flag[i] equal to 0 indicates that this constraint may not apply.

When hrd_fixed_atlas_rate_general_flag[i] is not present, it is inferred to be equal to 0.

hrd_fixed_atlas_rate_within_cas_flag[i] equal to 1 indicates that, when HighestTid is equal to i, the temporal distance between the HRD output times of consecutive atlases in output order is constrained as specified below. hrd_fixed_atlas_rate_within_cas_flag[i] equal to 0 indicates that this constraint may not apply.

When hrd_fixed_atlas_rate_general_flag[i] is equal to 1, the value of hrd_fixed_atlas_rate_within_cas_flag[i] is inferred to be equal to 1.

hrd_elemental_duration_in_tc_minus1[i] plus 1 (when present) specifies, when HighestTid is equal to i, the temporal distance, in clock ticks, between the elemental units that specify the HRD output times of consecutive atlases in output order as specified below. The value of hrd_elemental_duration_in_tc_minus1[i] shall be in the range of 0 to 2047, inclusive.

For each atlas n that is output and is not the last atlas in the bitstream (in output order) that is output, the value of the variable DabOutputElementalInterval[n] is specified by:

$$\text{DabOutputElementalInterval}[\ n\ ] = \text{DabOutputInterval}[\ n\ ] \div \text{DeltaToDivisor} \tag{F-1}$$

where DabOutputInterval[n] is specified in Equation D-12 and DeltaToDivisor is equal to 1.

When HighestTid is equal to i and hrd_fixed_atlas_rate_general_flag[i] is equal to 1 for a CAS containing atlas n, the value computed for DabOutputElementalInterval[n] shall be equal to ClockTick * (hrd_elemental_duration_in_tc_minus1[i] + 1), wherein ClockTick is as specified in Equation D-1(using the value of ClockTick for the CAS containing atlas n) when one of the following conditions is true for the following atlas in output order nextAtlasFrameInOutputOrder that is specified for use in Equation D-12 :

– the atlas frame nextAtlasFrameInOutputOrder is in the same CAS as the atlas frame n.

– the atlas frame nextAtlasFrameInOutputOrder is in a different CAS and hrd_fixed_atlas_rate_general_flag[i] is equal to 1 in the CAS containing the atlas frame nextAtlasFrameInOutputOrder, the value of ClockTick is the same for both CASs, and the value of hrd_elemental_duration_in_tc_minus1[i] is the same for both CASs.

When HighestTid is equal to i and hrd_fixed_atlas_rate_within_cas_flag[i] is equal to 1 for a CAS containing atlas frame n, the value computed for DabOutputElementalInterval[n] shall be equal to ClockTick * (hrd_elemental_duration_in_tc_minus1[i] + 1), wherein ClockTick is as specified in Equation D-1 (using the value of ClockTick for the CAS containing atlas frame n) when the following atlas frame in output order nextAtlasFrameInOutputOrder that is specified for use in Equation D-12 is in the same CAS as atlas frame n.

hrd_low_delay_flag[i] specifies the HRD operational mode, when HighestTid is equal to i, as specified in Annex D. When not present, the value of hrd_low_delay_flag[i] is inferred to be equal to 0.

NOTE 3 – When hrd_low_delay_flag[i] is equal to 1, "big atlases" that violate the nominal CAB removal times due to the number of bits used by an access unit are permitted. It is expected, but not required, that such "big atlases" occur only occasionally.

hrd_cab_cnt_minus1[i] plus 1 specifies the number of alternative CAB specifications in the bitstream of the CAS when HighestTid is equal to i. The value of hrd_cab_cnt_minus1[i] shall be in the range of 0 to 31, inclusive. When not present, the value of hrd_cab_cnt_minus1[i] is inferred to be equal to 0.

F.3.3 HRD Sub-layer parameters semantics

The variable CabCnt is set equal to hrd_cab_cnt_minus1[subLayerId].

hrd_bit_rate_value_minus1[j][i] (together with hrd_bit_rate_scale) specifies the maximum input bit rate for the i-th CAB when the CAB operates at the access unit level for a particular HRD type j. hrd_bit_rate_value_minus1[j][i] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any $i > 0$, hrd_bit_rate_value_minus1[j][i] shall be greater than hrd_bit_rate_value_minus1[j][i − 1].

The bit rate in bits per second is given by:

$$\text{BitRate}[\ j\ ][\ i\ ] = (\ \text{hrd\_bit\_rate\_value\_minus1}[\ j\ ][\ i\ ] + 1\ ) * 2^{(\ 6\ +\ \text{hrd\_bit\_rate\_scale}\ )} \quad \text{(F-2)}$$

When the hrd_bit_rate_value_minus1[j][i] syntax element is not present, the value of BitRate[j][i] is inferred to be equal to CalBrAclFactor * MaxBR for ACL HRD parameters, i.e. when j is equal to 0, and equal to CalBrNalFactor * MaxBR for NAL HRD parameters, i.e. when j is equal to 1, where MaxBR, CalBrAclFactor and CalBrNalFactor are specified in Annex A.

hrd_cab_size_value_minus1[j][i] is used together with hrd_cab_size_scale to specify the i-th CAB size when the CAB operates at the access unit level for a particular HRD type j. hrd_cab_size_value_minus1[j][i] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any i greater than 0, hrd_cab_size_value_minus1[j][i] shall be less than or equal to hrd_cab_size_value_minus1[j][i − 1].

The CAB size in bits is given by:

$$\text{CabSize}[\ j\ ][\ i\ ] = (\ \text{hrd\_cab\_size\_value\_minus1}[\ j\ ][\ i\ ] + 1\ ) * 2^{(\ 4\ +\ \text{hrd\_cab\_size\_scale}\ )} \quad \text{(F-3)}$$

When the hrd_cab_size_value_minus1[j][i] syntax element is not present, the value of CabSize[j][i] is inferred to be equal to CalBrAclFactor * MaxCAB for ACL HRD parameters, i.e. when j is equal to 1, and equal to CalBrNalFactor * MaxCAB for NAL HRD parameters, i.e. when j is equal to 0, where MaxCAB, CalBrAclFactor and CalBrNalFactor are specified in Annex A.

hrd_cbr_flag[j][i] equal to 0 specifies that to decode this CAS by the HRD type j using the i-th CAB specification, the hypothetical stream scheduler (HSS) operates in an intermittent bit rate mode. hrd_cbr_flag[j][i] equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode. When not present, the value of hrd_cbr_flag[j][i] is inferred to be equal to 0.

Annex G

Bibliography

[1] ISO/IEC 14496-10: *Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding*.

[2] ISO/IEC 23008-2: *Information technology – High efficiency coding and media delivery in heterogeneous environments – Part 2: High efficiency video coding*.

[3] Registration authority for code-points in "MP4 Family" files: https://mp4ra.org/#