



DPC: 25/30458568 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: 14 March 2025

Origin: International

Latest date for receipt of comments: 07 May 2025

Project No. 2022/01561

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

Interested committees:

Title: Draft BS ISO/IEC 23090-30 Information technology. Coded representation of immersive media. Part 30: Low latency, low complexity LiDAR coding.

Please notify 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 07 May 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. 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/>.

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 23090-30

Information technology — Coded representation of immersive media —

Part 30: Low latency, low complexity LiDAR coding

ICS: 35.040.40

ISO/IEC JTC 1/SC 29

Secretariat: JISC

Voting begins on:
2025-03-14

Voting terminates on:
2025-06-06

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 23090-30:2025(en)

Full document details: 25/30458568 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
Website: www.iso.org

Published in Switzerland

Contents

Foreword.....	viii
Introduction.....	ix
1 Scope.....	1
2 Normative references.....	1
3 Terms and definitions	1
3.1 General terms.....	1
3.2 High-level syntax and entropy coding terms	4
3.3 Tree structure terms	7
3.4 Geometry coding terms	7
3.5 Attribute coding terms.....	7
4 Abbreviated terms.....	8
4.1 Acronyms	8
4.2 Mnemonics	8
5 Conventions	9
5.1 General.....	9
5.2 Symbolic names	9
5.3 Numerical representation	9
5.4 Arithmetic operators.....	9
5.5 Logical operators	10
5.6 Relational operators.....	10
5.7 Bit-wise operators	10
5.8 Assignment operators	10
5.9 Range notation	11
5.10 Mathematical functions.....	11
5.10.1 General	11
5.10.2 IntCos and IntSin.....	12
5.10.3 IntSqrt.....	12
5.10.4 IntRecipSqrt.....	12
5.10.5 Div	13
5.10.6 Morton	14
5.10.7 FromMorton	14
5.11 Order of operation precedence	15
5.12 Named expressions.....	15
5.12.1 General	15
5.12.2 Scope	16
5.12.3 Arguments of named expressions	16
5.12.4 Sub-expressions	16
5.12.5 Definitions with multiple statements	17
5.12.6 Textual definitions	17
5.13 Variables, syntax elements and tables.....	17
6 Point cloud format and relationship to coded and output representations	17
6.1 General format.....	17
6.2 Attributes.....	18
6.2.1 General	18
6.2.2 Colour.....	18
6.2.3 Opacity.....	19
6.2.4 Reflectance.....	19

6.2.5	Normal vector.....	19
6.2.6	Material identifier.....	19
6.2.7	Frame number/index	19
6.2.8	User defined attributes	20
6.3	Codec-derived attributes	20
6.3.1	General.....	20
6.3.2	Slice identifier	20
6.3.3	Slice tag	20
6.3.4	Canonical point order	20
6.3.5	Point Morton order.....	20
6.4	Coded point cloud format.....	21
6.4.1	Sequence coordinate system.....	21
6.4.2	Coding coordinate system.....	22
6.4.3	Coded point cloud sequence.....	22
6.4.4	Coded point cloud frame	22
6.4.5	Slice of a coded point cloud frame	23
6.4.6	Repetition of slices.....	23
6.4.7	Relationship between tiles and slices.....	24
6.4.8	Parameter sets	24
6.5	Output point cloud format	25
6.5.1	General.....	25
6.5.2	Coordinate system	25
6.5.3	Fixed-point conformance output.....	25
6.5.4	Attributes	25
6.5.5	Output point cloud sequence	25
6.5.6	Output point cloud frame	26
7	Syntax and semantics.....	26
7.1	Method of specifying syntax in tabular form.....	26
7.2	Specification of syntax functions and descriptors.....	27
7.3	Syntax in tabular form	28
7.3.1	General.....	28
7.3.2	Parameter sets, ancillary data and byte alignment.....	28
7.3.3	Geometry data unit.....	34
7.3.4	Attribute data unit	38
7.3.5	Defaulted attribute data unit syntax.....	40
7.4	Semantics	40
7.4.1	General.....	40
7.4.2	Parameter sets, ancillary data and byte alignment.....	41
7.4.3	Geometry data unit.....	52
7.4.4	Attribute data unit	55
7.4.5	Defaulted attribute data unit semantics.....	55
8	Decoding process.....	55
8.1	General decoding process	55
8.2	Frame decoding processes	55
8.2.1	General.....	55
8.2.2	Frame counter	56
8.3	Slice decoding processes.....	56
8.3.1	General.....	56
8.3.2	State variables.....	56
8.3.3	Geometry decoding process	56
8.3.4	Default attribute values	57
8.3.5	Attribute decoding process	57

8.3.6	At the end of a slice	57
9	Slice geometry	57
9.1	General.....	57
9.2	Sensed point sequence.....	57
9.2.1	General	57
9.2.2	State variables and expressions.....	57
9.2.3	Syntax element semantics	58
9.2.4	Contextualization of elevation_offset.....	62
9.2.5	Reconstruction of sensed point positions.....	65
9.2.6	Reconstructed output point positions	75
9.2.7	Reconstructed output low latency attributes values.....	75
9.2.8	Reconstructed point positions for attributes decoding.....	75
10	Slice attributes	76
10.1	General	76
10.2	Point coordinates	76
10.2.1	General	76
10.2.2	Conversion to scaled angular coordinates	76
10.3	Syntax element semantics	76
10.4	Raw attribute decoding	77
10.5	Attribute decoding using the region-adaptive hierarchical transform	77
10.5.1	General	77
10.5.2	Transform tree	78
10.5.3	Coefficient order	80
10.5.4	Coefficient scaling.....	81
10.5.5	Transform domain prediction	83
10.5.6	Inverse transform.....	87
10.5.7	Reconstructed attribute values	89
10.6	Attribute decoding using levels of detail	89
10.6.1	General	89
10.6.2	Syntax element semantics	90
10.6.3	Reconstruction process.....	90
10.6.4	State variables	90
10.6.5	Levels of detail.....	91
10.6.6	Predictor search.....	97
10.6.7	Reconstruction of attribute values.....	103
10.6.8	Prediction mode coding	104
10.6.9	Scaling.....	106
10.6.10	Coefficient prediction.....	106
10.6.11	Transform coefficient weights	107
10.6.12	Transform.....	108
10.7	Attribute quantization parameters.....	108
10.7.1	Syntax element semantics	108
10.7.2	Per-point regional QP offset	109
10.7.3	Attribute coefficient QP	110
10.7.4	Definition of <i>AttrQstep</i>	110
11	Parsing process.....	110
11.1	General	110
11.2	Data unit buffer	112
11.2.1	General	112
11.2.2	State	112
11.2.3	Initialization at the start of parsing a data unit.....	112
11.2.4	Initialization at the start of parsing a geometry data unit footer	113

11.2.5	Definition of <i>DuNextBit</i>	113
11.3	Chunked bytestream parsing.....	113
11.3.1	General.....	113
11.3.2	Chunk syntax.....	114
11.3.3	Chunk semantics.....	114
11.3.4	State.....	114
11.3.5	Span of chunked bytestream data within a data unit.....	114
11.3.6	The chunk buffer	115
11.3.7	State update at the start of every CBS.....	115
11.3.8	Unpacking a single chunk.....	115
11.3.9	Definition of <i>ChunkNextAeBit</i>	115
11.3.10	Definition of <i>ChunkNextBpBit</i>	116
11.3.11	Boundary between spliced chunked bytestreams	116
11.3.12	Location of chunked bytestream boundaries	117
11.4	General inverse binarization processes	117
11.4.1	Parsing unsigned fixed-length codes (FL)	117
11.4.2	Parsing signed fixed-length codes (FL+S)	117
11.4.3	Parsing k -th order exp-Golomb codes (EGk).....	117
11.4.4	Parsing concatenated truncated unary and k -th order exp-Golomb codes (TU+EGk)	118
11.4.5	Parsing concatenated bounded truncated unary and k -th order exp-Golomb codes (BTU+EGk)	118
11.4.6	Parsing signed concatenated truncated unary and k -th order exp-Golomb codes (TU+EGk+S)	119
11.4.7	Parsing signed concatenated bounded truncated unary and k -th order exp-Golomb codes (BTU+EGk+S).....	119
11.4.8	Parsing signed concatenated asymmetrically bounded truncated unary and k -th order exp-Golomb codes (ABTU+EGk+S)	120
11.4.9	Parsing truncated unary codes (TU)	121
11.4.10	Mapping process for signed codes	121
11.4.11	Parsing ASN.1 object identifiers.....	121
11.5	CABAC parsing processes	122
11.5.1	Initialization.....	122
11.5.2	Definition of <i>AeReadBin</i>	122
11.5.3	Contextual probability models.....	122
11.5.4	Arithmetic decoding engine	126
11.6	Sensor Coding State	128
11.6.1	State variable	128
11.6.2	Initialization at the start of a GDU.....	130
11.6.3	Initialization with decoded coarse position of first sensed point	132
11.6.4	Update with azimuth_offset.....	132
11.6.5	Update while decoding elevation_offset	133
11.6.6	Update with sensed point coarse position	134
11.7	Parsing state memorization and restoration.....	134
11.7.1	General.....	134
11.7.2	Geometry data units.....	135
11.7.3	Attribute data units	135
11.7.4	Defaulted attribute data units.....	135
12	OBU parsing process	135
12.1	General.....	135
12.2	Creation of an OBU instance	136
12.2.1	Creation and initialization of OBU trees.....	136
12.2.2	Creation and initialization of an array of OBU ACPMs	137
12.3	Call and update of an OBU instance.....	137

12.3.1 Decode and update according to OBUF trees	137
12.4 Decode of a bin based on an OBUF ACPM.....	139
Annex A (normative) Profiles and levels.....	140
A.1 Overview of profiles and levels.....	140
A.2 Requirements on decoder capability.....	140
A.3 Profiles.....	140
A.3.1 General	140
A.3.2 Simple and Main profiles	140
A.4 Levels.....	141
A.4.1 Level limits.....	141
A.5 Permitted ranges for syntax elements	141
Annex B (normative) Type-length-value encapsulated bytestream format	145
B.1 General.....	145
B.2 Syntax and semantics	145
B.2.1 Syntax.....	145
B.2.2 Semantics.....	145
B.3 Parsing process.....	146
Annex C (informative) Arithmetic encoding engine	147
C.1 General.....	147
C.2 State variables.....	147
C.3 Initial state	147
C.4 Encoding process for a single binary symbol.....	147
C.5 Arithmetic encoder state renormalization process	148
C.6 Arithmetic encoding engine termination process	148
Bibliography	150
Index of named expressions and variables	151

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

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

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Advancements in 3D capturing and rendering technologies are enabling new applications and services in the fields of assisted and autonomous driving, cartography, and other industrial processes. Point clouds captured with LiDAR (Light Detection And Ranging) sensors, sometimes coupled with traditional camera sensors, have arisen as one of the main representations for such applications. A point cloud frame consists of a set of 3D points. Every point, in addition to having a 3D position, may also be associated with numerous other attributes such as intensity, colour, timestamp, and classification. Such representations require a large amount of data, which can be costly in terms of storage and transmission and are often used in real-time application which requires fast processing of the point cloud data. This document provides the method for efficiently compressing point cloud representations while ensuring low complexity and low latency constraints can be met.

Information technology — Coded representation of immersive media —

Part 30:

Low latency, low complexity LiDAR coding

1 Scope

This document specifies low latency, low complexity LiDAR coding.

2 Normative references

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

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

ISO/IEC 8825-1 (Rec. ITU-T X.690), 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 9834-1 (Rec. ITU-T X.660), Information technology — Procedures for the operation of object identifier registration authorities — Part 1: General procedures and top arcs of the international object identifier tree

ISO/IEC 9834-8 (Rec. ITU-T X.667), Information technology — Procedures for the operation of object identifier registration authorities — Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers

ISO/IEC 23091-2, Information technology — Coding-independent code points — Part 2: Video

3 Terms and definitions

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

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

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

IEC Electropedia: available at <http://www.electropedia.org/>

3.1 General terms

3.1.1 point

fundamental element of a *point cloud* (3.1.2) comprising a position specified as *Cartesian coordinates* (3.1.8) and zero or more *attributes* (3.1.19)

3.1.2

point cloud

unordered list of *points* (3.1.1)

3.1.3

point cloud sequence

sequence of one or more *point clouds* (3.1.2)

3.1.4

point cloud frame

point cloud (3.1.2) in a *point cloud sequence* (3.1.3)

3.1.5

coded point cloud frame

coded representation of a *point cloud frame* (3.1.4)

3.1.6

canonical point order

canonical decoding order

order of *points* (3.1.1) decoded from a *slice* (3.1.21) according to the decoding and parsing processes specified in this document

3.1.7

bounding box

axis-aligned cuboid defining a spatial region that bounds a set of *points* (3.1.1)

3.1.8

coordinates

<Cartesian> three scalar multiples of respective orthogonal *XYZ* (3.1.11) unit vectors with finite precision and bounds that specify a position relative to a fixed reference

3.1.9

coordinates

<angular> a position specified as the radial distance ρ from the V axis, an azimuth angle ϕ in the S-T plane and an indexed elevation

3.1.10

coordinates

<attribute> either *STV* (3.1.12) or scaled *RPI* (3.1.13) point coordinates used to code an attribute

3.1.11

XYZ (axes)

X, Y and Z axes, in that order, used to represent *Cartesian coordinates* (3.1.8)

3.1.12

STV (axes)

S, T and V axes, in that order, that are a sequence-dependent permutation of the *XYZ axes* (3.1.11); used to represent the coded *geometry* (3.1.18)

3.1.13

RPI (axes)

R, P and I axes, in that order, used to represent *angular coordinates* (3.1.9)

3.1.14

sequence coordinate system

scaled and translated application-specific coordinate system that applies to an entire coded *point cloud sequence* (3.1.3), and in which all *points* (3.1.1) have non-negative, fixed-point coordinates

3.1.15

coding coordinate system

scaled *sequence coordinate system* (3.1.14) that applies for an entire coded *point cloud sequence* (3.1.3), and in which all *points* (3.1.1) have non-negative integer coordinates

3.1.16

slice coordinate system

translated *coding coordinate system* (3.1.15) that applies for a single *slice* (3.1.21), and in which all *points* (3.1.1) in the *slice* have non-negative integer coordinates

3.1.17

sensing beam

sampler of point positions using angular coordinates by rays cast with a fixed elevation and from a point on and rotating around the V axis at the angular origin

3.1.18

geometry

point positions (3.4.1) associated with a set of *points* (3.1.1)

3.1.19

attribute

scalar or vector property associated with each *point* (3.1.1) in a *point cloud* (3.1.2)

EXAMPLE Colour, reflectance, frame index, etc.

3.1.20

position

<bit> bit in a binary string or value, representing the factor 2^{position}

EXAMPLE The LSB has bit position 0.

3.1.21

slice

geometry and attributes for part of, or an entire, *coded point cloud frame* (3.1.5)

Note 1 to entry: the bounding boxes of any two slices can intersect.

3.1.22

tile

set of *slices* (3.1.21) identified by a common *slice_tag syntax element* value (3.2.15) whose *geometry* (3.1.18) should be contained within a *bounding box* (3.1.7) specified in a tile inventory data unit

3.1.23

Morton code

non-negative integer obtained by interleaving the bits of three integers

3.1.24

Morton order

elements ordered according to their *Morton code* (3.1.23)

3.1.25

sparse array

array with fewer set elements than total addressable elements; unset elements can have an inferred value when accessed

3.1.26

coarse position

a coarse representation in 2D plane of position of each *point* (3.1.1) in a *point cloud* (3.1.2), used to represent the coded *geometry* (3.1.18)

3.1.27

lexicographic order

dictionary order used to order a list of points (3.1.1) by firstly in the azimuthal angle and secondly in the elevation angle of these points, and these points are decoded in the order

3.1.28

late points

points whose order is late according to the lexicographic order due to the noise of data, or transmission delay

3.1.29

Neighbour

point whose coarse position is preceding coarse position of current point according to the lexicographic order

3.1.30

Sensing heads

LiDAR capture systems may include more than one sensor (sensing head) in order to cover a larger field of view, each sensing head being able to cover a limited azimuthal range.

3.1.31

Sensing order index

The order index, per head, in which points are captured by the sensing head.

3.2 High-level syntax and entropy coding terms

3.2.1

ASN.1

abstract syntax notation one

notation specified by Rec. ITU-T X.660 | ISO/IEC 9834-1 that is used for the definition of data types, values and constraints on data types

[SOURCE: Rec. ITU-T X.660 | ISO/IEC 9834-1]

3.2.2

bin

binary symbol (bit) of the *binarized* (3.2.3) representation of a *syntax element* value (3.2.15)

3.2.3

binarization

specification of a *syntax element*'s value (3.2.15) as a sequence of *bins* (3.2.2)

3.2.4

bypass

<symbol> a static, equiprobable probability model

3.2.5

bypass

<stream> *bypass symbols* (3.2.4) that are not encoded in an arithmetic-coded *bitstream* (3.2.6)

3.2.6

bitstream

<data> sequence of bits

3.2.7

bitstream

<coded sequence> sequence of bits, in the form of encapsulated *data units* (3.2.12), that represents a coded *point cloud sequence* (3.1.3)

3.2.8

set bit

bit with the value 1

3.2.9

unset bit

bit with the value 0

3.2.10

byte

sequence of 8 bits, typeset with the most significant bit on the left and the least significant bit on the right.

Note 1 to entry: When represented in a bitstream, the most significant bit of a byte is first.

3.2.11

byte aligned

bitstream (3.2.6) position that is an integer multiple of eight bits from the position of the first bit in the bitstream

3.2.12

data unit

DU

sequence of *bytes* (3.2.10) conveying a single *syntax structure* (3.2.16) of known length

3.2.13

data unit header

parameters, located from the start of a *data unit* (3.2.12)

3.2.14

data unit footer

parameters, located from the end of a *data unit* (3.2.12)

3.2.15

syntax element

element of data represented in the *bitstream* (3.2.6)

3.2.16

syntax structure

zero or more *syntax elements* (3.2.15) present together in the *bitstream* (3.2.6) in a specified order

3.2.17

parameter set

collection of parameters that apply when activated

3.2.18

sequence parameter set

SPS

parameters for an entire coded *point cloud sequence* (3.1.3), conveyed by an SPS *data unit* (3.2.12) and activated when referenced by a geometry data unit

3.2.19**geometry parameter set****GPS**

parameters for the coding of *slice* (3.1.21) geometry, conveyed by a GPS *data unit* (3.2.12) and activated when referenced by a geometry data unit

3.2.20**attribute parameter set****APS**

parameters for the coding of a *slice* (3.1.21) attribute, conveyed by an APS *data unit* (3.2.12) and activated when referenced by an attribute data unit

3.2.21**object identifier****OID**

<ASN.1> ordered list of primary integer values from the root of the *international object identifier tree* (3.2.22) to a node, which unambiguously identifies that node

[SOURCE: Rec. ITU-T X.660 | ISO/IEC 9834-1]

3.2.22**international object identifier tree**

tree whose root corresponds to Rec. ITU-T X.660 | ISO/IEC 9834-1 and whose nodes correspond to *registration authorities* (3.2.24) responsible for allocating arcs from a parent node

[SOURCE: Rec. ITU-T X.660 | ISO/IEC 9834-1]

3.2.23**registration**

<object identifier> assignment of an unambiguous name to an object in a way which makes the assignment available to interested parties

[SOURCE: Rec. ITU-T X.660 | ISO/IEC 9834-1]

3.2.24**registration authority**

<international object identifier tree> an entity such as an organization, a standard or an automated facility that performs *registration* (3.2.23) of one or more types of objects

[SOURCE: Rec. ITU-T X.660 | ISO/IEC 9834-1]

3.2.25**application specific**

defined by an application or an application standard

3.2.26**unspecified**

when used in subclauses specifying values of a particular *syntax element*, indicates that the values have no specified meaning in this document and will not have a specified meaning in future versions of this document

3.2.27**geometry entropy stream**

Stream of entropy coded sequence of points information containing for each head, geometry information,

3.2.28**low latency attributes entropy stream**

Stream of entropy coded sequence of points information containing for each head, low-latency attributes

3.2.29**L3C2 entropy stream**

Stream of entropy coded sequence of points information containing for each head, geometry information, low-latency attributes and other attributes.

3.3 Tree structure terms**3.3.1****tree**

recursive structure of *nodes* (3.3.4) without loops, and containing a single *root node* (3.3.3)

3.3.2**tree level**

set of *nodes* (3.3.4) at the same *depth* (3.3.5) in a *tree* (3.3.1)

3.3.3**root node**

<tree> *node* (3.3.4) without a *parent node* (3.3.10)

3.3.4**node**

<tree> element of a *tree* (3.3.1)

3.3.5**depth**

<node> number of descendent hops from the *root node* (3.3.3) to a *node* (3.3.4)

3.4 Geometry coding terms**3.4.1****position**

<point> three-dimensional coordinates of a *point* (3.1.1)

3.5 Attribute coding terms**3.5.1****low latency attributes**

specific additional attributes coded altogether with geometry to support low latency decoding (9.2.5.3.3).

3.5.2**primary attribute component**

first, or only, attribute component, identified by the index 0

3.5.3**secondary attribute component**

attribute component other than the first component, identified by an index greater than 0

3.5.4**detail level**

set of *points* (3.1.1) that represent a subsampled version of the slice *geometry* (3.1.18)

3.5.5**refinement list**

set of *points* (3.1.1) present in one *detail level* (3.5.4) that are not present in the next coarsest *detail level*

3.5.6**refinement point**

point (3.1.1) in a *refinement list* (3.5.5)

3.5.7**predictor set**

set of neighbouring *points* (3.1.1) from which an *attribute* (3.1.19) value is predicted

4 Abbreviated terms**4.1 Acronyms**

APS	Attribute parameter set
ADU	Attribute data unit
CBS	Chunked bytestream
CPM	Contextual probability model
DU	Data unit
FBDU	Frame boundary marker data unit
FSAP	Frame-specific attribute properties
GDU	Geometry data unit
GPS	Geometry parameter set
L3C2	Low Latency, Low Complexity LiDAR coding
LiDAR	Light Detection And Ranging
LoD	Level(s) of detail
LSB	Least significant bit
MSB	Most significant bit
NA	Not applicable
QP	Quantization parameter
RAHT	Region adaptive hierarchical transform
SPS	Sequence parameter set

4.2 Mnemonics

EGk	Exponential Golomb code of order <i>k</i>
FL	Fixed-length code
FL+S	Fixed-length code plus conditional sign bit
TU	Truncated unary code
attr	Attribute
cnt	Count
geom	Geometry
idx	Index
ti	Tile inventory
tlv	Type-length-value
seq	Sequence

5 Conventions

5.1 General

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

5.2 Symbolic names

Variables and expressions use the following case-insensitive naming conventions to indicate their use:

- *b*: sensing beam index
- *h*: sensor head index
- *i, j*: general loop or index variable
- *k*: component of an XYZ/STV/RPI position, coordinate or location
- *c*: component of an attribute
- *qc*: quantization-parameterized component: 0 – primary; 1 – secondary
- *dpth*: the depth of a node or level in a tree
- *lvl*: tree level or detail level, counted from the bottom of a tree or hierarchy
- *m*: Morton-coded location
- *ns, nt, nv*: node coordinates
- *ptldx*: index of a point in canonical decoding order
- *rfmtldx*: index of a point in an array of LoD refinement points
- *ni*: index for an element in a point's predictor set

5.3 Numerical representation

binary	typeset as 'X...XX' where each base 2-digit X is 0 or 1
octal	typeset as X...XX ₈ where each base 8-digit X is 0 to 7
decimal	typeset as X...XX where each base 10-digit X is 0 to 9
hexadecimal	typeset as 0xX...XX where each base 16-digit X is 0 to 9 or A to F

5.4 Arithmetic operators

+	Addition
−	Subtraction (as a two-argument operator) or negation (as a unary prefix operator)
×	Multiplication
x^y	Exponentiation. Specifies <i>x</i> to the power of <i>y</i> . 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 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 $y > 0$

5.5 Logical operators

$x \&\& y$	Conditional boolean logical "and" of x and y ; the operand y is only evaluated if x is true.
$x y$	Conditional boolean logical "or" of x and y ; the operand y is only evaluated if x is false.
$\neg$	Boolean logical "not"
$x ? y : z$	If x is true or not equal to 0, evaluates to y ; otherwise, evaluates to z

5.6 Relational operators

$>$	Greater than
$\geq$	Greater than or equal to
$<$	Less than
$\leq$	Less than or equal to
$==$	Equal to
$\neq$	Not equal to

5.7 Bit-wise operators

$\&$	Bit-wise "and". When operating on integer arguments, operates upon a two's complement representation of the integer value. When operating upon a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding MSBs equal to 0.
$ $	Bit-wise "or". When operating on integer arguments, operates upon a two's complement representation of the integer value. When operating upon a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding MSBs equal to 0.
$\wedge$	Bit-wise "exclusive or". When operating on integer arguments, operates upon a two's complement representation of the integer value. When operating upon a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding MSBs equal to 0.
$x \gg y$	Arithmetic right shift as specified by $\text{DivExp2Floor}(x, y)$. It is equivalent to shifting a two's complement integer representation of x by y binary digits. This operator is defined only for non-negative integer values of y .
$x \ll y$	Arithmetic left shift of a two's complement integer representation of x by y binary digits. This operator 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.

NOTE According to the rules of precedence (5.11), the expressions $a + b \ll c + d$ and $a | b \ll c | d$ are identical to $(a + b) \ll (c + d)$ and $a | (b \ll c) | d$, respectively, and not $a + (b \ll c) + d$ or $(a | b) \ll (c | d)$.

5.8 Assignment operators

$=$	Assignment operator
-----	---------------------

<code>:=</code>	Expression definition (5.12)
<code>++</code>	Increment, i.e. <code>x++</code> is equivalent to <code>x = x + 1</code> ; when used in an array index, evaluates to the value of the variable prior to the increment operation
<code>--</code>	Decrement, i.e. <code>x--</code> is equivalent to <code>x = x - 1</code> ; when used in an array index, evaluates to the value of the variable prior to the decrement operation
<code>×</code>	Multiply by amount specified and update, i.e. <code>x ×= 3</code> is equivalent to <code>x = x × 3</code>
<code>+=</code>	Increment by amount specified, i.e. <code>x += 3</code> is equivalent to <code>x = x + 3</code> , and <code>x += (-3)</code> is equivalent to <code>x = x + (-3)</code>
<code>-=</code>	Decrement by amount specified, i.e. <code>x -= 3</code> is equivalent to <code>x = x - 3</code> , and <code>x -= (-3)</code> is equivalent to <code>x = x - (-3)</code>
<code>>>=</code>	Arithmetic right shift by amount specified, i.e. <code>x >>= 1</code> is equivalent to <code>x = x >> 1</code>
<code><<=</code>	Arithmetic left shift by amount specified, i.e. <code>x <<= 1</code> is equivalent to <code>x = x << 1</code>

5.9 Range notation

`x = a .. b` `x` takes on monotonically increasing integer values starting from `a` and proceeding to `b`, inclusive, with `x`, `a` and `b` being integer numbers

5.10 Mathematical functions

5.10.1 General

<code>ArcSin(x)</code>	Trigonometric arc sine function
<code>Abs(x)</code>	$\begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases}$
<code>Bit(x, i)</code>	$(x \gg i) \& 1$
<code>Clip3(min, max, x)</code>	$\begin{cases} min & ; \quad x < min \\ max & ; \quad x > max \\ x & ; \quad \text{otherwise} \end{cases}$
<code>Cos(x)</code>	Trigonometric cosine function
<code>DivExp2Floor(x, p)</code>	$\text{Floor}(x \div 2^p)$
<code>DivExp2Fz(x, p)</code>	$\text{RoundFz}(x \div 2^p)$
<code>DivExp2Up(x, p)</code>	$\text{RoundUp}(x \div 2^p)$
<code>Exp2(p)</code>	2^p
<code>Floor(x)</code>	Greatest integer less than or equal to <code>x</code>
<code>Gcd(a, b)</code>	Greatest integer that is a factor of both <code>a</code> and <code>b</code>
<code>IntLog2(x)</code>	$\text{Floor}(\text{Log}(x) \div \text{Log}(2))$
<code>Log(x)</code>	Natural logarithm of the argument <code>x</code>
<code>Min(x, y)</code>	$\begin{cases} x & ; \quad x \leq y \\ y & ; \quad x > y \end{cases}$
<code>Max(x, y)</code>	$\begin{cases} x & ; \quad x \geq y \\ y & ; \quad x < y \end{cases}$
<code>RoundFz(x)</code>	$(2 \times x + \text{Sign}(x)) / 2$
<code>RoundUp(x)</code>	$(2 \times x + 1) / 2$

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

$\text{Sin}(x)$ Trigonometric sine function

5.10.2 IntCos and IntSin

The functions $x = \text{IntCos}(\theta)$ and $y = \text{IntSin}(\theta)$ are 24-bit fixed-point approximations of the cosine and sine of θ . Their:

- parameter θ is an integer value representing a 24-bit fixed point which specify an angle measured in Radians;
- result for IntCos shall be equal to the value of the expression $i\text{Cos}[\theta]$ and result for IntSin shall be equal to the value of the expression $i\text{Sin}[\theta]$.

The expression $i\text{Cos}$ is obtained by a $\pi \div 2$ rotation of $i\text{Sin}$.

```
iCos[theta] := iSin[26353589 - Abs(theta)]
```

The expression $i\text{Sin}$ is obtained by mapping θ to $[0; \pi \div 2)$ interval and linearly interpolating the function approximation tabulated on this interval and adjusting the sign.

```
iSin[theta] := theta >= 0 ? sinOnHalfPi : -sinOnHalfPi
where
  nextSinFp := idx < 1608 ? SinFp[idx + 1] : (1 << 24)
  sinOnHalfPi := SinFp[idx] + (lambda * (nextSinFp - SinFp[idx]) >> 14)
  where
    theta1 := Abs(theta)
    theta2 := theta1 >= 52707179 ? theta1 - 52707179 : theta1
    theta3 := theta2 >= 26353589 ? 52707179 - theta2 : theta2
    idx := theta3 >> 14
    lambda := theta3 - (idx << 14)
```

The expression $\text{SinFp}[x]$ specifies the 24-bit fixed-point approximation of the sine function tabulated for integers x representing 10-bit precision fixed-point angle expressed in Radian in the interval $[0; \pi \div 2)$.

$$\text{SinFp}[x] := \left\lfloor 2^{24} \sin\left(\frac{x}{2^{10}}\right) + \frac{1}{2} \right\rfloor$$

5.10.3 IntSqrt

The function $r = \text{IntSqrt}(x)$ is an integer approximation of the principal square root of x . Its:

- parameter x is a non-negative integer;
- result shall be equal to the value of the expression $\text{intSqrt}[x]$.

It is specified in terms of the fixed-point reciprocal square root. If the parameter x is greater than or equal to 2^{46} , the calculation uses a quantized value of x to ensure computability using 64-bit arithmetic.

NOTE $\text{IntSqrt}(0)$ is 1.

```
intSqrt[x] := x <= Exp2(46)
  ? 1 + (x * IntRecipSqrt(x) >> 40)
  : 1 + (x8 * IntRecipSqrt(x8) >> 32)
where
  x8 := DivExp2(x, 16) + 1
```

5.10.4 IntRecipSqrt

The function $r\text{Recip} = \text{IntRecipSqrt}(x)$ is a 40-bit fixed-point approximation of the reciprocal square root of x . Its:

- parameter x is a non-negative integer;

— result shall be equal to the value of the expression $\text{IntRecipSqrt}[x]$.

NOTE $\text{IntRecipSqrt}(0)$ is 0.

The parameter x is scaled to be in the range $[2^{30}, 2^{32} - 1]$ in the expression x_{Scaled} by multiplying or dividing by a power of four. The expression $x_{\text{ScaleLog4}}$ is the log4 scale factor.

```
xScaled := Floor(x * Exp2(2 * xScaleLog4))
xScaleLog4 := 15 - IntLog2(x) / 2
```

The reciprocal square root shall be determined by two rounds of the Newton–Raphson method. The initial approximation for the scaled parameter x is specified by the expression approxR0 . Table 1 specifies the initial approximants over the domain of approxR with 18 fractional bit precision.

```
approxR0 := approxR[xScaled >> 25]
```

The second approximation from the first round of the Newton–Raphson method is specified by the expression approxR1 .

```
approxR1 := threeR0[approxR0] - (rCubed0[approxR0] * xScaled >> 32)
threeR0[r] := 3 * DivExp2Fz(r, 18) << 22
rCubed0[r] := DivExp2Fz(r * r * r, 54) << 8
```

The third approximation from the second round of the Newton–Raphson method is specified by the expression approxR2 .

```
approxR2 := threeR1[approxR1] - rCubed1[approxR1] >> 32
threeR1[r] := r * 3 << 28
rCubed1[r] := r * (r * (r * pInScaled >> 32) >> 32)
```

The result is obtained by scaling the third approximation by the square root of the initial scale factor.

```
intRecipSqrt[x] := x > 0 ? Floor(approxR2 * Exp2(xScaleLog4 - 3)) : 0
```

Table 1— Initial approximations $\text{approxR}[i+j]$ for $\text{IntRecipSqrt}(i+j << 25)$. Values are typeset in hexadecimal form without the 0x prefix.

j	i							
	0	1	2	3	4	5	6	7
32	3F7FFDA	3E7FFB7	3DBFFBD	3CC0013	3BFFFEE	3AFFFE1	3A3FFDE	397FF96
40	38FFFDE	383FFC0	3780063	36FFFCF	3640014	35BFFFA	34FFF8B	3480010
48	3400039	3380042	3300008	328002B	3200046	317FFC2	3100012	307FFDB
56	303FFC5	2FC004F	2F3FFE0	2EFFF93	2E7FF91	2E3FF83	2DC0037	2D7FFB6
64	2D0000B	2CBFF96	2C7FF66	2C00017	2BC0053	2B7FFA4	2AFFF43	2AC004B
72	2A80061	2A3FF6A	2A00032	29BFF3B	297FF74	293FFC9	28C001E	287FF6D
80	283FF9E	27FFF93	27BFFB1	27BFFE8	277FF3C	2700056	2700000	26C0069
88	26800CE	26400CD	25FFF5A	25BFFA8	25BFFD6	258008C	25400B9	2500020
96	24BFF93	24C00B6	24800E6	24400B3	2400011	2400054	23C0049	237FF98
104	2380104	233FFFC	2300047	2300024	22C0029	2280000	2280012	223FF79
112	223FF54	21FFF56	21BFFDE	21C0078	2180111	217FF3D	2140023	20FFF0F
120	21000F1	20C0019	20C0137	2080015	2080091	204004F	20400F7	200006B

5.10.5 Div

The function $\text{quotient} = \text{Div}(\text{dividend}, \text{divisor}, \text{fracBits})$ is a fixed-point approximation of $\text{dividend} \div \text{divisor}$. Its:

— parameters dividend and divisor are integers;

- parameter *fracBits* is the number of fractional bits in the fixed-point result;
- result is specified by the value of the expression *quotient*.

```
quotient := dividend × recipDivisor[idx] >> 16 + excess - fracBits
where
  idx := DivExp2Fz(divisor, excess)
  excess := Max(0, IntLog2(divisor) - 7)
  recipDivisor[idx] := RoundFz(Exp2(16) ÷ idx)
```

5.10.6 Morton

The function $m = \text{Morton}(s, t, v)$ converts its parameters to a 3D Morton code. Its:

- parameters *s*, *t* and *v* are non-negative integers;
- result is specified by the value of the expression *morton*.

The conversion interleaves the bits of each parameter *v*, *t* and *s*; in that order, starting from the LSBs. The LSB of *v* is the LSB of *m*. Table 2 illustrates the construction of 3D Morton codes from the bit string representation of the parameters *s*, *t* and *v*.

$$morton := \sum_{i=0} 2^{3i+2} \text{Bit}(s, i) + 2^{3i+1} \text{Bit}(t, i) + 2^{3i} \text{Bit}(v, i)$$

The expression $\text{Morton}[expr]$ performs the same conversion for an expression *expr* that takes an argument *k*, $k \in \{0, 1, 2\}$.

```
Morton[expr] := Morton(expr[0], expr[1], expr[2])
```

Table 2 — Construction of 3D Morton codes *m* from the tuple (*s*, *t*, *v*)

Bit string form				Decimal form
<i>s</i>	<i>t</i>	<i>v</i>	<i>m</i>	<i>m</i>
'0 0'	'0 0'	'0 0'	'0 0 0 0 0 0'	0
'0 0'	'0 0'	'0 1'	'0 0 0 0 0 1'	1
'0 1'	'1 1'	'1 0'	'0 1 1 1 1 0'	30
'0 1'	'1 1'	'1 1'	'0 1 1 1 1 1'	31
'1 0'	'0 1'	'1 0'	'1 0 1 0 1 0'	42
'1 0'	'0 1'	'1 1'	'1 0 1 0 1 1'	43
'1 1'	'1 0'	'0 0'	'1 1 0 1 0 0'	52
'1 1'	'1 0'	'0 1'	'1 1 0 1 0 1'	53
$s_n \dots s_1 s_0$	$t_n \dots t_1 t_0$	$v_n \dots v_1 v_0$	$s_n t_n v_n \dots s_1 t_1 v_1 s_0 t_0 v_0$	...

5.10.7 FromMorton

The function $(s, t, v) = \text{FromMorton}(m)$ is the inverse of $\text{Morton}(s, t, v)$. Its:

- parameter *m* is a non-negative, integer, 3D Morton code;
- result is the tuple specified by the value of the expressions *s*, *t* and *v*.

The conversion deinterleaves the bits of *v*, *t* and *s*; in that order, starting from the LSB. The LSB of *m* is the LSB of *v*.

$$\begin{aligned}
 s &:= \sum_{i=0} 2^i \text{Bit}(m, 3i + 2) \\
 t &:= \sum_{i=0} 2^i \text{Bit}(m, 3i + 1) \\
 v &:= \sum_{i=0} 2^i \text{Bit}(m, 3i)
 \end{aligned}$$

5.11 Order of operation precedence

When order of precedence in an expression is not indicated explicitly by round brackets, 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 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 3 — Operation precedence from highest (at top of table) to lowest (at bottom of table)

Operations (with operands x, y and z)
$x++, x--$
$\neg x, \neg x$ (as a unary prefix operator)
x^y
$x \times y, x / y, x \div y, \frac{x}{y}, x \% y$
$x + y, x - y$ (as a two-argument operator), $\sum_{i=x}^y f(i)$
$x << y, x >> y$
$x < y, x \leq y, x > y, x \geq y$
$x == y, x \neq y$
$x \& y$
$x \wedge y$
$x y$
$x \&\& y$
$x y$
$x ? y : z$
$x .. y$
$x = y, x += y, x -= y, x \times = y, x < < = y, x > > = y$

5.12 Named expressions

5.12.1 General

Operations and values in this document are sometimes specified in the form of named expressions. Exemplar named expressions are described in Table 4.

A named expression is a named macro-like statement. Every occurrence of a named expression is substituted by its definition when evaluated. The definition is provided by the $:=$ operator.

Substitution is atomic. For example, the substitution for $3 \times \text{ExAPlusB}$ is equivalent to $3 \times (a + b)$, not $(3 \times a) + b$.

The definition for a named expression is immutable. For example, ExTwo is equivalent to the value 2; unlike a variable it cannot be modified. All instances of ExTwo could be substituted by the numeric value 2.

The substitution for a named expression can be a variable. The substituted variable in such cases can be modified. For example, $\text{ExVar}++$ increments the variable Var .

Table 4 — Examples of named expressions

Example	Remarks
<code>ExTwo := 2</code>	<i>ExTwo</i> is equivalent to the value 2
<code>ExAPlusB := a + b</code>	<i>ExAPlusB</i> is equivalent to $(a + b)$
<code>ExTwoIndirect := ExTwo</code>	<i>ExTwoIndirect</i> is equivalent to <i>ExTwo</i>
<code>Var = 2</code> <code>ExVar := Var</code>	<i>ExVar</i> is equivalent to (an alias of) the variable <i>Var</i>
<code>ExTimesTwo[i] := 2 × i</code>	<i>ExTimesTwo</i> [<i>j</i> + 1] is equivalent to $2 \times (j + 1)$
<code>ExSquared[i] := i × i</code>	<i>ExSquared</i> [<i>ExVar</i> ++] is equivalent to <i>Var</i> × <i>Var</i> , with <i>Var</i> incremented after the evaluation of <i>ExSquared</i>
<code>for (Var = 0; Var ≤ 10; Var++)</code> <code>sum += ExTimesTwo[Var]</code>	<i>sum</i> is incremented, in total, by 110
<code>ExWhere[i] := ExTimesTwo[inner]</code> <code>where</code> <code>inner := i + 1</code>	<i>ExWhere</i> [<i>j</i>] is equivalent to $2 \times (j + 1)$
<code>ExSumA[i] := i > 0</code> <code>? i + ExSumA[i - 1]</code> <code>: 0</code>	Recursive definition. <i>ExSumA</i> [10] evaluates to 55
<code>ExSumB[i] :=</code> <code>ExSumB = 0</code> <code>for (; i > 0; i--)</code> <code>ExSumB += i</code>	Imperative definition using multiple statements. <i>ExSumB</i> [10] evaluates to 55
<code>ExSum10[expr] :=</code> <code>ExSum10 = 0</code> <code>for (i = 0; i ≤ 10; i++)</code> <code>ExSum10 += expr[i]</code>	<i>ExSum10</i> applies <i>expr</i> to each <i>i</i> , $i \in 0 \dots 10$, summing the result. <i>ExSum10</i> [<i>ExprTimesTwo</i>] evaluates to 110 <i>ExSum10</i> [<i>ExprSquared</i>] evaluates to 385

5.12.2 Scope

The scope of a named expression is not affected by the relative order of its definition and use; a named expression can be referenced earlier in the document than its definition.

Named expressions identified by a capital initial are "global" definitions that apply to the whole document. They may be directly referenced in other subclauses.

Named expressions identified by a lower-case initial are "local" definitions that apply to the subclause in which they are defined.

If a global definition references a local definition in the same subclause, that local definition is used when the global definition is referenced in another subclause.

5.12.3 Arguments of named expressions

The definition of an expression can be in terms of one or more parameters. Each parameter is enclosed in square brackets. For example, the definition *ExTimesTwo*[*i*] has a single parameter *i*.

A named expression can be applied to one or more arguments. When the definition is substituted for a named expression, every instance of each parameter is replaced by the text of the corresponding argument.

Replacements are atomic. For example, *ExTimesTwo*[*j* + 1] is equivalent to $2 \times (j + 1)$, not $(2 \times j) + 1$.

5.12.4 Sub-expressions

A definition can contain a where-clause that defines further named expressions. They apply only to the definition containing the where-clause. For example, the definition of *ExWhere*[*i*] defines the sub-expression *inner*.

5.12.5 Definitions with multiple statements

Some definitions cannot be succinctly expressed by a single statement. In such cases, a definition can consist of multiple statements. The evaluated value for the whole definition is specified by assignments or modifications to a variable with the same name as the named expression. For example, *ExSumB*.

5.12.6 Textual definitions

Some definitions are provided by a descriptive equivalence in textual or tabular form. For example:

- "The expression *Ex[i]* is specified by Table X (Value for *Ex[i]*)."
- "The value for the expression *Ex* is specified by Table X for each axis *k*."
- "The expression *Ex* is equivalent to the following [procedural code]."

5.13 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 use the values of variables derived from other syntax elements' values. Such variables appear in the syntax tables, or text, named by a mixture of lower- and upper-case letters and without any underscore characters. Variables with a capital initial are valid for the decoding of the current syntax structure and all dependent syntax structures. They may be used in the decoding process for later syntax structures without mentioning their origin. Variables with a lower-case initial are only used within the clause in which they are derived.

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

Functions that specify properties of the current position in the bitstream are referred to as syntax functions (7.2). These functions 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 brackets including zero or more parameter names (for definition) or arguments (for usage), separated by commas (if more than one).

Functions that are not syntax functions (including mathematical functions specified in 5.10) are described by their names, which start with a capital initial, contain a mixture of lower- and upper-case letters without any underscore characters and end with left and right round brackets surrounding zero or more parameter names (for definition) or arguments (for usage), separated by commas (if more than one).

Arrays are sequences of values identified by a common name. Both syntax elements and variables can be arrays. Subscripts or square brackets are used to index an array.

Boolean true and false values are interchangeable with the integers 1 and 0, respectively; non-zero integers are equivalent to true.

6 Point cloud format and relationship to coded and output representations

6.1 General format

A point cloud is an unordered list of points representing geometry, optional attribute information and associated metadata. Geometry information describes the location of points in a three-dimensional Cartesian coordinate system. Attributes are typed properties of each point, such as its colour or

reflectance. Metadata is information used to interpret the point cloud, the point geometry and the attribute data.

This specification is intended for point clouds where at least geometry information is captured by LiDAR. As such L3C2 point clouds are ordered according to the LiDAR sensor characteristics (typically following the capture order) and the ordering is mostly kept unchanged in order to ensure low latency processing.

Typical attributes that may be present in L3C2 point clouds are attributes captured by a LiDAR sensor such as reflectance, intensity, return number, number of returns, timestamp, classification. Additional attributes may also be present when the LiDAR sensor is coupled with other types of sensors : this is for instance the case of RGB attributes that may be captured by a traditional camera associated to the LiDAR sensor.

Each point in a point cloud is a tuple of a three-dimensional position and attribute values for every attribute present in the point cloud. All points shall have the same number of attributes in the same order.

Point cloud metadata may describe, for example, a geometric transformation used to map points to another coordinate system, spatial regions (tiles) within a point cloud, the identification of attribute types and how attribute values are interpreted.

6.2 Attributes

6.2.1 General

An attribute comprises one or more components.

A point cloud, unless otherwise specified, may contain more than one instance of a particular attribute. The significance or interpretation of multiple instances of the same attribute is unspecified.

Metadata can be associated with each attribute instance. Attribute metadata conveys sequence level characteristics such as an attribute label identifier or a frame level interpretation such as an attribute scale and offset information.

6.2.2 Colour

The colour attribute specifies the colour of a point. The attribute shall comprise one of the following configurations:

- A luma (Y') component (monochrome).
- A luma component and two chroma components ($Y' C_B C_R$ or $Y' C_G C_O$).
- Green, blue and red components ($G' B' R'$, also known as *RGB*).
- Other unspecified monochrome or tri-stimulus colour systems (e.g., *YZX*, also known as *XYZ*).

The actual colour representation method in use is described using ISO/IEC 23091-2 coding-independent video code points and is indicated using syntax specified in 7.3.2.7.

The ordering of attribute components is specified by Table 5.

Table 5 — Relationship between colour components and attribute components

Colour representation	Attribute component index		
	0	1	2
Monochrome	Y'	–	–
$Y' C_B C_R$	Y'	C_B	C_R
$Y' C_G C_O$	Y'	C_G	C_O
$G' B' R'$ or RGB	G'	B'	R'
YZX or XYZ	Y	Z	X

6.2.3 Opacity

Opacity is a single component attribute. When normalized to the interval $[0, 1]$, the value 0 indicates that a point is completely transparent and the value 1 indicates complete opacity. The opacity attribute may be used to control colour blending when rendering a colour attribute.

NOTE Opacity is often called an alpha channel or transparency.

6.2.4 Reflectance

Reflectance is a single component attribute that represents the ratio of incident light reflected by a point; it is a dimensionless quantity. Values are bounded by a minimum that indicates complete absorption and a maximum that indicates complete reflection or saturation.

6.2.5 Normal vector

A normal vector is a three-component attribute representing a vector perpendicular to the surface tangent plane at an associated point. The axes identification of the normal vectors is identical to that of the STV axes for the coded point cloud geometry. The length of a normal vector is not required to be one.

Normal vectors may be used when rendering a point cloud. A point's appearance may be modified according to the difference between the incident light direction and its normal vector.

6.2.6 Material identifier

A material identifier is a single component attribute that associates a point with a material from a range of materials. Points with a common material identifier share a characteristic that may be used to identify an object or type of object. Materials are not specified by this document.

6.2.7 Frame number/index

The frame number and frame index attributes are single component attributes that indicate how a point cloud frame may be partitioned into one or more ordered sub-frames. Each sub-frame is a partial representation of a point cloud frame, comprising points with the same frame number/index attribute value.

NOTE Sub-frame partitioning does not form part of the decoding or output processes specified by this document.

A point cloud sequence shall contain no more than one instance of a frame number/index attribute. A point cloud sequence shall not contain both frame number and frame index attributes.

The frame number attribute may be used to order all sub-frames over the entire point cloud sequence. Points from different point cloud frames shall not have the same value for the frame number attribute.

The frame index attribute may be used to order the sub-frames of a single point cloud frame.

An example of the relationship between frames, sub-frames and their ordering is shown in Table 6. The point cloud frames *a*, *b* and *c* are partitioned into sub-frames. Sub-frame orders are shown for the cases where the attribute is a frame number or a frame index.

Table 6 — Example partitioning of three consecutive frames a, b and c into sub-frames

	Frame							
	<i>a</i>		<i>b</i>		<i>c</i>			
Frame number attribute								
Sub-frame attribute value	0	2	1	3	5	4	6	7
Sub-frame presentation order	<i>a</i> ₀	<i>a</i> ₁	<i>a</i> ₂	<i>b</i> ₃	<i>c</i> ₄	<i>b</i> ₅	<i>c</i> ₆	<i>c</i> ₇
Frame index attribute								
Sub-frame attribute value	0	2	1	0	1	0	1	3
Sub-frame presentation order	<i>a</i> ₀	<i>a</i> ₁	<i>a</i> ₂	<i>b</i> ₀	<i>b</i> ₁	<i>c</i> ₀	<i>c</i> ₁	<i>c</i> ₃

6.2.8 User defined attributes

The point cloud format supports attributes other than those specified in this document. A user defined attribute shall be identified by an international object identifier. The international object identifier shall either be assigned by a registration authority in accordance with Rec. ITU-T X.660 | ISO/IEC 9834-1, or generated without registration using a universally unique identifier (UUID) as specified by Rec. ITU-T X.667 | ISO/IEC 9834-8.

6.3 Codec-derived attributes

6.3.1 General

Codec-derived attributes represent values that are determined as side-effects of a processes specified in this document.

A decoder may, but is not required to, output one or more codec-derived attributes. Any codec-derived attributes output by a decoder shall conform to the definitions in 6.3.

6.3.2 Slice identifier

The slice identifier attribute shall be a single component attribute that identifies the slice from which a point is decoded. Identification shall use the `slice_id` syntax element value.

6.3.3 Slice tag

The slice tag attribute shall be a single component attribute that identifies the group of slices from which a point is decoded. Identification shall use the `slice_tag` syntax element value.

6.3.4 Canonical point order

The canonical point order attribute shall be a single component attribute that specifies the order within a slice in which points are decoded by the geometry decoder as specified in this document.

Values of the point decoding order attribute shall be equal to `ptIdx` of the corresponding point `PointPos[ptIdx]` in a slice.

6.3.5 Point Morton order

The point Morton order attribute shall be a single component attribute that specifies the order of points within a slice according to ascending values of Morton-coded STV slice position (i.e. prior to 8.3.6).

The Morton order shall be equivalent to the order of points in the finest detail level specified in 10.6.5.2 as if both `attr_canonical_order_enabled` and `attr_coord_conv_enabled` are both 0.

For example, if three points *a*, *b* and *c* in canonical point order are ordered {*a*, *c*, *b*} in the finest detail level, then the respective values for the Morton order attribute are 0, 2 and 1.

6.4 Coded point cloud format

6.4.1 Sequence coordinate system

The sequence coordinate system is specified by the position of its origin in an externally defined application-specific coordinate system and by the length of its unit vectors.

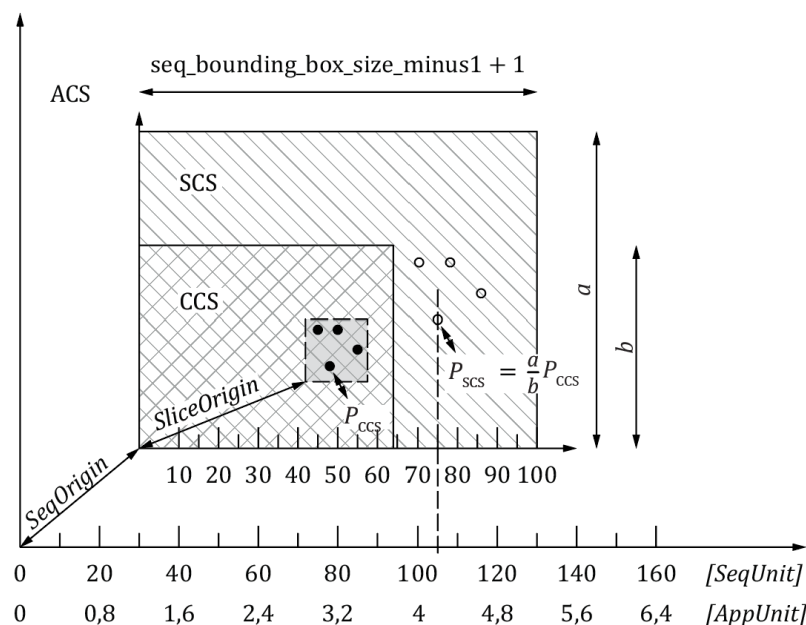
All points in a coded point cloud sequence shall have non-negative coordinates in the sequence coordinate system.

A position P_{SCS} in the sequence coordinate system is related to the position P_{ACS} in the application coordinate system by the sequence origin *SeqOrigin* and the unit vector length *SeqUnit* specified by the active SPS:

$$P_{ACS} = (P_{SCS} + SeqOrigin) \times SeqUnit$$

The maximum bound on the sequence coordinate system depends upon the level to which the coded point cloud sequence conforms, as specified in Annex A.

An example sequence coordinate system (marked SCS) is illustrated in Figure 1. A point P_{SCS} with an *x*-coordinate of 75 in the sequence coordinate system has a position in the application-specific coordinate system of 105 *SeqUnit*. If *SeqUnit* is 0,8 *AppUnit*, the *x*-coordinate of the point in the application-specific coordinate system (marked ACS) is 4,2.



Key

ACS	Application-specific coordinate system
CCS	Coding coordinate system
SCS	Sequence coordinate system
$a:b$	Ratio of SCS to CCS

Figure 1 — Relationship between application, sequence, and coding coordinate systems.**6.4.2 Coding coordinate system**

The coding coordinate system is a non-negative integer coordinate system used to code point positions. It is either identical to, or a geometric contraction of the sequence coordinate system. Its origin is coincident with the sequence coordinate system origin.

A position P_{CCS} in the coding coordinate system is related to the position P_{SCS} in the sequence coordinate system by the binary fixed-point scale factor *SeqCodedScale*:

$$P_{SCS} = P_{CCS} \times SeqCodedScale$$

An example coding coordinate system (marked CCS) is illustrated in Figure 1. A point P_{CCS} with an x -coordinate of 48 in the coding coordinate system has an x -coordinate, P_{SCS} , of 75 when scaled by the scale factor $SeqCodedScale = a \div b = 25 \div 16$.

Point position components in the coding coordinate system shall satisfy the following level dependent (Annex A) constraint:

$$P_{SCS} \times SeqCodedScaleN < \text{Min} \left(2^{MaxSeqBboxDimLog2}, 2^{32} - \frac{1}{2} SeqCodedScaleD \right)$$

Where *SeqCodedScaleN* and *SeqCodedScaleD* are the numerator and denominator, respectively, of *SeqCodedScale* when represented as an irreducible fraction:

$$\frac{SeqCodedScaleN}{SeqCodedScaleD} = SeqCodedScale, \quad \text{Gcd}(SeqCodedScaleN, SeqCodedScaleD) = 1$$

NOTE This constraint guarantees that conversion from the coding coordinate system to the sequence coordinate system can be performed using 32-bit arithmetic.

6.4.3 Coded point cloud sequence

The coded representation of a point cloud sequence comprises one or more point cloud frames encoded as a sequence of encapsulated DUs that convey syntax structures as specified in 7.3. An encapsulation format is specified by Annex B. Alternative encapsulation formats may be specified by the application.

The coded point cloud sequence shall include:

- An SPS that enumerates the attributes present in the coded point cloud format and conveys both metadata and decoding parameters that pertain to the whole coded point cloud sequence.
- Any GPSs that convey parameters required for the decoding of geometry data.
- Any APSs that convey parameters required for the decoding of attribute data.
- The slices comprising each coded point cloud frame.

6.4.4 Coded point cloud frame

A coded point cloud frame comprises a sequence of zero or more slices with the same value of a notional frame counter *FrameCtr* (8.2.2). An optional frame boundary marker data unit explicitly signals the end of a frame.

It is a requirement of bitstream conformance that:

- Every coded point cloud frame shall have a unique value of *FrameCtr* within the sequence.
- Coded point cloud frames shall be ordered such that the notional frame counter increases for each successive coded point cloud frame.

An empty frame shall be signalled by a frame boundary marker data unit without any preceding slices with the same value of *FrameCtr*.

A coded point cloud frame independently codes a single point cloud frame without dependencies upon any previous or subsequent point cloud frame.

A decoded point cloud frame is the concatenation of all points in all constituent slices of the frame.

Unless prohibited by an SPS constraint, coincident points in a point cloud frame may arise from:

- points coded in a single slice with a non-zero duplicate point count;
- distinct points with the same position in a single slice; or
- the concatenation of multiple slices.

6.4.5 Slice of a coded point cloud frame

Every slice shall include a GDU that codes the slice geometry and ADUs or defaulted attribute DUs that code the slice attributes. A slice is identified by the GDU *slice_id*.

The slice geometry is coded in the slice's coordinate system. The bounding boxes of slices may intersect, including within a single frame.

A slice shall start with a GDU. This GDU may be followed by optional redundant GDUs that duplicate the slice geometry. ADUs and defaulted attribute DUs shall occur after all GDUs in the slice. DUs belonging to different slices shall not be interleaved.

Within a slice, other DUs may be present. For example, an APS can occur within a slice to convey parameters for attribute decoding.

It is a requirement of bitstream conformance that:

- All GDUs present in a slice shall reconstruct the same geometry in the same canonical point order.
- Every slice shall have a corresponding ADU or defaulted attribute DU for every attribute enumerated in the SPS.
- All ADUs present in a slice with the same value of *adu_sps_attr_idx* shall reconstruct the same attribute values.

Only one GDU in a slice shall be decoded; all others shall be ignored when decoding (removed from the bitstream and discarded). A decoder shall choose which GDU is decoded.

ADU parsing depends upon certain GDU header parameters. ADU decoding depends upon the reconstructed slice geometry.

Slices are either independent or dependent. An independent slice does not require any other slice to be decoded first. A dependent slice requires that the immediately preceding slice in bitstream order is decoded first. A slice shall be directly depended upon by no more than a single dependent slice.

A dependent slice shall not depend upon a slice in a different point cloud frame.

6.4.6 Repetition of slices

Slices may be repeated within a coded point cloud frame. Repetition shall not change the value of *slice_id*.

A slice set is the set of slices with the same value of *slice_id* within a coded point cloud frame.

It is a requirement of bitstream conformance that all slices in each slice set shall reconstruct the same points in the same canonical order.

From each slice set, only one slice shall be decoded; all others shall be ignored for decoding (removed from the bitstream and discarded). A decoder shall choose which slice is decoded.

6.4.7 Relationship between tiles and slices

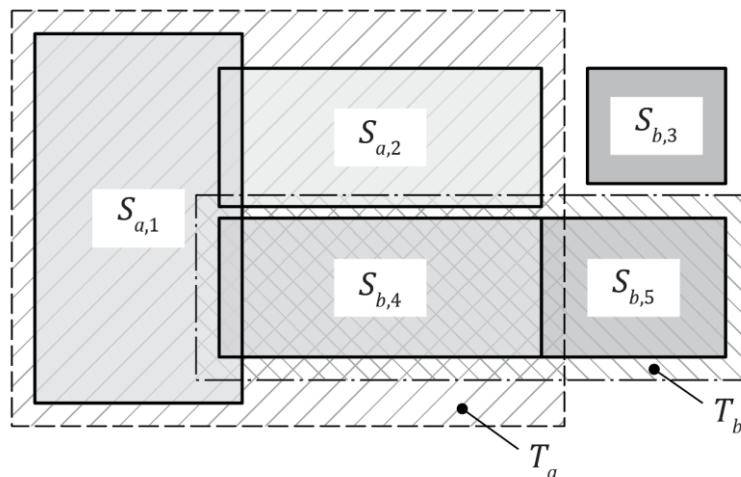
A group of slices can be identified by a common slice tag identifier (**slice_tag**).

The tile inventory DU provides a means to associate a bounding box with a group of slices. Each tile comprises a single bounding box and an identifier (**tileId**). Tile bounding boxes may overlap. Implementations can use a tile inventory to aid spatial random access.

When a tile inventory is present in the bitstream, **slice_tag** shall identify a tile by its **tileId**. Otherwise, the use of slice tags is application specific.

When a slice tag identifies a tile, a dependent slice should not depend upon a slice in a different tile. To do otherwise can prevent decoding of individual tiles (for example, in spatial random access decoding).

Tile information is not used by the decoding processes specified in this document.



Key

$S_{t,n}$	Slice n , associated with tile t
T_t	Bounding box of tile t

Figure 2 — Example arrangement of tiles and slices.

An example arrangement of tiles and slices within a coded point cloud frame is shown in Figure 2. Slices $S_{a,1}$ and $S_{a,2}$ are associated with tile T_a and slices $S_{b,3}$, $S_{b,4}$ and $S_{b,5}$ are associated with tile T_b ; the bounding box of T_b does not include $S_{b,3}$. A decoder that performs spatial random access to decode a region $\mathcal{R}$ (not shown) can use the tile inventory to determine tile IDs for the set of tiles that intersect $\mathcal{R}$. Only slices with matching slice tags would need to be decoded. Since the slice $S_{b,3}$ is not included in the bounding box of tile T_b , if $\mathcal{R}$ intersects $S_{b,3}$ but not T_b , the slice is not discoverable using the tile inventory. However, in the case that $\mathcal{R}$ and T_b intersect, $S_{b,3}$ would have a matching slice tag.

6.4.8 Parameter sets

6.4.8.1 Activation of parameter sets

The parameters contained in an SPS, GPS or APS shall not have any effect until the activation of the respective parameter set.

At most one SPS, GPS and APS are active at any given moment during the decoding process. The activation of a parameter-set shall deactivate any previously active parameter set of the same type.

At the start of a coded point cloud sequence, no parameter sets are active.

An SPS shall be activated by the parsing of a GDU. Once activated, it shall remain active for the whole of the coded point cloud sequence.

A GPS shall be activated by the parsing of a GDU.

An APS shall be activated by the parsing of an ADU.

NOTE Other DUs that contain references to SPS, GPS or APS DUs do not cause the referenced parameter-set to be activated.

6.4.8.2 Order of parameter sets

DUs shall be conveyed to a decoder in an order such that any parameter-set to be activated is available prior to the point of activation.

6.4.8.3 Duplication of parameter sets

Parameter-set DUs may be repeated at any point in the coded point cloud sequence.

All parameter-set DUs with the same parameter-set identifier shall be identical for the duration of the coded point cloud sequence.

NOTE Parameter-set identifiers are distinct for each type of parameter set.

6.5 Output point cloud format

6.5.1 General

Point cloud frames decoded from an L3C2 bitstream shall be output in the output point cloud format (6.5).

6.5.2 Coordinate system

A decoder shall output points in the sequence coordinate system.

The output point cloud format shall indicate the sequence origin *SeqOrigin* and the sequence unit *SeqUnit* as point cloud metadata.

6.5.3 Fixed-point conformance output

A decoder that is configured to output *n*-fractional-bit fixed-point positions shall round half-values of P_{SCS} away from zero prior to output as P_{OUT} :

$$P_{OUT} = \text{RoundFz}(P_{SCS} \times \text{SeqCodedScale} \times 2^n) \div 2^n$$

6.5.4 Attributes

Attribute values shall be interpreted according to the semantics of the attribute type and any per-sequence or frame-specific attribute properties. For example, if a frame-specific scale and offset property is present for an attribute, the output attribute values for that frame would be interpreted according to 7.4.2.2.5.

6.5.5 Output point cloud sequence

Decoding a conforming L3C2 bitstream generates a sequence of output point cloud frames.

6.5.6 Output point cloud frame

Each output point cloud frame is specified in terms of the following state variables:

- The variable *RecCloudPointCnt*, the cumulative number of points in the output point cloud frame.
- The array *RecCloudPos* of decoded point positions; *RecCloudPos*[*ptIdx*][*k*] is the *k*-th coordinate of the *ptIdx*-th output point in the coding coordinate system.
- The array *RecCloudAttr* of decoded point attributes; *RecCloudAttr*[*ptIdx*][*attrIdx*][*c*] is the *c*-th component of the identified attribute for the *ptIdx*-th point. Attributes are identified by the index *attrIdx* into the active SPS attribute list.

Decoder implementations may output points in a different order to the canonical order specified by this document.

Immediately prior to outputting the decoded point cloud frame, point positions shall be converted to the sequence coordinate system.

7 Syntax and semantics

7.1 Method of specifying syntax in tabular form

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

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

	Descriptor
/* A statement can be a syntax element with an associated descriptor or can be a statement used to specify conditions for the existence, type and quantity of syntax elements, as in the following two examples */	
syntax_element	ue(v)
statement	
/* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. */	
{	
statement	
statement	
...	
}	
/* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true */	
while(condition)	
statement	

/* A "do ... while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true */	
do {	
statement	
} while(condition)	
/* An "if ... else" structure specifies a test of whether a condition is true, and if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed */	
if(condition)	
primary statement	
else	
alternative statement	
/* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */	
for(initial statement; condition; subsequent statement)	
primary statement	

7.2 Specification of syntax functions and descriptors

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

`byte_aligned()` is specified as:

- If the next bit in the bitstream is the first bit in a byte ($\text{DataUnitReadIdx} \% 8 == 0$), the value of `byte_aligned()` is true.
- Otherwise, the value of `byte_aligned()` is false.

`more_data_in_data_unit()` is specified as:

- If parsing of the DU is incomplete ($\text{DataUnitReadIdx} / 8 < \text{DataUnitLength}$), the value of `more_data_in_data_unit()` is true.
- Otherwise, the value of `more_data_in_data_unit()` is false.

`Length(x)` is the length in bits of the coded syntax element *x* as measured by the change in *DataUnitReadIdx* between the start and end of the syntax element.

The following descriptors specify the parsing process of every syntax element. The parsing processes are specified in Clause 11.

- `ae(v)`: adaptive arithmetic entropy-coded syntax element.
- `de(v)`: dictionary coded syntax element.
- `oid(v)`: an ASN.1 object identifier.
- `s(n)`: signed integer using an *n*-bit magnitude and a sign bit.

- $se(v)$: signed integer 0-th order Exp-Golomb-coded syntax element.
- $u(n)$: unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent upon the value of other syntax elements.
- $ue(v)$: unsigned integer 0-th order Exp-Golomb-coded syntax element.

7.3 Syntax in tabular form

7.3.1 General

The syntax structures and the syntax elements within these structures are specified in 7.3.2. Any values that are not specified in the tables shall not be present in the bitstream unless otherwise specified in this document.

7.3.2 Parameter sets, ancillary data and byte alignment

7.3.2.1 Sequence parameter set data unit syntax

<code>seq_parameter_set() {</code>	Descriptor
<code>main_profile_compliant</code>	$u(1)$
<code>reserved_profile_21bits</code>	$u(21)$
<code>slice_reordering_constraint</code>	$u(1)$
<code>unique_point_positions_constraint</code>	$u(1)$
<code>level_idc</code>	$u(8)$
<code>sps_seq_parameter_set_id</code>	$u(4)$
<code>frame_ctr_lsb_bits</code>	$u(5)$
<code>slice_tag_bits</code>	$u(5)$
<code>seq_sensing_coverage_enabled</code>	$u(1)$
<code>seq_origin_bits</code>	$ue(v)$
<code>if(seq_origin_bits) {</code>	
<code>for($k = 0$; $k < 3$; $k++$)</code>	
<code>seq_origin_xyz[k]</code>	$s(v)$
<code>seq_origin_log2_scale</code>	$ue(v)$
<code>}</code>	
<code>seq_bbox_size_bits</code>	$ue(v)$
<code>if(seq_bbox_size_bits)</code>	
<code>for($k = 0$; $k < 3$; $k++$)</code>	
<code>seq_bbox_size_minus1_xyz[k]</code>	$u(v)$
<code>seq_unit_numerator_minus1</code>	$ue(v)$
<code>seq_unit_denominator_minus1</code>	$ue(v)$
<code>seq_unit_is_metres</code>	$u(1)$
<code>seq_coded_scale_exponent</code>	$ue(v)$
<code>seq_coded_scale_mantissa_bits</code>	$ue(v)$
<code>seq_coded_scale_mantissa</code>	$u(v)$
<code>num_attributes</code>	$ue(v)$
<code>for($attrIdx = 0$; $attrIdx < num_attributes$; $attrIdx++$) {</code>	
<code>attr_components_minus1[$attrIdx$]</code>	$ue(v)$
<code>attr_instance_id[$attrIdx$]</code>	$ue(v)$

attr_bitdepth_minus1 [<i>attrIdx</i>]	ue(v)
attr_label_known [<i>attrIdx</i>]	u(1)
if(<i>attr_label_known</i> [<i>attrIdx</i>])	
attr_label [<i>attrIdx</i>]	ue(v)
else	
attr_label_oid [<i>attrIdx</i>]	oid(v)
attr_property_cnt	ue(v)
byte_alignment()	
for(<i>j</i> = 0; <i>j</i> < <i>attr_property_cnt</i> ; <i>j</i> ++)	
attribute_property (<i>attrIdx</i>)	
}	
geom_axis_order	u(3)
bypass_stream_enabled	u(1)
entropy_continuation_enabled	u(1)
if(<i>entropy_continuation_enabled</i>)	
per_beam_coding_state_preserved_enabled	u(1)
sps_extension_present	u(1)
if(<i>sps_extension_present</i>)	
while(<i>more_data_in_data_unit</i> ())	
sps_extension_data	u(1)
byte_alignment()	
}	

7.3.2.2 Attribute property syntax

attribute_property(<i>attrIdx</i>) {	Descriptor
attr_prop_type	u(8)
attr_prop_len	u(8)
<i>AttrPropDataLen</i> = <i>attr_prop_len</i>	
if(<i>attr_prop_type</i> == 0) {	
attr_prop_itu_t_t35_country_code	u(8)
<i>AttrPropDataLen</i> --	
if(<i>attr_prop_itu_t_t35_country_code</i> == 255) {	
attr_prop_itu_t_t35_country_code_extension_byte	u(8)
<i>AttrPropDataLen</i> --	
}	
attribute_property_data (<i>attrIdx</i> , <i>AttrPropDataLen</i>)	
} else if(<i>attr_prop_type</i> == 1) {	
attr_prop_oid	oid(v)
<i>AttrPropDataLen</i> -= Length(<i>attr_prop_oid</i>) / 8	
attribute_property_data (<i>attrIdx</i> , <i>AttrPropDataLen</i>)	
} else if(<i>attr_prop_type</i> == 2) {	
attr_cicp_colour primaries [<i>attrIdx</i>]	ue(v)
attr_cicp_transfer_characteristics [<i>attrIdx</i>]	ue(v)

<code>attr_cicp_matrix_coeffs[attrIdx]</code>	<code>ue(v)</code>
<code>attr_cicp_video_full_range[attrIdx]</code>	<code>u(1)</code>
<code>} else if(attr_prop_type == 3) {</code>	
<code>attr_offset_bits</code>	<code>ue(v)</code>
<code>attr_offset[attrIdx]</code>	<code>s(v)</code>
<code>attr_scale_bits</code>	<code>ue(v)</code>
<code>attr_scale_minus1[attrIdx]</code>	<code>u(v)</code>
<code>attr_frac_bits[attrIdx]</code>	<code>ue(v)</code>
<code>} else if(attr_prop_type == 4) {</code>	
<code>for(c = 0; c ≤ attr_components_minus1[attrIdx]; c++)</code>	
<code>attr_default_value[attrIdx][c]</code>	<code>u(v)</code>
<code>} else</code>	
<code>attribute_property_data(attrIdx, attr_prop_len)</code>	
<code>byte_alignment()</code>	
<code>}</code>	

7.3.2.3 Attribute property data syntax

<code>attribute_property_data(attrIdx, numBytes) {</code>	Descriptor
<code>for(i = 0; i < numBytes; i++)</code>	
<code>attr_prop_byte[i]</code>	<code>u(8)</code>
<code>}</code>	

7.3.2.4 Tile inventory data unit syntax

<code>tile_inventory() {</code>	Descriptor
<code>ti_seq_parameter_set_id</code>	<code>u(4)</code>
<code>ti_frame_ctr_lsb_bits</code>	<code>u(5)</code>
<code>ti_frame_ctr_lsb</code>	<code>u(v)</code>
<code>tile_cnt</code>	<code>u(16)</code>
<code>if(tile_cnt > 0) {</code>	
<code>tile_id_bits</code>	<code>u(5)</code>
<code>tile_origin_bits_minus1</code>	<code>u(8)</code>
<code>tile_size_bits_minus1</code>	<code>u(8)</code>
<code>for(tileIdx = 0; tileIdx < tile_cnt; tileIdx++) {</code>	
<code>tile_id[tileIdx]</code>	<code>u(v)</code>
<code>tileId = tile_id_bits ? tile_id[tileIdx] : tileIdx</code>	
<code>for(k = 0; k < 3; k++)</code>	
<code>tile_origin_xyz[tileId][k]</code>	<code>s(v)</code>
<code>for(k = 0; k < 3; k++)</code>	
<code>tile_size_minus1_xyz[tileId][k]</code>	<code>u(v)</code>
<code>}</code>	
<code>ti_origin_bits_minus1</code>	<code>ue(v)</code>
<code>for(k = 0; k < 3; k++)</code>	
<code>ti_origin_xyz[k]</code>	<code>s(v)</code>

ti_origin_log2_scale	ue(v)
}	
byte_alignment()	
}	

7.3.2.5 Geometry parameter set data unit syntax

geometry_parameter_set() {	Descriptor
gps_geom_parameter_set_id	u(4)
gps_seq_parameter_set_id	u(4)
geom_sensing_coverage_enabled	u(1)
slice_geom_origin_scale_present	u(1)
if($\neg$ slice_geom_origin_scale_present)	
gps_geom_origin_log2_scale	ue(v)
geom_dup_points_enabled	u(1)
slice_angular_origin_present	u(1)
if($\neg$ slice_angular_origin_present) {	
gps_angular_origin_bits_minus1	ue(v)
for($k = 0; k < 3; k++$)	
gps_angular_origin_xyz[k]	s(v)
}	
num_heads	ue(v)
for($h = 0; h < \text{num_heads}; h++$) {	
num_beams_minus1[h]	ue(v)
beam_steps_per_rotation_minus1[h]	ue(v)
beam_elevation_init[h]	se(v)
beam_voffset_init[h]	se(v)
for($b = 1; b \leq \text{num_beams_minus1}[h]; b++$) {	
beam_elevation_diff[h][b]	se(v)
beam_voffset_diff[h][b]	se(v)
}	
}	
geom_use_num_azimuth_steps_per_quant_step	u(1)
geom_use_coarse_neighbours	u(1)
geom_use_vertical_prediction	u(1)
geom_unordered_points	u(1)
geom_quantization_step_size	u(14)
geom_quantization_scaling_log2	u(4)
geom_minimum_azimuth_quantization_step_size	u(25)
geom_low_latency_attributes	u(1)
if(geom_low_latency_attributes) {	
low_latency_attributes_qp	u(8)
low_latency_attributes_bit_depth	u(5)
}	

gps_extension_present	u(1)
if(gps_extension_present)	
while(more_data_in_data_unit())	
gps_extension_data	u(1)
byte_alignment()	
}	

7.3.2.6 Attribute parameter set data unit syntax

attribute_parameter_set() {	Descriptor
aps_attr_parameter_set_id	u(4)
aps_seq_parameter_set_id	u(4)
attr_coding_type	ue(v)
attr_primary_qp_minus4	ue(v)
attr_secondary_qp_offset	se(v)
attr_qp_offsets_present	u(1)
if(attr_coding_type == 0) {	
raht_prediction_enabled	u(1)
if(raht_prediction_enabled) {	
raht_prediction_subtree_min	ue(v)
raht_prediction_samples_min	ue(v)
}	
} else if(attr_coding_type ≤ 2) {	
pred_set_size_minus1	ue(v)
pred_inter_lod_search_range	ue(v)
for(<i>k</i> = 0; <i>k</i> < 3; <i>k</i> ++)	
pred_dist_bias_minus1_xyz[<i>k</i>]	ue(v)
if (attr_coding_type == 2)	
last_comp_pred_enabled	u(1)
lod_scalability_enabled	u(1)
if(lod_scalability_enabled)	
pred_max_range_minus1	ue(v)
else {	
lod_max_levels_minus1	ue(v)
if(¬lod_max_levels_minus1)	
attr_canonical_order_enabled	u(1)
else {	
lod_decimation_mode	ue(v)
if(lod_decimation_mode > 0)	
for(<i>lvl</i> = 0; <i>lvl</i> < lod_max_levels_minus1; <i>lvl</i> ++)	
lod_sampling_period_minus2[<i>lvl</i>]	ue(v)
lod_initial_dist_log2	ue(v)
lod_dist_log2_offset_present	u(1)
}	

}	
if(attr_coding_type == 1) {	
pred_direct_max_idx_plus1	ue(v)
if(pred_direct_max_idx_plus1) {	
pred_direct_threshold	u(8)
pred_direct_avg_disabled	u(1)
}	
pred_intra_lod_search_range	ue(v)
if(pred_intra_lod_search_range)	
pred_intra_min_lod	ue(v)
inter_comp_pred_enabled	u(1)
pred_blending_enabled	u(1)
}	
} else if(attr_coding_type == 3)	
raw_attr_width_present	u(1)
if(!lod_scalability_enabled)	
attr_coord_conv_enabled	u(1)
if(attr_coord_conv_enabled)	
for(k = 0; k < 3; k++) {	
attr_coord_conv_scale_bits_minus1[k]	u(5)
attr_coord_conv_scale[k]	u(v)
}	
aps_extension_present	u(1)
if(aps_extension_present)	
while(more_data_in_data_unit())	
aps_extension_data	u(1)
byte_alignment()	
}	

7.3.2.7 Frame-specific attribute properties data unit syntax

frame_specific_attribute_properties() {	Descriptor
fsap_seq_parameter_set_id	u(4)
fsap_frame_ctr_lsb_bits	u(5)
fsap_frame_ctr_lsb	u(v)
fsap_sps_attr_idx	ue(v)
fsap_num_props	ue(v)
byte_alignment()	
for(i = 0; i < fsap_num_props; i++)	
attribute_property(fsap_sps_attr_idx)	
}	

7.3.2.8 Frame boundary marker data unit syntax

frame_boundary_marker() {	Descriptor
fbdu_frame_ctr_lsb_bits	u(5)
fbdu_frame_ctr_lsb	u(v)
byte_alignment()	
}	

7.3.2.9 User data data unit syntax

userdata_data_unit() {	Descriptor
user_data_oid	oid(v)
while(more_data_in_data_unit())	
user_data_byte	u(8)
}	

7.3.2.10 Byte alignment syntax

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

7.3.3 Geometry data unit

7.3.3.1 Geometry data unit syntax

geometry_data_unit() {	Descriptor
geometry_data_unit_header()	
sensed_point_sequences()	
geometry_data_unit_footer()	
}	

7.3.3.2 Geometry data unit header syntax

geometry_data_unit_header() {	Descriptor	Semantics
gdu_geometry_parameter_set_id	u(4)	7.4.3.2
gdu_reserved_zero_3bits	u(3)	7.4.3.2
slice_id	ue(v)	7.4.3.2
slice_tag	u(v)	7.4.3.2
frame_ctr_lsb	u(v)	7.4.3.2

if(seq_sensing_coverage_enabled && geom_sensing_coverage_enabled) {		
sensing_coverage_present	u(1)	7.4.3.2
if(sensing_coverage_present) {		
sensing_coverage_may_have_missing_points	u(1)	7.4.3.2
for(<i>h</i> = 0; <i>h</i> < num_heads; <i>h</i> ++) {		
num_sensing_ranges_minus1_bits [<i>h</i>]	ue(<i>v</i>)	7.4.3.2
num_sensing_ranges_minus1 [<i>h</i>]	u(<i>v</i>)	7.4.3.2
sensing_range0_start_bits [<i>h</i>]	ue(<i>v</i>)	7.4.3.2
sensing_range_length_minus1_bits [<i>h</i>]	ue(<i>v</i>)	7.4.3.2
if(num_sensing_ranges_minus1[<i>h</i>])		
sensing_range_start_offset_minus1_bits [<i>h</i>]	ue(<i>v</i>)	7.4.3.2
sensing_range0_start [<i>h</i>]	u(<i>v</i>)	7.4.3.2
sensing_range0_length_minus1 [<i>h</i>]	u(<i>v</i>)	7.4.3.2
for(<i>i</i> = 0; <i>i</i> < num_sensing_ranges_minus1[<i>h</i>]; <i>i</i> ++) {		
sensing_range_start_offset_minus1 [<i>h</i>][<i>i</i>]	s(<i>v</i>)	7.4.3.2
sensing_range_length_minus1 [<i>h</i>][<i>i</i>]	u(<i>v</i>)	7.4.3.2
}		
}		
}		
}		
if(entropy_continuation_enabled) {		
slice_entropy_continuation	u(1)	7.4.3.2
if(slice_entropy_continuation) {		
prev_slice_id	ue(<i>v</i>)	7.4.3.2
if(per_sensor_coding_state_preserved_enabled)		
per_sensor_coding_state_preserved	u(1)	7.4.3.2
}		
}		
if(slice_geom_origin_scale_present)		
slice_geom_origin_log2_scale	ue(<i>v</i>)	7.4.3.2
slice_geom_origin_bits_minus1	ue(<i>v</i>)	7.4.3.2
for(<i>k</i> = 0; <i>k</i> < 3; <i>k</i> ++)		
slice_geom_origin_xyz [<i>k</i>]	u(<i>v</i>)	7.4.3.2
if(slice_angular_origin_present) {		
slice_angular_origin_bits_minus1	ue(<i>v</i>)	7.4.3.2
for(<i>k</i> = 0; <i>k</i> < 3; <i>k</i> ++)		
slice_angular_origin_xyz [<i>k</i>]	s(<i>v</i>)	7.4.3.2
}		
stream_cnt_minus1	ue(<i>v</i>)	7.4.3.2
if(stream_cnt_minus1) {		
stream_len_bits	u(6)	7.4.3.2
for(<i>i</i> = 0; <i>i</i> < stream_cnt_minus1; <i>i</i> ++)		
stream_len [<i>i</i>]	u(<i>v</i>)	7.4.3.2

}		
byte_alignment()		
}		

7.3.3.3 Geometry data unit footer syntax

geometry_data_unit_footer() {	Descriptor	Semantics
byte_alignment()		
slice_num_points_minus1	u(24)	7.4.3.3
}		

7.3.3.4 Sensed point sequences syntax

sensed_point_sequences() {	Descriptor	Semantics
PtnCnt = 0		
for(HeadIdx = 0; HeadIdx < num_heads; HeadIdx++) {		
num_points[HeadIdx]	u(20)	9.2.3.1
for(PtnIdx = 0; PtnIdx < num_points[HeadIdx]; PtnIdx++) {		
sensed_point()		
PtnCnt = PtnCnt + 1		
}		
end_of_entropy_stream	ae(v)	9.2.3.1
}		
}		

7.3.3.5 Sensed point syntax

sensed_point() {	Descriptor	Semantics
if(PtnIdx == 0 && !(per_sensor_coding_state_preserved && SensingHeadStartedCapturing[HeadIdx])) {		
azimuth_index_start[HeadIdx]	u(14)	9.2.3.2
elevation_index_start[HeadIdx]	u(8)	9.2.3.2
} else if(!NumCoarsePositionDuplicatesNext[HeadIdx][PtnIdx - 1])		
coarse_position()		
if(PtnIdx == 0 !NumCoarsePositionDuplicatesNext[HeadIdx][PtnIdx - 1])		
num_coarse_position_duplicates[HeadIdx][PtnIdx]	ae(v)	9.2.3.2
else if(geom_dup_point_enabled)		
dup_point[HeadIdx][PtnIdx]	ae(v)	9.2.3.2
if(!dup_point)		
point_position()		
if(geom_low_latency_attributes)		
point_low_latency_attributes()		
}		

7.3.3.6 Coarse position syntax

coarse_position() {	Descriptor	Semantics
<i>AzimuthIdx</i> = SensingHeadLastAzimuthIdx[HeadIdx]		
<i>ElevationIdx</i> = SensingHeadLastElevationIdx[HeadIdx] + 1		
if(geom_unordered_points)		
late_point[HeadIdx][PtnIdx]	ae(v)	9.2.3.3
if(late_point[HeadIdx][PtnIdx])		
<i>AzimuthIdx</i> --, <i>ElevationIdx</i> --		
<i>next</i> = 1		
while(<i>next</i>) {		
elevation_offset	ae(v)	9.2.3.3
<i>ElevationIdx</i> += elevation_offset		
if(<i>ElevationIdx</i> == num_beams_minus1[HeadIdx] + 1) {		
azimuth_offset	ae(v)	9.2.3.3
<i>AzimuthIdx</i> += AzimuthIdxOffset[AzimuthIdx][azimuth_offset]		
<i>ElevationIdx</i> = 0		
}		
else		
<i>next</i> = 0		
}		
}		

7.3.3.7 Point position syntax

point_position() {	Descriptor	Semantics
pred_idx[HeadIdx][PtnIdx]	ae(v)	9.2.3.4
if(geom_use_vertical_prediction && pred_idx[HeadIdx][PtnIdx] == 1 && VerticalPredEligible)		
vertical_radius_pred[HeadIdx][PtnIdx]	ae(v)	9.2.3.4
radius_res[HeadIdx][PtnIdx]	ae(v)	9.2.3.4
azimuth_res[HeadIdx][PtnIdx]	ae(v)	9.2.3.4
if(geom_quantization_step_size == 64) {		
x_res[HeadIdx][PtnIdx]	ae(v)	9.2.3.4
y_res[HeadIdx][PtnIdx]	ae(v)	9.2.3.4
}		
z_res[HeadIdx][PtnIdx]	ae(v)	9.2.3.4
}		