



DPC: 25/30512678 DC

**BSI Group Headquarters**

389 Chiswick High Road London W4 4AL

Tel: +44 (0)20 8996 9000

Fax: +44 (0)20 8996 7400

www.bsigroup.com

Date: 08 July 2025

Origin: International

**Latest date for receipt of comments: 31 August 2025**

Project No. 2025/01129

Responsible committee: IST/37 Coding of picture, audio, multimedia and hypermedia information

Interested committees:

Title: Draft BS ISO/IEC 23090-5 Information technology — Coded representation of immersive media. Part 5: Visual volumetric video-based coding (V3C) and video-based point cloud compression (V-PCC).

Please notify [admin.start@bsigroup.com](mailto:admin.start@bsigroup.com) if you are aware of any keywords that might assist in clarifying or identifying the standard or of the content of this standard:

- i) has any issues related to 3rd party IPR, patent or copyright
- ii) affects other national standard(s)
- iii) requires additional national guidance or information

**WARNING: THIS IS A DRAFT AND MUST NOT BE REGARDED OR USED AS A BRITISH STANDARD. THIS DRAFT IS NOT CURRENT BEYOND 31 August 2025**

This draft is issued to allow comments from interested parties; all comments will be given consideration prior to publication. No acknowledgement will normally be sent. **See overleaf for information on the submission of comments.**

No copying is allowed, in any form, without prior written permission from BSI except as permitted under the Copyright, Designs and Patent Act 1988 or for circulation within a nominating organization for briefing purposes. Electronic circulation is limited to dissemination by e-mail within such an organization by committee members.

Further copies of this draft may be purchased from BSI Shop <http://shop.bsigroup.com> or from BSI Customer Services, Tel: +44(0) 20 8996 9001 or email [cservices@bsigroup.com](mailto:cservices@bsigroup.com). British, International and foreign standards are also available from BSI Customer Services.

Information on the co-operating organizations represented on the committees referenced above may be obtained from <https://standardsdevelopment.bsigroup.com/>.

E-mail: [admin.start@bsigroup.com](mailto:admin.start@bsigroup.com)

## Introduction

This draft standard is based on national and international discussions. Your comments on this draft are invited and will assist in the preparation of the consequent standard.

For international standards, comments will be reviewed by the relevant UK national committee before sending the consensus UK vote and comments to the international committee, which will then decide appropriate action. If the international standard is approved, it is usual for the text to be published as a British Standard.

For national standards, comments will be reviewed by the relevant UK national committee and the resulting standards published as a British Standard.

## UK Vote

Please indicate whether you consider the UK should submit a negative (with supporting technical reasons) or positive vote on this draft. Please indicate if you are aware of any reason why this draft standard should not be published as a British Standard.

## Submission of Comments

- **Annotated drafts are not acceptable and will be rejected.**
- All comments should be submitted online at <https://standardsdevelopment.bsigroup.com/>. You will need to register in order to comment.

## Template for comments and secretariat observations

|                  |                        |
|------------------|------------------------|
| Date: xx/xx/20xx | Document: ISO/DIS xxxx |
|------------------|------------------------|

| 1  | 2                                          | (3)                          | 4               | 5                                                                                                                                  | (6)                                                                      | 7                                                  |
|----|--------------------------------------------|------------------------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------|----------------------------------------------------|
| MB | Clause No./ Subclause No./Annex (e.g. 3.1) | Paragraph/ Figure/Table/Note | Type of comment | Comment (justification for change) by the MB                                                                                       | Proposed change by the MB                                                | Secretariat observations on each comment submitted |
|    | 3.1                                        | Definition 1                 | ed              | Definition is ambiguous and needs clarifying.                                                                                      | Amend to read '...so that the mains connector to which no connection...' |                                                    |
|    | 6.4                                        | Paragraph 2                  | te              | The use of the UV photometer as an alternative cannot be supported as serious problems have been encountered in its use in the UK. | Delete reference to UV photometer.                                       |                                                    |



# DRAFT International Standard

## ISO/IEC DIS 23090-5

### Information technology — Coded representation of immersive media —

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

*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)*

ICS: 35.040.40

ISO/IEC JTC 1/SC 29

Secretariat: JISC

Voting begins on:  
**2025-07-08**

Voting terminates on:  
**2025-09-30**

This document is circulated as received from the committee secretariat.

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENTS AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

Reference number  
ISO/IEC DIS 23090-5:2025(en)

Full document details: 25/30512678 DC

© ISO/IEC 2025  **Documents**  
Document Sharing Hub



**COPYRIGHT PROTECTED DOCUMENT**

© ISO/IEC 2025

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office  
CP 401 • Ch. de Blandonnet 8  
CH-1214 Vernier, Geneva  
Phone: +41 22 749 01 11  
Email: [copyright@iso.org](mailto:copyright@iso.org)  
Website: [www.iso.org](http://www.iso.org)

Published in Switzerland

# Contents

Page

|                                                                                        |           |
|----------------------------------------------------------------------------------------|-----------|
| <b>Foreword</b>                                                                        | <b>v</b>  |
| <b>Introduction</b>                                                                    | <b>vi</b> |
| <b>1 Scope</b>                                                                         | <b>1</b>  |
| <b>2 Normative references</b>                                                          | <b>1</b>  |
| <b>3 Terms and definitions</b>                                                         | <b>2</b>  |
| <b>4 Abbreviated terms</b>                                                             | <b>16</b> |
| <b>5 Conventions</b>                                                                   | <b>17</b> |
| 5.1 General                                                                            | 17        |
| 5.2 Arithmetic operators                                                               | 17        |
| 5.3 Logical operators                                                                  | 17        |
| 5.4 Relational operators                                                               | 18        |
| 5.5 Bit-wise operators                                                                 | 18        |
| 5.6 Assignment operators                                                               | 18        |
| 5.7 Other operators                                                                    | 19        |
| 5.8 Mathematical functions                                                             | 19        |
| 5.9 Order of operation precedence                                                      | 19        |
| 5.10 Variables, syntax elements, and tables                                            | 20        |
| 5.11 Text description of logical operations                                            | 21        |
| 5.12 Processes                                                                         | 23        |
| <b>6 Overall V3C characteristics, decoding operations, and post-decoding processes</b> | <b>23</b> |
| 6.1 V3C characteristics                                                                | 23        |
| 6.2 V3C bitstream characteristics, decoding operations, and post-decoding processes    | 26        |
| 6.3 Extensibility of V3C: V-PCC, MIV and V-DMC                                         | 27        |
| <b>7 Bitstream format, partitioning, and scanning processes</b>                        | <b>27</b> |
| 7.1 General                                                                            | 27        |
| 7.2 V3C bitstream formats                                                              | 28        |
| 7.3 NAL bitstream formats                                                              | 28        |
| 7.4 Partitioning of atlas frames into tiles                                            | 28        |
| <b>8 Syntax and semantics</b>                                                          | <b>29</b> |
| 8.1 Method of specifying syntax in tabular form                                        | 29        |
| 8.1.1 General                                                                          | 29        |
| 8.1.2 Example of the syntax specification format                                       | 30        |
| 8.2 Specification of syntax functions and descriptors                                  | 31        |
| 8.3 Syntax in tabular form                                                             | 33        |
| 8.3.1 General                                                                          | 33        |
| 8.3.2 V3C unit syntax                                                                  | 34        |
| 8.3.3 Byte alignment syntax                                                            | 36        |
| 8.3.4 V3C parameter set syntax                                                         | 36        |
| 8.3.5 NAL unit syntax                                                                  | 42        |
| 8.3.6 Raw byte sequence payloads, trailing bits, and byte alignment syntax             | 42        |
| 8.3.7 Atlas tile data unit syntax                                                      | 50        |
| 8.3.8 Supplemental enhancement information message syntax                              | 54        |
| 8.4 Semantics                                                                          | 54        |
| 8.4.1 General                                                                          | 54        |
| 8.4.2 V3C unit semantics                                                               | 54        |
| 8.4.3 Byte alignment semantics                                                         | 57        |
| 8.4.4 V3C parameter set semantics                                                      | 57        |
| 8.4.5 NAL unit semantics                                                               | 67        |
| 8.4.6 Raw byte sequence payloads, trailing bits, and byte alignment semantics          | 76        |
| 8.4.7 Atlas tile data unit semantics                                                   | 92        |
| 8.4.8 Supplemental enhancement information message semantics                           | 99        |

|           |                                                                                |            |
|-----------|--------------------------------------------------------------------------------|------------|
| <b>9</b>  | <b>Decoding process</b>                                                        | <b>100</b> |
| 9.1       | General decoding process                                                       | 100        |
| 9.2       | Atlas data decoding process                                                    | 101        |
| 9.2.1     | General atlas data decoding process                                            | 101        |
| 9.2.2     | Decoding process for a coded atlas frame                                       | 101        |
| 9.2.3     | Atlas NAL unit decoding process                                                | 103        |
| 9.2.4     | Atlas tile header decoding process                                             | 103        |
| 9.2.5     | Decoding process for patch data units                                          | 108        |
| 9.2.6     | Decoding process of the block to patch map                                     | 123        |
| 9.2.7     | Conversion of tile level patch information to atlas level patch information    | 124        |
| 9.3       | Occupancy video decoding process                                               | 127        |
| 9.4       | Geometry video decoding process                                                | 128        |
| 9.5       | Attribute video decoding process                                               | 130        |
| 9.6       | Packed video decoding process                                                  | 132        |
| 9.7       | Common atlas data decoding process                                             | 134        |
| 9.7.1     | General common atlas data decoding process                                     | 134        |
| 9.7.2     | Decoding process for a coded common atlas frame                                | 134        |
| 9.7.3     | Common atlas NAL unit decoding process                                         | 134        |
| 9.7.4     | Common atlas frame order count derivation process                              | 134        |
| 9.8       | Sub-bitstream extraction process                                               | 136        |
| 9.8.1     | General                                                                        | 136        |
| 9.8.2     | V3C unit extraction                                                            | 136        |
| 9.8.3     | NAL unit extraction process                                                    | 137        |
| <b>10</b> | <b>Pre-reconstruction process</b>                                              | <b>137</b> |
| <b>11</b> | <b>Reconstruction process</b>                                                  | <b>137</b> |
| <b>12</b> | <b>Post-reconstruction process</b>                                             | <b>138</b> |
| <b>13</b> | <b>Adaptation process</b>                                                      | <b>138</b> |
| <b>14</b> | <b>Parsing process</b>                                                         | <b>138</b> |
| 14.1      | General                                                                        | 138        |
| 14.2      | Parsing process for 0-th order Exp-Golomb codes                                | 138        |
| 14.2.1    | General                                                                        | 138        |
| 14.2.2    | Mapping process for signed Exp-Golomb codes                                    | 139        |
| 14.3      | Parsing process for ASN.1 object identifiers                                   | 140        |
| 14.3.1    | Object identifier syntax                                                       | 140        |
| 14.3.2    | Object identifier semantics                                                    | 140        |
|           | <b>Annex A (normative) Profiles, tiers, and levels</b>                         | <b>141</b> |
|           | <b>Annex B (informative) Post-decoding conversion to nominal video formats</b> | <b>149</b> |
|           | <b>Annex C (informative) V3C sample stream format</b>                          | <b>175</b> |
|           | <b>Annex D (normative) NAL sample stream format</b>                            | <b>177</b> |
|           | <b>Annex E (normative) Atlas hypothetical reference decoder</b>                | <b>179</b> |
|           | <b>Annex F (normative) Supplemental enhancement information</b>                | <b>196</b> |
|           | <b>Annex G (normative) Volumetric usability information</b>                    | <b>253</b> |
|           | <b>Annex H (normative) Video-based Point Cloud Coding</b>                      | <b>264</b> |
|           | <b>Bibliography</b>                                                            | <b>352</b> |

## 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](http://www.iso.org/directives) or [www.iec.ch/members\\_experts/refdocs](http://www.iec.ch/members_experts/refdocs)).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at [www.iso.org/patents](http://www.iso.org/patents) and <https://patents.iec.ch>. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

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](http://www.iso.org/iso/foreword.html). In the IEC, see [www.iec.ch/understanding-standards](http://www.iec.ch/understanding-standards).

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*.

This fourth edition cancels and replaces the third edition (ISO/IEC 23090-5:2025), which has been technically revised.

The main changes are as follows:

— Additions needed for ISO/IEC 23090-29:—<sup>1)</sup>.

A list of all parts in the ISO/IEC 23090 series can be found on the ISO and IEC websites.

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](http://www.iso.org/members.html) and [www.iec.ch/national-committees](http://www.iec.ch/national-committees).

---

1) Under preparation. Stage at the time of publication: ISO/IEC DIS 23090-29:—.

## 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, may 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 may 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 may also be associated with a number of attributes, such as colour, reflectance, surface normal, etc.

# 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. Furthermore, this document specifies processes that may be needed for reconstruction of visual volumetric media, and may 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.

ISO/IEC 8825-1, *Information technology — ASN.1 encoding rules — Part 1: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

ISO/IEC 10646, *Information technology — Universal coded character set (UCS)*

ISO/IEC 14496-10, *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 23002-7, *Information technology — MPEG video technologies — Part 7: Versatile supplemental enhancement information messages for coded video bitstreams*

ISO/IEC 23008-2, *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, *Information technology — Coding-independent code points — Part 2: Video*

ISO/IEC 60559:2020, *Information technology — Microprocessor Systems — Floating-Point arithmetic*

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

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

IETF RFC 1321, *The MD5 Message-Digest Algorithm*

IETF RFC 5646, *Tags for Identifying Languages*

### 3 Terms and definitions

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

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

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <https://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* (3.91) that is associated with an *ACL NAL unit* (3.5) for the purpose of decoding or other operations specified by this document

#### 3.3

##### **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

#### 3.4

##### **atlas bitstream**

sequence of bits that forms the representation of *atlas frames* (3.7) and associated data forming one or more *CASs* (3.40)

#### 3.5

##### **atlas coding layer NAL unit**

##### **ACL NAL unit**

collective term for coded atlas tile layer *NAL units* (3.89) and the subset of *NAL units* (3.89) that have reserved values of *nal\_unit\_type* that are classified as being of type class equal to ACL in this document

#### 3.6

##### **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

#### 3.7

##### **atlas frame**

2D rectangular array of atlas samples onto which *patches* (3.99) are projected and additional information related to the *patches* (3.99), corresponding to a *volumetric frame* (3.151)

#### 3.8

##### **atlas frame parameter set**

##### **AFPS**

*syntax structure* (3.133) containing *syntax elements* (3.132) that apply to zero or more entire coded *atlas frames* (3.7) as determined by the content of a *syntax element* (3.132) found in each tile header

#### 3.9

##### **atlas sample**

position on the rectangular frame onto which *patches* (3.99) that are associated with an *atlas* (3.3) are projected

#### 3.10

##### **atlas sequence**

collection of *atlas frames* (3.7)

**3.11****atlas sequence parameter set****ASPS**

*syntax structure* (3.133) containing *syntax elements* (3.132) that apply to zero or more entire *coded atlas sequences* (3.40) as determined by the content of a *syntax element* (3.132) found in the AFPS referred to by a syntax element found in each tile header

**3.12****atlas sub-bitstream**

extracted *sub-bitstream* (3.121) from the *V3C bitstream* (3.141) containing a part of an atlas NAL bitstream

**3.13****atlas unit**

set of *NAL units* (3.89) that contain all *ACL NAL units* (3.5) of a *coded atlas* (3.38) and their *associated non-ACL NAL units* (3.91)

**3.14****attribute**

scalar or vector property optionally associated with each point in a *volumetric frame* (3.151) such as colour, reflectance, surface normal, transparency, material ID, etc.

**3.15****attribute access unit**

collection of *attribute maps* (3.17) and auxiliary attribute frames, if available, for a specific *attribute* (3.14) that correspond to the same time instance

**3.16****attribute frame**

2D rectangular array created through the aggregation of *patches* (3.99) containing values of a specific *attribute* (3.14)

**3.17****attribute map**

*attribute frame* (3.16) containing *attribute patch* (3.99) information projected at a particular depth indicated by the corresponding *geometry map* (3.74)

**3.18****auxiliary attribute frame**

2D rectangular array that is associated with *RAW patches* (3.116) and *EOM patches* (3.66), and contains values of a specific *attribute* (3.14)

**3.19****auxiliary geometry frame**

2D rectangular array that is associated with *RAW patches* (3.116) and contains *geometry* (3.71) values

**3.20****auxiliary video component**

*video component* (3.138) indicated as being of auxiliary type through an appropriate flag in the *VPS* (3.147) and containing data only associated with *RAW patches* (3.116) and/or *EOM patches* (3.66)

**3.21****auxiliary video sub-bitstream**

*video sub-bitstream* (3.140) indicated as being of auxiliary type through an appropriate flag in the *VPS* (3.147) and containing data only associated with *RAW patches* (3.116) and/or *EOM patches* (3.66)

**3.22****basemesh**

mesh structure containing possibly a reduced number of *Cartesian coordinates* (3.34) and their associated connectivity, along with scalar or vector property optionally associated with each coordinate and/or set of connectivity

**3.23****basemesh access unit**

set of data units of *basemesh bitstream* (3.24) that are associated with each other according to a specified classification rule, are consecutive in decoding order, and pertaining to one particular output time

**3.24****basemesh bitstream**

*bitstream* (3.27) conforming to a mesh specification that may represent a *V3C component* (3.142) as specified by this document

**3.25****basemesh data unit**

*syntax structure* (3.133) containing basemesh data information, such as a *NAL unit* (3.89) in the context of Annex H of ISO/IEC 23090-29:2025

**3.26****basemesh sub-bitstream**

extracted *sub-bitstream* (3.121) from the *V3C bitstream* (3.141) containing a part of a *basemesh bitstream* (3.24)

**3.27****bitstream**

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

**3.28****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

**3.29****broken link access access unit****BLA access unit**

access unit in which the *coded atlas* (3.38) with *nal\_layer\_id* equal to 0 is a *BLA coded atlas* (3.30)

**3.30****broken link access coded atlas frame****BLA coded atlas****BLA atlas frame**

*IRAP coded atlas* (3.85) frame for which each *ACL NAL unit* (3.5) 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.

**3.31****byte-aligned**

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

**3.32****byte-aligned position**

position in a *bitstream* (3.27) that is *byte-aligned* (3.31)

**3.33****byte-aligned byte**

byte (3.23) that appears in a position in a *bitstream* (3.27) that is *byte-aligned* (3.31)

**3.34****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)

**3.35****clean random access access unit****CRA access unit**

access unit in which the *coded atlas* (3.38) with *nal\_layer\_id* equal to 0 is a *CRA coded atlas* (3.36)

**3.36****clean random access coded atlas frame****CRA coded atlas****CRA atlas frame**

*IRAP coded atlas* (3.85) for which each *ACLNAL unit* (3.5) 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.

**3.37****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

**3.38****coded atlas****coded atlas frame**

coded representation of an *atlas* (3.3)

**3.39****coded atlas access unit**

set of atlas *NAL units* (3.89) that are associated with each other according to a specified classification rule, are consecutive in decoding order, and contain all atlas *NAL units* (3.89) pertaining to one particular output time

**3.40****coded atlas sequence****CAS**

sequence of *coded atlas access units* (3.39), in decoding order, of an *IRAP coded atlas access unit* (3.84) with *NoOutputBeforeRecoveryFlag* equal to 1, followed by zero or more *coded atlas access units* (3.39) that are not *IRAP coded atlas access units* (3.84) with *NoOutputBeforeRecoveryFlag* equal to 1, including all subsequent *access units* (3.39) up to but not including any subsequent *coded atlas access unit* (3.39) that is an *IRAP coded atlas access unit* (3.84) with *NoOutputBeforeRecoveryFlag* equal to 1

Note 1 to entry: An IRAP coded atlas access unit may be an IDR coded atlas access unit, a BLA coded atlas access unit, or a CRA coded atlas access unit. The value of *NoOutputBeforeRecoveryFlag* is equal to 1 for each IDR coded atlas access unit, each BLA coded atlas access unit, and each CRA coded atlas access unit that is the first coded atlas access unit in the atlas bitstream in decoding order, is the first coded atlas access unit that follows an end of sequence NAL unit in decoding order, or has *HandleCraAsBlaFlag* equal to 1.

**3.41****coded common atlas access unit**

set of common atlas *non-ACL NAL units* (3.91) that are associated with each other according to a specified classification rule, are consecutive in decoding order, and contain all common atlas *NAL units* (3.89) pertaining to one particular output time

**3.42****coded common atlas frame**

coded representation of a *common atlas frame* ([3.49](#))

**3.43****coded common atlas sequence****CCAS**

sequence of *coded common atlas access units* ([3.41](#)), in decoding order, of an *IRAP coded common atlas access unit* ([3.87](#)), followed by zero or more *coded common atlas access units* ([3.41](#)) that are not *IRAP coded common atlas access units* ([3.87](#)), including all subsequent *access units* ([3.41](#)) up to but not including any subsequent *coded common atlas access unit* ([3.41](#)) that is an *IRAP coded common atlas access unit* ([3.87](#))

**3.44****coded volumetric frame**

single *V3C composition unit* ([3.144](#)) pertaining to one particular time instance

**3.45****coded V3C sequence****CVS**

sequence of *V3C sub-bitstreams* ([3.146](#)) identified and separated by appropriate delimiters, required to start with a *VPS* ([3.147](#)), included in at least one *V3C unit* ([3.148](#)) or provided through external means, and contains one or more V3C units that can be factored into *V3C composition units* ([3.144](#)), where the first V3C composition unit is a *V3C IRAP composition unit* ([3.145](#))

**3.46****coded representation**

data element as represented in its coded form

**3.47****coded sub-bitstream sequence**

*sub-bitstream IRAP composition unit* ([3.123](#)) followed by zero or more *sub-bitstream composition units* ([3.122](#))

**3.48****coded V3C component**

coded representation of a *V3C component* ([3.142](#))

**3.49****common atlas frame****CAF**

common information (e.g. projection parameters) that applies to all *atlas frames* ([3.7](#)) in a *volumetric frame* ([3.151](#))

**3.50****common atlas sequence**

collection of *common atlas frames* ([3.49](#))

**3.51****common atlas sub-bitstream**

extracted *sub-bitstream* ([3.121](#)) from the *V3C bitstream* ([3.141](#)) containing a part of a common atlas NAL bitstream

**3.52****component bitstream**

*bitstream* ([3.27](#)) representing a *V3C component* ([3.142](#))

**3.53****component sub-bitstream**

portion of *component bitstream* ([3.49](#))

**3.54****composition time**

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

**3.55****composition time index**

index to an ordered list of *composition times* ([3.54](#))

**3.56****composition unit**

partition of a *bitstream* ([3.27](#)) that has a certain *composition time* ([3.54](#))

**3.57****decoder under test****DUT**

decoder that is tested for conformance to this document by operating the hypothetical stream scheduler to deliver a conforming *bitstream* ([3.27](#)) to the decoder and to the hypothetical reference decoder and comparing the values and timing or order of the output of the two decoders

**3.58****displacement**

set of 3D vectors that are added to the vertices of the subdivided mesh to closely approximate the input mesh surface

**3.59****displacement access unit**

set of data units of a *displacement bitstream* ([3.60](#)) that are associated with each other according to a specified classification rule, are consecutive in decoding order, and pertaining to one particular output time

**3.60****displacement bitstream**

*bitstream* ([3.27](#)) conforming to a mesh specification that may represent a *V3C component* ([3.142](#)) of type *displacement* ([3.58](#)) as specified by this document

**3.61****displacement data unit**

*syntax structure* ([3.133](#)) containing displacement data information, such as a *NAL unit* ([3.89](#)) in the context of Annex J of ISO/IEC 23090-29:2025

**3.62****displacement sub-bitstream**

extracted *sub-bitstream* ([3.121](#)) from the *V3C bitstream* ([3.141](#)) containing a part of a *displacement bitstream* ([3.60](#))

**3.63****decoding unit**

sub-set of a *coded atlas access unit* ([3.39](#)) consisting of one or more ACL NAL units in a *coded atlas access unit* ([3.39](#)) and the *associated non-ACL NAL units* ([3.91](#))

**3.64****enhanced occupancy mode****EOM**

patch coding mode where a *patch* ([3.99](#)) is associated with enhanced occupancy information

**3.65****EOM coded points**

coded representation of 3D points located at intermediate depth positions for which *geometry* ([3.71](#)) values are stored as codewords in the *occupancy frame* ([3.98](#)) and their corresponding attributes values are stored in additional patches, referred to as *EOM patches* ([3.66](#)), in the *attribute frames* ([3.16](#))

**3.66****EOM patch**

*patch* (3.99) with *patch mode* (3.101) equal to I\_EOM, P\_EOM, or P\_SKIP

**3.67****EOM patch type**

*patch type* (3.102) indicating an *EOM patch* (3.66)

**3.68****essential supplemental enhancement information****ESEI**

*SEI* (3.120) that is deemed as essential by the decoding process and should not be ignored or discarded

**3.69****essential supplemental enhancement information NAL unit****ESEI NAL unit**

*NAL unit* (3.89) corresponding to an *ESEI* (3.68) and has *nal\_unit\_type* equal to NAL\_PREFIX\_ESEI or NAL\_SUFFIX\_ESEI

**3.70****flag**

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

**3.71****geometry**

set of *Cartesian coordinates* (3.34) associated with a *volumetric frame* (3.141)

**3.72****geometry access unit**

collection of *geometry maps* (3.74) and auxiliary geometry frames, if present, corresponding to the same time instance

**3.73****geometry frame**

2D array created through the aggregation of the *geometry* (3.71) information associated with each *patch* (3.99)

**3.74****geometry map**

*geometry frame* (3.73) containing *geometry patch* (3.99) information projected at a particular depth

**3.75****global broken link access coded atlas frame****GBLA atlas frame**

*IRAP coded atlas* (3.85) frame for which each *ACL NAL unit* (3.5) has *nal\_unit\_type* equal to NAL\_GBLA\_W\_LP, NAL\_GBLA\_W\_RADL, or NAL\_GBLA\_N\_LP respectively

**3.76****global clean random access coded atlas frame****GCRA atlas frame**

*IRAP coded atlas* (3.85) frame for which each *ACL NAL unit* (3.5) has *nal\_unit\_type* equal to NAL\_GCRA

**3.77****global instantaneous decoding refresh coded atlas frame****GIDR atlas frame**

*coded atlas* (3.38) for which each *ACL NAL unit* (3.5) 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

**3.78****hypothetical reference decoder****HRD**

hypothetical decoder model that specifies constraints on the variability of conforming atlas *NAL unit* (3.89) streams or conforming *coded atlas* (3.38) sample streams that an encoding process may produce

**3.79****hypothetical stream scheduler****HSS**

hypothetical delivery mechanism used for checking the conformance of an *atlas sub-bitstream* (3.12) or a decoder with regards to the timing and data flow of the input of an *atlas sub-bitstream* (3.12) into the *hypothetical reference decoder* (3.78)

**3.80****instantaneous decoding refresh coded atlas access unit****IDR coded atlas access unit**

access unit in which the *coded atlas* (3.38) with *nal\_layer\_id* equal to 0 is an *IDR coded atlas* (3.81)

**3.81****instantaneous decoding refresh coded atlas frame****IDR coded atlas****IDR atlas frame**

*IRAP coded atlas* (3.85) for which each *ACL NAL unit* (3.5) 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.

**3.82****inter atlas tile**

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

**3.83****intra atlas tile**

atlas tile that is decoded using only intra prediction methods

**3.84****intra random access point coded atlas access unit****IRAP coded atlas access unit**

access unit in which the *coded atlas* (3.38) with *nal\_layer\_id* equal to 0 is an *IRAP coded atlas* (3.85)

**3.85****intra random access point coded atlas frame****IRAP coded atlas****IRAP coded atlas frame**

*coded atlas* (3.38) for which each *ACL NAL unit* (3.5) 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.

**3.86****intra random access point coded common atlas****IRAP coded common atlas**

*coded common atlas frame (3.42) for which at least one non-ACL NAL unit (3.91) has nal\_unit\_type equal to NAL\_CAF\_IDR*

**3.87****intra random access point coded common atlas access unit****IRAP coded common atlas access unit**

*coded common access unit (3.41) in which the coded common atlas frame (3.42) has nal\_layer\_id equal to 0 is and and nal\_unit\_type equal to NAL\_CAF\_IDR*

**3.88****multi-component collection of V3C sub-bitstreams**

*V3C sub-bitstreams (3.146) of multiple V3C components (3.142) that, when decoded, enable the reconstruction of volumetric content*

**3.89****network abstraction layer unit****NAL unit**

*syntax structure containing an indication of the type of data to follow and bytes (3.28) containing that data in the form of an RBSP (3.114)*

**3.90****network abstraction layer unit stream****NAL unit stream**

*sequence of NAL units (3.89)*

**3.91****non-ACL NAL unit**

*NAL unit (3.89) that is not an ACL NAL unit (3.5)*

**3.92****non-auxiliary video component****regular video component**

*video component (3.138) indicated as being of non-auxiliary type through an appropriate flag in the VPS (3.147)*

**3.93****non-auxiliary video sub-bitstream****regular video sub-bitstream**

*video sub-bitstream (3.140) indicated as being of non-auxiliary type through an appropriate flag in the VPS (3.147)*

**3.94****non-essential supplemental enhancement information****NSEI**

*SEI (3.130) that is deemed as non essential by the decoding process and may be ignored or discarded without any adverse effects*

**3.95****non-essential supplemental enhancement information NAL unit****NSEI NAL unit**

*NAL unit (3.89) corresponding to a NSEI (3.94) and has nal\_unit\_type equal to NAL\_PREFIX\_NSEI or NAL\_SUFFIX\_NSEI*

**3.96****occupancy**

*values that indicate whether atlas samples (3.9) correspond to associated samples in 3D space*

**3.97****occupancy access unit**

set of video data units of an occupancy bitstream that are associated with each other according to a specified classification rule, are consecutive in decoding order, and pertaining to one particular output time

**3.98****occupancy frame**

collection of *occupancy* (3.96) values that constitute a 2D array and represents the entire *occupancy* (3.96) information of a single *atlas frame* (3.7)

**3.99****patch**

rectangular region within an *atlas* (3.3) associated with volumetric information

**3.100****patch data**

data in an *atlas* (3.3) associated with a *patch* (3.99) that enables the conversion of 2D projected data back to 3D space

**3.101****patch mode**

*syntax element* (3.132) in the *atlas bitstream* (3.4) that indicates how a *patch* (3.99) is defined and associated with other *V3C components* (3.142) and provides information of how to reconstruct such *V3C components* (3.142)

**3.102****patch type**

classification of *patch modes* (3.101) based on how the characteristics of a *patch* (3.99) are signalled and interpreted

**3.103****point local reconstruction**

process of adding extra points in the reconstruction step that generates samples in 3D space associated to an *atlas sample* (3.9) from a *PROJECTED patch* (3.108), using information associated to neighboring atlas samples in the patch

**3.104****prefix ESEI NAL unit**

*essential SEI NAL unit* (3.69) that has *nal\_unit\_type* equal to NAL\_PREFIX\_ESEI

**3.105****prefix NSEI NAL unit**

*non-essential SEI NAL unit* (3.95) that has *nal\_unit\_type* equal to NAL\_PREFIX\_NSEI

**3.106****prefix SEI message**

*SEI* (3.130) message that is contained in a *prefix NSEI NAL unit* (3.105) or *prefix ESEI NAL unit* (3.104)

**3.107****prefix SEI NAL unit**

*SEI NAL unit* (3.131) that has *nal\_unit\_type* equal to NAL\_PREFIX\_NSEI or NAL\_PREFIX\_ESEI

**3.108****PROJECTED patch**

*patch* (3.99) with *patch mode* (3.101) equal to I\_INTRA, P\_INTRA, P\_INTER, P\_MERGE, or P\_SKIP that is associated with projected information onto a 2D image

**3.109****PROJECTED patch type**

*patch type* (3.102) indicating a *PROJECTED patch* (3.108)

**3.110****random access**

act of starting the decoding process for a *bitstream* (3.27) at a point other than the beginning of the stream

**3.111****random access decodable leading coded atlas frame****RADL coded atlas****RADL atlas frame**

*coded atlas* (3.38) for which each *ACL NAL unit* (3.5) 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.

**3.112****random access skipped leading access unit****RASL access unit**

access unit in which the *coded atlas* (3.38) with *nal\_layer\_id* equal to 0 is a *RASL coded atlas* (3.113)

**3.113****random access skipped leading coded atlas frame****RASL coded atlas****RASL atlas frame**

*coded atlas* (3.38) for which each *ACL NAL unit* (3.5) 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.

**3.114****raw byte sequence payload****RBSP**

*syntax structure* (3.133) containing an integer number of *bytes* (3.28) that is either empty or has the form of a *string of data bits* (3.120) containing *syntax elements* (3.132) followed by a bit equal to 1 and zero or more subsequent bits equal to 0

**3.115****RAW coded point**

*coded representation* (3.46) of a 3D point for which its *geometry* (3.71) value is directly stored in the corresponding region of a *RAW patch* (3.116) in either the first component of the first *geometry map* (3.74) or the *auxiliary geometry frame* (3.19), and for which its attribute values, if present, are directly stored in the corresponding region of a *RAW patch* (3.116) in either first *attribute map* (3.17) or the *auxiliary attribute frame* (3.18), without any form of projection

**3.116****RAW patch**

*patch* (3.99) with *patch mode* (3.101) equal to I\_RAW, P\_RAW, or P\_SKIP

**3.117****RAW patch type**

*patch type* (3.102) indicating a *RAW patch* (3.116)

**3.118****reserved**

term that may be used to specify that some values of a particular *syntax element* (3.132) are for future use by ISO/IEC and shall not be used in *bitstreams* (3.27) or atlas *bitstreams* (3.4) conforming to this version of this document, but may be used in *bitstreams* (3.27) or atlas *bitstreams* (3.4) conforming to future extensions of this document

**3.119****step-wise temporal sub-layer access coded atlas frame****STSA atlas frame**

*coded atlas* (3.38) for which each *ACL NAL unit* (3.5) has *nal\_unit\_type* equal to *NAL\_STSA\_R* or *NAL\_STSA\_N*

**3.120****string of data bits****SODB**

sequence of some number of bits representing *syntax elements* (3.132), represented as a sequence of bytes, where in of each byte 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

**3.121****sub-bitstream**

portion of a *bitstream* (3.27)

**3.122****sub-bitstream composition unit**

partition of a *sub-bitstream* (3.121) that has a certain composition time

**3.123****sub-bitstream IRAP composition unit**

*sub-bitstream composition unit* (3.122) that forms an independent random access point for the *sub-bitstream* (3.121)

**3.124****sub-layer**

temporal scalable layer of a temporal scalable *atlas sub-bitstream* (3.12), consisting of *ACL NAL units* (3.5) with a particular value of the *TemporalId* variable and the associated *non-ACL NAL units* (3.91)

**3.125****sub-layer non-reference coded atlas frame****SLNR coded atlas****SLNR atlas frame**

*atlas frames* (3.7) that contains *patches* (3.99) that cannot be used for inter prediction in the decoding process of subsequent *atlas frames* (3.7) of the same sub-layer in decoding order

**3.126****suffix ESEI NAL unit**

*ESEI NAL* (3.69) unit that has *nal\_unit\_type* equal to *NAL\_SUFFIX\_ESEI*

**3.127****suffix NSEI NAL unit**

*NSEI NAL unit* (3.95) that has *nal\_unit\_type* equal to *NAL\_SUFFIX\_NSEI*

**3.128****suffix SEI message**

*SEI* (3.130) message that is contained in either a *suffix NSEI NAL unit* (3.127) or a *suffix ESEI NAL unit* (3.126)

**3.129****suffix SEI NAL unit**

*SEI NAL unit* (3.131) that has *nal\_unit\_type* equal to *NAL\_SUFFIX\_NSEI* or *NAL\_SUFFIX\_ESEI*

**3.130****supplemental enhancement information****SEI**

information that may be inserted into the *bitstream* (3.27) for various purposes such as indicating *bitstream* (3.27) properties or for specifying various constraints that may apply to the *bitstream* (3.27)

**3.131****supplemental enhancement information NAL unit****SEI NAL unit**

*NAL unit* (3.89), intended for providing *supplemental enhancement information* (3.130) that has *nal\_unit\_type* equal to NAL\_PREFIX\_NSEI, NAL\_PREFIX\_ESEI, NAL\_SUFFIX\_NSEI, or NAL\_SUFFIX\_ESEI

**3.132****syntax element**

element of data represented in the *bitstream* (3.27)

**3.133****syntax structure**

zero or more *syntax elements* (3.122) present together in the *bitstream* (3.27) in a specified order

**3.134****temporal sub-layer access coded atlas frame****TSA coded atlas****TSA atlas frame**

*coded atlas* (3.38) for which each *ACL NAL unit* (3.5) has *nal\_unit\_type* equal to NAL\_TSA\_R or NAL\_TSA\_N

**3.135****tile**

independently decodable rectangular region of an *atlas frame* (3.7)

**3.136****unspecified**

term that may be used to specify some values of a particular *syntax element* (3.132) to indicate that the values have no specified meaning in this document and will not have a specified meaning in the future as an integral part of future versions of this document

**3.137****video bitstream**

*bitstream* (3.27) conforming to a video specification that may represent a *V3C component* (3.142) as specified by this document

**3.138****video component**

*V3C component* (3.142) of type *occupancy* (3.96), *geometry* (3.71), or *attribute* (3.14)

**3.139****video data unit**

*syntax structure* (3.133) containing video data information, such as a *NAL unit* (3.89) in the context of ISO/IEC 23008-2 or ISO/IEC 14496-10

**3.140****video sub-bitstream**

extracted *sub-bitstream* (3.121) from the *V3C bitstream* (3.141) containing a part of a *video bitstream* (3.137)

**3.141****visual volumetric video-based coding bitstream****V3C bitstream**

sequence of bits that forms the representation of *coded volumetric frames* (3.41) and associated data forming one or more *CVSs* (3.45)

**3.142****visual volumetric video-based coding component****V3C component**

*atlas* (3.3), *basemesh* (3.22), *occupancy* (3.96), *geometry* (3.71), or *attribute* (3.14) of a particular type that is associated with a V3C volumetric content representation

**3.143****visual volumetric video-based coding component frame****V3C component frame**

single *V3C component* ([3.142](#)) pertaining to one particular output time

**3.144****visual volumetric video-based coding composition unit****V3C composition unit**

set of all *sub-bitstream composition units* ([3.122](#)) that share the same *composition time* ([3.54](#)), where one of the *sub-bitstream composition units* ([3.122](#)) is a *coded atlas access unit* ([3.39](#))

**3.145****visual volumetric video-based coding IRAP composition unit****V3C IRAP composition unit**

*V3C composition unit* ([3.144](#)) for which all *sub-bitstream composition units* ([3.122](#)) are *sub-bitstream IRAP composition units* ([3.123](#)), where one of the *sub-bitstream composition units* ([3.122](#)) is a *coded atlas access unit* ([3.39](#)) for which each *ACL NAL unit* ([3.5](#)) 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

**3.146****visual volumetric video-based coding sub-bitstream****V3C sub-bitstream**

*sub-bitstream* ([3.121](#)) of a *V3C bitstream* ([3.141](#)) corresponding to a *V3C component* ([3.142](#))

**3.147****visual volumetric video-based coding parameter set****V3C parameter set****VPS**

*syntax structure* ([3.133](#)) containing *syntax elements* ([3.132](#)) that may be referred to by *syntax elements* ([3.132](#)) found in the *V3C unit header* ([3.149](#))

**3.148****visual volumetric video-based coding unit****V3C unit**

*syntax structure* ([3.133](#)) containing a *V3C unit header* ([3.149](#)) and a *V3C unit payload* ([3.150](#))

**3.149****visual volumetric video-based coding unit header****V3C unit header**

*syntax structure* ([3.133](#)) containing an indication of the type of data to follow

**3.150****visual volumetric video-based coding unit payload****V3C unit payload**

*syntax structure* ([3.133](#)) containing the *bytes* ([3.28](#)) containing *V3C sub-bitstream* ([3.146](#)) data

**3.151****volumetric frame**

set of 3D points specified by their *Cartesian coordinates* ([3.34](#)) and zero or more corresponding sets of *attributes* ([3.14](#)) at a particular time instance

**3.152****point**

data specified by single set of *Cartesian coordinates* ([3.34](#)) and zero or more corresponding *attributes* ([3.14](#))

**3.153****point cloud****point cloud frame**

set of *points* ([3.152](#)) at a particular time instance

**3.154****point cloud sequence**

sequence of *point cloud frames* ([3.153](#)) at a particular time instance

**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      |
| ASN.1 | abstract syntax notation one   |
| 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 lList          |
| 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

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 $x \geq 0$ and $y > 0$ .                                                            |

### 5.3 Logical operators

|                 |                                                                                                      |
|-----------------|------------------------------------------------------------------------------------------------------|
| $x \&\& y$      | Boolean logical "and" of x and y.                                                                    |
| $x \parallel y$ | 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".<br>When operating on integer arguments, operates on a two's complement representation of the integer value.                                                                                                                                                                           |
| &      | bit-wise "and".<br>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".<br>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".<br>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.<br>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.<br>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

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

$$\text{Ceil}(x) \text{ the smallest integer greater than or equal to } x. \quad (2)$$

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

$$\text{Floor}(x) \text{ the largest integer less than or equal to } x. \quad (4)$$

$$\text{Log2}(x) \text{ the base-2 logarithm of } x. \quad (5)$$

$$\text{Max}(x, y) = \begin{cases} x; x \geq y \\ y; x < y \end{cases} \quad (6)$$

$$\text{Mean}(A) \text{ the arithmetic average value of the input array } A. \quad (7)$$

$$\text{Median}(A) \text{ the median value of the input array } A. \quad (8)$$

$$\text{Min}(x, y) = \begin{cases} x; x \leq y \\ y; x > y \end{cases} \quad (9)$$

$$\text{Round}(x) = \text{Sign}(x) * \text{Floor}(\text{Abs}(x) + 0.5) \quad (10)$$

$$\text{Sign}(x) = \begin{cases} 1 & ; x > 0 \\ 0 & ; x = 0 \\ -1 & ; x < 0 \end{cases} \quad (11)$$

$$\text{Sqrt}(x) \text{ the square root of } x. \quad (12)$$

## 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  $\{ \{ \dots \} \{ \dots \} \}$ , 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 "x" 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 ( $\text{Dim}_0 - 1$ ), inclusive,  $y$  in the range of 0 to ( $\text{Dim}_1 - 1$ ), inclusive, and  $x$  in the range of 0 to ( $\text{Dim}_2 - 1$ ), inclusive, can also be referred to as being of size  $\text{Dim}_0 \times \text{Dim}_1 \times \text{Dim}_2$ .

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:

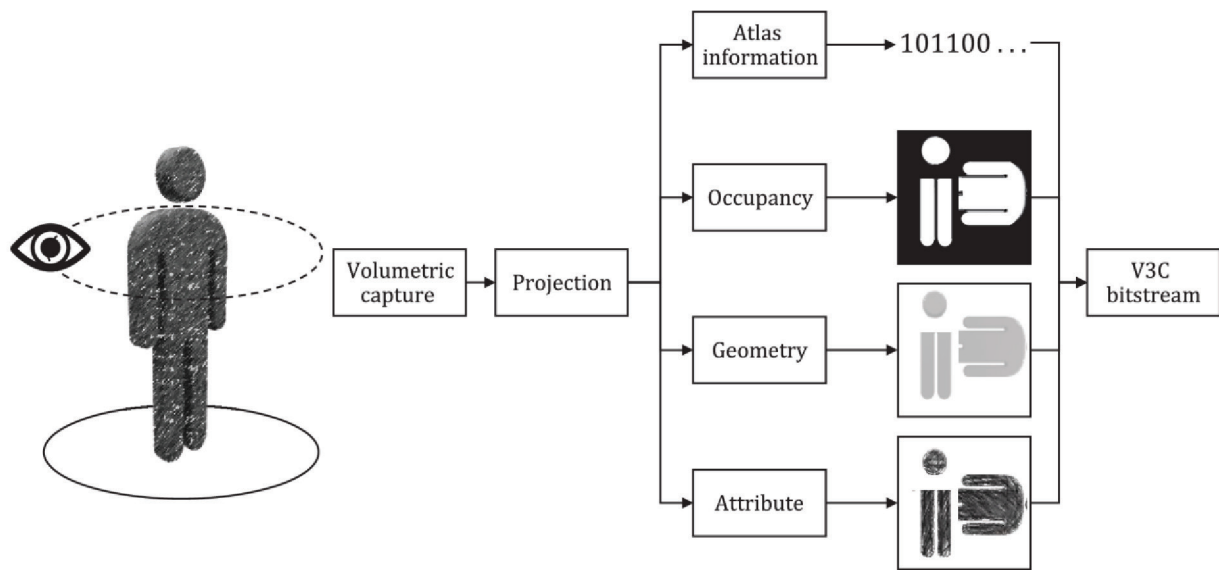
- If the variables at the invoking and the process specification do not have the same name, the variables are explicitly assigned to lower case input or output variables of the process specification.
- Otherwise (the variables at the invoking and the process specification have the same name), assignment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

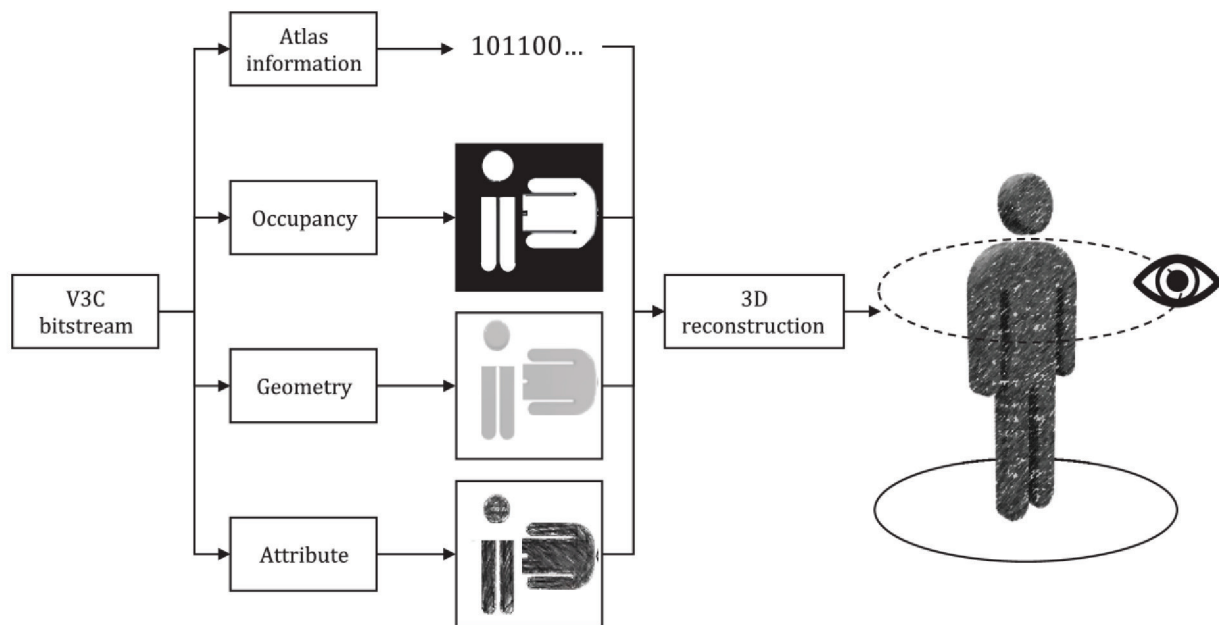
## 6 Overall V3C characteristics, decoding operations, and post-decoding processes

### 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 version of this document, the occupancy, geometry, and attribute components. The occupancy component can inform a V3C decoding and/or rendering system 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](#) where 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.



a) Encoder



b) Decoder and reconstruction

Figure 1 — Volumetric media conversion

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.

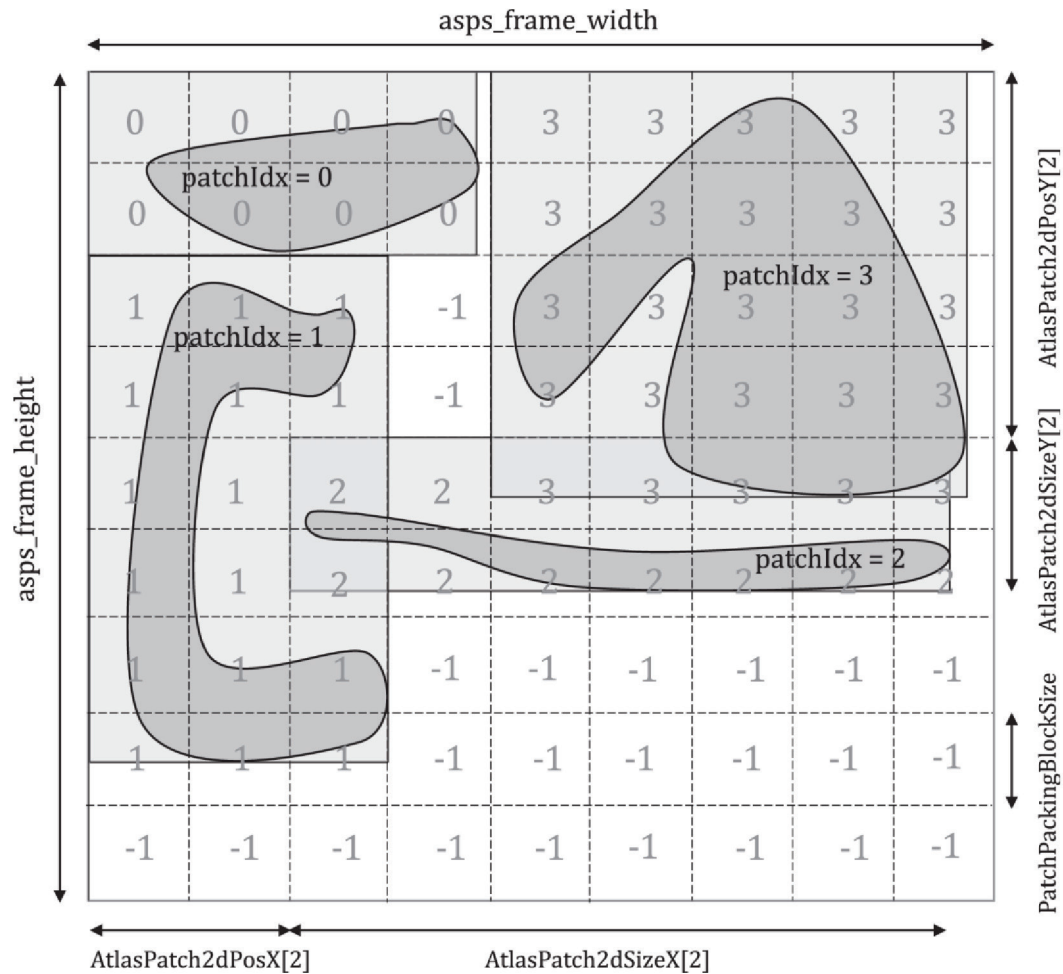
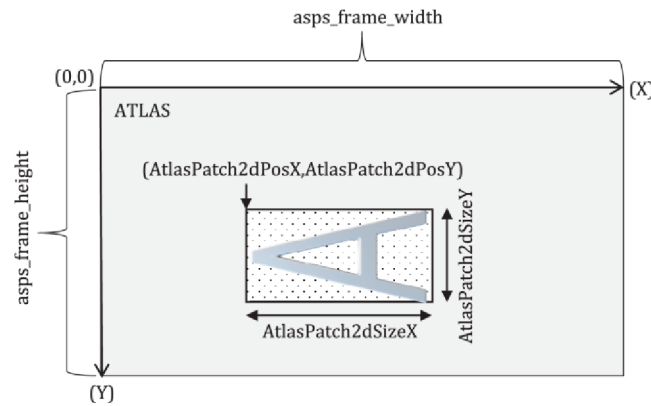


**Figure 2 — An example of block to patch mapping**

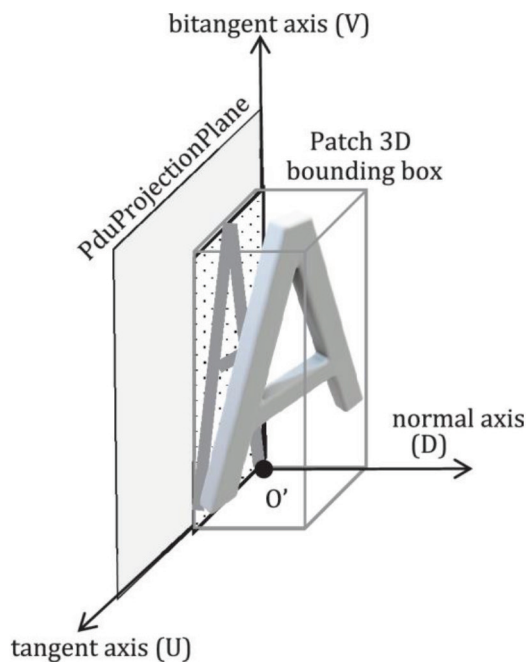
In this document, axes orientations are specified for internal operations. For instance, the origin of the atlas coordinates is located on the top-left corner of the atlas frame. For the reconstruction step, an intermediate axes definition for a local 3D patch coordinate system is used. The 3D local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

**NOTE** 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.

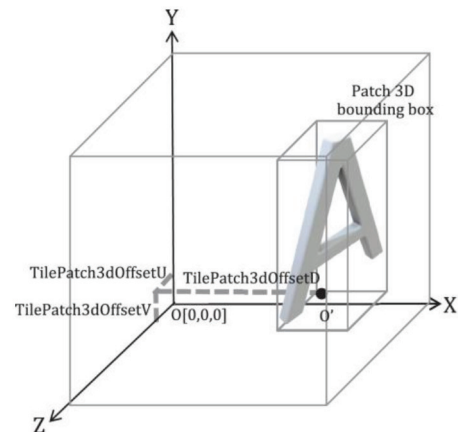
[Figure 3](#) a) shows an example of a single patch packed onto an atlas image. This patch is then converted to a local 3D patch coordinate system (U, V, D) defined by the projection plane with origin O', tangent (U), bi-tangent (V), and normal (D) axes. For an orthographic projection, the projection 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).



a) Example of an atlas coordinate system



b) Example of a local 3D patch coordinate system



c) Example of a final target 3D coordinate system

Figure 3 — Example of an atlas coordinate system, a local 3D patch coordinate system, and 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](#).

### 6.3 Extensibility of V3C: V-PCC, MIV and V-DMC

This document is designed to be extensible and thereby applicable to multiple video-based volumetric video representations. In this version of this document, extensions that provide one or more toolsets are Video-based point cloud coding (V-PCC), specified in [Annex H](#), MPEG immersive video (MIV), specified in ISO/IEC 23090-12, and Video-based dynamic mesh coding (V-DMC), specified in ISO/IEC 23090-29. Not all clauses and annexes apply to all extensions. [Table 2](#) provides a summary to guide the reader.

**Table 2 — Applicability of clauses and annexes**

| Clause / annex                   | <a href="#">Annex H</a> | ISO/IEC 23090-12 | ISO/IEC 23090-29 |
|----------------------------------|-------------------------|------------------|------------------|
| <a href="#">1 .. 9</a>           | Yes                     | Yes              | Yes              |
| <a href="#">10</a>               | Yes                     | No               | No               |
| <a href="#">11</a>               | Yes                     | Yes              | Yes              |
| <a href="#">12, 13</a>           | Yes                     | No               | No               |
| <a href="#">14, Annex A</a>      | Yes                     | Yes              | Yes              |
| <a href="#">Annex B</a>          | Yes                     | Partially        | Yes              |
| <a href="#">Annex C..Annex G</a> | Yes                     | Yes              | Yes              |
| <a href="#">Annex H</a>          | -                       | No               | No               |

## 7 Bitstream format, partitioning, and scanning processes

### 7.1 General

Coded V3C video components are referred to in this document as video bitstreams, while a basemesh component is referred to as the basemesh bitstream and an atlas component is referred to as the atlas bitstream. Video bitstreams, basemesh bitstreams, and atlas bitstreams may be further split into smaller units, referred to here as video, basemesh 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, NumPartitionRows and the width and height of each tile partition, respectively, is indicated in the atlas frame tile information syntax ([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 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.

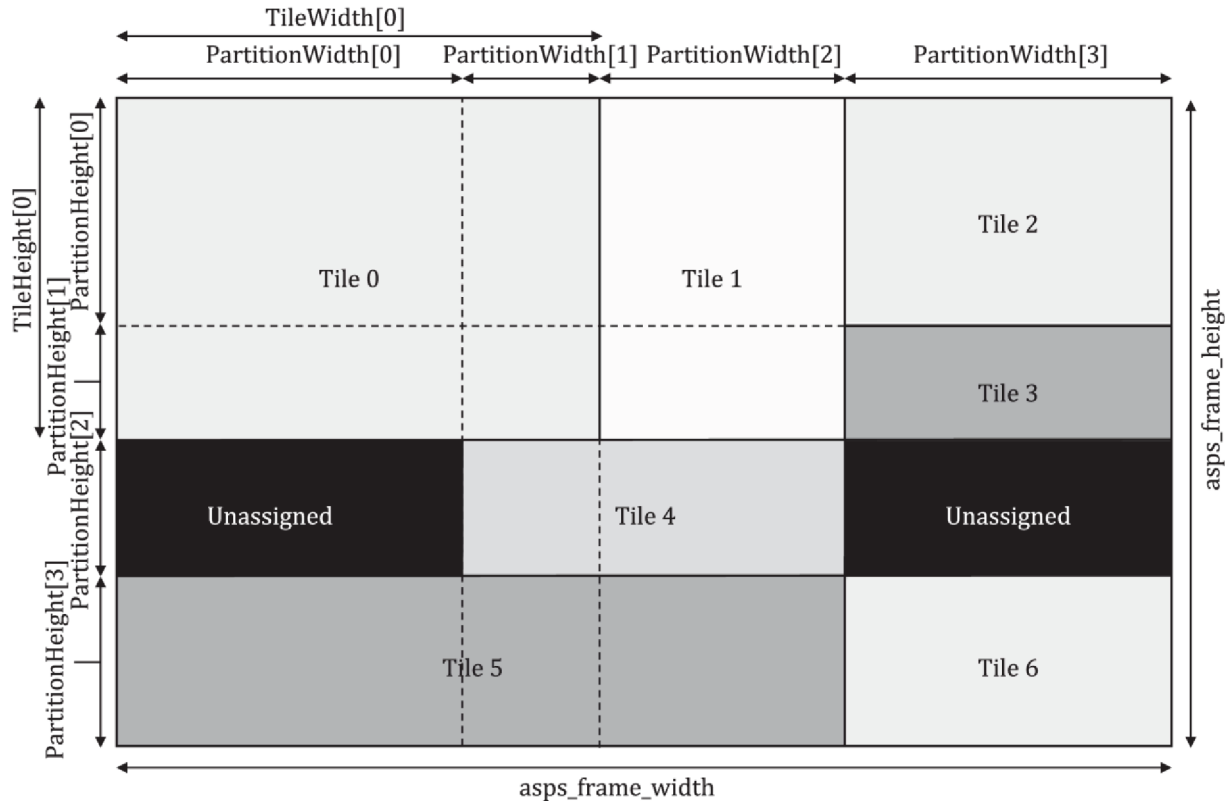


Figure 4 — An example of an atlas frame partitioned into 7 tiles

## 8 Syntax and semantics

### 8.1 Method of specifying syntax in tabular form

#### 8.1.1 General

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 can 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.

See 8.1.2 for an example 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.

## 8.1.2 Example of the syntax specification format

|                                                                                                                                                                                                                                                                                                                                                                 | 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 */                                                                                                                                           |            |
| <b>syntax_element</b>                                                                                                                                                                                                                                                                                                                                           | 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.

`length_aligned( )` is specified as follows:

- If the current position in the extension is on a byte boundary from the start of the extension, i.e. the next bit in the extension is the first bit in a byte, the return value of `length_aligned( )` is equal to TRUE.
- Otherwise, the return value of `length_aligned( )` is equal to FALSE.

`more_data_in_payload( payloadSize )` is specified as follows:

- If `byte_aligned( )` is equal to TRUE and the current position in the `sei_payload( )` syntax structure is  $8 * \text{payloadSize}$  bits from the beginning of the `sei_payload( )` syntax structure, the return value of `more_data_in_payload( payloadSize )` is equal to FALSE.
- Otherwise, the return value of `more_data_in_payload( payloadSize )` 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 (`rbps_stop_one_bit`) of the `rbps_trailing_bits( )` syntax structure, the following applies:
- If there is more data in an Rbsp before the `rbps_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 the position of the last (least significant, right-most) bit that is equal to 1 that is less than  $8 * \text{payloadSize}$  bits from the beginning of the syntax structure (i.e. the position of the `alignment_bit_equal_to_one` syntax element), the return value of `payload_extension_present( )` is equal to FALSE.
- Otherwise, the return value of `payload_extension_present( )` is equal to TRUE.

`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 specified in ISO/IEC 60559:2020.
- 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:

```
sign_extend( value, n ) {
    sign_bit = 1 << (n - 1)
    if( value < sign_bit )
        return value
    else
        return value | ~(sign_bit - 1)
}

i(n) {
    value = read_bits( n )
    return sign_extend( value, n )
}
```

- se(v): signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in [subclause 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  $(\text{stringLength} + 1) * 8$  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](#).
- oid(v): an ASN.1 object identifier as specified in ISO/IEC 8825-1. The parsing process for this descriptor is specified in [subclause 14.3](#).

## 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 3](#) 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.3.7](#).

The basemesh and displacement data unit payloads correspond to basemesh data units (e.g. NAL units defined in Annex H of ISO/IEC 23090-29) and AC displacement data units (e.g. NAL units defined in Annex J of ISO/IEC 23090-29), respectively, that could be decoded by an appropriate decoder as indicated by the profiles and, if present, the component codec mapping SEI message defined in ISO/IEC 23090-29.

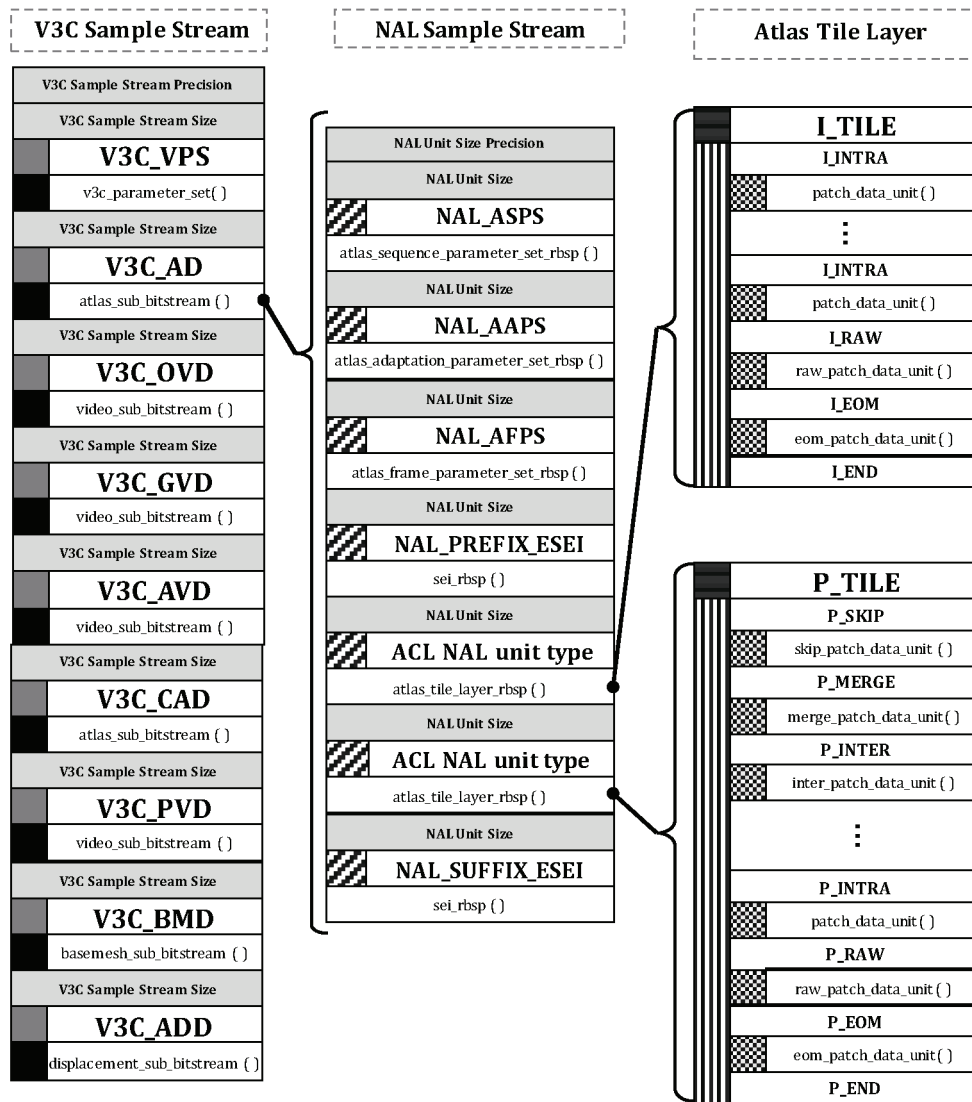


Figure 5 — Overview of V3C bitstream structure

### 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 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| <b>vuh_unit_type</b>                                                                                                                                                                                                                       | u(5)       |
| if( vuh_unit_type == V3C_AVD    vuh_unit_type == V3C_GVD   <br>vuh_unit_type == V3C_OVD    vuh_unit_type == V3C_AD   <br>vuh_unit_type == V3C_CAD    vuh_unit_type == V3C_PVD   <br>vuh_unit_type == V3C_BMD    vuh_unit_type == V3C_ADD ) |            |
| <b>vuh_v3c_parameter_set_id</b>                                                                                                                                                                                                            | u(4)       |
| if( vuh_unit_type == V3C_AVD    vuh_unit_type == V3C_GVD   <br>vuh_unit_type == V3C_OVD    vuh_unit_type == V3C_AD   <br>vuh_unit_type == V3C_PVD    vuh_unit_type == V3C_BMD   <br>vuh_unit_type == V3C_ADD )                             |            |
| <b>vuh_atlas_id</b>                                                                                                                                                                                                                        | u(6)       |
| if( vuh_unit_type == V3C_AVD ) {                                                                                                                                                                                                           |            |
| <b>vuh_attribute_index</b>                                                                                                                                                                                                                 | u(7)       |
| <b>vuh_attribute_partition_index</b>                                                                                                                                                                                                       | u(5)       |
| <b>vuh_map_index</b>                                                                                                                                                                                                                       | u(4)       |
| <b>vuh_auxiliary_video_flag</b>                                                                                                                                                                                                            | u(1)       |
| } else if( vuh_unit_type == V3C_GVD ) {                                                                                                                                                                                                    |            |
| <b>vuh_map_index</b>                                                                                                                                                                                                                       | u(4)       |
| <b>vuh_auxiliary_video_flag</b>                                                                                                                                                                                                            | u(1)       |
| <b>vuh_reserved_zero_12bits</b>                                                                                                                                                                                                            | u(12)      |
| } else if( vuh_unit_type == V3C_OVD    vuh_unit_type == V3C_AD   <br>vuh_unit_type == V3C_PVD    vuh_unit_type == V3C_BMD   <br>vuh_unit_type == V3C_ADD )                                                                                 |            |
| <b>vuh_reserved_zero_17bits</b>                                                                                                                                                                                                            | u(17)      |
| else if( vuh_unit_type == V3C_CAD )                                                                                                                                                                                                        |            |
| <b>vuh_reserved_zero_23bits</b>                                                                                                                                                                                                            | u(23)      |
| else                                                                                                                                                                                                                                       |            |
| <b>vuh_reserved_zero_27bits</b>                                                                                                                                                                                                            | 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<br>   vuh_unit_type == V3C_CAD )                                                                |            |
| atlas_sub_bitstream( numBytesInV3CPayload )                                                                                      |            |
| else if( vuh_unit_type == V3C_OVD   <br>vuh_unit_type == V3C_GVD   <br>vuh_unit_type == V3C_AVD   <br>vuh_unit_type == V3C_PVD ) |            |
| video_sub_bitstream( numBytesInV3CPayload )                                                                                      |            |
| else if( vuh_unit_type == V3C_BMD )                                                                                              |            |

|                                                                                           |  |
|-------------------------------------------------------------------------------------------|--|
| basemesh_sub_bitstream( numBytesInV3CPayload )<br>/* Specified in ISO/IEC 23090-29 */     |  |
| else if( vuh_unit_type == V3C_ADD )                                                       |  |
| displacement_sub_bitstream( numBytesInV3CPayload )<br>/* Specified in ISO/IEC 23090-29 */ |  |
| }                                                                                         |  |

#### 8.3.2.4 Atlas sub-bitstream syntax

| atlas_sub_bitstream( numBytes ) {                                          | Descriptor |
|----------------------------------------------------------------------------|------------|
| 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 |
|-----------------------------------------------------|------------|
| <b>alignment_bit_equal_to_one</b> /* equal to 1 */  | f(1)       |
| while( !byte_aligned( ) )                           |            |
| <b>alignment_bit_equal_to_zero</b> /* 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( )                               |            |
| <b>vps_v3c_parameter_set_id</b>                     | u(4)       |
| <b>vps_reserved_zero_8bits</b>                      | u(8)       |
| <b>vps_atlas_count_minus1</b>                       | u(6)       |
| for( k = 0; k < vps_atlas_count_minus1 + 1; k++ ) { |            |
| <b>vps_atlas_id[ k ]</b>                            | u(6)       |
| j = vps_atlas_id[ k ]                               |            |
| <b>vps_frame_width[ j ]</b>                         | ue(v)      |
| <b>vps_frame_height[ j ]</b>                        | ue(v)      |
| <b>vps_map_count_minus1[ j ]</b>                    | u(4)       |
| if( vps_map_count_minus1[ j ] > 0 )                 |            |
| <b>vps_multiple_map_streams_present_flag[ j ]</b>   | 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 ] )    |            |

|                                                                     |       |
|---------------------------------------------------------------------|-------|
| <b>vps_map_absolute_coding_enabled_flag[ j ][ i ]</b>               | u(1)  |
| else                                                                |       |
| vps_map_absolute_coding_enabled_flag[ j ][ i ] = 1                  |       |
| if( vps_map_absolute_coding_enabled_flag[ j ][ i ] == 0 ) {         |       |
| <b>vps_map_predictor_index_diff[ j ][ i ]</b>                       | ue(v) |
| }                                                                   |       |
| }                                                                   |       |
| <b>vps_auxiliary_video_present_flag[ j ]</b>                        | u(1)  |
| <b>vps_occupancy_video_present_flag[ j ]</b>                        | u(1)  |
| <b>vps_geometry_video_present_flag[ j ]</b>                         | u(1)  |
| <b>vps_attribute_video_present_flag[ j ]</b>                        | 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 )                                          |       |
| }                                                                   |       |
| <b>vps_extension_present_flag</b>                                   | u(1)  |
| if( vps_extension_present_flag ) {                                  |       |
| <b>vps_extension_count</b>                                          | u(8)  |
| }                                                                   |       |
| if( vps_extension_count ) {                                         |       |
| <b>vps_extensions_length_minus1</b>                                 | ue(v) |
| for( i = 0; i < vps_extension_count; i++ ) {                        |       |
| <b>vps_extension_type[ i ]</b>                                      | u(8)  |
| <b>vps_extension_length[ i ]</b>                                    | u(16) |
| vps_extension( vps_extension_type[ i ], vps_extension_length[ i ] ) |       |
| }                                                                   |       |
| }                                                                   |       |
| byte_alignment( )                                                   |       |
| }                                                                   |       |

#### 8.3.4.2 Profile, tier, and level syntax

| profile_tier_level( ) {               | Descriptor |
|---------------------------------------|------------|
| <b>ptl_tier_flag</b>                  | u(1)       |
| <b>ptl_profile_codec_group_idc</b>    | u(7)       |
| <b>ptl_profile_toolset_idc</b>        | u(8)       |
| <b>ptl_profile_reconstruction_idc</b> | u(8)       |
| <b>ptl_reserved_zero_16bits</b>       | u(16)      |
| <b>ptl_max_decodes_idc</b>            | u(4)       |
| <b>ptl_reserved_0xffff_12bits</b>     | u(12)      |
| <b>ptl_level_idc</b>                  | u(8)       |
| <b>ptl_num_sub_profiles</b>           | u(6)       |

|                                             |      |
|---------------------------------------------|------|
| <b>ptl_extended_sub_profile_flag</b>        | u(1) |
| for( i = 0; i < ptl_num_sub_profiles; i++ ) |      |
| <b>ptl_sub_profile_idc[ i ]</b>             | u(v) |
| <b>ptl_toolset_constraints_present_flag</b> | u(1) |
| if( ptl_toolset_constraints_present_flag )  |      |
| profile_toolset_constraints_information( )  |      |
| }                                           |      |

#### 8.3.4.3 Occupancy information syntax

|                                                            |                   |
|------------------------------------------------------------|-------------------|
| occupancy_information( atlasID ) {                         | <b>Descriptor</b> |
| <b>oi_occupancy_codec_id[ atlasID ]</b>                    | u(8)              |
| <b>oi_lossy_occupancy_compression_threshold[ atlasID ]</b> | u(8)              |
| <b>oi_occupancy_2d_bit_depth_minus1[ atlasID ]</b>         | u(5)              |
| <b>oi_occupancy_msb_align_flag[ atlasID ]</b>              | u(1)              |
| }                                                          |                   |

#### 8.3.4.4 Geometry information syntax

|                                                               |                   |
|---------------------------------------------------------------|-------------------|
| geometry_information( atlasID ) {                             | <b>Descriptor</b> |
| <b>gi_geometry_codec_id[ atlasID ]</b>                        | u(8)              |
| <b>gi_geometry_2d_bit_depth_minus1[ atlasID ]</b>             | u(5)              |
| <b>gi_geometry_msb_align_flag[ atlasID ]</b>                  | u(1)              |
| <b>gi_geometry_3d_coordinates_bit_depth_minus1[ atlasID ]</b> | u(5)              |
| if( vps_auxiliary_video_present_flag[ atlasID ] )             |                   |
| <b>gi_auxiliary_geometry_codec_id[ atlasID ]</b>              | u(8)              |
| }                                                             |                   |

#### 8.3.4.5 Attribute information syntax

|                                                                          |                   |
|--------------------------------------------------------------------------|-------------------|
| attribute_information( atlasID ) {                                       | <b>Descriptor</b> |
| <b>ai_attribute_count[ atlasID ]</b>                                     | u(7)              |
| for( i = 0; i < ai_attribute_count[ atlasID ]; i++ ) {                   |                   |
| <b>ai_attribute_type_id[ atlasID ][ i ]</b>                              | u(4)              |
| <b>ai_attribute_codec_id[ atlasID ][ i ]</b>                             | u(8)              |
| if( vps_auxiliary_video_present_flag[ atlasID ] )                        |                   |
| <b>ai_auxiliary_attribute_codec_id[ atlasID ][ i ]</b>                   | u(8)              |
| if( vps_map_count_minus1[ atlasID ] > 0 )                                |                   |
| <b>ai_attribute_map_absolute_coding_persistence_flag[ atlasID ][ i ]</b> | u(1)              |
| d = <b>ai_attribute_dimension_minus1[ atlasID ][ i ]</b>                 | u(6)              |
| if( d == 0 ) {                                                           |                   |
| ai_attribute_dimension_partitions_minus1[ atlasID ][ i ] = 0             |                   |
| m = 0                                                                    |                   |
| } else                                                                   |                   |
| m = <b>ai_attribute_dimension_partitions_minus1[ atlasID ][ i ]</b>      | u(6)              |
| for( k = 0; k < m; k++ ) {                                               |                   |

|                                                                 |       |
|-----------------------------------------------------------------|-------|
| if( k + d == m ) {                                              |       |
| ai_attribute_partition_channels_minus1[ atlasID ][ i ][ k ] = 0 |       |
| n = 0                                                           |       |
| } else                                                          |       |
| n = ai_attribute_partition_channels_minus1[ atlasID ][ i ][ k ] | ue(v) |
| d -= n + 1                                                      |       |
| }                                                               |       |
| ai_attribute_partition_channels_minus1[ atlasID ][ i ][ m ] = d |       |
| 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_restricted_geometry_flag                             | u(1)       |
| ptc_reserved_zero_5bits                                  | u(5)       |
| ptc_num_reserved_constraint_bytes                        | u(8)       |
| for( i = 0; i < ptc_num_reserved_constraint_bytes; i++ ) |            |
| ptc_reserved_constraint_byte[ i ]                        | u(8)       |
| }                                                        |            |

#### 8.3.4.7 Packing information syntax

| packing_information( j ) {                     | Descriptor |
|------------------------------------------------|------------|
| pin_codec_id[ j ]                              | u(8)       |
| pin_occupancy_present_flag[ j ]                | u(1)       |
| pin_geometry_present_flag[ j ]                 | u(1)       |
| pin_attribute_present_flag[ j ]                | u(1)       |
| if( pin_occupancy_present_flag[ j ] ) {        |            |
| pin_occupancy_2d_bit_depth_minus1[ j ]         | u(5)       |
| pin_occupancy_msb_align_flag[ j ]              | u(1)       |
| pin_lossy_occupancy_compression_threshold[ j ] | u(8)       |
| }                                              |            |
| if( pin_geometry_present_flag[ j ] ) {         |            |

|                                                                                                                     |       |
|---------------------------------------------------------------------------------------------------------------------|-------|
| <b>pin_geometry_2d_bit_depth_minus1[ j ]</b>                                                                        | u(5)  |
| <b>pin_geometry_msb_align_flag[ j ]</b>                                                                             | u(1)  |
| <b>pin_geometry_3d_coordinates_bit_depth_minus1[ j ]</b>                                                            | u(5)  |
| }                                                                                                                   |       |
| if( pin_attribute_present_flag[ j ] ) {                                                                             |       |
| <b>pin_attribute_count[ j ]</b>                                                                                     | u(7)  |
| for( i = 0; i < <b>pin_attribute_count[ j ]</b> ; i++ ) {                                                           |       |
| <b>pin_attribute_type_id[ j ][ i ]</b>                                                                              | u(4)  |
| <b>pin_attribute_2d_bit_depth_minus1[ j ][ i ]</b>                                                                  | u(5)  |
| <b>pin_attribute_msb_align_flag[ j ][ i ]</b>                                                                       | u(1)  |
| <b>pin_attribute_map_absolute_coding_persistence_flag[ j ][ i ]</b>                                                 | u(1)  |
| d = <b>pin_attribute_dimension_minus1[ j ][ i ]</b>                                                                 | u(6)  |
| if( d == 0 ) {                                                                                                      |       |
| pin_attribute_dimension_partitions_minus1[ j ][ i ] = 0                                                             |       |
| m = 0                                                                                                               |       |
| } else                                                                                                              |       |
| m = <b>pin_attribute_dimension_partitions_minus1[ j ][ i ]</b>                                                      | u(6)  |
| for( k = 0; k < m; k++ ) {                                                                                          |       |
| if( k + d == m ) {                                                                                                  |       |
| pin_attribute_partition_channels_minus1[ j ][ i ][ k ] = 0                                                          |       |
| n = 0                                                                                                               |       |
| } else                                                                                                              |       |
| n = <b>pin_attribute_partition_channels_minus1[ j ][ i ][ k ]</b>                                                   | ue(v) |
| d -= n + 1                                                                                                          |       |
| }                                                                                                                   |       |
| pin_attribute_partition_channels_minus1[ j ][ i ][ m ] = d                                                          |       |
| }                                                                                                                   |       |
| }                                                                                                                   |       |
| <b>pin_regions_count_minus1[ j ]</b>                                                                                | ue(v) |
| for( i = 0; i <= pin_regions_count_minus1[ j ]; i++ ) {                                                             |       |
| <b>pin_region_tile_id[ j ][ i ]</b>                                                                                 | u(8)  |
| <b>pin_region_type_id_minus2[ j ][ i ]</b>                                                                          | u(2)  |
| <b>pin_region_top_left_x[ j ][ i ]</b>                                                                              | u(16) |
| <b>pin_region_top_left_y[ j ][ i ]</b>                                                                              | u(16) |
| <b>pin_region_width_minus1[ j ][ i ]</b>                                                                            | u(16) |
| <b>pin_region_height_minus1[ j ][ i ]</b>                                                                           | u(16) |
| <b>pin_region_unpack_top_left_x[ j ][ i ]</b>                                                                       | u(16) |
| <b>pin_region_unpack_top_left_y[ j ][ i ]</b>                                                                       | u(16) |
| <b>pin_region_rotation_flag[ j ][ i ]</b>                                                                           | u(1)  |
| if( pin_region_type_id_minus2[ j ][ i ] + 2 == V3C_AVD   <br>pin_region_type_id_minus2[ j ][ i ] + 2 == V3C_GVD ) { |       |
| <b>pin_region_map_index[ j ][ i ]</b>                                                                               | u(4)  |
| <b>pin_region_auxiliary_data_flag[ j ][ i ]</b>                                                                     | u(1)  |
| }                                                                                                                   |       |

|                                                            |      |
|------------------------------------------------------------|------|
| if( pin_region_type_id_minus2[ j ][ i ] + 2 == V3C_AVD ) { |      |
| <b>pin_region_attr_index</b> [ j ][ i ]                    | u(7) |
| k = pin_region_attr_index[ j ][ i ]                        |      |
| if( pin_attribute_dimension_minus1[ j ][ k ] > 0 )         |      |
| <b>pin_region_attr_partition_index</b> [ j ][ i ]          | u(5) |
| }                                                          |      |
| }                                                          |      |
| }                                                          |      |

#### 8.3.4.8 VPS extension syntax

| vps_extension( extension_type, extension_length ) {        | Descriptor |
|------------------------------------------------------------|------------|
| if( extension_type == VPS_EXT_PACKED )                     |            |
| vps_packed_video_extension( )                              |            |
| else if( extension_type == VPS_EXT_MIV )                   |            |
| vps_miv_extension( ) /* Specified in ISO/IEC 23090-12 */   |            |
| else if( extension_type == VPS_EXT_MIV_2 )                 |            |
| vps_miv_2_extension( ) /* Specified in ISO/IEC 23090-12 */ |            |
| else if( extension_type == VPS_EXT_VDMC )                  |            |
| vps_vdmc_extension( ) /* Specified in ISO/IEC 23090-29 */  |            |
| else {                                                     |            |
| for( j = 0; j < extension_length; j++ )                    |            |
| <b>vps_extension_data_byte</b>                             | u(8)       |
| }                                                          |            |
| length_alignment( )                                        |            |
| }                                                          |            |

#### 8.3.4.9 Packed video extension syntax

| vps_packed_video_extension( ) {                  | Descriptor |
|--------------------------------------------------|------------|
| for( k = 0; k <= vps_atlas_count_minus1; k++ ) { |            |
| j = vps_atlas_id[ k ]                            |            |
| <b>vps_packed_video_present_flag</b> [ j ]       | u(1)       |
| if( vps_packed_video_present_flag[ j ] )         |            |
| packing_information( j )                         |            |
| }                                                |            |
| }                                                |            |

#### 8.3.4.10 Length alignment syntax

| length_alignment( ) {                                      | Descriptor |
|------------------------------------------------------------|------------|
| while( !length_aligned( ) )                                |            |
| <b>length_alignment_bit_equal_to_zero</b> /* equal to 0 */ | f(1)       |
| }                                                          |            |

### 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)       |