INTERNATIONAL STANDARD

ISO/IEC 23090-5

First edition
2021-06

Information technology — Coded representation of immersive media —

Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC)

Technologie de l'information — Représentation codée de média immersifs —

Partie 5: Codage basé sur la vidéo volumétrique (V3C) et compression de nuage de points basée sur la vidéo (V-PCC)

Reference number
ISO/IEC 23090-5:2021(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

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 document 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 and IEC 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) or the IEC list of patent declarations received (see http://patents.iec.ch).

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 Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 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, of which visual volumetric video is an integral part.

Visual volumetric video, a sequence of visual volumetric frames, if uncompressed, can be represented by a large amount of data, which can be costly in terms of storage and transmission. This has led to the need for a high coding efficiency standard for the compression of visual volumetric data.

Visual volumetric frames can be coded by converting the 3D volumetric information into a collection of 2D images and associated data. The converted 2D images can be coded using widely available video and image coding specifications, such as ISO/IEC 14496-10 and ISO/IEC 23008-2 and the associated data can be coded with mechanisms specified in this document. The coded images and the associated data can then be decoded and used to reconstruct the 3D volumetric information. This document specifies a generic mechanism for visual volumetric video coding, i.e. visual volumetric video-based coding. The generic mechanism can be used by applications targeting volumetric content, such as point clouds, immersive video with depth, mesh representations of visual volumetric frames, etc.

In addition to the generic mechanism of coding volumetric content, this document specifies one of the applications of visual volumetric video-based coding targeting point cloud representations of visual volumetric frames. In a point cloud sequence, each point cloud frame contains a collection of points. Each point has a 3D position, i.e., geometry information, and each point can also be associated with a number of attributes, such as colour, reflectance, surface normal, etc.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — Coded representation of immersive media —

Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC)

1 Scope

This document specifies the syntax, semantics and decoding for visual volumetric media using video-based coding methods. This document also specifies processes that can be needed for reconstruction of visual volumetric media, which can also include additional processes such as post-decoding, pre-reconstruction, post-reconstruction and adaptation.

2 Normative references

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

IEEE 754-2019, *IEEE Standard for Floating-Point Arithmetic*

IETF RFC 1321*The MD5 Message-Digest Algorithm*

IETF RFC 5646*Tags for Identifying Languages*

ISO/IEC 10646, *Information technology — Universal Coded Character Set (UCS)*

ISO/IEC 14496-10:2020, *Information technology — Coding of audio-visual objects — Part 10: Advanced Video Coding*

ISO/IEC 14496-12, *Information technology — Coding of audio-visual objects — Part 12: ISO base media file format*

ISO/IEC 14496-15, *Information technology — Coding of audio-visual objects — Part 15: Carriage of network abstraction layer (NAL) unit structured video in the ISO base media file format*

ISO/IEC 23008-2:2020, *Information technology — High efficiency coding and media delivery in heterogeneous environments — Part 2: High efficiency video coding*

ISO/IEC 23090-3, *Information technology — Coded representation of immersive media — Part 3: Versatile video coding*

ISO/IEC 23091-2, *Coding-independent code points for video signal type identification*

Rec. ITU-T T.35:2000, *Procedure for the allocation of ITU-T defined codes for non standard facilities*

Rec. ITU-T H.271:2006, *Video back-channel messages for conveyance of status information and requests from a video receiver to a video sender*

3 Terms and definitions

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

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

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at http://www.electropedia.org/

3.1
3D bounding box
volume defined as a cuboid solid having six rectangular faces placed at right angles

3.2
associated non-ACL NAL unit
non-ACL NAL unit (3.78) that is associated with an *ACL NAL unit* (3.6) for the purpose of decoding or other operations specified by this document

3.3
associated ACL NAL unit
preceding *ACL NAL unit* (3.6) in decoding order for a *non-ACL NAL unit* (3.78) with nal_unit_type equal to NAL_EOS, NAL_EOB, NAL_FD, NAL_SUFFIX_NSEI, or NAL_SUFFIX_ESEI, or in the ranges of NAL_RSV_NACL_48..NAL_RSV_NACL_52 or NAL_UNSPEC_53..NAL_UNSPEC_63; or otherwise the next *ACL NAL unit* (3.6) in decoding order

3.4
atlas
collection of 2D bounding boxes and their associated information placed onto a rectangular frame and corresponding to a volume in 3D space on which volumetric data is rendered

3.5
atlas bitstream
sequence of bits that forms the representation of *atlas frames* (3.8) and associated data forming one or more *CASs* (3.36)

3.6
atlas coding layer NAL unit
ACL NAL unit
collective term for coded atlas tile layer *NAL units* (3.76) and the subset of *NAL units* (3.76) that have reserved values of nal_unit_type that are classified as being of type class equal to ACL in this document

3.7
atlas coordinates
two scalars (x, y) with finite precision and dynamic range that indicate the location of an atlas sample relative to the top left corner of an atlas frame, with x and y indicating the horizontal and vertical direction, respectively

3.8
atlas frame
2D rectangular array of atlas samples onto which *patches* (3.85) are projected and additional information related to the *patches* (3.85), corresponding to a *volumetric frame* (3.138)

3.9
atlas frame parameter set
AFPS
syntax structure (3.120) containing *syntax elements* (3.119) that apply to zero or more entire coded *atlas frames* (3.8) as determined by the content of a *syntax element* (3.119) found in each tile header

3.10
atlas sample
position on the rectangular frame onto which *patches* (3.85) that are associated with an *atlas* (3.4) are projected

3.11
atlas sequence
collection of *atlas frames* (3.8)

3.12
atlas sequence parameter set
ASPS
syntax structure (3.120) containing *syntax elements* (3.119) that apply to zero or more entire *coded atlas sequences* (3.35) as determined by the content of a *syntax element* (3.119) found in the AFPS referred to by a syntax element found in each tile header

3.13
atlas sub-bitstream
extracted *sub-bitstream* (3.108) from the *V3C bitstream* (3.128) containing a part of an atlas NAL bitstream

3.14
atlas unit
set of *NAL units* (3.76) that contain all *ACL NAL units* (3.6) of a *coded atlas* (3.34) and their *associated non-ACL NAL units* (3.2)

3.15
attribute
scalar or vector property optionally associated with each point in a *volumetric frame* (3.138) such as colour, reflectance, surface normal, transparency, material ID, etc.

3.16
attribute access unit
collection of *attribute maps* (3.18) and auxiliary attribute frames, if available, for a specific *attribute* (3.15) that correspond to the same time instance

3.17
attribute frame
2D rectangular array created through the aggregation of *patches* (3.85) containing values of a specific *attribute* (3.15)

3.18
attribute map
attribute frame (3.17) containing attribute *patch* (3.85) information projected at a particular depth indicated by the corresponding *geometry map* (3.59)

3.19
auxiliary attribute frame
2D rectangular array that is associated with *RAW patches* (3.103) and *EOM patches* (3.51), and contains values of a specific *attribute* (3.15)

3.20
auxiliary geometry frame
2D rectangular array that is associated with *RAW patches* (3.103) and contains *geometry* (3.56) values

3.21
auxiliary video component
video component (3.125) indicated as being of auxiliary type through an appropriate flag in the VPS (3.134) and containing data only associated with *RAW patches* (3.103) or *EOM patches* (3.51), or both

3.22
auxiliary video sub-bitstream
video sub-bitstream (3.127) indicated as being of auxiliary type through an appropriate flag in the *VPS* (3.134) and containing data only associated with *RAW patches* (3.103) or *EOM patches* (3.51), or both

3.23
bitstream
ordered series of bits that forms the coded representation of the data

3.24
byte
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

3.25
broken link access access unit
BLA access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *BLA coded atlas* (3.26)

3.26
broken link access coded atlas
BLA coded atlas
IRAP coded atlas (3.73) frame for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_BLA_W_LP, NAL_GBLA_W_LP, NAL_BLA_W_RADL NAL_GBLA_W_RADL, NAL_BLA_N_LP or NAL_GBLA_N_LP.

Note 1 to entry: A BLA coded atlas does not use inter prediction in its decoding process, and could be the first coded atlas in the bitstream in decoding order, or could appear later in the bitstream. Each BLA coded atlas begins a new CAS, and has the same effect on the decoding process as an instantaneous decoding refresh (IDR) coded atlas. However, a BLA coded atlas contains syntax elements that specify a non-empty DAB. When a BLA coded atlas frame for which each ACL NAL unit has nal_unit_type equal to NAL_BLA_W_LP or NAL_GBLA_W_LP, it may have associated random access skipped leading (RASL) coded atlas frames, which are not output by the decoder and may not be decodable, as they may contain references to atlas frames that are not present in the bitstream. When a BLA coded atlas frame for which each ACL NAL unit has nal_unit_type equal to NAL_BLA_W_LP or NAL_GBLA_W_LP, it may also have associated RADL coded atlas frames, which are specified to be decoded. When a BLA coded atlas frame for which each ACL NAL unit has nal_unit_type equal to NAL_BLA_W_RADL or NAL_GBLA_W_RADL, it does not have associated RASL coded atlas frames but may have associated random access decodable leading (RADL) coded atlas frames. When a BLA coded atlas frame for which each ACL NAL unit has nal_unit_type equal to NAL_BLA_N_LP or NAL_GBLA_N_LP, it does not have any associated leading coded atlas frames.

3.27
byte-aligned
positioned as an integer multiple of 8 bits from the position of the first bit in the *bitstream* (3.23)

3.28
byte-aligned position
position in a *bitstream* (3.23) that is *byte-aligned* (3.27)

3.29
byte-aligned byte
byte (3.24) that appears in a position in a *bitstream* (3.23) that is *byte-aligned* (3.27)

3.30
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)

3.31
clean random access access unit
CRA access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *CRA coded atlas* (3.32)

3.32
clean random access coded atlas
CRA coded atlas
IRAP coded atlas (3.73)for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_CRA or NAL_GCRA

Note 1 to entry: A CRA coded atlas does not use inter prediction in its decoding process, and could be the first coded atlas in the bitstream in decoding order, or could appear later in the bitstream. A CRA coded atlas could have associated RADL or RASL coded atlas frames. When a CRA coded atlas has NoOutputBeforeRecoveryFlag equal to 1, the associated RASL coded atlas frames are not output by the decoder, because they might not be decodable, as they could contain references to coded atlas frames that are not present in the bitstream.

3.33
codec
specification, device, or system that specifies or uses well defined instructions for encoding or decoding a digital data, i.e. image or video, stream or signal

3.34
coded atlas
coded atlas frame
coded representation of an *atlas* (3.4)

3.35
coded atlas access unit
set of atlas *NAL units* (3.76) that are associated with each other according to a specified classification rule, are consecutive in decoding order, and contain all atlas *NAL units* (3.76) pertaining to one particular output time

3.36
coded atlas sequence
CAS
sequence of *coded atlas access units* (3.35), in decoding order, of an *IRAP coded atlas access unit* (3.74), followed by zero or more *coded atlas access units* (3.35) that are not *IRAP coded atlas access units* (3.74), including all subsequent *access units* (3.35) up to but not including any subsequent *coded atlas access unit* (3.35) that is an *IRAP coded atlas access unit* (3.74)

3.37
coded volumetric frame
collection of coded representations of an *atlas* (3.4), *occupancy* (3.83), *geometry access unit* (3.57), and, for each available *attribute* (3.15), *attribute access unit* (3.16), pertaining to one particular time instance

3.38
coded V3C sequence
CVS
sequence of *V3C sub-bitstreams* (3.133) identified and separated by appropriate delimiters, required to start with a *VPS* (3.134), included in at least one *V3C unit* (3.135) or provided through external means, and contains one or more V3C units that can be factored into *V3C composition units* (3.131), where the first V3C composition unit is a *V3C IRAP composition unit* (3.132)

3.39
coded representation
data element as represented in its coded form

3.40
coded sub-bitstream sequence
sub-bitstream IRAP composition unit (3.110) followed by zero or more sub-bitstream composition units (3.109)

3.41
coded V3C component
coded representation of a *V3C component* (3.129)

3.42
component bitstream
bitstream (3.23) representing a *V3C component* (3.129)

3.43
component sub-bitstream
portion of *component bitstream* (3.42)

3.44
composition time
time or time period at which a frame needs to be composed, used for reconstruction, or presented

3.45
composition time index
index to an ordered list of *composition times* (3.44)

3.46
composition unit
partition of a *bitstream* (3.23) that has a certain *composition time* (3.44)

3.47
decoder under test
DUT
decoder that is tested for conformance to this document by operating the hypothetical stream scheduler to deliver a conforming *bitstream* (3.23) to the decoder and to the hypothetical reference decoder and comparing the values and timing or order of the output of the two decoders

3.48
decoding unit
sub-set of a *coded atlas access unit* (3.35) consisting of one or more ACL NAL units in a *coded atlas access unit* (3.35) and the *associated non-ACL NAL units* (3.2)

3.49
enhanced occupancy mode
EOM
patch coding mode where a *patch* (3.85) is associated with enhanced occupancy information

3.50
EOM coded points
coded representation of 3D points located at intermediate depth positions for which *geometry* (3.56) values are stored as codewords in the *occupancy frame* (3.84) and their corresponding attributes values are stored in additional patches, referred to as *EOM patches* (3.51), in the *attribute frames* (3.17)

3.51
EOM patch
patch (3.85) with *patch mode* (3.87) equal to I_EOM, P_EOM, or P_SKIP associated with *EOM coded points* (3.50)

3.52
EOM patch type
patch type (3.88) indicating an *EOM patch* (3.51)

3.53
essential supplemental enhancement information
ESEI
SEI (3.117) that is deemed as essential by the decoding process and should not be ignored or discarded

3.54
essential supplemental enhancement information NAL unit
ESEI NAL unit
NAL unit (3.76) corresponding to an *ESEI* (3.53) and has nal_unit_type equal to NAL_PREFIX_ESEI or NAL_SUFFIX_ESEI

3.55
flag
variable or single-bit syntax element that can take one of the two possible values: 0 and 1

3.56
geometry
set of *Cartesian coordinates* (3.30) associated with a *volumetric frame* (3.138)

3.57
geometry access unit
collection of *geometry maps* (3.59) and auxiliary geometry frames, if present, corresponding to the same time instance

3.58
geometry frame
2D array created through the aggregation of the *geometry* (3.56) information associated with each *patch* (3.85)

3.59
geometry map
geometry frame (3.58) containing geometry *patch* (3.85) information projected at a particular depth

3.60
global broken link access access unit
GBLA access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *GBLA coded atlas* (3.61)

3.61
global broken link access coded atlas
GBLA coded atlas
IRAP coded atlas (3.73) frame for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_GBLA_W_LP, NAL_GBLA_W_RADL, or NAL_GBLA_N_LP respectively

3.62
global clean random access access unit
GCRA access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *GCRA coded atlas* (3.63)

3.63
global clean random access coded atlas
GCRA coded atlas
IRAP coded atlas (3.73) frame for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_GCRA

3.64
global instantaneous decoding refresh access unit
GIDR access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *GIDR coded atlas* (3.65)

3.65
global instantaneous decoding refresh coded atlas access unit
GIDR-coded atlas
coded atlas (3.34) for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_GIDR_W_RADL, NAL_GBLA_N_LP, or NAL_GCRA, or in the range of NAL_GBLA_W_LP to NAL_GBLA_N_LP, inclusive, and specify a random access association between the current *coded atlas* and its corresponding coded video frames at the same composition time

3.66
global intra random access point coded atlas
GIRAP coded atlas
IRAP coded atlas (3.73) for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_GIDR_W_RADL, NAL_GBLA_N_LP, or NAL_GCRA, or in the range of NAL_GBLA_W_LP to NAL_GBLA_N_LP, inclusive, and specify a random access association between the current coded atlas and the corresponding coded video frames at the same composition time

3.67
hypothetical reference decoder
HRD
hypothetical decoder model that specifies constraints on the variability of conforming atlas *NAL unit* (3.76) streams or conforming *coded atlas* (3.34) sample streams that an encoding process can produce

3.68
hypothetical stream scheduler
HSS
hypothetical delivery mechanism used for checking the conformance of an *atlas sub-bitstream* (3.13) or a decoder with regards to the timing and data flow of the input of an *atlas sub-bitstream* (3.13) into the *hypothetical reference decoder* (3.67)

3.69
instantaneous decoding refresh coded atlas access unit
IDR coded atlas access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is an *IDR coded atlas* (3.70)

3.70
instantaneous decoding refresh coded atlas
IDR coded atlas
IRAP coded atlas (3.73) for which each *ACL NAL unit* (3.6) 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 1 to entry: 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.

3.71
inter atlas tile
atlas tile that can be decoded using both intra or inter prediction methods

3.72
intra atlas tile
atlas tile that is decoded using only intra prediction methods

3.73
intra random access point coded atlas
IRAP coded atlas
IRAP coded atlas frame
coded atlas (3.34) for which each *ACL NAL unit* (3.6) has nal_unit_type in the range of NAL_BLA_W_LP to NAL_RSV_IRAP_ACL_29, inclusive

Note 1 to entry: An IRAP coded atlas does not refer to any coded atlases other than itself for prediction in its decoding process, and may be a BLA coded atlas, a CRA coded atlas, or an IDR 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.

3.74
intra random access point coded atlas access unit
IRAP coded atlas access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *IRAP coded atlas* (3.73)

3.75
multi-component collection of V3C sub-bitstreams
V3C sub-bitstreams (3.133) of multiple *V3C components* (3.129) that, when decoded, enable the reconstruction of volumetric content

3.76
network abstraction layer unit
NAL unit
syntax structure containing an indication of the type of data to follow and *bytes* (3.24) containing that data in the form of an *RBSP* (3.100)

3.77
network abstraction layer unit stream
NAL unit stream
sequence of *NAL units* (3.76)

3.78
non-ACL NAL unit
NAL unit (3.76) that is not an *ACL NAL unit* (3.6).

3.79
non-auxiliary video component
regular video component
video component (3.125) indicated as being of non-auxiliary type through an appropriate flag in the *VPS* (3.134)

3.80
non-auxiliary video sub-bitstream
regular video sub-bitstream
video sub-bitstream (3.127) indicated as being of non-auxiliary type through an appropriate flag in the *VPS* (3.134)

3.81
non-essential supplemental enhancement information
NSEI
SEI (3.117) that is deemed as non-essential by the decoding process and may be ignored or discarded without any adverse effects

3.82
non-essential supplemental enhancement information NAL unit
NSEI NAL unit
NAL unit (3.76) corresponding to a *NSEI* (3.81) and has nal_unit_type equal to NAL_PREFIX_NSEI or NAL_SUFFIX_NSEI

3.83
occupancy
values that indicate whether *atlas samples* (3.10) correspond to associated samples in 3D space

3.84
occupancy frame
collection of *occupancy* (3.83) values that constitute a 2D array and represents the entire *occupancy* (3.83) information of a single *atlas frame* (3.8)

3.85
patch
rectangular region within an *atlas* (3.4) associated with volumetric information

3.86
patch data
data in an *atlas* (3.4) associated with a *patch* (3.85) that enables the conversion of 2D projected data back to 3D space

3.87
patch mode
syntax element (3.119) in the *atlas bitstream* (3.5) that indicates how a *patch* (3.85) is defined and associated with other *V3C components* (3.129) and provides information of how to reconstruct such *V3C components* (3.129)

3.88
patch type
classification of *patch modes* (3.87) based on how the characteristics of a *patch* (3.85) are signalled and interpreted

3.89
prefix ESEI NAL unit
essential SEI NAL unit (3.54) that has nal_unit_type equal to NAL_PREFIX_ESEI

3.90
prefix NSEI NAL unit
non-essential SEI NAL unit (3.82) that has nal_unit_type equal to NAL_PREFIX_NSEI

3.91
prefix SEI message
SEI (3.117) message that is contained in a *prefix NSEI NAL unit* (3.90) or *prefix ESEI NAL unit* (3.89)

3.92
prefix SEI NAL unit
SEI NAL unit (3.118) that has nal_unit_type equal to NAL_PREFIX_NSEI or NAL_PREFIX_ESEI

3.93
projected patch
patch (3.85) with *patch mode* (3.87) equal to I_INTRA, P_INTRA, P_INTER, P_MERGE, or P_SKIP that is associated with projected information onto a 2D image

3.94
projected patch type
patch type (3.88) indicating a projected patch (3.93)

3.95
random access
the act of starting the decoding process for a *bitstream* (3.23) at a point other than the beginning of the stream

3.96
random access decodable leading access unit
RADL access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *RADL coded atlas* (3.97)

3.97
random access decodable leading coded atlas
RADL coded atlas
coded atlas (3.34) for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_RADL_R or NAL_RADL_N

Note 1 to entry: All RADL coded atlases are leading coded atlas. RADL coded atlases are not used as reference atlases for the decoding process of trailing coded atlases of the same associated IRAP coded atlas. When present, all RADL coded atlases precede, in decoding order, all trailing coded atlases of the same associated IRAP coded atlas.

3.98
random access skipped leading access unit
RASL access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *RASL coded atlas* (3.99)

3.99
random access skipped leading coded atlas
RASL coded atlas
coded atlas (3.34) for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_RASL_R or NAL_RASL_N

Note 1 to entry: All RASL coded atlases are leading coded atlas of an associated BLA or CRA coded atlas. When the associated IRAP coded atlas has NoOutputBeforeRecoveryFlag equal to 1, the RASL coded atlas is not output and may not be correctly decodable, as the RASL coded atlas may contain references to coded atlases that are not present in the bitstream. RASL coded atlases are not used as reference atlases for the decoding process of non-RASL coded atlases. When present, all RASL coded atlases precede, in decoding order, all trailing coded atlases of the same associated IRAP coded atlas.

3.100
raw byte sequence payload
RBSP
syntax structure (3.120) containing an integer number of *bytes* (3.24) that is encapsulated in a *NAL unit* (3.76) and that is either empty or has the form of a *string of data bits* (3.107) containing *syntax elements* (3.119) followed by an *RBSP stop bit* (3.101) and zero or more subsequent bits equal to 0

3.101
raw byte sequence payload stop bit
RBSP stop bit
bit equal to 1 present within a *raw byte sequence payload* (3.100) after a *string of data bits* (3.107), for which the location of the end within an *RBSP* (3.100) can be identified by searching from the end of the *RBSP* (3.100) for the for the last non-zero bit in the *RBSP* (3.100)

3.102
RAW coded point
coded representation (3.39) of a 3D point for which its *geometry* (3.56) value is directly stored in the corresponding region of a *RAW patch* (3.103) in either the first component of the first *geometry map* (3.59) or the *auxiliary geometry frame* (3.20), and for which its attribute values, if present, are directly stored in the corresponding region of a *RAW patch* (3.103) in either first *attribute map* (3.18) or the *auxiliary attribute frame* (3.19), without any form of projection

3.103
RAW patch
patch (3.85) with *patch mode* (3.87) equal to I_RAW, P_RAW, or P_SKIP associated with *RAW coded points* (3.102)

3.104
RAW patch type
patch type (3.88) indicating a *RAW patch* (3.103)

3.105
step-wise temporal sub-layer access access unit
STSA access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *STSA coded atlas* (3.106)

3.106
step-wise temporal sub-layer access coded atlas
STSA coded atlas
coded atlas (3.34) for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_STSA_R or NAL_STSA_N

3.107
string of data bits
SODB
sequence of some number of bits representing *syntax elements* (3.119) present within a *raw byte sequence payload* (3.100) prior to the *raw byte sequence payload stop bit* (3.101), where the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit

3.108
sub-bitstream
portion of a *bitstream* (3.23)

3.109
sub-bitstream composition unit
partition of a *sub-bitstream* (3.108) that has a certain composition time

3.110
sub-bitstream IRAP composition unit
sub-bitstream composition unit (3.109) that forms an independent random access point for the *sub-bitstream* (3.108)

3.111
sub-layer
temporal scalable layer of a temporal scalable *atlas sub-bitstream* (3.13), consisting of *ACL NAL units* (3.6) with a particular value of the TemporalId variable and the associated *non-ACL NAL units* (3.78)

3.112
sub-layer non-reference atlas frame
SLNR atlas frame
atlas frames (3.8) that contains *patches* (3.85) that cannot be used for inter prediction in the decoding process of subsequent *atlas frames* (3.8) of the same sub-layer in decoding order

3.113
suffix ESEI NAL unit
ESEI NAL (3.54) unit that has nal_unit_type equal to NAL_SUFFIX_ESEI

3.114
suffix NSEI NAL unit
NSEI NAL unit (3.82) that has nal_unit_type equal to NAL_SUFFIX_NSEI

3.115
suffix SEI message
SEI (3.117) message that is contained in either a *suffix NSEI NAL unit* (3.114) or a *suffix ESEI NAL unit* (3.113)

3.116
suffix SEI NAL unit
SEI NAL unit (3.118) that has nal_unit_type equal to NAL_SUFFIX_NSEI or NAL_SUFFIX_ESEI

3.117
supplemental enhancement information
SEI
information that can be inserted into the *bitstream* (3.23) for various purposes such as indicating *bitstream* (3.23) properties or for specifying various constraints that can apply to the *bitstream* (3.23)

3.118
supplemental enhancement information NAL unit
SEI NAL unit
NAL unit (3.76), intended for providing *supplemental enhancement information* (3.117) that has nal_unit_type equal to NAL_PREFIX_NSEI, NAL_PREFIX_ESEI, NAL_SUFFIX_NSEI, or NAL_SUFFIX_ESEI

3.119
syntax element
element of data represented in the *bitstream* (3.23)

3.120
syntax structure
zero or more *syntax elements* (3.119) present together in the *bitstream* (3.23) in a specified order

3.121
temporal sub-layer access access unit
TSA access unit
access unit in which the *coded atlas* (3.34) with nal_layer_id equal to 0 is a *TSA coded atlas* (3.122)

3.122
temporal sub-layer access coded atlas
TSA coded atlas
coded atlas (3.34) for which each *ACL NAL unit* (3.6) has nal_unit_type equal to NAL_TSA_R or NAL_TSA_N

3.123
tile
independently decodable rectangular region of an *atlas frame* (3.8)

3.124
video bitstream
bitstream (3.23) conforming to a video specification that can represent a *V3C component* (3.129)

3.125
video component
V3C component (3.129) of type *occupancy* (3.83), *geometry* (3.56) or *attribute* (3.15)

3.126
video data unit
syntax structure (3.120) containing video data information, such as a *NAL unit* (3.76)

Note 1 to entry: In the context of ISO/IEC 23008-2 or ISO/IEC 14496-10.

3.127
video sub-bitstream
extracted *sub-bitstream* (3.108) from the *V3C bitstream* (3.128) containing a part of a *video bitstream* (3.136)

3.128
visual volumetric video-based coding bitstream
V3C bitstream
sequence of bits that forms the representation of *coded volumetric frames* (3.37) and associated data forming one or more *CVSs* (3.38)

3.129
visual volumetric video-based coding component
V3C component
atlas (3.4), *occupancy* (3.83), *geometry* (3.56), or *attribute* (3.15) of a particular type that is associated with a V3C volumetric content representation

3.130
visual volumetric video-based coding component frame
V3C component frame
single *V3C component* (3.129) pertaining to one particular output time

3.131
visual volumetric video-based coding composition unit
V3C composition unit
set of all *sub-bitstream composition units* (3.109) that share the same *composition time* (3.44), where one of the *sub-bitstream composition units* (3.109) is a *coded atlas access unit* (3.35)

3.132
visual volumetric video-based coding IRAP composition unit
V3C IRAP composition unit
V3C composition unit (3.131) for which all *sub-bitstream composition units* (3.109) are *sub-bitsteam IRAP composition units* (3.110), where one of the *sub-bitstream composition units* (3.109) is a *coded atlas access unit* (3.35) for which each *ACL NAL unit* (3.6) 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

3.133
visual volumetric video-based coding sub-bitstream
V3C sub-bitstream
sub-bitstream (3.108) of a *V3C bitstream* (3.128) corresponding to a *V3C component* (3.129)

3.134
visual volumetric video-based coding parameter set
V3C parameter set
VPS
syntax structure (3.120) containing *syntax elements* (3.119) that apply to zero or more entire *CVSs* (3.38) and may be referred to by *syntax elements* (3.119) found in the *V3C unit header* (3.136)

3.135
visual volumetric video-based coding unit
V3C unit
syntax structure (3.120) containing a *V3C unit header* (3.136) and a *V3C unit payload* (3.137)

3.136
visual volumetric video-based coding unit header
V3C unit header
syntax structure (3.120) containing an indication of the type of data to follow

3.137
visual volumetric video-based coding unit payload
V3C unit payload
syntax structure (3.120) containing the *bytes* (3.24) containing *V3C sub-bitstream* (3.133) data

3.138
volumetric frame
set of 3D points specified by their *Cartesian coordinates* (3.30) and zero or more corresponding sets of *attributes* (3.14) at a particular time instance

3.139
volumetric sequence
sequence of *volumetric frames* (3.138)

3.140
point
<video-based point cloud coding> data specified by single set of *Cartesian coordinates* (3.30) and zero or more corresponding *attributes* (3.14)

3.141
point cloud
point cloud frame
<video-based point cloud coding> set of *points* (3.140) at a particular time instance

3.142
point cloud sequence
<video-based point cloud coding> sequence of *point cloud frames* (3.141)

4 Abbreviated terms

1D	one-dimensional
2D	two-dimensional
3D	three-dimensional
4D	four-dimensional
5D	five-dimensional
6D	six-dimensional
7D	seven-dimensional
ACL	atlas coding layer
AAPS	atlas adaptation parameter set
AFPS	atlas frame parameter set
ASPS	atlas sequence parameter set
AU	access unit
BLA	broken link access
CAB	coded atlas buffer
CRA	clean random access
DAB	decoded atlas buffer
HSS	hypothetical stream scheduler
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
PBF	patch border filtering
PCC	point cloud compression
PLR	point local reconstruction

RAFL	reference atlas frame list
RBSP	raw byte sequence payload
SODB	string of data bits
V3C	visual volumetric video-based coding
V-PCC	video-based point cloud compression

5 Conventions

5.1 General

NOTE The mathematical operators used in this document 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

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

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

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

5.5 Bit-wise operators

~	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

=	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 Other operators

y..z	range operator/notation This function is defined only for integer values of y and z. When z is larger than or equal to y, it defines an ordered set of values from y to z in increments of 1. Otherwise, when z is smaller than y, the output of this function is an empty set. If this operator is used within the context of a loop, it specifies that any subsequent operations defined are performed using each element of this set, unless this set is empty.

5.8 Mathematical functions

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

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

$$\text{Clip3}(\ x, y, z\ ) = \begin{cases} x & ; \quad z < x \\ y & ; \quad z > y \\ z & ; \quad \text{otherwise} \end{cases} \tag{3}$$

Floor(x) the largest integer less than or equal to x. (4)

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

Log10(x) the base-10 logarithm of x. (6)

$$\text{Max}(\ x, y\ ) = \begin{cases} x & ; \quad x >= y \\ y & ; \quad x < y \end{cases} \tag{7}$$

Mean(A) the arithmetic average value of the input array A. (8)

Median(A) the median value of the input array A. (9)

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

Round(x) = Sign(x) * Floor(Abs(x) + 0.5) (11)

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

Sort(A) the sorted version of the input array A in increasing order. (13)

Sqrt(x) the square root of x. (14)

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 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 document is the same as used in the C programming language.

Table 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 1 The syntax is described in a manner that closely follows the C-language syntactic constructs.

Functions that specify properties of the current position in the bitstream are referred to as syntax functions. These functions are specified in subclause 8.2 and assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round parentheses including zero or more variable names (for definition) or values (for usage), separated by commas (if more than one variable).

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

A one-dimensional array is referred to as a list. A two-dimensional array is referred to as a matrix. Arrays can either be syntax elements or variables. Subscripts or square parentheses are used for the indexing of arrays. In reference to a visual depiction of a matrix, the first subscript is used as a row (vertical) index and the second subscript is used as a column (horizontal) index. The same indexing order is used when using square parentheses. Thus, an element of a matrix s at horizontal position x and vertical position y may be denoted either as s[y][x] or as 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 2 In some video specifications a reverse indexing order can 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.

The symbol "×" is used to separate the individual dimensions of an array in this document. For example, a 3D array, array3D, having elements array3D[z][y][x], with z being in the range of 0 to ($\mathrm{Dim}_{0,}$ − 1), inclusive, y in the range of 0 to (Dim_1 − 1), inclusive, and x in the range of 0 to (Dim_2 − 1), inclusive, can also be referred to as being of size $\mathrm{Dim}_0 \times \mathrm{Dim}_1 \times \mathrm{Dim}_2$.

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

In addition, a "continue" statement, which is used within loops, is defined as follows:

The "continue" statement, when encountered inside a loop, jumps to the beginning of the loop for the next iteration. This results in skipping the execution of subsequent statements inside the body of the loop for the current iteration. For example:

```
for( j =0; j < N; j++ ) {
    statement 0
    if( condition 1 )
        continue
    statement 1
    statement 2
}
```

is equivalent to the following:

```
for( j =0; j < N; j++ ) {
    statement 0
    if( !condition 1 ) {
        statement 1
        statement 2
    }
}
```

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.

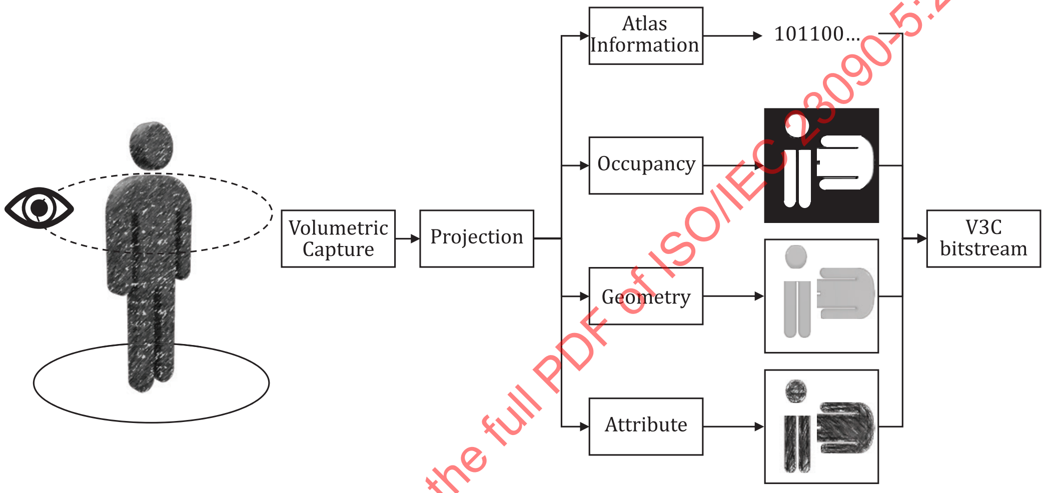
— Otherwise (the variables at the invoking and the process specification have the same name), assignment is implied.

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.

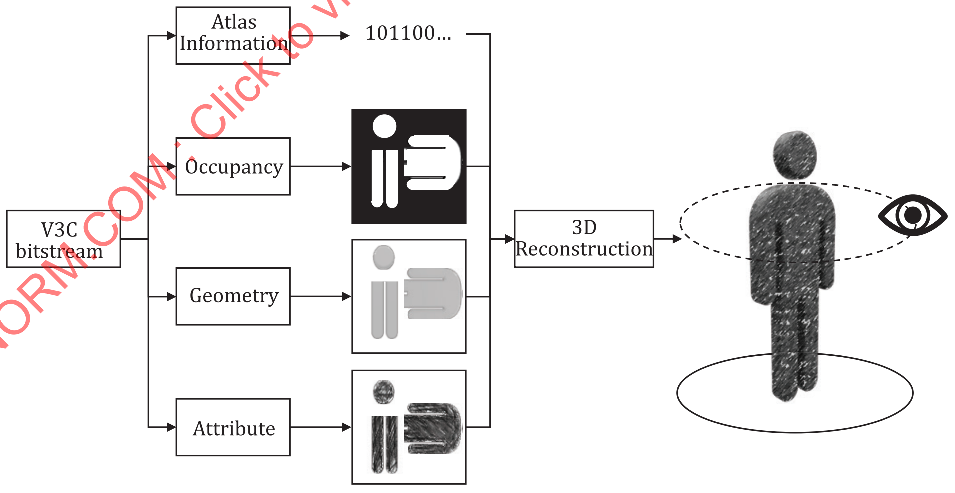
6 Overall V3C characteristics, decoding operations and post-decoding processes

6.1 V3C characteristics

This document enables the encoding and decoding processes of a variety of volumetric media by using video and image coding technologies. This is achieved through first a conversion of such media from their corresponding 3D representation to multiple 2D representations, also referred to as V3C components in this document, before coding such information. Such representations include, in this edition of this document, the occupancy, geometry and attribute components. The occupancy component can inform a V3C decoding or rendering system, or both, of which samples in the 2D components are associated with data in the final 3D representation. The geometry component contains information about the precise location of 3D data in space, while attribute components can provide additional properties, e.g. texture or material information, of such 3D data. An example is shown in Figure 1.



(a)



(b)

NOTE 1 The 3D media is converted to a series of 2D representations: occupancy, geometry and attributes. Additional atlas information is also included in the bitstream to enable inverse reconstruction.

Figure 1 — Volumetric media conversion at (a) encoder and reconstruction at (b) decoder side

Additional information that allows associating all these subcomponents and enables the inverse reconstruction, from a 2D representation back to a 3D representation is also included in a special component, referred to in this document as the atlas. An atlas consists of multiple elements, named as patches. Each patch identifies a region in all available 2D components and contains information necessary to perform the appropriate inverse projection of this region back to the 3D space. The shape of such regions is determined through a 2D bounding box associated with each patch as well as their coding order. The shape of these regions is also further refined after the consideration of the occupancy information.

Atlases are partitioned into patch packing blocks of equal size, as defined in subclause 8.4.6.1.1. The 2D bounding boxes of patches, as defined in subclause 9.2.5, and their coding order determine the mapping between the blocks of the atlas image and the patch indices. Figure 2 shows an example of block to patch mapping with 4 projected patches onto an atlas when asps_patch_precedence_order_flag is equal to 0. Projected points are represented with dark grey. The area that does not contain any projected points is represented with light grey. Patch packing blocks are represented with dashed lines. The number inside each patch packing block represents the patch index of the patch to which it is mapped.

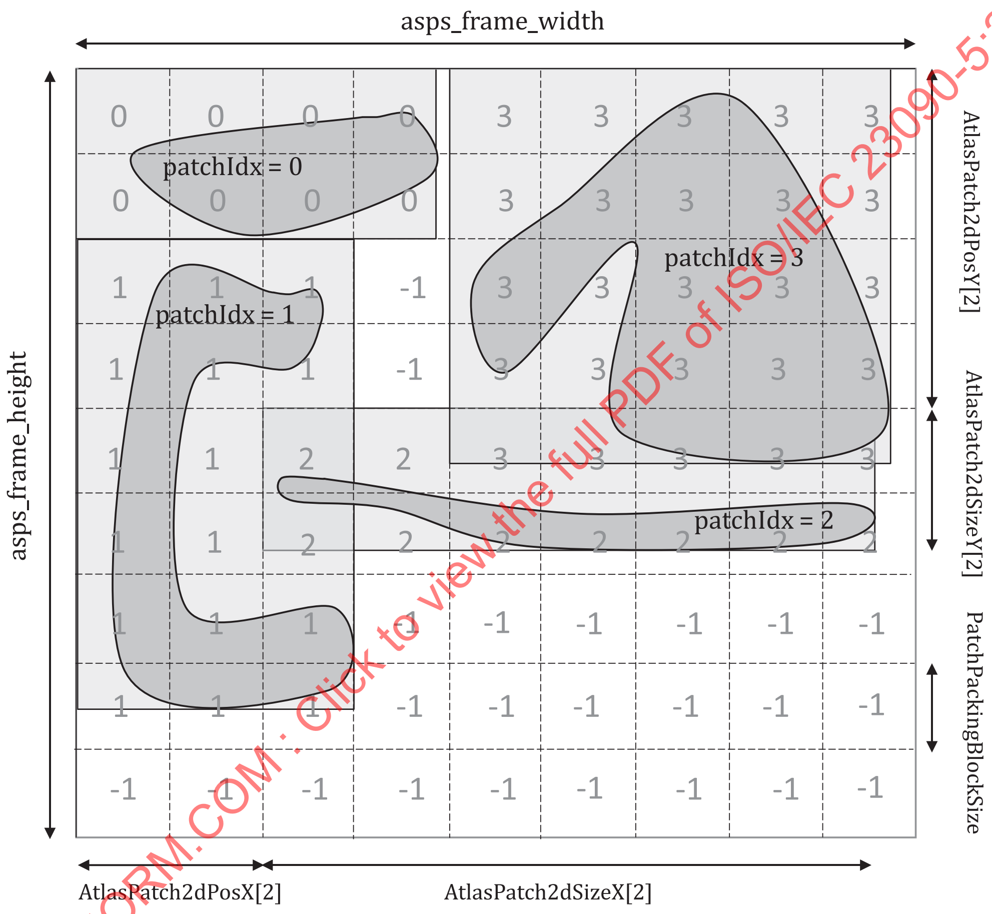


Figure 2 — An example of block to patch mapping

In this document, axes orientations are specified for internal operations. For instance, the origin of the atlas coordinates is located on the top-left corner of the atlas frame. For the reconstruction step, an intermediate axes definition for a local 3D patch coordinate system is used. The 3D local patch coordinate system is then converted to the final target 3D coordinate system using appropriate transformation steps.

NOTE 2 In some instances the transformation steps to convert from the local 3D patch coordinate system to the target 3D coordinate system are specified in this document.

Figure 3(a) shows an example of a single patch packed onto an atlas image. This patch is then converted to a local 3D patch coordinate system (U, V, D) defined by the projection plane with origin O’, tangent (U), bi-tangent (V), and normal (D) axes. For an orthographic projection, the projection plane is equal to the sides of an axis-aligned 3D bounding box, as shown in Figure 3(b). The location of the bounding box in the 3D model coordinate system, defined by a left-handed system with axes (X, Y, Z), can be obtained by adding offsets TilePatch3dOffsetU, TilePatch3DOffsetV, and TilePatch3DOffsetD, as illustrated in Figure 3(c).

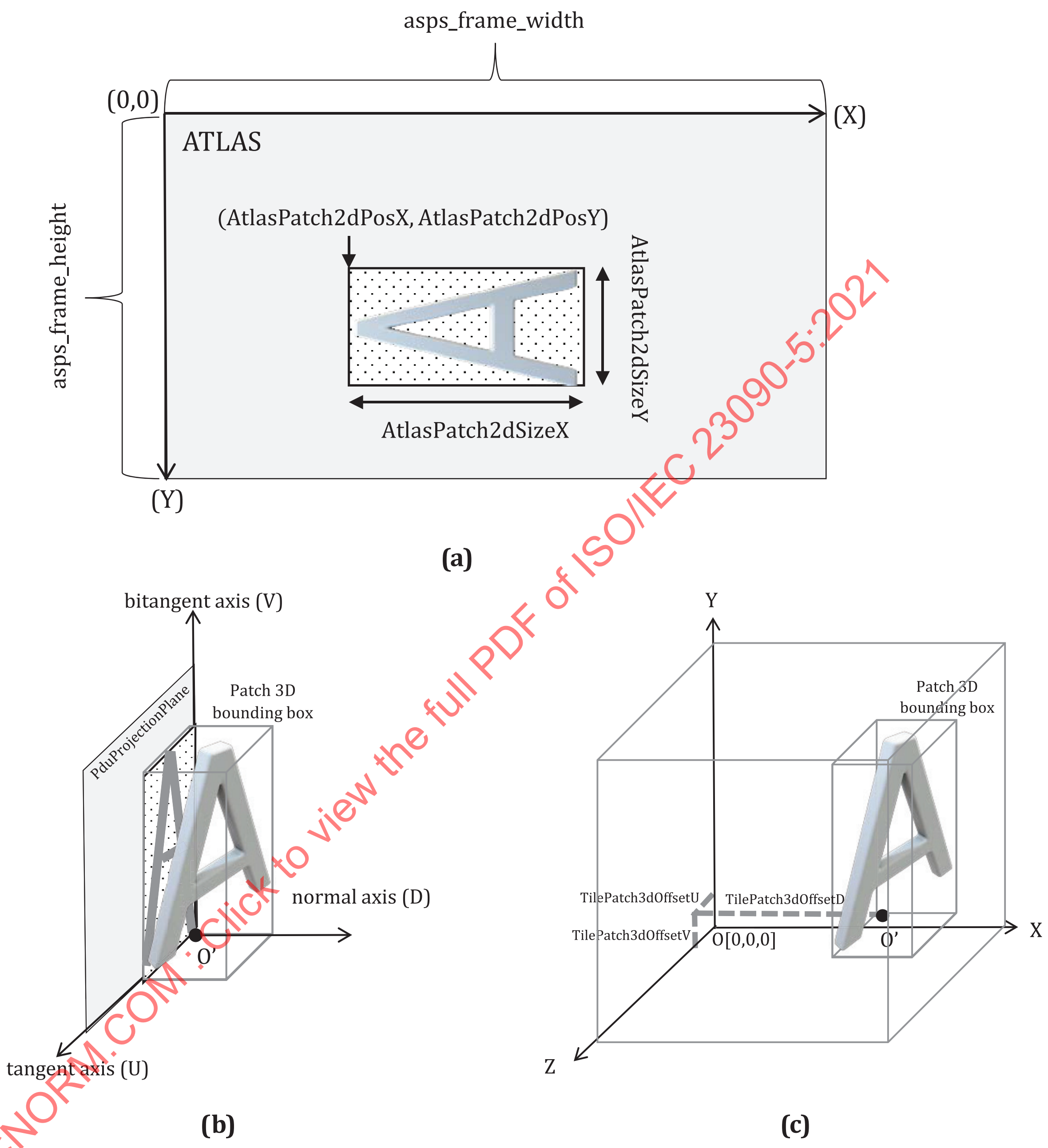


Figure 3 — Example of (a) an atlas coordinate system, (b) a local 3D patch coordinate system and (c) final target 3D coordinate system

6.2 V3C bitstream characteristics, decoding operations and post-decoding processes

This subclause provides high-level description of the characteristics and the operations needed for the decoding of V3C bitstreams and optional post-decoding processes needed by applications, which may include nominal format conversion, pre-reconstruction, reconstruction, post-reconstruction and adaptation.

As mentioned in subclause 6.1, the V3C bitstream is composed of a collection of V3C components, such as atlas, occupancy, geometry and attributes. Furthermore, to provide additional functionalities and similar flexibility as available in many video specifications, the atlas component may be divided into tiles and is encapsulated into NAL units. Clause 7 provides further details on the encapsulation of V3C component bitstreams into V3C units and NAL units.

V3C bitstream syntax elements and their semantics are specified in Clause 8. Particular focus is placed on the atlas bitstream, its characteristics, and any constraints that may apply on such syntax.

Clause 9 invokes the decoding process of a V3C bitstream or a collection of V3C sub-bitstreams, with outputs composed of decoded atlas information, a set of decoded video sub-bitstreams, corresponding to the occupancy, geometry and attribute, if available, and information of the CVS.

The decoded video frames may require the application of additional transformations, as described in Annex B, before any reconstruction operations. For example, the different components may need to be time-aligned and converted to a nominal video format.

With the V3C components in the nominal format, the 3D volumetric content is obtained according to the following steps: pre-reconstruction, reconstruction, post-reconstruction and adaptation.

For pre-reconstruction, as specified in Clause 10, some of the decoded V3C components may be further processed according to a specific application.

In the reconstruction stage, described in Clause 11, the video components, along with the decoded atlas data, are processed to reconstruct the volumetric content and associated information as outputs. For some specific applications, this output corresponds to a collection of points and their associated attributes, if present.

The reconstructed volumetric content can be further processed, depending on the application, by applying informative post-reconstruction methods, as described in Clause 12.

Depending on the application, the post-reconstructed volumetric content can be further optionally processed by additional transformations, as described in Clause 13.

7 Bitstream format, partitioning and scanning processes

7.1 General

Coded V3C video components are referred to in this document as video bitstreams, while an atlas component is referred to as the atlas bitstream. Video bitstreams and atlas bitstreams may be further split into smaller units, referred to here as video and atlas sub-bitstreams, respectively, and may be interleaved together, after the addition of appropriate delimiters, to construct a V3C bitstream. One particular form of a V3C bitstream, referred to as the V3C sample stream format, that encapsulates all V3C units, is presented in Annex C.

7.2 V3C bitstream formats

This subclause specifies the relationship between the V3C unit stream format and the V3C sample stream, either of which are referred to as the V3C bitstream. All V3C components including any associated V3C VPSs could be encapsulated using a different format depending on application.

The bitstream can be in one of two formats: the V3C unit stream format or the sample stream format. The V3C unit stream format is conceptually the more "basic" type. It consists of a sequence of syntax structures called V3C units. This sequence is ordered in decoding order. There are constraints imposed on the decoding order (and contents) of the V3C units in the V3C unit stream.

NOTE The V3C 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 V3C unit stream format by ordering the V3C units in decoding order and prefixing each V3C unit with a heading that specifies the exact size, in bytes, of the V3C unit. A sample stream header is included at the beginning of the sample stream bitstream that specifies the precision, in bytes, of the signalled V3C unit size. The V3C 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 V3C unit. Methods of framing V3C units in a manner other than the use of the sample stream format are outside the scope of this document. The sample stream format is specified in Annex C.

7.3 NAL bitstream formats

This subclause 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 subclause 8.4.5.3. 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 signalled 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 document. The sample stream format is specified in Annex D.

7.4 Partitioning of atlas frames into tiles

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

To enable parallelization, random access, as well as a variety of other functionalities, an atlas frame can be divided into one or more rectangular partitions, that are referred to as tiles. Tiles are not allowed to overlap. An atlas frame may contain regions that are not associated with a tile.

Partitioning of an atlas frame into tiles can be performed as follows:

— The atlas frame is first partitioned into NumPartitionColumns * NumPartitionRows number of tile partitions, where NumPartitionColumns and NumPartitionRows are derived in subclause 7.5. The width and height of each tile partition, respectively, is indicated in the atlas frame tile information syntax (subclause 8.3.6.2.2).

— One or more tile partitions are then combined into tiles by indicating the locations of the tile partitions that correspond to the top-left and bottom-right corners of the tile. All tile partitions within these two tile partitions collectively form a tile, which is essentially a rectangular region of the atlas frame.

Figure 4 shows an example tile partitioning of an atlas frame, where the atlas frame is divided into 16 tile partitions (based on the indication of 4 tile partition columns and 4 tile partition rows) and 7 tiles.

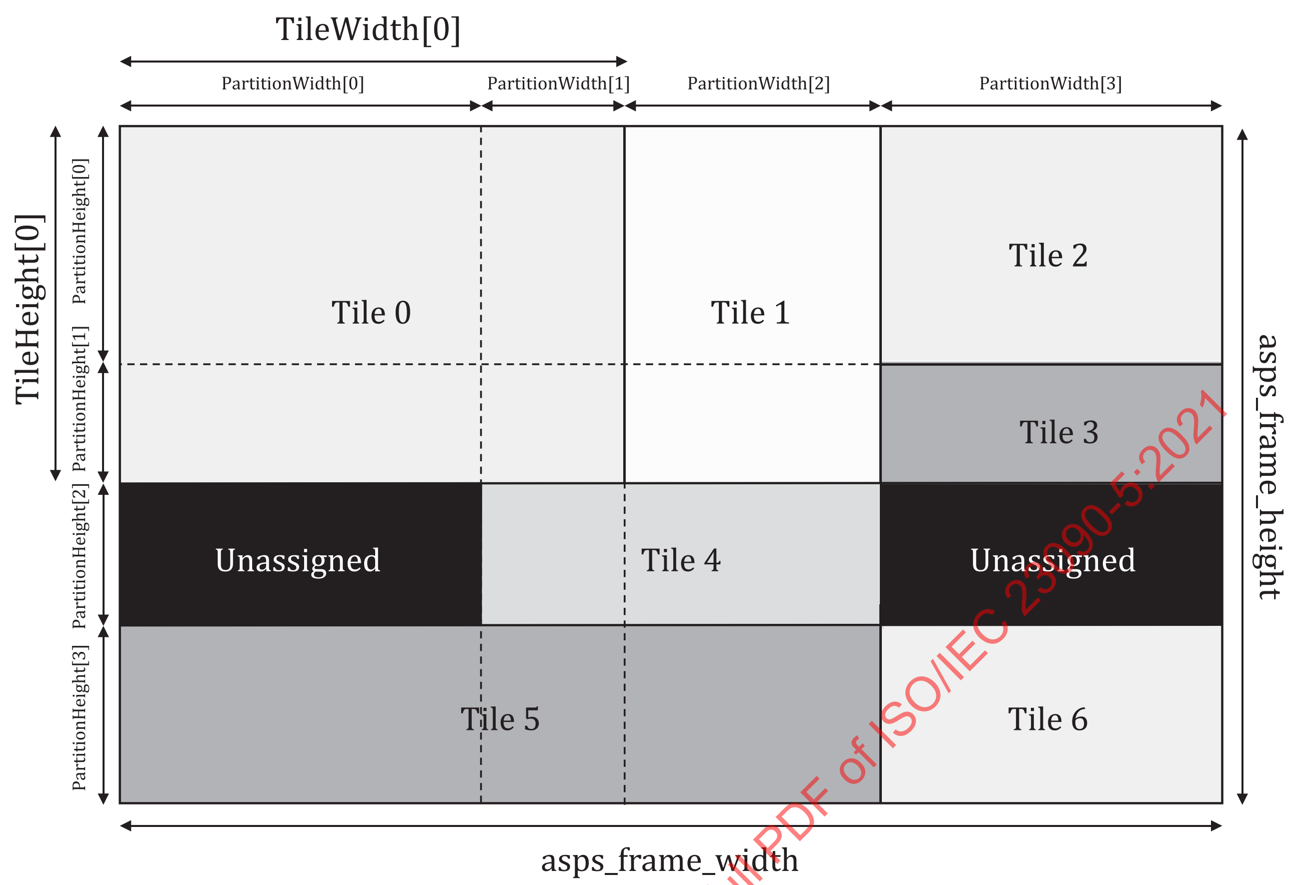


Figure 4 — Example of an atlas frame partitioned into 7 tiles

7.5 Tile partition scanning process

The list PartitionWidth[i], for i ranging from 0 to NumPartitionColumns − 1, inclusive, specifying the width of the i-th tile partition column in units of 64 samples, is derived, and the value of NumPartitionColumns is inferred, as follows:

```
if( afti_uniform_partition_spacing_flag ) {
    partitionWidth = ( afti_partition_cols_width_minus1 + 1 ) * 64
    NumPartitionColumns = asps_frame_width / partitionWidth
    PartitionPosX[ 0 ] = 0
    PartitionWidth[ 0 ] = partitionWidth
    for( i = 1; i < NumPartitionColumns – 1; i++ ) {
        PartitionPosX[ i ] = PartitionPosX[ i – 1 ] + PartitionWidth[ i – 1 ]
        PartitionWidth[ i ] = partitionWidth
    }
} else {
    NumPartitionColumns = afti_num_partition_columns_minus1 + 1
    PartitionPosX[ 0 ] = 0
    partitionWidth[ 0 ] = ( afti_partition_column_width_minus1[ 0 ] + 1 ) * 64
    for( i = 1; i < NumPartitionColumns – 1; i++ ) {
        PartitionPosX[ i ] = PartitionPosX[ i – 1 ] + PartitionWidth[ i – 1 ]
        PartitionWidth[ i ] = ( afti_partition_column_width_minus1[ i ] + 1 ) * 64
    }
}
if( NumPartitionColumns > 1 ) {
    PartitionPosX[ NumPartitionColumns – 1 ] =
        PartitionPosX[ NumPartitionColumns – 2 ] + PartitionWidth[ NumPartitionColumns – 2 ]
        PartitionWidth[ NumPartitionColumns – 1 ] =
                    asps_frame_width – PartitionPosX[ NumPartitionColumns – 1 ]
    }
```

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

```
    if( afti_uniform_partition_spacing_flag ) {
        partitionHeight = (afti_partition_rows_height_minus1 + 1) * 64
        NumPartitionRows = asps_frame_height / partitionHeight
        PartitionPosY[ 0 ] = 0
        PartitionHeight[ 0 ] = partitionHeight
        for( j = 1; j < NumPartitionRows – 1; j++ ) {
            PartitionPosY[ j ] = PartitionPosY[ j – 1 ] + PartitionHeight[ j – 1 ]
            PartitionHeight[ j ] = partitionHeight
        }
    } else {
        NumPartitionRows = afti_num_partition_rows_minus1 + 1
        PartitionPosY[ 0 ] = 0
        PartitionHeight[ 0 ] = ( afti_partition_row_height_minus1[ 0 ] + 1 ) * 64
        for( j = 1; j < NumPartitionRows – 1; j++ ) {
            PartitionPosY[ j ] = PartitionPosY[ j – 1 ] + PartitionHeight[ j – 1 ]
            PartitionHeight[ j ] = ( afti_partition_row_height_minus1[ j ] + 1 ) * 64
        }
    }
    if( NumPartitionRows > 1 ) {
        PartitionPosY[ NumPartitionRows – 1 ] =
            PartitionPosY[ NumPartitionRows – 2 ] + partitionHeight[ NumPartitionRows – 2 ]
        PartitionHeight[ NumPartitionRows – 1 ] =
            asps_frame_height – PartitionPosY[ NumPartitionRows – 1 ]
    }
```

It is a requirement of bitstream conformance that the values of partitionWidth[i] for i ranging from 0 to NumPartitionColumns – 1, inclusive, and partitionHeight[j] for j ranging from 0 to NumPartitionRows – 1, inclusive, are greater than or equal to 64 and are multiples of PatchPackingBlockSize.

8 Syntax and semantics

8.1 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 is expected to 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 document.

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

	Descriptor
/* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type and quantity of syntax elements, as in the following two examples */	
syntax_element	ue(v)

conditioning statement	
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
...	
}	
/* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */	
while(condition)	
statement	
/* A "do ... while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */	
do	
statement	
while(condition)	
/* An "if ... else" structure specifies a test of whether a condition is true and, if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */	
if(condition)	
primary statement	
else	
alternative statement	
/* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */	
for(initial statement; condition; subsequent statement)	
primary statement	

8.2 Specification of syntax functions and descriptors

The functions presented in subclause 8.2 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 is 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 D for applications that use the sample stream format).

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.

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).

— fl(n): binary floating point value using n bits. The parsing process for this descriptor is as specified in IEEE 754-2019.

— 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 the most significant (left) 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 subclause 14.2.

— 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. When n is equal to 0, the associated syntax element is not present in the bitstream and its corresponding value shall be inferred to be equal to 0 unless specified otherwise in the semantics.

— ue(v): unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 14.2.

8.3 Syntax in tabular form

8.3.1 General

The V3C bitstream is composed of a set of V3C units as shown in Figure 5. Each V3C unit has a V3C unit header and a V3C unit payload. The V3C unit header describes the V3C unit type. Table 2 describes the supported V3C unit types.

The attribute video data V3C unit header specifies also the index of the attribute, which allows identification of the attribute type based on VPS information, and the partition index, which enables an attribute that consists of multiple components to be segmented into smaller component partition units. Such segmentation allows such attribute types to be coded using legacy coding specifications that may be limited in terms of the number of components that they can support.

The occupancy, geometry and attribute video data unit payloads correspond to video data units (e.g. NAL units defined in ISO/IEC 23008-2) that could be decoded by an appropriate video decoder as indicated by the profiles defined in Annex A and, if present, the component codec mapping SEI message defined in subclause F.3.11.

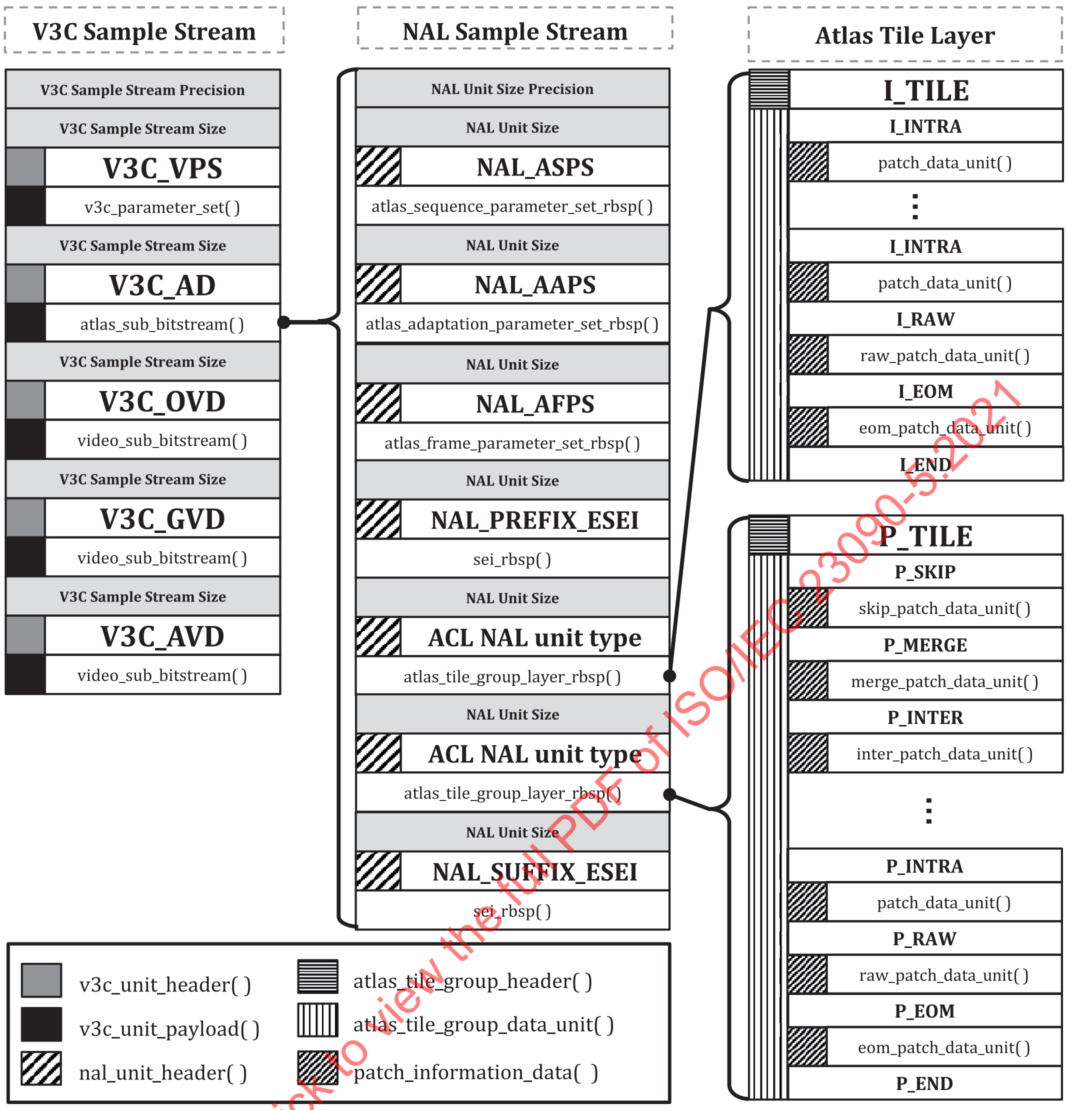


Figure 5 — Overview of V3C bitstream structure

8.3.2 V3C unit syntax

8.3.2.1 General V3C unit syntax

v3c_unit(numBytesInV3CUnit) {	**Descriptor**
v3c_unit_header( )	
v3c_unit_payload(numBytesInV3CUnit − 4)	
}	

8.3.2.2 V3C unit header syntax

v3c_unit_header( ) {	**Descriptor**
vuh_unit_type	u(5)

if(vuh_unit_type == V3C_AVD || vuh_unit_type == V3C_GVD ||
  vuh_unit_type == V3C_OVD || vuh_unit_type == V3C_AD) {	

vuh_v3c_parameter_set_id	u(4)

vuh_atlas_id	u(6)
}	
if(vuh_unit_type == V3C_AVD) {	
vuh_attribute_index	u(7)
vuh_attribute_partition_index	u(5)
vuh_map_index	u(4)
vuh_auxiliary_video_flag	u(1)
} else if(vuh_unit_type == V3C_GVD) {	
vuh_map_index	u(4)
vuh_auxiliary_video_flag	u(1)
vuh_reserved_zero_12bits	u(12)
} else if(vuh_unit_type == V3C_OVD || vuh_unit_type == V3C_AD)	
vuh_reserved_zero_17bits	u(17)
else	
vuh_reserved_zero_27bits	u(27)
}	

8.3.2.3 V3C unit payload syntax

v3c_unit_payload(numBytesInV3CPayload) {	**Descriptor**
if(vuh_unit_type == V3C_VPS)	
v3c_parameter_set(numBytesInV3CPayload)	
else if(vuh_unit_type == V3C_AD)	
atlas_sub_bitstream(numBytesInV3CPayload)	

else if(vuh_unit_type == V3C_OVD ||
            vuh_unit_type == V3C_GVD ||
            vuh_unit_type == V3C_AVD)	

video_sub_bitstream(numBytesInV3CPayload)	
}	

8.3.2.4 Atlas sub-bitstream syntax

atlas_sub_bitstream(numBytes) {	
sample_stream_nal_header( )	
numBytes−−	
while(numBytes > 0) {	
sample_stream_nal_unit( )	
numBytes −= ssnu_nal_unit_size + ssnh_unit_size_precision_bytes_minus1 + 1	
}	
}	

8.3.3 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)
}	

8.3.4 V3C parameter set syntax

8.3.4.1 General V3C parameter set syntax

v3c_parameter_set(numBytesInV3CPayload) {	**Descriptor**
profile_tier_level( )	
vps_v3c_parameter_set_id	u(4)
vps_reserved_zero_8bits	u(8)
vps_atlas_count_minus1	u(6)
for(k = 0; k < vps_atlas_count_minus1 + 1; k++) {	
vps_atlas_id[k]	u(6)
j = vps_atlas_id[k]	
vps_frame_width[j]	ue(v)
vps_frame_height[j]	ue(v)
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	
vps_map_predictor_index_diff[j][0] = 0	
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) {	
vps_map_predictor_index_diff[j][i]	ue(v)
}	
}	
vps_auxiliary_video_present_flag[j]	u(1)
vps_occupancy_video_present_flag[j]	u(1)
vps_geometry_video_present_flag[j]	u(1)
vps_attribute_video_present_flag[j]	u(1)
if(vps_occupancy_video_present_flag[j])	
occupancy_information(j)	
if(vps_geometry_video_present_flag[j])	
geometry_information(j)	
if(vps_attribute_video_present_flag[j])	
attribute_information(j)	
}	
vps_extension_present_flag	u(1)
if(vps_extension_present_flag)	
vps_extension_8bits	u(8)

if(vps_extension_8bits) {	
vps_extension_length_minus1	ue(v)
for(j = 0; j < vps_extension_length_minus1 + 1; j++) {	
vps_extension_data_byte	u(8)
}	
}	
byte_alignment( )	
}	

8.3.4.2 Profile, tier and level syntax

profile_tier_level( ) {	**Descriptor**
ptl_tier_flag	u(1)
ptl_profile_codec_group_idc	u(7)
ptl_profile_toolset_idc	u(8)
ptl_profile_reconstruction_idc	u(8)
ptl_reserved_zero_16bits	u(16)
ptl_reserved_0xffff_16bits	u(16)
ptl_level_idc	u(8)
ptl_num_sub_profiles	u(6)
ptl_extended_sub_profile_flag	u(1)
for(i = 0; i < ptl_num_sub_profiles; i++)	
ptl_sub_profile_idc[i]	u(v)
ptl_toolset_constraints_present_flag	u(1)
if(ptl_toolset_constraints_present_flag)	
profile_toolset_constraints_information( )	
}	

8.3.4.3 Occupancy information syntax

occupancy_information(atlasID) {	**Descriptor**
oi_occupancy_codec_id[atlasID]	u(8)
oi_lossy_occupancy_compression_threshold[atlasID]	u(8)
oi_occupancy_2d_bit_depth_minus1[atlasID]	u(5)
oi_occupancy_MSB_align_flag[atlasID]	u(1)
}	

8.3.4.4 Geometry information syntax

geometry_information(atlasID) {	**Descriptor**
gi_geometry_codec_id[atlasID]	u(8)
gi_geometry_2d_bit_depth_minus1[atlasID]	u(5)
gi_geometry_MSB_align_flag[atlasID]	u(1)
gi_geometry_3d_coordinates_bit_depth_minus1[atlasID]	u(5)
if(vps_auxiliary_video_present_flag[atlasID])	
gi_auxiliary_geometry_codec_id[atlasID]	u(8)

}	

8.3.4.5 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_auxiliary_video_present_flag[atlasID])	
ai_auxiliary_attribute_codec_id[atlasID][i]	u(8)
if(vps_map_count_minus1[atlasID] > 0)	
ai_attribute_map_absolute_coding_persistence_flag[atlasID][i]	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_2d_bit_depth_minus1[atlasID][i]	u(5)
ai_attribute_MSB_align_flag[atlasID][i]	u(1)
}	
}	

8.3.4.6 Profile toolset constraints information syntax

profile_toolset_constraints_information( ) {	**Descriptor**
ptc_one_v3c_frame_only_flag	u(1)
ptc_eom_constraint_flag	u(1)
ptc_max_map_count_minus1	u(4)
ptc_max_atlas_count_minus1	u(4)
ptc_multiple_map_streams_constraint_flag	u(1)
ptc_plr_constraint_flag	u(1)
ptc_attribute_max_dimension_minus1	u(6)
ptc_attribute_max_dimension_partitions_minus1	u(6)
ptc_no_eight_orientations_constraint_flag	u(1)

ptc_no_45degree_projection_patch_constraint_flag	u(1)
ptc_reserved_zero_6bits	u(6)
ptc_num_reserved_constraint_bytes	u(8)
for(i = 0; i < ptc_num_reserved_constraint_bytes; i++)	
ptc_reserved_constraint_byte[i]	u(8)
}	

8.3.5 NAL unit syntax

8.3.5.1 General NAL unit syntax

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

8.3.5.2 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_temporal_id_plus1	u(3)
}	

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

8.3.6.1 Atlas sequence parameter set RBSP syntax

8.3.6.1.1 General atlas sequence parameter set RBSP syntax

atlas_sequence_parameter_set_rbsp( ) {	**Descriptor**
asps_atlas_sequence_parameter_set_id	ue(v)
asps_frame_width	ue(v)
asps_frame_height	ue(v)
asps_geometry_3d_bit_depth_minus1	u(5)
asps_geometry_2d_bit_depth_minus1	u(5)
asps_log2_max_atlas_frame_order_cnt_lsb_minus4	ue(v)
asps_max_dec_atlas_frame_buffering_minus1	ue(v)
asps_long_term_ref_atlas_frames_flag	u(1)
asps_num_ref_atlas_frame_lists_in_asps	ue(v)
for(i = 0; i < asps_num_ref_atlas_frame_lists_in_asps; i++)	
ref_list_struct(i)	
asps_use_eight_orientations_flag	u(1)
asps_extended_projection_enabled_flag	u(1)

if(asps_extended_projection_enabled_flag)	
asps_max_number_projections_minus1	ue(v)
asps_normal_axis_limits_quantization_enabled_flag	u(1)
asps_normal_axis_max_delta_value_enabled_flag	u(1)
asps_patch_precedence_order_flag	u(1)
asps_log2_patch_packing_block_size	u(3)
asps_patch_size_quantizer_present_flag	u(1)
asps_map_count_minus1	u(4)
asps_pixel_deinterleaving_enabled_flag	u(1)
if(asps_pixel_deinterleaving_enabled_flag)	
for(j = 0; j <= asps_map_count_minus1; j++)	
asps_map_pixel_deinterleaving_flag[j]	u(1)
asps_raw_patch_enabled_flag	u(1)
asps_eom_patch_enabled_flag	u(1)
if(asps_eom_patch_enabled_flag && asps_map_count_minus1 == 0)	
asps_eom_fix_bit_count_minus1	u(4)
if(asps_raw_patch_enabled_flag || asps_eom_patch_enabled_flag)	
asps_auxiliary_video_enabled_flag	u(1)
asps_plr_enabled_flag	u(1)
if(asps_plr_enabled_flag)	
asps_plr_information(asps_map_count_minus1)	
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) {	
asps_vpcc_extension_present_flag	u(1)
asps_extension_7bits	u(7)
}	
if(asps_vpcc_extension_present_flag)	
asps_vpcc_extension( ) /* Specified in Annex H*/	
if(asps_extension_7bits)	
while(more_rbsp_data( ))	
asps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

8.3.6.1.2 Point local reconstruction information syntax

asps_plr_information(mapCountMinus1) {	**Descriptor**
for(i = 0; i < mapCountMinus1 + 1; i++) {	
plri_map_present_flag[i]	u(1)
if(plri_map_present_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[i]	u(6)
}	
}	
}	

8.3.6.2 Atlas frame parameter set RBSP syntax

8.3.6.2.1 General 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_output_flag_present_flag	u(1)
afps_num_ref_idx_default_active_minus1	ue(v)
afps_additional_lt_afoc_lsb_len	ue(v)
afps_lod_mode_enabled_flag	u(1)
afps_raw_3d_offset_bit_count_explicit_mode_flag	u(1)
afps_extension_present_flag	u(1)
if(afps_extension_present_flag)	
afps_extension_8bits	u(8)
if(afps_extension_8bits)	
while(more_rbsp_data( ))	
afps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

8.3.6.2.2 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_partition_spacing_flag	u(1)
if(afti_uniform_partition_spacing_flag) {	
afti_partition_cols_width_minus1	ue(v)
afti_partition_rows_height_minus1	ue(v)
} else {	
afti_num_partition_columns_minus1	ue(v)
afti_num_partition_rows_minus1	ue(v)
for(i = 0; i < afti_num_partition_columns_minus1; i++)	

afti_partition_column_width_minus1[i]	ue(v)
for(i = 0; i < afti_num_partition_rows_minus1; i++)	
afti_partition_row_height_minus1[i]	ue(v)
}	
afti_single_partition_per_tile_flag	u(1)
if(!afti_single_partition_per_tile_flag) {	
afti_num_tiles_in_atlas_frame_minus1	ue(v)
for(i = 0; i < afti_num_tiles_in_atlas_frame_minus1 + 1; i++) {	
afti_top_left_partition_idx[i]	u(v)
afti_bottom_right_partition_column_offset[i]	ue(v)
afti_bottom_right_partition_row_offset[i]	ue(v)
}	
}	
else	
afti_num_tiles_in_atlas_frame_minus1 = NumPartitionsInAtlasFrame − 1	
}	
else	
afti_num_tiles_in_atlas_frame_minus1 = 0	
if(asps_auxiliary_video_enabled_flag) {	
afti_auxiliary_video_tile_row_width_minus1	ue(v)
for(i = 0; i < afti_num_tiles_in_atlas_frame_minus1 + 1; i++)	
afti_auxiliary_video_tile_row_height[i]	ue(v)
}	
afti_signalled_tile_id_flag	u(1)
if(afti_signalled_tile_id_flag) {	
afti_signalled_tile_id_length_minus1	ue(v)
for(i = 0; i < afti_num_tiles_in_atlas_frame_minus1 + 1; i++) {	
afti_tile_id[i]	u(v)
TileIDToIndex[afti_tile_id[i]] = i	
TileIndexToID[i] = afti_tile_id[i]	
}	
}	
else	
for(i = 0; i < afti_num_tiles_in_atlas_frame_minus1 + 1; i++) {	
afti_tile_id[i] = i	
TileIDToIndex[i] = i	
TileIndexToID[i] = i	
}	
}	

8.3.6.3 Atlas adaptation parameter set RBSP syntax

atlas_adaptation_parameter_set_rbsp( ) {	**Descriptor**
aaps_atlas_adaptation_parameter_set_id	ue(v)
aaps_extension_present_flag	u(1)

if(aaps_extension_present_flag) {	
aaps_vpcc_extension_present_flag	u(1)
aaps_extension_7bits	u(7)
}	
if(aaps_vpcc_extension_present_flag)	
aaps_vpcc_extension( ) /* Specified in Annex H*/	
if(aaps_extension_7bits)	
while(more_rbsp_data( ))	
aaps_extension_data_flag	u(1)
rbsp_trailing_bits( )	
}	

8.3.6.4 Supplemental enhancement information RBSP syntax

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

8.3.6.5 Access unit delimiter RBSP syntax

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

8.3.6.6 End of sequence RBSP syntax

end_of_sequence_rbsp( ) {	**Descriptor**
}	

8.3.6.7 End of bitstream RBSP syntax

end_of_atlas_sub_bitstream_rbsp( ) {	**Descriptor**
}	

8.3.6.8 Filler data RBSP syntax

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

8.3.6.9 Atlas tile layer RBSP syntax

atlas_tile_layer_rbsp( ) {	**Descriptor**
atlas_tile_header( )	
atlas_tile_data_unit(tileID)	
rbsp_trailing_bits( )	
}	

8.3.6.10 RBSP trailing bit syntax

rbsp_trailing_bits( ) {	**Descriptor**
rbsp_stop_one_bit /* equal to 1 */	f(1)
while(!byte_aligned( ))	
rbsp_alignment_zero_bit /* equal to 0 */	f(1)
}	

8.3.6.11 Atlas tile header syntax

atlas_tile_header( ) {	**Descriptor**
if(nal_unit_type >= NAL_BLA_W_LP && nal_unit_type <= NAL_RSV_IRAP_ACL_29)	
ath_no_output_of_prior_atlas_frames_flag	u(1)
ath_atlas_frame_parameter_set_id	ue(v)
ath_atlas_adaptation_parameter_set_id	ue(v)
ath_id	u(v)
tileID = ath_id	
ath_type	ue(v)
if(afps_output_flag_present_flag)	
ath_atlas_output_flag	u(1)
ath_atlas_frm_order_cnt_lsb	u(v)
if(asps_num_ref_atlas_frame_lists_in_asps > 0)	
ath_ref_atlas_frame_list_asps_flag	u(1)
if(ath_ref_atlas_frame_list_asps_flag == 0)	
ref_list_struct(asps_num_ref_atlas_frame_lists_in_asps)	
else if(asps_num_ref_atlas_frame_lists_in_asps > 1)	
ath_ref_atlas_frame_list_idx	u(v)
for(j = 0; j < NumLtrAtlasFrmEntries[RlsIdx]; j++) {	
ath_additional_afoc_lsb_present_flag[j]	u(1)
if(ath_additional_afoc_lsb_present_flag[j])	
ath_additional_afoc_lsb_val[j]	u(v)
}	
if(ath_type != SKIP_TILE) {	
if(asps_normal_axis_limits_quantization_enabled_flag) {	
ath_pos_min_d_quantizer	u(5)
if(asps_normal_axis_max_delta_value_enabled_flag)	
ath_pos_delta_max_d_quantizer	u(5)
}	

if(asps_patch_size_quantizer_present_flag) {	
ath_patch_size_x_info_quantizer	u(3)
ath_patch_size_y_info_quantizer	u(3)
}	
if(afps_raw_3d_offset_bit_count_explicit_mode_flag)	
ath_raw_3d_offset_axis_bit_count_minus1	u(v)
if(ath_type == P_TILE && num_ref_entries[RlsIdx] > 1) {	
ath_num_ref_idx_active_override_flag	u(1)
if(ath_num_ref_idx_active_override_flag)	
ath_num_ref_idx_active_minus1	ue(v)
}	
}	
byte_alignment( )	
}	

8.3.6.12 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)	
straf_entry_sign_flag[rlsIdx][i]	u(1)
} else	
afoc_lsb_lt[rlsIdx][i]	u(v)
}	
}	

8.3.7 Atlas tile data unit syntax

8.3.7.1 General atlas tile data unit syntax

atlas_tile_data_unit(tileID) {	**Descriptor**
if(ath_type == SKIP_TILE) {	
for(p = 0; p < RefAtduTotalNumPatches[tileID]; p++)	
skip_patch_data_unit( )	
} else {	
p = 0	
do {	
atdu_patch_mode[tileID][p]	ue(v)
isEnd = (ath_type == P_TILE && atdu_patch_mode[tileID][p] == P_END) || (ath_type == I_TILE && atdu_patch_mode[tileID][p] == I_END)	
if(!isEnd) {	

patch_information_data(tileID , p , atdu_patch_mode[tileID][p])	
p++	
}	
} while(!isEnd)	
}	
AtduTotalNumPatches[tileID] = p	
}	

8.3.7.2 Patch information data syntax

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

8.3.7.3 Patch data unit syntax

patch_data_unit(tileID, patchIdx) {	**Descriptor**
pdu_2d_pos_x[tileID][patchIdx]	ue(v)
pdu_2d_pos_y[tileID][patchIdx]	ue(v)
pdu_2d_size_x_minus1[tileID][patchIdx]	ue(v)
pdu_2d_size_y_minus1[tileID][patchIdx]	ue(v)
pdu_3d_offset_u[tileID][patchIdx]	u(v)
pdu_3d_offset_v[tileID][patchIdx]	u(v)
pdu_3d_offset_d[tileID][patchIdx]	u(v)

if(asps_normal_axis_max_delta_value_enabled_flag)	
pdu_3d_range_d[tileID][patchIdx]	u(v)
pdu_projection_id[tileID][patchIdx]	u(v)
pdu_orientation_index[tileID][patchIdx]	u(v)
if(afps_lod_mode_enabled_flag) {	
pdu_lod_enabled_flag[tileID][patchIdx]	u(1)
if(pdu_lod_enabled_flag[tileID][patchIdx] > 0) {	
pdu_lod_scale_x_minus1[tileID][patchIdx]	ue(v)
pdu_lod_scale_y_idc[tileID][patchIdx]	ue(v)
}	
}	
if(asps_plr_enabled_flag)	
plr_data(tileID, patchIdx)	
}	

8.3.7.4 Skip patch data unit syntax

skip_patch_data_unit( ) {	**Descriptor**
}	

8.3.7.5 Merge patch data unit syntax

merge_patch_data_unit(tileID, patchIdx) {	**Descriptor**
if(NumRefIdxActive > 1)	
mpdu_ref_index[tileID][patchIdx]	ue(v)
OverridePlrFlag = 0	
mpdu_override_2d_params_flag[tileID][patchIdx]	u(1)
if(mpdu_override_2d_params_flag[tileID][patchIdx]) {	
mpdu_2d_pos_x[tileID][patchIdx]	se(v)
mpdu_2d_pos_y[tileID][patchIdx]	se(v)
mpdu_2d_delta_size_x[tileID][patchIdx]	se(v)
mpdu_2d_delta_size_y[tileID][patchIdx]	se(v)
if(asps_plr_enabled_flag)	
OverridePlrFlag = 1	
} else {	
mpdu_override_3d_params_flag[tileID][patchIdx]	u(1)
if(mpdu_override_3d_params_flag[tileID][patchIdx]) {	
mpdu_3d_offset_u[tileID][patchIdx]	se(v)
mpdu_3d_offset_v[tileID][patchIdx]	se(v)
mpdu_3d_offset_d[tileID][patchIdx]	se(v)
if(asps_normal_axis_max_delta_value_enabled_flag)	
mpdu_3d_range_d[tileID][patchIdx]	se(v)
if(asps_plr_enabled_flag) {	
mpdu_override_plr_flag[tileID][patchIdx]	u(1)
OverridePlrFlag = mpdu_override_plr_flag[tileID][patchIdx]	

}	
}	
}	
if(OverridePlrFlag && asps_plr_enabled_flag)	
plr_data(tileID, patchIdx)	
}	

8.3.7.6 Inter patch data unit syntax

inter_patch_data_unit(tileID, patchIdx) {	**Descriptor**
if(NumRefIdxActive > 1)	
ipdu_ref_index[tileID][patchIdx]	ue(v)
ipdu_patch_index[tileID][patchIdx]	se(v)
ipdu_2d_pos_x[tileID][patchIdx]	se(v)
ipdu_2d_pos_y[tileID][patchIdx]	se(v)
ipdu_2d_delta_size_x[tileID][patchIdx]	se(v)
ipdu_2d_delta_size_y[tileID][patchIdx]	se(v)
ipdu_3d_offset_u[tileID][patchIdx]	se(v)
ipdu_3d_offset_v[tileID][patchIdx]	se(v)
ipdu_3d_offset_d[tileID][patchIdx]	se(v)
if(asps_normal_axis_max_delta_value_enabled_flag)	
ipdu_3d_range_d[tileID][patchIdx]	se(v)
if(asps_plr_enabled_flag)	
plr_data(tileID, patchIdx)	
}	

8.3.7.7 RAW patch data unit syntax

raw_patch_data_unit(tileID, patchIdx) {	**Descriptor**
if(AuxTileHeight[TileIDToIndex[tileID]] > 0)	
rpdu_patch_in_auxiliary_video_flag[tileID][patchIdx]	u(1)
rpdu_2d_pos_x[tileID][patchIdx]	ue(v)
rpdu_2d_pos_y[tileID][patchIdx]	ue(v)
rpdu_2d_size_x_minus1[tileID][patchIdx]	ue(v)
rpdu_2d_size_y_minus1[tileID][patchIdx]	ue(v)
rpdu_3d_offset_u[tileID][patchIdx]	u(v)
rpdu_3d_offset_v[tileID][patchIdx]	u(v)
rpdu_3d_offset_d[tileID][patchIdx]	u(v)
rpdu_points_minus1[tileID][patchIdx]	ue(v)
}	

8.3.7.8 EOM patch data unit syntax

eom_patch_data_unit(tileID, patchIdx) {	**Descriptor**
if(AuxTileHeight[TileIDToIndex[tileID]] > 0)	
epdu_patch_in_auxiliary_video_flag[tileID][patchIdx]	u(1)

epdu_2d_pos_x[tileID][patchIdx]	ue(v)
epdu_2d_pos_y[tileID][patchIdx]	ue(v)
epdu_2d_size_x_minus1[tileID][patchIdx]	ue(v)
epdu_2d_size_y_minus1[tileID][patchIdx]	ue(v)
epdu_patch_count_minus1[tileID][patchIdx]	ue(v)
for(i = 0; i < epdu_patch_count_minus1[tileID][patchIdx] + 1; i++) {	
epdu_associated_patch_idx[tileID][patchIdx][i]	ue(v)
epdu_points[tileID][patchIdx][i]	ue(v)
}	
}	

8.3.7.9 Point local reconstruction data syntax

plr_data(tileID, patchIdx) {	**Descriptor**
blockCnt = BlockCnt(TilePatch2dSizeX[tileID][p], TilePatch2dSizeY[tileID][p])	
for(i = 0; i < asps_map_count_minus1 + 1; i++) {	
if(plri_map_present_flag[i]) {	
if(blockCnt > plri_block_threshold_per_patch_minus1[i] + 1)	
plrd_level[tileID][patchIdx][i]	u(1)
else	
plrd_level[tileID][patchIdx][i] = 1	
if(plrd_level[tileID][patchIdx][i] == 0) {	
for(j = 0; j < blockCnt; j++) {	
plrd_present_block_flag[tileID][patchIdx][i][j]	u(1)
if(plrd_present_block_flag[tileID][patchIdx][i][j])	
plrd_block_mode_minus1[tileID][patchIdx][i][j]	u(v)
}	
} else {	
plrd_present_flag[tileID][patchIdx][i]	u(1)
if(plrd_present_flag[tileID][patchIdx][i])	
plrd_mode_minus1[tileID][patchIdx][i]	u(v)
}	
}	
}	
}	

8.3.8 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)	
}	

8.4 Semantics

8.4.1 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 document.

For some applications, different encapsulation of a V3C bitstream may be used that may not utilize the V3C unit structure. As an example, in ISO/IEC DIS 23090-10[1)], the information provided by the V3C unit and V3C parameter sets may be provided through equivalent structures and the V3C unit structure may not be directly present. In such applications, all the syntax elements specified in subclauses 8.3.2 and 8.3.4, such as vuh_unit_type, vuh_atlas_id, vuh_map_index, vps_atlas_count_minus1, etc., may be obtained through external means.

8.4.2 V3C unit semantics

8.4.2.1 General V3C unit semantics

trailing_zero_8bits is a byte equal to 0x00.

8.4.2.2 V3C unit header semantics

Table 2 — V3C unit types

vuh_unit_type	Identifier	V3C unit type	Description
0	V3C_VPS	V3C parameter set	V3C level parameters
1	V3C_AD	Atlas data	Atlas information
2	V3C_OVD	Occupancy video data	Occupancy information
3	V3C_GVD	Geometry video data	Geometry information
4	V3C_AVD	Attribute video data	Attribute information
5…31	V3C_RSVD	Reserved	-

vuh_unit_type indicates the V3C unit type as specified in Table 2. Values indicated as reserved are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this edition of this document. Decoders conforming to this edition of this document should ignore such reserved unit types.

vuh_v3c_parameter_set_id specifies the value of vps_v3c_parameter_set_id for the active V3C VPS. The value of vuh_v3c_parameter_set_id shall be in the range of 0 to 15, inclusive.

vuh_atlas_id specifies the ID of the atlas that corresponds to the current V3C unit. The value of vuh_atlas_id shall be in the range of 0 to 63, inclusive.

1) Under preparation. Stage at time of publication: ISO/IEC DIS 23090-10:2020.

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_partition_index indicates the index of the attribute dimension group carried in the attribute video data unit. The value of vuh_attribute_partition_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 sub-bitstream is derived based on the type of the sub-bitstream and the operations described in subclauses 9.4 and 9.5 for geometry and attribute video sub-bitstreams 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_auxiliary_video_flag equal to 1 indicates that the associated geometry or attribute video data unit is a RAW or EOM coded points video only sub-bitstream, or both. vuh_auxiliary_video_flag equal to 0 indicates that the associated geometry or attribute video data unit may contain RAW or EOM coded points, or both. When vuh_auxiliary_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 edition of this document. 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 edition of this document. 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 edition of this document. 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.

8.4.2.3 V3C unit payload semantics

video_sub_bitstream(numBytes) contains a portion of a video unit stream of size numBytes as an ordered stream of bytes or bits within which the locations of unit boundaries are identifiable from patterns in the data. The format of such video unit stream is identified by a 4CC code as defined by ptl_profile_codec_group_idc or by a component codec mapping SEI message.

8.4.2.4 Atlas sub-bitstream semantics

atlas_sub_bitstream(numBytes) contains a portion of an atlas NAL unit stream of size numBytes as an ordered stream of bytes or bits within which the locations of atlas NAL unit boundaries are identifiable from patterns in the data. The format of such atlas NAL unit stream is defined in Annex D.

8.4.2.5 Order of V3C units and association to coded information

8.4.2.5.1 General

This subclause specifies constraints on the order of V3C units in the bitstream.

8.4.2.5.2 Order of VPSs and their activation

This subclause specifies the activation process of V3C parameter sets (VPSs)

NOTE The VPS mechanism decouples the transmission of infrequently changing information from the transmission of V3C units. VPSs can, in some applications, be conveyed "out-of-band".

A VPS includes parameters that can be referred to by one or more V3C units and their associated vuh_v3c_parameter_set_id. Each VPS is initially considered not active at the start of the operation of the decoding process. At most, one VPS is considered active at any given moment during the operation of the decoding process and the activation of any particular VPS results in the deactivation of the previously active VPS.

When a VPS (with a particular value of vps_v3c_parameter_set_id) is not already active for a particular V3C unit and it is referred to by a V3C unit (in which the associated vuh_v3c_parameter_set_id is equal to vps_v3c_parameter_set_id), it is activated for the particular V3C unit. This VPS is called the active VPS for the particular V3C unit until it is deactivated by the activation of another VPS. A VPS, with that particular value of vps_v3c_parameter_set_id, shall be available to the decoding process prior to its activation, included in at least one V3C unit or provided through external means. An activated VPS shall remain active for the entire coded V3C sequence (CVS). The activation of a new VPS starts a new CVS.

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 VPSs and other syntax elements are expressions of constraints that apply only to the active VPS. If any VPS 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 9), the values of parameters of the active VPS are considered in effect.

8.4.2.5.3 Order of V3C units and association to CVSs

A V3C bitstream conforming to this document consists of one or more CVSs. A CVS starts with a VPS, included in at least one V3C unit or provided through external means, and contains one or more V3C units that can be factored into V3C composition units. The first V3C composition unit in a CVS shall be an IRAP composition unit.

A CVS consists of multiple V3C sub-bitstreams, with each V3C sub-bitstream associated with a V3C component. For each component, its first corresponding V3C sub-bitstream shall start with a corresponding IRAP sub-bitstream unit.

For atlas sub-bitstreams the IRAP sub-bitstream unit corresponds to a coded atlas access unit for which each ACL NAL 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.

For video sub-bitstreams, the IRAP sub-bitstream unit is defined by the respective video specification and it is outside the scope of this document. For example, for ISO/IEC 14496-10 an IRAP sub-bitstream unit can be an IDR or a CRA access unit as defined in that document.

V3C units with different V3C unit headers may be decoded in parallel. V3C units with the same V3C unit headers can be decoded as if the associated sub-bitstreams are concatenated in the order received.

A V3C bitstream in general contains multiple sub-bitstreams. Each sub-bitstream is partitioned into sub-bitstream composition units.

8.4.3 Byte alignment semantics

alignment_bit_equal_to_one shall be equal to 1.

alignment_bit_equal_to_zero shall be equal to 0.

8.4.4 V3C parameter set semantics

8.4.4.1 General V3C parameter set semantics

vps_v3c_parameter_set_id provides an identifier for the V3C VPS for reference by other syntax elements. The value of vps_v3c_parameter_set_id shall be in the range of 0 to 15, inclusive.

vps_reserved_zero_8bits, when present, shall be equal to 0 in bitstreams conforming to this edition of this document. Other values for vps_reserved_zero_8bits are reserved for future use ISO/IEC. Decoders shall ignore the value of vps_reserved_zero_8bits.

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_atlas_id[k] specifies the ID of the atlas with index k. The value of vps_atlas_id[k] shall be in the range of 0 to 63, inclusive. It is a requirement of bitstream conformance to this edition of this document that the value of vps_atlas_id[k] shall not be equal to vps_atlas_id[j] for all j != k.

vps_frame_width[j] indicates the V3C frame width in terms of integer luma samples for the atlas with atlas ID j. This frame width is the nominal width that is associated with all V3C components for the atlas with atlas ID j.

vps_frame_height[j] indicates the V3C frame height in terms of integer luma samples for the atlas with atlas ID j. This frame height is the nominal height that is associated with all V3C components for the atlas with atlas ID 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 atlas ID 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 atlas ID 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 atlas ID 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 atlas ID 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 atlas ID 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 atlas ID j when vps_map_absolute_coding_enabled_flag[j][i] is equal to 0. More specifically, the map predictor index for map i, MapPredictorIndex[i], shall be computed as:

$$\text{MapPredictorIndex}[\,i\,] = (i - 1) - \text{vps\_map\_predictor\_index\_diff}[\,j\,][\,i\,] \qquad (15)$$

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_auxiliary_video_present_flag[j] equal to 1 indicates that additional information, i.e. information associated with RAW or EOM patch types, for a patch in the atlas with atlas ID j may be stored in a separate video stream, referred to as the auxiliary video stream. vps_auxiliary_video_present_flag[j] equal to 0 indicates that additional information, i.e. information associated with RAW or EOM patch types, for a patch in the atlas with atlas ID j shall not be stored in the auxiliary video stream. When vps_auxiliary_video_present_flag[j] is not present, it is inferred to be equal to 0.

vps_occupancy_video_present_flag[j] equal to 0 indicates that the atlas with atlas ID j does not have occupancy video data associated with it. vps_occupancy_video_present_flag[j] equal to 1 indicates that the atlas with atlas ID j shall have occupancy video data associated with it. When vps_occupancy_video_present_flag[j] is not present, it is inferred to be equal to 1.

vps_geometry_video_present_flag[j] equal to 0 indicates that the atlas with atlas ID j does not have geometry video data associated with it. vps_geometry_video_present_flag[j] equal to 1 indicates that the atlas with atlas ID j shall have geometry video data associated with it. When vps_geometry_video_present_flag[j] is not present, it is inferred to be equal to 1.

vps_attribute_video_present_flag[j] equal to 0 indicates that the atlas with atlas ID j does not have attribute video data associated with it. vps_attribute_video_present_flag[j] equal to 1 indicates that the atlas with atlas ID j shall have at least one or more attribute video data associated with it. When vps_attribute_video_present_flag[j] is not present, it is inferred to be equal to 1.

vps_extension_present_flag equal to 1 specifies that the syntax element vps_extension_8bits is present in the v3c_parameter_set( ) syntax structure. vps_extension_present_flag equal to 0 specifies that the syntax element vps_extension_8bits is not present. vps_extension_present_flag shall be equal to 0 in bitstreams conforming to this edition of this document.

vps_extension_8bits not equal to 0 specifies that the syntax element vps_extension_length is present in the v3c_parameter_set( ) syntax structure. vps_extension_8bits equal to 0 specifies that the syntax element vps_extension_length_minus1 is not present. vps_extension_8bits shall be equal to 0 in bitstreams conforming to this edition of this document. Other values of vps_extension_8bits are reserved for future use by ISO/IEC.

vps_extension_length_minus1 plus 1 specifies the number of vps_extension_data_byte elements that follow this syntax element.

vps_extension_data_byte may have any value.

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

ptl_profile_reconstruction_idc indicates the reconstruction profile component to which the CVS is recommended to conform to as specified in Annex A. 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_16bits, when present, shall be equal to 0 in bitstreams conforming to this edition of this document. Other values for ptl_reserved_zero_16bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ptl_reserved_zero_16bits.

ptl_reserved_0xffff_16bits, when present, shall be equal to 0xFFFF in bitstreams conforming to this edition of this document. Other values for ptl_reserved_0xffff_16bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ptl_reserved_0xffff_16bits.

ptl_level_idc indicates a level to which the CVS 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.

ptl_num_sub_profiles indicates the number of the ptl_sub_profile_idc[i] syntax elements.

ptl_extended_sub_profile_flag equal to 1 specifies that the ptl_sub_profile_idc[i] syntax elements, if present, should be represented using 64 bits. ptl_extended_sub_profile_flag equal to 0 specifies that the ptl_sub_profile_idc[i] syntax elements, if present, should be represented using 32 bits.

ptl_sub_profile_idc[i] indicates the i-th interoperability metadata registered as specified by Rec. ITU-T T.35, the content of which is not specified in this document. The number of bits used to represent ptl_sub_profile_idc[i] is equal to (ptl_extended_sub_profile_flag == 0 ? 32 : 64).

ptl_toolset_constraints_present_flag equal to 1 specifies that an additional structure, profile_toolset_constraints_information( ), is present in the bitstream. ptl_toolset_constraints_present_flag equal to 0 specifies that the structure profile_toolset_constraints_information( ) is not present.

8.4.4.3 Occupancy information semantics

oi_occupancy_codec_id[j] indicates the identifier of the codec used to compress the occupancy information for the atlas with atlas ID j. oi_occupancy_codec_id[j] shall be in the range of 0 to 255, inclusive. This codec may be identified through the profiles defined in Annex A, a component codec mapping SEI message, or through means outside this document.

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

oi_occupancy_2d_bit_depth_minus1[j] plus 1 indicates the nominal 2D bit depth to which the occupancy video for the atlas with atlas ID j shall be converted to. oi_occupancy_2d_bit_depth_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 atlas ID j are converted to samples at the nominal occupancy bit depth, as specified in Annex B.

8.4.4.4 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 atlas ID j. gi_geometry_codec_id[j] shall be in the range of 0 to 255, inclusive. This codec may be identified through the profiles defined in Annex A, a component codec mapping SEI message, or through means outside this document.

gi_geometry_2d_bit_depth_minus1[j] plus 1 indicates the nominal 2D bit depth to which all geometry videos for the atlas with atlas ID j shall be converted to. gi_geometry_2d_bit_depth_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 atlas ID j are converted to samples at the nominal geometry bit depth, as specified in Annex B.

gi_geometry_3d_coordinates_bit_depth_minus1[j] plus 1 indicates the bit depth of the geometry coordinates of the reconstructed volumetric content for the atlas with atlas ID j. gi_geometry_3d_coordinates_bit_depth_minus1[j] shall be in the range of 0 to 31, inclusive.

gi_auxiliary_geometry_codec_id[j], when present, indicates the identifier of the codec used to compress the geometry video data sub-bitstreams, when RAW coded points are encoded in an auxiliary video stream for the atlas with atlas ID j. gi_auxiliary_geometry_codec_id[j] shall be in the range of 0 to 255, inclusive. This codec may be identified through the profiles defined in Annex A, a component codec mapping SEI message, or through means outside this document. When not present the value of gi_auxiliary_geometry_codec_id[j] is inferred to be equal to gi_geometry_codec_id[j].

8.4.4.5 Attribute information semantics

ai_attribute_count[j] indicates the number of attributes associated with the atlas with atlas ID j. ai_attribute_count[j] 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 atlas ID j. Table 3 describes the list of supported attributes and their relationship with ai_attribute_type_id[j][i].

Table 3 — V3C 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 volumetric frame. 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 identifies the material type of a point in a volumetric frame. For example, the material type could be used as an indicator for identifying an object or the characteristic of a point within a volumetric frame. The interpretation of the values of such attribute frame type is outside the scope of this document.

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

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

ATTR_NORMAL indicates an attribute that contains a unit vector information associated with each point in a volumetric frame. 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 document and will not have a specified meaning in the future as an integral part of this document.

Values indicated as ATTR_RESERVED are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this edition of this document. Decoders conforming to this edition of this document should ignore attribute sub-bitstreams with an attribute type a value ai_attribute_type_id[j][i] equal to ATTR_RESERVED.

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 atlas ID j. ai_attribute_codec_id[j][i] shall be in the range of 0 to 255, inclusive. This codec may be identified through the profiles defined in Annex A, a component codec mapping SEI message, or through means outside this document.

ai_auxiliary_attribute_codec_id[j][i], when present, indicates the identifier of the codec used to compress the attribute video data for RAW or EOM coded points, or both, of attribute i, when RAW or EOM coded points, or both, are encoded in an auxiliary video stream for the atlas with atlas ID j. ai_auxiliary_attribute_codec_id[j][i] shall be in the range of 0 to 255, inclusive. This codec may be identified through the profiles defined in Annex A, a component codec mapping SEI message, or through means outside this document. When not present the value of ai_auxiliary_attribute_codec_id[j][i] is inferred to be equal to ai_attribute_codec_id[j][i].

ai_attribute_map_absolute_coding_persistence_flag[j][i] equal to 1 indicates that all attribute maps, for the attribute with index i, that correspond to the atlas with atlas ID j, are coded without any form of map prediction. ai_attribute_map_absolute_coding_persistence_flag[j][i] equal to 0 indicates that the attribute maps of the attribute with index i, that correspond to the atlas with atlas ID j, shall use the same map prediction method as used for the geometry component of the atlas with atlas ID j. If ai_attribute_map_absolute_coding_persistence_flag[j][i] is not present, its value shall be inferred to be equal to 1.

The 3D array AttributeMapAbsoluteCodingEnabledFlag, which indicates if a particular map of an attribute is to be coded with or without prediction, is obtained as follows:

```
if( ai_attribute_map_absolute_coding_persistance_flag[ j ][ i ] == 1) {
        for( k = 0; k < vps_map_count_minus1[ j ]; k++ )
                AttributeMapAbsoluteCodingEnabledFlag[ j ][ i ][ k ] = 1
}
else{
        for( k = 0; k < vps_map_count_minus1[ j ]; k++ )
                AttributeMapAbsoluteCodingEnabledFlag[ j ][ i ][ k ] =
                                                vps_map_absolute_coding_enabled_flag[ j ][ i ]
}
```

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 atlas ID 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 of the attribute with index i, for the atlas with atlas ID 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 the attribute with index i, for the atlas with atlas ID 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_2d_bit_depth_minus1[j][i] plus 1 indicates the nominal 2D bit depth to which all the attribute videos with attribute index i, for the atlas with atlas ID j, shall be converted to. ai_attribute_2d_bit_depth_minus1[j][i] shall be in the range of 0 to 31, inclusive.

ai_attribute_MSB_align_flag[j][i] indicates how the decoded attribute video samples with attribute index i, for the atlas with atlas ID j, are converted to samples at the nominal attribute bit depth, as specified in Annex B.

8.4.4.6 Profile toolset constraints information semantics

ptc_one_v3c_frame_only_flag, when present, has semantics specified in Annex A, where the profile indicated by ptl_profile_pcc_toolset_idc is a profile specified in Annex A. When not present, ptc_one_v3c_frame_only_flag is inferred to be equal to 0.

ptc_eom_constraint_flag equal to 1 specifies that the parameter asps_eom_patch_enabled_flag shall be equal to 0. ptc_eom_constraint_flag equal to 0 does not impose a constraint. When not present, ptc_eom_constraint_flag is inferred to be equal to 0.

ptc_max_map_count_minus1 specifies that vps_map_count_minus1 shall be less than or equal to ptc_max_map_count_minus1. When not present, ptc_max_map_count_minus1 is inferred to be equal to 15.

ptc_max_atlas_count_minus1 specifies that vps_atlas_count_minus1 shall be less than or equal to ptc_max_atlas_count_minus1. When not present, ptc_max_atlas_count_minus1 is inferred to be equal to 63.

ptc_multiple_map_streams_constraint_flag equal to 1 specifies that vps_multiple_map_streams_present_flag shall be equal to 0. ptc_multiple_map_streams_constraint_flag equal to 0 does not impose a constraint. When not present, ptc_multiple_map_streams_constraint_flag is inferred to be equal to 0.

ptc_plr_constraint_flag equal to 1 specifies that asps_plr_enabled_flag shall be equal to 0. ptc_plr_constraint_flag equal to 0 does not impose a constraint. When not present, ptc_plr_constraint_flag is inferred to be equal to 0.

ptc_attribute_max_dimension_minus1 specifies that ai_attribute_dimension_minus1 shall be less than or equal to ptc_attribute_max_dimension_minus1. When not present, ptc_attribute_max_dimension_minus1 is inferred to be equal to 63.

ptc_attribute_max_dimension_partitions_minus1 specifies that ai_attribute_dimension_partitions_minus1 shall be less than or equal to ptc_attribute_max_dimension_partitions_minus1. When not present, ptc_attribute_max_dimension_partitions_minus1 is inferred to be equal to 63.

ptc_no_eight_orientations_constraint_flag equal to 1 specifies that asps_use_eight_orientations_flag shall be equal to 0. ptc_no_eight_orientations_constraint_flag equal to 0 does not impose a constraint. When not present, ptc_no_eight_orientations_constraint_flag is inferred to be equal to 0.

ptc_45degree_no_projection_patch_constraint_flag equal to 1 specifies that asps_extended_projection_enabled_flag shall be equal to 0.

ptc_45degree_no_projection_patch_constraint_flag equal to 0 does not impose a constraint. When not present, ptc_45degree_no_projection_patch_constraint_flag is inferred to be equal to 0.

ptc_reserved_zero_6bits shall be equal to 0 in bitstreams conforming to this edition of this document. Other values of ptc_reserved_zero_6bits are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this edition of this document. Decoders conforming to this edition of this document shall ignore values of ptc_reserved_zero_6bits other than 0.

ptc_num_reserved_constraint_bytes specifies the number of the reserved constraint bytes. The value of ptc_num_reserved_constraint_bytes shall be 0 in bitstreams conforming to this edition of this document. Other values of ptc_num_reserved_constraint_bytes are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this edition of this document. Decoders conforming to this edition of this document shall ignore values of ptc_num_reserved_constraint_bytes other than 0.

ptc_reserved_constraint_byte[i] may have any value. Its presence and value do not affect decoder conformance to profiles specified in this edition of this document. Decoders conforming to this edition of this document shall ignore the values of all the ptc_reserved_constraint_byte[i] syntax elements.

8.4.5 NAL unit semantics

8.4.5.1 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 D for the sample stream format. Other methods of demarcation can be specified outside this document.

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 D 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.

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 4.

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.

8.4.5.2 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 4.

NAL units that have nal_unit_type in the range of NAL_UNSPEC_53..NAL_UNSPEC_63, inclusive, for which semantics are not specified, shall not affect the decoding process specified in this document.

NOTE 1 NAL unit types in the range of NAL_UNSPEC_53..NAL_UNSPEC_63 can be used as determined by the application. No decoding process for these values of nal_unit_type is specified in this document. Since different applications can use these NAL unit types for different purposes, particular care is expected 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 document does not define any management for these values. These nal_unit_type values could 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 document.

Table 4 — 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 1	NAL_TRAIL_N NAL_TRAIL_R	Coded tile of a non-TSA, non STSA trailing atlas frame atlas_tile_layer_rbsp( )	ACL
2 3	NAL_TSA_N NAL_TSA_R	Coded tile of a TSA atlas frame atlas_tile_layer_rbsp( )	ACL

Table 4 *(continued)*

nal_unit_type	Name of nal_unit_type	Content of NAL unit and RBSP syntax structure	NAL unit type class
4 5	NAL_STSA_N NAL_STSA_R	Coded tile of a STSA atlas frame atlas_tile_layer_rbsp( )	ACL
6 7	NAL_RADL_N NAL_RADL_R	Coded tile of a RADL atlas frame atlas_tile_layer_rbsp( )	ACL
8 9	NAL_RASL_N NAL_RASL_R	Coded tile of a RASL atlas frame atlas_tile_layer_rbsp( )	ACL
10 11	NAL_SKIP_N NAL_SKIP_R	Coded tile of a skipped atlas frame atlas_tile_layer_rbsp( )	ACL
12 14	NAL_RSV_ACL_N12 NAL_RSV_ACL_N14	Reserved non-IRAP sub-layer non-reference ACL NAL unit types	ACL
13 15	NAL_RSV_ACL_R13 NAL_RSV_ACL_R15	Reserved non-IRAP sub-layer reference ACL NAL unit types	ACL
16 17 18	NAL_BLA_W_LP NAL_BLA_W_RADL NAL_BLA_N_LP	Coded tile of a BLA atlas frame that is not a GBLA atlas_tile_layer_rbsp( )	ACL
19 20 21	NAL_GBLA_W_LP NAL_GBLA_W_RADL NAL_GBLA_N_LP	Coded tile of a GBLA atlas frame atlas_tile_layer_rbsp( )	ACL
22 23	NAL_IDR_W_RADL NAL_IDR_N_LP	Coded tile of an IDR atlas frame that is not a GIDR atlas_tile_layer_rbsp( )	ACL
24 25	NAL_GIDR_W_RADL NAL_GIDR_N_LP	Coded tile of a GIDR atlas frame atlas_tile_layer_rbsp( )	ACL
26	NAL_CRA	Coded tile of a CRA atlas frame that is not a GCRA atlas_tile_layer_rbsp( )	ACL
27	NAL_GCRA	Coded tile of a GCRA atlas frame atlas_tile_layer_rbsp( )	ACL
28 29	NAL_RSV_IRAP_ACL_28 NAL_RSV_IRAP_ACL_29	Reserved IRAP ACL NAL unit types	ACL
30..35	NAL_RSV_ACL_30.. NAL_RSV_ACL_35	Reserved non-IRAP ACL NAL unit types	ACL
36	NAL_ASPS	Atlas sequence parameter set atlas_sequence_parameter_set_rbsp( )	non-ACL
37	NAL_AFPS	Atlas frame parameter set atlas_frame_parameter_set_rbsp( )	non-ACL
38	NAL_AUD	Access unit delimiter access_unit_delimiter_rbsp( )	non-ACL
39	NAL_V3C_AUD	V3C access unit delimiter access_unit_delimiter_rbsp( )	non-ACL
40	NAL_EOS	End of sequence end_of_seq_rbsp( )	non-ACL
41	NAL_EOB	End of bitstream end_of_atlas_sub_bitstream_rbsp( )	non-ACL

Table 4 *(continued)*

nal_unit_type	Name of nal_unit_type	Content of NAL unit and RBSP syntax structure	NAL unit type class
42	NAL_FD	Filler filler_data_rbsp( )	non-ACL
43 44	NAL_PREFIX_NSEI NAL_SUFFIX_NSEI	Non-essential supplemental enhancement information sei_rbsp( )	non-ACL
45 46	NAL_PREFIX_ESEI NAL_SUFFIX_ESEI	Essential supplemental enhancement information sei_rbsp( )	non-ACL
47	NAL_AAPS	Atlas adaptation parameter set atlas_adaptation_parameter_set_rbsp( )	non-ACL
48..52	NAL_RSV_NACL_48 NAL_RSV_NACL_52	Reserved non-ACL NAL unit types	non-ACL
53..63	NAL_UNSPEC_53 NAL_UNSPEC_63	Unspecified non-ACL NAL unit types	non-ACL

NOTE 3 A clean random access (CRA) atlas frame can 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) atlas frame having nal_unit_type equal to NAL_BLA_W_LP or NAL_GBLA_W_LP, can have associated RASL or RADL atlas frames present in the bitstream. A BLA atlas frame having nal_unit_type equal to NAL_BLA_W_RADL or NAL_GBLA_W_RADL, does not have associated RASL atlas frames present in the bitstream, but can have associated RADL atlas frames in the bitstream. A BLA atlas frame having nal_unit_type equal to NAL_BLA_N_LP or NAL_GBLA_N_LP, does not have associated leading atlas frames present in the bitstream.

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

All coded tile 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 NAL units of the atlas frame or the access unit.

If an atlas frame has nal_unit_type equal to NAL_TRAIL_N, NAL_TSA_N, NAL_STSA_N, NAL_RADL_N, NAL_RASL_N, NAL_RSV_ACL_N12, or NAL_RSV_ACL_N14, the atlas frame is an SLNR atlas frame. Otherwise, the atlas frame is a sub-layer reference atlas frame.

If an atlas frame has nal_unit_type equal to NAL_GIDR_W_RADL, NAL_GBLA_N_LP, or NAL_GCRA, or in the range of NAL_GBLA_W_LP to NAL_GBLA_N_LP, inclusive, it specifies a random access association between the current coded atlas and the corresponding coded video frames at the same composition time.

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 coded 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 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 or NAL_GBLA_N_LP, an IDR atlas frame having nal_unit_type equal to NAL_IDR_N_LP or NAL_GIDR_N_LP.

NOTE 6 It is possible to perform random access at the position of an IRAP coded atlas access unit by discarding all access units before the IRAP coded atlas access unit (and to correctly decode the IRAP coded 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 document) when it needs to be activated.

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

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

A nal_unit_type equal to NAL_V3C_AUD that is associated with a particular atlas frame j means that all output V3C component frames, if present in their corresponding sub-bitstreams, and that have the same output time order as the atlas frame j, shall have the same decoding order, while accounting for missing frames in each sub-bitstream, as that of the atlas frame.

NOTE 7 An application can specify that all atlas frames in a V3C sequence are delimited using a nal_unit_type equal to NAL_V3C_AUD, in which case all V3C sub-bitstreams including sub-bitstreams that correspond to different maps will be aligned in both decoding order and output time.

NOTE 8 If a frame in a sub-bitstream is missing, and a NAL_V3C_AUD is encountered, it can be assumed that this missing frame is virtually present in its sub-bitstream and has the same decoding order as all other corresponding frames in the other sub-bitstreams. This frame can then be generated using such methods as described in Annex B.

When an ACL NAL unit of an atlas with atlas ID equal to atlasID, with decoding time equal to decodingTime, and composition time equal to compTime, is of nal_unit_type equal to NAL_GBLA_W_LP, NAL_GBLA_W_RADL, NAL_GBLA_N_LP, NAL_GIDR_W_RADL, or NAL_GIDR_N_LP, there is at least one associated video frame unit in each video sub-bitstream that has decoding time and composition time equal to decodingTime and compTime, respectively. Such video frame units shall also be coded using appropriate intra random access point types as specified by their respective video coding specification. Such intra random access point types are outside the scope of this document. These ACL NAL units can be used with their associated video frame units, after decoding, to reconstruct a volumetric frame with composition time equal to compTime. Assignment of decoding time decodingTime to atlas NAL units and to data of a video sub-bitstream is outside the scope of this document.

NOTE 9 Decoding time can be provided by the ISO base media file format for a V3C bitstream defined in ISO/IEC DIS 23090-10.

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 10 The value of 63 for nal_layer_id can be used to indicate an extended layer identifier in a future extension of this document.

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:

TemporalID = nal_temporal_id_plus1 − 1 (16)

When nal_unit_type is in the range of NAL_BLA_W_LP to NAL_RSV_IRAP_ACL29, inclusive, i.e., the coded tile belongs to an IRAP coded atlas frame, TemporalID shall be equal to 0.

When nal_unit_type is equal to NAL_TSA_R or NAL_TSA_N, TemporalID shall not be equal to 0.

When nal_layer_id is equal to 0 and nal_unit_type is equal to NAL_STSA_R or NAL_STSA_N, 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_V3C_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 11 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 can be greater than or equal to the TemporalID of the containing access unit, as all atlas frame parameter sets (AFPSs) can 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_NSEI, NAL_PREFIX_ESEI, NAL_SUFFIX_NSEI, or NAL_SUFFIX_ESEI, TemporalID can be greater than or equal to the TemporalID of the containing access unit, as an SEI NAL unit can 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.

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

8.4.5.3.1 General

This subclause specifies constraints on the order of NAL units and atlas frames in the atlas sub-bitstream.

Any order of NAL units in the atlas sub-bitstream obeying these constraints is referred to in the text as the decoding order of NAL units. Within a NAL unit, the syntax in subclause 8.3, specifies the decoding order of syntax elements. Decoders shall be capable of receiving NAL units and their syntax elements in decoding order.

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

This subclause specifies the activation process of atlas adaptation parameter sets (AAPSs), atlas sequence parameter sets (ASPSs) and atlas frame parameter sets (AFPSs).

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

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

When an AAPS RBSP (with a particular value of aaps_atlas_adaptation_parameter_set_id) is not active for a particular atlas and is referred to by a coded tile NAL unit with nal_layer_id equal to 0 (using a value of ath_atlas_adaptation_parameter_set_id equal to the aaps_atlas_adaptation_parameter_set_id value), it is then activated. This AAPS RBSP is called the active AAPS RBSP until it is deactivated by the activation of another AAPS RBSP. Let the variable aapsAtlasID be either set equal to vuh_atlas_id of the AAPS RBSP or determined through external means if the V3C unit header is unavailable. Let the variable activatingAtlasID be either set equal to vuh_atlas_id of the coded tile NAL unit or determined through external means if the V3C unit header is unavailable. An AAPS RBSP, with that particular value of aaps_atlas_adaptation_parameter_set_id, shall be available to the decoding process prior to its activation, included in at least one coded atlas access unit with TemporalID less than or equal to the TemporalID of the AAPS NAL unit or provided through external means, aapsAtlasID shall be equal to activatingAtlasID, and the AAPS NAL unit containing the AAPS RBSP shall have nal_layer_id equal to 0.

Any AAPS NAL unit containing the value of aaps_atlas_adaptation_parameter_set_id for the active AAPS RBSP for a coded atlas frame shall have the same content as that of the active AAPS 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 AFPS RBSP includes parameters that can be referred to by the coded tile NAL units of one or more coded atlas frames. Each AFPS RBSP is initially considered not active for any atlas at the start of the operation of the decoding process. At most one AFPS RBSP is considered active for each atlas at any given moment during the operation of the decoding process, and the activation of any particular AFPS RBSP for a particular atlas results in the deactivation of the previously active AFPS RBSP for the particular atlas.

When an AFPS RBSP (with a particular value of afps_atlas_frame_parameter_set_id) is not active for a particular atlas and is referred to by a coded tile NAL unit with nal_layer_id equal to 0 (using a value of ath_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. Let the variable afpsAtlasID be either set equal to vuh_atlas_id of the AFPS RBSP or determined through external means if the V3C unit header is unavailable. Let the variable activatingAtlasID be either set equal to vuh_atlas_id of the coded tile NAL unit or determined through external means if the V3C unit header is unavailable. 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 coded atlas access unit with TemporalID less than or equal to the TemporalID of the AFPS NAL unit or provided through external means, afpsAtlasID shall be equal to activatingAtlasID, 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 for a particular atlas and it is referred to by activation of an AFPS RBSP (in which afps_atlas_sequence_parameter_set_id is equal to the asps_atlas_sequence_parameter_set_id value with afpsAtlasID set equal to vuh_atlas_id or determined through external means), it is activated for the particular atlas. This ASPS RBSP is called the active ASPS RBSP for the particular atlas until it is deactivated by the activation of another ASPS RBSP for the particular atlas. Let the variable aspsAtlasID be either set equal to vuh_atlas_id or determined through external means if the V3C unit header is unavailable. Let the variable activatingAtlasID either be set equal to vuh_atlas_id of the coded tile or determined through external means if the V3C unit header is unavailable. 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, aspsAtlasID equal to activatingAtlasID, 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 any aspsAtlasID value and 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 AAPSs, ASPSs, AFPSs, and other syntax elements, are expressions of constraints that apply only to the active AAPS RBSP, the active ASPS RBSP, and the active AFPS RBSP. If any AAPS RBSP, 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 9), the values of parameters of the active AAPS RBSP, the active ASPS RBSP, and the active AFPS RBSP are considered in effect. For interpretation of SEI messages, the values of the active AAPS RBSP, 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.

8.4.5.3.3 Order of access units (AUs) and association to CASs

A bitstream conforming to this document 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, are described in subclause 8.4.5.3.4.

The first access unit of a CAS is an IRAP coded atlas access unit with NoOutputBeforeRecoveryFlag 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 a bitstream NAL unit shall be an IRAP coded atlas access unit, which may be an IDR access unit, a BLA access unit, or a CRA access unit.

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

This subclause specifies the order of NAL units and coded atlas frames and their association to access units for CASs that belong to a CVSs 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.

An access unit consists of one coded atlas 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 atlases is described in subclause 8.4.5.3.5.

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 can have nal_layer_id greater than 0.

— access unit delimiter or V3C access unit delimiter NAL unit with nal_layer_id equal to 0 (when present),

— AAPS 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_NACL_48..NAL_RSV_NACL_52 with nal_layer_id equal to 0 (when present),

— NAL units with nal_unit_type in the range of NAL_UNSPEC_53..NAL_UNSPEC_63 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 V3C 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 V3C access unit delimiter NAL unit with nal_layer_id equal to 0 in any access unit.

— When any ASPS NAL units, AAPS NAL units, AFPS NAL units, prefix SEI NAL units, NAL units with nal_unit_type in the range of NAL_RSV_NACL_48..NAL_RSV_NACL_50, or NAL units with nal_unit_type in the range of NAL_UNSPEC_53..NAL_UNSPEC_57 are present, they shall not follow the last ACL NAL unit of the access unit.

— NAL units having nal_unit_type equal to NAL_FD, NAL_SUFFIX_NSEI, NAL_SUFFIX_ESEI or in the range of NAL_RSV_NACL_51..NAL_RSV_NACL_52 or NAL_UNSPEC_58..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 a bitstream NAL unit (when present).

— When an end of a 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 V3C 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.

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

This subclause 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 ath_id equal to FirstTileID.

— The tiles of an atlas frame shall be in increasing order of their ath_id values.

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

8.4.6.1 Atlas sequence parameter set RBSP semantics

8.4.6.1.1 General 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.

asps_frame_width indicates the atlas frame width in terms of integer number of samples, where a sample corresponds to a luma sample of a video component. It is a requirement of V3C bitstream conformance that the value of asps_frame_width shall be equal to the value of vps_frame_width[j], where j is the ID of the current atlas.

asps_frame_height indicates the atlas frame height in terms of integer number of samples, where a sample corresponds to a luma sample of a video component. It is a requirement of V3C bitstream conformance that the value of asps_frame_height shall be equal to the value of vps_frame_height[j], where j is the ID of the current atlas.

The variable AspsFrameSize is set equal to asps_frame_height * asps_frame_width.

NOTE 1 During the reconstruction phase the decoded occupancy, geometry and attribute videos are converted to the nominal format of the current atlas as specified in Annex B.

asps_geometry_3d_bit_depth_minus1 plus 1 indicates the bit depth of the geometry coordinates of the reconstructed volumetric content. asps_geometry_3d_bit_depth_minus1 shall be in the range of 0 to 31, inclusive.

asps_geometry_2d_bit_depth_minus1 plus 1 indicates the bit depth of the geometry when projected onto 2D images. asps_geometry_2d_bit_depth_minus1 shall be in the range of 0 to 31, inclusive.

NOTE 2 For some applications, such as the application defined in Annex H, the syntax elements asps_geometry_3d_bit_depth_minus1 and asps_geometry_2d_bit_depth_minus1 can differ from their corresponding elements in the VPS, gi_geometry_3d_coordinates_bit_depth_minus1 and gi_geometry_2d_bit_depth_minus1, respectively. This can be due to the fact that the elements asps_geometry_3d_bit_depth_minus1 and asps_geometry_2d_bit_depth_minus1 potentially represent an intermediate space that can require different precision for coding purposes than the target representation space, which the elements gi_geometry_3d_coordinates_bit_depth_minus1 and gi_geometry_2d_bit_depth_minus1 indicate.

asps_log2_max_atlas_frame_order_cnt_lsb_minus4 plus 4 specifies the values of the variables Log2MaxAtlasFrmOrderCntLsb and MaxAtlasFrmOrderCntLsb that are used in the decoding process for the atlas frame order count as follows:

$$\text{Log2MaxAtlasFrmOrderCntLsb} = \text{asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4} + 4 \quad (17)$$

$$\text{MaxAtlasFrmOrderCntLsb} = 2^{\text{Log2MaxAtlasFrmOrderCntLsb}} \quad (18)$$

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 15, inclusive.

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 3 A decoder allocates memory for a total number of ref_list_struct(rlsIdx) syntax structures equal to (asps_num_ref_atlas_frame_lists_in_asps + 1) since there can be one ref_list_struct(rlsIdx) syntax structure directly signalled in the atlas tile headers of the current atlas tile.

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

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

asps_max_number_projections_minus1 plus 1 specifies the maximum value that can be indicated for the patch projection ID syntax element, pdu_projection_id[i][j], for a patch with index j in a tile with tile ID equal to i. When asps_max_number_projections_minus1 is not present, its value is inferred to be equal to 5.

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 delta 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 delta 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 indicated in the bitstream for each patch data unit, a merge patch data unit, or an inter patch data unit.

asps_patch_precedence_order_flag specifies the patch precedence that is utilized for assigning atlas samples to patches as specified in subclause 9.2.6. asps_patch_precedence_order_flag equal to 1 specifies that patch precedence for the current atlas is the same as the decoding order of the patches. asps_patch_precedence_order_flag equal to 0 specifies that patch precedence for the current atlas is the reverse of the decoding order of the patches.

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:

$$\text{PatchPackingBlockSize} = 2^{\text{asps\_log2\_patch\_packing\_block\_size}} \quad (19)$$

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

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

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 document that asps_map_count_minus1 is equal to vps_map_count_minus1[atlasID], where atlasID is the atlas ID of the current atlas.

asps_pixel_deinterleaving_enabled_flag equal to 1 indicates that the decoded geometry and attribute data may require an additional spatial interpolation process during reconstruction. asps_pixel_deinterleaving_enabled_flag equal to 0 indicates that the decoded geometry and attribute data do not require any additional spatial interpolation process.

asps_map_pixel_deinterleaving_flag[i] equal to 1 indicates that an additional spatial interpolation process needs to be performed on the associated geometry and attribute data of a projected patch in a map with index i in the current atlas. asps_map_pixel_deinterleaving_flag[i] equal to 0 indicates that no additional spatial interpolation process is performed. When not present, the value of asps_map_pixel_deinterleaving_flag[i] is inferred to be 0.

NOTE 4 In the context of reconstruction profiles for V-PCC, an example usage is specified in subclause H.7.2.3.

asps_raw_patch_enabled_flag equal to 1 indicates that the decoded geometry and attribute videos for the current atlas contain information related to RAW coded points. asps_raw_patch_enabled_flag equal to 0 indicates that the decoded geometry and attribute videos do not contain information related to RAW coded points.

asps_eom_patch_enabled_flag equal to 1 indicates that the decoded occupancy video for the current atlas contains information related to whether intermediate depth positions between two depth maps are occupied. asps_eom_patch_enabled_flag equal to 0 indicates that the decoded occupancy 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_eom_patch_enabled_flag is equal to 1, oi_lossy_occupancy_compression_threshold[atlasID] shall be equal to 0.

asps_eom_fix_bit_count_minus1 plus 1 indicates the size in bits of the EOM codeword. asps_eom_fix_bit_count_minus1 shall be in the range of 0 to Min(15, oi_occupancy_2d_bit_depth_minus1[atlasID] − 1), inclusive, where atlasID is the atlas ID of the current atlas.

asps_auxiliary_video_enabled_flag equal to 1 indicates that information associated with RAW and EOM patch types could be placed in auxiliary video sub-bitstreams. asps_auxiliary_video_enabled_flag equal to 0 indicates that information associated with RAW and EOM patch types can only be placed in non-auxiliary video sub-bitstreams.

It is a requirement of bitstream conformance that if vps_auxiliary_video_present_flag[atlasID] , where atlasID is the atlas ID of the current atlas, is equal to 0 then asps_auxiliary_video_enabled_flag is equal to 0.

asps_plr_enabled_flag equal to 1 indicates that point local reconstruction mode information may be present in the bitstream for the current atlas. asps_plr_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 when asps_pixel_deinterleaving_enabled_flag is equal to 1, asps_plr_enabled_flag shall be equal to 0.

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

asps_extension_present_flag equal to 1 specifies that the syntax elements asps_vpcc_extension_present_flag and asps_extension_7bits are present in the atlas_sequence_parameter_set_rbsp syntax structure. asps_extension_present_flag equal to 0 specifies that the syntax elements asps_vpcc_extension_present_flag and asps_extension_7bits are not present.

asps_vpcc_extension_present_flag equal to 1 specifies that the asps_vpcc_extension( ) syntax structure is present in the atlas_sequence_parameter_set_rbsp syntax structure. asps_vpcc_extension_present_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of asps_vpcc_extension_present_flag is inferred to be equal to 0.

asps_extension_7bits equal to 0 specifies that no asps_extension_data_flag syntax elements are present in the ASPS RBSP syntax structure. When present, asps_extension_7bits shall be equal to 0 in bitstreams conforming to this edition of this document. Values of asps_extension_7bits not equal to 0 are reserved for future use by ISO/IEC. Decoders shall allow the value of asps_extension_7bits to be not equal to 0 and shall ignore all asps_extension_data_flag syntax elements in an ASPS NAL unit. When not present, the value of asps_extension_7bits is inferred to be equal to 0.

asps_extension_data_flag may have any value. Its presence and value do not affect decoder conformance to profiles specified in this edition of this document. Decoders conforming to this edition of this document shall ignore all asps_extension_data_flag syntax elements.

8.4.6.1.2 Point local reconstruction information semantics

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

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

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

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

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

plri_block_threshold_per_patch_minus1[i] plus 1 specifies the value representative of the threshold used to define the values contained in the array plrd_level. plri_block_threshold_per_patch_minus1[i] shall be in the range of 0 to 63, inclusive.

8.4.6.2 Atlas frame parameter set RBSP semantics

8.4.6.2.1 General atlas frame parameter set RBSP semantics

afps_atlas_frame_parameter_set_id identifies the atlas frame parameter set for reference by other syntax elements.

afps_atlas_sequence_parameter_set_id specifies the value of asps_atlas_sequence_parameter_set_id for the active atlas sequence parameter set.

afps_output_flag_present_flag equal to 1 indicates that the ath_atlas_output_flag syntax element is present in the associated tile headers. afps_output_flag_present_flag equal to 0 indicates that the ath_atlas_output_flag syntax element is not present in the associated tile headers.

afps_num_ref_idx_default_active_minus1 plus 1 specifies the inferred value of the variable NumRefIdxActive for the tile with ath_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:

$$\text{MaxLtAtlasFrmOrderCntLsb} = 2 * (\,\text{Log2MaxAtlasFrmOrderCntLsb} + \text{afps\_additional\_lt\_afoc\_lsb\_len}) \qquad (20)$$

The value of afps_additional_lt_afoc_lsb_len shall be in the range of 0 to 32 – Log2MaxAtlasFrmOrderCntLsb, 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_lod_mode_enabled_flag equal to 1 indicates that the LOD parameters may be present in a patch. afps_lod_mode_enabled_flag equal to 0 indicates that the LOD parameters shall not be present in a patch.

afps_raw_3d_offset_bit_count_explicit_mode_flag equal to 1 indicates that the number of bits in the fixed-length representation of rpdu_3d_offset_u[tileID][p], rpdu_3d_offset_v[tileID][p], and rpdu_3d_offset_d[tileID][p] for a patch with index p in a tile with tile ID equal to tileID, is explicitly coded by ath_raw_3d_offset_axis_bit_count_minus1 in the atlas tile header that refers to afps_atlas_frame_parameter_set_id. afps_raw_3d_offset_bit_count_explicit_mode_flag equal to 0 indicates that the value of ath_raw_3d_offset_axis_bit_count_minus1 is implicitly derived.

afps_extension_present_flag equal to 1 specifies that the syntax element afps_extension_8bits is present in the atlas_frame_parameter_set_rbsp syntax structure. afps_extension_present_flag equal to 0 specifies that the syntax element afps_extension_8bits is not present. The value of afps_extension_present_flag shall be 0 in this edition of this document

afps_extension_8bits equal to 0 specifies that no afps_extension_data_flag syntax elements are present in the AFPS RBSP syntax structure. When present, afps_extension_8bits shall be equal to 0 in bitstreams conforming to this edition of this document. Values of afps_extension_8bits not equal to 0 are reserved for future use by ISO/IEC. Decoders shall allow the value of afps_extension_8bits to be not equal to 0 and shall ignore all afps_extension_data_flag syntax elements in an AFPS NAL unit. When not present, the value of afps_extension_8bits is inferred to be equal to 0.

afps_extension_data_flag may have any value. Its presence and value do not affect decoder conformance to profiles specified in this edition of this document. Decoders conforming to this edition of this document shall ignore all afps_extension_data_flag syntax elements.

8.4.6.2.2 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 may be more than one tile in each atlas frame referring to the AFPS.

afti_uniform_partition_spacing_flag equal to 1 specifies that the tile partitioning of an atlas uses a method that distributes column and row partition boundaries uniformly across the atlas frame. The information corresponding to these boundaries is signalled using the syntax elements afti_partition_cols_width_minus1 and afti_partition_rows_height_minus1, respectively. afti_uniform_partition_spacing_flag equal to 0 specifies that the tile partitioning of an atlas uses a method that may result in column and row partition boundaries that may or may not be distributed uniformly across the atlas frame. In this case, these boundaries are signalled using the syntax elements afti_num_partition_columns_minus1 and afti_num_partition_rows_minus1 and a list of syntax element pairs afti_partition_column_width_minus1[i] and afti_partition_row_height_minus1[i]. When not present, the value of afti_uniform_partition_spacing_flag is inferred to be equal to 1.

afti_partition_cols_width_minus1 plus 1 specifies the width of the tile partition columns excluding the right-most tile partition column of the atlas frame in units of 64 samples when afti_uniform_partition_spacing_flag is equal to 1. The value of afti_partition_cols_width_minus1 shall be in the range of 0 to asps_frame_width / 64 − 1, inclusive. When not present, the value of afti_partition_cols_width_minus1 is inferred to be equal to asps_frame_width / 64 – 1**.**

afti_partition_rows_height_minus1 plus 1 specifies the height of the tile partition rows excluding the bottom tile partition row of the atlas frame in units of 64 samples when afti_uniform_partition_spacing_flag is equal to 1. The value of afti_partition_rows_height_minus1 shall be in the range of 0 to asps_frame_height / 64 − 1, inclusive. When not present, the value of afti_partition_rows_height_minus1 is inferred to be equal to asps_frame_height / 64 – 1.

afti_num_partition_columns_minus1 plus 1 specifies the number of tile partition columns used to partition the atlas frame when afti_uniform_partition_spacing_flag is equal to 0. The value of afti_num_partition_columns_minus1 shall be in the range of 0 to asps_frame_width / 64 − 1, inclusive. If afti_single_tile_in_atlas_frame_flag is equal to 1, the value of afti_num_partition_columns_minus1 is inferred to be equal to 0. Otherwise, when afti_uniform_partition_spacing_flag is equal to 1, the value of afti_num_partition_columns_minus1 is inferred as specified in subclause 7.5.

afti_num_partition_rows_minus1 plus 1 specifies the number of tile partition rows used to partition the atlas frame when afti_uniform_partition_spacing_flag is equal to 0. The value of afti_num_partition_rows_minus1 shall be in the range of 0 to asps_frame_height / 64 − 1, inclusive. If afti_single_tile_in_atlas_frame_flag is equal to 1, the value of afti_num_partition_rows_minus1 is inferred to be equal to 0. Otherwise, when afti_uniform_partition_spacing_flag is equal to 1, the value of afti_num_partition_rows_minus1 is inferred as specified in subclause 7.5.

The variable NumPartitionsInAtlasFrame is set equal to NumPartitionColumns * NumPartitionRows, where NumPartitionColumns and NumPartitionRows are derived in subclause 7.5.

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

afti_partition_column_width_minus1[i] plus 1 specifies the width of the i-th tile partition column in units of 64 samples.

afti_partition_row_height_minus1[i] plus 1 specifies the height of the i-th tile partition row in units of 64 samples.

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

afti_num_tiles_in_atlas_frame_minus1 plus 1 specifies the number of tiles in each atlas frame referring to the AFPS. The value of afti_num_tiles_in_atlas_frame_minus1 shall be in the range of 0 to NumPartitionsInAtlasFrame − 1, inclusive. When not present and afti_single_partition_per_tile_flag is equal to 1, the value of afti_num_tiles_in_atlas_frame_minus1 is inferred to be equal to NumPartitionsInAtlasFrame − 1.

afti_top_left_partition_idx[i] specifies the partition index of the tile partition located at the top-left corner of the i-th tile. The value of afti_top_left_partition_idx[i] shall be in the range of 0 to NumPartitionsInAtlasFrame − 1, inclusive. When not present, the value of afti_top_left_partition_idx[i] is inferred to be equal to i. The length of the afti_top_left_partition_idx[i] syntax element is Ceil(Log2(NumPartitionsInAtlasFrame)) bits.

afti_bottom_right_partition_column_offset[i] specifies the offset between the column position of the tile partition located at the bottom-right corner of the i-th tile and the column position of the tile partition with partition index equal to afti_top_left_partition_idx[i]. When afti_single_partition_per_tile_flag is equal to 1, the value of afti_bottom_right_partition_column_offset[i] is inferred to be equal to 0.

afti_bottom_right_partition_row_offset[i] specifies the offset between the row position of the tile partition located at the bottom-right corner of the i-th tile and the row position of the tile partition with partition index equal to afti_top_left_partition_idx[i]. When afti_single_partition_per_tile_flag is equal to 1, the value of afti_bottom_right_partition_row_offset[i] is inferred to be equal to 0.

The variables topLeftColumn[i], topLeftRow[i], bottomRightColumn[i], and bottomRightRow[i], which specify the corresponding tile column and row positions for the top left and bottom right tiles in a tile are computed as follows:

```
topLeftColumn[ i ] = afti_top_left_partition_idx[ i ] % NumPartitionColumns
topLeftRow[ i ] = afti_top_left_partition_idx[ i ] / NumPartitionColumns
bottomRightColumn[ i ] = topLeftColumn[ i ] + afti_bottom_right_partition_column_offset[ i ]
bottomRightRow[ i ] = topLeftRow[ i ] + afti_bottom_right_partition_row_offset[ i ]
```

It is a requirement of bitstream conformance that the values of bottomRightColumn[i] and bottomRightRow[i] shall be smaller or equal to (asps_frame_width + 63) / 64 − 1 and (asps_frame_height + 63) / 64 − 1, respectively.

It is also a requirement of bitstream conformance that there shall not be a value of j, where j != i, that satisfies either one of these properties:

— topLeftColumn[i] <= topLeftColumn[j] <= bottomRightColumn[i]

— topLeftRow[i] <= topLeftRow[j] <= bottomRightRow[i]

The variables TileOffsetX[i], TileOffsetY[i], TileWidth[i], and TileHeight[i], which specify the horizontal position, vertical position, width, and height of a tile respectively, are then computed, using the lists PartitionWidth[i] and PartitionHeight[j] that are computed as specified in subclause 7.5, as follows:

```
TileOffsetX[ i ] = PartitionPosX[ topLeftColumn[ i ] ]
TileOffsetY[ i ] = PartitionPosY[ topLeftColumn[ i ] ]
TileWidth[ i ] = 0
TileHeight[ i ] = 0
for( j = topLeftColumn[ i ]; j <= bottomRightColumn[ i ]; j++) {
    TileWidth[ i ] += PartitionWidth[ j ]
}
for( j = topLeftRow[ i ]; j <= bottomRightRow[ i ]; j++) {
    TileHeight[ i ] += PartitionHeight[ j ]
}
```

afti_auxiliary_video_tile_row_width_minus1 plus 1 indicates the nominal width of all auxiliary video sub-bitstreams in units of 64 integer samples. When afti_auxiliary_video_tile_row_width_minus1 is not present, its value shall be inferred to be equal to −1.

afti_auxiliary_video_tile_row_height[i] indicates the nominal height in units of 64 integer samples of the i-th vertical sub-region in each auxiliary video sub-bitstream that is associated with the i-th tile of the atlas. When afti_auxiliary_video_tile_row_height[i] is not present, its value shall be inferred to be equal to 0.

The height, AuxTileHeight[i] of each auxiliary sub-region associated with the i-th tile of the atlas is computed as:

$$\text{AuxTileHeight}[\ i\ ] = \text{afti\_auxiliary\_video\_tile\_row\_height}[\ i\ ] * 64$$

The vertical position, AuxTileOffset[i] of each auxiliary sub-region associated with the i-th tile of the atlas is computed as:

$$\text{AuxTileOffset}[\ 0\ ] = 0$$

$$\text{AuxTileOffset}[\ i\ ] = \text{AuxTileOffset}[\ i - 1\ ] + \text{AuxTileHeight}[\ i - 1\ ], \text{ for all } i > 0.$$

The nominal width, AuxVideoWidthNF, and height, AuxVideoHeightNF, of all auxiliary video sub-bitstreams associated with an atlas shall be computed as:

$$\text{AuxVideoWidthNF} = (\ \text{afti\_auxiliary\_video\_tile\_row\_width\_minus1} + 1\ ) * 64$$

$$\text{AuxVideoHeightNF} = \sum_{n=0}^{N} \text{AuxTileHeight}[n]$$

where N is equal to afti_num_tiles_in_atlas_frame_minus1

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

afti_signalled_tile_id_length_minus1 plus 1 specifies the number of bits used to represent the syntax element afti_tile_id[i] when present, and the syntax element ath_id in a tile header. The value of afti_signalled_tile_id_length_minus1 shall be in the range of 0 to 15, inclusive. When not present, the value of afti_signalled_tile_id_length_minus1 is inferred to be equal to Ceil(Log2(afti_num_tiles_in_atlas_frame_minus1 + 1)) − 1.

afti_tile_id[i] specifies the tile ID of the i-th tile. The length of the afti_tile_id[i] syntax element is afti_signalled_tile_id_length_minus1 + 1 bits. When not present, the value of afti_tile_id[i] is inferred to be equal to i, for each i in the range of 0 to afti_num_tiles_in_atlas_frame_minus1, inclusive. It is a requirement of bitstream conformance that afti_tile_id[i] shall not be equal to afti_tile_id[j] for all i != j. The length of the afti_tile_id[i] syntax element is afti_signalled_tile_id_length_minus1 + 1 bits.

The variable FirstTileID is computed as follows:

```
FirstTileID = afti_tile_id[ 0 ]
for ( i = 1; i < afti_num_tiles_in_atlas_frame_minus1 + 1; i++ )
    FirstTileID = Min(FirstTileID, afti_tile_id[ i ])
```

The arrays TileIDToIndex and TileIndexToID provide a forward and inverse mapping, respectively, of the ID associated with each tile and the order index of how each tile was specified in the atlas frame tile information syntax.

8.4.6.3 Atlas adaptation parameter set RBSP semantics

aaps_atlas_adaptation_parameter_set_id identifies the atlas adaptation parameter set for reference by other syntax elements.

aaps_extension_present_flag equal to 1 specifies that the syntax elements aaps_vpcc_extension_present_flag and aaps_extension_7bits are present in the atlas_adaptation_parameter_set_rbsp( ) syntax structure. aaps_extension_present_flag equal to 0 specifies that the syntax elements aaps_vpcc_extension_present_flag and aaps_extension_7bits are not present.

aaps_vpcc_extension_present_flag equal to 1 specifies that the aaps_vpcc_extension( ) syntax structure is present in the atlas_adaptation_parameter_set_rbsp( ) syntax structure. aaps_vpcc_extension_present_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of aaps_vpcc_extension_present_flag is inferred to be equal to 0.

aaps_extension_7bits equal to 0 specifies that no aaps_extension_data_flag syntax elements are present in the AAPS RBSP syntax structure. When present, aaps_extension_7bits shall be equal to 0 in bitstreams conforming to this edition of this document. Values of aaps_extension_7bits not equal to 0 are reserved for future use by ISO/IEC. Decoders shall allow the value of aaps_extension_7bits to be not equal to 0 and shall ignore all aaps_extension_data_flag syntax elements in an AAPS NAL unit. When not present, the value of aaps_extension_7bits is inferred to be equal to 0.

aaps_extension_data_flag may have any value. Its presence and value do not affect decoder conformance to profiles specified in this edition of this document. Decoders conforming to this edition of this document shall ignore all aaps_extension_data_flag syntax elements.

8.4.6.4 Supplemental enhancement information RBSP semantics

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

8.4.6.5 Access unit delimiter RBSP semantics

The access unit delimiter may be used to indicate the type of tiles present in the coded atlas frames in the access unit containing the access unit delimiter or V3C access unit delimiter NAL unit and to simplify the detection of the boundary between access units. There is no mandatory decoding process associated with the access unit delimiter or V3C access unit delimiter.

aframe_type indicates that the ath_type values for all tiles of the coded atlas frame in the access unit containing the access unit delimiter or V3C access unit delimiter NAL unit are members of the set listed in Table 5 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 edition of this document. Other values of aframe_type are reserved for future use by ISO/IEC. Decoders conforming to this edition of this document shall ignore reserved values of aframe_type.

Table 5 — Interpretation of aframe_type

aframe_type	ath_type values that may be present in the coded atlas frame
0	I_TILE
1	P_TILE, I_TILE
2	SKIP_TILE, P_TILE, I_TILE
3	SKIP_TILE
4 – 7	RESERVED

8.4.6.6 End of sequence RBSP semantics

When included in a NAL unit with nal_layer_id equal to 0, 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 atlas sub-bitstream in decoding order (if any) is an IRAP coded atlas access unit with NoOutputBeforeRecoveryFlag equal to 1. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty.

8.4.6.7 End of bitstream RBSP semantics

The end of bitstream RBSP indicates that no additional NAL units are present in the atlas sub-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.

8.4.6.8 Filler data RBSP semantics

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

ff_byte is a byte equal to 0xFF.

8.4.6.9 Atlas tile layer RBSP semantics

None.

8.4.6.10 RBSP trailing bit semantics

rbsp_stop_one_bit shall be equal to 1.

rbsp_alignment_zero_bit shall be equal to 0.

8.4.6.11 Atlas tile header semantics

When present, the value of the atlas tile header syntax elements ath_atlas_frame_parameter_set_id, ath_atlas_adaptation_parameter_set_id, ath_atlas_output_flag, ath_no_output_of_prior_atlas_frames_flag, and ath_atlas_frm_order_cnt_lsb, shall be the same in all atlas tile headers of a coded atlas frame.

ath_no_output_of_prior_atlas_frames_flag affects the output of previously-decoded atlas frames in the DAB after the decoding of an atlas in a CAS AU that is not the first AU in the bitstream as specified in Annex E. When ath_no_output_of_prior_atlas_frames_flag is not present, its value is inferred to be equal to 0.

It is a requirement of bitstream conformance that the value of ath_no_output_of_prior_atlas_frames_flag shall be the same for all atlas frames in an AU.

The value of ath_no_output_of_prior_atlas_frames_flag in the atlas tile headers is also referred to as the output_of_prior_atlas_frames_flag value of the AU.

ath_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.

ath_atlas_adaptation_parameter_set_id specifies the value of aaps_atlas_adaptation_parameter_set_id for the active atlas adaptation parameter set for the current atlas tile. It is a requirement of bitstream conformance that all tiles of an atlas frame shall have the same ath_atlas_adaptation_parameter_set_id.

ath_id specifies the tile ID associated with the current tile. When not present, the value of ath_id is inferred to be equal to 0.

The following applies:

— The length of ath_id is afti_signalled_tile_id_length_minus1 + 1 bits.

— The value of ath_id shall be in the range of values specified by the array TileIndexToID[i], for i in the range from 0 to afti_num_tiles_in_atlas_frame_minus1, inclusive.

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

— The value of ath_id shall not be equal to the value of ath_id of any other coded atlas tile unit of the same coded atlas frame.

— The tiles of an atlas frame shall be in increasing order of their ath_id values.

ath_type specifies the coding type of the current atlas tile group according to Table 6. The value of ath_type shall be equal to 0, 1, or 2 in bitstreams conforming to this edition of this document. Other values of ath_type are reserved for future use by ISO/IEC. Decoders conforming to this edition of this document shall ignore reserved values of ath_type.

Table 6 — Name association to ath_type

ath_type	Name of ath_type
0	P_TILE (Inter atlas tile)
1	I_TILE (Intra atlas tile)
2	SKIP_TILE (SKIP atlas tile)
3 - ...	RESERVED

ath_atlas_output_flag affects the decoded atlas output and removal processes as specified in Annex E. When ath_atlas_output_flag is not present, it is inferred to be equal to 1.

ath_atlas_frm_order_cnt_lsb specifies the atlas frame order count modulo MaxAtlasFrmOrderCntLsb for the current atlas tile. The length of the ath_atlas_frm_order_cnt_lsb syntax element is equal to Log2MaxAtlasFrmOrderCntLsb bits. The value of ath_atlas_frm_order_cnt_lsb shall be in the range of 0 to MaxAtlasFrmOrderCntLsb − 1, inclusive.

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

ath_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. The syntax element ath_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 ath_ref_atlas_frame_list_idx is inferred to be equal to 0. The value of ath_ref_atlas_frame_list_idx shall be in the range of 0 to asps_num_ref_atlas_frame_lists_in_asps − 1, inclusive. When ath_ref_atlas_frame_list_asps_flag is equal to 1 and asps_num_ref_atlas_frame_lists_in_asps is equal to 1, the value of ath_ref_atlas_frame_list_idx is inferred to be equal to 0.

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

$$\text{RlsIdx} = \text{ath\_ref\_atlas\_frame\_list\_asps\_flag} \; ? \; \text{ath\_ref\_atlas\_frame\_list\_idx} : \text{asps\_num\_ref\_atlas\_frame\_lists\_in\_asps} \quad (21)$$

ath_additional_afoc_lsb_present_flag[j] equal to 1 specifies that ath_additional_afoc_lsb_val[j] is present for the current atlas tile. ath_additional_afoc_lsb_present_flag[j] equal to 0 specifies that ath_additional_afoc_lsb_val[j] is not present.

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

$$\text{FullAtlasFrmOrderCntLsbLt}[\,\text{RlsIdx}\,][\,j\,] = \text{ath\_additional\_afoc\_lsb\_val}[\,j\,] * \text{MaxAtlasFrmOrderCntLsb} + \text{afoc\_lsb\_lt}[\,\text{RlsIdx}\,][\,j\,] \quad (22)$$

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

ath_pos_min_d_quantizer specifies the quantizer that is to be applied to the pdu_3d_offset_d[tileID][p] value of the patch with index p in the tile with tile ID equal to tileID. If ath_pos_min_d_quantizer is not present, its value shall be inferred to be equal to 0.

ath_pos_delta_max_d_quantizer specifies the quantizer that is to be applied to the pdu_3d_range_d[tileID][p] value of the patch with index p in the tile with tile ID equal to tileID. If ath_pos_delta_max_d_quantizer is not present, its value shall be inferred to be equal to 0.

ath_patch_size_x_info_quantizer specifies the value of the quantizer PatchSizeXQuantizer that is to be applied to the syntax elements pdu_2d_size_x_minus1[tileID][p], mpdu_2d_delta_size_x[tileID][p], ipdu_2d_delta_size_x[tileID][p], rpdu_2d_size_x_minus1[tileID][p], and epdu_2d_size_x_minus1[tileID][p] of a patch with index p in a tile with tile ID equal to tileID. If ath_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 << ath_patch_size_x_info_quantizer (23)

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

ath_patch_size_y_info_quantizer specifies the value of the quantizer PatchSizeYQuantizer that is to be applied to the syntax elements pdu_2d_size_y_minus1[tileID][p], mpdu_2d_delta_size_y[tileID][p], ipdu_2d_delta_size_y[tileID][p], rpdu_2d_size_y_minus1[tileID][p], and epdu_2d_size_y_minus1[tileID][p] of a patch with index p in a tile with tile ID equal to tileID. If ath_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 << ath_patch_size_y_info_quantizer (24)

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

ath_raw_3d_offset_axis_bit_count_minus1 plus 1 specifies the number of bits in the fixed-length representation of the syntax elements rpdu_3d_offset_u[tileID][p], rpdu_3d_offset_v[tileID][p], and rpdu_3d_offset_d[tileID][p] of a patch with index p in a tile with tile ID equal to tileID. When present, the length of the syntax element used to represent ath_raw_3d_offset_axis_bit_count_minus1 is equal to Floor(Log2(asps_geometry_3d_bit_depth_minus1 + 1)).

When ath_raw_3d_offset_axis_bit_count_minus1 is not present, its value shall be inferred to be equal to
Max(0, asps_geometry_3d_bit_depth_minus1 − asps_geometry_2d_bit_depth_minus1) − 1.

A variable RawShift is defined as follows:

If afps_raw_3d_offset_bit_count_explicit_mode_flag is equal to 1,

RawShift =
asps_geometry_3d_bit_depth_minus1 − ath_raw_3d_offset_axis_bit_count_minus1 (25)

Otherwise

RawShift = asps_geometry_2d_bit_depth_minus1 + 1 (26)

ath_num_ref_idx_active_override_flag equal to 1 specifies that the syntax element ath_num_ref_idx_active_minus1 is present for the current atlas tile. ath_num_ref_idx_active_override_flag equal to 0 specifies that the syntax element ath_num_ref_idx_active_minus1 is not present. If ath_num_ref_idx_active_override_flag is not present, its value shall be inferred to be equal to 0.

ath_num_ref_idx_active_minus1 is used for the derivation of the variable NumRefIdxActive as specified by Formula (27) for the current atlas tile. The value of ath_num_ref_idx_active_minus1 shall be in the range of 0 to 14, inclusive.

When the current atlas tile is a P_TILE atlas tile, ath_num_ref_idx_active_override_flag is equal to 1, and ath_num_ref_idx_active_minus1 is not present, ath_num_ref_idx_active_minus1 is inferred to be equal to 0.

The variable NumRefIdxActive is derived as follows:

```
if( ath_type == P_TILE || ath_type == SKIP_TILE ) {
    if( ath_num_ref_idx_active_override_flag == 1 )
        NumRefIdxActive = ath_num_ref_idx_active_minus1 + 1                                  (27)
    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
```

NumRefIdxActive minus 1 specifies the maximum value of the atlas reference frame index that may be used to decode the current atlas tile.

8.4.6.12 Reference list structure semantics

num_ref_entries[rlsIdx] specifies the number of entries in the ref_list_struct(rlsIdx) syntax structure, where rlsIdx is the index of an atlas frame reference list. For P_TILE and SKIP_TILE, the value of num_ref_entries[rlsIdx] shall be in the range of 1 to asps_max_dec_atlas_frame_buffering_minus1 + 1. Otherwise, the value of num_ref_entries[rlsIdx] shall be in the range of 0 to asps_max_dec_atlas_frame_buffering_minus1 + 1.

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:

```
NumLtrAtlasFrmEntries[ rlsIdx ] = 0
for( i = 0; i < num_ref_entries[ rlsIdx ]; i++ )
    if( !st_ref_atlas_frame_flag[ rlsIdx ][ i ] )                                            (28)
        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 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.

straf_entry_sign_flag[rlsIdx][i] equal to 1 specifies that the i-th entry in the syntax structure ref_list_struct(rlsIdx) has a value greater than or equal to 0. straf_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 straf_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 * straf_entry_sign_flag[ rlsIdx ][ i ] – 1 ) * abs_delta_afoc_st[ rlsIdx ][ i ]        (29)
    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 Log2MaxAtlasFrmOrderCntLsb bits.

8.4.7 Atlas tile data unit semantics

8.4.7.1 General atlas tile data unit semantics

atdu_patch_mode[tileID][p] indicates the patch mode for the patch with index p in the current atlas tile with tile ID equal to tileID. The permitted values for atdu_patch_mode[tileID][p] are specified in Table 7 for atlas tiles with ath_type equal to I_TILE, Table 8 for atlas tiles with ath_type equal to P_TILE, and Table 9 for atlas tiles with ath_type equal to SKIP_TILE. When not present, the value of atdu_patch_mode[tileID][p] is inferred to be equal to P_SKIP. Modes indicated as reserved shall not be present in bitstreams conforming to this edition of this document.

NOTE A tile with ath_type equal to SKIP_TILE implies that the entire tile information is copied directly from the tile with the same id as that of the current tile that corresponds to the first reference atlas frame in the RefAtlasFrmList.

Table 7 — Patch modes for I_TILE type atlas tiles

atdu_patch_mode[tileID][p]	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 for future use by ISO/IEC
14	I_END	Patch termination mode

Table 8 — Patch modes for P_TILE type atlas tiles

atdu_patch_mode[tileID][p]	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 for future use by ISO/IEC
14	P_END	Patch termination mode

Table 9 — Patch modes for SKIP_TILE type atlas tiles

atdu_patch_mode[tileID][p]	Identifier	Description
0	P_SKIP	Patch Skip mode

Table 10 — Patch types

Value	Identifier	Description
0	PROJECTED	patch with patch mode equal to I_INTRA, P_INTRA, P_INTER, P_MERGE, or P_SKIP that is associated with projected information onto a 2D image
1	EOM	patch with patch mode equal to I_EOM, P_EOM, or P_SKIP associated with EOM coded points
2	RAW	patch with patch mode equal to I_RAW, P_RAW, or P_SKIP associated with RAW coded points

8.4.7.2 Patch information data semantics

None

8.4.7.3 Patch data unit semantics

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

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

pdu_2d_size_x_minus1[tileID][p] plus 1 specifies the quantized width value of the patch with index p in the current atlas tile, with tile ID equal to tileID.

pdu_2d_size_y_minus1[tileID][p] plus 1 specifies the quantized height value of the patch with index p in the current atlas tile, with tile ID equal to tileID.

pdu_3d_offset_u[tileID][p] specifies the shift to be applied to the reconstructed patch points in patch with index p of the current atlas tile, with tile ID equal to tileID, along the tangent axis. The value of pdu_3d_offset_u[tileID][p] shall be in the range of 0 to $2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1} + 1} - 1$, inclusive. The number of bits used to represent pdu_3d_offset_u[tileID][p] is asps_geometry_3d_bit_depth_minus1 + 1.

pdu_3d_offset_v[tileID][p] specifies the shift to be applied to the reconstructed patch points in patch with index p of the current atlas tile, with tile ID equal to tileID, along the bi-tangent axis. The value of pdu_3d_offset_v[tileID][p] shall be in the range of 0 to $2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1} + 1} - 1$, inclusive. The number of bits used to represent pdu_3d_offset_v[tileID][p] is asps_geometry_3d_bit_depth_minus1 + 1.

pdu_3d_offset_d[tileID][p] specifies the shift to be applied to the reconstructed patch points in patch with index p of the current atlas tile, with tile ID equal to tileID, along the normal axis, Pdu3dOffsetD[tileID][p], as follows:

$$\text{Pdu3dOffsetD[tileID][p]} = \text{pdu\_3d\_offset\_d[tileID][p]} << \text{ath\_pos\_min\_d\_quantizer} \quad (30)$$

The value of Pdu3dOffsetD[tileID][p] shall be in the range of 0 to $2^{(\text{ asps\_geometry\_3d\_bit\_depth\_minus1} + 1\,)} - 1$, inclusive.

The number of bits used to represent pdu_3d_offset_d[tileID][p] is equal to (asps_geometry_3d_bit_depth_minus1 − ath_pos_min_d_quantizer + 1).

pdu_3d_range_d[tileID][p], if present, specifies the nominal maximum value of the shift expected to be present in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation, in patch with index p of the current atlas tile, with tile ID equal to tileID, along the normal axis, Pdu3dRangeD[tileID][p], as follows:

```
if( pdu_3d_range_d[ tileID ][ p ] == 0 )
        Pdu3dRangeD[ tileID ][ p ] = 0                                        (31)
else {
        range = pdu_3d_range_d[ tileID ][ p ] << ath_pos_delta_max_d_quantizer
         Pdu3dRangeD[ tileID ][ p ] = range − 1                               (32)
}
```

Let the variable rangeDBitDepth be computed as follows:

```
  rangeDBitDepth =
        Min( asps_geometry_2d_bit_depth_minus1, asps_geometry_3d_bit_depth_minus1 ) + 1
```

If pdu_3d_range_d[tileID][p] is not present the value of Pdu3dRangeD[tileID][p] is assumed to be equal to $2^{\text{rangeDBitDepth}} - 1$. When present, the value of Pdu3dRangeD[tileID][p] shall be in the range of 0 to $2^{\text{rangeDBitDepth}} - 1$, inclusive.

The number of bits used to represent pdu_3d_range_d[tileID][p] is equal to (rangeDBitDepth − ath_pos_delta_max_d_quantizer).

pdu_projection_id[tileID][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, with tile ID equal to tileID. The value of pdu_projection_id[tileID][p] shall be in range of 0 to asps_max_number_projections_minus1, inclusive.

The number of bits used to represent pdu_projection_id[tileID][p] is Ceil(Log2(asps_max_number_projections_minus1 + 1)).

pdu_orientation_index[tileID][p] specifies the patch orientation index, for the patch with index p of the current atlas tile, with tile ID equal to tileID, used to determine the transformation matrices, $\mathbf{R_o}$ and $\mathbf{R_s}$, as indicated in Table 11, that are to be used to transform atlas coordinates of a patch to a local patch coordinate system denoted by coordinates (u, v), before conversion to 3D space coordinates. The number of bits used to represent pdu_orientation_index[tileID][p] is (asps_use_eight_orientations_flag ? 3 : 1).

pdu_lod_enabled_flag[tileID][p] equal to 1 specifies that the LOD parameters are present for the current patch p of the current atlas tile, with tile ID equal to tileID. If pdu_lod_enabled_flag[tileID][p] is equal to 0, no LOD parameters are present for the current patch. If pdu_lod_enabled_flag[tileID][p] is not present, its value shall be inferred to be equal to 0.

pdu_lod_scale_x_minus1[tileID][p] specifies the LOD scaling factor to be applied to the local x-coordinate of a point in a patch with index p of the current atlas tile, with tile ID equal to tileID, prior to its addition to the patch coordinate TilePatch3dOffsetU[tileID][p]. If pdu_lod_scale_x_minus1[tileID][p] is not present, its value shall be inferred to be equal to 0.

pdu_lod_scale_y_idc[tileID][p] indicates the LOD scaling factor to be applied to the local y-coordinate of a point in a patch with index p of the current atlas tile, with tile ID equal to tileID, prior to its addition to the patch coordinate TilePatch3dOffsetV[tileID][p]. If pdu_lod_scale_y_idc[tileID][p] is not present, its value shall be inferred to be equal to 0.

8.4.7.4 Skip patch data unit semantics

None

8.4.7.5 Merge patch data unit semantics

mpdu_ref_index[tileID][p] specifies the atlas reference frame index, refIdx, for the current patch with index p of the current atlas tile, with tile ID equal to tileID. The value of mpdu_ref_index[tileID][p] shall be in the range of 0 to NumRefIdxActive − 1, inclusive. When mpdu_ref_index[tileID][p] is not present, it is inferred to be equal to 0.

mpdu_override_2d_params_flag[tileID][p] specifies whether the 2D parameters for the current patch with index p of the current atlas tile, with tile ID equal to tileID, are present in the bitstream.

mpdu_2d_pos_x[tileID][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, with tile ID equal to tileID, and of the x-coordinate of the top-left corner of the patch bounding box of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx, expressed as a multiple of PatchPackingBlockSize. When mpdu_2d_pos_x[tileID][p] is not present, it is inferred to be equal to 0.

mpdu_2d_pos_y[tileID][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, with tile ID equal to tileID, and of the y-coordinate of the top-left corner of the patch bounding box of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx, expressed as a multiple of PatchPackingBlockSize. When mpdu_2d_pos_x[tileID][p] is not present, it is inferred to be equal to 0.

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

mpdu_2d_delta_size_y[tileID][p] specifies the difference of the height values of the patch with index p in the current atlas tile, with tile ID equal to tileID, and the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to the reference refIdx. When mpdu_2d_delta_size_y[tileID][p] is not present, it is inferred to be equal to 0.

mpdu_override_3d_params_flag[tileID][p] specifies whether the 3D parameters for the current patch of the current atlas tile, with tile ID equal to tileID, are present in the bitstream. When mpdu_override_3d_params_flag[tileID][p] is not present, it is inferred to be equal to 0.

mpdu_3d_offset_u[tileID][p] specifies the difference between the shift to be applied to the reconstructed patch points along the tangent axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the tangent axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to the reference refIdx. The value of mpdu_3d_offset_u[tileID][p] shall be in the range of ($-2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1}+1} + 1$) to ($2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1}+1} - 1$), inclusive. When mpdu_3d_offset_u[tileID][p] is not present, it is inferred to be equal to 0.

mpdu_3d_offset_v[tileID][p] specifies the difference between the shift to be applied to the reconstructed patch points along the bi-tangent axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the bi-tangent axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx. The value of mpdu_3d_offset_v[tileID][p] shall be in the range of ($-2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1}+1} + 1$) to ($2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1}+1} - 1$), inclusive. When mpdu_3d_offset_v[tileID][p] is not present, it is inferred to be equal to 0.

mpdu_3d_offset_d[tileID][p] specifies the difference between the shift to be applied to the reconstructed patch points along the normal axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the normal axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx, Mpdu3dOffsetD[tileID][p], as follows:

$$\text{Mpdu3dOffsetD[tileID][p]} = \text{mpdu\_3d\_offset\_d[tileID][p]} \ll \text{ath\_pos\_min\_d\_quantizer} \tag{33}$$

The value of mpdu_3d_offset_d[tileID][p] shall be in the range of ($-2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1}+1} + 1$) to ($2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1}+1} - 1$), inclusive. When mpdu_3d_offset_d[tileID][p] is not present, it is inferred to be equal to 0.

mpdu_3d_range_d[tileID][p], if present, specifies the difference between the nominal maximum value of the shift expected to be present in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation, in the patch with index p of the current atlas tile, with tile ID equal to tileID, along the normal axis and of the nominal maximum value of the shift expected to be present in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx, Mpdu3dRangeD[tileID][p], as follows:

$$\begin{aligned}&\text{Mpdu3dRangeD[tileID][p]} = (\text{mpdu\_3d\_range\_d[tileID][p]} == 0)\ ?\\&\qquad 0 : (\text{mpdu\_3d\_range\_d[tileID][p]} \ll \text{ath\_pos\_delta\_max\_d\_quantizer}) - 1\end{aligned} \tag{34}$$

If mpdu_3d_range_d[tileID][p] is not present the value of Mpdu3dRangeD[tileID][p] is assumed to be equal to 0.

mpdu_override_plr_flag[tileID][p] specifies whether the point local reconstruction parameters may be overwritten by new parameters that are present in the bitstream for the patch with index p of the current atlas tile, with tile ID equal to tileID.

8.4.7.6 Inter patch data unit semantics

ipdu_ref_index[tileID][p] specifies the atlas reference frame index, refIdx , for the current patch with index p of the current atlas tile, with tile ID equal to tileID. The value of ipdu_ref_index[tileID][p] shall be in the range of 0 to NumRefIdxActive − 1, inclusive. When ipdu_ref_index[tileID][p] is not present, it is inferred to be equal to 0.

ipdu_patch_index[tileID][p] specifies the index, RefPatchIdx, of the patch with index p in the atlas tile, with tile ID equal to tileID, with the same ID as the current tile address in the atlas frame that corresponds to index refIdx in the current reference atlas frame list.

ipdu_2d_pos_x[tileID][p] specifies the difference of the x-coordinate of the top-left corner of the patch bounding box of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the x-coordinate of the top-left corner of the patch bounding box of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx, expressed as a multiple of PatchPackingBlockSize.

ipdu_2d_pos_y[tileID][p] specifies the difference of the y-coordinate of the top-left corner of the patch bounding box of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the y-coordinate of the top-left corner of the patch bounding box of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx, expressed as a multiple of PatchPackingBlockSize.

ipdu_2d_delta_size_x[tileID][p] specifies the difference of the width values of the patch with index p in the current atlas tile, with tile ID equal to tileID, and the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx.

ipdu_2d_delta_size_y[tileID][p] specifies the difference of the height values of the patch with index p in the current atlas tile, with tile ID equal to tileID, and the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to the reference refIdx.

ipdu_3d_offset_u[tileID][p] specifies the difference between the shift to be applied to the reconstructed patch points along the tangent axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the tangent axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to the reference refIdx. The value of ipdu_3d_offset_u[tileID][p] shall be in the range of $(-2^{\text{asps\_geometry\_3d\_bit depth\_minus1+1}} + 1)$ to $(2^{\text{asps\_geometry\_3d\_bit depth\_minus1+1}} - 1)$, inclusive.

ipdu_3d_offset_v[tileID][p] specifies the difference between the shift to be applied to the reconstructed patch points along the bi-tangent axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the bi-tangent axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx. The value of ipdu_3d_offset_v[tileID][p] shall be in the range of $(-2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1+1}} + 1)$ to $(2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1+1}} - 1)$, inclusive.

ipdu_3d_offset_d[tileID][p] specifies the difference between the shift to be applied to the reconstructed patch points along the normal axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the normal axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx, Ipdu3dOffsetD[tileID][p], as follows:

Ipdu3dOffsetD[tileID][p] = ipdu_3d_offset_d[tileID][p] << ath_pos_min_d_quantizer (35)

The value of ipdu_3d_offset_d[tileID][p] shall be in the range of $(-2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1+1}} + 1)$ to $(2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1+1}} - 1)$, inclusive.

ipdu_3d_range_d[tileID][p], if present, specifies the difference between the nominal maximum value of the shift expected to be present in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation, in the patch with index p of the current atlas, with tile ID equal to tileID, tile along the normal axis and of the nominal maximum value of the shift expected to be presented in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx, Ipdu3dRangeD[tileID][p], as follows:

```
Ipdu3dRangeD[ tileID ][ p ] = ( ipdu_3d_range_d[ tileID ][ p ] == 0 ) ?
              0 : ( ipdu_3d_range_d[ tileID ][ p ] << ath_pos_delta_max_d_quantizer ) – 1          (36)
```

If ipdu_3d_range_d[tileID][p] is not present, the value of Ipdu3dRangeD[tileID][p] is assumed to be equal to 0.

8.4.7.7 RAW patch data unit semantics

rpdu_patch_in_auxiliary_video_flag[tileID][p] specifies whether the geometry and attribute data associated with the RAW coded patch with index p in the current atlas tile, with tile ID equal to tileID, are encoded in an auxiliary video sub-bitstream. If rpdu_patch_in_auxiliary_video_flag[tileID][p] is equal to 0, the geometry and attribute data associated with the RAW coded patch p in the current atlas tile are encoded in the corresponding 1st non auxiliary map video sub-bitstream. If rpdu_patch_in_auxiliary_video_flag[tileID][p] is equal to 1, the geometry and attribute data associated with the RAW coded patch p in the current atlas tile are encoded in the corresponding auxiliary video sub-bitstream. If rpdu_patch_in_auxiliary_video_flag[tileID][p] is not present, its value shall be inferred to be equal to 0.

rpdu_2d_pos_x[tileID][p] specifies the x-coordinate of the top-left corner of the patch bounding box size for the RAW coded patch p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

rpdu_2d_pos_y[tileID][p] specifies the y-coordinate of the top-left corner of the patch bounding box size for the RAW coded patch p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

rpdu_2d_size_x_minus1[tileID][p] plus 1 specifies the width value of the RAW coded patch with index p in the current atlas tile, with tile ID equal to tileID.

rpdu_2d_size_y_minus1[tileID][p] plus 1 specifies the height value of the RAW coded patch with index p in the current atlas tile, with tile ID equal to tileID.

rpdu_3d_offset_u[tileID][p] specifies the shift to be applied to the reconstructed RAW patch points in the patch with index p of the current atlas tile, with tile ID equal to tileID, along the tangent axis. The number of bits used to represent rpdu_3d_offset_u[tileID][p] is (ath_raw_3d_offset_axis_bit_count_minus1 + 1).

rpdu_3d_offset_v[tileID][p] specifies the shift to be applied to the reconstructed RAW patch points in the patch with index p of the current atlas tile, with tile ID equal to tileID, along the bi-tangent axis. The number of bits used to represent rpdu_3d_offset_v[tileID][p] is (ath_raw_3d_offset_axis_bit_count_minus1 + 1).

rpdu_3d_offset_d[tileID][p] specifies the shift to be applied to the reconstructed RAW patch points in the patch with index p of the current atlas tile, with tile ID equal to tileID, along the normal axis. The number of bits used to represent rpdu_3d_offset_d[tileID][p] is (ath_raw_3d_offset_axis_bit_count_minus1 + 1).

rpdu_points_minus1[tileID][p] plus 1 specifies the number of RAW points present in the RAW coded patch with index p in the current atlas tile, with tile ID equal to tileID. The value of rpdu_points_minus1[tileID][p] shall be in the range of 0 to ((rpdu_2d_size_x_minus1[tileID][p] + 1) * (rpdu_2d_size_y_minus1[tileID][p] + 1)) / 3 − 1, inclusive.

8.4.7.8 EOM patch data unit semantics

epdu_patch_in_auxiliary_video_flag[tileID][p] specifies whether the attribute data associated with the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID, are encoded in an auxiliary video sub-bitstream. If epdu_patch_in_auxiliary_video_flag[tileID][p] is equal to 0, the attribute data associated with the EOM patch with index p in the current atlas tile are encoded in the corresponding 1st non auxiliary map video sub-bitstream. If epdu_patch_in_auxiliary_video_flag[tileID][p] is equal to 1, the attribute data associated with the current EOM patch with index p in the current atlas tile are encoded in the corresponding auxiliary video sub-bitstream. If epdu_patch_in_auxiliary_video_flag[tileID][p] is not present, its value shall be inferred to be equal to 0.

epdu_2d_pos_x[tileID][p] specifies the x-coordinate of the top-left corner of the patch bounding box size for the current EOM patch p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

epdu_2d_pos_y[tileID][p] specifies the y-coordinate of the top-left corner of the patch bounding box size for the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

epdu_2d_size_x_minus1[tileID][p] plus 1 specifies the width value of the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID.

epdu_2d_size_y_minus1[tileID][p] plus 1 specifies the height value of the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID.

epdu_patch_count_minus1[tileID][p] plus 1 indicates the number of associated patches that contain EOM points in the decoded occupancy video for the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID. When not present epdu_patch_count_minus1[tileID][p] is inferred to be equal to 0.

NOTE An EOM patch infers additional geometry information from the occupancy. If attributes are present, additional information for the attribute data exists in the EOM associated patches in the map with index 0 of the corresponding attributes.

epdu_associated_patch_idx[tileID][p][i] indicates the associated patch index of the i-th patch specified in the current EOM patch with index p, in the current atlas tile with tile ID equal to tileID, which is associated with a number of EOM attribute points as indicated by the syntax element epdu_points[tileID][p][i]. The value of epdu_associated_patch_idx[tileID][p][i] shall be in the range 0 to AtduTotalNumPatches[tileID] − 1, inclusive.

It is a requirement of bitstream conformance that, if ath_type is equal to I_TILE, that the value of atdu_patch_mode[tileID][epdu_associated_patch_idx[tileID][p][i]] shall not be equal to I_RAW or I_EOM, and, if ath_type is equal to P_TILE, that the value of atdu_patch_mode[tileID][epdu_associated_patch_idx[tileID][p][i]] shall not be equal to P_RAW or P_EOM.

epdu_points[tileID][p][i] specifies the number of EOM points associated with the i-th patch specified in the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID, that contain EOM attributes. When not present epdu_points[tileID][p][i] is inferred to be equal to 0.

8.4.7.9 Point local reconstruction data semantics

Let the variable BSize be equal to PatchPackingBlockSize. Then the function BlockCnt(xSize, ySize) is defined as follows:

$$\text{BlockCnt(xSize, ySize)} = ((\text{xSize} + \text{BSize} - 1) / \text{BSize}) * ((\text{ySize} + \text{BSize} - 1) / \text{BSize}) \quad (37)$$

This function can be used to determine the number of blocks of size PatchPackingBlockSize × PatchPackingBlockSize that are associated with a patch. For example, for a patch p of size TilePatch2dSizeX[tileID][p] × TilePatch2dSizeY[tileID][p], the number of blocks is equal to BlockCnt(TilePatch2dSizeX[tileID][p], TilePatch2dSizeY[tileID][p]).

NOTE 1 In order to parse point local reconstruction data, the TilePatch2dSizeX[tileID][p] and TilePatch2dSizeY[tileID][p] information needs to be previously derived.

plrd_level[tileID][p][mapIdx] indicates the level at which point local reconstruction data, when enabled, is present in the bitstream for a map, with index mapIdx, associated with the current patch with index p. in the current atlas tile, with tile ID equal to tileID. When plrd_level[tileID][p][mapIdx] is equal to 0 this syntax element indicates that point local reconstruction data and the syntax elements plrd_present_block_flag[tileID][p][mapIdx][i] and plrd_block_mode_minus1[tileID][p][mapIdx][i] are present in the bitstream for each block of the map. When plrd_level[tileID][p][mapIdx] is equal to 1, this syntax element indicates that point local reconstruction data and the syntax elements plrd_present_flag[tileID][p][mapIdx] and plrd_mode_minus1[tileID][p][mapIdx] are present in the bitstream for the entire map, with index mapIdx, of the patch p in the current atlas tile with tile ID equal to tileID. In this case, all blocks of the map mapIdx shall use the same point local reconstruction data.

plrd_present_block_flag[tileID][p][mapIdx][i] equal to 1 indicates that the syntax element plrd_block_mode_minus1[tileID][p][mapIdx][i] is present in the bitstream for block i of the map mapIdx of the patch p in the current atlas tile, with tile ID equal to tileID. plrd_present_block_flag[tileID][p][mapIdx][i] equal to 0 indicates that the syntax element plrd_block_mode_minus1[tileID][p][mapIdx][i] is not present in the bitstream.

plrd_block_mode_minus1[tileID][p][mapIdx][i] plus 1 indicates the point local reconstruction mode, plrBlkMode, selected from the list of point local reconstruction modes of size plri_number_of_modes_minus1[mapIdx] plus 1 and defined by the syntax elements plri_interpolate_flag[mapIdx][plrBlkMode], plri_filling_flag[mapIdx][plrBlkMode], plri_minimum_depth[mapIdx][plrBlkMode], and plri_neighbour_minus1[mapIdx][plrBlkMode] for block i of the map with index mapIdx of the patch p in the current atlas tile, with tile ID equal to tileID. plrd_block_mode_minus1[tileID][p][mapIdx][i] shall be in the range of 0 to plri_number_of_modes_minus1[mapIdx], inclusive.

NOTE 2 When PLR mode is equal to 0 at the block or patch level, it is not signalled in the bitstream. The value 0 corresponds to a default mode value of PLR which is implemented by the values of the syntax elements plri_interpolate_flag[mapIdx][0], plri_filling_flag[mapIdx][0], plri_minimum_depth[mapIdx][0], and plri_neighbour_minus1[mapIdx][0].

plrd_present_flag[tileID][p][mapIdx] equal to 1 indicates that the syntax element plrd_mode_minus1[tileID][p][mapIdx] is present in the bitstream for the map with index mapIdx of the patch p in the current atlas tile, with tile ID equal to tileID. plrd_present_flag[tileID][p][mapIdx] equal to 0 indicates that the syntax element plrd_block_mode_minus1[tileID][p][mapIdx][i] is not present in the bitstream.

plrd_mode_minus1[tileID][p][mapIdx] plus 1 indicates the point local reconstruction mode, plrMode, selected from the list of point local reconstruction modes of size plri_number_of_modes_minus1[mapIdx] plus 1 and defined by the syntax elements plri_interpolate_flag[mapIdx][plrMode], plri_filling_flag[mapIdx][plrMode], plri_minimum_depth[mapIdx][plrMode], and plri_neighbour_minus1[mapIdx][plrMode] for all blocks of the map with index mapIdx of the patch p in the current atlas tile, with tile ID equal to tileID. plrd_mode_minus1[tileID][p][mapIdx] shall be in the range of 0 to plri_number_of_modes_minus1[mapIdx], inclusive.

8.4.8 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 F. 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.

9 Decoding process

9.1 General decoding process

Input to this process is a V3C bitstream or a collection of V3C sub-bitstreams.

Output of this process is the decoded atlas information, a set of decoded video streams, corresponding to the occupancy, geometry and attribute, if available, V3C components and other information associated with the CVS, all of which are needed to perform the 3D reconstruction process as specified in Clause 10.

The decoding process is specified such that all decoders that conform to a specific image/video coding specification and, potentially, a specific profile, tier and level that may be defined within such specification, 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 document.

If the input to this process is a V3C bitstream then the V3C bitstream is parsed and demultiplexed into several V3C sub-bitstreams, as specified in subclause 9.6.2 or by external means, with each sub-bitstream corresponding to a different set of information. In particular, the V3C bitstream is demultiplexed into one or more atlas bitstreams, if multiple atlases are present, and their associated video bitstreams. Video bitstreams for each atlas may include occupancy, geometry and attribute components.

For each atlas, identified by a variable DecAtlasID, which is set equal to vuh_atlas_id or determined through external means if the V3C unit header is unavailable, the following decoding processes are invoked on each of its associated V3C sub-bitstream component:

— The decoding process for the V3C sub-bitstream component corresponding to the atlas component, which is determined through either examining if vuh_unit_type is equal to V3C_AD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.2. This V3C sub-bitstream is also referred to as AtlasBitstreamToDecode in this document.

— The decoding process for the V3C sub-bitstream component corresponding to the occupancy component, which is determined through either examining if vuh_unit_type is equal to V3C_OVD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.3.

— The decoding process for the V3C sub-bitstream component corresponding to the geometry components, which is determined through either examining if vuh_unit_type is equal to V3C_GVD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.4.

— The decoding process for the V3C sub-bitstream component corresponding to the attribute components, which is determined through either examining if vuh_unit_type is equal to V3C_AVD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.5.

The decoding process may include additional processing steps required to convert the decoded video streams to a nominal format, i.e. a nominal resolution, bit depth, chroma format, etc.. Such conversion process is outside the scope of this document. However, for purposes of conformance to point B, as defined in Annex A, an example process is described in Annex B.

Outputs defined in subclauses 9.2, 9.3, 9.4 and 9.5 may have an assigned composition time and output order index. The composition time and the output order index may be provided either in the bitstream or by external means not specified in this document. The composition time and the output order index may be utilized to determine timing relationships of and synchronize V3C components, and to perform operations as defined in Annex B, if required.

NOTE For example, for a video sub-bitstream coded using ISO/IEC 23008-2, the output order index can be associated with the picture order count value of each frame in the sub-bitstream, as specified in ISO/IEC 23008-2.

9.2 Atlas data decoding process

9.2.1 General atlas data decoding process

Input to this process is the atlas sub-bitstream corresponding to the atlas NAL units of the atlas component with atlas ID DecAtlasID of the V3C bitstream.

Outputs of this process are:

— the number of decoded atlas frames, NumDecAtlasFrames,

— for each decoded atlas frame, the corresponding upper-case variables from subclauses 9.2.5 or 9.2.7, or both,

— for each decoded atlas frame, the associated geometry, attribute and occupancy parameters contained in non-ACL NAL units, and

— the following 1D arrays of size NumDecAtlasFrames:

  — DecAtlasOutOrdIdx, indicating the atlas output order index, and

  — DecAtlasCompTime, indicating the atlas composition time.

If the atlas sub-bitstream is encapsulated using the V3C sample stream format, as defined in Annex C, then the composition time can be derived using the atlas frame order count, AFOC, or atlas frame timing information, if available. If the atlas sub-bistream is encapsulated within the ISO media file format, the atlas composition time, DecAtlasCompTime, for the atlas sub-bitstream is indicated according to ISO/IEC DIS 23090-10. Otherwise, the atlas composition time is provided through external means and it is outside the scope of this document.

Subclause 9.2.2 is repeatedly invoked for each coded atlas frame in the current atlas bitstream to be decoded, in decoding order.

9.2.2 Decoding process for a coded atlas frame

The decoding processes specified in this subclause apply to each coded atlas frame with nal_layer_id equal to 0, referred to as the current atlas frame and denoted by the atlas ID DecAtlasID, in the current atlas component bitstream.

The decoding process for the current atlas frame takes as inputs the syntax elements and upper-case variables from Clause 8. When interpreting the semantics of each syntax element in each NAL unit, the term "the bitstream" (or part thereof, e.g., a CAS of the bitstream) refers to the current atlas component bitstream (or part thereof).

When the current atlas frame is a BLA atlas frame that has nal_unit_type equal to NAL_BLA_W_LP or NAL_GBLA_W_LP, or is a CRA atlas frame, the following applies:

— If some external means not specified in this document is available to set the variable UseAltCabParamsFlag to a value, UseAltCabParamsFlag is set equal to the value provided by the external means.

— Otherwise, the value of UseAltCabParamsFlag is set equal to 0.

When the current atlas frame is an IRAP coded atlas frame, the following applies:

— If the current atlas frame is an IDR atlas frame, a BLA atlas frame, the first atlas frame in the bitstream in decoding order, or the first atlas frame that follows an end of sequence NAL unit in decoding order, the variable NoOutputBeforeRecoveryFlag is set equal to 1.

— Otherwise, if some external means not specified in this document is available to set the variable HandleCraAsBlaFlag to a value for the current atlas frame, the variable HandleCraAsBlaFlag is set equal to the value provided by the external means and the variable NoOutputBeforeRecoveryFlag is set equal to HandleCraAsBlaFlag.

— Otherwise, the variable HandleCraAsBlaFlag is set equal to 0 and the variable NoOutputBeforeRecoveryFlag is set equal to 0.

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

a) The decoding of atlas NAL units is specified in subclause 9.2.3.

b) The processes in subclause 9.2.4 specify the following decoding processes using syntax elements in the atlas tile header layer and above:

   1) Variables and functions relating to the atlas frame order count are derived as specified in subclause 9.2.4.1.

   2) At the beginning of the decoding process for each atlas tile, the reference atlas frame list construction process specified in subclause 9.2.4.3 is invoked for the derivation of the reference atlas frame list, RefAtlasFrmList.

   3) The reference atlas frame marking process in subclause 9.2.4.4 is invoked, wherein reference atlas frames might be marked as "unused for reference" or "used for long-term reference".

   4) When the current atlas frame is a BLA atlas frame or is a CRA atlas frame with NoOutputBeforeRecoveryFlag equal to 1, the decoding process for generating unavailable reference atlas frames specified in subclause 9.2.4.2 is invoked, which needs to be invoked only for the first atlas tile of an atlas frame.

   5) The variable AtlasFrameOutputFlag is set as follows:

      i) If the current atlas frame is a RASL atlas frame and NoOutputBeforeRecoveryFlag of the associated IRAP coded atlas frame is equal to 1, AtlasFrameOutputFlag is set equal to 0.

ii) Otherwise, AtlasFrameOutputFlag is set equal to ath_atlas_output_flag.

c) The processes in subclause 9.2.5 specify the patch decoding processes according to the patch mode as follows:

1) Decoding of intra coded patches is specified in subclause 9.2.5.2.

2) Decoding of skip mode coded patches is specified in subclause 9.2.5.3.

3) Decoding of merge mode coded patches is specified in subclause 9.2.5.4.

4) Decoding of inter coded patches is specified in subclause 9.2.5.5.

5) Decoding of RAW coded patches is specified in subclause 9.2.5.6.

6) Decoding of EOM coded patches is specified in subclause 9.2.5.7.

d) The process in subclause 9.2.6 specifies the creation of the block to patch map after the decoding of all patches present within the current atlas tile.

e) After all atlas tiles associated with the current atlas have been decoded, the current atlas is marked as "used for short-term reference".

f) A process is defined in subclause 9.2.7 that specifies the conversion of tile level patch information to atlas level patch information.

9.2.3 Atlas NAL unit decoding process

Inputs to this process are atlas NAL units of the current atlas frame and their associated non-ACL NAL units.

Outputs of this process are the parsed RBSP syntax structures encapsulated within the atlas NAL units.

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

9.2.4 Atlas tile header decoding process

9.2.4.1 Atlas frame order count derivation process

Output of this process is AtlasFrmOrderCntVal, the atlas frame order count of the current atlas tile.

Atlas frame order counts are used to identify the output order of 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 the current atlas frame is not an IRAP coded atlas with NoOutputBeforeRecoveryFlag equal to 1, the variables prevAtlasFrmOrderCntLsb and prevAtlasFrmOrderCntMsb are derived as follows:

— Let prevAtlasFrm be the previous atlas frame in decoding order that has TemporalID equal to 0 and that is not a RASL, RADL, or SLNR coded atlas.

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

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

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

— If the current atlas is an IRAP coded atlas with NoOutputBeforeRecoveryFlag equal to 1, AtlasFrmOrderCntMsb is set equal to 0.

— Otherwise, AtlasFrmOrderCntMsb is derived as follows:

```
        if( ( ath_atlas_frm_order_cnt_lsb < prevAtlasFrmOrderCntLsb ) &&
              ( ( prevAtlasFrmOrderCntLsb − ath_atlas_frm_order_cnt_lsb ) >=
              ( MaxAtlasFrmOrderCntLsb / 2 ) ) )
    AtlasFrmOrderCntMsb =
        prevAtlasFrmOrderCntMsb + MaxAtlasFrmOrderCntLsb                    (38)
        else if( ( ath_atlas_frm_order_cnt_lsb > prevAtlasFrmOrderCntLsb ) &&
              ( ( ath_atlas_frm_order_cnt_lsb − prevAtlasFrmOrderCntLsb ) >
              ( MaxAtlasFrmOrderCntLsb / 2 ) ) )
    AtlasFrmOrderCntMsb =
        prevAtlasFrmOrderCntMsb − MaxAtlasFrmOrderCntLsb                    (39)
        else
    AtlasFrmOrderCntMsb = prevAtlasFrmOrderCntMsb                           (40)
```

AtlasFrmOrderCntVal is derived as follows:

AtlasFrmOrderCntVal = AtlasFrmOrderCntMsb + ath_atlas_frm_order_cnt_lsb (41)

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 with the same value of nal_layer_id shall not be the same.

The function AtlasFrmOrderCnt(aFrmX) is specified as follows:

AtlasFrmOrderCnt(aFrmX) = AtlasFrmOrderCntVal of the atlas frame aFrmX (42)

The function DiffAtlasFrmOrderCnt(aFrmA, aFrmB) is specified as follows:

DiffAtlasFrmOrderCnt(aFrmA, aFrmB) =
AtlasFrmOrderCnt(aFrmA) − AtlasFrmOrderCnt(aFrmB) (43)

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

NOTE Let X be the current atlas frame 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.

9.2.4.2 Decoding process for generating unavailable reference atlas frames

9.2.4.2.1 General

This process is invoked once per coded atlas frame when the current atlas frame is a BLA atlas frame or a CRA atlas frame with NoOutputBeforeRecoveryFlag equal to 1.

NOTE This process is primarily for the specification of syntax constraints for RASL atlas frames. The entire specification of the decoding process for RASL atlas frames associated with an IRAP coded atlas frame that has NoOutputBeforeRecoveryFlag equal to 1 is included herein only for purposes of specifying constraints on the allowed syntax content of such RASL atlas frames. During the decoding process, any RASL atlas frames associated with an IRAP coded atlas frame that has NoOutputBeforeRecoveryFlag equal to 1 can be ignored, as these atlas frames are not specified for output and have no effect on the decoding process of any other atlas frames that are specified for output. However, in HRD operations as specified in Annex E, RASL access units can need to be taken into consideration in derivation of coded atlas frame buffer (CAB) arrival and removal times.

When this process is invoked, the following applies:

— For each RefAtlasFrmList[j], with j in the range of 0 to num_ref_entries[RlsIdx] – 1, inclusive, that is equal to "no reference atlas frame", an atlas frame is generated as specified in subclause 9.2.4.2.2 and the following applies:

  — The value of nal_layer_id for the generated atlas frame is set equal to the nal_layer_id value of the current atlas frame.

  — If st_ref_atlas_frame_flag[RlsIdx][j] is equal to 1 the value of AtlasFrmOrderCntVal for the generated atlas frame is set equal to RefAtlasFrmList[j] and the generated atlas frame is marked as "used for short-term reference".

  — Otherwise, when st_ref_atlas_frame_flag[RlsIdx][j] is equal to 0 the value of AtlasFrmOrderCntVal for the generated atlas frame is set equal to RefAtlasFrmList[j], the value of ath_atlas_frm_order_cnt_lsb for all tiles in the generated atlas frame is inferred to be equal to (RefAtlasFrmList[j] & (MaxAtlasFrmOrderCntLsb – 1)), and the generated atlas frame is marked as "used for long-term reference".

  — The value of AtlasFrameOutputFlag for the generated reference atlas frame is set equal to 0.

  — RefAtlasFrmList[j] is set to be the generated reference atlas frame.

  — The value of TemporalID for the generated atlas frame is set equal to the TemporalID value of the current atlas frame.

  — The value of ath_atlas_frame_parameter_set_id for the generated atlas frame is set equal to ath_atlas_frame_parameter_set_id of the current atlas frame.

  — The value of nal_layer_id for the generated atlas frame is set equal to nal_layer_id of the current atlas frame.

9.2.4.2.2 Generation of one unavailable atlas frame

When this process is invoked, an unavailable atlas frame is generated as follows:

— For all tiles that are associated with this atlas frame the following applies:

```
for( t = 0; t<= afti_num_tiles_in_atlas_frame_minus1; t++ ) {
    tileID = TileIndexToID( t )
    AtduTotalNumPatches[ tileID ] = MaxNumProjPatches
    for( p = 0; p<= AtduTotalNumPatches[ tileID ]; p++ ) {
        TilePatchType[ tileID ][ p ] = PROJECTED
        TilePatch2dPosX[ tileID ][ p ] = 0
        TilePatch2dPosY[ tileID ][ p ] = 0
        TilePatch2dSizeX[ tileID ][ p ] = 1
```

```
                TilePatch2dSizeY[ tileID ][ p ] = 1
                TilePatch3dOffsetU[ tileID ][ p ] = 0
                TilePatch3dOffsetV[ tileID ][ p ] = 0
                TilePatch3dOffsetD[ tileID ][ p ] = 0
                TilePatch3dRangeD[ tileID ][ p ] = 1
                TilePatchProjectionID[ tileID ][ p ] = 0
                TilePatchOrientationIndex[ tileID ][ p ] = 0
                TilePatchLoDScaleX[ tileID ][ p ] = 1
                TilePatchLoDScaleY[ tileID ][ p ] = 1

                if( asps_plr_enabled_flag == 1 ) {
                    for( m = 0; m < asps_map_count_minus1 + 1; m++ )
                        TilePatchPlrdLevel[ tileID ][ p ][ m ] = 0
                }
            }
        }
```

9.2.4.3 Reference atlas frame list construction process

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

Reference atlas frames are addressed through reference indices. A reference index is an index into a reference atlas frame list (RAFL). When decoding an I_TILE atlas tile, no RAFL is used in decoding of the atlas tile data. When decoding a SKIP_TILE or a P_TILE atlas tile, a single reference atlas frame list, RefAtlasFrmList, is used in decoding of the atlas tile data.

At the beginning of the decoding process for each atlas tile, the RAFL RefAtlasFrmList is derived. The RAFL is used in marking of reference atlas frames as specified in subclause 9.2.4.4 or in decoding of the atlas tile data.

NOTE For an I_TILE atlas tile of an atlas frame, RefAtlasFrmList could be derived for bitstream conformance checking purposes, but its derivation is not necessary for the decoding of the current atlas frame or for atlas frames that follow the current atlas frame in decoding order.

The reference atlas frame list RefAtlasFrmList is constructed as follows:

```
for( j = 0, afocBase = AtlasFrmOrderCntVal; j < num_ref_entries[ RlsIdx ]; j++) {      (44)
    if( st_ref_atlas_frame_flag[ RlsIdx ][ j ] ) {
        RefAtlasFrmAfocList[ j ] = afocBase − DeltaAfocSt[ RlsIdx ][ j ]
        if( there is a reference atlas frame afA in the DAB with
            AtlasFrmOrderCntVal equal to RefAtlasFrmAfocList[ j ] )
            RefAtlasFrmList[ j ] = afA
        else
            RefAtlasFrmList[ j ] = "no reference atlas frame"
        afocBase = RefAtlasFrmAfocList[ j ]
    } else {
        if( there is a reference atlas frame afA in the DAB with
                AtlasFrmOrderCntVal & ( MaxLtAtlasFrmOrderCntLsb − 1 )
                equal to FullAtlasFrmOrderCntLsbLt[ RlsIdx ][ j ] )
            RefAtlasFrmList[ j ] = afA
        else
            RefAtlasFrmList[ j ] = "no reference atlas frame"
    }
}
```

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

If the current tile is a SKIP_TILE, then the array RefAtduTotalNumPatches is set equal to the array AtduTotalNumPatches that corresponds to the first entry in the RefAtlasFrmList, RefAtlasFrmList[0].

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

— num_ref_entries[RlsIdx] shall not be less than NumRefIdxActive.

— The atlas frame referred to by each active entry in RefAtlasFrmList shall be present in the DAB and shall have TemporalID less than or equal to that of the current atlas frame.

— 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 tile 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 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 that have the same nal_layer_id as the current atlas frame. The number of atlas frames in setOfRefAtlasFrms shall be less than or equal to asps_max_dec_atlas_frame_buffering_minus1, and setOfRefAtlasFrms shall be the same for all atlas tiles of an atlas frame.

— The atlas frames referred to by each active entry in RefAtlasFrmList shall have exactly the same number of tiles as the current atlas frame.

— The RefAtlasFrmList of all tiles in the current atlas frame shall contain the same unique atlas frames, without, however, any restrictions in ordering of the reference atlas frames.

— When the current atlas frame, with nal_layer_id equal to a particular value layerID, is an IRAP coded atlas, there shall be no atlas referred to by an entry in RefAtlasFrmList that precedes, in output order or decoding order, any preceding IRAP coded atlas with nal_layer_id equal to layerID in decoding order (when present).

— When the current atlas frame is not a RASL coded atlas associated with a CRA coded atlas with NoOutputBeforeRecoveryFlag equal to 1, there shall be no atlas referred to by an active entry in RefAtlasFrmList that was generated by the decoding process for generating unavailable reference atlas frames for the CRA coded atlas associated with the current atlas.

— When the current atlas frame follows an IRAP coded atlas having the same value of nal_layer_id in both decoding order and output order, there shall be no atlas frame referred to by an active entry in RefAtlasFrmList that precedes that IRAP coded atlas in output order or decoding order.

— When the current atlas frame follows an IRAP coded atlas having the same value of nal_layer_id and all the leading atlas frames, if any, associated with that IRAP coded atlas in both decoding order and output order, there shall be no atlas referred to by an entry in RefAtlasFrmList that precedes that IRAP coded atlas in output order or decoding order.

— When the current atlas frame is a RADL coded atlas, there shall be no active entry in RefAtlasFrmList that is an atlas frame that precedes the associated IRAP coded atlas of the RADL coded atlas in decoding order.

9.2.4.4 Reference atlas frame marking process

This process is invoked once per atlas frame, after decoding of an atlas tile header and the decoding process for RAFL construction for the atlas frame as specified in subclause 9.2.4.3, but prior to the decoding of the atlas tile data. This process might result in one or more reference atlas frames in the DAB being marked as "unused for reference" or "used for long-term reference".

A decoded atlas frame in the DAB 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 nal_layer_id and AtlasFrmOrderCntVal values. Long term reference atlas frames are identified by their nal_layer_id values and 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 atlas frame is marked as "used for short-term reference" and has the same nal_layer_id as the current atlas frame, the atlas frame is marked as "used for long-term reference".

— Each reference atlas frame in the same nal_layer_id as the current atlas frame in the DAB that is not referred to by any entry in RefAtlasFrmList is marked as "unused for reference".

9.2.5 Decoding process for patch data units

9.2.5.1 General decoding process for patch data units

Inputs to this process are the current patch index, p, of the current tile with tile ID equal to tileID, and the current patch mode, atdu_patch_mode[tileID][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.

More specifically the following variables are derived:

TilePatchType[tileID][p], which indicates whether the current patch with patch index p, of the current tile with tile ID equal to tileID, is using a projected, EOM, or RAW patch type.

TilePatchInAuxVideo[tileID][p] indicates whether the current patch with patch index p, of the current tile with tile ID equal to tileID, is associated with the auxiliary video.

Let the variables horLimit and verLimit be computed as follows:

```
if( TilePatchInAuxVideo[ tileID ][ p ] ) {
    horLimit = AuxVideoWidthNF
    verLimit = AuxTileHeight[ TileIDToIndex[ tileID ] ]
}
else {
    horLimit = TileWidth[ TileIDToIndex[ tileID ] ]
    verLimit = TileHeight[ TileIDToIndex[ tileID ] ]
}
```

TilePatch2dPosX[tileID][p] specifies the x-coordinate of the top-left corner of the patch bounding box for the current patch with patch index p, of the current tile with tile ID equal to tileID, relative to the top left corner of the current tile, in units of atlas samples. The value of TilePatch2dPosX[tileID][p] shall be in the range of 0 to horLimit − 1, inclusive.

TilePatch2dPosY[tileID][p] specifies the y-coordinate of the top-left corner of the patch bounding box for the current patch with patch index p, of the current tile with tile ID equal to tileID, relative to the top left corner of the current tile, in units of atlas samples. The value of TilePatch2dPosY[tileID][p] shall be in the range of 0 to verLimit − 1, inclusive.

TilePatch2dSizeX[tileID][p] specifies the width of the bounding box of the current patch with patch index p, of the current tile with tile ID equal to tileID, in units of atlas samples. The value of TilePatch2dSizeX[tileID][p] shall be in the range of 1 to horLimit, inclusive.

TilePatch2dSizeY[tileID][p] specifies the height of the bounding box of the current patch with patch index p, of the current tile with tile ID equal to tileID, in units of atlas samples. The value of TilePatch2dSizeY[tileID][p] shall be in the range of 1 to verLimit, inclusive.

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

$$\text{TilePatch2dPosX[tileID][p] + TilePatch2dSizeX[tileID][p]} <= \text{horLimit} \tag{45}$$

$$\text{TilePatch2dPosY[tileID][p] + TilePatch2dSizeY[tileID][p]} <= \text{verLimit} \tag{46}$$

If TilePatchType[tileID][p] is equal to EOM then TilePatch2dSizeX[tileID][p] and TilePatch2dSizeY[tileID][p] shall be integer multiples of PatchPackingBlockSize.

TilePatch3dOffsetU[tileID][p] specifies the shift to be applied to the reconstructed patch points in the current patch with patch index p, of the current tile with tile ID equal to tileID, along the tangent axis. The value of TilePatch3dOffsetU[tileID][p] shall be in the range of 0 to ($2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1} + 1}$ – 1), inclusive.

TilePatch3dOffsetV[tileID][p] specifies the shift to be applied to the reconstructed patch points in the current patch with patch index p, of the current tile with tile ID equal to tileID, along the bi-tangent axis. The value of TilePatch3dOffsetV[tileID][p] shall be in the range of 0 to ($2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1} + 1}$ – 1), inclusive.

TilePatch3dOffsetD[tileID][p] specifies the shift to be applied to the reconstructed patch points in the current patch with patch index p, of the current tile with tile ID equal to tileID, along the normal axis. The value of TilePatch3dOffsetD[tileID][p] shall be in the range of 0 to ($2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1} + 1}$ – 1), inclusive.

TilePatch3dRangeD[tileID][p] specifies for the current patch with patch index p, of the current tile with tile ID equal to tileID, the expected range of the geometry data along the normal axis in the corresponding geometry maps. Let the variable rangeDBitDepth be equal to Min(asps_geometry_2d_bit_depth_minus1, asps_geometry_3d_bit_depth_minus1) + 1. The value of TilePatch3dRangeD[tileID][p] shall be in the range of 0 to $2^{\text{rangeDBitDepth}}$ – 1, inclusive.

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

TilePatchOrientationIndex[tileID][p] specifies the patch orientation index to Table 11 for the current patch with patch index p, of the current tile with tile ID equal to tileID.

TilePatchLoDScaleX[tileID][p] specifies the LOD scaling factor to be applied to the tangent axis of the current patch with patch index p, of the current tile with tile ID equal to tileID.

TilePatchLoDScaleY[tileID][p] specifies the LOD scaling factor to be applied to the bi-tangent axis of the current patch with patch index p, of the current tile with tile ID equal to tileID.

TilePatchRawPoints[tileID][p] specifies the number of raw coded points in the current patch with patch index p, of the current tile with tile ID equal to tileID.

TilePatchEomPatchCount[tileID][p] specifies the number of patches that may be associated with the current patch p, when patch p is an EOM patch type.

TilePatchAssociatedPatchIndex[tileID][p][i] specifies the index of the i-th patch associated with the current patch with patch index p, of the current tile with tile ID equal to tileID where i is in the range of 0 to (TilePatchEomPatchCount[tileID][p] – 1), inclusive.

TilePatchEomPoints[tileID][p][i] specifies the number of EOM coded points in the i-th patch associated with the current patch with patch index p, of the current tile with tile ID equal to tileID, where i is in the range of 0 to (TilePatchEomPatchCount[tileID][p] – 1), inclusive.

TilePatchPlrdLevel[tileID][p][m] specifies the level, i.e. whether it is at the block or patch level, where PLR data information is present for the m-th map associated with the current patch with patch index p, of the current tile with tile ID equal to tileID, where m is in the range if 0 to asps_map_count_minus1, inclusive.

TilePatchPlrdPresentBlockFlag[tileID][p][m][j] specifies the presence of TilePatchPlrdBlockModeMinus1[tileID][p][m][j] for the j-th block of the map with index m of the patch with patch index p, in the tile with tile ID equal to tileID.

TilePatchPlrdBlockModeMinus1[tileID][p][m][j] specifies the PLR mode selected in the list of PLR modes, for the j-th block of the map with index m of the patch with patch index p, in the tile with tile ID equal to tileID.

TilePatchPlrdPresentFlag[tileID][p][m] specifies the presence of the variable TilePatchPlrdModeMinus1[tileID][p][m] for the current patch with patch index p, of the current tile with tile ID equal to tileID, and the map with index m.

TilePatchPlrdModeMinus1[tileID][p][m] specifies the selected PLR mode index and its associated parameters from the parameters indicated in the ASPS, as in subclause 8.3.6.1.2, for the current patch with patch index p, in the current tile with tile ID equal to tileID, for a map with index m.

These variables are initially set as follows:

TilePatchType[tileID][p] = PROJECTED
TilePatchInAuxVideo[tileID][p] = 0
TilePatch2dPosX[tileID][p] = 0
TilePatch2dPosY[tileID][p] = 0
TilePatch2dSizeX[tileID][p] = 1
TilePatch2dSizeY[tileID][p] = 1
TilePatch3dOffsetU[tileID][p] = 0
TilePatch3dOffsetV[tileID][p] = 0
TilePatch3dOffsetD[tileID][p] = 0
TilePatch3dRangeD[tileID][p] = 0
TilePatchProjectionID[tileID][p] = 0
TilePatchOrientationIndex[tileID][p] = 0
TilePatchLoDScaleX[tileID][p] = 1
TilePatchLoDScaleY[tileID][p] = 1
TilePatchRawPoints[tileID][p] = 0
TilePatchEomPatchCount[tileID][p] = 0

If atdu_patch_mode[tileID][p] is equal to I_INTRA or P_INTRA, then the process for decoding intra coded patches in subclause 9.2.5.2 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu_patch_mode[tileID][p] is equal to P_SKIP, then the process for decoding skip coded patches in subclause 9.2.5.3 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu_patch_mode[tileID][p] is equal to P_MERGE, then the process for decoding merge coded patches in subclause 9.2.5.4 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu_patch_mode[tileID][p] is equal to P_INTER, then the process for decoding inter coded patches in subclause 9.2.5.5 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu_patch_mode[tileID][p] is equal to I_RAW or P_RAW, then the process for decoding RAW coded patches in subclause 9.2.5.6 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu_patch_mode[tileID][p] is equal to I_EOM or P_EOM, then the process for decoding EOM coded patches in subclause 9.2.5.7 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

It is a requirement of bitstream conformance that patches of EOM or RAW patch type do not intersect with any other patch.

For projected patches then the following applies:

— A variable oIdx is set to TilePatchOrientationIndex[tileID][p].

— Variables posX, posY, sizeX, sizeY, lodX, and lodY are assigned as follows:

```
posX = TilePatch2dPosX[ tileID ][ p ]
posY = TilePatch2dPosY[ tileID ][ p ]
sizeX = TilePatch2dSizeX[ tileID ][ p ]
sizeY = TilePatch2dSize[ tileID ][ p ]
lodX = TilePatchLoDScaleX[ tileID ][ p ]
lodY = TilePatchLoDScaleY[ tileID ][ p ]
```

— For each sample with coordinates (x, y) that belongs to the patch p, where (x, y) is relative to the tile origin of the tile with index tileID, the conversion to the 3D coordinate system is performed by first transforming (x, y) to a local patch coordinate pair (u, v), as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \boldsymbol{R}_o(\text{oIdx}) * \begin{bmatrix} \text{lodX} & 0 \\ 0 & \text{lodY} \end{bmatrix} * \begin{bmatrix} x - \text{posX} \\ y - \text{posY} \end{bmatrix} + \boldsymbol{R}_s(\text{oIdx}) * \begin{bmatrix} \text{lodX} & 0 \\ 0 & \text{lodY} \end{bmatrix} * \begin{bmatrix} \text{sizeX} - 1 \\ \text{sizeY} - 1 \end{bmatrix} \quad (47)$$

where $\mathbf{R}_o$ and $\mathbf{R}_s$ are specified in Table 11.

The final conversion from (u, v) to the 3D coordinate system depends on the application.

Table 11 — Transformation matrices according to an indicated patch orientation index, oIdx

oIdx	R_o(oIdx)	R_s(oIdx)	Description
0	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	No transformation
1	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	The x and y axes are swapped
2	$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$	The x axis is inverted first and then x and y axes are swapped
3	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	The x and y axes are inverted
4	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$	The y axis is inverted first and then x and y axes are swapped
5	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$	The x axis is inverted
6	$\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	The x and y axes are inverted first and then are swapped
7	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$	The y axis is inverted

9.2.5.2 Decoding process for patch data units coded in intra mode

Inputs to this process are the current patch index, p, and the current tile ID, tileID.

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

TilePatch2dPosX[tileID][p] = pdu_2d_pos_x[tileID][p] * PatchPackingBlockSize (48)

TilePatch2dPosY[tileID][p] = pdu_2d_pos_y[tileID][p] * PatchPackingBlockSize (49)

TilePatch3dOffsetU[tileID][p] = pdu_3d_offset_u[tileID][p] (50)

TilePatch3dOffsetV[tileID][p] = pdu_3d_offset_v[tileID][p] (51)

TilePatch3dOffsetD[tileID][p] = Pdu3dOffsetD[tileID][p] (52)

TilePatch3dRangeD[tileID][p] = Pdu3dRangeD[tileID][p] (53)

TilePatchProjectionID[tileID][p] = pdu_projection_id[tileID][p] (54)

TilePatchOrientationIndex[tileID][p] = pdu_orientation_index[tileID][p] (55)

TilePatchLoDScaleX[tileID][p] =
pdu_lod_enabled_flag[p] ? pdu_lod_scale_x_minus1[tileID][p] + 1 : 1 (56)

offsetY = ((pdu_lod_scale_x_minus1[tileID][p] > 0) ? 1 : 2)

TilePatchLoDScaleY[tileID][p] = pdu_lod_enabled_flag[tileID][p] ?
pdu_lod_scale_y_idc[tileID][p] + offsetY : 1 (57)

TilePatch2dSizeX[tileID][p] =
(pdu_2d_size_x_minus1[tileID][p] + 1) * PatchSizeXQuantizer (58)

TilePatch2dSizeY[tileID][p] =
(pdu_2d_size_y_minus1[tileID][p] + 1) * PatchSizeYQuantizer (59)

If asps_plr_enabled_flag is equal to 1, the following applies:

```
blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )

for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {
    if( plri_map_present_flag[ m ] == 1 ) {
        TilePatchPlrdLevel[ tileID ][ p ][ m ] = plrd_level[ tileID ][ p ][ m ]
        if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {
            for( j = 0; j < blockCnt; j++ ) {
                TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =
                    plrd_present_block_flag[ tileID ][ p ][ m ][ j ]
                    if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )
                    TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =
                        plrd_block_mode_minus1[ tileID ][ p ][ m ][ j ]
            }
        } else {
          TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = plrd_present_flag[ tileID ][ p ][ m ]
          if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )
                TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] =
                    plrd_mode_minus1[ tileID ][ p ][ m ]
        }
        }
    }
```

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

Inputs to this process are the current patch index, p, and current tile ID, tileID.

First, refIdx is set to 0.

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the (refIdx + 1)-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[refIdx].

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

The variable RefPatchIdx, which corresponds to the predictor patch index in the tile with ID equal to tileID, in the reference atlas frame, refAtlasFrm, is computed as:

RefPatchIdx = PredictorIdx (60)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause 9.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, and refPatchEomPatchCount, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

Initially, the following parameters are derived:

TilePatchType[tileID][p] = refPatchType (61)

TilePatchInAuxVideo[tileID][p] = refPatchInAuxVideo (62)

TilePatch2dPosX[tileID][p] = refPatch2dPosX (63)

TilePatch2dPosY[tileID][p] = refPatch2dPosY (64)

TilePatch2dSizeX[tileID][p] = refPatch2dSizeX (65)

TilePatch2dSizeY[tileID][p] = refPatch2dSizeY (66)

In addition, if refPatchType is equal to RAW, the associated 2D and 3D patch parameters are derived as follows:

TilePatch3dOffsetU[tileID][p] = refPatch3dOffsetU (67)

TilePatch3dOffsetV[tileID][p] = refPatch3dOffsetV (68)

TilePatch3dOffsetD[tileID][p] = refPatch3dOffsetD (69)

TilePatch3dRangeD[tileID][p] = refPatch3dRangeD (70)

TilePatchRawPoints[tileID][p] = refPatchRawPoints (71)

Otherwise, if refPatchType is equal to EOM, the following applies:

TilePatchEomPatchCount[tileID][p] = refPatchEomPatchCount (72)

```
for( i = 0; i < TilePatchEomPatchCount[ tileID ][ p ]; i++ ) {
    TilePatchAssociatedPatchIndex[ tileID ][ p ][ i ] = refPatchAssociatedPatchIndex[ i ]    (73)

    TilePatchEomPoints[ tileID ][ p ][ i ] = refPatchEomPoints[ i ]    (74)

}
```

Otherwise, the associated 2D and 3D patch parameters are derived as follows:

TilePatch3dOffsetU[tileID][p] = refPatch3dOffsetU (75)

TilePatch3dOffsetV[tileID][p] = refPatch3dOffsetV (76)

TilePatch3dOffsetD[tileID][p] = refPatch3dOffsetD (77)

TilePatch3dRangeD[tileID][p] = refPatch3dRangeD (78)

TilePatchProjectionID[tileID][p] = refPatchProjectionID (79)

TilePatchOrientationIndex[tileID][p] = refPatchOrientationIndex (80)

TilePatchLoDScaleX[tileID][p] = refPatchLoDScaleX (81)

TilePatchLoDScaleY[tileID][p] = refPatchLoDScaleY (82)

If asps_plr_enabled_flag is equal to 1, the PLR patch parameters are derived as follows:

```
blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )
for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {    (83)
    if( plri_map_present_flag[ m ] == 1 ) {
        TilePatchPlrdLevel[ tileID ][ p ][ m ] = refPatchPlrdLevel[ m ]
        if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {
            for( j = 0; j < blockCnt; j++ ) {
                TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =
                    refPatchPlrdPresentBlockFlag[ m ][ j ]
                if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )
                    TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =
                        refPatchPlrdBlockModeMinus1[ m ][ j ]
            }
        } else {
            TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = refPatchPlrdPresentFlag[ m ]
            if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )
                TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] = refPlrdModeMinus1[ m ]
        }
    }
}
```

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

Inputs to this process are the current patch index, p, and the current tile ID, tileID.

First, the reference atlas frame index, refIdx, is derived as mpdu_ref_index[tileID][p].

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the (refIdx + 1)-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[refIdx].

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

Then the predictor patch index, RefPatchIdx, in the tile with ID equal to tile ID, in the reference atlas frame, refAtlasFrm, is computed as:

RefPatchIdx = PredictorIdx (84)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause 9.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, and refPatchEomPatchCount, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

It is a requirement of bitstream conformance that refPatchType is equal to projected.

First, TilePatchType[tileID][p] is assigned as follows:

TilePatchType[tileID][p] = refPatchType (85)

If mpdu_override_2d_params_flag[tileID][p] is equal to 1, the associated 2D patch parameters are derived as follows:

TilePatch2dPosX[tileID][p] =
refPatch2dPosX + mpdu_2d_pos_x[tileID][p] * PatchPackingBlockSize (86)

TilePatch2dPosY[tileID][p] =
refPatch2dPosY + mpdu_2d_pos_y[tileID][p] * PatchPackingBlockSize (87)

TilePatch2dSizeX[tileID][p] =
refPatch2dSizeX + mpdu_2d_delta_size_x[tileID][p] * PatchSizeXQuantizer (88)

TilePatch2dSizeY[tileID][p] =
refPatch2dSizeY + mpdu_2d_delta_size_y[tileID][p] * PatchSizeYQuantizer (89)

Otherwise,

TilePatch2dPosX[tileID][p] = refPatch2dPosX (90)

TilePatch2dPosY[tileID][p] = refPatch2dPosY (91)

TilePatch2dSizeX[tileID][p] = refPatch2dSizeX (92)

TilePatch2dSizeY[tileID][p] = refPatch2dSizeY (93)

If mpdu_override_3d_params_flag[tileID][p] is equal to 1, then the associated 3D patch parameters are derived as follows:

TilePatch3dOffsetU[tileID][p] = refPatch3dOffsetU + mpdu_3d_offset_u[tileID][p] (94)

TilePatch3dOffsetV[tileID][p] = refPatch3dOffsetV + mpdu_3d_offset_v[tileID][p] (95)

TilePatch3dOffsetD[tileID][p] = refPatch3dOffsetD + Mpdu3dOffsetD[tileID][p] (96)

TilePatch3dRangeD[tileID][p] = refPatch3dRangeD + Mpdu3dRangeD[tileID][p] (97)

Otherwise,

TilePatch3dOffsetU[tileID][p] = refPatch3dOffsetU (98)

TilePatch3dOffsetV[tileID][p] = refPatch3dOffsetV (99)

TilePatch3dOffsetD[tileID][p] = refPatch3dOffsetD (100)

TilePatch3dRangeD[tileID][p] = refPatch3dRangeD (101)

The following variables are set:

TilePatchProjectionID[tileID][p] = refPatchProjectionID (102)

TilePatchOrientationIndex[tileID][p] = refPatchOrientationIndex (103)

TilePatchLoDScaleX[tileID][p] = refPatchLoDScaleX (104)

TilePatchLoDScaleY[tileID][p] = refPatchLoDScaleY (105)

If asps_plr_enabled_flag is equal to 1, the following applies:

blockCnt = BlockCnt(TilePatch2dSizeX[tileID][p], TilePatch2dSizeY[tileID][p])

If OverridePlrFlag is equal to 1, the associated PLR parameters are derived as follows:

```
for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {
     if( plri_map_present_flag[ m ] == 1 ) {
          TilePatchPlrdLevel[ tileID ][ p ][ m ] = plrd_level[ tileID ][ p ][ m ]
          if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {
               for( j = 0; j < blockCnt; j++ ) {
                    TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =
                         plrd_present_block_flag[ tileID ][ p ][ m ][ j ]
                    if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )
                         TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =
                              plrd_block_mode_minus1[ tileID ][ p ][ m ][ j ]
               }
          } else {
               TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = plrd_present_flag[ tileID ][ p ][ m ]
               if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )
```

```
                TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] = plrd_mode_minus1[ tileID ][ p ][ m ]
            }
        }
    }
```

Otherwise (OverridePlrFlag is equal to 0),

```
    for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {
        if( plri_map_present_flag[ m ] == 1 ) {
            TilePatchPlrdLevel[ tileID ][ p ][ m ] = refPatchPlrdLevel[ m ]
            if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {
            for( j = 0; j < blockCnt; j++ ) {
                TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =
                    refPatchPlrdPresentBlockFlag[ m ][ j ]
                if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )
                    TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =
                        refPatchPlrdBlockModeMinus1[ m ][ j ]
            }
        } else {
            TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = refPatchPlrdPresentFlag[ m ]
            if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )
                TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] = refPlrdModeMinus1[ m ]
        }
    }
```

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

9.2.5.5.1 General decoding process for patch data units coded in inter prediction mode

Inputs to this process are the current patch index, p, and the current tile ID, tileID.

First, the reference atlas frame index, refIdx, is derived as ipdu_ref_index[tileID][p].

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the (refIdx + 1)-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[refIdx].

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

Then the corresponding patch index, RefPatchIdx, in the tile with ID equal to tile ID, in the reference atlas frame, refAtlasFrm, is computed as:

$$\text{RefPatchIdx} = \text{PredictorIdx} + \text{ipdu\_patch\_index}[\ \text{tileID}\ ][\ \text{p}\ ] \quad (106)$$

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause 9.2.5.5.2 is invoked using the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, and refPatchEomPatchCount, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

It is a requirement of bitstream conformance that refPatchType is equal to projected.

Then, the associated 2D and 3D patch parameters are derived as follows:

$$\text{TilePatchType}[\ \text{tileID}\ ][\ \text{p}\ ] = \text{refPatchType} \quad (107)$$

```
TilePatch2dPosX[ tileID ][ p ] =
        refPatch2dPosX + ipdu_2d_pos_x[ tileID ][ p ] * PatchPackingBlockSize                (108)

TilePatch2dPosY[ tileID ][ p ] =
        refPatch2dPosY + ipdu_2d_pos_y[ tileID ][ p ] * PatchPackingBlockSize                (109)

TilePatch2dSizeX[ tileID ][ p ] =
        refPatch2dSizeX + ipdu_2d_delta_size_x[ tileID ][ p ] * PatchSizeXQuantizer          (110)

TilePatch2dSizeY[ tileID ][ p ] =
        refPatch2dSizeY + ipdu_2d_delta_size_y[ tileID ][ p ] * PatchSizeYQuantizer          (111)

TilePatch3dOffsetU[ tileID ][ p ] = refPatch3dOffsetU + ipdu_3d_offset_u[ tileID ][ p ]      (112)

TilePatch3dOffsetV[ tileID ][ p ] = refPatch3dOffsetV + ipdu_3d_offset_v[ tileID ][ p ]      (113)

TilePatch3dOffsetD[ tileID ][ p ] = refPatch3dOffsetD + Ipdu3dOffsetD[ tileID ][ p ]         (114)

TilePatch3dRangeD[ tileID ][ p ] = refPatch3dRangeD + Ipdu3dRangeD[ tileID ][ p ]            (115)

TilePatchProjectionID[ tileID ][ p ] = refPatchProjectionID                                  (116)

TilePatchOrientationIndex[ tileID ][ p ] = refPatchOrientationIndex                          (117)

TilePatchLoDScaleX[ tileID ][ p ] = refPatchLoDScaleX                                        (118)

TilePatchLoDScaleY[ tileID ][ p ] = refPatchLoDScaleY                                        (119)
```

If asps_plr_enabled_flag is equal to 1, the following applies:

```
blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )
for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {     (120)
    if( plri_map_present_flag[ m ] == 1 ) {
        TilePatchPlrdLevel[ tileID ][ p ][ m ] = plrd_level[ tileID ][ p ][ m ]
        if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {
            for( j = 0; j < blockCnt; j++ ) {
                TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =
                    plrd_present_block_flag[ tileID ][ p ][ m ][ j ]
                if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )
                    TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =
                        plrd_block_mode_minus1[ tileID ][ p ][ m ][ j ]
            }
        } else {
            TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = plrd_present_flag[ tileID ][ p ][ m ]
            if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )
              TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] = plrd_mode_minus1[ tileID ][ p ][ m ]
        }
    }
}
```

9.2.5.5.2 Derivation of inter reference patch parameters

9.2.5.5.2.1 General derivation of inter reference patch parameters

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

Outputs to this process are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, and refPatchEomPatchCount, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

First, the patch, refPatch, is determined that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the (refIdx + 1)-th entry of the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[refIdx].

Then, the outputs of this process are derived based on the associated parameters of the patch, refPatch as follows:

refPatchType = TilePatchType[tileID][refPatchIdx] (121)

refPatchInAuxVideo = TilePatchInAuxVideo[tileID][refPatchIdx] (122)

refPatch2dPosX = TilePatch2dPosX[tileID][refPatchIdx] (123)

refPatch2dPosY = TilePatch2dPosY[tileID][refPatchIdx] (124)

refPatch2dSizeX = TilePatch2dSizeX[tileID][refPatchIdx] (125)

refPatch2dSizeY = TilePatch2dSizeY[tileID][refPatchIdx] (126)

If refPatchType is equal to projected, then the process in subclause 9.2.5.5.2.2 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatchProjectionID, refPatch3dOffsetD, refPatch3dRangeD, refPatchOrientationIndex, refPatchLoDScaleX, and refPatchLoDScaleY, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

If refPatchType is equal to RAW, then the process in subclause 9.2.5.5.2.3 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, and refPatchRawPoints.

If refPatchType is equal to EOM, then the process in subclause 9.2.5.5.2.4 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variable refPatchEomPatchCount, and the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints.

9.2.5.5.2.2 Reference patches in projected mode

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

Outputs to this process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, and refPachLoDScaleY, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

First, the patch, refPatch, that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList is determined.

Then, the outputs of this process are derived as follows:

refPatch3dOffsetU = TilePatch3dOffsetU[tileID][refPatchIdx] (127)

refPatch3dOffsetV = TilePatch3dOffsetV[tileID][refPatchIdx] (128)

refPatch3dOffsetD = TilePatch3dOffsetD[tileID][refPatchIdx] (129)

refPatch3dRangeD = TilePatch3dRangeD[tileID][refPatchIdx] (130)

refPatchProjectionID = TilePatchProjectionID[tileID][refPatchIdx] (131)

refPatchOrientationIndex = TilePatchOrientationIndex[tileID][refPatchIdx] (132)

refPatchLoDScaleX = TilePatchLoDScaleX[tileID][refPatchIdx] (133)

refPatchLoDScaleY = TilePatchLoDScaleY[tileID][refPatchIdx] (134)

If asps_plr_enabled_flag is equal to 1, the following applies:

```
blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ refPatchIdx ],
                TilePatch2dSizeY[ tileID ][ refPatchIdx ] )
for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {
    if( plri_map_present_flag[ m ] == 1 ) {
        refPatchPlrdLevel[ m ] = TilePatchPlrdLevel[ tileID ][ refPatchIdx ][ m ]
        if( refPatchPlrdLevel[ m ] == 0 ) {
            for( j = 0; j < blockCnt; j++ ) {
                refPatchPlrdPresentBlockFlag[ m ][ j ] =
                    TilePatchPlrdPresentBlockFlag[ tileID ][ refPatchIdx ][ m ][ j ]
                        if( refPatchPlrdPresentBlockFlag[ m ][ j ] == 1 )
                            refPatchPlrdBlockModeMinus1[ m ][ j ] =
                                TilePatchPlrdBlockModeMinus1[ tileID ][ refPatchIdx ][ m ][ j ]
            }
        } else {
            refPatchPlrdPresentFlag[ m ] = TilePatchPlrdPresentFlag[ tileID ][ refPatchIdx ][ m ]
            if( refPatchPlrdPresentFlag[ m ] == 1 )
                refPlrdModeMinus1[ m ] = TilePatchPlrdModeMinus1[ tileID ][ refPatchIdx ][ m ]
        }
    }
}
```

where TilePatch3dOffsetU, TilePatch3dOffsetV, TilePatch3dOffsetD, TilePatch3dRangeD, TilePatchProjectionID TilePatchOrientationIndex, TilePatchLoDScaleX, TilePatchLoDScaleY, TilePatch2dSizeX, TilePatch2dSizeY, TilePatchPlrdLevel, TilePatchPlrdPresentBlockFlag, TilePatchPlrdBlockModeMinus1, TilePatchPlrdPresentFlag, and TilePatchPlrdModeMinus1 are the associated tile patch parameters from the patch with index refPatchIdx in the tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList.

9.2.5.5.2.3 Reference patches in RAW mode

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

Outputs to this process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, and refPatchRawPoints.

First, the patch, refPatch, that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList is determined.

Then, the outputs of this process are derived as follows:

```
refPatch3dOffsetU = TilePatch3dOffsetU[ tileID ][ refPatchIdx ]                (135)

refPatch3dOffsetV = TilePatch3dOffsetV[ tileID ][ refPatchIdx ]                (136)

refPatch3dOffsetD = TilePatch3dOffsetD[ tileID ][ refPatchIdx ]                (137)

refPatchRawPoints = TilePatchRawPoints[ tileID ][ refPatchIdx ]                (138)
```

where TilePatch3dOffsetU, TilePatch3dOffsetV, TilePatch3dOffsetD, and TilePatchRawPoints are the associated tile patch parameters from the patch with index refPatchIdx in the tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList.

9.2.5.5.2.4 Reference patches in EOM mode

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

Outputs to this process are the variable refPatchEomPatchCount and the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints.

First, the patch, refPatch, that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the (refIdx + 1)-th entry of the reference atlas frame list struct, RefAtlasFrmList is determined.

Then, the outputs of this process are derived as follows:

```
refPatchEomPatchCount = TilePatchEomPatchCount[ tileID ][ refPatchIdx ]        (139)
for( i = 0; i < refPatchEomPatchCount; i++ ) {

    refPatchAssociatedPatchIndex[ i ]
        = TilePatchAssociatedPatchIndex[ tileID ][ refPatchIdx ][ i ]          (140)

    refPatchEomPoints[ i ] = TilePatchEomPoints[ tileID ][ refPatchIdx ][ i ]  (141)
}
```

where TilePatchEomPatchCount, TilePatchAssociatedPatchIndex, and TilePatchEomPoints are the associated tile patch parameters from the patch with index refPatchIdx in the tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList.

9.2.5.6 Decoding process for patch data units coded in RAW mode

Inputs to this process are the current patch index, p, and the current tile ID, tileID.

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

TilePatchType[tileID][p] = RAW (142)

TilePatchInAuxVideo[tileID][p] = rpdu_patch_in_auxiliary_video_flag[tileID][p] (143)

TilePatch2dPosX[tileID][p] = rpdu_2d_pos_x[tileID][p] * PatchPackingBlockSize (144)

TilePatch2dPosY[tileID][p] = rpdu_2d_pos_y[tileID][p] * PatchPackingBlockSize (145)

Then the variables TilePatch2dSizeX[tileID][p], TilePatch2dSizeY[tileID][p], and TilePatchRawPoints[tileID][p] are derived as follows:

TilePatch2dSizeX[tileID][p] =
(rpdu_2d_size_x_minus1[tileID][p] + 1) * PatchSizeXQuantizer (146)

TilePatch2dSizeY[tileID][p] =
(rpdu_2d_size_y_minus1[tileID][p] + 1) * PatchSizeYQuantizer (147)

TilePatchRawPoints[tileID][p] = rpdu_points_minus1[tileID][p] + 1 (148)

Then, the 3D patch variables TilePatch3dOffsetU[tileID][p], TilePatch3dOffsetV[tileID][p], and TilePatch3dOffsetD[tileID][p] are derived and set as output of this process as follows:

TilePatchRawPoints[tileID][refPatchIdx] = rpdu_points[tileID][p] (149)

TilePatch3dOffsetU[tileID][p] = rpdu_3d_offset_u[tileID][p] << RawShift (150)

TilePatch3dOffsetV[tileID][p] = rpdu_3d_offset_v[tileID][p] << RawShift (151)

TilePatch3dOffsetD[tileID][p] = rpdu_3d_offset_d[tileID][p] << RawShift (152)

9.2.5.7 Decoding process for patch data units coded in EOM mode

Inputs to this process are the current patch index, p, and the current tile ID, tileID.

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

TilePatchType[tileID][p] = EOM (153)

TilePatchInAuxVideo[tileID][p] = epdu_patch_in_auxiliary_video_flag[tileID][p] (154)

TilePatch2dPosX[tileID][p] = epdu_2d_pos_x[tileID][p] * PatchPackingBlockSize (155)

TilePatch2dPosY[tileID][p] = epdu_2d_pos_y[tileID][p] * PatchPackingBlockSize (156)

TilePatch2dSizeX[tileID][p] =
(epdu_2d_size_x_minus1[tileID][p] + 1) * PatchSizeXQuantizer (157)

TilePatch2dSizeY[tileID][p] =
(epdu_2d_size_y_minus1[tileID][p] + 1) * PatchSizeYQuantizer (158)

```
TilePatchEomPatchCount[ tileID ][ p ] = ( epdu_patch_count_minus1[ tileID ][ p ] + 1 )          (159)

for( i = 0; i < TilePatchEomPatchCount[ tileID ][ p ]; i++ ) {

TilePatchAssociatedPatchIndex[ tileID ][ p ][ i ] =
        epdu_associated_patch_idx[ tileID ][ p ]                                                 (160)

TilePatchEomPoints[ tileID ][ p ][ i ] = epdu_points[ tileID ][ p ]                              (161)

}
```

9.2.6 Decoding process of the block to patch map

Input to this process is the current tile ID, tileID.

Then the variables TileBlockToPatchMapWidth[tileID] and TileBlockToPatchMapHeight[tileID], and the two-dimensional array TileBlockToPatchMap[tileID] are computed as follows:

```
offset = PatchPackingBlockSize – 1

TileBlockToPatchMapWidth[ tileID ] =
        ( TileWidth[ TileIDToIndex[ tileID ] ] + offset ) / PatchPackingBlockSize                (162)

TileBlockToPatchMapHeight[ tileID ]=
        ( TileHeight[ TileIDToIndex[ tileID ] ] + offset ) / PatchPackingBlockSize               (163)
```

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

```
for( y = 0; y < TileBlockToPatchMapHeight[ tileID ]; y++ )
    for( x = 0; x < TileBlockToPatchMapWidth[ tileID ]; x++ )
        TileBlockToPatchMap[ tileID ][ y ][ x ] = −1
```

Then the TileBlockToPatchMap array is updated as follows:

```
for( p = 0; p < AtduTotalNumPatches[ tileID ]; p++ ) {
    patchType = TilePatchType[ tileID ][ p ]
    if( patchType == PROJECTED ) {
            xOrg = TilePatch2dPosX[ tileID ][ p ] / PatchPackingBlockSize
            yOrg = TilePatch2dPosY[ tileID ][ p ] / PatchPackingBlockSize
            tilePatchWidthBlk = ( TilePatch2dSizeX[ tileID ][ p ] + offset ) / PatchPackingBlockSize
            tilePatchHeightBlk = ( TilePatch2dSizeY[ tileID ][ p ] + offset ) / PatchPackingBlockSize
            for( y = 0; y < tilePatchHeightBlk ; y++)
                for( x = 0; x < tilePatchWidthBlk; x++ ) {
                    if(( asps_patch_precedence_order_flag == 0 ) ||
                        ( TileBlockToPatchMap[ tileID ][ yOrg + y ][ xOrg + x ] == −1 ) )
                            TileBlockToPatchMap[ tileID ][ yOrg + y ][ xOrg + x ] = p
            }
        }
    }
    else if(TilePatchInAuxVideo[ tileID ][ p ] == 0 ) {
        xOrg = TilePatch2dPosX[ tileID ][ p ] / PatchPackingBlockSize
        yOrg = TilePatch2dPosY[ tileID ][ p ] / PatchPackingBlockSize
        tilePatchWidthBlk = ( TilePatch2dSizeX[ tileID ][ p ] + offset ) / PatchPackingBlockSize
        tilePatchHeightBlk = ( TilePatch2dSizeY[ tileID ][ p ] + offset ) / PatchPackingBlockSize
        for( y = 0; y < tilePatchHeightBlk ; y++)
            for( x = 0; x < tilePatchWidthBlk; x++ )
                TileBlockToPatchMap[ tileID ][ yOrg + y ][ xOrg + x ] = p
        }
    }
```

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

9.2.7.1 General

In some implementations it is possible that the reconstruction process may be performed using tile information directly, i.e. the reconstruction process is tile based. In other implementations the reconstruction process may be performed using all tile information, i.e. is atlas based. In that particular case, a conversion of tile level patch information to atlas level patch information may be necessary. This conversion is performed in this subclause.

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

In this subclause, the block to patch maps from each tile are combined into a single two-dimensional array, AtlasBlockToPatchMap. This array and the variables AtlasBlockToPatchMapWidth, and AtlasBlockToPatchMapHeight are computed as follows:

offset = PatchPackingBlockSize − 1

AtlasBlockToPatchMapWidth = (asps_frame_width + offset) / PatchPackingBlockSize (164)

AtlasBlockToPatchMapHeight = (asps_frame_height + offset) / PatchPackingBlockSize (165)

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

```
for( y = 0; y < AtlasBlockToPatchMapHeight; y++ )
    for( x = 0; x < AtlasBlockToPatchMapWidth; x++ )
        AtlasBlockToPatchMap[ y ][ x ] = −1
```

Then the AtlasBlockToPatchMap array is updated as follows:

```
offsetPatch = 0
for( t = 0; t <= afti_num_tiles_in_atlas_frame_minus1; t++ ) {
    tileID = TileIndexToID[ t ]
    tileOffsetX = TileOffsetX[ t ]
    tileOffsetY = TileOffsetY[ t ]
    for( blkY = 0; blkY < TileBlockToPatchMapHeight[ tileID ]; blkY++ ) {
        for( blkX = 0; blkX < TileBlockToPatchMapWidth[ tileID ]; blkX++ ) {
            y = tileOffsetY / PatchPackingBlockSize + blkY
            x = tileOffsetX / PatchPackingBlockSize + blkX
            offset = ( TileBlockToPatchMap[ tileID ][ blkY ][ blkX ] == −1 ) ? 0 : offsetPatch
            AtlasBlockToPatchMap[ y ][ x ] = TileBlockToPatchMap[ tileID ][ blkY ][ blkX ] + offset
        }
    }
    offsetPatch += AtduTotalNumPatches[ tileID]
}
AtlasTotalNumPatches = offsetPatch
```

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

9.2.7.3.1 General

In this subclause tile level patch information is converted to atlas level patch information, using also the processes CommonTilePatchParamsToAtlas(tileID, p) and ApplicationTilePatchParamsToAtlas(tileID, p) defined in subclauses 9.2.7.3.2 and 9.2.7.3.3, respectively, as follows:

```
NumRawPoints = 0
NumEomPoints = 0
AtlasTotalNumPatches = 0
AtlasTotalNumProjPatches = 0
AtlasTotalNumEomPatches = 0
AtlasTotalNumRawPatches = 0
atlasPatchIdx = 0
offsetPatch = 0

for( t = 0; t <= afti_num_tiles_in_atlas_frame_minus1; t++ ) {
    tileID = TileIndexToID[ t ]
    tileOffsetX = TilePatchInAuxVideo[ tileID ][ p ] ? 0 : TileOffsetX[ t ]
    tileOffsetY = TilePatchInAuxVideo[ tileID ][ p ] ? AuxTileOffset[ t ] : TileOffsetY[ t ]
    for( p = 0; p < AtduTotalNumPatches[ tileID ]; p++ ) {
        CommonTilePatchParamsToAtlas( atlasPatchIdx, tileID, p)
        ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch )
        atlasPatchIdx += 1
    }
    offsetPatch += AtduTotalNumPatches[ tileID]
}
AtlasTotalNumPatches = atlasPatchIdx
```

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

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

```
CommonTilePatchParamsToAtlas( atlasPatchIdx, tileID, p ) {
    AtlasPatchInAuxVideo[ atlasPatchIdx ] = TilePatchInAuxVideo[ tileID ][ p ]
    AtlasPatchType[ atlasPatchIdx ] = TilePatchType[ tileID ][ p ]
    AtlasPatch2dSizeX[ atlasPatchIdx ] = TilePatch2dSizeX[ tileID ][ p ]
    AtlasPatch2dSizeY[ atlasPatchIdx ] = TilePatch2dSizeY[ tileID ][ p ]
    AtlasPatch2dPosX[ atlasPatchIdx ] = TilePatch2dPosX[ tileID ][ p ] + tileOffsetX
    AtlasPatch2dPosY[ atlasPatchIdx ] = TilePatch2dPosY[ tileID ][ p ] + tileOffsetY
    AtlasPatch3dOffsetU[ atlasPatchIdx ] = TilePatch3dOffsetU[ tileID ][ p ]
    AtlasPatch3dOffsetV[ atlasPatchIdx ] = TilePatch3dOffsetV[ tileID ][ p ]
    AtlasPatch3dOffsetD[ atlasPatchIdx ] = TilePatch3dOffsetD[ tileID ][ p ]
    AtlasPatch3dRangeD[ atlasPatchIdx ] = TilePatch3dRangeD[ tileID ][ p ]
    AtlasPatchProjectionID[ atlasPatchIdx ] = TilePatchProjectionID[ tileID ][ p ]
    AtlasPatchOrientationIndex[ atlasPatchIdx ] = TilePatchOrientationIndex[ tileID ][ p ]
    AtlasPatchLoDScaleX[ atlasPatchIdx ] = TilePatchLoDScaleX[ tileID ][ p ]
    AtlasPatchLoDScaleY[ atlasPatchIdx ] = TilePatchLoDScaleY[ tileID ][ p ]
    AtlasPatchRawPoints[ atlasPatchIdx ] = TilePatchRawPoints[ tileID ][ p ]
    AtlasPatchEomPatchCount[ atlasPatchIdx ] = TilePatchEomPatchCount[ tileID ][ p ]
}
```

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

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

```
ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch ) {
    if( AtlasPatchType[ atlasPatchIdx ] == PROJECTED ) {
        AtlasTotalNumProjPatches += 1
    blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )
    for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {
            if( plri_map_present_flag[ m ] == 1 ) {
                AtlasPlrdLevel[ atlasPatchIdx ][ m ] = TilePatchPlrdLevel[ tileID ][ p ][ m ]
                if( AtlasPlrdLevel[ atlasPatchIdx ][ m ] == 0 ) {
                    for( j = 0; j < blockCnt; j++ ) {
                        AtlasPlrdPresentBlockFlag[ atlasPatchIdx ][ m ][ j ] =
                            TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ]
                        if( AtlasPlrdPresentBlockFlag[ atlasPatchIdx ][ m ][ j ] == 1 )
                            AtlasPlrdBlockModeMinus1[ atlasPatchIdx ][ m ][ j ] =
                                    TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ]
                    }
                } else {
                    AtlasPlrdPresentFlag[ atlasPatchIdx ][ m ] =
                        TilePatchPlrdPresentFlag[ tileID ][ p ][ m ]
                    if( AtlasPlrdPresentFlag[ atlasPatchIdx ][ m ] == 1 )
                        AtlasPlrdModeMinus1[ atlasPatchIdx ][ m ] =
                            TilePatchPlrdModeMinus1[ tileID ][ p ][ m ]
                }
            }
        }
    }
    if( AtlasPatchType[ atlasPatchIdx ] == EOM ) {
        AtlasTotalNumEomPatches += 1
        for( i = 0; i < AtlasPatchEomPatchCount[ atlasPatchIdx ]; i++ ) {
            NumEomPoints += TilePatchEomPoints[ tileID ][ p ][ i ]
            AtlasPatchEomPoints[ atlasPatchIdx ][ i ] = TilePatchEomPoints[ tileID ][ p ][ i ]
            AtlasPatchAssociatedPatchIndex[ atlasPatchIdx ][ i ] =
                TilePatchAssociatedPatchIndex[ tileID ][ p ][ i ] + offsetPatch
        }
    }
    if( AtlasPatchType[ atlasPatchIdx ] == RAW ) {
        AtlasTotalNumRawPatches += 1
        NumRawPoints += TilePatchRawPoints[ tileID ][ p ]
    }
}
```

9.3 Occupancy video decoding process

The decoding process of an occupancy video component associated with the atlas, with atlas ID DecAtlasID, is performed as follows.

For the occupancy video component, the codec is first determined using either the profiles defined in Annex A or the value of oi_occupancy_codec_id[DecAtlasID] and a component codec mapping SEI message specified in subclause F.2.11, if present. Then, the occupancy video decoding process, according to the corresponding coding specification, is invoked using the occupancy video sub-bitstreams present in the V3C bitstream as the input.

Outputs of this process are:

— NumDecOccFrames, indicating the number of decoded occupancy video frames,

— a 4D array DecOccFrames, the decoded occupancy video frames, where the dimensions correspond to the decoded occupancy video frame index, the component index, the row index, and the column index, respectively, and

— the following 1D arrays:

  — DecOccBitDepth, indicating the occupancy video bit depth,

  — DecOccHeight, indicating the occupancy video height,

  — DecOccWidth, indicating the occupancy video width ,

  — DecOccChromaFormat, indicating the attribute chroma format,

  — DecOccChromaSamplingPosition, indicating, if present, the video chroma sampling position as specified in ISO/IEC 23091-2,

  — DecOccFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,

  — DecOccColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,

  — DecOccTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,

  — DecOccMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,

  — DecOccOutOrdIdx, indicating the occupancy video output order index, and

  — DecOccCompTime, indicating the occupancy video composition time.

where the dimension corresponds to the decoded occupancy video frame index.

If the array DecOccFullRange is missing, all of its elements shall be set to 1.

If any of the elements of the array DecOccTransferCharacteristics are missing or are set to a value of 2, i.e. unspecified, those elements shall be set to 8, i.e. linear.

If the array DecOccChromaSamplingPosition is missing, all of its elements shall be set to 0.

If the array DecOccColourPrimaries is missing, all of its elements shall be set to 2.

If the array DecOccMatrixCoeffs is missing, all of its elements shall be set to 2.

The values of the arrays DecOccChromaSamplingPosition, DecOccColourPrimaries, and DecOccMatrixCoeffs, shall not be used for any further processing of the decoded occupancy frames.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

NOTE Any existing video coding specification such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video coding specification can be used if included in oi_occupancy_codec_id.

9.4 Geometry video decoding process

The decoding process of a geometry video component associated with the atlas, with atlas ID DecAtlasID, is performed as follows.

For a geometry video component associated with syntax elements vuh_map_index, and vuh_auxiliary_video_flag, the codec is first determined using either the profiles defined in Annex A or the value of gi_geometry_codec_id[DecAtlasID] and gi_geometry_auxiliary_codec_id[DecAtlasID], if vuh_auxiliary_video_flag is equal to 0 or 1, respectively, and the component codec mapping SEI specified in subclause F.2.11, if present. Then, the geometry video decoding process, according to the corresponding coding specification, is invoked using the geometry video sub-bitstreams present in the V3C bitstream as the input.

If the geometry video sub-bitstream is not an auxiliary geometry video, either identified by vuh_auxiliary_video_flag equal to 0 or determined through external means if the V3C unit header is unavailable, the outputs of this process are:

— the 1D array NumDecGeoFrames, indicating the number of decoded geometry video frames, where the dimension corresponds to vuh_map_index ,

— the 5D array DecGeoFrames, the decoded geometry video frames, where the dimensions correspond to vuh_map_index, the decoded geometry video frame index, the component index, the row index, and the column index, respectively,

— the following 2D arrays:

  — DecGeoBitDepth, indicating the geometry video bit depth,

  — DecGeoHeight, indicating the geometry video height,

  — DecGeoWidth, indicating the geometry video width ,

  — DecGeoChromaFormat, indicating the attribute chroma format,

  — DecGeoChromaSamplingPosition, indicating, if present, the geometry chroma sampling as specified in ISO/IEC 23091-2,

  — DecGeoFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,

  — DecGeoColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,

  — DecGeoTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,

  — DecGeoMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,

  — DecGeoOutOrdIdx, indicating the geometry video output order index, and

  — DecGeoCompTime, indicating the geometry video composition time,

where the dimensions correspond to vuh_map_index and the decoded geometry video frame index, respectively.

If the array DecGeoFullRange is missing, all of its elements shall be set to 1.

If any of the elements of the array DecGeoTransferCharacteristics are missing or are set to a value of 2, i.e. unspecified, those elements shall be set to 8, i.e. linear.

If the array DecGeoChromaSamplingPosition is missing, all of its elements shall be set to 0.

If the array DecGeoColourPrimaries is missing, all of its elements shall be set to 2.

If the array DecGeoMatrixCoeffs is missing, all of its elements shall be set to 2.

The values of the arrays DecGeoChromaSamplingPosition, DecGeoColourPrimaries, and DecGeoMatrixCoeffs, shall not be used for any further processing of the decoded geometry frames.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

Otherwise, if the geometry video sub-bitstream is an auxiliary geometry video either identified by vuh_auxiliary_video_flag equal to 1 or determined through external means if the V3C unit header is unavailable, the outputs of this process are:

— NumDecGeoAuxFrames, indicating the number of decoded auxiliary geometry video frames,

— the 4D array DecGeoAuxFrames, the decoded auxiliary geometry video frames, where the dimensions correspond to the decoded auxiliary geometry video frame index, the component index, the row index, and the column index, respectively, and

— the following 1D arrays:

  — DecGeoAuxBitDepth, indicating the auxiliary geometry video bit depth,

  — DecGeoAuxHeight, indicating the auxiliary geometry video height,

  — DecGeoAuxWidth, indicating the auxiliary geometry video width ,

  — DecGeoAuxChromaFormat, indicating the attribute chroma format,

  — DecGeoAuxChromaSamplingPosition, indicating, if present, the geometry chroma sampling as specified in ISO/IEC 23091-2,

  — DecGeoAuxFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,

  — DecGeoAuxColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,

  — DecGeoAuxTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,

  — DecGeoAuxMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,

  — DecGeoAuxOutOrdIdx, indicating the auxiliary geometry video output order index, and

  — DecGeoAuxCompTime, indicating the auxiliary geometry video composition time,

where the dimension corresponds to the decoded auxiliary geometry video frame index.

If the array DecGeoAuxFullRange is missing, all of its elements shall be set to 1.

If any of the elements of the array DecGeoAuxTransferCharacteristics are missing or are set to a value of 2, i.e. unspecified, those elements shall be set to 8, i.e. linear.

If the array DecGeoAuxChromaSamplingPosition is missing, all of its elements shall be set to 0.

If the array DecGeoAuxColourPrimaries is missing, all of its elements shall be set to 2.

If the array DecGeoAuxMatrixCoeffs is missing, all of its elements shall be set to 2.

The values of the arrays DecGeoAuxChromaSamplingPosition, DecGeoAuxColourPrimaries, and DecGeoAuxMatrixCoeffs, shall not be used for any further processing of the decoded auxiliary geometry frames.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

NOTE Any existing video coding specification such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video coding specification can be used if included in gi_geometry_codec_id.

9.5 Attribute video decoding process

The decoding process of an attribute video component associated with the atlas with atlas ID DecAtlasID, is performed as follows:

— If ai_attribute_count[DecAtlasID] is equal to 0, no attribute video frames are decoded and no attribute information is associated with the final, reconstructed volumetric frame.

— Otherwise (if ai_attribute_count[DecAtlasID] is not equal to 0), and for each attribute with index attrIdx, for the attribute video component associated with the syntax elements vuh_map_index, vuh_attribute_index equal to attrIdx, vuh_attribute_partition_index, and vuh_auxiliary_video_flag, the codec is first determined using either the profiles defined in Annex A or the value of ai_attribute_codec_id[DecAtlasID][attrIdx] or ai_attribute_auxiliary_codec_id[DecAtlasID][attrIdx], if vuh_auxiliary_video_flag is equal to 0 or 1, respectively and the component codec mapping SEI specified in subclause F.2.11, if present. Then, the attribute video decoding process, according to the corresponding coding specification, is invoked using the attribute video sub-bitstreams present in the V3C bitstream as the input.

If the attribute video sub-bitstream is not an auxiliary attribute video, either identified by vuh_auxiliary_video_flag equal to 0 or determined through external means if the V3C unit header is unavailable, the outputs of this process are:

— the 1D array DecAttrType, indicating the attribute type, where the dimension corresponds to the attribute index,

— the 3D array NumDecAttrFrames, indicating the number of decoded attribute video frames, where the dimensions correspond to the attribute index, the attribute partition index, and vuh_map_index,

— the 7D array DecAttrFrames, the decoded attribute video frames, where the dimensions correspond to the attribute index, the attribute partition index, vuh_map_index, the decoded attribute video frame index, the component index, the row index, and the column index, respectively, and

— the following 4D arrays:

  — DecAttrBitDepth, indicating the attribute bit depth,

  — DecAttrHeight, indicating the attribute height,

  — DecAttrWidth, indicating the attribute width,

  — DecAttrNumComp, indicating the attribute number of components,

  — DecAttrChromaFormat, indicating the attribute chroma format,

  — DecAttrChromaSamplingPosition, indicating, if present, the attribute chroma sampling as specified in ISO/IEC 23091-2,

  — DecAttrFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,

  — DecAttrColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,

  — DecAttrTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,

  — DecAttrMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,

  — DecAttrOutOrdIdx, indicating the attribute output order index, and

  — DecAttrCompTime, indicating the attribute composition time,

where the dimensions correspond to the attribute index, the attribute partition index, vuh_map_index, and the decoded attribute frame index, respectively.

If the array DecAttrChromaSamplingPosition, is missing, all of its elements shall be set to 0.

If the array DecAttrFullRange, is missing, all of its elements shall be set to 1.

If the array DecAttrColourPrimaries, is missing, all of its elements shall be set to 1.

If the array DecAttrTransferCharacteristics, is missing, all of its elements shall be set to 1.

If the array DecAttrMatrixCoeffs, is missing, all of its elements shall be set to 1.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

Otherwise, if the attribute video sub-bitstream is an auxiliary attribute video, either identified by vuh_auxiliary_video_flag equal to 1 or determined through external means if the V3C unit header is unavailable, the outputs of this process are:

— the 1D array DecAttrAuxType, indicating the attribute type, where the dimension corresponds to the attribute index,

— the 2D array NumDecAttrAuxFrames, indicating the number of decoded auxiliary attribute video frames, where the dimensions correspond to the attribute index and the attribute partition index,

— the 6D array DecAttrAuxFrames, the decoded auxiliary attribute video frames, where the dimensions correspond to the attribute index, the attribute partition index, the decoded attribute video frame index, the component index, the row index and the column index, respectively, and

— the following 3D arrays:

  — DecAttrAuxBitDepth, indicating the auxiliary attribute bit depth,

  — DecAttrAuxHeight, indicating the auxiliary attribute height,

  — DecAttrAuxWidth, indicating the auxiliary attribute width,

  — DecAttrAuxNumComp, indicating the attribute number of components,

  — DecAttrAuxChromaFormat, indicating the auxiliary attribute chroma format,

  — DecAttrAuxChromaSamplingPosition, indicating, if present, the attribute chroma sampling as specified in ISO/IEC 23091-2,

  — DecAttrAuxFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,

  — DecAttrAuxColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,

  — DecAttrAuxTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,

  — DecAttrAuxMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,

  — DecAttrAuxOutOrdIdx, indicating the auxiliary attribute output order index, and

  — DecAttrAuxCompTime, indicating the auxiliary attribute composition time,

where the dimensions correspond to the attribute index, the attribute partition index and the decoded attribute frame index, respectively.

If the array DecAttrAuxChromaSamplingPosition, is missing, all of its elements shall be set to 0.

If the array DecAttrAuxFullRange, is missing, all of its elements shall be set to 1.

If the array DecAttrAuxColourPrimaries, is missing, all of its elements shall be set to 1.

If the array DecAttrAuxTransferCharacteristics, is missing, all of its elements shall be set to 1.

If the array DecAttrAuxMatrixCoeffs, is missing, all of its elements shall be set to 1.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

NOTE Any existing video coding specification such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video coding specification can be used if included in ai_attribute_codec_id.

9.6 Sub-bitstream extraction process

9.6.1 General

In order to perform the decoding processes identified in subclauses 9.2, 9.3, 9.4 and 9.5, it may be necessary to extract the appropriate V3C units, e.g. atlas or occupancy data, from the V3C bitstream, or ACL and non-ACL NAL units of specific type from an atlas sub-bitstream. Subclause 9.6.2 describes the extraction of V3C units. Subclause 9.6.3 describes the extraction of NAL units.

9.6.2 V3C unit extraction

Inputs to this process are a V3C bitstream, such as a bitstream in the V3C sample stream format defined in Annex C, a target atlas identifier, targetAtlasId, and a set of parameters specifying the component or components to be extracted.

First the locations of the V3C unit boundaries in a V3C bitstream are identified. For example, for the case of the V3C sample stream format specified in Annex C, the location of each boundary can be identified using the syntax element ssvu_v3c_unit_size.

Then, the extraction process for each V3C unit is performed as follows:

If vuh_unit_type in the V3C unit headers is equal to V3C_AVD, V3C_GVD, V3C_OVD, or V3C_AD, and the vuh_atlas_id is equal to targetAtlasId, then the following applies based on the input extraction parameters:

— If extraction of atlas sub-bitstreams is indicated, then if vuh_unit_type is equal to V3C_AD in the current V3C unit, then the V3C unit payload is extracted as an output.

— If extraction of occupancy sub-bitstreams is indicated, then if vuh_unit_type is equal to V3C_OVD in the current V3C unit, then the V3C unit payload is extracted as an output.

— If extraction of geometry sub-bitstreams with map index mapIdx is indicated, then if vuh_unit_type is equal to V3C_GVD, vuh_map_index is equal to mapIdx, and vuh_auxiliary_video_flag is equal to 0 in the current V3C unit, then the V3C unit payload is extracted as an output.

— If extraction of auxiliary geometry sub-bitstreams is indicated, then if vuh_unit_type is equal to V3C_GVD, and vuh_auxiliary_video_flag is equal to 1 in the current V3C unit, then the V3C unit payload is extracted as an output.

— If extraction of attribute sub-bitstreams with attribute index attrIdx, map index mapIdx, and partition index partIdx, is indicated, then if vuh_unit_type is equal to V3C_AVD, vuh_attribute_index is equal to attrIdx, vuh_map_index is equal to mapIdx, vuh_attribute_partition_index is equal to partIdx, and vuh_auxiliary_video_flag is equal to 0 in the current V3C unit, then the V3C unit payload is extracted as an output.

— If extraction of auxiliary attribute sub-bitstreams with attribute index attrIdx and partition index partIdx, is indicated, then if vuh_unit_type is equal to V3C_AVD, vuh_attribute_index is equal to attrIdx, vuh_attribute_partition_index is equal to partIdx, and vuh_auxiliary_video_flag is equal to 1 in the current V3C unit, then the V3C unit payload is extracted as an output.

It is a requirement of bitstream conformance for the input bitstream that any output sub-bitstream that is the output of the process specified in this clause shall be a conforming bitstream according to the codec specification identified by either the CodecGroup profile component or, if present, the corresponding codec indicated by the component codec mapping SEI message.

9.6.3 NAL unit extraction process

Inputs to this process are an atlas sub-bitstream, such as a bitstream in the NAL sample stream format defined in Annex D, and a set of parameters specifying whether it is an ACL or non-ACL NAL unit and when it is an ACL NAL unit, optionally, the tile ID to be extracted.

First the locations of the NAL unit boundaries in a NAL bitstream are identified. For example, for the case of the NAL sample stream format specified in Annex D, the location of each boundary can be identified using the syntax element ssnu_nal_unit_size.

Then, the extraction process for each NAL unit is performed as follows:

— If extraction of ACL NAL units with a specific tile ID, tileID, is indicated, then all ACL NAL units with ath_id equal to tileID and their associated non-ACL NAL units are extracted.

— If extraction of ACL NAL units is indicated without specifying a specific tile ID, then all ACL NAL units and their associated non-ACL NAL units are extracted.

— If extraction of a non-ACL NAL unit of a specific NAL type, nalType, is indicated, then all NAL units with nal_unit_type equal to nalType, and their associated non-ACL NAL units are extracted.

10 Pre-reconstruction process

The pre-reconstruction process depends on the profiles defined in Annex A.

Some applications within the scope of this document define explicit pre-reconstruction processing steps. However, some other applications may not, in which case the pre-reconstruction process is outside the scope of this document. For example, in the context of the video-based point cloud decoding applications, as defined in Annex H, and the associated profiles, a pre-reconstruction process, which performs occupancy synthesis operations, is specified in Clause H.6.

11 Reconstruction process

The reconstruction process depends on the profiles defined in Annex A.

Some applications within the scope of this document define explicit reconstruction processing steps. However, some other applications may not, in which case the reconstruction process is outside the scope of this document. For example, in the context of the video-based point cloud decoding applications, as defined in Annex H, and the associated profiles, a nominal reconstruction process is specified in Clause H.7.

12 Post-reconstruction process

The post-reconstruction process depends on the profiles defined in Annex A.

Some applications within the scope of this document define explicit post-reconstruction processing steps. However, some other applications may not, in which case the post-reconstruction process is outside the scope of this document. For example, in the context of the video-based point cloud decoding applications, as defined in Annex H, and the associated profiles, a post-reconstruction process, which performs a number of smoothing operations on the reconstructed point cloud data, is specified in Clause H.8.

13 Adaptation process

The adaptation process depends on the profiles defined in Annex A

Some applications within the scope of this document define explicit adaptation process steps. However, some other applications may not, in which case the adaptation process is outside the scope of this document. For example, in the context of the video-based point cloud decoding applications, as defined in Annex H, and the associated profiles, an adaptation process is specified in Clause H.9.

14 Parsing process

14.1 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 subclause 14.2).

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

14.2.1 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 non-zero bit and counting the number of leading bits that are equal to 0. This process is specified as follows:

```
leadingZeroBits = −1
for( b = 0; !b; leadingZeroBits++ ) (166)
        b = read_bits( 1 )
```

The variable codeNum is then assigned as follows:

$$\text{codeNum} = 2^{\text{leadingZeroBits}} - 1 + \text{read\_bits(leadingZeroBits)} \quad (167)$$

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

Table 12 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 12. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as x_i in Table 12, with i in the range of 0 to leadingZeroBits − 1, inclusive. Each x_i is equal to either 0 or 1.

Table 12 — Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges

Bit string form	Range of codeNum
1	0
0 1 x_0	1..2
0 0 1 x_1 x_0	3..6
0 0 0 1 x_2 x_1 x_0	7..14
0 0 0 0 1 x_3 x_2 x_1 x_0	15..30

Table 12 *(continued)*

Bit string form	Range of codeNum
0 0 0 0 0 1 x_4 x_3 x_2 x_1 x_0	31..62
...	...

Table 13 illustrates explicitly the assignment of bit strings to codeNum values.

Table 13 — Exp-Golomb bit strings and codeNum in explicit form and used as ue(v)

Bit string	codeNum
1	0
0 1 0	1
0 1 1	2
0 0 1 0 0	3
0 0 1 0 1	4
0 0 1 1 0	5
0 0 1 1 1	6
0 0 0 1 0 0 0	7
0 0 0 1 0 0 1	8
0 0 0 1 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 subclause 14.2.2 with codeNum as input.

14.2.2 Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in subclause 14.1.

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 14 provides the assignment rule.

Table 14 — Assignment of syntax element to codeNum

codeNum	syntax element value
0	0
1	1
2	−1
3	2
4	−2
5	3
6	−3
k	$((k + 1) / 2) * (-1)^{k+1}$

Annex A
(normative)

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 document. 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 V3C decoders conforming to this edition of this document are specified in terms of the ability to decode V3C bitstreams or collection of V3C sub-bitstreams 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 can include the reconstruction in 3D space, i.e. “decode” can refer to only decoding the 2D video sub-bitstreams (i.e. geometry, occupancy, attribute(s)) and the associated atlas sub-bitstream, or it can include further reconstruction into 3D space. The respective steps included in the “decoding” are specified for each profile.

In this annex, phrases like "the bitstream" should be intepreted as "the V3C bitstream of the operation point", and phrases like "coded atlas access unit n" should be interpreted as "coded atlas access unit n in the bitstream of the operation point", where "the operation point" is the operation point with which the profile, tier, or level is associated with.

The variables HtidOP and TemporalID, which indicate the highest temporal layer present in an atlas sub-bitstream and the temporal ID of an atlas sub-bitstream, respectively, shall be equal to 0 in this edition ofthe document.

For each operation point identified by HtidOP, the profile, tier and level information are indicated by a combination of syntax elements, including ptl_profile_codec_group_idc and ptl_profile_toolset_idc, that are found in or derived from the profile_tier_level( ) syntax structure in the VPS, if available, or determined through external means.

A.2 Profile, tier and level structure

V3C 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.

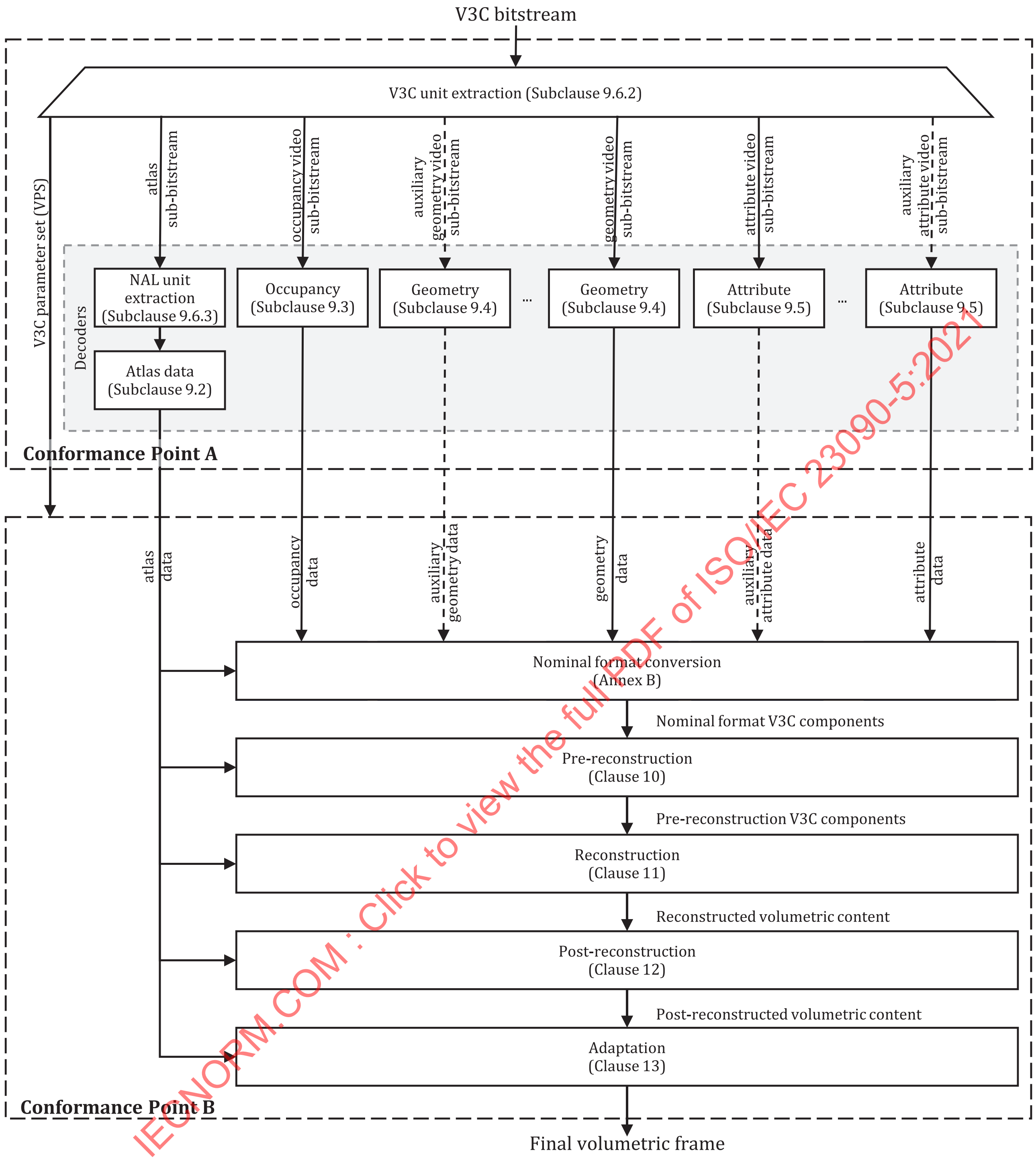

V3C bitstream
V3C unit extraction (Subclause 9.6.2)
V3C parameter set (VPS)
atlas sub-bitstream
occupancy video sub-bitstream
auxiliary geometry video sub-bitstream
geometry video sub-bitstream
attribute video sub-bitstream
auxiliary attribute video sub-bitstream
Decoders
NAL unit extraction (Subclause 9.6.3)
Atlas data (Subclause 9.2)
Occupancy (Subclause 9.3)
Geometry (Subclause 9.4)
...
Geometry (Subclause 9.4)
Attribute (Subclause 9.5)
...
Attribute (Subclause 9.5)
Conformance Point A
atlas data
occupancy data
auxiliary geometry data
geometry data
auxiliary attribute data
attribute data
Nominal format conversion (Annex B)
Nominal format V3C components
Pre-reconstruction (Clause 10)
Pre-reconstruction V3C components
Reconstruction (Clause 11)
Reconstructed volumetric content
Post-reconstruction (Clause 12)
Post-reconstructed volumetric content
Adaptation (Clause 13)
Conformance Point B
Final volumetric frame

Figure A.1 — V3C decoding block diagram with decoding conformance points A and B

The first conformance point, point A, covers the decoded video sub-bitstreams and atlas sub-bitstream. It also covers the derived block to patch map information. It does not, however, cover the reconstruction process.

The second conformance point, point B, covers the reconstruction process.

A V3C profile decoding capability is defined by a combination of syntax elements ptl_profile_codec_group_idc and ptl_profile_toolset_idc that are present in the VPS or obtained through external means if the VPS is unavailable. A V3C profile reconstruction capability is optionally defined by 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 decoding specifications and their profiles (e.g., Progressive High as specified in ISO/IEC 14496-10:2020, Annex A, Main or Main 10 as specified in ISO/IEC 23008-2:2020, Annex A), referred to as the **CodecGroup** profile component, and

— V3C 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 pre-reconstruction, reconstruction, post-reconstruction, and adaptation tools supported or recommended to achieve conformance in terms of 3D reconstruction. This is referred to as the **Reconstruction** profile component.

Together, these profile components form a V3C profile, which follows the naming convention **CodecGroup Toolset Reconstruction**. For example, taking the first blocks of Figure A.2, the resulting V3C profile is named as **AVC V-PCC Basic Rec0.**

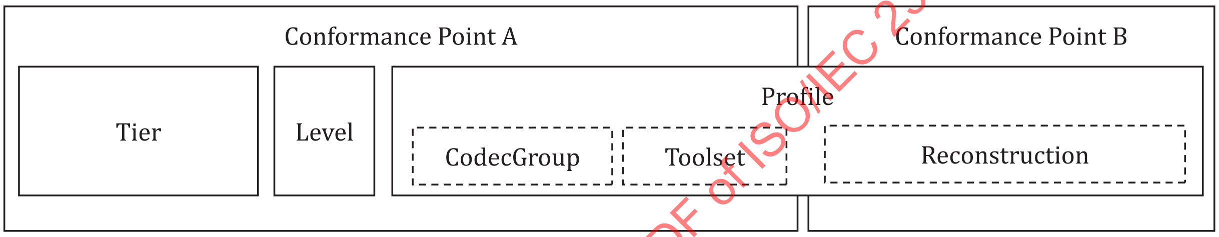


Figure A.2 — V3C profile, tier and level structure

Conformance can be specified for a single V3C bitstream, as defined in Annex C, or for a collection of V3C sub-bitstreams that are associated with each other and provide information for the reconstruction of a sequence of volumetric frames.

A V3C bitstream may consist of one or more atlas sub-bitstreams, and each atlas may include additional sub-bitstreams, such as an occupancy sub-bitstream, 0 or more geometry map sub-bitstreams, and 0 or more attributes of different type.

Any V3C bitstream, collection of V3C sub-bitstreams, or V3C decoder can claim support of both conformance point A and conformance point B. However, indicating conformance point A is mandatory, while conformance point B is optional. For example:

— **AVC V-PCC Basic**, is a valid profile that clearly indicates its CodecGroup and Toolset profile components while providing no indication of a Reconstruction profile component.

— **AVC V-PCC Basic Rec0**, is a valid profile that clearly indicates its CodecGroup and Toolset profile components while also indicating support of a Reconstruction profile component that provides for elementary 3D reconstruction.

— **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 or decoder at conformance point A, it is only needed to consider the CodecGroup and Toolset profile components. Decoders conforming to a V3C profile at conformance point A (identified by syntax elements ptl_profile_codec_group_idc and ptl_profile_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 V3C bitstreams or collection of V3C sub-bitstreams, according to Clause 9, for which all of the following conditions apply:

— The V3C bitstream or the collection of V3C sub-bitstreams are indicated to conform to the supported CodecGroup and Toolset profile components, as indicated in Clauses A.3 (CodecGroup) and A.4 (Toolset).

— The V3C bitstream or the collection of V3C sub-bitstreams are indicated to conform to a level that is lower than or equal to the specified level, as indicated in Clause A.6.

— The V3C bitstream or the collection of V3C sub-bitstreams are indicated to conform to a tier that is lower than or equal to the specified tier, as indicated in Clause A.6.

A decoder may support and may be able to conform to multiple reconstruction profiles as specified in Clause A.5. However, the reconstruction profile a decoder selects to operate in, is outside the scope of this document. When a decoder claims to be operating at a particular conformance point B, it shall be capable of reconstructing a volumetric frame according to the reconstruction recommendations specified by that profile.

A.3 CodecGroup profile components

Table A.1 provides a list of defined CodecGroup profile components for V3C and respectively the allowed values for syntax element ptl_profile_codec_group_idc.

Table A.1 — Available CodecGroup profile components

CodecGroup	ptl_profile_codec_group_idc	4CC code
AVC Progressive High	0	'avc3'
HEVC Main10	1	'hev1'
HEVC444	2	'hev1'
VVC Main10	3	'vvc1'
Reserved	4..126	–
MP4RA	127	provided by component codec mapping SEI message (F.2.11)

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			VVC Main 10			MP4RA		
	Capability	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes
Chroma format	Mono							✓	✓	✓	✓	✓	✓	–	–	–
	4:2:0	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
	4:4:4									✓				–	–	–
Bit depth	8 bit	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–
	10 bit				✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–

Table A.2 *(continued)*

		AVC Progressive High			HEVC Main10			HEVC444			VVC Main 10			MP4RA		
	Capability	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes	Occupancy	Geometry	Attributes
Video coding specification	name	ISO/IEC 14496-10			ISO/IEC 23008-2						ISO/IEC 23090-3 (Under preparation)			As defined by a component codec mapping SEI (F.2.11)		
	profile	Progressive High as specified in ISO/IEC 14496-10			Main 10 as specified in ISO/IEC 23008-2			Main 4:4:4 10 as specified in ISO/IEC 23008-2			VVC Main 10 as specified in ISO/IEC 23090-3			–		

A tick mark in Table A.2 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 document.

All video sub-bitstreams of a V3C bitstream or a collection of V3C sub-bitstreams conform to the stream format indicated in the component codec mapping SEI (F.2.11).

For ptl_profile_codec_group_idc equal to 0, all video sub-bitstreams shall conform to the stream format identified by 4CC code equal to 'avc3'. The codec ids associated with all video sub-bitstreams in the active VPS are inferred to be mapped to 'avc3'. The component codec mapping SEI message (F.2.11), when present, shall also indicate the value of 'avc3' for all instances j of ccm_codec_4cc[j] present in this SEI message.

For ptl_profile_codec_group_idc equal to 1 or 2, all video sub-bitstreams shall conform to the stream format identified by 4CC code equal to 'hev1'. The codec ids associated with all video sub-bitstreams in the active VPS are inferred to be mapped to 'hev1'. The component codec mapping SEI message (F.2.11), when present, shall also indicate the value of 'hev1' for all instances j of ccm_codec_4cc[j] present in this SEI message.

For ptl_profile_codec_group_idc equal to 3, all video sub-bitstreams shall conform to the stream format identified by 4CC code equal to 'vvc1'. The codec ids associated with all video sub-bitstreams in the active VPS are inferred to be mapped to 'vvc1'. The component codec mapping SEI message (F.2.11), when present, shall also indicate the value of 'vvc1' for all instances j of ccm_codec_4cc[j] present in this SEI message.

For ptl_profile_codec_group_idc equal to 127, a video sub-bitstream of an atlas with atlas ID equal to atlasID that is associated with a particular codec id codecID, where codecID may correspond to the value of oi_occupancy_codec_id[atlasID], gi_geometry_codec_id[atlasID], gi_auxiliary_geometry_codec_id[atlasID], ai_attribute_codec_id[atlasID][i], and ai_auxiliary_attribute_codec_id[atlasID][i] depending on the type of the video, i.e. occupancy, geometry, auxiliary geometry, attribute with index i, or auxiliary attribute with index i, shall conform to the stream format indicated by the value of ccm_codec_4cc[codecID] as defined in the component codec mapping SEI message (F.2.11).

When stream format is identified by a 4CC code defined in ISO/IEC 14496-15, the number of bytes that indicate the NAL unit length, NALUnitLength, defined in ISO/IEC 14496-15 as being equal to the field LengthSizeMinusOne + 1, shall be equal to 4.

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 document.

A.4 Toolset profile components

Table A.3 provides a list of defined toolset profile components for V3C and their corresponding identifying syntax element values, e.g. ptl_profile_toolset_idc and ptc_one_v3c_frame_only flag, that are defined for use by this document. The syntax element ptl_profile_toolset_idc provides the primary definition of a toolset profile. However, additional syntax elements, such as ptc_one_v3c_frame_only_flag, can indicate additional characteristics or restrictions of a defined profile. In the case of ptc_one_v3c_frame_only_flag, this syntax element indicates that such a profile can only support a single V3C frame. Reserved values of ptl_profile_toolset_idc are reserved for future use by ISO/IEC and shall not be present in V3C bitstreams conforming to this edition of this document. The defined profiles in Table A.3 are also characterized as either dynamic or static, depending on whether they are targeting dynamic or static, i.e. single, volumetric frame applications.

Table A.3 — Available toolset profile components

ptl_profile_toolset_idc	ptc_one_v3c_frame_only_flag	Toolset profile component	Type
0	0	V-PCC Basic /* Specified in Annex H*/	Dynamic
	1	V-PCC Basic Still /* Specified in Annex H*/	Static
1	0	V-PCC Extended /* Specified in Annex H*/	Dynamic
	1	V-PCC Extended Still /* Specified in Annex H*/	Static
2..255	-	Reserved	-

A.5 Reconstruction profile components

Table A.4 provides a list of defined reconstruction profile components for V3C and of the allowed values for syntax element ptl_profile_reconstruction_idc. Values of ptl_profile_reconstruction_idc in the range of 0 to 254, inclusive, may be specified by a V3C application and may depend on the value of ptl_profile_toolset_idc. For example, for the case of ptl_profile_toolset_idc equal to 0 or 1, which indicates a V-PCC application, subclause H.11.5 specifies reconstruction profiles appropriate for that application.

For the Rec Unconstrained reconstruction profile, the reconstruction process is outside the scope of this document.

Table A.4 — Defined reconstruction profile components

ptl_profile_reconstruction_idc	Reconstruction profile component
0..254	Application dependent
255	Rec Unconstrained

A.6 Tiers and levels

A.6.1 General tier and level limits

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.

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.

The following is specified for expressing the constraints in this annex:

— Let atlas coded access unit n be the n-th access unit in decoding order, with the first access unit being access unit 0 (i.e., the 0-th access unit).

Let atlas n be the coded atlas or the corresponding decoded atlas of access unit n.

When a specified level is not level 8.5, V3C bitstreams conforming to all profiles specified in this document at a specified tier and level shall obey the following constraints for each atlas sub-bitstream conformance test as specified in Annex E:

a) AspsFrameSize shall be less than or equal to MaxAtlasSize, where MaxAtlasSize is specified in Table A.6.

b) vps_map_count_minus1 + 1 shall be less than or equal to LevelMapCount, where LevelMapCount is specified in Table A.5.

c) ai_attribute_count shall be less than or equal to MaxNumAttributeCount, where MaxNumAttributeCount is specified in Table A.5.

d) ai_attribute_dimension_minus1 + 1 shall be less than or equal to MaxNumAttributeDims, where MaxNumAttributeDims is specified in Table A.5.

e) afti_num_tiles_in_atlas_frame_minus1 + 1 shall be less than MaxNumTiles is specified in Table A.6.

f) When the current atlas frame is an IDR coded atlas ath_atlas_frm_order_cnt_lsb shall be equal to 0.

A.6.2 Profile specific level limits for the V3C dynamic profiles

A tier and level to which V3C 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.5.

Table A.5 — General V3C or VPS related level limits

Level	Max # of projected points per second MaxNumProjPointsPerSec	Max # of EOM points per second MaxNumEomPointsPerSec	Max # of RAW points per second MaxNumRawPointsPerSec	Max # of projected points per atlas MaxNumProjPoints	Max # EOM points per atlas MaxNumEomPoints	Max # RAW points per atlas MaxNumRawPoints	Max # of maps LevelMapCount	Max # of attributes MaxNumAttributeCount	Max # of attribute dimensions MaxNumAttributeDims
1.0	30 000 000	1 500 000	1 500 000	1 000 000	50 000	50 000	2	1	3
1.5	60 000 000	3 000 000	3 000 000	2 000 000	100 000	100 000	2	3	3
2.0	120 000 000	6 000 000	6 000 000	4 000 000	200 000	200 000	4	4	3
2.5	240 000 000	12 000 000	12 000 000	8 000 000	400 000	400 000	4	8	4
3.0	480 000 000	24 000 000	24 000 000	16 000 000	800 000	800 000	8	16	5
3.5	960 000 000	48 000 000	48 000 000	32 000 000	1 600 000	1 600 000	8	24	6

Table A.5 specifies the general V3C and VPS related limits for each level of each tier.

Let the variable t0 denote the time instance in milliseconds at which the first bit of the coded atlas access unit 0 begins to enter the CAB.

For any time t1 greater than or equal to t0, the variables NumProjPointsPerSec, NumRawPointsPerSec, and NumEomPointsPerSec, are defined and set equal to the total number of NumProjPoints, NumRawPoints, and NumEomPoints, respectively, of the volumetric frames that are output for all values of t, with t1 <= t < t1 + 1 000. NumRawPoints and NumEomPoints are specified in subclause 9.2.7, and NumProjPoints is specified in subclause B.2.2.

For any time t1 greater than or equal to t0, the variables AtlasTotalNumProjPatchesPerSec, AtlasTotalNumRawPatchesPerSec, and AtlasTotalNumEomPatchesPerSec, are defined and set equal to the total number of AtlasTotalNumProjPatches, AtlasTotalNumRawPatches, and AtlasTotalNumEomPatches, respectively, of the atlas frames that are decoded for all values of t, with t1 <= t < t1 + 1 000. AtlasTotalNumProjPatches, AtlasTotalNumRawPatches, and AtlasTotalNumEomPatches are specified in subclause 9.2.7.

V3C bitstreams conforming to dynamic profiles at a specified tier and level shall obey the following constraints for each atlas bitstream conformance test as specified in Annex E.

— NumProjPoints shall be less than or equal to MaxNumProjPoints,

— NumRawPoints shall be less than or equal to MaxNumRawPoints,

— NumEomPoints shall be less than or equal to MaxNumEomPoints,

— NumProjPointsPerSec shall be less than or equal to MaxNumProjPointsPerSec,

— NumRawPointsPerSec shall be less than or equal to MaxNumRawPointsPerSec, and

— NumEomPointsPerSec shall be less than or equal to MaxNumEomPointsPerSec

where MaxNumProjPoints, MaxNumRawPoints, MaxNumEomPoints, MaxNumProjPointsPerSec, MaxNumRawPointsPerSec and MaxNumEomPointsPerSec are specified in Table A.5.

Table A.6 — General atlas ASPS and tile related level limits

Level	Max # projected patches per atlas MaxNumProjPatches	Max # RAW patches per atlas MaxNumRawPatches	Max # EOM patches per atlas MaxNumEomPatches	Max CAB size MaxCABSize in 1 000 bits	Max atlas bit rate MaxAtlasBR in 1 000 bits/s	Max # number of tiles per atlas MaxNumTiles	Max atlas size MaxAtlasSize	Max # of projected patches per second MaxProjPatchesPerSec	Max # of RAW patches per second MaxRawPatchesPerSec	Max # of EOM patches per second MaxEomPatchesPerSec
1.0	2 048	32	32	15 000	15 000	50	2 228 224	65 536	1 024	1 024
1.5	4 096	64	32	30 000	30 000	50	2 228 224	131 072	2 048	1 024
2.0	16 384	128	64	120 000	120 000	200	8 912 896	524 288	4 096	2 048
2.5	32 384	128	64	240 000	240 000	200	8 912 896	1 036 288	4 096	2 048
3.0	65 536	512	128	480 000	480 000	500	35 651 584	2 097 152	16 384	4 096
3.5	65 536	512	512	480 000	480 000	500	35 651 584	4 194 304	32 768	32 768

Table A.6 specifies the atlas ASPS and tile related level limits.

The specified limits apply to all atlas tiles forming the V3C bitstream.

— AtlasTotalNumProjPatches shall be less than or equal to MaxNumProjPatches,

— AtlasTotalNumRawPatches shall be less than or equal to MaxNumRawPatches,

— AtlasTotalNumEomPatches shall be less than or equal to MaxNumEomPatches,

— AtlasTotalNumProjPatchesPerSec shall be less than or equal to MaxProjPatchesPerSec,

— AtlasTotalNumRawPatchesPerSec shall be less than or equal to MaxRawPatchesPerSec,

— AtlasTotalNumEomPatchesPerSec shall be less than or equal to MaxEomPatchesPerSec, and

— afti_num_tiles_in_atlas_frame_minus1 + 1 shall be less than or equal to MaxNumTiles,

where MaxNumProjPatches, MaxNumRawPatches, MaxNumEomPatches, MaxProjPatchesPerSec, MaxRawPatchesPerSec, MaxEomPatchesPerSec and MaxNumTiles are specified in Table A.6.

Table A.7 — General video bitstream level limits

Level	Max luma picture size in MaxPictureSize (samples)	Max aggregate luma sample rate MaxAggregateLumaSr (samples/sec)	Max # 1000 bits/s per video stream MaxBitRatePerStream		Max aggregate # 1000 bits/s MaxAggregateBitRate	
			MT	HT	MT	HT
1.0	2 228 224	133 693 440	12 000	30 000	25 000	100 000
1.5	2 228 224	267 386 880	20 000	50 000	40 000	160 000
2.0	8 912 896	534 773 760	25 000	100 000	60 000	240 000
2.5	8 912 896	1 069 547 520	60 000	240 000	120 000	480 000
3.0	35 651 584	2 139 095 040	60 000	240 000	120 000	480 000
3.5	35 651 584	4 278 190 080	240 000	800 000	480 000	1 600 000

Table A.7 specifies the general video bitstream level limits.

For example, when ptl_profile_codec_group_idc is equal to 0, the maximum sample rate of a single video sub-bitstream is specified in ISO/IEC 14496-10:2020, Table A.1 as MaxMBPS * MaxFS. When ptl_profile_codec_group_idc is equal to 1 or equal to 2, the maximum sample rate of a single video sub-bitstream is specified as MaxLumaSr in ISO/IEC 23008-2:2020, Table A.7. When ptl_profile_codec_group_idc is equal to 127, the definition of the maximum sample rate of a single video sub-bitstream is outside the scope of this document.

Let the variable t0 to denote the time instance in milliseconds when the first bit across all V3C video sub-bitstreams is encountered.

For any time t1 greater than or equal to t0, the bit rate for each individual V3C video sub-bitstream, V3CVideoBitRate, is defined and set equal to the total number of decoded bits for the individual V3C video sub-bitstream for all values of t, with t1 <= t < t1 + 1 000.

It is a requirement of V3C bitstream conformance that V3CVideoBitrate is less than or equal to MaxBitRatePerStream, as specified in Table A.7, for all individual V3C video sub-bitstreams.

For any time t1 greater than or equal to t0, the aggregate sample rate, AggregateLumaSr, and the aggregate bit rate, AggregateBitRate, are defined and set equal to the total number of decoded luma samples and the total number of decoded bits, respectively, across all video components for all values of t, with t1 <= t < t1 + 1 000.

It is a requirement of V3C bitstream conformance that AggregateLumaSr and AggregateBitRate are less than or equal to MaxAggregateLumaSr and MaxAggregateBitRate, respectively, as specified in Table A.7.

For each video sub-bistream, the coded picture buffer size shall be less than or equal to the maximum coded picture buffer size, as specified by the corresponding video codec specification. For example, when ptl_profile_codec_group_idc is equal to 0, the maximum coded picture buffer size is identified by a variable MaxCPB specified in ISO/IEC 14496-10:2014, Table A.1. When ptl_profile_codec_group_idc is equal to 1 or equal to 2, the maximum coded picture buffer size is also identified by a variable MaxCPB specified in ISO/IEC 23008-2:2020, Annex A. When ptl_profile_codec_group_idc is equal to 127, the derivation of the maximum coded picture buffer size is outside the scope of this document.

The following variables are specified:

— Let the variable fR be set equal to 1 ÷ 1 000.

— Let the variable MaxPatchSize, which represents the maximum total number of bytes that is expected to be associated with all decoded elements in a patch, be equal to 40 bytes.

— Let the variable aR = AspsFrameSize ÷ MaxAggregateLumaSr.

— Let access unit n be the n-th access unit in decoding order, with the first access unit being access unit 0 (i.e., the 0-th access unit).

— Let atlas n be the coded atlas or the corresponding decoded atlas of access unit n.

— Let the variable MaxNumSubLayersMinus1 be equal to 0.

Then the following constraints apply:

a) The value of asps_max_dec_atlas_frame_buffering_minus1 + 1 shall be less than or equal to 16.

b) For the ACL HRD parameters, CabSize[1][HtidOP][i] shall be less than or equal to MaxCABSize for at least one value of i in the range of 0 to hrd_cab_cnt_minus1[HtidOP], inclusive, where CabSize[1][HtidOP][i] is specified in subclause G.3.3 based on parameters selected as specified in Clause E.1 and MaxCABSize is specified in Table A.5 in bits.

c) For the NAL HRD parameters, CabSize[0][HtidOP][i] shall be less than or equal to MaxCABSize for at least one value of i in the range of 0 to hrd_cab_cnt_minus1[HtidOP], inclusive, where CabSize[0][HtidOP][i] is specified in subclause G.3.3 based on parameters selected as specified in Clause E.1 and MaxCABSize is specified in Table A.1 in bits.

d) The nominal removal time of a coded atlas access unit n (with n greater than 0) from the CAB, as specified in subclause E.2.3, shall satisfy the constraint that AuNominalRemovalTime[n] − AuCabRemovalTime[n − 1] is greater than or equal to Max(aR, fR).

e) The difference between consecutive output times of atlases from the DAB, as specified in subclause E.3.3, shall satisfy the constraint that DabOutputInterval[n] is greater than or equal to Max(aR, fR) provided that atlas is not the last atlas of the atlas bitstream that is output.

f) For the ACL HRD parameters, BitRate[1][HtidOP][i] shall be less than or equal to MaxAtlasBR for at least one value of i in the range of 0 to hrd_cab_cnt_minus1[HtidOP], inclusive, where BitRate[1][HtidOP][i] is specified in subclause G.3.3 based on parameters selected as specified in Clause E.1 and MaxAtlasBR is specified in Table A.6.

g) For the NAL HRD parameters, BitRate[0][HtidOP][i] shall be less than or equal to MaxAtlasBR for at least one value of i in the range of 0 to hrd_cab_cnt_minus1[HtidOP], inclusive, where BitRate[0][HtidOP][i] is specified in subclause G.3.3 based on parameters selected as specified in Clause E.1 and MaxAtlasBR is specified in Table A.7.

h) For coded atlas access unit 0, let the variables maxNumPatches, maxPatchBytes and nB be defined as follows:

```
maxNumPatches = MaxNumProjPatches + MaxNumRawPatches + MaxNumEomPatches
maxPatchBytes = MaxPatchRate * MaxPatchSize
nB = Max( maxNumPatches, fR * MaxPatchRate ) +
    maxPatchBytes * ( AuCabRemovalTime[ 0 ] − AuNominalRemovalTime[ 0 ] )
```

where the values of MaxNumProjPatches, MaxNumRawPatches, MaxNumEomPatches and MaxPatchRate are specified in Table A.6.

The sum of the NumBytesInNalUnit variables for coded atlas access unit 0 shall be less than or equal to nB.

i) For coded atlas access unit n, with n greater than 0, let the variables maxPatchBytes and nB be defined as follows:

```
    maxPatchBytes = MaxPatchRate * MaxPatchSize
nB = maxPatchBytes * ( AuCabRemovalTime[ n ] − AuCabRemovalTime[ n − 1 ] )
```

where MaxPatchRate is specified in Table A.6.

The sum of the NumBytesInNalUnit variables for access unit n shall be less than or equal to nB.

Annex B
(normative)

Post-decoding conversion to nominal video formats

B.1 General

The video frames provided by the decoder may require additional processing steps before being input to the reconstruction process. Such processing steps may include conversion of the decoded video frames to a nominal format (e.g. a nominal resolution, bit depth, chroma format, etc.) or a conversion operation (e.g. map extraction, multi-dimensional attribute packing, etc.), as described in Clause B.2. This clause provides methods of how to perform such processing steps. These may be substituted by alternative processes, which are outside of the scope of this document, that may produce similar visual quality. The definition of similar visual quality is outside the scope of this document. However, for purposes of conformance to point B and as an example, a post-decoding operation flow is presented in Figure B.1. The order of processing steps may not necessarily be the same as presented in Figure B.1.

The processes described in Annex B are invoked for decoded video components associated with the same atlas ID, identified by variable ConvAtlasID, which is set equal to vuh_atlas_id or determined through external means if the V3C unit header is unavailable.

This clause takes as inputs the syntax elements and upper-case variables from Clauses 8 and 9.

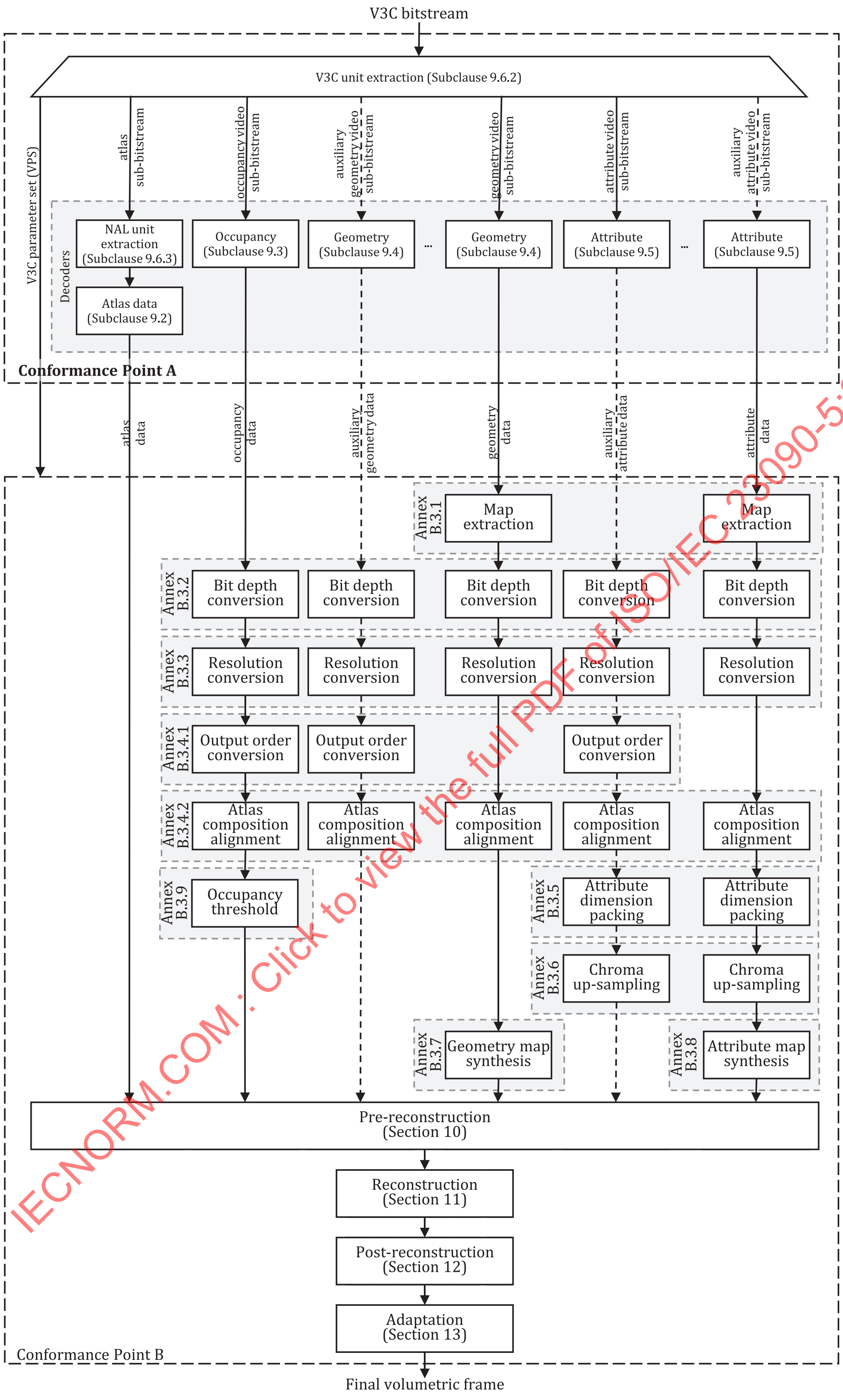


Figure B.1 — Post-processing conversion for a V3C bitstream with a single atlas

B.2 Nominal format conversion

B.2.1 General

The nominal format refers collectively to the nominal bit depth, resolution, chroma format and composition time index that the decoded videos should be converted to.

Each video sub-bitstream is associated with a nominal bit depth, which is the target bit depth that all operations for reconstruction are expected to be performed in. The nominal bit depth for the occupancy component, OccBitDepthNF, is set equal to oi_occupancy_2d_bit_depth_minus1[ConvAtlasID] + 1. The nominal bit depth for each geometry video component, GeoBitDepthNF, is set equal to gi_geometry_2d_bit_depth_minus1[ConvAtlasID] + 1. Finally, the nominal bit depth for each attribute video component with attribute index attrIdx, AttrBitDepthNF[attrIdx], is set equal to ai_attribute_2d_bit_depth_minus1[ConvAtlasID][attrIdx] + 1.

The nominal frame resolution for non-auxiliary video components is defined by the nominal width, VideoWidthNF, set equal to asps_frame_width, and the nominal height, VideoHeightNF, set equal to asps_frame_height. The nominal frame resolution for auxiliary video components is defined by the nominal width and height specified by the variables AuxVideoWidthNF and AuxVideoHeightNF, respectively.

The nominal chroma format is defined to be 4:4:4.

The nominal composition time is defined based on the timing information of the atlas sub-bitstream and specified by the decoded atlas composition time, DecAtlasCompTime. The timing information of the atlas sub-bitstream may be provided either in the bitstream or by external means not specified in this document. For example, for V3C bitstreams encapsulated in the ISO media file format, the composition time for the atlas sub-bitstream is indicated according to ISO/IEC DIS 23090-10.

The conversion processes to the nominal format are defined as follows:

— for the occupancy video component subclause B.2.2 is invoked,

— for the geometry video component subclause B.2.3 is invoked,

— if the auxiliary geometry video component is present, for the auxiliary geometry video component subclause B.2.4 is invoked, and

— when the number of attributes is greater than zero, i.e. ai_attribute_count[ConvAtlasID] is greater than zero or determined through external means if the VPS is not available, then:

  — for each attribute video component subclause B.2.5 is invoked and

  — if auxiliary attribute video component is present, for each auxiliary attribute video component, subclause B.2.6 is invoked.

NOTE For checking conformance point B it is expected to follow the order of processes as described in this subclause, although the output of this process can also be achieved by a different combination of processes or variations of the processes described in this subclause.

B.2.2 Occupancy nominal format conversion

This process converts the decoded occupancy frames, DecOccFrames, to the nominal format.

Let the variables occNumComp, occMSBAlignFlag and occThreshold be set as follows:

```
occNumComp = 1
occMSBAlignFlag = oi_occupancy_MSB_align_flag[ ConvAtlasID ]
occThreshold = oi_lossy_occupancy_compression_threshold[ ConvAtlasID ]
```

Output of this process is:

— the 4D array OccFramesNF, indicating the decoded occupancy frames in the nominal format, where the dimensions correspond to the occupancy video frame index, the component index, the row index and the column index, respectively.

To convert the decoded occupancy video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

— To convert the bit depth of the decoded occupancy video frames to the required nominal bit depth, subclause B.3.2 is invoked for each frame index frameIdx, with variables DecOccBitDepth[frameIdx], DecOccHeight[frameIdx], DecOccWidth[frameIdx], OccBitDepthNF, and occMSBAlignFlag, and array DecOccFrames[frameIdx][0] as inputs. The output of this process is the decoded occupancy frame at the nominal bit depth, decOccFramesNBD[frameIdx][0].

— To convert the occupancy frames to the nominal frame resolution, subclause B.3.3 is invoked for each frame index frameIdx, with variables DecOccHeight[frameIdx], DecOccWidth[frameIdx], VideoHeightNF, and VideoWidthNF, and array decOccFramesNBD[frameIdx][0] as inputs. The output of this process is the decoded occupancy frame at the nominal resolution, decOccFramesNR[frameIdx][0].

— To identify missing frames in the sequence and re-order the input, subclause B.3.4.1 is invoked with variables NumDecOccFrames, occNumComp, VideoHeightNF, and VideoWidthNF, and arrays decOccFramesNR, DecOccOutOrdIdx, and DecOccCompTime as inputs. The outputs of this process are the number of frames being output, numOutOrdOccFrames, the decoded occupancy frames ordered according to the output order index, decOccFramesONF, a 1D array, framePresent, indicating the presence of a frame, and a 1D array, decOccCompTimeONF, indicating the composition time of each frame ordered according to the output order index.

— To align the composition times of the decoded occupancy frames in output order and the decoded atlas frames, subclause B.3.4.2 is invoked with variables NumDecAtlasFrames, numOutOrdOccFrames, occNumComp, VideoHeightNF, and VideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decOccFramesONF, decOccCompTimeONF, and framePresent as inputs. The outputs of this process are the number of frames being output, rangeCompTimeIdx, the output composition time of each frame, occCompTime, and a sequence of frames in the nominal format aligned with the decoded atlas composition time, OccFramesNF.

— If asps_eom_patch_enabled_flag is equal to 0, to convert the sample values in the decoded occupancy frames to binary values, subclause B.3.9 is invoked for each frame index frameIdx, with variables occThreshold, OccBitDepthNF, VideoHeightNF, and VideoWidthNF, and array OccFramesNF[frameIdx][0] as inputs. The output of this process is the updated occupancy frame, OccFramesNF[frameIdx][0], containing binary values only.

NOTE In an alternative implementation, a decoder can perform the thresholding process before the occupancy is upscaled to the nominal resolution.

Given OccFramesNF, the variable NumProjPoints is derived as follows:

```
NumProjPoints = 0
for ( y = 0; y < asps_frame_height; y++ )
    for( x = 0; x < asps_frame_width; x++ )
        if( OccFramesNF[ compTimeIdx ][ 0 ][ y ][ x ] != 0 )
            NumProjPoints++

projectedPointsScale = asps_map_count_minus1 + 1
for ( i = 0; i < asps_map_count_minus1+1; i++ ) {
    addPoints = ( asps_map_pixel_deinterleaving_flag[ i ] !=0 ) || ( plri_map_present_flag[ i ] !=0 )
    projectedPointsScale += addPoints
}
NumProjPoints = NumProjPoints * projectedPointsScale
```

B.2.3 Geometry nominal format conversion

This process converts the decoded geometry frames, DecGeoFrames, to the nominal format.

Let the variables geoNumComp, geoMultipleMapsPresentFlag, geoMapCount and geoMSBAlignFlag, be set as follows:

```
geoNumComp = 1
geoMultipleMapsPresentFlag = vps_multiple_map_streams_present_flag[ ConvAtlasID ]
geoMapCount = vps_map_count_minus1[ ConvAtlasID ] + 1
geoMSBAlignFlag = gi_geometry_MSB_align_flag[ ConvAtlasID ]
```

Let the 1D array geoMapAbsCodingFlag be set as follows:

```
for( c = 0; c < geoMapCount; c++ ) {
    geoMapAbsCodingFlag[ c ] = vps_map_absolute_coding_enabled_flag[ ConvAtlasID ][ c ]
}
```

Output of this process is:

— the 5D array GeoFramesNF, the decoded geometry frames in the nominal format, where the dimensions correspond to the map index, the geometry video frame index, the components, the column index and the row index of the frames.

NOTE The order of processes or variations of the processes described in this annex is a suggestion. The maps are first separated and then converted. The conversion can happen before the map separation, right after decoding.

To convert the decoded geometry video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

— To convert the bit depth of the geometry frame to the required nominal bit depth resolution, subclause B.3.2 is invoked for each map index, mapIdx, and each frame index, frameIdx, with variables DecGeoBitDepth[mapIdx][frameIdx], DecGeoHeight[mapIdx][frameIdx], DecGeoWidth[mapIdx][frameIdx], GeoBitDepthNF, and geoMSBAlignFlag, and array DecGeoFrames[mapIdx][frameIdx][0] as inputs. Output of this process is the decoded geometry frame at the nominal bit depth, decGeoFramesNBD[mapIdx][frameIdx][0].

— To convert the frame resolution of the geometry frame to the required nominal frame resolution, subclause B.3.3 is invoked for each map index, mapIdx, and each frame index, frameIdx, with variables DecGeoHeight[mapIdx][frameIdx], DecGeoWidth[mapIdx][frameIdx], VideoHeightNF, and VideoWidthNF, and array decGeoFramesNBD[mapIdx][frameIdx][0] as inputs. Output of this process is the decoded geometry frame at the nominal resolution, decGeoFramesNR[mapIdx][frameIdx][0].

— To extract the maps, subclause B.3.1 is invoked, with variables geoMultipleMapsPresentFlag, geoMapCount, geoNumComp, VideoHeightNF, and VideoWidthNF, and arrays NumDecGeoFrames, decGeoFramesNR, DecGeoOutOrdIdx, and DecGeoCompTime as inputs. Outputs of this process are the number of frames being output, rangeOutOrdIdx, the decoded nominal resolution geometry frames ordered according to the output order index, decGeoFramesONF, a 1D array, framePresent, indicating if a frame is present, and a 1D array, decGeoCompTime, indicating the composition time of each frame ordered according to the output order index.

— To align the composition times of the decoded geometry and the decoded atlas frames, subclause B.3.4.2 is invoked for each map index, mapIdx, with variables NumDecAtlasFrames, rangeOutOrdIdx[mapIdx], geoNumComp, VideoHeightNF, and VideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decGeoFramesONF[mapIdx], decGeoCompTimeONF[mapIdx], and framePresent[mapIdx] as inputs. The outputs of this process are the number of frames being output, rangeCompTimeIdx, the output composition time of each frame, geoCompTime[mapIdx], and a sequence of frames in the nominal format aligned with the decoded atlas composition time, GeoFramesNF[mapIdx].

— To synthesize the maps of the geometry, subclause B.3.7 is invoked with variables geoMultipleMapsPresentFlag, geoMapCount, rangeCompTimeIdx, GeoBitDepthNF, geoNumComp, VideoHeightNF, and VideoWidthNF, and arrays GeoFramesNF and geoMapAbsCodingFlag as inputs. The output of this process is the updated decoded geometry frame in the nominal format, GeoFramesNF, containing synthesized geometry values.

B.2.4 Auxiliary geometry nominal format conversion

This process converts the decoded auxiliary geometry frames, DecGeoAuxFrames, to the nominal format.

Let the variables geoAuxNumComp and geoMSBAlignFlag, be set as follows:

geoAuxNumComp = 1
geoMSBAlignFlag = gi_geometry_MSB_align_flag[ConvAtlasID]

Output of this process is:

— the 4D array GeoAuxFramesNF, the decoded geometry frames in the nominal format, where the dimensions correspond to the geometry video frame index, the components, the column index and the row index of the frames.

NOTE The output of this process can also be achieved by a different combination of processes or variations of the processes described in this annex. The specification and order described in this annex is a suggestion.

To convert the decoded auxiliary geometry video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

— To convert the bit depth of the auxiliary geometry to the required nominal bit depth resolution, subclause B.3.2 is invoked for each frame index frameIdx, with variables DecGeoAuxBitDepth[frameIdx], DecGeoAuxHeight[frameIdx], DecGeoAuxWidth[frameIdx], GeoBitDepthNF, and geoMSBAlignFlag, and array DecGeoAuxFrames[frameIdx][0] as inputs. The output of this process is the decoded auxiliary geometry frame at the nominal bit depth, decGeoAuxFramesNBD[frameIdx][0].

— To convert the frame resolution of the auxiliary geometry to the required nominal frame resolution, subclause B.3.3 is invoked for each frame index frameIdx, with variables DecGeoAuxHeight[frameIdx], DecGeoAuxWidth[frameIdx], AuxVideoWidthNF, and AuxVideoHeightNF, and array decGeoAuxFramesNBD[frameIdx][0] as inputs. The output of this process is the decoded auxiliary geometry frame at the nominal resolution, decGeoAuxFramesNR[frameIdx][0].

— To convert the auxiliary geometry frame order defined by frameIdx, to the required output order defined by ordIdx, subclause B.3.4.1 is invoked with variables NumDecGeoAuxFrames, geoAuxNumComp, AuxVideoHeightNF, and AuxVideoWidthNF, and arrays decGeoAuxFramesNR, DecGeoAuxOutOrdIdx, and DecGeoAuxCompTime as inputs. The outputs of this process are the number of frames being output, numOutOrdGeoAuxFrames, the decoded auxiliary geometry frames ordered according to the output order index, decGeoAuxFramesONF, a 1D array, framePresent, indicating the presence of a frame, and a 1D array, decGeoAuxCompTimeONF, indicating the composition time of each frame ordered according to the output order index.

— To align the composition times of the decoded auxiliary geometry frames in output order and the decoded atlas frames, subclause B.3.4.2 is invoked with variables NumDecAtlasFrames, numOutOrdGeoAuxFrames, geoAuxNumComp, AuxVideoHeightNF, and AuxVideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decGeoAuxFramesONF, decGeoAuxCompTimeONF, and framePresent as inputs. The outputs of this process are the number of frames, rangeCompTimeIdx, the output composition time of each frame, auxGeoCompTime, and a sequence of frames in the nominal format aligned with the decoded atlas composition time, GeoAuxFramesNF.

B.2.5 Attribute nominal format conversion

This process converts the decoded attribute frames DecAttrFrames to the nominal format.

The processes described below are invoked for decoded attribute frames associated with the same attribute index, identified by variable attrIdx, which is set equal to vuh_attribute_index or determined through external means if the V3C unit header is unavailable.

Let the variables, attrMultipleMapsPresentFlag, attrMapCount, attrMSBAlignFlag, attrDim, and attrNumPart be set as follows:

```
attrMultipleMapsPresentFlag = vps_multiple_map_streams_present_flag[ ConvAtlasID ]
attrMapCount = vps_map_count_minus1[ ConvAtlasID ] + 1
attrMSBAlignFlag = ai_attribute_MSB_align_flag[ ConvAtlasID ][ attrIdx ]
attrDim = ai_attribute_dimension_minus1[ ConvAtlasID ][ attrIdx ] + 1
attrNumPart = ai_attribute_dimension_partitions_minus1[ ConvAtlasID ][ attrIdx ] + 1
```

Let the 1D array attrPartChannelsMinus1, of size attrNumPart, be set as follows:

```
for( c = 0; c < attrNumPart; c++ ) {
    attrPartChannelsMinus1[ c ] =
        ai_attribute_partition_channels_minus1[ ConvAtlasID ][ attrIdx ][ c ]
}
```

Let the 1D array attrMapAbsCodingFlag, of size attrMapCount, be set as follows:

```
for( c = 0; c < attrMapCount; c++ ) {
    attrMapAbsCodingFlag[ c ] =
        AttributeMapAbsoluteCodingEnabledFlag[ ConvAtlasID ][ attrIdx ][ c ]
}
```

Output of this process is:

— the 7D array AttrAuxFramesNF, the decoded attribute frames in the nominal format, where the dimensions correspond to the attribute index, the attribute partition index, the map index, the decoded attribute video frame index, the component index, the row index and the column index, respectively.

NOTE The order of processes or variations of the processes described in this annex is a suggestion. The maps are first separated and then converted. The conversion can happen before the map separation, right after decoding.

To convert the decoded attribute video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

— To convert the bit depth of the attribute frame to the required nominal bit depth resolution, subclause B.3.2 is invoked for each attribute partition index, partIdx, each map index, mapIdx, each frame index, frameIdx, and each component index, compIdx, with variables DecAttrBitDepth[attrIdx][partIdx][mapIdx][frameIdx], DecAttrHeight[attrIdx][partIdx][mapIdx][frameIdx], DecAttrWidth[attrIdx][partIdx][mapIdx][frameIdx], AttrBitDepthNF[attrIdx], and attrMSBAlignFlag, and array DecAttrFrames[attrIdx][partIdx][mapIdx][frameIdx][compIdx] as inputs. Output of this process is the decoded attribute frame at the nominal bit depth, decAttrFramesNBD[attrIdx][partIdx][mapIdx][frameIdx][compIdx].

— To convert the chroma format of the attribute frame with attributes type DecAttrType[attrIdx] equal to ATTR_TEXTURE, to the required nominal chroma format, subclause B.3.6 is invoked for each attribute partition index, partIdx, each map index, mapIdx, and each frame index, frameIdx, with variables DecAttrHeight[attrIdx][partIdx][mapIdx][frameIdx], DecAttrWidth[attrIdx][partIdx][mapIdx][frameIdx], AttrBitDepthNF[attrIdx], DecAttrChromaFormat[attrIdx][partIdx][mapIdx][frameIdx], and DecAttrChromaSamplingPosition[attrIdx][partIdx][mapIdx][frameIdx], and array decAttrFramesNBD[attrIdx][partIdx][mapIdx][frameIdx] as inputs. The output of this process is the decoded attribute frame with a 4:4:4 colour format, decAttrFramesNCF[attrIdx][mapIdx][frameIdx]

— To convert the frame resolution of the attribute frame to the required nominal frame resolution, subclause B.3.3 is invoked for each attribute partition index, partIdx, each map index, mapIdx, each frame index frameIdx, and each component index, compIdx, with variables DecAttrHeight[attrIdx][partIdx][mapIdx][frameIdx], DecAttrWidth[attrIdx][partIdx][mapIdx][frameIdx], VideoHeightNF, and VideoWidthNF, and array decAttrFramesNCF[attrIdx][partIdx][mapIdx][frameIdx][compIdx] as inputs. Output of this process is the decoded attribute frame with nominal resolution, decAttrFramesNR[attrIdx][partIdx][mapIdx][frameIdx][compIdx].

— To extract the maps, subclause B.3.1 is invoked for each attribute partition index, partIdx, with variables attrMultipleMapsPresentFlag, attrMapCount, DecAttrNumComp[attrIdx][partIdx], VideoHeightNF, and VideoWidthNF, and arrays NumDecAttrFrames[attrIdx][partIdx], decAttrFramesNR[attrIdx][partIdx], DecAttrOutOrderIdx[attrIdx][partIdx], and DecAttrCompTime[attrIdx][partIdx] as inputs. Outputs of this process are the number of frames being output, rangeOutOrdIdx[attrIdx][partIdx], the decoded attribute frames ordered according to the output order index, decAttrFramesONF[attrIdx][partIdx], a 1D array, framePresent[attrIdx][partIdx], indicating if a frame is present, and a 1D array, decAttrCompTime[attrIdx][partIdx], indicating the composition time of each frame ordered according to the output order index.

— To align the composition times of the decoded attributes and the decoded atlas frames, subclause B.3.4.2 is invoked for each attribute partition index, partIdx, and each map index, mapIdx, with variables NumDecAtlasFrames, rangeOutOrdIdx[attrIdx][partIdx][mapIdx], DecAttrNumComp[attrIdx][partIdx][mapIdx], VideoHeightNF, and VideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decAttrFramesONF[attrIdx][partIdx][mapIdx], decAttrCompTimeONF[attrIdx][partIdx][mapIdx], and framePresent[attrIdx][partIdx][mapIdx] as inputs. The outputs of this process are the number of frames being output, rangeCompTimeIdx[attrIdx], the output composition time of each frame, attrCompTime[attrIdx][partIdx][mapIdx], and a sequence of frames in the nominal format aligned with the decoded atlas composition time, decAttrFramesANF[attrIdx][partIdx][mapIdx].

— To perform dimension packing of attributes, subclause B.3.5.1 is invoked, with variables attrNumPart, attrMapCount, rangeCompTimeIdx[attrIdx], (attrDim-(attrNumPart-1)), VideoHeightNF, VideoWidthNF, and attrDim, and arrays decAttrFramesANF[attrIdx], and attrPartChannelsMinus1 as input. The output of this process is a sequence of attribute frames with merged partitions at the nominal format, AttrFramesNF[attrIdx].

— To synthesize the maps of the attribute, subclause B.3.8 is invoked with variables attrMultipleMapsPresentFlag, attrMapCount, rangeCompTimeIdx[attrIdx], AttrBitDepthNF[attrIdx], attrDim[attrIdx], VideoHeightNF, and VideoWidthNF, and arrays AttrFramesNF[attrIdx] and AttributeMapAbsoluteCodingEnabledFlag[ConvAtlasID][attrIdx] as inputs. The output of this process is the updated decoded attribute frame in the nominal format, AttrFramesNF[attrIdx], containing synthesized attribute values.

B.2.6 Auxiliary attribute nominal format conversion

This process converts the decoded auxiliary attribute frames DecAttrAuxFrames to the nominal format.

The processes described below are invoked for decoded auxiliary attribute frames associated with the same attribute index, identified by variable attrIdx, which is set equal to vuh_attribute_index or determined through external means if the V3C unit header is unavailable.

Let the variable attrMSBAlignFlag, attrDim and attrNumPart be set as follows:

```
attrMSBAlignFlag = ai_attribute_MSB_align_flag[ ConvAtlasID ][ attrIdx ]
attrDim = ai_attribute_dimension_minus1[ ConvAtlasID ][ attrIdx ] +1
attrNumPart = ai_attribute_dimension_partitions_minus1[ ConvAtlasID ][ attrIdx ] +1
```

Let the 1D array attrPartChannelsMinus1, of size attrNumPart, be set as follows:

```
for( c = 0; c < attrNumPart; c++ ) {
    attrPartChannelsMinus1[ c ] =
        ai_attribute_partition_channels_minus1[ ConvAtlasID ][ attrIdx ][ c ]
}
```

Output of this process is:

— the 6D array AttrAuxFramesNF, the decoded auxiliary attribute frames in the nominal format, where the dimensions correspond to the attribute index, the attribute partition index, the decoded attribute video frame index, the component index, the row index and the column index, respectively.

NOTE 1 The output of this process can also be achieved by a different combination of processes or variations of the processes described in this annex. The specification and order described in this annex is a suggestion.

To convert the decoded auxiliary attribute video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

— To convert the bit depth of the decoded auxiliary attribute video frames to the required nominal bit depth, subclause B.3.2 is invoked for each partition index partIdx, each frame index frameIdx, and each component index compIdx, with variables DecAttrAuxBitDepth[attrIdx][partIdx][frameIdx], DecAttrAuxHeight[attrIdx][partIdx][frameIdx], DecAttrAuxWidth[attrIdx][partIdx][frameIdx], AttrBitDepthNF[attrIdx], and attrMSBAlignFlag, and array DecAttrAuxFrames[attrIdx][partIdx][frameIdx][compIdx] as inputs. The output of this process is the decoded auxiliary attribute frame at the nominal bit depth, decAttrAuxFrameNBD[attrIdx][partIdx][frameIdx][frameIdx].

— To convert the chroma format of the auxiliary attribute frame with attribute type DecAttrAuxType[attrIdx] equal to ATTR_TEXTURE, to the required nominal chroma format, subclause B.3.6 is invoked for each attribute partition index, partIdx, and each frame index, frameIdx, with variables DecAttrAuxHeight[attrIdx][partIdx][frameIdx], DecAttrAuxWidth[attrIdx][partIdx][frameIdx], AttrBitDepthNF[attrIdx], DecAttrAuxChromaFormat[attrIdx][partIdx][frameIdx], and DecAttrAuxChromaSamplingPosition[attrIdx][partIdx][frameIdx], and array decAttrAuxFrameNBD[attrIdx][partIdx][frameIdx] as inputs. The output of this process is a decoded auxiliary attribute frame with 4:4:4 colour format, decAttrAuxFrameNCF[attrIdx][mapIdx][compTimeIdx].

NOTE 2 When asps_pixel_deinterleaving_enabled_flag is equal to 1, it is possible that the chroma format conversion process described in subclause B.3.6 will not work as intended or produce the desired visual quality.

— To convert the frame resolution of the auxiliary attribute to the required nominal frame resolution, subclause B.3.3 is invoked for each partition index, partIdx, each frame index, frameIdx, and each component index, compIdx with variables DecAttrAuxHeight[attrIdx][partIdx][frameIdx], DecAttrAuxWidth[attrIdx][partIdx][frameIdx], AuxVideoHeightNF, and AuxVideoWidthNF, and array decAttrAuxFrameNCF[attrIdx][partIdx][frameIdx][compIdx] as inputs. The output of this process is the decoded auxiliary attribute frame at the nominal resolution, decAttrAuxFrameNR[attrIdx][partIdx][frameIdx][compIdx].

— To identify missing frames in the sequence and re-order the input, subclause B.3.4.1 is invoked for each partition index partIdx, with variables NumDecAttrAuxFrames[attrIdx][partIdx], DecAttrAuxNumComp[attrIdx][partIdx], AuxVideoWidthNF, and AuxVideoHeightNF, and arrays decAttrAuxFrameNR, DecAttrAuxOutOrdIdx[attrIdx][partIdx], and DecAttrAuxCompTime[attrIdx][partIdx] as inputs. The outputs of this process are the number of frames being output, numOutOrdAttrAuxFrames[attrIdx][partIdx], the decoded auxiliary attribute frames ordered according to the output order index, decAttrAuxFrameONF[attrIdx][partIdx], a 1D array, framePresent[attrIdx][partIdx], indicating the presence of a frame, and a 1D array, decAttrAuxCompTimeONF[attrIdx][partIdx], indicating the composition times of each frame ordered according to the output order index.

— To align the composition times of the decoded auxiliary attribute and the decoded atlas frames, subclause B.3.4.2 is invoked for each partition index partIdx with variables NumDecAtlasFrames, numOutOrdAttrAuxFrames[attrIdx][partIdx], AuxVideoHeightNF, and AuxVideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decAttrAuxFrameONF[attrIdx][partIdx], decAttrAuxCompTimeONF[attrIdx][partIdx], and framePresent[attrIdx][partIdx] as inputs. The outputs of this process are the number of frames being output, rangeCompTimeIdx[attrIdx], the output composition time of each frame, auxAttrCompTime[attrIdx][partIdx], and a sequence of frames in the nominal format aligned with the decoded atlas composition time, decAttrAuxFrameANF[attrIdx][partIdx].

— To perform dimension packing of auxiliary attributes, subclause B.3.5.2 is invoked with variables attrNumPart, rangeCompTimeIdx[attrIdx], (attrDim-(attrNumPart-1)), VideoHeightNF, VideoWidthNF, and attrDim, and arrays decAttrAuxFrameANF[attrIdx] and attrPartChannels as input. The output of this process is a decoded auxiliary attribute frame with merged partitions, AttrAuxFramesNF[attrIdx].

B.3 Conversion operations

B.3.1 Map Extraction

Inputs to this process are:

— a variable multipleMapStreamsPresentFlag, indicating the presence of multiple streams,

— a variable mapCount, indicating the number of maps

— a variable iNumComp, indicating the number of components for each frame,

— a variable iHeight, indicating the height of each frame,

— a variable iWidth, indicating the width of each frame,

— a 1D array numFrames, of size mapCount, indicating the number of input frames,

— a 5D array inputFrame of size mapCount × numInputFrames × iNumComp × iHeight × iWidth,

— a 2D array iOutputOrderIdx, of size mapCount × numInputFrames, indicating the output order index, and

— a 2D array iCompTime, of size mapCount × numInputFrames , indicating the composition time,

where numInputFrames is the maximum value of the numFrames array.

Outputs of this process are:

— a 1D array rangeOrdIdx, of size mapCount, indicating the number of frames of each map,

— a 5D array outputSamples, of size mapCount × numOutputFrames × iNumComp × iHeight × iWidth,

— a 2D array framePresent, of size mapCount × numOutputFrames, indicating the presence of a frame,

— a 2D array outputCompTime, of size mapCount × numOutputFrames, indicating the composition time of each frame,

where numOutputFrames is the maximum value of the array rangeOrdIdx.

The map extraction process is performed as follows:

```
for( i = 0; i < mapCount; i++ )
    rangeOrdIdx[ i ] = 0
if( mapCount – 1 == 0 ) {
    for( f = 0; f < numFrames[ 0 ]; f++ ) {
        g = iOutputOrderIdx[ 0 ][ f ]
        for( i = rangeOrdIdx[ 0 ]; i < g; i++ )
            framePresent[ 0 ][ i ] = 0
        framePresent[ 0 ][ g ] = 1
```

```
            rangeOrdIdx[ 0 ] = g + 1
            outputCompTime[ 0 ][ g ] = inputCompTime[ 0 ][ f ]
            for( c = 0; c < iNumComp; c++ )
                for( y = 0; y < iHeight; y++ )
                    for( x = 0; x < iWidth; x++ )
                        outputSamples[ 0 ][ g ][ c ][ y ][ x ] = inputSamples[ 0 ][ f ][ c ][ y ][ x ]
        }
    } else {
        if( multipleMapStreamsPresentFlag == 0 ) {
            for( f = 0; f < numFrames[ 0 ]; f++ ) {
                g = iOutputOrderIdx[ 0 ][ f ] / ( vps_map_count_minus1[ ConvAtlasID ] + 1 )
                m = iOutputOrderIdx[ 0 ][ f ] % ( vps_map_count_minus1[ ConvAtlasID ] + 1 )
                for( i = rangeOrdIdx[ m ]; i < g; i++ )
                    framePresent[ m ][ i ] = 0
                framePresent[ m ][ g ] = 1
                rangeOrdIdx[ m ] = g + 1
                outputCompTime[ m ][ g ] = iCompTime[ 0 ][ f ]
                for( c = 0; c < iNumComp; c++ )
                    for( y = 0; y < iHeight; y++ )
                        for( x = 0; x < iWidth; x++ )
                            outputSamples[ m ][ g ][ c ][ y ][ x ] =
                                    inputSamples[0][ f ][ c ][ y ][ x ]
            }
        } else {
            for( m = 0; m <= vps_map_count_minus1[ ConvAtlasID ]; m++ ) {
                for( f = 0; f < numFrames[ m ]; f++ ) {
                    g = inputOrderIdx[ m ][ f ]
                    for( i = rangeOrdIdx[ m ]; i < g; i++ )
                        framePresent[ m ][ i ] = 0
                    framePresent[ m ][ g ] = 1
                    rangeOrdIdx[ m ] = g + 1
                    outputCompTime[ m ][ g ] = iCompTime[ i ][ f ]
                    for( c = 0; c < iNumComp; c++ )
                        for( y = 0; y < iHeight; y++ )
                            for( x = 0; x < iWidth; x++ )
                                outputSamples[ m ][ g ][ c ][ y ][ x ] =
                                        inputSamples[ m ][ f ][ c ][ y ][ x ]
                }
            }
        }
    }
```

B.3.2 Bit depth conversion

Inputs to this process are:

— a variable inputBitDepth, indicating the bit depth of inputSamples,

— a variable inputHeight, indicating the height of inputSamples,

— a variable inputWidth, indicating the width of inputSamples,

— a variable nominalBitDepth, indicating the nominal bit depth,

— a variable alignmentFlag, indicating the MSB alignment, and

— a 2D array inputSamples, of size inputHeight × inputWidth.

Output of this process is:

— a 2D array outputSamples of size inputWidth × inputHeight.

The variable bitDepthDifference is derived as follows:

— bitDepthDifference = inputBitDepth – nominalBitDepth

Then the array outputSamples is generated in the following manner:

```
if( ( bitDepthDifference > 0 ) && ( alignmentFlag == 1 ) ) {
    for( y=0; y < inputHeight; y++ )
        for( x=0; x < inputWidth; x++ )
            outputSamples[ y ][ x ] = ( inputSamples[ y ][ x ] >> bitDepthDifference)
}
else if( ( bitDepthDifference > 0 ) ) {
    for( y=0; y < inputHeight; y++ )
        for( x=0; x < inputWidth; x++ )
            outputSamples[ y ][ x ] = Min( inputSamples[ y ][ x ], ( 1 << nominalBitDepth ) – 1 )
} else if( alignmentFlag == 1 ) {
    for( y=0; y < inputHeight; y++ )
        for( x=0; x < inputWidth; x++ )
            outputSamples[ y ][ x ] = ( inputSamples[ y ][ x ] << (-bitDepthDifference) )
} else {
    for( y=0; y < inputHeight; y++ )
        for( x=0; x < inputWidth; x++ )
            outputSamples[ y ][ x ] = inputSamples[ y ][ x ]
}
```

B.3.3 Resolution conversion

Inputs to this process are:

— a variable inputHeight, indicating the height of inputSamples

— a variable inputWidth, indicating the width of inputSamples

— a variable nominalHeight, indicating the nominal height, and

— a variable nominalWidth, indicating the nominal width,

— a 2D array inputSamples, of size inputHeight × inputWidth.

Output of this process is:

— a 2D array outputSamples of size nominalHeight × nominalWidth.

The variables scaleY and scaleX are derived as follows:

```
scaleY = inputHeight ÷ nominalHeight
scaleX = inputWidth ÷ nominalWidth
```

Then the array outputSamples is generated as follows:

```
for( y=0; y < nominalHeight; y++ )
    for( x=0; x < nominalWidth; x++ )
        outputSamples[ y ][ x ] = inputSamples[ Floor( y * scaleY ) ][ Floor( x * scaleX ) ]
```

NOTE In the above process pixel replication is used. A more sophisticated resampling process can be used. For example, the upsampling process using the filters specified in ISO/IEC 23008-2:2020, H.8.1.4.2.2. Other filters such as Lanczos can also be used.

B.3.4 Output composition time conversion

B.3.4.1 Output order conversion

Inputs to this process are:

— a variable numInputFrames, indicating the number of input frames,

— a variable iNumComp, indicating the number of components for each frame,

— a variable iHeight, indicating the height of each frame,

— a variable iWidth, indicating the width of each frame,

— a 4D array inputFrames, of size numInputFrames × iNumComp × iHeight × iWidth

— a 1D array outOrdIdx, of size numInputFrames, indicating the output order index of each frame, and

— a 1D array iCompTime, of size numInputFrames, indicating the composition time of each frame.

Outputs of this process are:

— a variable rangeOrdIdx, indicating the number of output frames,

— a 4D array outputFrames, of size rangeOrdIdx × iNumComp × iHeight × iWidth

— a 1D array framePresent, of size rangeOrdIdx, indicating the presence of a frame, and

— a 1D array outputCompTime, of size rangeOrdIdx, indicating the composition time of each frame.

This process copies the input frames to an output structure according to their output order index and assigns to them a composition time. It also generates a list identifying any missing frames. The following applies:

```
rangeOrdIdx = 0
for( f = 0; f < numFrames; f++ ) {
    ordIdx = outOrdIdx[ 0 ][ f ]

    for( i = rangeOrdIdx ; i < ordIdx; i++ )
        framePresent[ i ] = 0

    framePresent[ ordIdx ] = 1
    outputCompTime[ ordIdx ] = iCompTime[ f ]
    for( c=0; c < iNumComp; c++ ) {
        for( y=0; y < iHeight; y++ ) {
            for( x=0; x < iWidth; x++ ) {
                outputFrames[ ordIdx ][ c ][ y ][ x ] = inputFrames[ f ][ c ][ y ][ x ]
            }
        }
    }

    rangeOrdIdx = ordIdx + 1
}
```

B.3.4.2 Atlas composition time alignment

Inputs to this process are:

— a variable numAtlFrm, indicating the number of atlas frames,

— a variable numInFrames, indicating the number of input frames,

— a variable iNumComp, indicating the number of components for each frame,

— a variable iHeight, indicating the height of each frame,

— a variable iWidth, indicating the width of each frame,

— a 1D array atlasOutOrdIdx, of size numAtlFrm, indicating the atlas output order index,

— a 1D array atlasCompTime, of size numAtlFrm, indicating the atlas composition time,

— a 4D array inputFrames, of size numInFrames × iNumComp × iHeight × iWidth,

— a 1D array iCompTime, of size numInFrames, indicating the composition time of each frame, and

— a 1D array framePresent, of size numInFrames, indicating the presence of a frame.

Outputs of this process are:

— a variable rangeCompTimeIdx, indicating the number of output frames the number of frames,

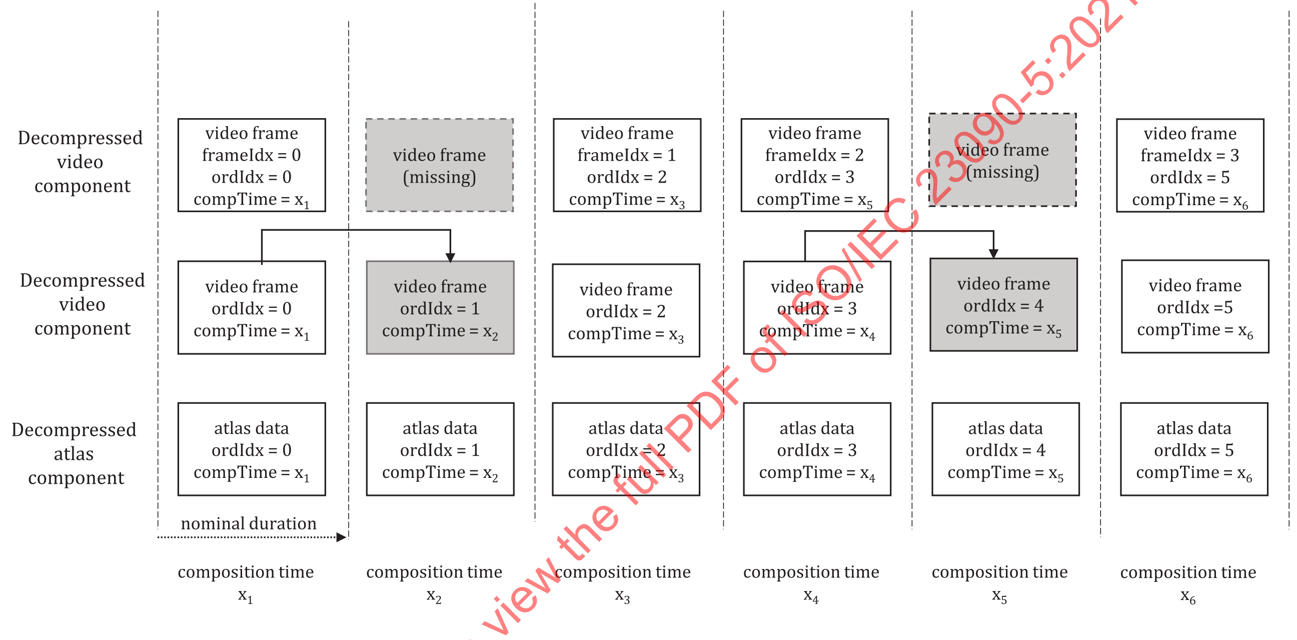
— a 4D array outputFrames, of size rangeCompTimeIdx × iNumComp × iHeight × iWidth,

— a 1D array outputCompTime, of size rangeCompTimeIdx, indicating the composition time of each frame.

This process copies the input frames to an output structure according to the relationship of their composition time to that of the atlas. It then re-assigns to them the composition time of the atlas. In particular, the following applies:

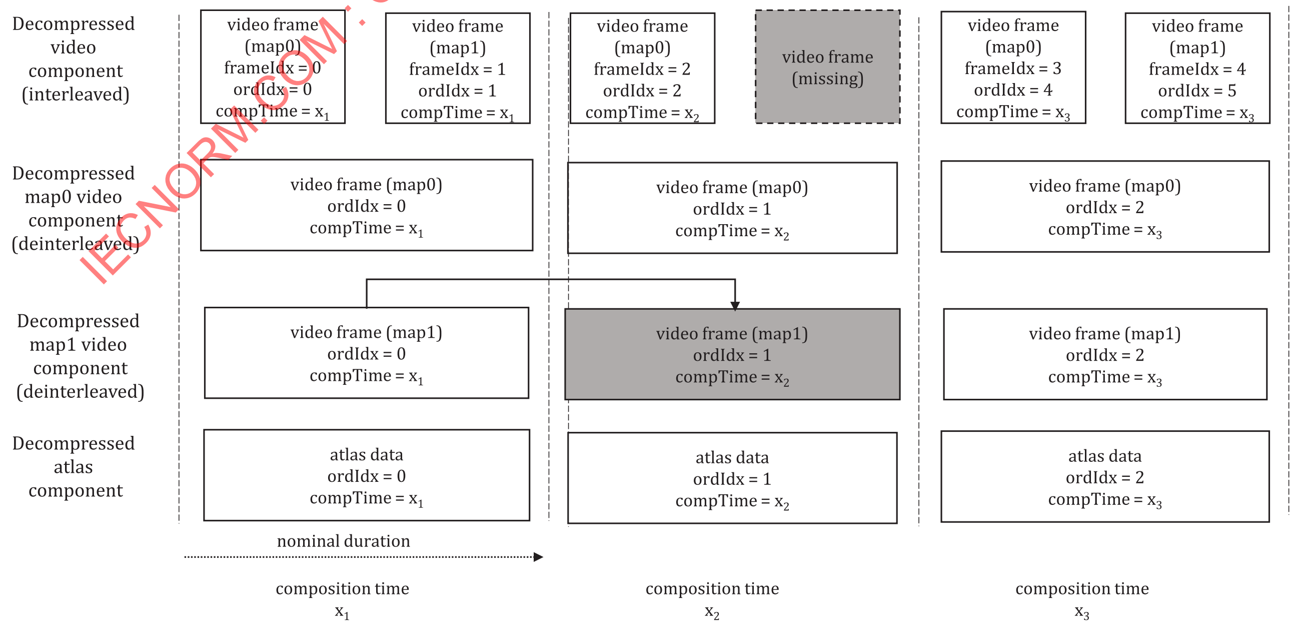
```
rangeCompTimeIdx = 0
inputOrdIdx = 0
for( i=0; i < numAtlFrm; i++ ) {
    compTimeIdx = atlasOutOrdIdx[ i ]
    compTime = atlasCompTime[ i ]

    foundCompositionTime = 0
    while( ( inputOrdIdx <numInFrames ) && ( foundCompositionTime == 0 ) ) {
        skipFrames = 0
        while( ( ( inputOrdIdx+ skipFrames + 1 ) < numInFrames ) &&
            ( framePresent[ inputOrdIdx + skipFrames ] == 0 ) )
            skipFrames = skipFrames + 1

        inputOrdIdx = inputOrdIdx + skipFrames
        if( iCompTime[ inputOrdIdx ] == compTime ) {
            foundCompositionTime = 1
            savedInputOrdIdx = inputOrdIdx
        } else {
            if( iCompTime[ inputOrdIdx ] > compTime )
                foundCompositionTime = 1
            else {
                savedInputOrdIdx = inputOrdIdx
                inputOrdIdx = inputOrdIdx +1
            }
        }
    }

    for( i = rangeCompTimeIdx; i <= compTimeIdx; i++ ) {
        outputCompTime[ i ] = compTime
        for( c=0; c < iNumComp; c++ ) {
            for( y=0; y < iHeight; y++ ) {
                for( x=0; x < iWidth; x++ ) {
                    outputSamples[ i ][ c ][ y ][ x ] =
                        inputSamples[ savedInputOrdIdx ][ c ][ y ][ x ]
                }
            }
        }
    }

    rangeCompTimeIdx = compTimeIdx + 1
}
```

Two examples of alignment of the composition times of the decoded video component are presented in Figure B.2 and Figure B.3.



Decompressed video component
video frame frameIdx = 0 ordIdx = 0 compTime = x_1
video frame (missing)
video frame frameIdx = 1 ordIdx = 2 compTime = x_3
video frame frameIdx = 2 ordIdx = 3 compTime = x_5
video frame (missing)
video frame frameIdx = 3 ordIdx = 5 compTime = x_6
Decompressed video component
video frame ordIdx = 0 compTime = x_1
video frame ordIdx = 1 compTime = x_2
video frame ordIdx = 2 compTime = x_3
video frame ordIdx = 3 compTime = x_4
video frame ordIdx = 4 compTime = x_5
video frame ordIdx =5 compTime = x_6
Decompressed atlas component
atlas data ordIdx = 0 compTime = x_1
atlas data ordIdx = 1 compTime = x_2
atlas data ordIdx = 2 compTime = x_3
atlas data ordIdx = 3 compTime = x_4
atlas data ordIdx = 4 compTime = x_5
atlas data ordIdx = 5 compTime = x_6
nominal duration
composition time x_1
composition time x_2
composition time x_3
composition time x_4
composition time x_5
composition time x_6

Figure B.2 — Example of alignment of the composition times of the decoded video component with different frame rate and the decoded atlas frames



Decompressed video component (interleaved)
video frame (map0) frameIdx = 0 ordIdx = 0 compTime = x_1
video frame (map1) frameIdx = 1 ordIdx = 1 compTime = x_1
video frame (map0) frameIdx = 2 ordIdx = 2 compTime = x_2
video frame (missing)
video frame (map0) frameIdx = 3 ordIdx = 4 compTime = x_3
video frame (map1) frameIdx = 4 ordIdx = 5 compTime = x_3
Decompressed map0 video component (deinterleaved)
video frame (map0) ordIdx = 0 compTime = x_1
video frame (map0) ordIdx = 1 compTime = x_2
video frame (map0) ordIdx = 2 compTime = x_3
Decompressed map1 video component (deinterleaved)
video frame (map1) ordIdx = 0 compTime = x_1
video frame (map1) ordIdx = 1 compTime = x_2
video frame (map1) ordIdx = 2 compTime = x_3
Decompressed atlas component
atlas data ordIdx = 0 compTime = x_1
atlas data ordIdx = 1 compTime = x_2
atlas data ordIdx = 2 compTime = x_3
nominal duration
composition time x_1
composition time x_2
composition time x_3

Figure B.3 — Example of alignment of the composition times of the decoded video component containing pixel interleaved maps and the decoded atlas frames

B.3.5 Attribute dimension packing

B.3.5.1 Attribute dimension packing process

Inputs to this process are:

— a variable iNumPart, indicating the number of dimension partitions in which an attribute has been partitioned,

— a variable iNumMaps, indicating the number of maps,

— a variable iNumFrames, indicating the number of frames,

— a variable iNumComp, indicating the number of components for each frame,

— a variable iHeight, indicating the height of each frame,

— a variable iWidth, indicating the width of each frame,

— a variable iAttrDim, indicating the dimensions of the attribute,

— a 6D array, iFrames, of size iNumPart × iNumMaps × iNumFrames × iNumComp × iHeight × iWidth, and

— a 1D array iNumPartChannelsMinus1, of size iNumPart, indicating the number of channels used in each partition.

Output of this process is:

— a 5D array, oFrames, of size iNumMaps × iNumFrames × iAttrDim × inputHeight × inputWidth.

The array oFrames is generated as follows:

```
outComp = 0
for( m = 0; m < iNumMap; m++) {
    for( f = 0; f < iNumFrames; f++) {
        oCompIdx = 0
        for( p = 0; p < iNumPart; p++) {
            for( c = 0; c < iNumPartChannelsMinus1[ p ] + 1; c++ ) {
                for( h = 0; h < iHeight; h++ ) {
                    for( w = 0; w < iWidth; w++ ) {
                        oFrames[ m ][ f ][ oCompIdx ][ h ][ w ] =
                            iFrames[ p ][ m ][ f ][ c ][ j ][ i ]
                    }
                }
                oCompIdx++
            }
        }
    }
}
```

B.3.5.2 Attribute dimension packing process for an auxiliary attribute

Inputs to this process are:

— a variable iNumPart, indicating the number of dimension partitions in which an attribute has been partitioned,

— a variable iNumFrames, indicating the number of frames,

— a variable iNumComp, indicating the number of components for each frame,

— a variable iHeight, indicating the height of each frame,

— a variable iWidth, indicating the width of each frame,

— a variable iAttrDim, indicating the dimensions of the attribute,

— a 5D array, iFrames, of size iNumPart × iNumFrames × iNumComp × iHeight × iWidth, and

— a 1D array iNumPartChannelsMinus1, of size iNumPart, indicating the number of channels used in each partition.

Output of this process is:

— a 4D array, oFrames, of size iNumFrames × iAttrDim × inputHeight × inputWidth.

The array oFrames is generated as follows:

```
outComp =       0
for( f = 0; f < iNumFrames; f++) {
    oCompIdx = 0
    for( p = 0; p < iNumPart; p++) {
        for( c = 0; c < iNumPartChannelsMinus1[ p ] + 1; c++ ) {
            for( h = 0; h < iHeight; h++ ) {
                for( w = 0; w < iWidth; w++ ) {
                    oFrames[ f ][ oCompIdx ][ h ][ w ] =
                        iFrames[ p ][ f ][ c ][ j ][ i ]
                }
            }
        oCompIdx++
        }
    }
}
```

B.3.6 Chroma up-sampling

This process can be invoked when an attribute with identifier ATTR_TEXTURE has three components with a chroma format indicated by decAttrChromaFormat other than 4:4:4, and the reconstruction system has selected to perform chroma format conversion to a 4:4:4 chroma format.

NOTE 1 This process can also be invoked for an individual partition of an attribute with identifier ATTR_TEXTURE when that partition has three components with a chroma format indicated by decAttrChromaFormat other than 4:4:4.

Inputs to this process are:

— a variable inputHeight, indicating the height of inputChromaSamples,

— a variable inputWidth, indicating the width of inputChromaSamples,

— a variable nominalBitDepth, indicating the chroma bit depth,

— a variable chromaFormat,

— a variable chromaSamplingPosition, and

— a 3D array inputChromaSamples, of size iNumComp × inputHeight × inputWidth.

Output of this process is:

— a 3D array outputChromaSamples, of size iNumComp × inputHeight × inputWidth, of up-sampled chroma values.

Assuming that the chromaFormat indicates the 4:2:0 chroma format and the chromaSamplingPosition is equal to 0, an example chroma conversion process is specified as follows:

Table B.1 specifies the 4-tap filter coefficients $f_C[\ p,\ x\ ]$ with p = 0..1 and x = 0..3 that is used for up-sampling by a factor of 2. The same filter is applied both vertically and horizontally.

Table B.1 — 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	16	0	0
1	–1	9	9	–1

NOTE 2 This is essentially a "Lanczos 2" filter. Higher-precision and higher-order filters could potentially be used, especially when up-sampling content of very high quality or with no compression.

The chroma values are up-sampled as follows:

```
shift = 8
offset = 1 << ( shift − 1 )
for( c = 1; c < 3 ; c++ ) {
    for( y = 0; y < inputHeight ; y++ ) {
        yRef = ( y >> 1 )
        yPhase = y % 2
        for( x = 0; x < inputWidth ; x++ ) {
            xRef = ( x >> 1 )
            xPhase = x % 2
            usChromaSample = 0
            for( n = 0; n < 4; n++ ) {
                yPos = Clip3( 0, ( inputHeight >> 2 ) − 1, yRef + n − 1 )
                temp = 0
                for( m = 0; m < 4; m++ ) {
                    xPos = Clip3( 0, ( inputWidth >> 2 ) − 1, xRef + m − 1 )
                    temp += fC[ xPhase, m ] * inputChromaSample[ c ][ yPos ][ xPos ]
                }
                usChromaSample += $f_c$[ yPhase, 0 ] * temp
            }
            usChromaSample = ( usChromaSample + offset ) >> shift
            outputChromaSample[ c ][ y ][ x ] =
                Clip3( 0, ( 1 << nominalBitDepth ) − 1, usChromaSample )
        }
    }
}
```

NOTE 3 Instead of clipping the chroma output to the nominal bit depth for the attribute, a point cloud reconstruction system can choose to store the attribute in 4:4:4 chroma format at a higher bit depth.

For other chroma formats or chroma sampling positions different processes could be used that are outside the scope of this document.

B.3.7 Geometry map synthesis process

Inputs to this process are:

— a variable multipleMapStreamsPresentFlag, indicating the presence of multiple streams,

— a variable mapCount, indicating the number of maps

— a variable numInputFrames, indicating the number of input frames,

— a variable iBitDepth, indicating the decoded geometry frames bit depth,

— a variable iNumComp, indicating the number of components for each frame,

— a variable iHeight, indicating the height of each frame,

— a variable iWidth, indicating the width of each frame,

— a 5D array iFrames of size mapCount × numInputFrames × iNumComp × iHeight × iWidth, and

— a 1D array mapAbsoluteCodingFlag, of size mapCount.

Output of this process is:

— a 5D array oFrames, of size mapCount × numInputFrames × iNumComp × iHeight × iWidth.

The geometry video map synthesis process is performed as follows:

```
for( f = 0; f < numInputFrames; f++ ) {
    if( multipleMapStreamsPresentFlag == 0 ) {
        for( i = 0; i < mapCount; i++ ) {
            for( c = 0; c < iNumComp ; y++ ) {
                for( y = 0; y < iHeight ; y++ ) {
                    for( x = 0; x < iWidth ; x++ ) {
                        oFrames[ i ][ f ][ c ][ y ][ x ] = iFrames[ i ][ f ][ c ][ y ][ x ]
                    }
                }
            }
        }
    } else {
        geoMaxValue = ( 1 << iBitDepth ) – 1
        for( i = 0; i < mapCount; i++ ) {
            if( mapAbsoluteCodingFlag[ i ] == 0) {
                j = MapPredictorIndex[ i ]
                for( c = 0; c < iNumComp ; y++ ) {
                    for( y = 0; y < iHeight ; y++ ) {
                        for( x = 0; x < iWidth ; x++ ) {
                            oFrames[ i ][ f ][ c ][ y ][ x ] =
                                Clip3( 0, geoMaxValue, iFrames[ i ][ f ][ c ][ y ][ x ] +
                                    oFrames[ j ][ f ][ c ][ y ][ x ] )
                        }
                    }
                }
            } else {
                for( c = 0; c < iNumComp ; y++ ) {
                    for( y = 0; y < iHeight ; y++ ) {
                        for( x = 0; x < iWidth ; x++ ) {
                            oFrames[ i ][ f ][ 0 ][ y ][ x ] = iFrames[ i ][ f ][ 0 ][ y ][ x ]
                        }
                    }
                }
            }
        }
    }
}
```

B.3.8 Attribute map synthesis process

Inputs to this process are:

— a variable multipleMapStreamsPresentFlag, indicating the presence of multiple streams,

— a variable mapCount, indicating the number of maps

— a variable numInputFrames, indicating the number of input frames,

— a variable iBitDepth, indicating the decoded geometry frames bit depth,

— a variable iNumComp, indicating the number of components for each frame,

— a variable iHeight, indicating the height of each frame,

— a variable iWidth, indicating the width of each frame,

— a 5D array iFrames of size mapCount × numInputFrames × iNumComp × iHeight × iWidth, and

— a 1D array mapAbsoluteCodingFlag, of size mapCount.

Outputs of this process are:

— a 5D array oFrames, of size mapCount × numInputFrames × iNumComp × iHeight × iWidth.

The attribute video map synthesis process is performed as follows:

```
for( f = 0; f < numInputFrames; f++ ) {
    if( multipleMapStreamsPresentFlag == 0 ) {
        for( m = 0; m < mapCount; m++ ) {
            for( c = 0; c < iNumComp ; c++ ) {
                for( y = 0; y < iHeight ; y++ ) {
                    for( x = 0; x < iWidth ; x++ ) {
                        oFrames[ m ][ f ][ c ][ y ][ x ] =iFrames[ m ][ f ][ c ][ y ][ x ]
                    }
                }
            }
        }
    } else {
        attrMaxValue = ( 1 << iBitDepth ) – 1
        attrOffset = 1 << (iBitDepth – 1 )
        for( i = 0; i < mapCount; i++ ) {
            if( mapAbsoluteCodingFlag[ i ] == 0 ) {
                j = MapPredictorIndex[ i ]
                for( c = 0; c < iNumComp; c++ )
                    for( y = 0; y < iHeight ; y++ )
                        for( x = 0; x < iWidth ; x++ )
                            oFrames[ i ][ f ][ c ][ y ][ x ] =
                                Clip3( 0, attrMaxValue, – attrOffset +
                                    iFrames[ i ][ f ][ c ][ y ][ x ] +
                                        oFrames[ j ][ f ][ c ][ y ][ x ] )
            } else {
                for( c = 0; c < iNumComp; c++ ) {
                    for( y = 0; y < iHeight ; y++ ) {
                        for( x = 0; x < iWidth ; x++ ) {
                            oFrames[ i ][ f ][ c ][ y ][ x ] = iFrames[ i ][ f ][ c ][ y ][ x ]
                        }
                    }
                }
            }
        }
    }
}
```

B.3.9 Occupancy thresholding process

Inputs to this process are:

— a variable inputThreshold, indicating the threshold value,

— a variable inputBitDepth, indicating the bit depth of inputSamples,

— a variable inputHeight, indicating the height of inputSamples,

— a variable inputWidth, indicating the width of inputSamples, and

— a 2D array inputSamples, of size inputHeight × inputWidth.

Output of this process is:

— a 2D array outputSamples, of size inputHeight × inputWidth.

The occupancy thresholding process is performed as follows:

```
lossyThreshold = inputThreshold << ( inputBitDepth – 8 )

for( y=0; y < inputHeight; y++ )
    for( x=0; x < inputWidth; x++ )
        if( inputSamples[ y ][ x ] <= lossyThreshold )
            outputSamples[ y ][ x ] = 0
        else
            outputSamples[ y ][ x ] = 1
```

Annex C
(normative)

V3C sample stream format

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 V3C unit stream as an ordered stream of bytes or bits within which the locations of V3C 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 V3C unit syntax structures. Each sample stream V3C unit syntax structure contains one element, named ssvu_v3c_unit_size, that specifies a size, followed by one v3c_unit(ssvu_v3c_unit_size) syntax structure. This v3c_unit is essentially of size ssvu_v3c_unit_size. The number of bits used to encode the element ssvu_v3c_unit_size is specified in the sample stream header.

C.2 Sample stream V3C unit syntax and semantics

C.2.1 Sample stream V3C header syntax

sample_stream_v3c_header( ) {	**Descriptor**
ssvh_unit_size_precision_bytes_minus1	u(3)
ssvh_reserved_zero_5bits	u(5)
}	

C.2.2 Sample stream V3C unit syntax

sample_stream_v3c_unit( ) {	**Descriptor**
ssvu_v3c_unit_size	u(v)
v3c_unit(ssvu_v3c_unit_size)	
}	

C.2.3 Sample stream V3C header semantics

The sample stream V3C header shall always be at the beginning of the sample stream V3C stream.

ssvh_unit_size_precision_bytes_minus1 plus 1 specifies the precision, in bytes, of the ssvu_v3c_unit_size element in all sample stream V3C 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 edition of this document. 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.

C.2.4 Sample stream V3C unit semantics

The order of sample stream V3C units in the sample stream shall follow the decoding order of the V3C units contained in the sample stream V3C units (see subclause 8.4.2.5.3).

ssvu_v3c_unit_size specifies the size, in bytes, of the subsequent v3c_unit. The number of bits used to represent ssvu_v3c_unit_size is equal to (ssvh_unit_size_precision_bytes_minus1 + 1) * 8.

Annex D
(normative)

NAL sample stream format

D.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.

D.2 Sample stream NAL unit syntax and semantics

D.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)
}	

D.2.2 Sample stream NAL unit syntax

sample_stream_nal_unit( ) {	**Descriptor**
ssnu_nal_unit_size	u(v)
nal_unit(ssnu_nal_unit_size)	
}	

D.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 edition of this document. 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.

D.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 (see subclause 8.4.5.3). The content of each sample stream NAL unit is associated with the same coded atlas access unit as the NAL unit contained in the sample stream NAL unit (see subclause 8.4.5.3).

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 E
(normative)

Atlas hypothetical reference decoder

E.1 General

This annex specifies the hypothetical reference decoder (HRD) and its use to check atlas sub-bitstream and decoder conformance. In this section, a bitstream refers to an atlas sub-bitstream unless otherwise indicated.

Two types of bitstreams or bitstream subsets are subject to HRD conformance checking for this document. The first type, called a Type I bitstream, is a NAL unit stream containing the ACL NAL units and non-ACL NAL units with nal_unit_type equal to NAL_FD (filler data NAL units) for all coded atlas access units in the atlas sub-bitstream. The second type, called a Type II bitstream, is a NAL unit stream contains, in addition to the ACL NAL units and and filler data NAL units for all coded atlas access units in the atlas sub-bitstream, at least one of the following:

— additional non-ACL NAL units other than filler data NAL units,

— the sample stream nal header structure syntax elements, ssnh_unit_size_precision_bytes_minus1

and ssnh_reserved_zero_5bits, and all ssnu_nal_unit_size syntax elements that form a NAL sample stream from the NAL unit stream (as specified in Annex D).

Figure E.1 shows the types of bitstream conformance points checked by the atlas sub-bitstream HRD.

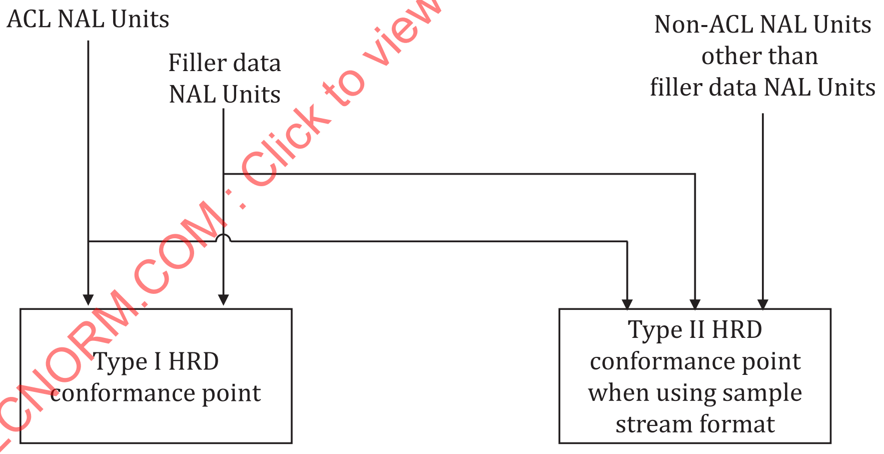

ACL NAL Units
Filler data NAL Units
Non-ACL NAL Units other than filler data NAL Units
Type I HRD conformance point
Type II HRD conformance point when using sample stream format

Figure E.1 — Structure of atlas NAL unit streams for HRD conformance checks

The syntax elements of non-ACL NAL units and their default values, when applicable, required for the HRD, are specified in Clause 8, Annex F and Annex G.

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 specified in Annex G and the buffering period SEI message specified in subclauses F.2.13 and F.3.13. The first set, named as the entireBitstreamTestSet, is to test conformance of the entire bitstream. The second set, named as ACLTestSet, is to test conformance of the bitstream without including non-ACL units since they may not impact the decoding process or may be provided through external means.

Multiple tests may be needed for checking the conformance of an atlas sub-bitstream, AtlasBitstreamToDecode, which is referred to as the bitstream under test. For each test, the following steps apply in the order listed:

a) An operation point under test, denoted as TargetOp, is set by selecting a target highest TemporalID value OpTid. The value of OpTid is in the range of 0 to bp_max_sub_layers_minus1, inclusive. The value of OpTid is such that the sub-bitstream BitstreamToDecode satisfy both of the following conditions:

   1) There is at least one ACL NAL unit in BitstreamToDecode with nal_layer_id equal to 0.

   2) There is at least one ACL NAL unit with TemporalId equal to OpTid in BitstreamToDecode.

b) The value of Htid is set equal to opTid of TargetOp.

c) The hrd_parameters( ) syntax structure in the VUI of the active ASPS (or provided through some external means not specified in this document) that applies to TargetOp is selected.

   Within the selected hrd_parameters( ) syntax structure, if AtlasBitstreamToDecode is a Type I bitstream, the hrd_sub_layer_parameters(1, Htid) syntax structure that immediately follows the condition "if(hrd_acl_parameters_present_flag)" is selected and the variable NalHrdModeFlag is set equal to 0.

   Otherwise (AtlasBitstreamToDecode is a Type II bitstream), when using the ACLTestSet, the variable NalHrdModeFlag is first set to 0. Then the parameters corresponding to the hrd_sub_layer_parameters(1, Htid) syntax structure that immediately follows the condition "if(hrd_acl_parameters_present_flag)" are selected and used. Otherwise, when using the entireBitstreamTestSet, the variable NalHrdModeFlag is set equal to 1 and the parameters corresponding to the hrd_sub_layer_parameters(0, Htid) syntax structure that immediately follows the condition "if(hrd_nal_parameters_present_flag)" are selected and used.

   When AtlasBitstreamToDecode is a Type II bitstream and NalHrdModeFlag is equal to 0, all non-ACL NAL units except filler data NAL units and all the syntax elements that form a sample stream from the NAL unit stream, as specified in Annex D, when present, are discarded from AtlasBitstreamToDecode and the remaining atlas bitstream is assigned to AtlasBitstreamToDecode.

d) A coded atlas access unit associated with a buffering period SEI message, present in AtlasBitstreamToDecode or available through external means not specified in this document, applicable to TargetOp, is selected as the HRD initialization point and referred to as the coded atlas access unit 0.

e) For each coded atlas access unit in AtlasBitstreamToDecode starting from coded atlas access unit 0, the buffering period SEI message and the atlas frame timing SEI message, present in AtlasBitstreamToDecode or available through external means not specified in this document, that are associated with the coded atlas access unit and applies to TargetOp, are selected.

f) A value of SchedSelIdx is selected. The selected value of SchedSelIdx shall be in the range of 0 to hrd_cab_cnt_minus1[Htid], inclusive, where hrd_cab_cnt_minus1[Htid] is found in the buffering period SEI syntax structure, as specified in F.2.13.

g) When the coded atlas frame in access unit 0 has nal_unit_type equal to NAL_CRA, NAL_GCRA, NAL_BLA_W_LP, or 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 the selection of the initial CAB removal delay and delay offset:

   1) If NalHrdModeFlag is equal to 1, the default initial CAB removal delay and the delay offset represented by bp_nal_initial_cab_removal_delay[Htid][SchedSelIdx] and bp_nal_initial_cab_removal_offset[Htid][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[Htid][SchedSelIdx] and bp_acl_initial_cab_removal_

offset[Htid][SchedSelIdx], respectively, in the selected buffering period SEI message are selected. The variable DefaultInitCabParamsFlag is set equal to 1.

2) If NalHrdModeFlag is equal to 1, the alternative initial CAB removal delay and delay offset represented by bp_nal_initial_alt_cab_removal_delay[Htid][SchedSelIdx] and bp_nal_initial_alt_cab_removal_offset[Htid][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[Htid][SchedSelIdx] and bp_acl_initial_alt_cab_removal_offset[Htid][SchedSelIdx], respectively, in the selected buffering period SEI message are selected. The variable DefaultInitCabParamsFlag is set equal to 0, and the RASL coded atlas access units associated with the coded atlas 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[Htid] + 1.

— n2 is the number of coded atlas 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_CRA, NAL_BLA_W_LP, NAL_GCRA, or NAL_GBLA_W_LP 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 coded atlas 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_CRA, NAL_BLA_W_LP, NAL_GCRA, or NAL_GBLA_W_LP, or 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 1.

NOTE 1 n0 corresponds to conformance tests for Type I bitstream conformance and Type II bitstream conformance. n1 corresponds to conformance tests for each CAB delivery schedule. n2 corresponds to conformance tests for bitstreams starting at each CRA coded atlas frames with associated RASL coded atlas frames and alternative initial CAB removal delay and delay offset present. These test are carried twice: once for bitstream keeping and once for bitstreams removing RASL coded atlas associated with the CRA. n3 corresponds to conformance tests for bitstreams starting at each IRAP coded atlas that is not a CRA with associated RASL coded atlas frames and alternative initial CAB removal delay and delay offset present. n4 corresponds to conformance tests for access unit based conformance.

When AtlasBitstreamToDecode is a Type II patch bitstream, the following applies:

— If the hrd_sub_layer_parameters(1, Htid) syntax structure that immediately follows the condition "if(hrd_acl_parameters_present_flag)" is selected, the test is conducted at the Type I conformance point shown in Figure E.1, and only ACL and filler data NAL units are counted for the input bit rate and CAB storage.

— Otherwise (the hrd_sub_layer_parameters(0, Htid) syntax structure that immediately follows the condition "if(hrd_nal_parameters_present_flag)" is selected), the test is conducted at the Type II conformance point shown in Figure E.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 E.1 are sufficient to also establish ACL HRD conformance for the Type I conformance point shown in Figure E.1 for the same values of InitCabRemovalDelay[Htid][SchedSelIdx], BitRate[!NalHrdModeFlag][Htid][SchedSelIdx] and CabSize[!NalHrdModeFlag][Htid][SchedSelIdx] for the variable bit rate (VBR) case (hrd_cbr_flag[!NalHrdModeFlag][Htid][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 frame is scheduled to begin to arrive. For example, when decoding a CAS that is associated with a CVS conforming to one or more of the profiles specified in Annex A using the decoding process specified in subclause 9.2.

All AAPS, ASPS and AFPSs referred to in the ACL NAL units and the corresponding buffering period and atlas frame timing 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 document.

In Annex E and Annex F, the specification for "presence" of non-ACL NAL units that contain AAPSs, ASPSs, AFPSs, buffering period SEI messages, and atlas frame timing SEI messages is also satisfied when those NAL units (or just some of them) are conveyed to the atlas frame decoders (or to the HRD) by other means not specified in this document. For the purpose of counting bits, only the appropriate bits that are actually present in the atlas sub-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 document.

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 subclause 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 is supplied by some other means not specified in this document.

The HRD contains a coded atlas buffer (CAB), an instantaneous decoding process, a decoded atlas buffer (DAB), and the required output widow as shown in Figure E.2. The size of the output window is asps_frame_height × asps_frame_width.

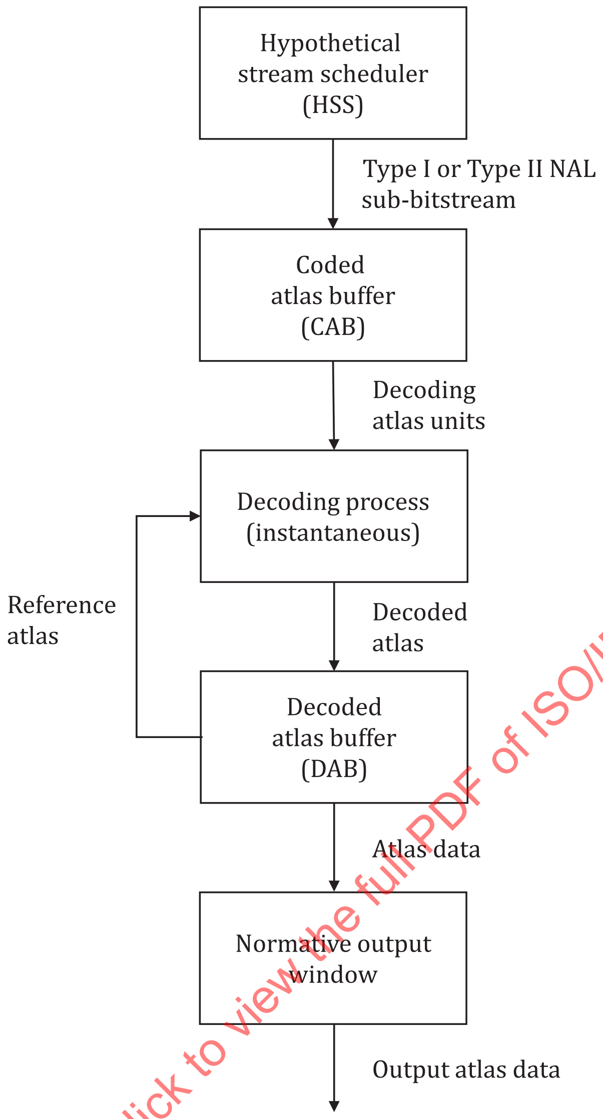

Hypothetical stream scheduler (HSS)
Type I or Type II NAL sub-bitstream
Coded atlas buffer (CAB)
Decoding atlas units
Decoding process (instantaneous)
Reference atlas
Decoded atlas
Decoded atlas buffer (DAB)
Atlas data
Normative output window
Output atlas data

Figure E.2 — Atlas frame HRD buffer model

For each atlas bitstream conformance test, the CAB size (number of bits) is CabSize[!NalHrdModeFlag][Htid][SchedSelIdx] as specified in subclause G.3.3, where SchedSelIdx and the HRD parameters are specified in this subclause. 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 coded atlas access unit is referred to as coded atlas access unit n, where the number n identifies the particular coded atlas access unit. Coded atlas access unit 0 is selected per step 4 as defined in this subclause. The value of n is incremented by 1 for each subsequent coded atlas access unit in decoding order.

— Atlas frame n refers to the coded atlas frame or the decoded atlas frame of coded atlas 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 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 E.2, the instantaneous decoder operation is specified in Clauses 2 through 14, and the operation of the DAB is specified in Clause E.3.

HSS and HRD information concerning the number of enumerated delivery schedules and their associated bit rates and buffer sizes is specified in subclauses G.2.2 and G.3.2. The HRD is initialized as specified by the buffering period SEI message specified in subclauses F.2.13 and F.3.13. The removal timing of decoding units from the CAB and the output timing of decoded atlas frames from the DAB is specified using information in atlas frame timing SEI messages, as specified in subclauses F.2.14 and F.3.14. 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 E.4, and the HRD is used to check conformance of atlas sub-bitstreams as specified in Clause E.1 and to check conformance of decoders as specified in Clause E.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 can 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:

$$\text{ClockTick} = \text{vui\_num\_units\_in\_tick} \div \text{vui\_time\_scale} \qquad \text{(E.1)}$$

E.2 Operation of coded atlas frame buffer

E.2.1 General

The specifications in this subclause 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 E.1. The set of CAB parameters is selected as specified in Clause E.1.

E.2.2 Timing of decoding unit arrival

The decoding unit is considered as a coded atlas access unit, for derivation of the initial and final CAB arrival times for coded atlas access unit n.

The variables InitCabRemovalDelay[Htid][SchedSelIdx] and InitCabRemovalDelayOffset[Htid][SchedSelIdx] are derived as follows:

— If one or more of the following conditions are true, InitCabRemovalDelay[Htid][SchedSelIdx] and InitCabRemovalDelayOffset[Htid][SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements bp_nal_initial_alt_cab_removal_delay[Htid][SchedSelIdx] and bp_nal_initial_alt_cab_removal_offset[Htid][SchedSelIdx], respectively, when NalHrdModeFlag is equal to 1 or bp_acl_initial_alt_cab_removal_delay[Htid][SchedSelIdx] and bp_acl_initial_alt_cab_ removal_offset[Htid][SchedSelIdx], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message syntax elements are selected as specified in Clause E.1:

— Coded atlas access unit 0 is a BLA access unit, for which the coded atlas frame has nal_unit_type equal to NAL_BLA_W_RADL, NAL_BLA_N_LP, NAL_GBLA_W_RADL, or NAL_GBLA_N_LP and the value of bp_irap_cab_params_present_flag of the buffering period SEI message is equal to 1.

— Coded atlas access unit 0 is a BLA access unit, for which the coded picture has nal_unit_type equal to NAL_BLA_W_LP or NAL_GBLA_W_LP, or is a CRA coded atlas 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:

  — UseAltCabParamsFlag for coded atlas access unit 0 is equal to 1.

  — DefaultInitCabParamsFlag is equal to 0.

— Otherwise, InitCabRemovalDelay[Htid][SchedSelIdx] and InitCabRemovalDelayOffset[Htid][SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements bp_nal_initial_cab_removal_delay[Htid][SchedSelIdx] and bp_nal_initial_cab_removal_offset[Htid][SchedSelIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_acl_initial_cab_removal_delay[Htid][SchedSelIdx] and bp_acl_initial_cab_removal_offset[Htid][SchedSelIdx], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message syntax elements are selected as specified in Clause E.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[!NalHrdModeFlag][Htid][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.,

$$\text{initArrivalTime}[\ m\ ] = \text{AuFinalArrivalTime}[\ m - 1\ ] \quad \text{(E.2)}$$

Otherwise (hrd_cbr_flag[!NalHrdModeFlag][Htid][SchedSelIdx] is equal to 0), the initial arrival time for decoding unit m is derived as follows:

$$\text{initArrivalTime}[\ m\ ] = \text{Max}(\ \text{AuFinalArrivalTime}[\ m - 1\ ],\ \text{initArrivalEarliestTime}[\ m\ ]\ ) \quad \text{(E.3)}$$

where initArrivalEarliestTime[m] is derived as follows:

— The variable tmpNominalRemovalTime is derived as follows:

$$\text{tmpNominalRemovalTime} = \text{AuNominalRemovalTime}[\ m\ ] \quad \text{(E.4)}$$

where AuNominalRemovalTime[m] is the nominal CAB removal time of coded atlas access unit m respectively, as specified in subclause E.2.3.

— If decoding unit m is not the first decoding unit of a subsequent buffering period, initArrivalEarliestTime[m] is derived as follows:

$$\text{initArrivalEarliestTime}[\ m\ ] = \text{tmpNominalRemovalTime} - (\ \text{InitCabRemovalDelay}[\ \text{Htid}\ ][\ \text{SchedSelIdx}\ ] + \text{InitCabRemovalDelayOffset}[\ \text{Htid}\ ][\ \text{SchedSelIdx}\ ]\ ) \div 90000 \quad \text{(E.5)}$$

— Otherwise (decoding unit m is the first decoding unit of a subsequent buffering period), initArrivalEarliestTime[m] is derived as follows:

$$\text{initArrivalEarliestTime[m]} = \text{tmpNominalRemovalTime} - (\text{InitCabRemovalDelay[Htid][SchedSelIdx]} \div 90000) \quad \text{(E.6)}$$

The final arrival time for decoding unit m is derived as follows:

$$\text{AuFinalArrivalTime[m]} = \text{initArrivalTime[m]} + \text{sizeInBits[m]} \div \text{BitRate[!NalHrdModeFlag][Htid][SchedSelIdx]} \quad \text{(E.7)}$$

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 E.1.

The values of SchedSelIdx, BitRate[!NalHrdModeFlag][Htid][SchedSelIdx], and CabSize[!NalHrdModeFlag][Htid][SchedSelIdx] are constrained as follows:

— If the content of the selected hrd_parameters( ) syntax structures for the coded atlas access unit containing decoding unit m and the previous coded atlas 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 coded atlas access unit containing decoding unit m that results in a BitRate[!NalHrdModeFlag][Htid][SchedSelIdx1] or CabSize[!NalHrdModeFlag][Htid][SchedSelIdx1] for the access unit containing decoding unit m. The value of BitRate[!NalHrdModeFlag][Htid][SchedSelIdx1] or CabSize[!NalHrdModeFlag][Htid][SchedSelIdx1] may differ from the value of BitRate[!NalHrdModeFlag][Htid][SchedSelIdx0] or CabSize[!NalHrdModeFlag][Htid][SchedSelIdx0] for the value SchedSelIdx0 of SchedSelIdx that was in use for the previous coded atlas access unit.

— Otherwise, the HSS continues to operate with the previous values of SchedSelIdx, BitRate[!NalHrdModeFlag][Htid][SchedSelIdx], and CabSize[!NalHrdModeFlag][Htid][SchedSelIdx].

When the HSS selects values of BitRate[!NalHrdModeFlag][Htid][SchedSelIdx] or CabSize[!NalHrdModeFlag][Htid][SchedSelIdx] that differ from those of the previous access unit, the following applies:

— The variable BitRate[!NalHrdModeFlag][Htid][SchedSelIdx] comes into effect at the initial CAB arrival time of the current coded atlas access unit.

— The variable CabSize[!NalHrdModeFlag][Htid][SchedSelIdx] comes into effect as follows:

  — If the new value of CabSize[!NalHrdModeFlag][Htid][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[!NalHrdModeFlag][Htid][SchedSelIdx] comes into effect at the CAB removal time of the current access unit.

E.2.3 Timing of decoding unit removal and decoding of decoding unit

The variables InitCabRemovalDelay[Htid][SchedSelIdx], InitCabRemovalDelayOffset[Htid][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[Htid][SchedSelIdx] and InitCabRemovalDelayOffset[Htid][SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements bp_nal_initial_alt_cab_removal_delay[Htid][SchedSelIdx] and bp_nal_initial_alt_cab_removal_offset[Htid][SchedSelIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_acl_initial_alt_cab_removal_ delay[Htid][SchedSelIdx] and bp_acl_initial_alt_cab_removal_offset[Htid][SchedSelIdx], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message containing the syntax elements is selected as specified in Clause E.1.

— Coded atlas access unit 0 is a BLA access unit for which the coded atlas has nal_unit_type is equal to NAL_BLA_W_RADL, NAL_BLA_N_LP, NAL_GBLA_W_RADL, or NAL_GBLA_N_LP and the value of bp_irap_cab_params_present_flag of the buffering period SEI message is equal to 1.

— Coded atlas access unit 0 is a BLA access unit for which the coded atlas access unit has nal_unit_type equal to NAL_BLA_W_LP or NAL_GBLA_W_LP, or is a CRA coded atlas 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:

  — UseAltCabParamsFlag for coded atlas access unit 0 is equal to 1.

  — DefaultInitCabParamsFlag is equal to 0.

— Otherwise, InitCabRemovalDelay[Htid][SchedSelIdx] and InitCabRemovalDelayOffset[Htid][SchedSelIdx] are set equal to the values of the buffering period SEI message syntax elements bp_nal_initial_cab_removal_delay[Htid][SchedSelIdx] and bp_nal_initial_cab_removal_offset[Htid][SchedSelIdx], respectively, when NalHrdModeFlag is equal to 1, or bp_acl_initial_cab_removal_delay[Htid][SchedSelIdx] and bp_acl_initial_cab_removal_offset[Htid][SchedSelIdx], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message containing the syntax elements is selected as specified in Clause E.1, and CabDelayOffset and DabDelayOffset are both set equal to 0.

The nominal removal time of the coded atlas access unit n from the CAB is specified as follows:

— If coded atlas 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:

$$\text{AuNominalRemovalTime}[\ 0\ ] = \text{InitCabRemovalDelay}[\ \text{Htid}\ ][\ \text{SchedSelIdx}\ ] \div 90000 \qquad \text{(E.8)}$$

— Otherwise, the following applies:

When coded atlas access unit n is the first coded atlas access unit of a buffering period that does not initialize the HRD, the following applies:

The nominal removal time of the coded atlas access unit n from the CAB is specified by:

```
if( !concatenationFlag ) {
    baseTime = AuNominalRemovalTime[ firstAtlasInPrevBuffPeriod ]
    removalDelay = AuCabRemovalDelayVal
} else {
    offsetTime = AuFinalArrivalTime[ n − 1 ] − AuNominalRemovalTime[ n − 1 ]
    baseTime = AuNominalRemovalTime[ PrevNonDiscardableAtlasFrame ]
    removalDelay = Max( auCabRemovalDelayDelta,                          (E.9)
        Ceil( ( InitCabRemovalDelay[ Htid ][ SchedSelIdx ] ÷ 90000 + offsetTime )
        ÷ ClockTick ) )
}
AuNominalRemovalTime( n ) = ClockTick * ( removalDelay − CabDelayOffset ) + baseTime
```

where AuNominalRemovalTime[firstAtlasInPrevBuffPeriod] is the nominal removal time of the first coded atlas access unit of the previous buffering period, AuNominalRemovalTime[PrevNonDiscardableAtlasFrame] is the nominal removal time of the preceding coded atlas access unit in decoding order with TemporalID equal to 0 that is not a RASL, RADL, or SLNR atlas frame, AuCabRemovalDelayVal is the value of AuCabRemovalDelayVal derived according to aft_au_cab_removal_delay_minus1[Htid] in the atlas frame timing SEI message, selected as specified in Clause E.1, associated with coded atlas access unit n, and concatenationFlag and auCabRemovalDelayDelta are equal to bp_concatenation_flag and (bp_atlas_cab_removal_delay_delta_minus1 + 1), respectively, selected as specified in Clause E.1, and associated with coded atlas access unit n.

After the derivation of the nominal CAB removal time and before the derivation of the DAB output time of coded atlas 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 E.1:

  — Coded atlas access unit n is a BLA access unit for which the nal_unit_type is equal to NAL_BLA_W_RADL, NAL_GBLA_W_RADL, NAL_BLA_N_LP, or NAL_GBLA_N_LP and the value of bp_irap_cab_params_present_flag of the buffering period SEI message is equal to 1.

  — Coded atlas access unit n is a BLA access unit for which the nal_unit_type is equal to NAL_BLA_W_LP or NAL_GBLA_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 DabDelayOffset are both set equal to 0.

— When coded atlas access unit n is not the first coded atlas access unit of a buffering period, the nominal removal time of the coded atlas access unit n from the CAB is specified by:

$$\begin{aligned}&\text{AuNominalRemovalTime}[\ n\ ] =\\&\quad\text{AuNominalRemovalTime}[\ \text{firstAtlasInCurrBuffPeriod}\ ] +\\&\quad\text{ClockTick} * (\ \text{AuCabRemovalDelayVal} - \text{CabDelayOffset}\ )\end{aligned} \tag{E.10}$$

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_au_cab_removal_delay_minus1[Htid] in the atlas timing SEI message, selected as specified in Clause E.1, and associated with access unit n.

E.3 Operation of the decoded atlas frame buffer

E.3.1 General

The specifications in this subclause apply independently to each set of decoded atlas frame buffer (DAB) parameters selected as specified in Clause E.1.

The DAB 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 subclauses E.3.2 and E.3.3 are sequentially applied.

E.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 header of the first tile of the current atlas frame) happens instantaneously at the CAB removal time of the first decoding unit of coded atlas access unit n (containing the current atlas frame) and proceeds as follows:

— The reference atlas list (RAFL) construction process, as specified in subclause 9.2.4.3, followed by the reference atlas frame marking process, as specified in subclause 9.2.4.4, are invoked.

— When the current atlas frame is an IRAP coded atlas frame with NoOutputBeforeRecoveryFlag equal to 1 that is not atlas frame 0, the following ordered steps are applied:

  — 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 (regardless of the value of ath_no_output_of_prior_atlas_frames_flag).

    — Otherwise, if the value of asps_frame_width, asps_frame_height, or asps_max_dec_atlas_frame_buffering_minus1 derived for the current atlas frame is different from the value of asps_frame_width, asps_frame_height, or asps_max_dec_atlas_frame_buffering_minus1, respectively, derived for the preceding atlas frame in decoding order, NoOutputOfPriorAtlasFramesFlag may (but should not) be set equal to 1 by the decoder under test, regardless of the value of ath_no_output_of_prior_atlas_frames_flag of the current coded atlas access.

    NOTE Although setting NoOutputOfPriorAtlasFramesFlag equal to ath_no_output_of_prior_atlas_frames_flag is preferred under these conditions, it is possible for the decoder under test to set NoOutputOfPriorAtlasFramesFlag to 1 in this case.

    — Otherwise, NoOutputOfPriorAtlasFramesFlag is set equal to ath_no_output_of_prior_atlas_frames_flag.

— 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 frame 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 atlas frame 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 AuCabRemovalTime[m].

  — For each atlas frame that is removed from the DAB, the DAB fullness is decremented by one.

E.3.3 Atlas frame output

The processes specified in this subclause happen instantaneously at the CAB removal time of the coded atlas access unit n, AuCabRemovalTime[n].

When atlas frame 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 coded atlas access unit n is the first access unit of a buffering period and 0 otherwise:

```
DabOutputTime[ n ] = AuCabRemovalTime[ n ] + ClockTick * atlasFrameDabOutputDelay
if( firstAtlasFrameInBufferingPeriodFlag )                                    (E.11)
    DabOutputTime[ n ] −= ClockTick * DabDelayOffset
```

where atlasFrameDabOutputDelay is the value of aft_dab_output_delay[Htid] in the atlas frame timing SEI message associated with coded atlas access unit n.

The output of the current atlas is specified as follows:

— If the ath_atlas_output_flag is equal to 1 for at least one atlas tile with tile ID equal to tileID, then the variable AtlasFrameOutputFlag is set to be equal to 1. Otherwise the variable AtlasFrameOutputFlag is set to be equal to 0.

— 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 subclause E.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 subclause E.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:

$$\text{DabOutputInterval[n]} = \text{DabOutputTime[nextAtlasFrameInOutputOrder]} - \text{DabOutputTime[n]} \quad \text{(E.12)}$$

where nextAtlasFrameInOutputOrder is the atlas frame that follows atlas frame n in output order and has AtlasFrameOutputFlag equal to 1.

E.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, and 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 tiles 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, it is expected that decoders start storing decoded parts of the current atlas frames into the DAB when the first tile is decoded and continue storing more decoded samples as the decoding process proceeds.

E.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 coded atlas 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.

E.4 Bitstream conformance

A bitstream of coded data conforming to this document shall fulfil all requirements specified in this subclause.

The bitstream shall be constructed according to the syntax, semantics, and constraints specified in this document outside this annex.

The first coded atlas access unit in a bitstream shall be an IRAP coded atlas frame, i.e., an IDR atlas frame, a CRA atlas frame, or a BLA atlas frame.

The atlas sub-bitstream is tested by the HRD for conformance as specified in Clause E.1.

For each current atlas frame, let the variables maxAtlasFrameOrderCnt and minAtlasFrameOrderCnt be set equal to the maximum and the minimum, respectively, of the AtlasFrmOrderCntVal values of the following atlas frames:

— The current atlas frame.

— The previous atlas frame in decoding order that has TemporalId equal to 0 and that is not a RASL, RADL, or SLNR atlas frame

— The short-term reference atlas frames in the reference atlas list (RAFL) 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:

a) For each coded 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:

deltaTime90k[n] = 90 000 * (AuNominalRemovalTime[n] − AuFinalArrivalTime[n − 1]) (E.13)

The value of InitCabRemovalDelay[Htid][SchedSelIdx] is constrained as follows:

1) If hrd_cbr_flag[!NalHrdModeFlag][Htid][SchedSelIdx] is equal to 0, the following condition shall be true:

InitCabRemovalDelay[Htid][SchedSelIdx] <= Ceil(deltaTime90k[n]) (E.14)

2) Otherwise (hrd_cbr_flag[!NalHrdModeFlag][Htid][SchedSelIdx] is equal to 1), the following condition shall be true:

Floor(deltaTime90k[n]) <=
    InitCabRemovalDelay[Htid][SchedSelIdx] <= Ceil(deltaTime90k[n]) (E.15)

NOTE 1 The exact number of bits in the CAB at the removal time of each atlas frame can depend on which buffering period SEI message is selected to initialize the HRD. Encoders are expected to take this into account to ensure that all specified constraints are obeyed regardless of which buffering period SEI message is selected to initialize the HRD, as the HRD can be initialized at any one of the buffering period SEI messages.

b) 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.

c) When hrd_low_delay_flag[Htid] is equal to 0, the CAB shall never underflow. A CAB underflow is specified as follows:

1) the condition in which the nominal CAB removal time of the coded atlas access unit n, AuNominalRemovalTime[n], is less than the final CAB arrival time of the coded atlas access unit n, AuFinalArrivalTime[n], for at least one value of n.

d) The nominal removal times of atlas frames 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.

e) For each current atlas frame, after invocation of the process for removal of atlases from the DAB as specified in subclause E.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.

f) 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 E.3

g) For each current atlas frame that is not an IRAP coded atlas frame with NoOutputBeforeRecoveryFlag equal to 1, the value of maxAtlasFrameOrderCnt − minAtlasFrameOrderCnt shall be less than MaxAtlasFrmOrderCntLsb / 2.

h) The value of DabOutputInterval[n] as given by Formula (E.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 subclause 9.2.

i) For any two atlas frames m and n in the same CAS, when DabOutputTime[m] is greater than DabOutputTime[n], the AtlasFrmOrderCntVal of atlas frame m shall be greater than the AtlasFrmOrderCntVal of atlas frame n.

NOTE 2 All atlas frames of an earlier CAS in decoding order that are output are output before any atlas frames of a later CAS in decoding order. Within any particular CAS, the atlas frames that are output are output in increasing AtlasFrmOrderCntVal order.

E.5 Decoder conformance

E.5.1 General

A decoder conforming to this document shall fulfil all requirements specified in this subclause.

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 E.4, in the manner specified in Annex A, provided that all AAPSs, 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 document.

When an atlas sub-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 sub-bitstream is otherwise conforming to this document, a conforming decoder shall decode the atlas sub-bitstream in the same manner as it would decode a conforming atlas sub-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 E.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 information associated with all atlas tiles and patches that are output shall be equivalent to the information associated with the atlas tiles and patches 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[Htid] greater than 0, the atlas frame decoder shall be capable of decoding the atlas sub-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:

$$\alpha = (r - \text{BitRate}[\ !\text{NalHrdModeFlag}\ ][\ \text{Htid}\ ][\ \text{SchedSelIdx} - 1\ ]\ ) \div (\text{BitRate}[\ !\text{NalHrdModeFlag}\ ][\ \text{Htid}\ ][\ \text{SchedSelIdx}\ ] - \text{BitRate}[\ !\text{NalHrdModeFlag}\ ][\ \text{Htid}\ ][\ \text{SchedSelIdx} - 1\ ]\ ) \qquad \text{(E.16)}$$

$$c(r) = \alpha * \text{CabSize}[!\text{NalHrdModeFlag}][\text{Htid}][\text{SchedSelIdx}] + (1 - \alpha) * \text{CabSize}[!\text{NalHrdModeFlag}][\text{Htid}][\text{SchedSelIdx} - 1] \quad \text{(E.17)}$$

$$f(r) = \alpha * \text{InitCabRemovalDelay}[\text{Htid}][\text{SchedSelIdx}] * \text{BitRate}[!\text{NalHrdModeFlag}][\text{Htid}][\text{SchedSelIdx}] + (1 - \alpha) * \text{InitCabRemovalDelay}[\text{Htid}][\text{SchedSelIdx} - 1] * \text{BitRate}[!\text{NalHrdModeFlag}][\text{Htid}][\text{SchedSelIdx} - 1] \quad \text{(E.18)}$$

for any SchedSelIdx > 0 such that:

$$\text{BitRate}[!\text{NalHrdModeFlag}][\text{Htid}][\text{SchedSelIdx} - 1] <= r <= \text{BitRate}[!\text{NalHrdModeFlag}][\text{Htid}][\text{SchedSelIdx}]$$

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[Htid][SchedSelIdx] can be different from one buffering period to another and has 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 buffer of the DUT could be as small as the size of the largest decoding unit.

— A modified HRD, as described in subclause E.5.2, 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 output order of atlas frames shall be the same for both the HRD and the DUT.

— The HRD CAB size is given by CabSize[!NalHrdModeFlag][Htid][SchedSelIdx], as specified in subclause G.3.3, where SchedSelIdx and the HRD parameters are selected as specified in Clause E.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 subclauses E.5.2 through E.5.2.3.

E.5.2 Operation of the output order DAB

E.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 subclause E.5.2.2 is invoked, the invocation of the process for current decoded atlas frame marking and storage as specified in subclause E.3.4, further followed by the invocation of the process for removal of the atlas frame from the DAB after decoding of the current atlas frame as specified in subclause E.3.5, and finally followed by the invocation of the process for additional bumping as specified in subclause E.5.2.3. The "bumping" process is specified in subclause E.5.2.4 and is invoked as specified in subclauses E.5.2.2 and E.5.2.3.

E.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 header of the first tile 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:

a) The decoding process for RAFL as specified in subclause 9.2.4.3, is invoked.

b) If the current picture is an IRAP coded atlas frame with NoOutputOfPriorAtlasFramesFlag equal to 1 that is not atlas frame 0, the following ordered steps are applied:

  1) The variable NoOutputOfPriorAtlasFrameFlag is derived for the decoder under test as follows:

    i) If the current atlas frame is CRA atlas frame, NoOutputOfPriorAtlasFramesFlag is set equal to 1 (regardless of the value of ath_no_output_of_prior_atlas_frames_flag).

    ii) Otherwise, if the value of asps_frame_width, asps_frame_height, or asps_max_dec_atlas_frame_buffering_minus1 derived for the current atlas frame is different from the value of asps_frame_width, asps_frame_height, or asps_max_dec_atlas_frame_buffering_minus1, respectively, derived from the ASPS active for the preceding atlas, NoOutputOfPriorAtlasFramesFlag may (but should not) be set equal to 1 by the decoder under test, regardless of the value of ath_no_output_of_prior_atlas_frames_flag of the current atlas frame.

    iii) Otherwise, NoOutputOfPriorAtlasFramesFlag is set equal to ath_no_output_of_prior_atlas_frames_flag if the current coded atlas access unit.

  NOTE Although, under these conditions, setting NoOutputOfPriorAtlasFramesFlag equal to ath_no_output_of_prior_atlas_frames_flag of the current coded atlas frame is preferred, it is possible for the decoder under test to set, in this case, NoOutputOfPriorAtlasFramesFlag equal to 1.

  2) The value of NoOutputOfPriorAtlasFramesFlag derived for the decoder under test is applied for the HRD as follows:

    i) If NoOutputOfPriorAtlasFramesFlag is equal to 1, all atlas frame storage buffers in the DAB are emptied without output of the atlas frames they contain, and the DAB fullness is set equal to 0.

    ii) 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 frame storage buffers in the DAB are emptied by repeatedly invoking the "bumping" process specified in subclause E.5.2.4 and the DAB fullness is set equal to 0.

c) Otherwise (the current atlas frame is not an IRAP coded atlas frame with NoOutputBeforeRecoveryFlag equal to 1), all atlas frame 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. The "bumping" process specified in subclause E.5.2.4 is invoked repeatedly for each additional atlas frame storage buffer that is emptied, until the number of atlas frames in the DAB is less than asps_max_dec_atlas_frame_buffering_minus1 + 1.

E.5.2.3 Additional bumping

The processes specified in this subclause happen instantaneously when the last decoding unit of access unit n containing the current atlas frame is removed from the CAB.

When the current atlas frame has AtlasFrameOutputFlag equal to 1, for each atlas frame in the DAB that is marked as "needed for output" and follows the current atlas frame in output order, the associated variable AtlasFrameLatencyCount is set equal to AtlasFrameLatencyCount + 1.

The following applies:

— If the current decoded atlas frame 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 frame has AtlasFrameOutputFlag equal to 0), it is marked as "not needed for output".

When the following condition is true, the "bumping" process specified in subclause E.5.2.4 is invoked repeatedly until the following conditions is not 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 + 1.

E.5.2.4 "Bumping" process

The "bumping" process consists of the following ordered steps:

a) The atlas frame that is first for output is selected as the one having the smallest value of AtlasFrmOrderCntVal of all atlas frames in the DAB is marked as "needed for output".

b) The atlas frame is output, and the atlas frame is marked as "not needed for output".

c) When the atlas frame storage buffer that included the atlas frame that was output contains an atlas frame marked as "unused for reference", the atlas frame storage buffer is emptied.

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 AtlasFrmOrderCntVal of atlasFrameA is less than the value of AtlasFrmOrderCntVal of atlasFrameB.

Annex F
(normative)

Supplemental enhancement information

F.1 General

This annex specifies syntax and semantics for SEI message payloads associated with an atlas, identified with an atlas ID, SeiAtlasID, which is set equal to vuh_atlas_id or determined through external means if the V3C unit header is unavailable.

SEI messages assist in processes related to decoding, reconstruction, display, or other purposes. This annex defines two types of SEI messages: essential and non-essential.

Non-essential SEI messages are not required by the decoding process. Conforming decoders are not required to process this information for output order conformance to this document (see Annex A for the specification of conformance).

In subclause E.5.2 including its subclauses, specification for presence of non-essential 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 document. When present in the bitstream, non-essential SEI messages shall obey the syntax and semantics specified in subclause 8.3.8 and this annex. When the content of a non-essential 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.

Essential SEI messages are an integral part of the V3C bitstream and should not be removed from the bitstream. The essential SEI messages are categorized into two types, based on the conformance point they are related to, as specified in Annex A.

a) Conformance point A, named as Type-A, essential SEI messages: These SEIs contain information required to check bitstream conformance and for output timing decoder conformance. Every V3C decoder conforming to point A should not discard any relevant Type-A essential SEI messages and shall consider them for bitstream conformance and for output timing decoder conformance.

b) Conformance point B, named as Type-B, essential SEI messages: V3C decoders that wish to conform to a particular reconstruction profile as specified in Annex A should not discard any relevant Type-B essential SEI messages and should consider them for volumetric frame reconstruction and conformance purposes.

Table F.1 lists the essential and non-essential SEI messages. In case of essential SEI messages, the type is also specified.

Table F.1 — Essential and non-essential SEI messages

SEI message	NAL Type	Conformance Type
Filler payload SEI	NAL_PREFIX_NSEI/ NAL_SUFFIX_NSEI	N/A
User data registered by Recommendation ITU-T T.35	NAL_PREFIX_NSEI/ NAL_SUFFIX_NSEI	N/A
User data unregistered	NAL_PREFIX_NSEI/ NAL_SUFFIX_NSEI	N/A
Recovery point	NAL_PREFIX_NSEI	N/A

Table F.1 *(continued)*

SEI message	NAL Type	Conformance Type
No reconstruction	NAL_PREFIX_NSEI	N/A
Reserved	N/A	N/A
SEI manifest	NAL_PREFIX_ESEI	Type-A, Type-B
SEI prefix indication	NAL_PREFIX_ESEI	Type-A, Type-B
Active sub-bitstreams	NAL_PREFIX_ESEI	Type-A
Component codec mapping	NAL_PREFIX_ESEI	Type-A
Scene object information	NAL_PREFIX_NSEI	N/A
Object label information	NAL_PREFIX_NSEI	N/A
Patch information	NAL_PREFIX_NSEI	N/A
Volumetric rectangle information	NAL_PREFIX_NSEI	N/A
Atlas object association	NAL_PREFIX_NSEI	N/A
Buffering period	NAL_PREFIX_ESEI	Type-A
Atlas frame timing	NAL_PREFIX_ESEI	Type-A
Viewport camera parameters	NAL_PREFIX_NSEI	N/A
Viewport position	NAL_PREFIX_NSEI	N/A
Decoded atlas information hash	NAL_SUFFIX_NSEI	N/A
Time code	NAL_PREFIX_NSEI	N/A
Attribute transformation parameters	NAL_PREFIX_NSEI	Type-B
Occupancy synthesis	NAL_PREFIX_ESEI	Type-B
Geometry smoothing	NAL_PREFIX_ESEI	Type-B
Attribute smoothing	NAL_PREFIX_ESEI	Type-B

F.2 SEI payload syntax

F.2.1 General SEI message syntax

sei_payload(payloadType, payloadSize) {	**Descriptor**
if((nal_unit_type == NAL_PREFIX_NSEI) || (nal_unit_type == NAL_PREFIX_ESEI)) {	
if(payloadType == 0)	
buffering_period(payloadSize)	
else if(payloadType == 1)	
atlas_frame_timing(payloadSize)	
else if(payloadType == 2)	
filler_payload(payloadSize)	
else if(payloadType == 3)	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType == 4)	
user_data_unregistered(payloadSize)	
else if(payloadType == 5)	
recovery_point(payloadSize)	
else if(payloadType == 6)	
no_reconstruction(payloadSize)	
else if(payloadType == 7)	

time_code(payloadSize)	
else if(payloadType == 8)	
sei_manifest(payloadSize)	
else if(payloadType == 9)	
sei_prefix_indication(payloadSize)	
else if(payloadType == 10)	
active_sub_bitstreams(payloadSize)	
else if(payloadType == 11)	
component_codec_mapping(payloadSize)	
else if(payloadType == 12)	
scene_object_information(payloadSize)	
else if(payloadType == 13)	
object_label_information(payloadSize)	
else if(payloadType == 14)	
patch_information(payloadSize)	
else if(payloadType == 15)	
volumetric_rectangle_information(payloadSize)	
else if(payloadType == 16)	
atlas_object_association(payloadSize)	
else if(payloadType == 17)	
viewport_camera_parameters(payloadSize)	
else if(payloadType == 18)	
viewport_position(payloadSize)	
else if(payloadType == 64)	
attribute_transformation_params(payloadSize) /* Specified in Annex H */	
else if(payloadType == 65)	
occupancy_synthesis(payloadSize) /* Specified in Annex H */	
else if(payloadType == 66)	
geometry_smoothing(payloadSize) /* Specified in Annex H */	
else if(payloadType == 67)	
attribute_smoothing(payloadSize) /* Specified in Annex H */	
else	
reserved_sei_message(payloadSize)	
}	
else { /*(nal_unit_type == NAL_SUFFIX_NSEI) || (nal_unit_type == NAL_SUFFIX_ESEI)*/	
if(payloadType == 2)	
filler_payload(payloadSize)	
else if(payloadType == 3)	
user_data_registered_itu_t_t35(payloadSize)	
else if(payloadType == 4)	
user_data_unregistered(payloadSize)	
else if(payloadType == 19)	
decoded_atlas_information_hash(payloadSize)	

else	
reserved_sei_message(payloadSize)	
}	
if(more_data_in_payload( )) {	
if(payload_extension_present( ))	
sp_reserved_payload_extension_data	u(v)
byte_alignment( )	
}	
}	

F.2.2 Filler payload SEI message syntax

filler_payload(payloadSize) {	**Descriptor**
for(k = 0; k < payloadSize; k++)	
ff_byte /* equal to 0xFF */	f(8)
}	

F.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)	
}	

F.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)
}	

F.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)

}	

F.2.6 No reconstruction SEI message syntax

no_reconstruction(payloadSize) {	**Descriptor**
}	

F.2.7 Reserved SEI message syntax

reserved_sei_message(payloadSize) {	**Descriptor**
for(i = 0; i < payloadSize; i++)	
reserved_sei_message_payload_byte	b(8)
}	

F.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)
}	
}	

F.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)
}	
}	

F.2.10 Active sub-bitstreams SEI message syntax

active_sub_bitstreams(payloadSize) {	**Descriptor**
asb_active_sub_bitstreams_cancel_flag	u(1)
if(!asb_active_sub_bitstreams_cancel_flag) {	
asb_active_attributes_changes_flag	u(1)
asb_active_maps_changes_flag	u(1)
asb_auxiliary_sub_bitstreams_active_flag	u(1)
if(asb_active_attributes_changes_flag) {	
asb_all_attributes_active_flag	u(1)

if(!asb_all_attributes_active_flag) {	
asb_active_attribute_count_minus1	u(7)
for(i = 0; i <= asb_active_attribute_count_minus1; i++)	
asb_active_attribute_idx[i]	u(7)
}	
}	
if(asb_active_maps_changes_flag) {	
asb_all_maps_active_flag	u(1)
if(!asb_all_maps_active_flag) {	
asb_active_map_count_minus1	u(4)
for(i = 0; i <= asb_active_map_count_minus1)	
asb_active_map_idx[i]	u(4)
}	
}	
}	
}	

F.2.11 Component codec mapping SEI message syntax

component_codec_mapping(payloadSize) {	**Descriptor**
ccm_component_codec_cancel_flag	u(1)
if(!ccm_component_codec_cancel_flag) {	
ccm_codec_mappings_count_minus1	u(8)
for(i = 0; i <= ccm_codec_mappings_count_minus1; i++) {	
ccm_codec_id	u(8)
ccm_codec_4cc[ccm_codec_id]	st(v)
}	
}	
}	

F.2.12 Volumetric annotation SEI message family syntax

F.2.12.1 Scene object information SEI message syntax

scene_object_information(payloadSize) {	**Descriptor**
soi_persistence_flag	u(1)
soi_reset_flag	u(1)
soi_num_object_updates	ue(v)
if(soi_num_object_updates > 0) {	
soi_simple_objects_flag	u(1)
if(soi_simple_objects_flag == 0) {	
soi_object_label_present_flag	u(1)
soi_priority_present_flag	u(1)
soi_object_hidden_present_flag	u(1)
soi_object_dependency_present_flag	u(1)
soi_visibility_cones_present_flag	u(1)

soi_3d_bounding_box_present_flag	u(1)
soi_collision_shape_present_flag	u(1)
soi_point_style_present_flag	u(1)
soi_material_id_present_flag	u(1)
soi_extension_present_flag	u(1)
}	
else {	
soi_object_label_present_flag = 0	
soi_priority_present_flag = 0	
soi_object_hidden_present_flag = 0	
soi_object_dependency_present_flag = 0	
soi_visibility_cones_present_flag = 0	
soi_3d_bounding_box_present_flag = 0	
soi_collision_shape_present_flag = 0	
soi_point_style_present_flag = 0	
soi_material_id_present_flag = 0	
soi_extension_present_flag = 0	
}	
if(soi_3d_bounding_box_present_flag) {	
soi_3d_bounding_box_scale_log2	u(5)
}	
soi_log2_max_object_idx_updated_minus1	u(5)
if(soi_object_dependency_present_flag)	
soi_log2_max_object_dependency_idx	u(5)
for(i = 0; i < soi_num_object_updates; i++) {	
soi_object_idx[i]	u(v)
k = soi_object_idx[i]	
soi_object_cancel_flag[k]	u(1)
ObjectTracked[k] = !soi_object_cancel_flag[k]	
if(!soi_object_cancel_flag[k]) {	
if(soi_object_label_present_flag) {	
soi_object_label_update_flag[k]	u(1)
if(soi_object_label_update_flag[k])	
soi_object_label_idx[k]	ue(v)
}	
if(soi_priority_present_flag) {	
soi_priority_update_flag[k]	u(1)
if(soi_priority_update_flag[k])	
soi_priority_value[k]	u(4)
}	
if(soi_object_hidden_present_flag)	
soi_object_hidden_flag[k]	u(1)
if(soi_object_dependency_present_flag) {	
soi_object_dependency_update_flag[k]	u(1)

if(soi_object_dependency_update_flag[k]) {	
soi_object_num_dependencies[k]	u(4)
for(j = 0; j < soi_object_num_dependencies[k]; j++)	
soi_object_dependency_idx[k][j]	u(v)
}	
}	
if(soi_visibility_cones_present_flag) {	
soi_visibility_cones_update_flag[k]	u(1)
if(soi_visibility_cones_update_flag[k]) {	
soi_direction_x[k]	i(16)
soi_direction_y[k]	i(16)
soi_direction_z[k]	i(16)
soi_angle[k]	u(16)
}	
}	
if(soi_3d_bounding_box_present_flag) {	
soi_3d_bounding_box_update_flag[k]	u(1)
if(soi_3d_bounding_box_update_flag[k]) {	
soi_3d_bounding_box_x[k]	ue(v)
soi_3d_bounding_box_y[k]	ue(v)
soi_3d_bounding_box_z[k]	ue(v)
soi_3d_bounding_box_size_x[k]	ue(v)
soi_3d_bounding_box_size_y[k]	ue(v)
soi_3d_bounding_box_size_z[k]	ue(v)
}	
}	
if(soi_collision_shape_present_flag) {	
soi_collision_shape_update_flag[k]	u(1)
if(soi_collision_shape_update_flag[k])	
soi_collision_shape_id[k]	u(16)
}	
if(soi_point_style_present_flag) {	
soi_point_style_update_flag[k]	u(1)
if(soi_point_style_update_flag[k]) {	
soi_point_shape_id[k]	u(8)
soi_point_size[k]	u(16)
}	
}	
if(soi_material_id_present_flag) {	
soi_material_id_update_flag[k]	u(1)
if(soi_material_id_update_flag[k])	
soi_material_id[k]	u(16)
}	
}	

}	
}	
}	

F.2.12.2 Object label information SEI message syntax

object_label_information(payloadSize) {	**Descriptor**
oli_cancel_flag	u(1)
if(!oli_cancel_flag) {	
oli_label_language_present_flag	u(1)
if(oli_label_language_present_flag) {	
while(!byte_aligned( ))	
oli_bit_equal_to_zero /* equal to 0 */	f(1)
oli_label_language	st(v)
}	
oli_num_label_updates	ue(v)
for(i = 0; i < oli_num_label_updates ; i++) {	
oli_label_idx[i]	ue(v)
k = oli_label_idx[i]	
oli_label_cancel_flag[k]	u(1)
LabelSetFlag[k] = !oli_label_cancel_flag[k]	
if(!oli_label_cancel_flag[k]) {	
while(!byte_aligned( ))	
oli_bit_equal_to_zero /* equal to 0 */	f(1)
oli_label[k]	st(v)
}	
}	
oli_persistence_flag	u(1)
}	
}	

F.2.12.3 Patch information SEI message syntax

patch_information(payloadSize) {	**Descriptor**
pi_persistence_flag	u(1)
pi_reset_flag	u(1)
pi_num_tile_updates	ue(v)
if(pi_num_tile_updates > 0) {	
pi_log2_max_object_idx_tracked_minus1	u(5)
pi_log2_max_patch_idx_updated_minus1	u(4)
}	
for(i = 0; i < pi_num_tile_updates; i++) {	
pi_tile_id[i]	ue(v)
j = pi_tile_id[i]	
pi_tile_cancel_flag[j]	u(1)

pi_num_patch_updates[j]	ue(v)
for(k = 0; k < pi_num_patch_updates[j]; k++) {	
pi_patch_idx[j][k]	u(v)
p = pi_patch_idx[j][k]	
pi_patch_cancel_flag[j][p]	u(1)
if(!pi_patch_cancel_flag[j][p]) {	
pi_patch_number_of_objects_minus1[j][p]	ue(v)
m = pi_patch_number_of_objects_minus1[j][p] + 1	
for(n = 0; n < m; n++)	
pi_patch_object_idx[j][p][n]	u(v)
}	
}	
}	
}	

F.2.12.4 Volumetric rectangle information SEI message syntax

volumetric_rectangle_information(payloadSize) {	**Descriptor**
vri_persistence_flag	u(1)
vri_reset_flag	u(1)
vri_num_rectangles_updates	ue(v)
if(vri_num_rectangles_updates > 0) {	
vri_log2_max_object_idx_tracked_minus1	u(5)
vri_log2_max_rectangle_idx_updated_minus1	u(4)
}	
for(k = 0; k < vri_num_rectangles_updates; k++) {	
vri_rectangle_idx[k]	u(v)
p = vri_rectangle_idx[k]	
vri_rectangle_cancel_flag[p]	u(1)
if(!vri_rectangle_cancel_flag[p]) {	
vri_bounding_box_update_flag[p]	u(1)
if(vri_bounding_box_update_flag[p]) {	
vri_bounding_box_top[p]	u(v)
vri_bounding_box_left[p]	u(v)
vri_bounding_box_width[p]	u(v)
vri_bounding_box_height[p]	u(v)
}	
vri_rectangle_number_of_objects_minus1[p]	ue(v)
m = vri_rectangle_number_of_objects_minus1[p] + 1	
for(n = 0; n < m; n++)	
vri_rectangle_object_idx[p][n]	u(v)
}	
}	
}	

F.2.12.5 Atlas object association SEI message syntax

atlas_object_association(payloadSize) {	**Descriptor**
aoa_persistence_flag	u(1)
aoa_reset_flag	u(1)
aoa_num_atlases_minus1	u(6)
aoa_num_updates	ue(v)
if(aoa_num_updates > 0) {	
aoa_log2_max_object_idx_tracked_minus1	u(5)
for(j = 0; j < aoa_num_atlases_minus1 + 1; j++)	
aoa_atlas_id[j]	u(6)
for(i = 0; i < aoa_num_updates ; i++) {	
aoa_object_idx[i]	u(v)
k = aoa_object_idx[i]	
for(j = 0; j < aoa_num_atlases_minus1 + 1; j++)	
aoa_object_in_atlas[k][aoa_atlas_id[j]]	u(1)
}	
}	
}	

F.2.13 Buffering period SEI message syntax

buffering_period(payloadSize) {	**Descriptor**
bp_nal_hrd_params_present_flag	u(1)
bp_acl_hrd_params_present_flag	u(1)
bp_initial_cab_removal_delay_length_minus1	u(5)
bp_au_cab_removal_delay_length_minus1	u(5)
bp_dab_output_delay_length_minus1	u(5)
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	u(3)
for(i = 0; i <= bp_max_sub_layers_minus1; i++) {	
bp_hrd_cab_cnt_minus1[i]	ue(v)
if(bp_nal_hrd_params_present_flag) {	
for(j = 0; j < bp_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][j]	u(v)
bp_nal_initial_alt_cab_removal_offset[i][j]	u(v)

}	
}	
}	
if(bp_acl_hrd_params_present_flag) {	
for(j = 0; j < bp_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][j]	u(v)
bp_acl_initial_alt_cab_removal_offset[i][j]	u(v)
}	
}	
}	
}	
}	

F.2.14 Atlas frame timing SEI message syntax

atlas_frame_timing(payloadSize) {	**Descriptor**
if(CabDabDelaysPresentFlag) {	
for(i = 0; i <= bp_max_sub_layers_minus1; i++) {	
aft_au_cab_removal_delay_minus1[i]	u(v)
aft_dab_output_delay[i]	u(v)
}	
}	
}	

F.2.15 Viewport SEI messages family syntax

F.2.15.1 Viewport camera parameters SEI messages syntax

viewport_camera_parameters(payloadSize) {	**Descriptor**
vcp_camera_id	u(10)
vcp_cancel_flag	u(1)
if(vcp_camera_id > 0 && !vcp_cancel_flag) {	
vcp_persistence_flag	u(1)
vcp_camera_type	u(3)
if(vcp_camera_type == 0) { /* equirectangular */	
vcp_erp_horizontal_fov	u(32)
vcp_erp_vertical_fov	u(32)
} else if(vcp_camera_type == 1) { /* perspective */	
vcp_perspective_aspect_ratio	fl(32)
vcp_perspective_horizontal_fov	u(32)
} else if(vcp_camera_type == 2) { /* orthographic */	
vcp_ortho_aspect_ratio	fl(32)
vcp_ortho_horizontal_size	fl(32)

}	
vcp_clipping_near_plane	fl(32)
vcp_clipping_far_plane	fl(32)
}	
}	

F.2.15.2 Viewport position SEI messages syntax

viewport_position(payloadSize) {	**Descriptor**
vp_viewport_id	ue(v)
vp_camera_parameters_present_flag	u(1)
if(vp_camera_parameters_present_flag)	
vp_vcp_camera_id	u(10)
vp_cancel_flag	u(1)
if(! vp_cancel_flag) {	
vp_persistence_flag	u(1)
for(d = 0 ; d < 3; d++)	
vp_position[d]	fl(32)
vp_rotation_qx	i(16)
vp_rotation_qy	i(16)
vp_rotation_qz	i(16)
vp_center_view_flag	u(1)
if(! vp_center_view_flag)	
vp_left_view_flag	u(1)
}	
}	

F.2.16 Decoded atlas information hash SEI message syntax

decoded_atlas_information_hash(payloadSize) {	**Descriptor**
daih_cancel_flag	u(1)
if(!daih_cancel_flag) {	
daih_persistence_flag	u(1)
daih_hash_type	u(8)
daih_decoded_high_level_hash_present_flag	u(1)
daih_decoded_atlas_hash_present_flag	u(1)
daih_decoded_atlas_b2p_hash_present_flag	u(1)
daih_decoded_atlas_tiles_hash_present_flag	u(1)
daih_decoded_atlas_tiles_b2p_hash_present_flag	u(1)
daih_reserved_zero_1bit	u(1)
if(daih_decoded_high_level_hash_present_flag)	
decoded_high_level_hash(daih_hash_type)	
if(daih_decoded_atlas_hash_present_flag)	
decoded_atlas_hash(daih_hash_type)	
if(daih_decoded_atlas_b2p_hash_present_flag)	

decoded_atlas_b2p_hash(daih_hash_type)	
if(daih_decoded_atlas_tiles_hash_present_flag || daih_decoded_atlas_tiles_b2p_hash_present_flag) {	
daih_num_tiles_minus1	ue(v)
daih_tile_id_len_minus1	ue(v)
for(t = 0; t <= daih_num_tiles_minus1; t++)	
daih_tile_id[t]	u(v)
byte_align( )	
for(t = 0; t <= daih_num_tiles_minus1; t++) {	
j = daih_tile_id[t]	
if(daih_decoded_atlas_tiles_hash_present_flag)	
decoded_atlas_tile_hash(daih_hash_type, j)	
if(daih_decoded_atlas_tiles_b2p_hash_present_flag)	
decoded_atlas_tile_b2p_hash(daih_hash_type, j)	
}	
}	
}	
}	

F.2.16.1 Decoded high level hash unit syntax

decoded_high_level_hash(hashType) {	**Descriptor**
if(hashType == 0)	
for(i = 0; i < 16; i++)	
daih_high_level_md5[i]	b(8)
else if(hashType == 1)	
daih_high_level_crc	u(16)
else if(hashType == 2)	
daih_high_level_checksum	u(32)
}	

F.2.16.2 Decoded atlas hash unit syntax

decoded_atlas_tile_hash(hashType) {	**Descriptor**
if(hashType == 0)	
for(i = 0; i < 16; i++)	
daih_atlas_md5[i]	b(8)
else if(hashType == 1)	
daih_atlas_crc	u(16)
else if(hashType == 2)	
daih_atlas_checksum	u(32)
}	

F.2.16.3 Decoded atlas b2p hash unit syntax

decoded_atlas_tile_b2p_hash(hashType) {	**Descriptor**
if(hashType == 0)	
for(i = 0; i < 16; i++)	
daih_atlas_b2p_md5[i]	b(8)
else if(hashType == 1)	
daih_atlas_b2p_crc	u(16)
else if(hashType == 2)	
daih_atlas_b2p_checksum	u(32)
}	

F.2.16.4 Decoded atlas tile hash unit syntax

decoded_atlas_tile_hash(hashType, j) {	**Descriptor**
if(hashType == 0)	
for(i = 0; i < 16; i++)	
daih_atlas_tile_md5[j][i]	b(8)
else if(hashType == 1)	
daih_atlas_tile_crc[j]	u(16)
else if(hashType == 2)	
daih_atlas_tile_checksum[j]	u(32)
}	

F.2.16.5 Decoded atlas tile b2p hash unit syntax

decoded_atlas_b2p_hash(hashType, j) {	**Descriptor**
if(hashType == 0)	
for(i = 0; i < 16; i++)	
daih_atlas_tile_b2p_md5[j][i]	b(8)
else if(hashType == 1)	
daih_atlas_tile_b2p_crc[j]	u(16)
else if(hashType == 2)	
daih_atlas_tile_b2p_checksum[j]	u(32)
}	

F.2.17 Time code SEI message syntax

time_code(payloadSize) {	**Descriptor**
num_units_in_tick	u(32)
time_scale	u(32)
counting_type	u(5)
full_timestamp_flag	u(1)
discontinuity_flag	u(1)
cnt_dropped_flag	u(1)
n_frames	u(9)
if(full_timestamp_flag) {	

seconds_value /* 0..59 */	u(6)
minutes_value /* 0..59 */	u(6)
hours_value /* 0..23 */	u(5)
} else {	
seconds_flag	u(1)
if(seconds_flag) {	
seconds_value /* range 0..59 */	u(6)
minutes_flag	u(1)
if(minutes_flag) {	
minutes_value /* 0..59 */	u(6)
hours_flag	u(1)
if(hours_flag)	
hours_value /* 0..23 */	u(5)
}	
}	
}	
time_offset_length	u(5)
if(time_offset_length > 0)	
time_offset_value	i(v)
}	

F.3 SEI payload semantics

F.3.1 General SEI payload semantics

sp_reserved_payload_extension_data shall not be present in bitstreams conforming to this edition of this document. However, decoders conforming to this edition of this document 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 NAL units.

The semantics and persistence scope for each SEI message are specified in the semantics specification for each particular SEI message. Unless indicated otherwise, all defined SEI messages and their persistence scope are associated only with the current atlas with atlas ID SeiAtlasID.

NOTE 2 Persistence information for SEI messages is summarized in Table F.2.

Table F.2 — Persistence scope of SEI messages

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

Table F.2 *(continued)*

SEI message	**Persistence scope**
User data unregistered	Unspecified
Recovery point	Specified by the syntax of the SEI message
Decoded atlas information hash	Specified by the syntax of the SEI message
No reconstruction	The coded atlas access unit containing the SEI message
Time code	The coded atlas access unit containing the SEI message
SEI manifest	The remainder of the bitstream
SEI prefix indication	The remainder of the bitstream
Active sub-bitstreams	The remainder of the bitstream or until a new active attributes SEI message
Component codec mapping	The remainder of the bitstream or until a new active attributes SEI message
Scene object information	Specified by the syntax of the SEI message
Object label information	Specified by the syntax of the SEI message
Patch information	Specified by the syntax of the SEI message
Volumetric rectangle information	Specified by the syntax of the SEI message
Atlas object association	Specified by the syntax of the SEI message
Viewport camera parameters	Specified by the syntax of the SEI message
Viewport position	Specified by the syntax of the SEI message
Attribute transformation parameters	Specified by the syntax of the SEI message
Occupancy synthesis	Specified by the syntax of the SEI message
Geometry smoothing	Specified by the syntax of the SEI message
Attribute smoothing	Specified by the syntax of the SEI message

The values of some SEI message syntax elements 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 document 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.

It is a requirement of bitstream conformance that the following restrictions apply on containing of SEI messages in SEI NAL units:

— When an SEI NAL unit contains a buffering period SEI message or an atlas frame timing SEI message, the SEI NAL unit shall not contain any other SEI message with payloadType not equal to 0 (buffering period) or 1 (atlas frame timing).

Let prevAclNalUnitInAu of an SEI NAL unit or an SEI message be the preceding ACL NAL unit in decoding order, if any, in the same access unit, and nextAclNalUnitInAu of an SEI NAL unit or an SEI message be the next ACL NAL unit in decoding order, if any, in the same access unit.

It is a requirement of bitstream conformance that the following restrictions apply on decoding order of SEI messages:

— When a buffering period SEI message is present in an access unit, it shall not follow any other SEI message that follows the prevAclNalUnitInAu of the buffering period SEI message and precedes the nextAclNalUnitInAu of the buffering period SEI message.

— When an atlas frame timing SEI message is present in an access unit, it shall not follow any other SEI message that follows the prevAclNalUnitInAu of the atlas frame timing SEI message and precedes the nextAclNalUnitInAu of the atlas frame timing SEI message.

— When payloadType is equal to 0 (buffering period) or 1 (atlas frame timing), the SEI NAL unit containing the SEI message shall precede all NAL units of any atlas unit that has nal_layer_id greater than highestAppLayerId, where highestAppLayerId is the greatest value of nal_layer_id of all the layers in all the operation points that the SEI message applies to.

The following applies on the applicable operation points or layers of SEI messages:

— For an SEI message when payloadType is equal to 0 (buffering period) the SEI message applies to the operation point that has OpTid equal to the greatest value of nal_temporal_id_plus1 among all ACL NAL units in the bitstream, and that has OpLayerIdList containing all values of nal_layer_id in all ACL units in the bitstream..

— For an SEI message, when payloadType is equal to 1 (atlas frame timing), the SEI message applies to the operation point that has OpTid equal to the greatest value of nal_temporal_id_plus1 among all ACL NAL units in the bitstream, and that has OpLayerIdList containing all values of nal_layer_id in all ACL units in the bitstream.

It is a requirement of bitstream conformance that the following restrictions apply on the values of nal_layer_id and TemporalId of SEI NAL units:

— When an SEI message has payloadType equal to 0 or 1 the SEI NAL unit containing the SEI message shall have nal_layer_id equal to 0.

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 "CAS" apply to the bitstream BitstreamToDecode that is the sub-bitstream of the particular operation point.

F.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.

F.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 document.

itu_t_t35_country_code shall be a byte having a value specified as a country code by Recommendation ITU-T T.35:2000, Annex A.

itu_t_t35_country_code_extension_byte shall be a byte having a value specified as a country code by Recommendation ITU-T T.35:2000, Annex B.

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.

F.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 document.

uuid_iso_iec_11578 shall have a value specified as a UUID according to the procedures of ISO/IEC 11578:1996, Annex A.

user_data_payload_byte shall be a byte containing data having syntax and semantics as specified by the UUID generator.

F.3.5 Recovery point SEI message semantics

The recovery point SEI message assists a decoder in determining when the decoding process will produce acceptable atlas frames for reconstruction and display after the decoder initiates random access or after the encoder indicates a broken link in the CAS. When the decoding process is started with the access unit in decoding order associated with the recovery point SEI message, all decoded atlas 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 atlas frames produced by random access at or before the atlas 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 atlas frame associated with the recovery point SEI message may contain references to atlas 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 atlas 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 atlas frames can 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 atlas frame was the first atlas frame in the bitstream in decoding order, and the variables prevAtlasFrmOrderCntLsb and prevAtlasFrmOrderCntMsb used in the derivation of AtlasFrmOrderCntVal are both set equal to 0.

NOTE 2 When HRD information is present in the bitstream, a buffering period SEI message can 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.

Any ASPS or AFPS RBSP that is referred to by an atlas frame associated with a recovery point SEI message or by any atlas frame following such an atlas frame in decoding order shall be available to the decoding process prior to its activation, regardless of whether or not the decoding process is started at the beginning of the bitstream or with the access unit, in decoding order, that is associated with the recovery point SEI message.

recovery_afoc_cnt specifies the recovery point of decoded atlas frames in output order. If there is an atlas frame aFrmA that follows the current atlas frame (i.e., the atlas frame associated with the current SEI message) in decoding order in the CAS and that has AtlasFrmOrderCntVal equal to the AtlasFrmOrderCntVal of the current atlas frame plus the value of recovery_afoc_cnt, the atlas frame aFrmA is referred to as the recovery point atlas frame. Otherwise, the first atlas frame in output order that has AtlasFrmOrderCntVal greater than the AtlasFrmOrderCntVal of the current atlas frame plus the value of recovery_afoc_cnt is referred to as the recovery point atlas frame. The recovery point atlas frame shall not precede the current atlas frame in decoding order. All decoded atlas frames in output order are indicated to be correct or approximately correct in content starting at the output order position of the recovery point atlas 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 atlas 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 atlas frames that would be produced by starting the decoding process at the location of a previous IRAP coded atlas 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 atlas 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 atlas frames that would be produced by starting the decoding process at the location of a previous IRAP coded atlas 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 document.

broken_link_flag indicates the presence or absence of a broken link in the atlas data 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, atlas frames produced by starting the decoding process at the location of a previous IRAP coded atlas access unit may contain undesirable visual artefacts to the extent that decoded atlas 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, atlas frames subsequent to the specified recovery point in output order are specified to be correct or approximately correct in content.

F.3.6 No reconstruction SEI message semantics

The no reconstruction SEI message indicates that the current atlas frame should not be used for reconstruction and that no V3C volumetric frame should be reconstructed and displayed using this atlas frame.

F.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.

reserved_sei_message_payload_byte is a byte reserved for future use by ISO/IEC.

F.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:

a) The indication that certain types of SEI messages are expected (i.e., likely) to be present (although not guaranteed to be present) in the CAS.

b) For each type of SEI message that is indicated as expected (i.e., likely) to be present in the CAS, 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 CAS. However, it is suggested that some SEI messages should typically be considered as necessary.

c) 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 CAS.

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 CAS is suitable for delivery to a receiving and decoding system, based on whether the receiving system can properly process the CAS to enable an adequate user experience or whether the CAS satisfies the application needs.

When an SEI manifest SEI message is present in any access unit of a CAS, an SEI manifest SEI message shall be present in the first access unit of the CAS. The SEI manifest SEI message persists in decoding order from the current access unit until the end of the CAS. When there are multiple SEI manifest SEI messages present in a CAS, 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 F.3.

Table F.3 — 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 CAS.

Table F.3 *(continued)*

Value	Description
1	Indicates that there are SEI messages with payloadType equal to manifest_sei_payload_type[i] expected to be present in the CAS, 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 CAS, 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 CAS, 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 edition of this document. 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.

F.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 CAS 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 CAS is suitable for delivery to a receiving and decoding system, based on whether the receiving system can properly process the CAS to enable an adequate user experience or whether the CAS satisfies the application needs (as determined in some manner by external means outside the scope of this document).

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 CAS, an SEI prefix indication SEI message shall be present in the first access unit of the CAS. The SEI prefix indication SEI message persists in decoding order from the current access unit until the end of the CAS. When there are multiple SEI prefix indication SEI messages present in a CAS 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 CAS, 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 CAS.

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 can 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 document.

byte_alignment_bit_equal_to_one shall be equal to 1.

F.3.10 Active sub-bitstreams SEI message semantics

This SEI message informs the V3C decoder which attributes or maps are not available and can therefore be skipped in the decoding process. Similarly, it can also inform the decoder that auxiliary sub-bitstream are not available. This is done by signalling changes to active attributes/maps and whether the auxiliary sub-bitstreams are active or not. An active sub-bitstreams SEI message references a specific V3C 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 V3C 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 or rendering process, or both.

NOTE The active sub-bitstreams SEI message allows the possibility of enabling only a subset of attributes and subset of maps to be active at any given time.

The persistence scope for this SEI message is the remainder of the bitstream (i.e., the signalled active attributes persist until the end of the stream) or when a new active attributes SEI message is encountered. Previously defined parameters from an earlier SEI message shall persist if not modified and if the value of asb_active_sub_bitstream_cancel_flag is not equal to 1. This SEI message should not be ignored by the decoder.

The semantics of the fields of the active sub-bitstreams SEI message are as follows:

asb_active_sub_bitstream_cancel_flag equal to 1 indicates that all sub-bitstreams are active. asb_active_sub_bitstream_cancel_flag equal to 0 indicates that some sub-bitstreams may not be active.

asb_active_attributes_changes_flag indicates whether there are activation changes for any of the attribute sub-bitstreams of the V3C stream. asb_active_attributes_changes_flag equal to 1 indicates an activation change for at least one attribute and its associated sub-bitstreams. asb_active_attributes_changes_flag equal to 0 indicates no changes.

asb_active_maps_changes_flag indicates whether there are activation changes for any of the V3C map sub-bitstreams. asb_active_maps_changes_flag equal to 1 indicates an activation change for at least one map and its associated sub-bitstreams. asb_active_maps_changes_flag equal to 0 indicates no changes.

asb_auxiliary_sub_bitstreams_active_flag indicates whether the auxiliary sub-bitstreams for the geometry and all attributes are active or not. asb_auxiliary_sub_bitstreams_active_flag equal to 1 indicates that such sub-bitstreams are active. asb_auxiliary_sub_bitstreams_active_flag equal to 0 indicates that such sub-bitstreams are not active.

asb_all_attributes_active_flag indicates whether all the attributes signalled in the referenced VPS shall be active. asb_all_attributes_active_flag equal to 1 indicates that all attributes shall be active. asb_all_attributes_active_flag equal to 0 indicates that only a sub-set of the attributes may be active.

asb_active_attribute_count_minus1 plus 1 indicates the number of active attributes signalled in the active sub-bitstreams SEI message.

asb_active_attribute_idx[i] indicates the attribute index in the VPS for the active attribute at index i in the associated SEI message.

asb_all_maps_active_flag indicates whether all the maps signalled in the referenced VPS shall be active. asb_all_maps_active_flag equal to 1 indicates that all maps shall be active. asb_all_maps_active_flag equal to 0 indicates that only a sub-set of the maps may be active.

asb_active_map_count_minus1 plus 1 indicates the number of active maps signalled in the active sub-bitstreams SEI message.

asb_active_map_idx[i] indicates the map index in the VPS for the active map at index i in the associated SEI message.

F.3.11 Component codec mapping SEI message semantics

This SEI message informs the V3C decoder of the codec mapping for the codec ids of the component sub-bitstreams signalled in the VPS. Each component sub-bitstream codec id is mapped to a specific codec index in a codec lookup table. The codec ids for the component sub-bitstreams in the 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 V3C bitstream as well as signalling updated mappings when the codec of one or more of the V3C components sub-bitstreams changes. A V3C decoder receiving a component codec mapping SEI message should instantiate new video decoders for the respective component sub-bitstreams signalled in the message.

The persistence scope for this SEI message is the remainder of the bitstream (i.e., the codec changes for the signalled 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 and if the value of ccm_component_codec_cancel_flag is not equal to 1. 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 CVS, a component codec mapping SEI message shall be present in the first access unit of the CVS. The component codec mapping SEI message persists in decoding order from the current access unit until the end of the CVS. When there are multiple component codec mapping SEI messages present in a CVS, they shall have the same content.

The semantics of the fields of the component codec mapping SEI message are as follows:

ccm_component_codec_cancel_flag indicates whether the component codec mapping should be reset to the default mapping defined by the CodecGroup profile type for the bitstream. ccm_component_codec_cancel_flag equal to 1 indicates that the component codec mapping should be reset to the default mapping defined by the CodecGroup profile type for the bitstream. ccm_component_codec_cancel_flag equal to 0 indicates that the component codec mapping for some components may be updated.

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 sub-bitstreams in a V3C bitstream, as specified within the active 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 a registered code according to ISO/IEC 14496-12.

F.3.12 Volumetric annotation SEI messages family syntax

F.3.12.1 Scene object information SEI message semantics

F.3.12.1.1 General

This SEI message defines a set of objects that may be present in a volumetric scene, and optionally assigns different properties to these objects. These objects could then potentially be associated with different types of information, including patches and 2D volumetric rectangles that may be defined using the patch information and volumetric rectangle information SEI messages.

Let the variable MaxNumObjectsTracked be set equal to either $2^{16} - 1$ or a value determined by the application.

Let the variable MaxObjectNumDependencies be set equal to 15.

NOTE An application can limit the value of the variable MaxNumObjectsTracked so as to constrain the required memory.

At the start of each sequence,

— the 1D arrays ObjectIndex, ObjectTracked, ObjectLabelIndex, ObjectPriorityValue, ObjectHiddenFlag, ObjectNumDependencies, ObjectVisibilityDirectionX, ObjectVisibilityDirectionY, ObjectVisibilityDirectionZ, ObjectVisibilityAngle, Object3DBoundingBoxX, Object3DBoundingBoxY, Object3DBoundingBoxZ, Object3DBoundingBoxSizeX, Object3DBoundingBoxSizeY, Object3DBoundingBoxSizeZ, ObjectCollisionShapeID, ObjectPointShapeID, ObjectPointSize, and ObjectMaterialID, of size MaxNumObjectsTracked, and

— the 2D array ObjectDependencyIndex, of size MaxNumObjectsTracked × MaxObjectNumDependencies,

are initialized as follows:

```
for ( i = 0; i < MaxNumObjectsTracked; i++ ) {
    ObjectIndex[ i ] = i
    ObjectInit( ObjectIndex[ i ] )
}
```

where the function ObjectInit( ) is defined in the subclause F.3.12.1.2.

soi_persistence_flag specifies the persistence of the scene object information SEI message for the current layer. soi_persistence_flag equal to 0 specifies that the scene object information SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. soi_persistence_flag equal to 1 specifies that the scene object information SEI message persists for the current layer in output order until any of the following conditions are true:

— A new CAS begins.

— The bitstream ends.

— An atlas frame aFrmB in the current layer in a coded atlas access unit containing a scene object information SEI message with the same value of soi_persistence_flag and applicable to the current layer is output for which AtlasFrmOrderCnt(aFrmB) is greater than AtlasFrmOrderCnt(aFrmA), where AtlasFrmOrderCnt(aFrmB) and AtlasFrmOrderCnt(aFrmA) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

soi_reset_flag indicates that all arrays associated with the scene object information SEI message are reset as follows:

```
for ( i = 0; i < MaxNumObjectsTracked; i++ ) {
    ObjectIndex[ i ] = i
    ObjectInit( ObjectIndex[ i ] )
}
```

soi_num_object_updates indicates the number of objects that are to be updated by the current SEI message. The value of soi_num_object_updates shall be in the range from 0 to MaxNumObjectsTracked, inclusive. The default value of soi_num_object_updates is equal to 0.

soi_simple_objects_flag equal to 1 indicates that no additional information for an updated or newly introduced object will be signalled. soi_simple_objects_flag equal to 0 indicates that additional information for an updated or newly introduced object may be signalled.

soi_object_label_present_flag equal to 1 indicates that object label information is present in the current scene object information SEI message. soi_object_label_present_flag equal to 0 indicates that object label information is not present.

soi_priority_present_flag equal to 1 indicates that priority information is present in the current scene object information SEI message. soi_priority_present_flag equal to 0 indicates that priority information is not present.

soi_object_hidden_present_flag equal to 1 indicates that hidden object information is present in the current scene object information SEI message. soi_object_hidden_present_flag equal to 0 indicates that hidden object information is not present.

soi_object_dependency_present_flag equal to 1 indicates that object dependency information is present in the current scene object information SEI message. soi_object_dependency_present_flag equal to 0 indicates that object dependency information is not present.

soi_visibility_cones_present_flag equal to 1 indicates that visibility cones information is present in the current scene object information SEI message. soi_visibility_cones_present_flag equal to 0 indicates that visibility cones information is not present.

soi_3d_bounding_box_present_flag equal to 1 indicates that 3D bounding box information is present in the current scene object information SEI message. soi_3d_bounding_box_present_flag equal to 0 indicates that 3D bounding box information is not present.

soi_collision_shape_present_flag equal to 1 indicates that collision information is present in the current scene object information SEI message. soi_collision_shape_present_flag equal to 0 indicates that collision shape information is not present.

soi_point_style_present_flag equal to 1 indicates that point style information is present in the current scene object information SEI message. soi_point_style_present_flag equal to 0 indicates that point style information is not present.

soi_material_id_present_flag equal to 1 indicates that material ID information is present in the current scene object information SEI message. soi_material_id_present_flag equal to 0 indicates that material ID information is not present.

soi_extension_present_flag equal to 1 indicates that additional extension information shall be present in the current scene object information SEI message. soi_extension_present_flag equal to 0 indicates that additional extension information is not present. It is a requirement of bitstream conformance to this edition of this document that soi_extension_present_flag shall be equal to 0.

soi_3d_bounding_box_scale_log2 indicates the scale to be applied to the 3D bounding box parameters that may be specified for an object.

soi_log2_max_object_idx_updated_minus1 plus 1 specifies the number of bits used to signal the value of an object index in the current scene object information SEI message.

soi_log2_max_object_dependency_idx specifies the number of bits used to signal the value of a dependency object index in the current scene object information SEI message. The default value of soi_log2_max_object_dependency_idx is equal to 0.

soi_object_idx[i] indicates the object index of the i-th object to be updated. The number of bits used to represent soi_object_idx[i] is equal to soi_log2_max_object_idx_updated_minus1 + 1.

soi_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] shall be set to 0. Furthermore, all of its associated parameters, including the object label, 3D bounding box parameters, priority information, hidden flag, dependency information, visibility cones, collision shapes, point style and material id, shall be reset by invoking the function ObjectInit(i). soi_object_cancel_flag[i] equal to 0 indicates that the object with index equal to soi_object_idx[i] shall be updated with information that follows this element and that that the variable ObjectTracked[i] shall be set to 1.

soi_object_label_update_flag[i] equal to 1 indicates that object label update information is present for an object with object index i. soi_object_label_update_flag[i] equal to 0 indicates that object label update information is not present.

soi_object_label_idx[i] indicates the label index, ObjectLabelIndex[i], of an object with index i. The value of soi_object_label_idx[i] shall be in the range from 0 to MaxNumObjectsTracked − 1, inclusive.

soi_priority_update_flag[i] equal to 1 indicates that priority update information is present for an object with object index i. soi_priority_update_flag[i] equal to 0 indicates that object priority information is not present.

soi_priority_value[i] indicates the priority, ObjectPriorityValue[i], of an object with index i. The lower the priority value, the higher the priority. The default value of soi_priority_value[i] is equal to 0.

soi_object_hidden_flag[i] indicates the value of ObjectHiddenFlag[i] of an object with index i. soi_object_hidden_flag[i] equal to 1 indicates that the object with index i shall be hidden. soi_object_hidden_flag[i] equal to 0 indicates that the object with index i shall become present.

soi_object_dependency_update_flag[i] equal to 1 indicates that object dependency update information is present for an object with object index i. soi_object_dependency_update_flag[i] equal to 0 indicates that object dependency update information is not present.

soi_object_num_dependencies[i] indicates the value of ObjectNumDependencies[i] of an object with index i, which specifies the number of object dependencies for the object with index i.

soi_object_dependency_idx[i][j] indicates the value of ObjectDependencyIndex[i][j] of the j-th object that has a dependency with the object with object index i.

soi_visibility_cones_update_flag[i] equal to 1 indicates that visibility cones update information is present for an object with object index i. soi_visibility_cones_update_flag[i] equal to 0 indicates that visibility cones update information is not present.

soi_direction_x[i] specifies the normalized x-component value, ObjectVisibilityDirectionX[i], of the direction vector for the visibility cone of an object with object index i. The value of soi_direction_x[i] shall be in the range of -2^{14} to 2^{14}, inclusive. The default value of soi_direction_x[i] is equal to 2^{14}.

The value of ObjectVisibilityDirectionX[i] is computed as follows:

$$\text{ObjectVisibilityDirectionX[i]} = \text{soi\_direction\_x[i]} \div 2^{14}$$

soi_direction_y[i] specifies the normalized y-component value, ObjectVisibilityDirectionY[i], of the direction vector for the visibility cone of an object with object index i. The value of soi_direction_y[i] shall be in the range of -2^{14} to 2^{14}, inclusive. The default value of soi_direction_y[i] is equal to 2^{14}.

The value of ObjectVisibilityDirectionY[i] is computed as follows:

$$\text{ObjectVisibilityDirectionY[i]} = \text{soi\_direction\_y[i]} \div 2^{14}$$

soi_direction_z[i] specifies the normalized z-component value, ObjectVisibilityDirectionZ[i], of the direction vector for the visibility cone of an object with object index i. The value of soi_direction_z[i] shall be in the range of -2^{14} to 2^{14}, inclusive. The default value of soi_direction_z[i] is equal to 2^{14}.

The value of ObjectVisibilityDirectionZ[i] is computed as follows:

ObjectVisibilityDirectionZ[i] = soi_direction_z[i] ÷ 2^{14}

soi_angle[i] indicates the angle, ObjectVisibilityAngle[i], of the visibility cone along the direction vector in degrees. The value of soi_angle[i] shall be in the range of 0 to $2^{16} - 1$, inclusive. The default value of soi_angle[i] is equal to $2^{16} - 1$.

The value of ObjectVisibilityAngle[i] is computed as follows:

ObjectVisibilityAngle[i] = (soi_angle[i] ÷ ($2^{16} - 1$)) * 180

soi_3d_bounding_box_update_flag[i] equal to 1 indicates that 3D bounding box information is present for an object with object index i. soi_3d_bounding_box_update_flag[i] equal to 0 indicates that 3D bounding box information is not present.

soi_3d_bounding_box_x[i] specifies the quantized x coordinate value of the origin position of the 3D bounding box of an object with index i. The default value of soi_3d_bounding_box_x[i] is equal to 0.

soi_3d_bounding_box_y[i] specifies the quantized y coordinate value of the origin position of the 3D bounding box of an object with index i. The default value of soi_3d_bounding_box_y[i] is equal to 0.

soi_3d_bounding_box_z[i] specifies the quantized z coordinate value of the origin position of the 3D bounding box of an object with index i. The default value of soi_3d_bounding_box_z[i] is equal to 0.

soi_3d_bounding_box_size_x[i] specifies the quantized size of the bounding box on the x axis of an object with index i. The default value of soi_3d_bounding_box_size_x[i] is equal to 0.

soi_3d_bounding_box_size_y[i] specifies the quantized size of the bounding box on the y axis of an object with index i. The default value of soi_3d_bounding_box_size_y[i] is equal to 0.

soi_3d_bounding_box_size_z[i] specifies the quantized size of the bounding box on the z axis of an object with index i. The default value of soi_3d_bounding_box_size_z[i] is equal to 0.

The bounding box origin and size for an object with index i are computed as follows:

```
Object3DBoundingBoxX[ i ] = soi_3d_bounding_box_x[ i ] << soi_3d_bounding_box_scale_log2
Object3DBoundingBoxY[ i ] = soi_3d_bounding_box_y[ i ] << soi_3d_bounding_box_scale_log2
Object3DBoundingBoxZ[ i ] = soi_3d_bounding_box_z[ i ] << soi_3d_bounding_box_scale_log2
Object3DBoundingBoxSizeX[ i ] = soi_3d_bounding_box_size_x[ i ] << soi_3d_bounding_box_scale_log2
Object3DBoundingBoxSizeY[ i ] = soi_3d_bounding_box_size_y[ i ] << soi_3d_bounding_box_scale_log2
Object3DBoundingBoxSizeZ[ i ] = soi_3d_bounding_box_size_z[ i ] << soi_3d_bounding_box_scale_log2
```

soi_collision_shape_update_flag[i] equal to 1 indicates that collision shape update information is present for an object with object index i. soi_collision_shape_update_flag[i] equal to 0 indicates that collision shape update information is not present.

soi_collision_shape_id[i] indicates the collision shape ID, ObjectCollisionShapeID[i], of an object with index i. The collision shape ID is identified through means outside this document. The default value of soi_collision_shape_id[i] is equal to 0.

soi_point_style_update_flag[i] equal to 1 indicates that point style update information is present for an object with object index i. soi_point_style_update_flag[i] equal to 0 indicates that point style update information is not present.

soi_point_shape_id[i] indicates the point shape ID, ObjectPointShapeID[i], of an object with index i. The default value of soi_point_shape_id[i] is equal to 0. The value of soi_point_shape_id[i] shall be in the range of 0 to 2, inclusive in bitstreams conforming to this edition of this document. Other values of soi_point_shape_id[i] are reserved for future use by ISO/IEC. Decoders conforming to this edition of this document shall ignore reserved values of soi_point_shape_id[i]. See Table F.4.

Table F.4 — soi_point_shape_id[i] values

value	Description
0	Circle
1	Square
2	Diamond
3..255	Reserved

soi_point_size[i] indicates the point size, ObjectPointSize[i], of an object with index i. The default value of soi_point_size[i] is equal to 1.

soi_material_id_update_flag[i] equal to 1 indicates that material ID update information is present for an object with object index i. soi_point_style_update_flag[i] equal to 0 indicates that point style update information is not present.

soi_material_id[i] indicates the material ID, ObjectMaterialID[i] of an object with index i. The default value of soi_material_id[i] is equal to 0. The material ID is identified through means outside this document.

F.3.12.1.2 Object initialization

Let the function ObjectInit(k) be defined as follows:

```
ObjectInit( k ) {
    ObjectTracked[ k ] = 0
    ObjectLabelIndex[ k ] = 0
    ObjectPriorityValue[ k ] = 0
    ObjectHiddenFlag[ k ] = 0
    ObjectNumDependencies[ k ] = 0
    for ( j = 0; j < MaxObjectNumDependencies; j++ ) {
        ObjectDependencyIndex[ k ][ j ] = 0
    }
    ObjectVisibilityDirectionX[ k ] = 1
    ObjectVisibilityDirectionY[ k ] = 1
    ObjectVisibilityDirectionZ[ k ] = 1
    ObjectVisibilityAngle[ k ] = 180
    Object3DBoundingBoxX[ k ] = 0
    Object3DBoundingBoxY[ k ] = 0
    Object3DBoundingBoxZ[ k ] = 0
    Object3DBoundingBoxSizeX[ k ] = 0
    Object3DBoundingBoxSizeY[ k ] = 0
    Object3DBoundingBoxSizeZ[ k ] = 0
    ObjectCollisionShapeID[ k ] = 0
    ObjectPointShapeID[ k ] = 0
    ObjectPointSize[ k ] = 0
    ObjectMaterialID[ k ] = 0
}
```

F.3.12.2 Object label information SEI message semantics

F.3.12.2.1 General

This SEI message defines a set of labels that could be associated with objects in a volumetric scene.

Let the variable MaxNumLabels be set equal to either $2^{16} - 1$ or a value determined by the application.

NOTE An application can limit the value of the variable MaxNumLabels so as to constrain the required memory.

At the start of each sequence,

— the variable LabelLanguage, and

— the 1D arrays LabelIndex, LabelSetFlag, and Label,

are initialized as follows:

```
LabelLanguage = ""
for ( i = 0; i < MaxNumLabels; i++ ) {
    LabelIndex[ i ] = i
    LabelInit( LabelIndex[ i ] )
}
```

where the function LabelInit( ) is defined in the subclause F.3.12.2.2.

oli_cancel_flag equal to 1 indicates that the object label information SEI message cancels the persistence of any previous object label information SEI message in output order and sets all variables and arrays associated with the object label SEI message, as follows:

```
LabelLanguage = ""
for ( i = 0; i < MaxNumLabels; i++ ) {
    LabelIndex[ i ] = i
    LabelInit( LabelIndex[ i ] )
}
```

oli_label_language_present_flag equal to 1 indicates that label language information is present in the object label information SEI message. oli_label_language_present_flag equal to 0 indicates that label language information is not present.

oli_bit_equal_to_zero shall be equal to 0.

oli_label_language contains a language tag, LabelLanguage, as specified by IETF RFC 5646 followed by a null termination byte equal to 0x00. The length of the oli_label_language syntax element shall be less than or equal to 255 bytes, not including the null termination byte.

oli_num_label_updates indicates the number of labels that are to be updated by the current SEI message. The value of oli_num_label_updates shall be in the range from 0 to MaxNumLabels, inclusive.

oli_label_idx[i] indicates the label index, LabelIndex[i], of the i-th label to be updated. The value of oli_label_idx[i] shall be in the range from 0 to MaxNumLabels – 1, inclusive.

oli_label_cancel_flag[i] equal to 1 indicates that all label information, for label with index i, shall be reset by invoking the function LabelInit(i). oli_label_cancel_flag[i] equal to 0 indicates that the label with index i shall be updated with information that follows this element and that the variable LabelSetFlag[i] shall be set to 1.

oli_bit_equal_to_zero shall be equal to 0.

oli_label[i] indicates the label, Label[i], of the i-th label. The length of the oli_label[i] syntax element shall be less than or equal to 255 bytes, not including the null termination byte. The default value of oli_label[i] is equal to "".

oli_persistence_flag specifies the persistence of the scene object information SEI message for the current layer. oli_persistence_flag equal to 0 specifies that the object label information SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. oli_persistence_flag equal to 1 specifies that the object label information SEI message persists for the current layer in output order until any of the following conditions are true:

— A new CAS begins.

— The bitstream ends.

— An atlas frame aFrmB in the current layer in a coded atlas access unit containing an object label information SEI message, with either an oli_cancel_flag equal to 1 or the same value of oli_persistence_flag, applicable to the current layer is output for which AtlasFrmOrderCnt(aFrmB) is greater than AtlasFrmOrderCnt(aFrmA), where AtlasFrmOrderCnt(aFrmB) and AtlasFrmOrderCnt(aFrmA) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

F.3.12.2.2 Label initialization

Let the function LabelInit(k) be defined as follows:

```
LabelInit( k ) {
    LabelSetFlag[ k ] = 0
    Label[ k ] = ""
}
```

F.3.12.3 Patch information SEI message semantics

F.3.12.3.1 General

This SEI message defines a table that indicates the set of tiles and corresponding patches that may be present in the current frame. These patches can be assigned to one or more objects that may be present in a volumetric scene and which were previously defined through one or more scene object information SEI messages.

Let the variables MaxNumTiles, MaxNumPatches and MaxNumObjectsPerPatch be set equal to either 1024, $2^{16} - 1$, and $2^{16} - 1$, respectively, or a value determined by the application in bitstreams conforming to this document.

NOTE An application can limit the value of the variable MaxNumTiles, MaxNumPatches and MaxNumObjectsPerPatch so as to constrain the required memory.

At the start of each sequence,

— the 2D array PatchNumObjects, of size MaxNumTiles × MaxNumPatches, and

— the 3D array PatchObjectIndex, of size MaxNumTiles × MaxNumPatches × MaxNumObjectsPerPatch,

are initialized as follows:

```
for ( i = 0; i < MaxNumTiles; i++ ) {
    TileInit( i )
}
```

where the function TileInit( ) is defined in the subclause F.3.12.3.2.

pi_persistence_flag specifies the persistence of the patch information SEI message for the current layer. pi_persistence_flag equal to 0 specifies that the patch information SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. pi_persistence_flag equal to 1 specifies that the patch information SEI message persists for the current layer in output order until any of the following conditions are true:

— A new CAS begins.

— The bitstream ends.

— An atlas frame aFrmB in the current layer in a coded atlas access unit containing a patch information SEI message with the same value of pi_persistence_flag and applicable to the current layer is output for which AtlasFrmOrderCnt(aFrmB) is greater than AtlasFrmOrderCnt(aFrmA), where AtlasFrmOrderCnt(aFrmB) and AtlasFrmOrderCnt(aFrmA) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

pi_reset_flag equal to 1 indicates that all entries in the patch information table shall be removed as follows:

```
for ( i = 0; i < MaxNumTiles; i++ ) {
    TileInit( i )
}
```

pi_num_tile_updates indicates the number of tiles that are to be updated in the patch information table by the current SEI message. The value of pi_num_tile_updates shall be in the range from 0 to MaxNumTiles, inclusive.

pi_log2_max_object_idx_tracked_minus1 plus 1 specifies the number of bits used to signal the value of a tracked object index in the current patch information SEI message.

pi_log2_max_patch_idx_updated_minus1 plus 1 specifies the number of bits used to signal the value of an updated patch index in the current patch information SEI message.

pi_tile_id[i] specifies the tile ID for the i-th updated tile in the current SEI message.

pi_tile_cancel_flag[i] equal to 1 indicates that the tile with index i shall be reset by invoking the function TileInit(i). pi_tile_cancel_flag[i] equal to 0 indicates that all patches previously assigned to the tile with index i will be retained.

pi_num_patch_updates[i] indicates the number of patches that are to be updated by the current SEI message within the tile with index i in the patch information table. The value of pi_num_patch_updates[i] shall be in the range from 0 to MaxNumPatches, inclusive.

pi_patch_idx[tileID][i] indicates the patch index of the i-th patch in tile with tile ID equal to tileID that is to be updated in the patch information table. The number of bits used to represent pi_patch_idx[tileID][i] is equal to pi_log2_max_patch_idx_updated_minus1 + 1.

pi_patch_cancel_flag[tileID][i] equal to 1 indicates that the information related to the patch with index i in tile with tile ID equal to tileID shall be reset from the patch information table by invoking the function PatchInit(tileID, i).

pi_patch_number_of_objects_minus1[tileID][i] plus 1 indicates the number of objects, PatchNumObjects[tileID][i], that are to be associated with the patch with index i in tile with tile ID equal to tileID. The value of pi_patch_number_of_objects_minus1[tileID][i] shall be in the range from 0 to MaxNumObjectsPerPatch, inclusive.

pi_patch_object_idx[tileID][i][k] indicates the k-th object index, PatchObjectIndex[tileID][i][k], that is associated with the i-th patch in tile with tile ID equal to tileID. The number of bits used to represent pi_patch_object_idx[tileID][i][k] is equal to pi_log2_max_object_idx_tracked_minus1 + 1.

F.3.12.3.2 Tile and patch initialization

Let the function PatchInit(i, j) be defined as follows:

```
PatchInit( i, j ) {
    PatchNumObjects[ i ][ j ] = 0
    for ( k = 0; k < MaxNumObjectsPerPatch; k++ ) {
        PatchObjectIndex[ i ][ j ][ k ] = 0
    }
}
```

Then, also the function TileInit(i) be defined as follows:

```
TileInit( i ) {
    for ( j = 0; j < MaxNumPatches; j++ ) {
        PatchInit( i, j )
    }
}
```

F.3.12.4 Volumetric rectangle information SEI message semantics

F.3.12.4.1 General

This SEI message defines a table that indicates a set of volumetric rectangles within an atlas. These volumetric rectangles can be assigned to one or more objects that may be present in a volumetric scene and that were previously defined through one or more scene object information SEI messages.

Let the variables MaxNumRectangles and MaxNumObjectsPerRectangle be set equal to either $2^{16} - 1$ and $2^{16} - 1$, respectively, or a value determined by the application in bitstreams conforming to this document.

NOTE An application can limit the value of the variable MaxNumRectangles and MaxNumObjectsPerRectangle so as to constrain the required memory.

At the start of each sequence,

— the 1D arrays VriBoundingBoxTop, VriBoundingBoxLeft, VriBoundingBoxWidth, VriBoundingBoxHeight and VriNumberOfObjects, of size MaxNumRectangles,

— the 2D array VriObjectIndex, of size MaxNumRectangles × MaxNumObjectsPerRectangle,

are initialized as follows:

```
for ( i = 0; i < MaxNumRectangles; i++ ) {
    RectangleInit( i )
}
```

where the function RectangleInit( ) is defined in the subclause F.3.12.4.2.

vri_persistence_flag specifies the persistence of the volumetric rectangle information SEI message for the current layer. vri_persistence_flag equal to 0 specifies that the volumetric rectangle information SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. vri_persistence_flag equal to 1 specifies that the volumetric rectangle information SEI message persists for the current layer in output order until any of the following conditions are true:

— A new CAS begins.

— The bitstream ends.

— An atlas frame aFrmB in the current layer in a coded atlas access unit containing a volumetric rectangle information SEI message with the same value of vri_persistence_flag and applicable to the current layer is output for which AtlasFrmOrderCnt(aFrmB) is greater than AtlasFrmOrderCnt(aFrmA), where AtlasFrmOrderCnt(aFrmB) and AtlasFrmOrderCnt(aFrmA) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

vri_reset_flag equal to 1 indicates that all entries in the volumetric rectangle information table of the current atlas with atlas ID SeiAtlasID shall be initialized as follows:

```
for ( i = 0; i < MaxNumRectangles; i++ ) {
    RectangleInit( i )
}
```

vri_num_rectangles_updates indicates the number of volumetric rectangles that are to be updated by the current SEI message. The value of vri_num_rectangles_updates shall be in the range from 0 to MaxNumRectangles , inclusive.

vri_log2_max_object_idx_tracked_minus1 plus 1 specifies the number of bits used to signal the value of a tracked object index in the current volumetric rectangle information SEI message.

vri_log2_max_rectangle_idx_updated_minus1 plus 1 specifies the number of bits used to signal the value of an updated volumetric rectangle index in the current volumetric rectangle information SEI message.

vri_rectangle_idx[i] indicates the i-th volumetric rectangle index that is to be updated in the volumetric rectangle information table. The number of bits used to represent vri_rectangle_idx[i] is equal to (vri_log2_max_rectangle_idx_updated_minus1 + 1).

vri_rectangle_cancel_flag[i] equal to 1 indicates that the volumetric rectangle with index i shall be removed from the volumetric rectangle information table by invoking the function RectangleInit(i).

vri_bounding_box_update_flag[i] equal to 1 indicates that 2D bounding box information for the volumetric rectangle with index i should be updated. vri_bounding_box_update_flag[i] equal to 0 indicates that 2D bounding box information for the volumetric rectangle with index i should not be updated.

vri_bounding_box_top[i] indicates the vertical coordinate value, VriBoundingBoxTop[i], of the top-left position of the bounding box of the i-th volumetric rectangle within the current atlas frame. The default value of vri_bounding_box_top[i] is equal to 0.

vri_bounding_box_left[i] indicates the horizonal coordinate value, VriBoundingBoxLeft[i], of the top-left position of the bounding box of the i-th volumetric rectangle within the current atlas frame. The default value of vri_bounding_box_left[i] is equal to 0.

vri_bounding_box_width[i] indicates the width of the bounding box, VriBoundingBoxWidth[i], of the i-th volumetric rectangle. The default value of vri_bounding_box_width[i] is equal to 0.

vri_bounding_box_height[i] indicates the height of the bounding box, VriBoundingBoxHeight[i], of the i-th volumetric rectangle. The default value of vri_bounding_box_height[i] is equal to 0.

vri_rectangle_number_of_objects_minus1[i] plus 1 indicates the number of objects, VriNumberOfObjects[i], that are to be associated with the i-th volumetric rectangle. The value of vri_rectangle_number_of_objects_minus1[i] shall be in the range from 0 to MaxNumObjectsPerRectangle − 1, inclusive.

vri_rectangle_object_idx[i][j] indicates the j-th object index that is associated with the i-th volumetric rectangle, VriObjectIndex[i][j]. The number of bits used to represent vri_rectangle_object_idx[i] is equal to vri_log2_max_object_idx_tracked_minus1 + 1.

F.3.12.4.2 Rectangle initialization

Let the function RectangleInit(i) be defined as follows:

```
RectangleInit( i ) {
    VriBoundingBoxTop[ i ] = 0
```

```
        VriBoundingBoxLeft[ i ] = 0
        VriBoundingBoxWidth[ i ] = 0
        VriBoundingBoxHeight[ i ] = 0
        VriNumberOfObjects[ i ] = 0
        for ( j = 0; j < MaxNumObjectsPerRectangle; j++ ) {
            VriObjectIndex[ i ][ j ] = 0
        }
    }
```

F.3.12.5 Atlas object association SEI message semantics

This SEI message defines a mask that indicates the mapping between objects and atlases. Objects can be contained in one or more atlases that describe a volumetric scene. Objects are defined through scene object information SEI messages.

Let the variable MaxNumObjects be set equal to $2^{16} - 1$ or a value determined by the application in bitstreams conforming to this document.

NOTE An application can limit the value of the variable MaxNumObjects so as to constrain the required memory.

At the start of each sequence,

— the 2D array ObjectInAtlas, of size MaxNumObjects × 64,

are initialized as follows:

```
    for ( i = 0; i < MaxNumObjects; i++ ) {
        for ( j = 0; j < 64; i++ ) {
            ObjectInAtlas[ i ][ j ] = 0
        }
    }
```

aoa_persistence_flag specifies the persistence of the atlas object association SEI message for the current layer. aoa_persistence_flag equal to 0 specifies that the atlas object association SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. aoa_persistence_flag equal to 1 specifies that the atlas object association SEI message persists for the current layer in output order until any of the following conditions are true:

— A new CAS begins.

— The bitstream ends.

— An atlas frame aFrmB in the current layer in a coded atlas access unit containing an atlas object association SEI message with the same value of aoa_persistence_flag and applicable to the current layer is output for which AtlasFrmOrderCnt(aFrmB) is greater than AtlasFrmOrderCnt(aFrmA), where AtlasFrmOrderCnt(aFrmB) and AtlasFrmOrderCnt(aFrmA) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

aoa_reset_flag equal to 1 indicates that all entries in the atlas object association table shall be removed as follows:

```
    for ( i = 0; i < MaxNumObjects; i++ ) {
        for ( j = 0; j < 64; i++ ) {
            ObjectInAtlas[ i ][ j ] = 0
        }
    }
```

aoa_num_atlases_minus1 plus 1 indicates the number of atlases present in the CVS. aoa_num_atlases_minus1 shall be equal to vps_atlas_count_minus1.

aoa_num_updates indicates the number of objects that are to be updated in the object to atlas table by the current SEI message.

aoa_log2_max_object_idx_tracked_minus1 plus 1 specifies the number of bits used to signal the value of an object index in the current atlas object association SEI message.

aoa_atlas_id[j] specifies the atlas ID of the j-th atlas**.**

aoa_object_idx[i] indicates the i-th object index. The number of bits used to represent aoa_object_idx[i] is equal to aoa_log2_max_object_idx_tracked_minus1 + 1.

aoa_object_in_atlas[i][j] indicates the value of ObjectInAtlas[i][j] for atlas ID equal to j and object index equal to i. aoa_object_in_atlas[i][j] equal to 1 indicates that the object with index i is present in atlas with atlas ID equal to j. aoa_object_in_atlas[i][j] equal to 0 indicates that the object with index i is not present in atlas with atlas ID equal to j.

F.3.13 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 coded atlas access unit in decoding order.

The following applies for the buffering period SEI message syntax and semantics:

— The variables CabSize[!NalHrdModeFlag][Htid][i], BitRate[!NalHrdModeFlag][Htid][i] and CabCnt are derived from syntax elements found in the hrd_sub_layer_parameters(!NalHrdModeFlag, Htid) 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 bp_hrd_cab_cnt_minus1[tIdA] and bp_hrd_cab_cnt_minus1[tIdB] 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 coded atlas access unit is an IRAP coded atlas 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 SLNR atlas frame.

  — Otherwise, the coded atlas 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 coded atlas 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 can be desirable (e.g., for random access at an IRAP coded atlas frame or a non-IRAP coded atlas frame or for bitstream splicing).

bp_nal_hrd_params_present_flag equal to 1 specifies that a list of syntax element pairs bp_nal_initial_cab_removal_delay[i][j] and bp_nal_initial_cab_removal_offset[i][j] are present in the buffering period SEI message. bp_nal_hrd_params_present_flag equal to 0 specifies that no syntax element pairs bp_nal_initial_cab_removal_delay[i][j] and bp_nal_initial_cab_removal_offset[i][j] are present in the buffering period SEI message.

It is a requirement of bitstream conformance that the value of bp_nal_hrd_params_present_flag shall be equal to hrd_nal_parameters_present_flag.

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:

  — bp_nal_hrd_params_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 document.

— Otherwise, the value of NalHrdBpPresentFlag is set equal to 0.

bp_acl_hrd_params_present_flag equal to 1 specifies that a list of syntax element pairs bp_acl_initial_cab_removal_delay[i][j] and bp_acl_initial_cab_removal_offset[i][j] are present in the buffering period SEI message. bp_acl_hrd_params_present_flag equal to 0 specifies that no syntax element pairs bp_acl_initial_cab_removal_delay[i][j] and bp_acl_initial_cab_removal_offset[i][j] are present in the buffering period SEI message.

It is a requirement of bitstream conformance that the value of bp_acl_hrd_params_present_flag shall be equal to hrd_acl_parameters_present_flag.

It is a requirement of bitstream conformance that the values of bp_acl_hrd_params_present_flag and bp_nal_hrd_params_present_flag in a buffering period SEI message shall not be both equal to 0.

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:

  — bp_acl_hrd_params_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 document.

— 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:

  — bp_nal_hrd_params_present_flag is present in the bitstream and is equal to 1.

  — bp_acl_hrd_params_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 document.

— Otherwise, the value of CabDabDelaysPresentFlag is set equal to 0.

bp_initial_cab_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the j-th default and alternative initial CAB removal delays and offsets, respectively, for the i-th temporal sub-layer. of the buffering period SEI messages. When not present, the value of bp_initial_cab_removal_delay_length_minus1 is inferred to be equal to 23.

bp_au_cab_removal_delay_length_minus1 plus 1 specifies the length, in bits, of the syntax elements bp_cab_delay_offset in the buffering period SEI message and the syntax elements aft_au_cab_removal_delay_minus1[i] in the atlas frame timing SEI messages in the current buffering period. When not present, the value of bp_au_cab_removal_delay_length_minus1 is inferred to be equal to 23.

bp_dab_output_delay_length_minus1 plus 1 specifies the length, in bits, of the syntax elements bp_dab_delay_offset syntax element in the buffering period SEI message and aft_dab_output_delay[i] in the atlas frame timing SEI message. When not present, the value of bp_dab_output_delay_length_minus1 is inferred to be equal to 23.

bp_irap_cab_params_present_flag equal to 1 specifies the presence of the bp_nal_initial_alt_cab_removal_delay[Htid][i], bp_nal_initial_alt_cab_removal_delay[Htid][i], bp_acl_initial_alt_cab_removal_offset[Htid][i] and bp_acl_initial_alt_cab_removal_offset[Htid][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 frame is neither a CRA atlas frame nor a BLA atlas fame, 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 coded atlas frame associated with the buffering period SEI message when the RASL coded atlas access units associated with the CRA or BLA access unit are not present. The syntax element has a length in bits given by bp_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 coded atlas access unit associated with the buffering period SEI message when the RASL coded atlas access units associated with the CRA or BLA coded atlas access unit are not present. The syntax element has a length in bits given by bp_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 frame is not the first atlas frame in the bitstream in decoding order, let PrevNonDiscardableAtlasFrame be the preceding atlas in decoding order with TemporalID equal to 0 that is not a RASL, RADL or SLNR 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 frame PrevNonDiscardableAtlasFrame.

bp_atlas_cab_removal_delay_delta_minus1 plus 1, when the current atlas is not the first atlas frame in the bitstream in decoding order, specifies a CAB removal delay increment value relative to the nominal CAB removal time of the atlas frame PrevNonDiscardableAtlasFrame. This syntax element has a length in bits given by bp_atlas_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 frame PrevNonDiscardableAtlasFrame is not associated with a buffering period SEI message, the aft_au_cab_removal_delay_minus1[i] of the current atlas shall be equal to the aft_au_cab_removal_delay_minus1[i] of PrevNonDiscardableAtlasFrame plus bp_atlas_cab_removal_delay_delta_minus1+ 1.

— Otherwise, aft_au_cab_removal_delay_minus1[i] 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_au_cab_removal_delay_minus1[i] for the current atlas frame 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 coded atlas frame 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 frame PrevNonDiscardableAtlasFrame.

bp_max_sub_layers_minus1 plus 1 specifies the maximum number of temporal sub-layers. It is a condition of bitstream conformance that for this edition of this document the value of bp_max_sub_layers_minus1 shall be equal to 0.

bp_hrd_cab_cnt_minus1[i] plus 1 specifies the number of alternative CAB specifications in the bitstream of the CAS when Htid is equal to i. The value of bp_hrd_cab_cnt_minus1[i] shall be in the range of 0 to 31, inclusive. When not present, the value of bp_hrd_cab_cnt_minus1[i] is inferred to be equal to 0.

It is a requirement of bitstream conformance that the value of bp_hrd_cab_cnt_minus1[i] shall be the same as the value of hrd_cab_cnt_minus1[i].

bp_nal_initial_cab_removal_delay[i][j] and **bp_nal_initial_alt_cab_removal_delay**[i][j] specify respectively the j-th default and alternative initial CAB removal delay for the NAL HRD in units of a 90 kHz clock of the i-th temporal sub-layer. The length of bp_nal_initial_cab_removal_delay[i][j] and bp_nal_initial_alt_cab_removal_delay[i][j] is bp_cab_initial_removal_delay_length_minus1 + 1 bits. The value of bp_nal_initial_cab_removal_delay[i][j] and bp_nal_initial_alt_cab_removal_delay[i][j] shall not be equal to 0 and shall be less than or equal to 90000 * (CabSize[!NalHrdModeFlag][i][j] ÷ BitRate[!NalHrdModeFlag][i][j]), the time-equivalent of the CAB size in 90 kHz clock units. When not present, the values of bp_nal_initial_cab_removal_delay[i][j] and bp_nal_initial_alt_cab_removal_delay[i][j] are inferred to be equal to 90000 * (CabSize[!NalHrdModeFlag][i][j] ÷ BitRate[!NalHrdModeFlag][i][j]).

bp_nal_initial_cab_removal_offset[i][j] and **bp_nal_initial_alt_cab_removal_offset**[i][j] 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.

bp_acl_initial_cab_removal_delay[i][j] and **bp_acl_initial_alt_cab_removal_delay**[i][j] specify respectively the j-th default and alternative initial CAB removal delay of the i-th temporal sub-layer for the ACL HRD in units of a 90 kHz clock. The length of bp_acl_initial_cab_removal_delay[i][j] and bp_acl_initial_alt_cab_removal_delay[i][j] is bp_cab_initial_removal_delay_length_minus1 + 1 bits. The value of bp_acl_initial_cab_removal_delay[i][j] and bp_acl_initial_alt_cab_removal_delay[i][j] shall not be equal to 0 and shall be less than or equal to 90000 * (CabSize[!NalHrdModeFlag][i][j] ÷ BitRate[!NalHrdModeFlag][i][j]), the time-equivalent of the CAB size in 90 kHz clock units. When not present, the values of bp_acl_initial_cab_removal_delay[i][j] and bp_acl_initial_alt_cab_removal_delay[i][j] are inferred to be equal to 90000 * (CabSize[!NalHrdModeFlag][i][j] ÷ BitRate[!NalHrdModeFlag][i][j]).

bp_nal_initial_cab_removal_offset[i][j] and **bp_nal_initial_alt_cab_removal_offset**[i][j] specify the j-th default and alternative initial CAB removal offset of the i-th temporal sub-layer for the NAL HRD in units of a 90 kHz clock. The length of bp_nal_initial_cab_removal_offset[i][j] and bp_nal_initial_alt_cab_removal_offset[i][j] is bp_cab_initial_removal_delay_length_minus1 + 1 bits. When not present, the values of bp_nal_initial_cab_removal_offset[i][j] and bp_nal_initial_alt_cab_removal_offset[i][j] are inferred to be equal to 0.

Over the entire CAS, for each value pair of i and j, the sum of bp_nal_initial_cab_removal_delay[i][j] and bp_nal_initial_cab_removal_offset[i][j] shall be constant, and the sum of bp_nal_initial_alt_cab_removal_delay[i][j] and bp_nal_initial_alt_cab_removal_offset[i][j] shall be constant.

bp_acl_initial_cab_removal_offset[i][j] and **bp_acl_initial_alt_cab_removal_offset**[i][j] specify the j-th default and alternative initial CAB removal offset of the i-th temporal sub-layer for the ACL HRD in units of a 90 kHz clock. The length of bp_acl_initial_cab_removal_offset[i] and bp_acl_initial_alt_ cab_removal_offset[i][j] is bp_cab_initial_removal_delay_length_minus1 + 1 bits. When not present, the values of bp_acl_initial_cab_removal_offset[i][j] and bp_acl_initial_alt_cab_removal_offset[i][j] are inferred to be equal to 0.

Over the entire CAS, for each value pair of i and j the sum of bp_acl_initial_cab_removal_delay[i][j] and bp_acl_initial_cab_removal_offset[i][j] shall be constant, and the sum of bp_acl_initial_alt_cab_removal_delay[i][j] and bp_acl_initial_alt_cab_removal_offset[i][j] shall be constant.

Encoders should not 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 atlases follows one or more of its associated RADL atlas in decoding order.

F.3.14 Atlas frame timing SEI message semantics

The atlas frame timing SEI message provides CAB removal delay and DAB output delay information for the coded atlas access unit associated with the SEI message.

The following applies for the atlas timing SEI message syntax and semantics:

— The variable CabDabDelaysPresentFlag is derived from syntax elements found in the buffering period SEI message that are 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 1 The syntax of the atlas frame timing SEI message is dependent on the content of the hrd_parameters( ) syntax structures applicable to the operation points to which the atlas frame timing SEI message applies. These hrd_parameters( ) syntax structures are in the ASPS that are active for the coded atlas access unit associated with the atlas frame timing SEI message. When the atlas frame timing SEI message is associated with an IRAP coded atlas access unit with NoOutputBeforeRecoveryFlag 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 coded atlas frames with NoOutputBeforeRecoveryFlag equal to 1 that are not the first atlas in the bitstream in decoding order, the determination that the coded atlas is an IRAP NoOutputBeforeRecoveryFlag equal to 1) does not occur until the decoding of the first coded tile NAL unit of the coded atlas. Since the coded tile NAL unit of the coded atlas follows the atlas frame timing SEI message in NAL unit order, there can be cases in which it is necessary for a decoder to store the RBSP containing the atlas frame timing SEI message until determining the activeASPS for the coded atlas, and then perform the parsing of the atlas frame timing SEI message.

aft_au_cab_removal_delay_minus1[i] plus 1 specifies, for the i-th sub-layer, the number clock ticks between the nominal CAB removal time of the coded atlas access unit associated with the atlas timing SEI message and the preceding coded atlas 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 bp_au_cab_removal_delay_length_minus1 + 1.

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 $2^{\text{bp\_au\_cab\_removal\_delay\_length\_minus1} + 1}$.

  — Let prevAuCabRemovalDelayMinus1 and prevAuCabRemovalDelayMsb be set equal to aft_au_cab_removal_delay_minus1[i] and AuCabRemovalDelayMsb, respectively, of the previous atlas frame in decoding order that has TemporalID equal to 0, that is not a RASL, RADL or SLNR atlas frame, and that is within the same buffering period as the current atlas frame.

  — AuCabRemovalDelayMsb is derived as follows:

```
if( aft_au_cab_removal_delay_minus1[ i ] <= prevAuCabRemovalDelayMinus1 )
    AuCabRemovalDelayMsb = prevAuCabRemovalDelayMsb + maxCabRemovalDelay          (F.1)
else
    AuCabRemovalDelayMsb = prevAuCabRemovalDelayMsb
```

The variable AuCabRemovalDelayVal is derived as follows:

$$\text{AuCabRemovalDelayVal} = \text{AuCabRemovalDelayMsb} + \text{aft\_au\_cab\_removal\_delay\_minus1}[\ i\ ] + 1 \tag{F.2}$$

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[i] is used to compute the DAB output time of the atlas frame, for the i-th sub-layer. 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 2 An atlas frame 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[i] is given in bits by bp_dab_output_delay_length_minus1 + 1. When asps_max_dec_atlas_frame_buffering_minus1 associated with minTid, is equal to 0, where minTid is the minimum of the OpTid values of all operation points the atlas frame timing SEI message applies to, aft_dab_output_delay[i] shall be equal to 0.

The output time derived from the aft_dab_output_delay[i] of any atlas frame that is output from an output timing conforming decoder shall precede the output time derived from the aft_dab_output_delay[i] of all atlas frames 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 AtlasFrmOrderCntVal.

For atlas frames that are not output by the "bumping" process because they precede, in decoding order, an IRAP coded atlas frame with NoOutputBeforeRecoveryFlag equal to 1, the output times derived from aft_dab_output_delay[i] shall be increasing with increasing value of AtlasFrmOrderCntVal relative to all atlas frames within the same CAS.

F.3.15 Viewport SEI messages family semantics

F.3.15.1 Viewport camera parameters SEI message semantics

This SEI message defines viewport camera parameters that can be assigned to a viewport signalled by the viewport position SEI message.

The persistence scope for this SEI message is the remainder of the bitstream or until a new viewport camera parameters 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 vcp_cancel_flag is not equal to 1.

vcp_camera_id contains an identifying number that may be used to identify viewport camera parameters.

vcp_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous viewport camera parameters SEI message in output order with the same value of vcp_camera_id. vcp_cancel_flag equal to 0 indicates that viewport camera parameters information follows.

vcp_persistence_flag specifies the persistence of the viewport camera parameters SEI message for the current layer. vcp_persistence_flag equal to 0 specifies that the viewport camera parameters SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. vcp_persistence_flag equal to 1 specifies that the viewport camera parameters SEI message persists for the current layer in output order until any of the following conditions are true:

— A new CAS begins.

— The bitstream ends.

— An atlas frame aFrmB in the current layer in a coded atlas access unit containing a viewport camera parameters SEI message, with either an vcp_cancel_flag equal to 1 or the same value of vcp_persistence_flag, applicable to the current layer is output for which AtlasFrmOrderCnt(aFrmB) is greater than AtlasFrmOrderCnt(aFrmA), where AtlasFrmOrderCnt(aFrmB) and AtlasFrmOrderCnt(aFrmA) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

vcp_camera_type indicates the projection method of the viewport camera. vcp_camera_type equal to 0 specifies equirectangular projection. vcp_camera_type equal to 1 specifies a perspective projection. vcp_camera_type equal to 2 specifies an orthographic projection. vcp_camera_type values in tbe range of 3 to 7, inclusive, are reserved for future use by ISO/IEC.

vcp_erp_horizontal_fov indicates the horizontal size of the viewport region, in units of 2^{-16} degrees. The value of vcp_erp_horizontal_fov shall be in the range of 0 to 360 * 2^{16} – 1, inclusive.

vcp_erp_vertical_fov indicates the vertical size of the viewport region, in units of 2^{-16} degrees. The value of vcp_erp_vertical_fov shall be in the range of 0 to 180 * 2^{16} – 1, inclusive.

vcp_perspective_aspect_ratio indicates the relative aspect ratio of the viewport for perspective projection (horizontal/vertical).

vcp_perspective_horizontal_fov indicates the horizontal field of view for perspective projection in units of 2^{-16} degrees. The value of vcp_perspective_horizontal_fov shall be in the range of 0 to 180 * 2^{16} – 1, inclusive.

vcp_ortho_aspect_ratio indicates the relative aspect ratio of the viewport for orthogonal projection (horizontal/vertical).

vcp_ortho_horizontal_size indicates the horizontal size of an orthogonal viewport in metres.

vcp_clipping_near_plane indicates the distance of the near clipping plane for a given viewport in metres.

vcp_clipping_far_plane indicates the distance of the far clipping plane for a given viewport in metres.

F.3.15.2 Viewport position SEI message semantics

This SEI message defines a viewport position and orientation. The viewport could then be associated with different types of information such as viewport camera parameters indicated by a viewport camera parameters SEI message.

The persistence scope for this SEI message is the remainder of the bitstream or until a new viewport position 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 vp_cancel_flag is not equal to 1.

vp_viewport_id specifies an identifying number that may be used to identify a viewport.

vp_camera_parameters_present_flag equal to 1 indicates that vp_vcp_camera_id is present in the viewport position SEI message. vp_camera_parameters_present_flag equal to 0 indicates that vp_vcp_camera_id is not present in viewport position SEI message

vp_vcp_camera_id specifies the value of vcp_camera_id of viewport parameter SEI message that should provide the camera information for this viewport. When vp_vcp_camera_id is not present, it shall be inferred to be equal to 0.

vp_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous viewport position SEI message in output order with the same value of vp_view_id. vp_cancel_flag equal to 0 indicates that viewport position information follows.

vp_persistence_flag specifies the persistence of the viewport position SEI message for the current layer. vp_persistence_flag equal to 0 specifies that the viewport position SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. vp_persistence_flag equal to 1 specifies that the viewport position SEI message persists for the current layer in output order until any of the following conditions are true:

— A new CAS begins.

— The bitstream ends.

— An atlas frame aFrmB in the current layer in a coded atlas access unit containing a viewport position SEI message, with either an vp_cancel_flag equal to 1 or the same value of vp_persistence_flag, applicable to the current layer is output for which AtlasFrmOrderCnt(aFrmB) is greater than AtlasFrmOrderCnt(aFrmA), where AtlasFrmOrderCnt(aFrmB) and AtlasFrmOrderCnt(aFrmA) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

vp_position[d] indicates a position of a viewport along the d axis. When vp_position[d] is not present, it shall be inferred to be equal to represent a view position equal to (0.0, 0.0, 0.0). The values of d equal to 0, 1 and 2 correspond to the X, Y and Z axis, respectively.

vp_rotation_qx specifies the x component, qX, for the rotation of the viewport region using the quaternion representation. The value of vp_rotation_qx shall be in the range of -2^{14} to 2^{14}, inclusive. When vp_rotation_qx is not present, its value shall be inferred to be equal to 0. The value of qX is computed as follows:

$$qX = vp\_rotation\_qx \div 2^{14}$$

vp_rotation_qy specifies the y component, qY, for the rotation of the viewport region using the quaternion representation. The value of vp_rotation_qy shall be in the range of 0 to -2^{14} to 2^{14}, inclusive. When vp_rotation_qy is not present, its value shall be inferred to be equal to 0. The value of qY is computed as follows:

$$qY = vp\_rotation\_qy \div 2^{14}$$

vp_rotation_qz specifies the z component, qZ, for the rotation of the viewport region using the quaternion representation. The value of vp_rotation_qz shall be in the range of 0 to -2^{14} to 2^{14}, inclusive. When vp_rotation_qz is not present, its value shall be inferred to be equal to 0. The value of qZ is computed as follows:

$$qZ = vp\_rotation\_qz \div 2^{14}$$

The fourth component, qW, for the rotation of the current camera model using the quaternion representation is calculated as follows:

$$qW = Sqrt(1 - (qX^2 + qY^2 + qZ^2))$$

NOTE In the context of this document, qW is always positive. If a negative qW is desired, all three syntax elements (vp_rotation_qx, vp_rotation_qy and vp_rotation_qz) can be signalled with an opposite sign, which is equivalent.

A unit quaternion can be represented as a rotation matrix vpRotMatrix as follows:

$$\text{vpRotMatrix} = \begin{bmatrix} 1-2*\left(qY^2+qZ^2\right) & 2*(qX*qY-qZ*qW) & 2*(qX*qZ+qY*qW) & 0 \\ 2*(qX*qY+qZ*qW) & 1-2*\left(qX^2+qZ^2\right) & 2*(qY*qZ-qX*qW) & 0 \\ 2*(qX*qZ-qY*qW) & 2*(qY*qZ+qX*qW) & 1-2*\left(qX^2+qY^2\right) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It is a requirement of bitstream conformance that $qX^2 + qY^2 + qZ^2 <= 1$.

vp_center_view_flag equal to 1 indicates that the viewport position signalled correspond to the center of the viewport. vp_center_view_flag equal to 0 indicates that the viewport position signalled correspond to one of two stereo positions of the viewport.

vp_left_view_flag equal to 1 indicates that the viewport parameters signalled correspond to the left stereo position of the viewport. vp_left_view_flag equal to 0 indicates that the viewport position signalled correspond to the right stereo positions of the viewport.

F.3.16 Decoded atlas information hash SEI message semantics

F.3.16.1 General

This SEI message defines hash values for decoded atlas information that are associated with the current atlas SeiAtlasID. Prior to computing the hash, the decoded atlas information and corresponding derived data, such as the block to patch information, are arranged into strings of bytes as defined in subclause F.3.16.2, and the hash values are calculated as specified in the semantics of the corresponding syntax elements.

The persistence scope for this SEI message is the remainder of the bitstream or until a new decoded atlas information hash 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.

daih_cancel_flag equal to 1 indicates that the SEI message cancels the persistence of any previous decoded atlas information hash SEI message in coding order that applies to the atlas. daih_cancel_flag equal to 0 indicates that decoded atlas information hash follows.

daih_persistence_flag specifies the persistence of the decoded atlas information hash SEI message. daih_persistence_flag equal to 0 specifies that the decoded atlas information hash SEI message applies to the current decoded access unit only.

Let aFrmA be the current atlas frame. daih_persistence_flag equal to 1 specifies that the decoded atlas information hash SEI message persists for the current layer in output order until any of the following conditions are true:

— A new CAS begins.

— The bitstream ends.

— An atlas frame aFrmB in the current layer in a coded atlas access unit containing a decoded atlas information hash SEI message, with either an daih_cancel_flag equal to 1 or the same value of daih_persistence_flag, applicable to the current layer is output for which AtlasFrmOrderCnt(aFrmB) is greater than AtlasFrmOrderCnt(aFrmA), where AtlasFrmOrderCnt(aFrmB) and AtlasFrmOrderCnt(aFrmA) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

daih_hash_type indicates the method used to calculate the checksum according to Table F.5. Values of daih_hash_type that are not listed in Table F.5 are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this edition of this document. Decoders shall ignore decoded atlas information hash SEI messages that contain reserved values of hash_type.

Table F.5 — Atlas hash type

hash_type	Method
0	MD5 (RFC 1321)
1	CRC
2	Checksum

daih_decoded_high_level_hash_present_flag equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the high-level syntax elements associated with the atlas, with atlas ID equal to SeiAtlasID, is present. daih_decoded_high_level_hash_present_flag equal to 0 specifies the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the high-level syntax elements associated with the atlas, with atlas ID equal to SeiAtlasID, will not be present.

daih_decoded_atlas_hash_present_flag equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the patch information associated with the atlas, with atlas ID equal to SeiAtlasID, is present. daih_decoded_atlas_hash_present_flag equal to 0 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the patch information associated with the atlas, with atlas ID equal to SeiAtlasID, will not be present.

daih_decoded_atlas_b2p_hash_present_flag equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the block to patch map associated with the atlas, with atlas ID equal to SeiAtlasID, is present. daih_decoded_atlas_b2p_hash_present_flag equal to 0 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the block to patch map associated with the atlas, with atlas ID equal to SeiAtlasID, will not be present.

daih_decoded_atlas_tiles_hash_present_flag equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum values for the patch information associated with the tiles in the atlas, with atlas ID equal to SeiAtlasID, are present. daih_decoded_atlas_tiles_hash_present_flag equal to 0 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum values for the patch information associated with the tiles in the atlas, with atlas ID equal to SeiAtlasID, will not be present.

daih_decoded_atlas_tiles_b2p_hash_present_flag equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum values for the block to patch map associated with the tiles in the atlas, with atlas ID SeiAtlasID, are present. daih_atlas_tiles_b2p_hash_present_flag equal to 0 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum values for the block to patch map associated with the tiles in the atlas, with atlas ID SeiAtlasID, will not be present.

daih_reserved_1bit shall be equal to 0 in bitstreams conforming to this edition of this document. Other values for daih_reserved_1bit are reserved for future use by ISO/IEC. Decoders shall ignore the value of daih_reserved_1bit.

daih_high_level_md5[i] is the i-th byte of the 16-byte MD5 hash of the high-level syntax elements of the decoded atlas associated with SeiAtlasID. The value of daih_high_level_md5[i] shall be equal to the value digestVal[i], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes HighLevelAtlasData with length HighLevelAtlasDataBytes, as specified in subclause F.3.16.2.2:

```
MD5Init( context )
MD5Update( context, HighLevelAtlasData, HighLevelAtlasDataBytes )
MD5Final( digestVal, context )
```

daih_high_level_crc is the cyclic redundancy check (CRC) of the high-level syntax elements of the decoded atlas associated with SeiAtlasID. The value of daih_high_level_crc shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes HighLevelAtlasData with length HighLevelAtlasDataBytes, as specified in subclause F.3.16.2.2:

```
crc = 0xFFFF
HighLevelAtlasData[ HighLevelAtlasDataBytes ] = 0
HighLevelAtlasData[ HighLevelAtlasDataBytes + 1 ] = 0
for( bitIdx = 0; bitIdx < ( HighLevelAtlasDataBytes + 2 ) * 8; bitIdx++ ) {
        dataByte = HighLevelAtlasData[ bitIdx >> 3 ]
        crcMsb = ( crc >> 15 ) & 1
        bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1
        crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb * 0x1021 )
    }
    crcVal = crc
```

daih_high_level_checksum is the checksum of the high-level syntax elements of the decoded atlas associated with SeiAtlasID. The value of daih_high_level_checksum shall be equal to the value of checksumVal, obtained using the string of bytes HighLevelAtlasData with length HighLevelAtlasDataBytes, as specified in subclause F.3.16.2.2:

```
    checksum = 0
    for( i = 0; i < HighLevelAtlasDataBytes ; i++ ) {
        xorMask = ( i & 0xFF ) ^ ( i >> 8 )
        checksum = ( checksum + ( HighLevelAtlasData[ i ] & 0xFF) ^ xorMask) & 0xFFFFFFFF
        checksum = ( checksum + ( HighLevelAtlasData[ i ] >> 8) ^ xorMask) & 0xFFFFFFFF
    }
```

daih_atlas_md5[i] is the i-th byte of the 16-byte MD5 hash of the decoded atlas associated with SeiAtlasID. The value of daih_atlas_md5[i] shall be equal to the value of digestVal[i], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes AtlasData with length AtlasDataBytes, as specified in subclause F.3.16.2.3:

```
    MD5Init( context )
    MD5Update( context, AtlasData, AtlasDataBytes )
    MD5Final( digestVal, context )
```

daih_atlas_crc is the cyclic redundancy check (CRC) of the decoded atlas associated with SeiAtlasID. The value of daih_atlas_crc shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes AtlasData with length AtlasDataBytes, as specified in subclause F.3.16.2.3:

```
    crc = 0xFFFF
    AtlasData[ AtlasDataBytes ] = 0
    AtlasData[ AtlasDataBytes + 1 ] = 0
    for( bitIdx = 0; bitIdx < ( AtlasDataBytes + 2 ) * 8; bitIdx++ ) {
        dataByte = AtlasData[ bitIdx >> 3 ]
        crcMsb = ( crc >> 15 ) & 1
        bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1
        crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb * 0x1021 )
    }
    crcVal = crc
```

daih_atlas_checksum is the checksum of the decoded atlas associated with SeiAtlasID. The value of daih_atlas_checksum shall be equal to the value of checksumVal, obtained using the string of bytes AtlasData with length AtlasDataBytes, as specified in subclause F.3.16.2.3:

```
    checksum = 0
    for( i = 0; i < AtlasDataBytes ; i++ ) {
        xorMask = ( i & 0xFF ) ^ ( i >> 8 )
        checksum = ( checksum + ( AtlasData[ i ] & 0xFF ) ^ xorMask ) & 0xFFFFFFFF
        checksum = ( checksum + ( AtlasData[ i ] >> 8 ) ^ xorMask ) & 0xFFFFFFFF
    }
```

daih_atlas_b2p_md5[i] is the i-th byte of the 16-byte MD5 hash of the block to patch map of the atlas associated with SeiAtlasID. The value of daih_atlas_b2p_md5[i] shall be equal to the value digestVal[i], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes AtlasB2PData with length AtlasB2PDataBytes, as specified in subclause F.3.16.2.5:

```
MD5Init( context )
MD5Update( context, AtlasB2PData, AtlasB2PDataBytes )
MD5Final( digestVal, context )
```

daih_atlas_b2p_crc is the cyclic redundancy check (CRC) of the block to patch map of the atlas associated with SeiAtlasID. The value of daih_atlas_b2p_crc shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes AtlasB2PData with length AtlasB2PDataBytes, as specified in subclause F.3.16.2.5:

```
crc = 0xFFFF
AtlasB2PData[ AtlasB2PDataBytes ] = 0
AtlasB2PData[ AtlasB2PDataBytes + 1 ] = 0
for( bitIdx = 0; bitIdx < ( AtlasB2PDataBytes + 2 ) * 8; bitIdx++ ) {
    dataByte = AtlasB2PData[ bitIdx >> 3 ]
    crcMsb = ( crc >> 15 ) & 1
    bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1
    crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb * 0x1021 )
}
crcVal = crc
```

daih_atlas_b2p_checksum is the checksum of the block to patch map of the atlas associated with SeiAtlasID. The value of daih_atlas_b2p_checksum shall be equal to the value of checksumVal, obtained using the string of bytes AtlasB2PData with length AtlasB2PDataBytes , as specified in subclause F.3.16.2.5:

```
checksum =0
for( i = 0; i < AtlasB2PDataBytes ; i++ ) {
    xorMask = ( i & 0xFF ) ^ ( i >> 8 )
    checksum = ( checksum + ( AtlasB2PData[ i ] & 0xFF ) ^ xorMask ) & 0xFFFFFFFF
    checksum = ( checksum + ( AtlasB2PData[ i ] >> 8) ^ xorMask ) & 0xFFFFFFFF
}
```

daih_num_tiles_minus1 plus 1 specifies the number of tiles for which hash values will be signalled. The value of daih_num_tiles_minus1 should be in the range of 0 to afti_num_tiles_in_atlas_frame_minus1, inclusive.

daih_tile_id_len_minus1 plus 1 specifies the number of bits used to represent the syntax element daih_tile_id[t]. The value of daih_tile_id_len_minus1 shall be in the range of 0 to 15, inclusive.

daih_tile_id[t] specifies the tile ID of the t-th tile. The length of the daih_tile_id[t] syntax element is daih_tile_id_len_minus1 + 1 bits. When not present, the value of daih_tile_id[t] is inferred to be equal to t. It is a requirement of bitstream conformance that daih_tile_id[t] should not be equal to daih_tile_id[j] for all t != j.

daih_atlas_tiles_md5[t][i] is the i-th byte of the 16-byte MD5 hash of the decoded atlas tile with tile ID equal to t, associated with SeiAtlasID. The value of daih_atlas_tiles_md5[t][i] shall be equal to the value digestVal[i], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes TileData with length TileDataBytes, as specified in subclause F.3.16.2.4:

```
MD5Init( context )
MD5Update( context, TileData, TileDataBytes )
MD5Final( digestVal, context )
```

daih_atlas_tiles_crc[t] is the cyclic redundancy check (CRC) of the decoded atlas tile with tile ID equal to t, associated with SeiAtlasID. The value of daih_atlas_tiles_crc[t] shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes TileData with length TileDataBytes, as specified in subclause F.3.16.2.4:

```
crc = 0xFFFF
TileData[ TileDataBytes ] = 0
TileData[ TileDataBytes + 1 ] = 0
```

```
for( bitIdx = 0; bitIdx < ( TileDataBytes + 2 ) * 8; bitIdx++ ) {
    dataByte = TileData[ bitIdx >> 3 ]
    crcMsb = ( crc >> 15 ) & 1
    bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1
    crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb * 0x1021 )
}
crcVal = crc
```

daih_atlas_tiles_checksum[t] is the checksum of the decoded atlas tile with tile ID equal to t, associated with SeiAtlasID. The value of daih_atlas_tile_checksum[t] shall be equal to the value of checksumVal, obtained using the string of bytes TileData with length TileDataBytes, as specified in subclause F.3.16.2.4:

```
checksum = 0
for( i = 0; i < TileDataBytes ; i++ ) {
    xorMask = ( i & 0xFF ) ^ ( i >> 8 )
    checksum = ( checksum + ( TileData[ i ] & 0xFF) ^ xorMask ) & 0xFFFFFFFF
    checksum = ( checksum + ( TileData[ i ] >> 8 ) ^ xorMask ) & 0xFFFFFFFF
}
```

daih_atlas_tiles_b2p_md5[t][i] is the i-th byte of the 16-byte MD5 hash of the block to patch map of the decoded atlas tile, with tile ID equal to t, associated with SeiAtlasID. The value of daih_atlas_tiles_b2p_md5[t][i] shall be equal to the value digestVal[i], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes TileB2PData with length TileB2PDataBytes, as specified in subclause F.3.16.2.6:

```
MD5Init( context )
MD5Update( context, TileB2PData, TileB2PDataBytes )
MD5Final( digestVal, context )
```

daih_atlas_tiles_b2p_crc[t] is the cyclic redundancy check (CRC) of the block to patch map of the decoded atlas tile, with tile ID equal to t, associated with SeiAtlasID. The value of daih_atlas_tiles_b2p_crc[t] shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes TileB2PData with length TileB2PDataBytes, as specified in subclause F.3.16.2.6:

```
crc = 0xFFFF
TileB2PData[ TileB2PDataBytes ] = 0
TileB2pData[ TileB2PDataBytes + 1 ] = 0
for( bitIdx = 0; bitIdx < ( TileB2PDataBytes + 2 ) * 8; bitIdx++ ) {
    dataByte = TileB2PData[ bitIdx >> 3 ]
    crcMsb = ( crc >> 15 ) & 1
    bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1
    crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb * 0x1021 )
}
crcVal = crc
```

daih_atlas_tiles_b2p_checksum[t] is the checksum of the block to patch map of the decoded atlas tile, with tile ID equal to t, associated with SeiAtlasID. The value of daih_atlas_tile_checksum[t] shall be equal to the value of checksumVal, obtained using the string of bytes TileB2PData with length TileB2PDataBytes, as specified in subclause F.3.16.2.6:

```
checksum = 0
for( i = 0; i < TileB2PDataBytes ; i++ ) {
    xorMask = ( i & 0xFF ) ^ ( i >> 8 )
    checksum = ( checksum + ( TileB2PData[ i ] & 0xFF) ^ xorMask ) & 0xFFFFFFFF
    checksum = ( checksum + ( TileB2PData[ i ] >> 8) ^ xorMask ) & 0xFFFFFFFF
}
```

F.3.16.2 Strings of bytes arrangement for hash calculation

F.3.16.2.1 General

The string of bytes for the high-level syntax, the atlas patch data and for each tile patch data that is associated with the atlas are generated as a string of unsigned 16 bit data as specified in subclauses F.3.16.2.2, F.3.16.2.3 and F.3.16.2.4. Similarly, the block to patch string of bytes for the atlas and for each tile that is associated with the atlas are generated as string of unsigned 16 bit data as specified in subclauses F.3.16.2.5 and F.3.16.2.6, respectively.

F.3.16.2.2 High-level atlas information string of bytes

In this subclause, using ASPS and AFPS parameters, as described in subclauses 8.4.6.1 and 8.4.6.2, a 1D array, HighLevelAtlasData, and a variable, HighLevelAtlasDataBytes, providing the length of the array in bytes, are derived and set as output of this process as follows:

```
HighLevelAtlasDataBytes = 0
HighLevelAtlasDataBytes =
        ASPSCommonByteString( HighLevelAtlasData, HighLevelAtlasDataBytes)
HighLevelAtlasDataBytes =
        ASPSApplicationByteString( HighLevelAtlasData, HighLevelAtlasDataBytes)
HighLevelAtlasDataBytes =
        AFPSCommonByteString( HighLevelAtlasData, HighLevelAtlasDataBytes)
HighLevelAtlasDataBytes =
        AFPSApplicationByteString( HighLevelAtlasData, HighLevelAtlasDataBytes)
```

where the functions ASPSCommonByteString, ASPSApplicationByteString, AFPSCommonByteString, and AFPSApplicationByteString are defined in subclauses F.3.16.2.2.1, F.3.16.2.2.2, F.3.16.2.2.3 and F.3.16.2.2.4, respectively.

F.3.16.2.2.1 Common ASPS level string of bytes

Let the function ASPSCommonByteString(stringByte, posByte) be defined as follows:

```
ASPSCommonByteString( stringByte, posByte ) {
    stringByte[ posByte++ ] = asps_frame_width && 0xFF
    stringByte[ posByte++ ] = (asps_frame_width >> 8 ) && 0xFF
    stringByte[ posByte++ ] = (asps_frame_width >> 16 ) && 0xFF
    stringByte[ posByte++ ] = (asps_frame_width >> 24 ) && 0xFF
    stringByte[ posByte++ ] = asps_frame_height && 0xFF
    stringByte[ posByte++ ] = (asps_frame_height >> 8 ) && 0xFF
    stringByte[ posByte++ ] = (asps_frame_height >> 16 ) && 0xFF
    stringByte[ posByte++ ] = (asps_frame_height >> 24 ) && 0xFF
    stringByte[ posByte++ ] = asps_geometry_3d_bit_depth_minus1 && 0xFF
    stringByte[ posByte++ ] = asps_geometry_2d_bit_depth_minus1 && 0xFF
    stringByte[ posByte++ ] = asps_map_count_minus1 && 0xFF
    stringByte[ posByte++ ] = asps_max_number_projections_minus1 && 0xFF
    stringByte[ posByte++ ] = asps_patch_precedence_order_flag && 0xFF
    return posByte
}
```

F.3.16.2.2.2 Application-based ASPS level string of bytes

Let the function ASPSApplicationByteString(stringByte, posByte) be defined as follows:

```
ASPSApplicationByteString( stringByte, posByte ) {
    if( asps_pixel_deinterleaving_enabled_flag ) {
        for( j = 0; j <= asps_map_count_minus1; j++ )
            stringByte[ posByte++ ] = asps_map_pixel_deinterleaving_flag[ j ] && 0xFF
        }

        stringByte[ posByte++ ] = asps_raw_patch_enabled_flag && 0xFF
        stringByte[ posByte++ ] = asps_eom_patch_enabled_flag && 0xFF
        if( asps_eom_patch_enabled_flag && asps_map_count_minus1 == 0 ) {
            stringByte[ posByte++ ] = asps_eom_fix_bit_count_minus1 && 0xFF
        }

        if( ( asps_auxiliary_video_enabled_flag ) &&
            ( asps_raw_patch_enabled_flag || asps_eom_patch_enabled_flag ) ) {
            stringByte[ posByte++ ] = AuxVideoWidthNF && 0xFF
            stringByte[ posByte++ ] = (AuxVideoWidthNF >> 8 ) && 0xFF
            stringByte[ posByte++ ] = AuxVideoHeightNF && 0xFF
            stringByte[ posByte++ ] = ( AuxVideoHeightNF >> 8 ) && 0xFF
        }

        stringByte[ posByte++ ] = asps_plr_enabled_flag && 0xFF
        if( asps_plr_enabled_flag ) {
            for( i = 0; i < asps_map_count_minus1 + 1; i++) {
                stringByte[ posByte++ ] = plri_map_present_flag[ i ] && 0xFF
                if( plri_map_present_flag[ i ] ) {
                    stringByte[ posByte++ ] = plri_number_of_modes_minus1[ i ] && 0xFF
                    for( j = 0; j < plri_number_of_modes_minus1[ i ] + 1; j++ ) {
                        stringByte[ posByte++ ] = plri_interpolate_flag[ i ][ j ] && 0xFF
                        stringByte[ posByte++ ] = plri_filling_flag[ i ][ j ] && 0xFF
                        stringByte[ posByte++ ] = plri_minimum_depth[ i ][ j ] && 0xFF
                        stringByte[ posByte++ ] = plri_neighbour_minus1[ i ][ j ] && 0xFF
                    }
                    stringByte[ posByte++ ] = plri_block_threshold_per_patch_minus1[ i ] && 0xFF
                }
            }
        }
        return posByte
    }
```

F.3.16.2.2.3 Common AFPS level string of bytes

Let the function AFPSCommonByteString(stringByte, posByte) be defined as follows:

```
    AFPSCommonByteString( stringByte, posByte ) {
        stringByte[ posByte++ ] = afti_num_tiles_in_atlas_frame_minus1 && 0xFF
        for ( i = 1; i < afti_num_tiles_in_atlas_frame_minus1 + 1; i++ ) {
            stringByte[ posByte++ ] = TileOffsetX[ i ] && 0xFF
            stringByte[ posByte++ ] = (TileOffsetX[ i ] >> 8 )&& 0xFF
            stringByte[ posByte++ ] = TileOffsetY[ i ] && 0xFF
            stringByte[ posByte++ ] = (TileOffsetY[ i ] >> 8 )&& 0xFF
            stringByte[ posByte++ ] = TileWidth[ i ] && 0xFF
            stringByte[ posByte++ ] = (TileWidth[ i ] >> 8 )&& 0xFF
            stringByte[ posByte++ ] = TileHeight[ i ] && 0xFF
            stringByte[ posByte++ ] = (TileHeight[ i ] >> 8 )&& 0xFF
            stringByte[ posByte++ ] = AuxTileOffset[ i ] && 0xFF
            stringByte[ posByte++ ] = (AuxTileOffset[ i ] >> 8 )&& 0xFF
            stringByte[ posByte++ ] = AuxTileHeight[ i ] && 0xFF
            stringByte[ posByte++ ] = (AuxTileHeight[ i ] >> 8 )&& 0xFF
            stringByte[ posByte++ ] = afti_tile_id[ i ] && 0xFF
            stringByte[ posByte++ ] = (afti_tile_id[ i ] >> 8 )&& 0xFF
        }
        return posByte
    }
```

F.3.16.2.2.4 Application-based AFPS level string of bytes

Let the function AFPSApplicationByteString(stringByte, posByte) be defined as follows:

```
    AFPSApplicationByteString( stringByte, posByte ) {
        return posByte
    }
```

F.3.16.2.3 Atlas information string of bytes

In this subclause, using the atlas patch level information parameters, as described in subclause 9.2.7, an array, AtlasData, and a variable, AtlasDataBytes, providing the length of the array in bytes, are derived and set as output of this process as follows:

```
    AtlasDataBytes = 0
    for( atlasPatchIdx = 0; atlasPatchIdx < AtlasTotalNumPatches ; atlasPatchIdx++ ) {
        AtlasDataBytes = AtlasPatchCommonByteString( AtlasData, AtlasDataBytes, atlasPatchIdx )
        AtlasDataBytes = AtlasPatchApplicationByteString( AtlasData, AtlasDataBytes, atlasPatchIdx )
    }
```

where the functions AtlasPatchCommonByteString and AtlasPatchApplicationByteString are defined in subclauses F.3.16.2.3.1 and F.3.16.2.3.2, respectively.

F.3.16.2.3.1 Common atlas patch string of bytes

Let the function AtlasPatchCommonByteString(stringByte, posByte, p) be defined as follows:

```
    AtlasPatchCommonByteString( stringByte, posByte, p ) {
        stringByte[ posByte++ ] = AtlasPatchType[ p ] & 0xFF

        stringByte[ posByte++ ] = AtlasPatch2dPosX[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch2dPosX[ p ] >> 8 ) & 0xFF
        stringByte[ posByte++ ] = AtlasPatch2dPosY[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch2dPosY[ p ] >> 8 ) & 0xFF

        stringByte[ posByte++ ] = AtlasPatch2dSizeX[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch2dSizeX[ p ] >> 8 ) & 0xFF
        stringByte[ posByte++ ] = AtlasPatch2dSizeY[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch2dSizeY[ p ] >> 8 ) & 0xFF

        stringByte[ posByte++ ] = AtlasPatch3dOffsetU[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch3dOffsetU[ p ] >> 8 ) & 0xFF
        stringByte[ posByte++ ] = AtlasPatch3dOffsetV[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch3dOffsetV[ p ] >> 8 ) & 0xFF
        stringByte[ posByte++ ] = AtlasPatch3dOffsetV[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch3dOffsetV[ p ] >> 8 ) & 0xFF
        stringByte[ posByte++ ] = AtlasPatch3dOffsetD[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch3dOffsetD[ p ] >> 8 ) & 0xFF
        stringByte[ posByte++ ] = AtlasPatch3dRangeD[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatch3dRangeD[ p ] >> 8 ) & 0xFF

        stringByte[ posByte++ ] = AtlasPatchProjectionID[ p ] & 0xFF
        stringByte[ posByte++ ] = AtlasPatchOrientationIndex[ p ] & 0xFF

        stringByte[ posByte++ ] = AtlasPatchLoDScaleX[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatchLoDScaleX[ p ] >> 8 ) & 0xFF
        stringByte[ posByte++ ] = AtlasPatchLoDScaleY[ p ] & 0xFF
        stringByte[ posByte++ ] = ( AtlasPatchLoDScaleY[ p ] >> 8 ) & 0xFF

        return posByte
    }
```

F.3.16.2.3.2 Application-based atlas patch string of bytes

Let the function AtlasPatchApplicationByteString(stringByte, posByte, p) be defined as follows:

```
    AtlasPatchApplicationByteString( stringByte, posByte, p ) {
        stringByte[ posByte++ ] = AtlasPatchInAuxVideo[ p ] & 0xFF
        if( AtlasPatchType[ p ] == RAW ) {
            stringByte[ posByte++ ] = AtlasPatchRawPoints[ p ] & 0xFF
            stringByte[ posByte++ ] = ( AtlasPatchRawPoints[ p ] >> 8 ) & 0xFF
        } else if( AtlasPatchType[ p ] == PROJECTED ) {
            blockCnt = BlockCnt( AtlasPatch2dSizeX[ p ], AtlasPatch2dSizeY[ p ] )
            for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {
                if( plri_map_present_flag[ m ] == 1 ) {
                    if(AtlasPlrdLevel[ p ][ m ] == 0 ) {
                        for( j = 0; j < blockCnt; j++ ) {
                            stringByte[ posByte++ ] =
                                AtlasPlrdBlockModeMinus1[ p ][ m ][ j ] & 0xFF
                            stringByte[ posByte++ ] =
                                (AtlasPlrdBlockModeMinus1[ p ][ m ][ j ] >> 8 ) & 0xFF
                        }
                    } else {
                        stringByte[ posByte++ ] = AtlasPlrdModeMinus1[ p ][ m ] & 0xFF
                        stringByte[ posByte++ ] =
                            (AtlasPlrdBlockModeMinus1[ p ][ m ] >> 8 ) & 0xFF
                    }
                }
            }
        } else if( AtlasPatchType[ p ] == EOM ) {
            stringByte[ posByte++ ] = AtlasPatchEomPatchCount[ p ] & 0xFF
            stringByte[ posByte++ ] = ( AtlasPatchEomPatchCount[ p ] >> 8 ) & 0xFF
            for( i = 0; i < AtlasPatchEomPatchCount[ p ]; i++ ) {
                stringByte[ posByte++ ] = AtlasPatchEomPoints[ p ][ i ] & 0xFF
                stringByte[ posByte++ ] = (AtlasPatchEomPoints[ p ][ i ] >> 8 ) & 0xFF
            }
        }
        return posByte
    }
```

F.3.16.2.4 Tile information string of bytes

In this subclause, using tile patch information variables, as defined in subclause 9.2.5, an array of string of bytes, TileData, and a variable providing the length in bytes of the array, TileDataBytes, for the tile with tile ID, tileID are generated as the output of this process and the following applies:

```
    TileDataBytes = 0
    for( tilePatchIdx = 0 ; tilePatchIdx < AtduTotalNumPatches[ tileID ] ; tilePatchIdx++ ) {
        TileDataBytes = TilePatchCommonByteString( TileData, TileDataBytes, tileID, atlasPatchIdx )

TileDataBytes = TilePatchApplicationByteString( TileData, TileDataBytes, tileID, atlasPatchIdx )
    }
```

where the functions TilePatchCommonByteString and TilePatchApplicationByteString are defined in subclauses F.3.16.2.4.1 and F.3.16.2.4.2, respectively.

F.3.16.2.4.1 Common tile patch string of bytes

Let the function TilePatchCommonByteString(stringByte, posByte, t, p) be defined as follows:

```
TilePatchCommonByteString( stringByte, posByte, tileID, p ) {
    stringByte[ posByte++ ] = TilePatchType[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatchType[ t ][ p ] >> 8 ) & 0xFF

    stringByte[ posByte++ ] = TilePatch2dPosX[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch2dPosX[ t ][ p ] >> 8 ) & 0xFF
    stringByte[ posByte++ ] = TilePatch2dPosY[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch2dPosY[ t ][ p ] >> 8 ) & 0xFF
    stringByte[ posByte++ ] = TilePatch2dSizeX[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch2dSizeX[ t ][ p ] >> 8 ) & 0xFF
    stringByte[ posByte++ ] = TilePatch2dSizeY[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch2dSizeY[ t ][ p ] >> 8 ) & 0xFF

    stringByte[ posByte++ ] = TilePatch3dOffsetU[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch3dOffsetU[ t ][ p ] >> 8) & 0xFF
    stringByte[ posByte++ ] = TilePatch3dOffsetV[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch3dOffsetV[ t ][ p ] >> 8 ) & 0xFF
    stringByte[ posByte++ ] = TilePatch3dOffsetV[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch3dOffsetV[ t ][ p ] >> 8 ) & 0xFF
    stringByte[ posByte++ ] = TilePatch3dOffsetD[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch3dOffsetD[ t ][ p ] >> 8 ) & 0xFF
    stringByte[ posByte++ ] = TilePatch3dRangeD[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatch3dRangeD[ t ][ p ] >> 8 ) & 0xFF

    stringByte[ posByte++ ] = TilePatchProjectionID[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = TilePatchOrientationIndex[ t ][ p ] & 0xFF

    stringByte[ posByte++ ] = TilePatchLoDScaleX[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatchLoDScaleX[ t ][ p ] >> 8 ) & 0xFF
    stringByte[ posByte++ ] = TilePatchLoDScaleY[ t ][ p ] & 0xFF
    stringByte[ posByte++ ] = (TilePatchLoDScaleY[ t ][ p ] >> 8) & 0xFF

    return posByte
}
```

F.3.16.2.4.2 Application based tile patch string of bytes

Let the function TilePatchApplicationByteString(stringByte, posByte, t, p) be defined as follows:

```
TilePatchApplicationByteString( stringByte, posByte, t, p) {
    stringByte[ posByte++ ] = TilePatchInAuxVideo[ p ] & 0xFF
    if( TilePatchType[ t ][ p ] == RAW ) {
        stringByte[ posByte++ ] = TilePatchRawPoints[ p ] & 0xFF
        stringByte[ posByte++ ] = ( TilePatchRawPoints[ p ] >> 8 ) & 0xFF
    } else if( TilePatchType[ t ][ p ] == PROJECTED ) {
        blockCnt = BlockCnt( AtlasPatch2dSizeX[ t ][ p ], AtlasPatch2dSizeY[ t ][ p ] )
        for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {
            if( plri_map_present_flag[ m ] == 1 ) {
                if(TilePlrdLevel[ t ][ p ][ m ] == 0 ) {
                    for( j = 0; j < blockCnt; j++ ) {
                        stringByte[ posByte++ ] =
                            TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ][ j ] & 0xFF
                        stringByte[ posByte++ ]=
                            (TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ][ j ] >> 8 ) & 0xFF
                    }
                    } else {
                        stringByte[ posByte++ ] = TilePatchPlrdModeMinus1[ t ][ p ][ m ] & 0xFF
                        stringByte[ posByte++ ] =
                            (TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ] >> 8 ) & 0xFF
                    }
                }
            }
        } else if( TilePatchType[ tileID ][ p ] == EOM ) {
            stringByte[ posByte++ ] = TilePatchEomPatchCount[ t ][ p ] & 0xFF
            stringByte[ posByte++ ] = (TilePatchEomPatchCount[ t ][ p ] >> 8 ) & 0xFF
            for( i = 0; i < TilePatchEomPatchCount[ t ][ p ]; i++ ) {
                stringByte[ posByte++ ] = TilePatchEomPoints[ t ][ p ][ i ] & 0xFF
                stringByte[ posByte++ ] = (TilePatchEomPoints[ t ][ p ][ i ] >> 8 ) & 0xFF
            }
        }
        return posByte
    }
```

F.3.16.2.5 Atlas block to patch string of bytes

In this subclause, using the block to patch maps from each atlas, as defined in subclause 9.2.7.2, an array, AtlasB2PData and a variable AtlasB2PDataBytes,providing the length of the array in bytes, are derived and set as output of this process as follows:

```
    AtlasB2PDataBytes = 0
    for( y = 0; y < AtlasBlockToPatchMapHeight; y++ ) {
        for( x = 0; x < AtlasBlockToPatchMapWidth; x++ ) {
            b2pVal = ( AtlasBlockToPatchMap[ y ][ x ] == −1 ) ?
                0xFFFF : AtlasBlockToPatchMap[ y ][ x ]
            AtlasB2pData[ AtlasB2PDataBytes++ ] = b2pVal & 0xFF
            AtlasB2pData[ AtlasB2PDataBytes++ ] = (b2pVal >> 8 ) & 0xFF
        }
    }
```

F.3.16.2.6 Tile block to patch string of bytes

In this subclause, using the block to patch maps for a tile with tile ID equal to t, as defined in subclause 9.2.6, an array, TileB2PData and the variable TileB2PDataBytes, providing the length in bytes of the array, are derived and set as output of this process as follows:

```
    TileB2PDataBytes = 0
    for( y = 0; y < TileBlockToPatchMapHeight[ t ]; y++ ) {
        for( x = 0; x < TileBlockToPatchMapWidth[ t ]; x++ ) {
            b2pVal = ( TileBlockToPatchMap[ t ][ y ][ x ] == −1 ) ?
                0xFFFF : TileBlockToPatchMap[ t ][ y ][ x ]
            TileB2pData[ TileB2PDataBytes++ ] = b2pVal & 0xFF
            TileB2pData[ TileB2PDataBytes++ ] = (b2pVal >> 8 ) & 0xFF
        }
    }
```

F.3.17 Time code SEI message semantics

The time code SEI message provides time code information similar to that defined by SMPTE ST 12-1 (2014) for atlas frame(s).

The contents of the clock timestamp syntax elements indicate a time of origin, capture, or alternative ideal display. This indicated time is computed as

$$\text{clockTimestamp} = ((\text{hH} * 60 + \text{mM}) * 60 + \text{sS}) * \text{time\_scale} + \text{nFrames} * \text{num\_units\_in\_tick} + \text{tOffset} \quad \text{(F.3)}$$

in units of clock ticks of a clock with clock frequency equal to time_scale Hz, relative to some unspecified point in time for which clockTimestamp would be equal to 0. Output order and DAB output timing are not affected by the value of clockTimestamp.

NOTE 1 clockTimestamp time indications can aid display on devices with refresh rates other than those well-matched to DAB output times. However, the time code SEI message does not affect the output order and DPB output timing defined in this document.

num_units_in_tick is the number of time units of a clock operating at the frequency time_scale Hz that corresponds to one increment (called a clock tick) of a clock tick counter. num_units_in_tick shall be greater than 0. A clock tick, in units of seconds, is equal to the quotient of num_units_in_tick divided by time_scale. For example, when the frame rate of an atlas signal is 25 Hz, time_scale may be equal to 27 000 000 and num_units_in_tick may be equal to 1 080 000 and consequently a clock tick may be equal to 0.04 seconds. When vui_timing_info_present_flag is equal to 1, it is a requirement of V3C bitstream conformance that the value of num_units_in_tick shall be equal to the value of vui_num_units_in_tick.

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 time_scale of 27 000 000. The value of time_scale shall be greater than 0. When vui_timing_info_present_flag is equal to 1, it is a requirement of V3C bitstream conformance that the value of time_scale shall be equal to the value of vui_time_scale.

counting_type specifies the method of dropping values of the n_frames syntax element as specified in Table F.6.

Table F.6 — Definition of counting_type values

Value	Interpretation
0	no dropping of n_frames count values and no use of time_offset_value
1	no dropping of n_frames count values
2	dropping of individual zero values of n_frames count
3	dropping of individual values of n_frames count equal to MaxFPS − 1
4	dropping of unspecified individual n_frames count values
5	dropping of unspecified numbers of unspecified n_frames count values
6..31	reserved

full_timestamp_flag equal to 1 specifies that the n_frames syntax element is followed by seconds_value, minutes_value and hours_value. full_timestamp_flag equal to 0 specifies that the n_frames syntax element is followed by seconds_flag.

discontinuity_flag equal to 0 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous set of clock timestamp syntax elements in output order can be interpreted as the time difference between the times of origin or capture of the associated frames. discontinuity_flag equal to 1 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous set of clock timestamp syntax elements in output order should not be interpreted as the time difference between the times of origin or capture of the associated frames. When discontinuity_flag is equal to 0, the value of clockTimestamp shall be greater than or equal to the value of clockTimestamp for the previous set of clock timestamp syntax elements in output order (when present).

cnt_dropped_flag specifies the skipping of one or more values of n_frames using the counting method specified by counting_type.

n_frames specifies the value of nFrames used to compute clockTimestamp. n_frames shall be less than MaxFPS, where MaxFPS is specified by

$$\text{MaxFPS} = \text{Ceil}(\text{time\_scale} \div \text{num\_units\_in\_tick}) \quad \text{(F.4)}$$

When counting_type is equal to 2 and cnt_dropped_flag is equal to 1, n_frames shall be equal to 1 and the value of n_frames for the previous set of clock timestamp syntax elements in output order (when present) shall not be equal to 0 unless discontinuity_flag is equal to 1.

NOTE 2 When counting_type[i] is equal to 2, the need for increasingly large magnitudes of tOffset in Formula (F.3) when using fixed non-integer frame rates (e.g., 12.5 frames per second with time_scale equal to 50 and num_units_in_tick equal to 2) can be avoided by occasionally skipping over the value n_frames equal to 0 when counting (e.g., counting n_frames from 0 to 12, then incrementing seconds_value and counting n_frames from 1 to 12, then incrementing seconds_value and counting n_frames from 0 to 12, etc.).

When counting_type is equal to 3, and cnt_dropped_flag is equal to 1, n_frames shall be equal to 0 and the value of n_frames for the previous set of clock timestamp syntax elements in output order (when present) shall not be equal to MaxFPS − 1 unless discontinuity_flag is equal to 1.

NOTE 3 When counting_type is equal to 3, the need for increasingly large magnitudes of tOffset in Formula (F.3) when using fixed non-integer frame rates (e.g., 12.5 frames per second with time_scale equal to 50 and num_units_in_tick equal to 2) can be avoided by occasionally skipping over the value n_frames equal to MaxFPS − 1 when counting (e.g., counting n_frames from 0 to 12, then incrementing seconds_value and counting n_frames from 0 to 11, then incrementing seconds_value and counting n_frames from 0 to 12, etc.).

When counting_type is equal to 4 or 5, and cnt_dropped_flag is equal to 1, n_frames shall not be equal to 1 plus the value of n_frames for the previous set of clock timestamp syntax elements in output order (when present) modulo MaxFPS unless discontinuity_flag is equal to 1.

NOTE 4 When counting_type is equal to 4 or 5, the need for increasingly large magnitudes of tOffset in Formula (F.3) when using fixed non-integer frame rates can be avoided by occasionally skipping over some values of n_frames when counting. The specific values of n_frames that are skipped are not specified when counting_type is equal to 4 or 5.

seconds_flag equal to 1 specifies that seconds_value and minutes_flag are present when full_timestamp_flag is equal to 0. seconds_flag equal to 0 specifies that seconds_value and minutes_flag are not present.

seconds_value specifies the value of sS used to compute clockTimestamp. The value of seconds_value shall be in the range of 0 to 59, inclusive. When seconds_value is not present, its value is inferred to be equal to the value of seconds_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous seconds_value shall have been present.

minutes_flag equal to 1 specifies that minutes_value and hours_flag are present when full_timestamp_flag is equal to 0 and seconds_flag is equal to 1. minutes_flag equal to 0 specifies that minutes_value and hours_flag are not present.

minutes_value specifies the value of mM used to compute clockTimestamp. The value of minutes_value shall be in the range of 0 to 59, inclusive. When minutes_value is not present, its value is inferred to be equal to the value of minutes_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous minutes_value shall have been present.

hours_flag equal to 1 specifies that hours_value is present when full_timestamp_flag is equal to 0 and seconds_flag is equal to 1 and minutes_flag is equal to 1.

hours_value specifies the value of hH used to compute clockTimestamp. The value of hours_value shall be in the range of 0 to 23, inclusive. When hours_value is not present, its value is inferred to be equal to the value of hours_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous hours_value shall have been present.

time_offset_length greater than 0 specifies the length in bits of the time_offset_value syntax element. time_offset_length equal to 0 specifies that the time_offset_value syntax element is not present. When counting_type is equal to 0, time_offset_length shall be equal to 0.

NOTE 5 time_offset_length is expected to be the same for all atlas frames in the CAS.

time_offset_value specifies the value of tOffset used to compute clockTimestamp. The number of bits used to represent time_offset_value is equal to time_offset_length. When time_offset_value is not present, its value is inferred to be equal to 0.

Annex G
(normative)

Volumetric usability information

G.1 General

This annex specifies syntax and semantics of the VUI parameters of the ASPSs associated with an atlas, identified with an atlas ID, VUIAtlasID, which is set equal to vuh_atlas_id or determined through external means if the V3C unit header is unavailable.

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 E for the specification of output order conformance). Some VUI parameters are required to check bitstream conformance and for output timing decoder conformance.

In this annex, 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 document. 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.

G.2 VUI syntax

G.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(0)	
}	
vui_tile_restrictions_present_flag	u(1)
if(vui_tile_restrictions_present_flag) {	
vui_fixed_atlas_tile_structure_flag	u(1)
vui_fixed_video_tile_structure_flag	u(1)
vui_constrained_tiles_across_v3c_components_idc	ue(v)
vui_max_num_tiles_per_atlas_minus1	ue(v)
}	

vui_max_coded_video_resolution_present_flag	u(1)
if(vui_max_coded_video_resolution_present_flag)	
max_coded_video_resolution( )	
vui_coordinate_system_parameters_present_flag	u(1)
if(vui_coordinate_system_parameters_present_flag)	
coordinate_system_parameters( )	
vui_unit_in_metres_flag	u(1)
vui_display_box_info_present_flag	u(1)
if(**vui_display_box_info_present_flag**) {	
for(d = 0; d < 3; d++) {	
vui_display_box_origin[d]	u(v)
vui_display_box_size[d]	u(v)
}	
}	
vui_anchor_point_present_flag	u(1)
if(vui_anchor_point_present_flag)	
for(d = 0; d < 3; d++)	
vui_anchor_point[d]	u(v)
}	

G.2.2 HRD parameters syntax

hrd_parameters(maxNumSubLayersMinus1) {	**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)
}	
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)
else	
hrd_fixed_atlas_rate_within_cas_flag[i] = 1	
if(hrd_fixed_atlas_rate_within_cas_flag[i]) {	
hrd_elemental_duration_in_tc_minus1[i]	ue(v)
hrd_low_delay_flag[i] = 0	
} else	
hrd_low_delay_flag[i]	u(1)
if(!hrd_low_delay_flag[i])	
hrd_cab_cnt_minus1[i]	ue(v)
else	
hrd_cab_cnt_minus1[i] = 0	
if(hrd_nal_parameters_present_flag)	

hrd_sub_layer_parameters(0, i)	
if(hrd_acl_parameters_present_flag)	
hrd_sub_layer_parameters(1, i)	
}	
}	

G.2.3 Sub-layer HRD parameters syntax

hrd_sub_layer_parameters(hrdMode, subLayerID) {	**Descriptor**
for(j = 0; j <= CabCnt; j++) {	
hrd_bit_rate_value_minus1[hrdMode][subLayerID][j]	ue(v)
hrd_cab_size_value_minus1[hrdMode][subLayerID][j]	ue(v)
hrd_cbr_flag[hrdMode][subLayerID][j]	u(1)
}	
}	

G.2.4 Maximum coded video resolution syntax

max_coded_video_resolution( ) {	**Descriptor**
mcv_occupancy_resolution_present_flag	u(1)
mcv_geometry_resolution_present_flag	u(1)
mcv_attribute_resolution_present_flag	u(1)
if(mcv_occupancy_resolution_present_flag) {	
mcv_occupancy_width	ue(v)
mcv_occupancy_height	ue(v)
}	
if(mcv_geometry_resolution_present_flag) {	
mcv_geometry_width	ue(v)
mcv_geometry_height	ue(v)
}	
if(mcv_attribute_resolution_present_flag) {	
mcv_attribute_width	ue(v)
mcv_attribute_height	ue(v)
}	
}	

G.2.5 Coordinate system parameters syntax

coordinate_system_parameters( ) {	**Descriptor**
csp_forward_axis	u(2)
csp_delta_left_axis_minus1	u(1)
csp_forward_sign	u(1)
csp_left_sign	u(1)
csp_up_sign	u(1)
}	

G.3 VUI semantics

G.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 frame in the CAS that is not the first atlas frame in the CAS, in decoding order, is proportional to the output time of the atlas frame relative to the output time of the first atlas frame in the CAS. vui_poc_proportional_to_timing_flag equal to 0 indicates that the atlas frame order count value for each atlas frame in the CAS that is not the first atlas frame in the CAS, in decoding order, may or may not be proportional to the output time of the atlas frame relative to the output time of the first atlas frame 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.

vui_tile_restrictions_present_flag equal to 1 specifies that the syntax elements vui_fixed_atlas_tile_structure_flag, vui_fixed_video_tile_structure_flag, vui_constrained_tiles_across_v3c_components_idc, and vui_max_num_tiles_per_atlas_minus1 are present in the vui_parameters( ) syntax structure. vui_tile_restrictions_present_flag equal to 0 specifies that the syntax elements vui_fixed_atlas_tile_structure_flag, vui_fixed_video_tile_structure_flag, vui_constrained_tiles_across_v3c_components_idc, and vui_max_num_tiles_per_atlas_minus1 are not present in the vui_parameters( ) syntax structure.

vui_fixed_atlas_tile_structure_flag equal to 1 indicates that all the atlas frames of the current atlas shall have the same tiling structure. vui_fixed_atlas_tile_structure_flag equal to 0 indicates that atlas frames of the current atlas may or may not have the same tiling structure. When the vui_fixed_atlas_tile_structure_flag syntax element is not present, it is inferred to be equal to 0.

vui_fixed_video_tile_structure_flag equal to 1 indicates that for each video sub-bitstream associated with the current atlas, all of its frames shall have the same tiling structure. vui_fixed_video_tile_structure_flag equal to 0 indicates that for each video sub-bitstream associated with the current atlas, frames may or may not have the same tiling structure. When the vui_fixed_video_tile_structure_flag syntax element is not present, it is inferred to be equal to 0.

vui_constrained_tiles_across_v3c_components_idc indicates whether any constraints apply to the sizes of tiles in the atlas sub-bitstream and the video tiles in the video sub-bitstreams, as specified in Table G.7.

Table G.7 — Definition of vui_constrained_tiles_across_v3c_components_idc

Value	Interpretation
0	Unconstrained
1	Proportionally constrained video tiles
2	Atlas based constrained video tiles with exact match
3	Atlas based constrained video tiles
4	Edge based constrained video tiles

Values outside the 0 to 4 range, inclusive, are reserved for future use by ISO/IEC. It is a requirement of bitstream conformance that bitstreams conforming to this edition of this document shall not contain such values of vui_constrained_tiles_across_v3c_components_idc.

Video tiles can be independent coding units defined by the video or image coding specification and may vary in name according to such specification. Names could include slices, partitions, tiles, or sub-pictures, among others. Some specifications may not support multiple tiles, i.e. are limited to a single video tile. The specification of such tiles is outside the scope of this document.

vui_constrained_tiles_across_v3c_components_idc equal to 0 means that the tile sizes of the video are constrained only by the video coding specification used. The tile sizes of the atlas sub-bitstream are constrained by this document.

vui_constrained_tiles_across_v3c_components_idc equal to 1 means that the tile sizes of the video and atlas sub-bitstreams are constrained as follows:

For each tile in the atlas sub-bitstream, there is a corresponding video tile for each video sub-bitstream present (attributes, geometry and occupancy) such that the video tile, when scaled to the nominal format, as defined in Annex B.2, represents exactly the same area on the atlas as the corresponding atlas tile.

NOTE 1 In this edition of this document, the nominal format has the same resolution as the atlas.

Let the width and height of the video be videoWidth and videoHeight, respectively. Let the width and height of the video tile be videoTileWidth and videoTileHeight, respectively. Let the width and height of the atlas tile be atlasTileWidth and atlasTileHeight, respectively. Then,

videoTileWidth = atlasTileWidth * videoWidth / asps_frame_width and

videoTileHeight = atlasTileHeight * videoHeight / asps_frame_height.

It shall be possible to decode each video tile that corresponds to a tile in the atlas sub-bitstream without reference to any information from other video tiles in that sub-bitstream.

vui_constrained_tiles_across_v3c_components_idc equal to 2 means that the tile sizes of the video and atlas sub-bitstreams are constrained as follows:

For each tile in the atlas sub-bitstream, there is a set of video tiles for each video sub-bitstream present (occupancy, geometry and attributes) such that the set of video tiles, when scaled to the nominal format, as defined in Annex B.2, together, represent exactly the same area on the atlas as the atlas tile.

NOTE 2 In this edition of this document, the nominal format has the same resolution as the atlas.

It shall be possible to decode each set of video tiles that corresponds to a tile in the atlas sub-bitstream without reference to any information from video tiles from that sub-bitstream that are outside that set.

vui_constrained_tiles_across_v3c_components_idc equal to 3 means that the tile sizes of the video and atlas sub-bitstreams are constrained as follows:

For each tile in the atlas sub-bitstream, there is a corresponding video tile for each video sub-bitstream present (occupancy, geometry and attributes) such that the video tile, when scaled to the nominal format, as defined in Annex B.2, represents an area on the atlas that is greater than or equal to the area represented by the atlas tile.

NOTE 3 In this edition of this document, the nominal format has the same resolution as the atlas.

The number of luma samples in the video tile shall be less than or equal to the number of samples in the atlas tile.

It shall be possible to decode each video tile that corresponds to a tile in the atlas sub-bitstream without reference to any information from other video tiles in that sub-bitstream.

vui_constrained_tiles_across_video_components_idc equal to 4 means that for all video sub-bitstreams, the tiling pattern, after scaling to the nominal format, as defined in Annex B.2, is considered, is such that one of the following must hold for any pair of tiles, X from one sub-bitstream and Y from another sub-bitstream:

— X and Y are non-overlapping

— X is completely contained in Y

— Y is completely contained in X.

It shall be possible to decode each tile in the video sub-bitstreams without reference to any information from other tiles in that video bitstream.

NOTE 4 In this edition of this document, the nominal format has the same resolution as the atlas.

The sizes of the tiles in the auxiliary atlas sub-bitstream and the video tiles in the auxiliary video sub-bitstreams are also constrained depending on the value of vui_constrained_tiles_across_video_components_idc.

vui_max_num_tiles_per_atlas_minus1 plus 1 indicates the maximum number of tiles present in the CAS.

vui_max_coded_video_resolution_present_flag equal to 1 specifies that the syntax structure max_coded_video_resolution( ) is present in the vui_parameters( ) syntax structure. vui_max_coded_video_resolution_present_flag equal to 0 specifies that the syntax structure max_coded_video_resolution( ) is not present in the vui_parameters( ) syntax structure.

vui_coordinate_system_parameters_present_flag equal to 1 specifies that the syntax structure coordinate_system_parameters( ) is present in the vui_parameters( ) syntax structure. vui_coordinate_system_parameters_present_flag equal to 0 specifies that the syntax structure coordinate_system_parameters( ) is not present in the vui_parameters( ) syntax structure.

vui_unit_in_metres_flag equal to 1 specifies that the real-world coordinates information is expressed in metres. vui_unit_in_metres_flag equal to 0 specifies that the world coordinates are unitless. If vui_unit_in_metres_flag is not present, it shall be inferred to be equal to 0.

vui_display_box_info_present_flag equal to 1 specifies that the syntax elements vui_display_box_origin[d] and vui_display_box_size[d] are present in the vui_parameters( ) syntax structure. vui_display_box_info_present_flag equal to 0 specifies that the syntax elements vui_display_box_origin[d] and vui_display_box_size[d] elements are not present in the vui_parameters( ) syntax structure. The values of d equal to 0, 1 and 2 correspond to the X, Y and Z axis, respectively.

vui_display_box_origin[d] specifies an offset with respect to the coordinate system origin point along the axis d. When elements of the vui_display_box_origin[d] are not present, its values shall be inferred to be equal to 0. The number of bits used to represent vui_display_box_origin[d] is asps_geometry_3d_bit_depth_minus1 +1. The values of d equal to 0, 1 and 2 correspond to the X, Y and Z axis, respectively.

vui_display_box_size[d] specifies the size of the display box in terms of samples in the direction of the axis d. When elements of the vui_display_box_size[d] are not present, its values are unknown. The number of bits used to represent vui_display_box_origin[d] is asps_geometry_3d_bit_depth_minus1 + 1.

The following variables are derived from the display box parameters:

$$\text{minOffset}[\,d\,] = \text{vui\_display\_box\_origin}[\,d\,] \qquad \text{(G.1)}$$

$$\text{maxOffset}[\,d\,] = \text{vui\_display\_box\_origin}[\,d\,] + \text{vui\_display\_box\_size}[\,d\,] \qquad \text{(G.2)}$$

where the values of d equal to 0, 1 and 2 correspond to the X, Y and Z axis, respectively.

vui_anchor_point_present_flag equal to 1 indicates that vui_anchor_point[d] syntax elements are present in the vui_parameters( ) syntax structure. vui_anchor_point_present_flag equal to 0 indicates that vui_anchor_point[d] elements are not present.

vui_anchor_point[d] indicates a position of an anchor point along the d axis. The value of vui_anchor_point[d] shall be in range of 0 to $2^{\text{asps\_geometry\_3d\_bit\_depth\_minus1}+1} - 1$. If vui_anchor_point[d] is not present it shall be inferred to be equal to 0. The number of bits used to represent vui_anchor_point[d] is asps_geometry_3d_bit_depth_minus1 + 1. The values of d equal to 0, 1 and 2 correspond to the X, Y and Z axis, respectively.

G.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 edition of this document 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 document.

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 document.

hrd_bit_rate_scale, together with hrd_bit_rate_value_minus1[h][i][j], specifies the maximum input bit rate of the j-th CAB when Htid is equal to i. When hrd_bit_rate_scale is not present, its value is inferred to be equal to 1.

hrd_cab_size_scale, together with hrd_cab_size_value_minus1[h][i][j], specifies the CAB size of the j-th CAB when Htid is equal to i and when the CAB operates at the coded atlas access unit level. When hrd_cab_size_scale is not present, its value is inferred to be equal to 1.

hrd_fixed_atlas_rate_general_flag[i] equal to 1 indicates that, when Htid is equal to i, the temporal distance between the HRD output times of consecutive atlas frames in output order is constrained using additional syntax elements. 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 Htid is equal to i, the temporal distance between the HRD output times of consecutive atlases in output order is constrained using additional syntax elements. hrd_fixed_atlas_rate_within_cas_flag[i] equal to 0 indicates that this constraint may not apply. When hrd_fixed_atlas_rate_within_cas_flag[i] is not present (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 Htid 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.

When Htid is equal to i and hrd_fixed_atlas_rate_general_flag[i] is equal to 1 for a CAS containing atlas frame n, the value computed for DabOutputInterval[n] shall be equal to ClockTick * (hrd_elemental_duration_in_tc_minus1[i] + 1), wherein ClockTick is as specified in Formula (E.1) (using the value of ClockTick for the CAS containing atlas frame n) when one of the following conditions is true for the following atlas in output order nextAtlasFrameInOutputOrder that is specified for use in Formula (E.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 Htid 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 DabOutputInterval[n] shall be equal to ClockTick * (hrd_elemental_duration_in_tc_minus1[i] + 1), wherein ClockTick is as specified in Formula (E.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 Formula (E.12) is in the same CAS as atlas frame n.

hrd_low_delay_flag[i] specifies the HRD operational mode, when Htid is equal to i, as specified in Annex E. 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, it is possible for "big atlases" that violate the nominal CAB removal times due to the number of bits used by an access unit. 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 Htid 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.

G.3.3 HRD sub-layer parameters semantics

The variable CabCnt is set equal to hrd_cab_cnt_minus1[subLayerID].

hrd_bit_rate_value_minus1[h][i][j] (together with hrd_bit_rate_scale) specifies the maximum input bit rate for the j-th CAB, for the i-th sub-layer, when the CAB operates at the access unit level, with h indicating the type of bitstream conformance. hrd_bit_rate_value_minus1[h][i][j] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any j greater than 0 and any particular value of i, hrd_bit_rate_value_minus1[h][i][j] shall be greater than hrd_bit_rate_value_minus1[h][i][j − 1].

The bit rate in bits per second is given by:

$$\text{BitRate[h][i][j]} = (\text{hrd\_bit\_rate\_value\_minus1[h][i][j]} + 1) * 2^{(6 + \text{hrd\_bit\_rate\_scale})} \tag{G.3}$$

When the hrd_bit_rate_value_minus1[h][i][j] syntax element is not present, the value of BitRate[h][i][j] is inferred to be equal to MaxAtlasBR, where MaxAtlasBR is specified in Annex A.

hrd_cab_size_value_minus1[h][i][j] is used together with cpb_size_scale to specify the j-th CAB size, for the i-th sub-layer, when the CAB operates at the access unit level, with h indicating the type of bitstream conformance. hrd_cab_size_value_minus1[h][i][j] shall be in the range of 0 to $2^{32} - 2$, inclusive. For any j greater than 0 and any particular value of i, hrd_cab_size_value_minus1[h][i][j] shall be less than or equal to hrd_cab_size_value_minus1[h][i][j − 1].

The CAB size in bits is given by:

$$\text{CabSize[h][i][j]} = (\text{hrd\_cab\_size\_value\_minus1[h][i][j]} + 1) * 2^{(4 + \text{hrd\_cab\_size\_scale})} \tag{G.4}$$

When the hrd_cab_size_value_minus1[h][i][j] syntax element is not present, the value of CabSize[h][i][j] is inferred to be equal to MaxCABSize, where MaxCABSize is specified in Annex A.

hrd_cbr_flag[h][i][j] equal to 0 specifies that to decode this CAS bitstream by the HRD using the j-th CAB specification, for the i-th sub-layer and with h indicating the type of bitstream conformance, the hypothetical stream scheduler (HSS) operates in an intermittent bit rate mode. hrd_cbr_flag[h][i][j] equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode. When not present, the value of hrd_cbr_flag[h][i][j] is inferred to be equal to 0.

G.3.4 Maximum coded video resolution semantics

mcv_occupancy_resolution_present_flag equal to 1 specifies that the syntax elements mcv_occupancy_width and mcv_occupancy_height are present in the max_coded_video_resolution( ) syntax structure. mcv_occupancy_resolution_present_flag equal to 0 specifies that the syntax elements mcv_occupancy_width and mcv_occupancy_height are not present. When mcv_occupancy_resolution_present_flag is not present, its value is inferred to be equal to 0.

mcv_geometry_resolution_present_flag equal to 1 specifies that the syntax elements mcv_geometry_width and mcv_geometry_height are present in the max_coded_video_resolution( ) syntax structure. mcv_geometry_resolution_present_flag equal to 0 specifies that the syntax elements mcv_geometry_width and mcv_geometry_height are not present. When mcv_geometry_resolution_present_flag is not present, its value is inferred to be equal to 0.

mcv_attribute_resolution_present_flag equal to 1 specifies that the syntax elements mcv_attribute_width and mcv_attribute_height are present in max_coded_video_resolution( ) syntax structure. mcv_attribute_resolution_present_flag equal to 0 specifies that the syntax elements mcv_attribute_width and mcv_attribute_height are not present. When mcv_attribute_resolution_present_flag is not present, its value is inferred to be equal to 0.

mcv_occupancy_width specifies the maximum width value allowed for any frame in the occupancy sub-bitstream of the current atlas, VUIAtlasID. mcv_occupancy_width shall be larger than or equal to DecOccWidth[i], as defined in subclause 9.3, for i in the range of 0 to NumDecOccFrames − 1, inclusive. If mcv_occupancy_width is not present, then no constraint to the width of the occupancy sub-bitstream frames is specified.

mcv_occupancy_height specifies the maximum height value allowed for any frame in the occupancy sub-bitstream of the current atlas, VUIAtlasID. mcv_occupancy_height shall be larger than or equal to DecOccHeight[i], as defined in subclause 9.3, for i in the range of 0 to NumDecOccFrames − 1, inclusive. If mcv_occupancy_height is not present, then no constraint to the height of the occupancy sub-bitstream frames is specified.

mcv_geometry_width specifies the maximum width value allowed for any frame in the geometry auxiliary and non-auxiliary sub-bitstreams of the current atlas, VUIAtlasID. mcv_geometry_width shall be larger than or equal to DecGeoWidth[geoBitstreamIdx][i] and DecGeoAuxWidth[j], as defined in subclause 9.4, with geoBitstreamIdx in the range of 0 to vps_multiple_map_streams_present_flag[VUIAtlasID] ? vps_map_count_minus1[j] : 0, inclusive, i in the range of 0 to NumDecGeoFrames − 1, inclusive, and j in the range of 0 to NumDecGeoAuxFrames − 1, inclusive. If mcv_geometry_width is not present, then no constraint to the width of the geometry auxiliary and non-auxiliary sub-bitstream frames is specified.

mcv_geometry_height specifies the maximum height value allowed for any frame in the geometry auxiliary and non-auxiliary sub-bitstreams of the current atlas, VUIAtlasID. mcv_geometry_height shall be larger than or equal to DecGeoHeight[geoBitstreamIdx][i] and DecGeoAuxHeight[j], as defined in subclause 9.4, with geoBitstreamIdx in the range of 0 to vps_multiple_map_streams_present_flag[VUIAtlasID] ? vps_map_count_minus1[j] : 0, inclusive, i in the range of 0 to NumDecGeoFrames − 1, inclusive, and j in the range of 0 to NumDecGeoAuxFrames − 1, inclusive. If mcv_geometry_height is not present, then no constraint to the height of the geometry auxiliary and non-auxiliary sub-bitstream frames is specified.

mcv_attribute_width specifies the maximum width value allowed for any frame in the attribute auxiliary and non-auxiliary sub-bitstreams of the current atlas, VUIAtlasID. mcv_attribute_width shall be larger than or equal to DecAttrWidth[attrIdx][partIdx][attrBitstreamIdx][i] and DecAttrAuxWidth[attrIdx][partIdx][j], as defined in subclause 9.5, with attrIdx in the range of 0 to ai_attribute_count[VUIAtlasID] − 1, inclusive, partIdx in the range of 0 to ai_attribute_dimension_partitions_minus1[VUIAtlasID][attrIdx], inclusive, attrBitstreamIdx in the range of 0 to vps_multiple_map_streams_present_flag[VUIAtlasID] ? vps_map_count_minus1[j] : 0, inclusive, , i in the range of 0 to NumDecAttrFrames − 1, inclusive, and j in the range of 0 to NumDecAttrAuxFrames − 1, inclusive. If mcv_attribute_width is not present, then no constraint to the width of the attribute auxiliary and non-auxiliary sub-bitstream frames is specified.

mcv_attribute_height specifies the maximum height value allowed for any frame in the attribute auxiliary and non-auxiliary sub-bitstreams of the current atlas, VUIAtlasID. mcv_attribute_height shall be larger than or equal to DecAttrHeight[attrIdx][partIdx][attrBitstreamIdx][i] and DecAttrAuxHeight[attrIdx][partIdx][j], as defined in subclause 9.5, with attrIdx in the range of 0 to ai_attribute_count[VUIAtlasID] − 1, inclusive, partIdx in the range of 0 to ai_attribute_dimension_partitions_minus1[VUIAtlasID][attrIdx], inclusive, attrBitstreamIdx in the range of 0 to vps_multiple_map_streams_present_flag[VUIAtlasID] ? vps_map_count_minus1[j] : 0, inclusive, , i in the range of 0 to NumDecAttrFrames − 1, inclusive, and j in the range of 0 to NumDecAttrAuxFrames − 1, inclusive. If mcv_attribute_height is not present, then no constraint to the height of the attribute auxiliary and non-auxiliary sub-bitstream frames is specified.

G.3.5 Coordinate system parameters semantics

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

csp_forward_axis specifies the axis index of the forward direction. The variable CspForwardAxisIndex is derived as follows:

CspForwardAxisIndex = csp_forward_axis (G.5)

csp_delta_left_axis_minus1 plus 1 specifies the difference between the left axis index, CspLeftAxisIndex, and the forward axis index, CspForwardAxisIndex. The variable CspLeftAxisIndex is derived as follows:

CspLeftAxisIndex = (CspForwardAxisIndex + csp_delta_left_axis_minus1 + 1) % 3 (G.6)

The variable CspUpAxisIndex is derived from the CspForwardAxisIndex and CspLeftAxisIndex variables:

```
if( CspForwardAxisIndex != (CspLeftAxisIndex + 1) % 3)
    CspUpAxisIndex = (CspLeftAxisIndex + 1) % 3                                   (G.7)
else
    CspUpAxisIndex = (CspLeftAxisIndex + 2) % 3                                   (G.8)
```

csp_forward_sign specifies the forward direction in relation to the forward axis. csp_forward_sign equal to 1 specifies that the forward direction is equal to the forward axis direction. csp_forward_sign equal to 0 specifies that the forward direction is opposite to the forward axis direction.

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

```
for( i = 0; i < 3; i++ )
    CspForwardVector[ i ] = (i == CspForwardAxisIndex ) ? (2 * csp_forward_sign – 1 ) : 0       (G.9)
```

csp_left_sign specifies the left direction in relation to the left axis. csp_left_sign equal to 1 specifies that the left direction is equal to the left axis direction. csp_left_sign equal to 0 specifies that the left direction is opposite to the left axis direction.

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

```
for( i = 0; i < 3; i++ )
    CspLeftVector[ i ] = ( i == CspLeftAxisIndex ) ? (2 * csp_left_sign – 1 ) : 0               (G.10)
```

csp_up_sign specifies the up direction in relation to the up axis. csp_up_sign equal to 1 specifies that the up direction is equal to the up axis direction. csp_up_sign equal to 0 specifies that the up direction is opposite to the up axis direction.

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

```
for( i = 0; i < 3; i++ )
    CspUpVector[ i ] = ( i == CspUpAxisIndex ) ? ( 2 * csp_up_sign – 1 ) : 0                    (G.11)
```

NOTE The three vectors CspForwardVector, CspLeftVector and CspUpVector can be combined into a coordinate transformation matrix for rendering purposes. The layout of the coordinate transformation matrix is out of scope of this document.

Annex H
(normative)

Video-based point cloud coding

H.1 General

This annex specifies the syntax, semantics, decoding, post-decoding, pre-reconstruction, reconstruction, post-reconstructions, and adaptation processes for video-based point cloud compression that use the syntax, semantics, decoding, post-decoding, pre-reconstruction, reconstruction, post-reconstructions, and adaptation processes specified in Clauses 2 through 14, inclusive and Annex A through Annex F, inclusive. This annex also specifies profiles, tiers and levels for video-based point cloud compression.

H.2 Overall V-PCC characteristics, decoding operations and post-decoding processes

H.2.1 V3C characteristics

The specifications in subclause 6.1 apply.

H.2.2 V3C bitstream characteristics, decoding operations and post-decoding processes

This subclause provides high-level description of the characteristics and the operations needed for the decoding of V3C bitstreams and optional post-decoding and reconstruction related processes needed by applications, which may include nominal format conversion, pre-reconstruction, reconstruction, post-reconstruction and adaptation.

As mentioned in subclause 6.1, a V3C bitstream is composed of a collection of V3C components, such as atlas, occupancy, geometry and attribute components. Furthermore, to provide additional functionalities and similar flexibility as available in many video specifications, the atlas component may be divided into tiles and is encapsulated into NAL units. Clause 7 provides further details on the encapsulation of V3C component bitstreams into V3C units and NAL units.

V3C bitstream syntax elements and their semantics are specified in Clause H.4. Particular focus is placed on the atlas bitstream, its characteristics, and any constraints that may apply on such syntax.

Clause H.5 invokes the decoding process of a V3C bitstream or a collection of V3C sub-bitstreams, with outputs composed of decoded atlas information, a set of decoded video streams, corresponding to the occupancy, geometry and attribute components, if available, and any associated information from the VPS if available.

The decoded video frames may require the application of additional transformations, as described in Annex B, before any reconstruction operations. For example, the different components may need to be time-aligned and converted to a nominal video format. The outputs of Annex B are the following videos in the nominal format: OccFramesNF and GeoFramesNF, and, if present, GeoAuxFramesNF, AttrFramesNF, and AttrAuxFramesNF.

With the V3C components in the nominal format, the point cloud content is obtained by invoking several additional steps, including pre-reconstruction, reconstruction, post-reconstruction and adaptation.

The pre-reconstruction process, which is specified in Clause H.6, may specify additional processes to be applied on the reconstructed video frames, such as certain forms of post-filtering, which may result in improved visual quality or other benefits. Currently a process that allows for the modification of the occupancy frames, OccFramesNF, is specified. In particular, the technique named as the occupancy synthesis process is specified in subclause H.6.2.

In the reconstruction stage, described in Clause H.7, the video components in the nominal format, OccFramesNF and GeoFramesNF, and, if present, GeoAuxFramesNF, AttrFramesNF, and AttrAuxFramesNF, along with the decoded atlas data, are processed to reconstruct the point cloud content.

The reconstructed point cloud content can be further processed by applying post-reconstruction methods, as described in Clause H.8. For example, point cloud post-reconstruction in the form of applying a geometry smoothing, attribute smoothing, or attribute transfer process, may be performed on the reconstructed point cloud, as specified in subclauses H.8.2, H.8.3 and H.8.8, respectively.

Depending on the application, the post-reconstructed point cloud content can be further optionally processed by additional transformations, as described in Clause H.9.

H.3 Bitstream format, partitioning and scanning process

The specifications in Clause 7 and its subclauses apply.

H.4 Syntax and semantics

H.4.1 Method of specifying syntax in tabular form

The specifications in subclause 8.1 apply.

H.4.2 Specification of syntax functions and descriptors

The specifications in subclause 8.2 apply

H.4.3 Syntax in tabular form

H.4.3.1 General

The specifications in subclause 8.3.1 and its subclauses apply.

H.4.3.2 V3C unit syntax

The specifications in subclause 8.3.2 and its subclauses apply.

H.4.3.3 Byte alignment syntax

The specifications in subclause 8.3.3 and its subclauses apply.

H.4.3.4 V3C parameter set syntax

The specifications in subclause 8.3.4 and its subclauses apply.

H.4.3.5 NAL unit syntax

The specifications in subclause 8.3.5 and its subclauses apply.

H.4.3.6 Raw byte sequence payloads, trailing bits and byte alignment syntax

H.4.3.6.1 Atlas sequence parameter set RBSP syntax

H.4.3.6.1.1 General atlas sequence parameter set RBSP syntax

The specifications in subclause 8.3.6.1.1 apply.

H.4.3.6.1.2 Point local reconstruction information syntax

The specifications in subclause 8.3.6.1.2 apply.

H.4.3.6.1.3 ASPS V-PCC extension syntax

asps_vpcc_extension( ) {	**Descriptor**
asps_vpcc_remove_duplicate_point_enabled_flag	u(1)
if(asps_pixel_deinterleaving_enabled_flag || asps_plr_enabled_flag)	
asps_vpcc_surface_thickness_minus1	ue(v)
}	

H.4.3.6.2 Atlas frame parameter set RBSP syntax

The specifications in subclause 8.3.6.2 and its subclauses apply.

H.4.3.6.3 Atlas adaptation parameter set RBSP syntax

H.4.3.6.3.1 General atlas adaptation parameter set RBSP syntax

The specifications in subclause 8.3.6.3 apply.

H.4.3.6.3.2 AAPS V-PCC extension syntax

aaps_vpcc_extension ( ) {	**Descriptor**
aaps_vpcc_camera_parameters_present_flag	u(1)
if(aaps_vpcc_camera_parameters_present_flag)	
atlas_camera_parameters( )	
}	

H.4.3.6.3.3 Atlas camera parameters syntax

atlas_camera_parameters( ) {	**Descriptor**
acp_camera_model	u(8)
if(acp_camera_model == 1) {	
acp_scale_enabled_flag	u(1)
acp_offset_enabled_flag	u(1)
acp_rotation_enabled_flag	u(1)
if(acp_scale_enabled_flag)	
for(d = 0; d < 3; d++)	
acp_scale_on_axis[d]	u(32)
if(acp_offset_enabled_flag)	

for(d = 0; d < 3; d++)	
acp_offset_on_axis[d]	i(32)
if(acp_rotation_enabled_flag) {	
acp_rotation_qx	i(16)
acp_rotation_qy	i(16)
acp_rotation_qz	i(16)
}	
}	
}	

H.4.3.6.4 Supplemental enhancement information RBSP syntax

The specifications in subclause 8.3.6.4 and its subclauses apply.

H.4.3.6.5 Access unit delimiter RBSP syntax

The specifications in subclause 8.3.6.5 and its subclauses apply.

H.4.3.6.6 End of sequence RBSP syntax

The specifications in subclause 8.3.6.6 and its subclauses apply.

H.4.3.6.7 End of bitstream RBSP syntax

The specifications in subclause 8.3.6.7 and its subclauses apply.

H.4.3.6.8 Filler data RBSP syntax

The specifications in subclause 8.3.6.8 and its subclauses apply.

H.4.3.6.9 Atlas tile layer RBSP syntax

The specifications in subclause 8.3.6.9 and its subclauses apply.

H.4.3.6.10 RBSP trailing bit syntax

The specifications in subclause 8.3.6.10 and its subclauses apply.

H.4.3.6.11 Atlas tile header syntax

The specifications in subclause 8.3.6.11 and its subclauses apply.

H.4.3.6.12 Reference list structure syntax

The specifications in subclause 8.3.6.12 and its subclauses apply.

H.4.3.7 Atlas tile data unit syntax

The specifications in subclause 8.3.7 and its subclauses apply.

H.4.3.8 Supplemental enhancement information message syntax

The specifications in subclause 8.3.8 and its subclauses apply.

H.4.4 Semantics

H.4.4.1 General

The specifications in subclause 8.4.1 and its subclauses apply.

H.4.4.2 V3C unit semantics

The specifications in subclause 8.4.2 and its subclauses apply.

H.4.4.3 Byte alignment semantics

The specifications in subclause 8.4.3 and its subclauses apply.

H.4.4.4 V3C parameter set semantics

The specifications in subclause 8.4.4 and its subclauses apply.

H.4.4.5 NAL unit semantics

The specifications in subclause 8.4.5 and its subclauses apply.

H.4.4.6 Raw byte sequence payloads, trailing bits and byte alignment semantics

H.4.4.6.1 Atlas sequence parameter set RBSP semantics

H.4.4.6.1.1 General atlas sequence parameter set RBSP semantics

The specifications in subclause 8.4.6.1.1 and its subclauses apply, with the following modifications:

asps_max_number_projections_minus1 plus 1 specifies the maximum value of pdu_projection_id[tileID][p] in the patch_data_unit( ) syntax structure for a patch with index p in a tile with tile ID equal to tileID. When asps_extended_projection_enabled_flag is set to 1, asps_max_number_projections_minus1 shall be equal to 17. When asps_max_number_projections_minus1 is not present, it shall be inferred to be equal to 5.

H.4.4.6.1.2 Point local reconstruction information syntax

The specifications in subclause 8.4.6.1.2 apply.

H.4.4.6.1.3 ASPS V-PCC extension semantics

asps_vpcc_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 indexed map associated with the same patch. asps_vpcc_remove_duplicate_point_enabled_flag equal to 0 indicates that all points shall be reconstructed.

NOTE This process only caters for duplicated points that can be associated with a single patch because of the presence of multiple maps. Duplicated points due to other projected patches or RAW and EOM patches are not accounted for in this process.

asps_vpcc_surface_thickness_minus1 plus 1 specifies the maximum absolute difference between an explicitly coded depth value and interpolated depth value when asps_pixel_deinterleaving_enabled_flag or asps_plr_enabled_flag is equal to 1.

H.4.4.6.2 Atlas frame parameter set RBSP semantics

The specifications in subclause 8.4.6.2 and its subclauses apply.

H.4.4.6.3 Atlas adaptation parameter set RBSP semantics

H.4.4.6.3.1 General atlas adaptation parameter set RBSP semantics

The specifications in subclause 8.4.6.3 apply.

H.4.4.6.3.2 AAPS V-PCC extension semantics

aaps_vpcc_camera_parameters_present_flag equal to 1 specifies that camera parameters shall be present in the current atlas adaptation parameter set. aaps_vpcc_camera_parameters_present_flag equal to 0 specifies that camera parameters for the current adaptation parameter set shall not be present.

H.4.4.6.3.3 Atlas camera parameters semantics

This subclause describes camera parameters related to scale, offset and rotation that may be used for reconstruction operations after decoding as described in subclause H.9.

acp_camera_model indicates the camera model for point cloud frames that are associated with the current adaptation parameter set, as listed in Table H.1. The value of acp_camera_model shall be equal to 0 or 1 in bitstreams conforming to this edition of this document. Other values of acp_camera_model are reserved for future use by ISO/IEC. Decoders conforming to this edition of this document shall ignore reserved values of acp_camera_model.

Table H.1 — Interpretation of acp_camera_model

acp_camera_model	Name of acp_camera_model
0	UNSPECIFIED
1	Orthographic camera model
2 - 255	RESERVED

acp_scale_enabled_flag equal to 1 indicates that scale parameters for the current camera model are present. acp_scale_enabled_flag equal to 0 indicates that scale parameters for the current camera model are not present. When acp_scale_enabled_flag is not present, it shall be inferred to be equal to 0.

acp_offset_enabled_flag equal to 1 indicates that offset parameters for the current camera model are present. acp_offset_enabled_flag equal to 0 indicates that offset parameters for the current camera model are not present. When acp_offset_enabled_flag is not present, it shall be inferred to be equal to 0.

acp_rotation_enabled_flag equal to 1 indicates that rotation parameters for the current camera model are present. acp_rotation_enabled_flag equal to 0 indicates that rotation parameters for the current camera model are not present. When acp_rotation_enabled_flag is not present, it shall be inferred to be equal to 0.

acp_scale_on_axis[d] specifies the value of the scale, Scale[d], along the d axis for the current camera model in increments of 2^{-16}. The value of d is in the range of 0 to 2, inclusive, with the values of 0, 1 and 2 corresponding to the X, Y and Z axis, respectively. The value of acp_scale_on_axis[d], shall be in the range of 1 to $2^{32} - 1$, inclusive. When acp_scale_on_axis[d] is not present, it shall be inferred to be equal to 2^{16}. The value of Scale[d] is computed as follows:

$$\text{Scale}[\,d\,] = \text{acp\_scale\_on\_axis}[\,d\,] \div 2^{16}$$

A ScaleMatrix can be represented as follows:

$$\text{ScaleMatrix} = \begin{bmatrix} \text{Scale}[0] & 0 & 0 & 0 \\ 0 & \text{Scale}[1] & 0 & 0 \\ 0 & 0 & \text{Scale}[2] & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

acp_offset_on_axis[d] indicates the value of the offset, Offset[d], along the d axis for the current camera model in increments of 2^{-16}. The value of acp_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 acp_offset_on_axis[d] is not present, it shall be inferred to be equal to 0.

Offset[d] = acp_offset_on_axis[d] ÷ 2^{16}

An offset matrix can be represented as follows:

$$\text{OffsetMatrix} = \begin{bmatrix} 1 & 0 & 0 & \text{Offset}[0] \\ 0 & 1 & 0 & \text{Offset}[1] \\ 0 & 0 & 1 & \text{Offset}[2] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

acp_rotation_qx specifies the x component, qX, for the rotation of the current camera model using the quaternion representation. The value of acp_rotation_qx shall be in the range of -2^{14} to 2^{14}, inclusive. When acp_rotation_qx is not present, its value shall be inferred to be equal to 0. The value of qX is computed as follows:

qX = acp_rotation_qx ÷ 2^{14}

acp_rotation_qy specifies the y component, qY, for the rotation of the current camera model using the quaternion representation. The value of acp_rotation_qy shall be in the range of -2^{14} to 2^{14}, inclusive. When acp_rotation_qy is not present, its value shall be inferred to be equal to 0. The value of qY is computed as follows:

qY = acp_rotation_qy ÷ 2^{14}

acp_rotation_qz specifies the z component, qZ, for the rotation of the current camera model using the quaternion representation. The value of acp_rotation_qz shall be in the range of -2^{14} to 2^{14}, inclusive. When acp_rotation_qz is not present, its value shall be inferred to be equal to 0. The value of qZ is computed as follows:

qZ = acp_rotation_qz ÷ 2^{14}

The fourth component, qW, for the rotation of the current camera model using the quaternion representation is calculated as follows:

qW = Sqrt(1 – ($qX^2 + qY^2 + qZ^2$))

NOTE In the context of this document qW is always positive. If a negative qW is desired, all three syntax elements (acp_rotation_qx, acp_rotation_qy and acp_rotation_qz) can be signalled with an opposite sign, which is equivalent.

A unit quaternion can be represented as a rotation matrix R as follows:

$$\text{RotationMatrix} = \begin{bmatrix} 1-2*\left(qY^2+qZ^2\right) & 2*(qX*qY-qZ*qW) & 2*(qX*qZ+qY*qW) & 0 \\ 2*(qX*qY+qZ*qW) & 1-2*\left(qX^2+qZ^2\right) & 2*(qY*qZ-qX*qW) & 0 \\ 2*(qX*qZ-qY*qW) & 2*(qY*qZ+qX*qW) & 1-2*\left(qX^2+qY^2\right) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It is a requirement of bitstream conformance that $qX^2 + qY^2 + qZ^2 <= 1$.

H.4.4.6.4 Supplemental enhancement information RBSP semantics

The specifications in subclause 8.4.6.4 and its subclauses apply.

H.4.4.6.5 Access unit delimiter RBSP semantics

The specifications in subclause 8.4.6.5 and its subclauses apply.

H.4.4.6.6 End of sequence RBSP semantics

The specifications in subclause 8.4.6.6 and its subclauses apply.

H.4.4.6.7 End of bitstream RBSP semantics

The specifications in subclause 8.4.6.7 and its subclauses apply.

H.4.4.6.8 Filler data RBSP semantics

The specifications in subclause 8.4.6.8 and its subclauses apply.

H.4.4.6.9 Atlas tile layer RBSP semantics

The specifications in subclause 8.4.6.9 and its subclauses apply.

H.4.4.6.10 RBSP trailing bit semantics

The specifications in subclause 8.4.6.10 and its subclauses apply.

H.4.4.6.11 Atlas tile header semantics

The specifications in subclause 8.4.6.11 and its subclauses apply.

H.4.4.6.12 Reference list structure semantics

The specifications in subclause 8.4.6.12 and its subclauses apply.

H.4.4.7 Atlas tile data unit semantics

The specifications in subclause 8.4.7 and its subclauses apply.

H.4.4.8 Supplemental enhancement information message semantics

The specifications in subclause 8.4.8 and its subclauses apply.

H.5 Decoding process

The specifications in Clause 9 and its subclauses apply.

H.5.1 General decoding process

The specifications in subclause 9.1 apply.

H.5.2 Atlas data decoding process

H.5.2.1 General atlas data decoding process

The specifications in subclause 9.2.1 and its subclauses apply

H.5.2.2 Decoding process for a coded atlas frame

The specifications in subclause 9.2.2 and its subclauses apply

H.5.2.3 Atlas NAL unit decoding process

The specifications in subclause 9.2.3 and its subclauses apply

H.5.2.4 Atlas tile header decoding process

The specifications in subclause 9.2.4 and its subclauses apply

H.5.2.5 Decoding process for patch data units

H.5.2.5.1 General decoding process for patch data units

The specifications in subclause 9.2.5.1 apply with the following modifications and additions:

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

TilePatchAxisV[tileID][p] is the index of the bi-tangent to the projection plane for the current patch with patch index p. The value of TilePatchAxisV[tileID][p] shall be in range of 0 to 2, inclusive.

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

TilePatchProjectionFlag[tileID][p] specifies on which one of two projection planes, indicated by the TilePatchAxisD[tileID][p] plane, the current patch with patch index p is to be projected.

TilePatch45DegreeMode[tileID][p] specifies the mode that indicates the axis around which the patch projection rotation is applied for the patch with index p.

If atdu_patch_mode[tileID][p] is equal to I_INTRA or P_INTRA, then the process for decoding intra coded patches in subclause H.5.2.5.2 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu_patch_mode[tileID][p] is equal to P_SKIP, then the process for decoding skip coded patches in subclause H.5.2.5.3 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu_patch_mode[tileID][p] is equal to P_MERGE, then the process for decoding merge coded patches in subclause H.5.2.5.4 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu_patch_mode[tileID][p] is equal to P_INTER, then the process for decoding inter coded patches in subclause H.5.2.5.5 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu_patch_mode[tileID][p] is equal to I_RAW or P_RAW, then the process for decoding RAW coded patches in subclause H.5.2.5.6 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu_patch_mode[tileID][p] is equal to I_EOM or P_EOM, then the process for decoding EOM coded patches in subclause H.5.2.5.7 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

These additional variables are initially set as follows:

```
TilePatchAxisU[ tileID ][ p ] = 0
TilePatchAxisV[ tileID ][ p ] = 0
TilePatchAxisD[ tileID ][ p ] = 0
TilePatchProjectionFlag[ tileID ][ p ] = 0
TilePatch45DegreeMode[ tileID ][ p ] = 0
```

H.5.2.5.2 Decoding process for patch data units coded in intra mode

The specifications in subclause 9.2.5.2 apply with the following modifications and additions:

In addition to the variables derived in subclause 9.2.5.2 the variables TilePatchAxisU[tileID][p], TilePatchAxisV[tileID][p], TilePatchAxisD[tileID][p], TilePatch45DegreeMode[tileID][p], and TilePatchProjectionFlag[tileID][p], for patch with index p in the atlas tile with index tileID, are derived as follows:

projectionPlane = pdu_projection_id[tileID][p] (H.1)

rotationAxisMode = Max(0, (projectionPlane − 2)/4) (H.2)

If rotationAxisMode is equal to 0 then the following applies:

TilePatchProjectionFlag[tileID][p] = Clip3(0, 1, projectionPlane / 3) (H.3)

TilePatchAxisD[tileID][p] = projectionPlane % 3 (H.4)

Otherwise:

TilePatchProjectionFlag[tileID][p] = Min(1, ((projectionPlane − 6) % 4) / 2) (H.5)

```
if( rotationAxisMode == 1 )
        TilePatchAxisD[ tileID ][ p ] = 2 * ( projectionPlane % 2 )
else if( rotationAxisMode == 2 )
        TilePatchAxisD[ tileID ][ p ] = 2 − ( projectionPlane % 2 )          (H.6)
else
        TilePatchAxisD[ tileID ][ p ] = 1 − ( projectionPlane % 2 )
```

TilePatchAxisU[tileID][p] = (TilePatchAxisD[tileID][p] == 2) ? 0 : 2 (H.7)

TilePatchAxisV[tileID][p] = (TilePatchAxisD[tileID][p] == 1) ? 0 : 1 (H.8)

The values of the variables TilePatchAxisU[tileID][p], TilePatchAxisV[tileID][p], TilePatchAxisD[tileID][p], TilePatch45DegreeMode[tileID][p], and TilePatchProjectionFlag[tileID][p] that are derived from Formulae (H.1) to (H.8) are also presented in tabular form in Table H.2.

Table H.2 — Derivation of patch projection variables based on pdu_projection_id[tileID][p] values

	pdu_projection_id[tileID][p] values																	
Projection variables	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
TilePatchAxisU[tileID][p]	2	2	0	2	2	0	2	0	2	0	0	2	0	2	2	2	2	2
TilePatchAxisV[tileID][p]	1	0	1	1	0	1	1	1	1	1	1	0	1	0	0	1	0	1
TilePatchAxisD[tileID][p]	0	1	2	0	1	2	0	2	0	2	2	1	2	1	1	0	1	0
TilePatchProjectionFlag[tileID][p]	0	0	0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1
TilePatch45DegreeMode[tileID][p]	0	0	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3

H.5.2.5.3 Decoding process for patch data units coded in skip prediction mode

The specifications in subclause 9.2.5.3 apply with the following modifications and additions:

First, refIdx is set to 0.

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

Then the corresponding patch index, RefPatchIdx, in the atlas frame corresponding to the first entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[refIdx] for the current tile is computed as:

RefPatchIdx = PredictorIdx (H.9)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause H.5.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, refPatchEomPatchCount, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatch45DegreeMode, and refPatchProjectionFlag, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

If refPatchType is equal to projected then, these additional variables are derived:

TilePatchAxisU[tileID][p] = refPatchAxisU (H.10)

TilePatchAxisV[tileID][p] = refPatchAxisV (H.11)

TilePatchAxisD[tileID][p] = refPatchAxisD (H.12)

TilePatch45DegreeMode[tileID][p] = refPatch45DegreeMode (H.13)

TilePatchProjectionFlag[tileID][p] = refPatchProjectionFlag (H.14)

H.5.2.5.4 Decoding process for patch data units coded in merge prediction mode

The specifications in subclause 9.2.5.4 apply with the following modifications and additions:

First, the reference atlas frame index, refIdx, is derived as mpdu_ref_index[tileID][p].

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the (refIdx + 1)-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[refIdx].

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

Then the corresponding patch index, RefPatchIdx, in the reference atlas frame, refAtlasFrm, for the current tile is computed as:

$$\text{RefPatchIdx} = \text{PredictorIdx} \tag{H.15}$$

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause H.5.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, refPatchEomPatchCount, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatch45DegreeMode, and refPatchProjectionFlag, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

It is a requirement of bitstream conformance that refPatchType is equal to projected.

Then, these additional variables are derived:

$$\text{TilePatchAxisU[tileID][p]} = \text{refPatchAxisU} \tag{H.16}$$

$$\text{TilePatchAxisV[tileID][p]} = \text{refPatchAxisV} \tag{H.17}$$

$$\text{TilePatchAxisD[tileID][p]} = \text{refPatchAxisD} \tag{H.18}$$

$$\text{TilePatch45DegreeMode[tileID][p]} = \text{refPatch45DegreeMode} \tag{H.19}$$

$$\text{TilePatchProjectionFlag[tileID][p]} = \text{refPatchProjectionFlag} \tag{H.20}$$

H.5.2.5.5 Decoding process for patch data units coded in inter prediction mode

H.5.2.5.5.1 General

The specifications in subclause 9.2.5.5.1 apply with the following modifications and additions:

First, the reference atlas frame index, refIdx, is derived as ipdu_ref_index[tileID][p].

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the (refIdx + 1)-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[refIdx].

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

Then the corresponding patch index, RefPatchIdx, in the reference atlas frame, refAtlasFrm, for the current tile is computed as:

RefPatchIdx = PredictorIdx + ipdu_patch_index[tileID][p] (H.21)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause H.5.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, refPatchEomPatchCount, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatch45DegreeMode, and refPatchProjectionFlag, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

It is a requirement of bitstream conformance that refPatchType is equal to projected.

Then, these additional variables are derived:

TilePatchAxisU[tileID][p] = refPatchAxisU (H.22)

TilePatchAxisV[tileID][p] = refPatchAxisV (H.23)

TilePatchAxisD[tileID][p] = refPatchAxisD (H.24)

TilePatch45DegreeMode[tileID][p] = refPatch45DegreeMode (H.25)

TilePatchProjectionFlag[tileID][p] = refPatchProjectionFlag (H.26)

H.5.2.5.5.2 Derivation of inter reference patch parameters

H.5.2.5.5.2.1 General derivation of inter reference patch parameters

The specifications in subclause 9.2.5.5.2.1 apply with the following modifications and additions:

Outputs to this process are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, refPatchEomPatchCount, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatchProjectionFlag, and refPatch45DegreeMode, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

First, the patch, refPatch, is determined that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the (refIdx + 1)-th entry of the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[refIdx].

Then, the outputs of this process are derived based on the associated parameters of the patch, refPatch, as follows:

refPatchType = TilePatchType[tileID][refPatchIdx] (H.27)

refPatchInAuxVideo = TilePatchInAuxVideo[tileID][refPatchIdx] (H.28) refPatch2dPosX = TilePatch2dPosX[tileID][refPatchIdx] (H.29)

refPatch2dPosY = TilePatch2dPosY[tileID][refPatchIdx] (H.30)

refPatch2dSizeX = TilePatch2dSizeX[tileID][refPatchIdx] (H.31)

refPatch2dSizeY = TilePatch2dSizeY[tileID][refPatchIdx] (H.32)

If refPatchType is equal to projected, then the process in subclause H.5.2.5.5.2.2 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatchProjectionID, refPatch3dOffsetD, refPatch3dRangeD, refPatchOrientationIndex, refPatchLoDScaleX, refPatchLoDScaleY, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatchProjectionFlag, and refPatch45DegreeMode, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

If refPatchType is equal to RAW, then the process in subclause H.5.2.5.5.2.3 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, and refPatchRawPoints.

If refPatchType is equal to EOM, then the process in subclause H.5.2.5.5.2.4 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variable refPatchEomPatchCount, and the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints.

H.5.2.5.5.2.2 Reference patches in projected mode

The specifications in subclause 9.2.5.5.2.2 apply with the following modifications and additions:

Outputs to this process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatchProjectionFlag, and refPatch45DegreeMode, and, if asps_plr_enabled_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

These additional output variables are derived:

refPatchAxisU = TilePatchAxisU[tileID][refPatchIdx] (H.33)

refPatchAxisV = TilePatchAxisV[tileID][refPatchIdx] (H.34)

refPatchAxisD = TilePatchAxisD[tileID][refPatchIdx] (H.35)

refPatch45DegreeMode = TilePatch45DegreeMode[tileID][refPatchIdx] (H.36)

refPatchProjectionFlag = TilePatchProjectionFlag[tileID][refPatchIdx] (H.37)

H.5.2.5.5.2.3 Reference patches in RAW mode

The specifications in subclause 9.2.5.5.2.3 apply

H.5.2.5.5.2.4 Reference patches in EOM mode

The specifications in subclause 9.2.5.5.2.4 apply

H.5.2.5.6 Decoding process for patch data units coded in RAW mode

The specifications in subclause 9.2.5.6 apply

H.5.2.5.7 Decoding process for patch data units coded in EOM mode

The specifications in subclause 9.2.5.7 apply

H.5.2.6 Decoding process of the block to patch map

The specifications in subclause 9.2.6 and its subclauses apply

H.5.2.7 Conversion of tile level information to atlas level information

H.5.2.7.1 General

The specifications in subclause 9.2.7.1 apply

H.5.2.7.2 Conversion of tile level blockToPatch information to atlas level blockToPatch information

The specifications in subclause 9.2.7.2 apply.

H.5.2.7.3 Conversion of tile level patch information to atlas level patch information

H.5.2.7.3.1 General

The specifications in subclause 9.2.7.3.1 apply.

H.5.2.7.3.2 Process of copying common patch parameters from a tile to an atlas representation

The specifications in subclause 9.2.7.3.2 apply.

H.5.2.7.3.3 Process of copying application specific patch parameters from a tile to an atlas representation

The specifications in subclause 9.2.7.3.3 apply with the following modifications and additions:

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

```
ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch ) {
    blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )
    if( AtlasPatchType[ atlasPatchIdx ] == PROJECTED ) {
        AtlasTotalNumProjPatches += 1
        AtlasPatchProjectionFlag[ atlasPatchIdx ] = TilePatchProjectionFlag[ tileID ][ p ]
        AtlasPatchAxisU[ atlasPatchIdx ] = TilePatchAxisU[ tileID ][ p ]
        AtlasPatchAxisV[ atlasPatchIdx ] = TilePatchAxisV[ tileID ][ p ]
        AtlasPatchAxisD[ atlasPatchIdx ] = TilePatchAxisD[ tileID ][ p ]
        AtlasPatch45DegreeMode[ atlasPatchIdx ] = TilePatch45DegreeMode[ tileID ][ p ]
        for( m = 0; m < asps_map_count_minus1 + 1; m++ ) {
            if( plri_map_present_flag[ m ] == 1 ) {
                AtlasPlrdLevel[ atlasPatchIdx ][ m ] = TilePatchPlrdLevel[ tileID ][ p ][ m ]
                if( AtlasPlrdLevel[ atlasPatchIdx ][ m ] == 0 ) {
                    for( j = 0; j < blockCnt; j++ ) {
                        AtlasPlrdPresentBlockFlag[ atlasPatchIdx ][ m ][ j ] =
                            TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ]
                        if( AtlasPlrdPresentBlockFlag[ atlasPatchIdx ][ m ][ j ] == 1 )
```

```
                                        AtlasPlrdBlockModeMinus1[ atlasPatchIdx ][ m ][ j ] =
                                                TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ]
                                }
                        } else {
                                AtlasPlrdPresentFlag[ atlasPatchIdx ][ m ] =
                                        TilePatchPlrdPresentFlag[ tileID ][ p ][ m ]
                                if( AtlasPlrdPresentFlag[ atlasPatchIdx ][ m ] == 1 )
                                        AtlasPlrdModeMinus1[ atlasPatchIdx ][ m ] =
                                                TilePatchPlrdModeMinus1[ tileID ][ p ][ m ]
                        }
                }
        }
    }
    if( AtlasPatchType[ atlasPatchIdx ] == EOM ) {
        AtlasTotalNumEomPatches += 1
        for( i = 0; i < AtlasPatchEomPatchCount[ atlasPatchIdx ]; i++ ) {
            NumEomPoints += TilePatchEomPoints[ tileID ][ p ][ i ]
            AtlasPatchEomPoints[ atlasPatchIdx ][ i ] = TilePatchEomPoints[ tileID ][ p ][ i ]
            AtlasPatchAssociatedPatchIndex[ atlasPatchIdx ][ i ] =
                TilePatchAssociatedPatchIndex[ tileID ][ p ][ i ] + offsetPatch
        }
    }
    if( AtlasPatchType[ atlasPatchIdx ] == RAW ) {
        AtlasTotalNumRawPatches += 1
        NumRawPoints += TilePatchRawPoints[ tileID ][ p ]
    }
}
```

H.5.3 Occupancy video decoding process

The specifications in subclause 9.3 apply.

H.5.4 Geometry video decoding process

The specifications in subclause 9.4 apply.

H.5.5 Attribute video decoding process

The specifications in subclause 9.5 apply.

H.5.6 Sub-bitstream extraction process

The specifications in subclause 9.6 apply.

H.6 Pre-reconstruction process

H.6.1 General

The pre-reconstruction process depends on the profiles defined in Clause H.11.

The processes described in this subclause are invoked for decoded occupancy frames at nominal resolution, and syntax elements associated with the same atlas ID, identified by a variable OccSynAtlasID.

The pre-reconstruction process, for the current point cloud frame with a composition time index compTimeIdx, takes as inputs the syntax elements and upper-case variables from Clauses H.4, H.5 and Annex B.

If a bitstream contains an occupancy synthesis SEI message with a value of os_method_type[k] larger than zero at time instance compTimeIdx, and the point cloud reconstruction system has selected to use this k-th occupancy synthesis instance, subclause H.6.2.2 is invoked.

H.6.2 Occupancy synthesis process

H.6.2.1 General

If os_method_type[k] is equal to 1, which indicates the patch border filtering method, then subclause H.6.2.2 is invoked with OccFramesNF[compTimeIdx][0] and GeoFramesNF[0][compTimeIdx][0] as inputs and the modified array OccFramesNF[compTimeIdx][0] as output.

NOTE If subclause H.6.2.2 is not invoked, OccFramesNF[compTimeIdx][0] is not modified.

H.6.2.2 Patch border filtering (PBF)

H.6.2.2.1 General

Inputs to this process are:

— a variable k, indicating the selected occupancy synthesis filter instance,

— a 2D array oFrame, of size asps_frame_height× asps_frame_width, and

— a 2D array gFrame, of size asps_frame_height× asps_frame_width.

Output of this process is the modified array oFrame.

NOTE It is possible that this process will not be effective when the occupancy video is coded at the nominal resolution.

The following applies:

— Let the variable boxExtension be set to 8.

— If asps_eom_patch_enabled_flag is equal to 1, the array oFrame is returned without modification.

— Otherwise (asps_eom_patch_enabled_flag is equal to 0), the following process is performed:

  — First, the following arrays are defined:

    — borderPointCnt, a 1D array of size AtlasTotalNumPatches,

    — neighPatchCnt, a 1D array of size AtlasTotalNumPatches,

    — neighbouringPatches, a 2D array of size AtlasTotalNumPatches × AtlasTotalNumPatches,

    — borderPoints, a 3D array of size AtlasTotalNumPatches × AspsFrameSize × 3,

    — bBox, a 3D array of size AtlasTotalNumPatches × 2 × 3, and

    — neighbourDepth, a 2D array of size asps_frame_height × asps_frame_width.

  NOTE The allocated sizes of neighbouringPatches and borderPoints use the maximum possible values according to this document. Different, more efficient implementations can be used.

- For pIdx = 0 .. AtlasTotalNumPatches − 1, the following applies:
  - subclause H.6.2.2.2 is invoked with pIdx, oFrame, and gFrame, as inputs, and borderPointCnt[pIdx], borderPoints[pIdx], and bBox[pIdx] as outputs.
- For pIdx = 0 .. AtlasTotalNumPatches − 1, the following applies:
  - subclause H.6.2.2.3 is invoked with pIdx and bBox as inputs, and neighPatchCnt[pIdx] and neighbouringPatches[pIdx] as outputs,
- subclause H.6.2.2.4 is invoked with variables PbfThreshold[k], boxExtension, and borderPointCnt, and arrays borderPoints, bBox, neighPatchCnt, and neighbouringPatches, as inputs, and the array neighbourDepth as output,
- subclause H.6.2.2.5 is invoked with variables PbfPasses[k], PbfSizeX[k], and PbfSizeY[k], and arrays oFrame, gFrame, and neighbourDepth as inputs, and the modified array oFrame as output.

H.6.2.2.2 Bounding box and border point determination

Inputs to this process are:

- a variable pIdx, indicating the patch index,
- a 2D array oFrame, of size asps_frame_height × asps_frame_width, corresponding to the occupancy frame in the nominal format, and
- a 2D array gFrame, of size asps_frame_height × asps_frame_width, corresponding to the geometry frame in the nominal format.

Outputs of this process are:

- a variable borderPointCnt, indicating the number of border points,
- a 2D array borderPoints, of size AspsFrameSize × 3, and
- a 2D array bBox, of size 2 × 3.

Let borderPointCnt and bBox be initialized as follows:

```
borderPointCnt = 0.
for( k = 0; k < 3; k++ ) {
    bBox[ 0 ][ k ] = 0xFFFF
    bBox[ 1 ][ k ] = 0
}
```

Let yStart, yEnd, xStart and xEnd be computed as follows:

```
yStart = AtlasPatch2dPosY[ pIdx ]
yEnd = AtlasPatch2dPosY[ pIdx ] + AtlasPatch2dSizeY[ pIdx ]
xStart = AtlasPatch2dPosX[ pIdx ]
xEnd = AtlasPatch2dPosX[ pIdx ] + AtlasPatch2dSizeX[ pIdx ]
```

Then the following applies:

```
for( y = yStart; y < yEnd; y++ ) {
    yBlock = y / PatchPackingBlockSize
    for( x = xStart; x < xEnd; x++ ) {
        xBlock = x / PatchPackingBlockSize
        isBoundaryPoint = 0
```