



DPC: 25/30458564 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: 11 March 2025

Origin: International

Latest date for receipt of comments: 04 May 2025

Project No. 2022/01560

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

Interested committees:

Title: Draft BS ISO/IEC 23090-29 Information technology. Coded representation of immersive media. Part 29: Video-based dynamic mesh coding (V-DMC).

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 04 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/>.

E-mail: 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 23090-29

Information technology — Coded representation of immersive media —

Part 29: Video-based dynamic mesh coding (V-DMC)

ICS: 35.040.40

ISO/IEC JTC 1/SC 29

Secretariat: **JISC**

Voting begins on:
2025-03-11

Voting terminates on:
2025-06-03

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 23090-29:2025(en)

Full document details: 25/30458564 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

Page

Foreword	vi
Introduction	vii
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Abbreviated terms	3
5 Conventions	3
5.1 General	3
5.2 Arithmetic operators	3
5.3 Logical operators	3
5.4 Relational operators	3
5.5 Bit-wise operators	3
5.6 Assignment operators	3
5.7 Other operators	3
5.8 Mathematical functions	4
5.9 Order of operation precedence	7
5.10 Variables, syntax elements, and tables	7
5.11 Text description of logical operations	7
5.12 Processes	7
5.13 Array operations	7
6 Overall V-DMC characteristics, decoding operations, and post-decoding processes	8
6.1 V-DMC characteristics	8
6.2 V-DMC bitstream characteristics, decoding operations, and post-decoding processes	10
7 Bitstream format, partitioning, and scanning processes	11
7.1 General	11
7.2 V3C bitstream formats	11
7.3 NAL bitstream formats	11
7.4 Partitioning of atlas frames into tiles	11
7.5 Tile partition scanning processes	11
7.6 Extension of partitioning process for tile attribute components	11
8 Syntax and semantics	12
8.1 Method of specifying syntax in tabular form	12
8.2 Specification of syntax functions and descriptors	12
8.3 Syntax in tabular form	12
8.3.1 General	12
8.3.2 V3C unit syntax	12
8.3.3 Byte alignment syntax	12
8.3.4 V3C parameter set syntax	12
8.3.5 NAL unit syntax	14
8.3.6 Raw byte sequence payloads, trailing bits, and byte alignment syntax	14
8.3.7 Atlas tile data unit syntax	23
8.3.8 Supplemental enhancement information message syntax	23
8.3.9 VDMC atlas tile data unit syntax	23
8.4 Semantics	33
8.4.1 General semantics	33
8.4.2 V3C unit semantics	33
8.4.3 Byte alignment semantics	33
8.4.4 V3C parameter set semantics	33
8.4.5 NAL unit semantics	37
8.4.6 Raw byte sequence payloads, trailing bits, and byte alignment semantics	37
8.4.7 Atlas tile data unit semantics	52
8.4.8 Supplemental enhancement information message semantics	52

8.4.9	VDMC atlas tile data unit semantics	53
9	Decoding process	66
9.1	General decoding process	66
9.2	Atlas data decoding process	66
9.2.1	General atlas decoding process	66
9.2.2	Decoding process for a coded atlas frame	67
9.2.3	Atlas NAL unit decoding process	67
9.2.4	Atlas tile header decoding process	67
9.2.5	Decoding process for patch data units	69
9.2.6	Decoding process of the block to patch map	69
9.2.7	Conversion of tile level patch information to atlas level patch information	69
9.2.8	Decoding process for meshpatch data units	69
9.2.9	Conversion of tile level meshpatch information to atlas level meshpatch information	89
9.3	Occupancy video decoding process	91
9.4	Geometry video decoding process	91
9.5	Attribute video decoding process	92
9.6	Packed video decoding process	92
9.7	Common atlas decoding process	92
9.8	Sub-bitstream extraction process	92
9.9	Basemesh decoding process	92
10	Pre-reconstruction process	93
11	Reconstruction process	93
11.1	General	93
11.2	Submesh Reconstruction Process	96
11.2.1	General	96
11.2.2	Texture coordinate derivation	97
11.2.3	Subdivision process	106
11.2.4	Inverse image packing of transform coefficients	123
11.2.5	Inverse quantization	125
11.2.6	Inverse transform	127
11.2.7	Normal, tangent, and bitangent vector generation	129
11.2.8	Vertex coordinates reconstruction	132
11.2.9	Texture coordinate adaptation process	132
11.3	Append submesh to a mesh process	133
12	Post-reconstruction process	134
12.1	General	134
12.2	Zippering process	135
12.2.1	Zippering method by distance	135
12.2.2	Zippering method by index matching	136
12.2.3	Zippering method by boundary connect	136
12.2.4	Identification of boundary vertices	147
12.2.5	Identification of matched boundary vertices by distance	148
12.2.6	Identification of matched boundary vertices by explicit signaling	150
12.2.7	Border vertex fusion	151
12.2.8	Handling of unmatched LoD vertices	153
13	Adaptation process	159
14	Parsing process	160
Annex A (normative)	Profiles, tiers, and levels	161
Annex B (informative)	Post-decoding conversion to nominal video formats	169
Annex C (informative)	V3C sample stream format	178
Annex D (normative)	NAL sample stream format	179
Annex E (normative)	Atlas hypothetical reference decoder	180

Annex F (normative) Supplemental enhancement information	181
Annex G (informative) Volumetric usability information	194
Annex H (normative) Basemesh sub-bistream	196
Annex I (Normative) MPEG EdgeBreaker Static Mesh Coding	285
Annex J (Normative) Arithmetic coded displacement sub-bitstream.....	364
Annex K (Normative) Inverse Binarization Process	422
Annex L (Normative) MPEG Motion Coding	433
References	441

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

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

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

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

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

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

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

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

Introduction

The advances in 3D capture, modelling, and rendering have promoted the ubiquitous presence of 3D content across several platforms and devices.

3D meshes, along with point clouds and immersive video, are widely used to represent such immersive content. A mesh is composed of several polygons that describe the boundary surface of a volumetric object. Each polygon is defined by its vertices in 3D space and the information on how the vertices are connected, referred to as connectivity information. Optionally, vertex attributes, such as colours, normal, etc., could be associated with the mesh vertices. Attributes could also be associated with the surface of the mesh by exploiting mapping information that describes a parameterization of the mesh onto 2D regions of the plane. Such mapping is usually described by a set of parametric coordinates, referred to as UV coordinates or texture coordinates. 2D attribute maps are used to store high resolution attribute information such as texture, normal, material ID etc. Such information could be used for various purposes such as texture mapping and shading.

To achieve a realistic representation of an object or scene, 3D meshes are becoming increasingly sophisticated, and a significant amount of data is generated for the creation and consumption of those meshes. When uncompressed, these data can be costly in terms of storage and transmission. Furthermore, dynamic mesh sequences may require even larger amounts of data, since it may consist of a significant quantity of information changing over time. A visual volumetric video-based coding (V3C) standard has been created to support the distribution of volumetric data, currently addressing point clouds (ISO/IEC 23090-5) and immersive video content (ISO/IEC 23090-12). Therefore, this document was developed to support compression of 3D video-based mesh coding utilizing the ISO/IEC V3C standard (ISO/IEC 23090-5).

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

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

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

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

Information technology — Coded representation of immersive media —

Part 29: Video-based dynamic mesh coding (V-DMC)

1 Scope

This document specifies syntax, semantics, and decoding for video based dynamic mesh coding (V-DMC), basemesh coding, MPEG edgebreaker static mesh coding, and arithmetic coded displacement. 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 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.

ISO /IEC 23090-5, (4E)¹⁾, *Information technology — Coded Representation of Immersive Media — Part 5: Visual Volumetric Video-based Coding (V3C) and Video-based Point Cloud Compression (V-PCC)*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 23090-5(4E):2025 and the following 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

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 and a list of metadata corresponding to a part of a surface of a mesh in 3D space

3.2

atlas frame

2D rectangular array of atlas samples onto which *patches* (ISO/IEC 23090-5(4E):2025:3.91) are projected and additional information related to the *patches* (ISO/IEC 23090-5(4E):2025:3.91), corresponding to a *volumetric frame* (ISO/IEC 23090-5(4E):2025:3.142) and a list of *meshpatches* (3.9) and additional information related to the *meshpatches* (3.9), corresponding to a *volumetric frame* (ISO/IEC 23090-5(4E):2025:3.142)

1) Under preparation. Stage at time of publication: ISO/IEC DIS 23090-5(4E):2025

3.3**atlas sample**

position on the rectangular frame onto which *patches* (ISO/IEC 23090-5(4E):2025:3.91) that are associated with an *atlas* (3.1) are projected or element of a list of *meshpatches* (3.9) that are associated with an *atlas* (3.1)

3.4**attribute frame**

2D rectangular array created through the aggregation of *patches* (ISO/IEC 23090-5(4E):2025:3.91) and *meshpatches* (3.9) containing values of a specific *attribute* (ISO/IEC 23090-5(4E):2025:3.14)

3.5**attribute map**

image for mapping an attribute into a surface of 3D shape

3.6**basemesh frame**

array of *submeshes* (3.13) associated with each *meshpatch* (3.9) corresponding to a volumetric frame

3.7**geometry frame**

2D array created through the aggregation of the *geometry* (ISO/IEC 23090-5(4E):2025:3.64) information associated with each *patch* (ISO/IEC 23090-5(4E):2025:3.91) and *meshpatch* (3.9)

3.8**displacement**

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

3.9**meshpatch**

element of an *atlas* (3.1) associated with basemesh information

3.10**meshpatch data**

data in an *atlas* (3.1) associated with a *meshpatch* (3.9) that enables the conversion of basemesh into the reconstructed mesh

3.11**orthoAtlas**

orthographic projection and packing of the faces into a texture atlas

3.12**subdivision**

process of dividing the mesh faces into a number of sub-faces

3.13**submesh**

independently decodable region of a *basemesh* (ISO/IEC 23090-5(4E):2025:3.22)

3.14**subpatch**

a set of parameters in the *meshpatch* (3.9) to construct a homography transform used to map the vertex coordinates of a mesh surface associated with a given *faceId* to a set of UV texture coordinates and their connectivity indices.

3.15**texture mapping**

process of transferring attributes from a 2D attribute (texture) map into surface of a 3D mesh

3.16**texture tile**

independently decodable rectangular region of an attribute map frame associated with a texture map

4 Abbreviated terms

The following abbreviations apply in addition to the abbreviations in ISO/IEC 23090-5(4E):2025:4.

BMD	Basemesh Data
CPM	Contextual Probability Model
LOD	Level Of Detail
V-DMC	Video-based Dynamic Mesh Coding

5 Conventions

5.1 General

The specifications in ISO/IEC 23090-5(4E):2025:5.1 apply.

5.2 Arithmetic operators

The specifications in ISO/IEC 23090-5(4E):2025:5.2 apply.

5.3 Logical operators

The specifications in ISO/IEC 23090-5(4E):2025:5.3 apply.

5.4 Relational operators

The specifications in ISO/IEC 23090-5(4E):2025:5.4 apply.

5.5 Bit-wise operators

The specifications in ISO/IEC 23090-5(4E):2025:5.5 apply.

5.6 Assignment operators

The specifications in ISO/IEC 23090-5(4E):2025:5.6 apply.

5.7 Other operators

The specifications in ISO/IEC 23090-5(4E):2025:5.7 applies with the following addition.

`p = GetGeometryPatchIdxInAtlas( submeshIdx, lodIdx )` return patch index `p` of a geometry patch in atlas describing submesh with submesh index equal to `submeshIdx` and level of details with level of details index equal to `lodIdx`.

```
p = GetGeometryPatchIdxInAtlas( submeshIdx, lodIdx) {
    submeshId = SubmeshIndexToID[ submeshIdx ]
    for( p = 0; p < AtlasTotalNumMeshpatches; p++ ) {
        if( AtlasFrameSubmeshID[ p ] == submeshId &&
            AtlasFrameLoDIdx[ p ] == lodIdx &&
            AtlasFramePatchType[ p ] == 0 ) {
            return p
        }
    }
    return -1
}
```

`p = GetAttributePatchIdxInAtlas( submeshIdx, attributeIdx )` return patch index `p` of a patch in atlas attribute nominal frame with index equal to `attributeIdx` describing submesh with submesh index equal to `submeshIdx`.

```
p = GetAttributePatchIdxInAtlas( submeshIdx, attributeIdx ) {
    submeshId = SubmeshIndexToID[ submeshIdx ]
    for( p = 0; p < AtlasTotalNumMeshpatches; p++ ) {
        if( AtlasFrameSubmeshID[ p ] == submeshId &&
            AtlasFrameAttributeIndex[ p ] == attributeIdx &&
            AtlasFramePatchType[ p ] == 1 ) {
            return p
        }
    }
    return -1
}
```

`s = GetSubmeshIdxInBaseMesh( basemeshFrame, numSubmeshes, submeshId )` return submesh index `s` of a submesh in base mesh frame `basemeshFrame` with submesh ID equal to `submeshId`. `basemeshFrame` contains number of submeshes equal to `numSubmeshes`.

```
s = GetSubmeshIdxInBaseMesh( basemeshFrame, numSubmes, submeshId ) {
    for( i = 0; i < numSubmes; i++ ) {
        if( basemeshFrame[ i ].submeshId == submeshId ){
            return i
        }
    }
    return -1
}
```

5.8 Mathematical functions

The specifications in ISO/IEC 23090-5(4E):2025:5.8 applies with the following addition.

`Cross( x, y )` cross product function, operating on two vectors `x` and `y`

```
vec = Cross( x, y ) {
    vec[0] = x[ 1 ] * y[ 2 ] - x[ 2 ] * y[ 1 ]
    vec[1] = x[ 2 ] * y[ 0 ] - x[ 0 ] * y[ 2 ]
    vec[2] = x[ 0 ] * y[ 1 ] - x[ 1 ] * y[ 0 ]
    return vec
}

vec = Dot( vec0, vec1 ) {
    out = 0
    for( d = 0; d < 3; d++ ) {
        out = out + vec0[d] * vec1[d]
    }
    return out
}

vec = squareNorm( vec ) {
    return Dot( vec, vec )
}

vec = norm( vec ) {
    return sqrt( squareNorm( vec ) )
}

vec = Normalize( vec ) {
    if( norm( vec ) > 0 )
        vec = vec / norm( vec )
    return vec
}

value = extractOddBits( x ) {
    y = x & 0x55555555
    y = ( y | ( y >> 1 ) ) & 0x33333333
    y = ( y | ( y >> 2 ) ) & 0x0F0F0F0F
    y = ( y | ( y >> 4 ) ) & 0x00FF00FF
    y = ( y | ( y >> 8 ) ) & 0x0000FFFF
}
```

```

    return y
}

value = ComputeMorton2D( i ) {
    x = extractOddBits( i >> 1 )
    y = extractOddBits( i )
    return ( x, y )
}

value = Pow( a, b ) {
    return ab
}

value = CopySign( mag, sgn ) {
    return (sgn >= 0 ) ? +mag : -mag
}

value = IntRecipSqrt( a64 ) {
    k3timesR[ 96 ] = {
        3196059648, 3145728000, 3107979264, 3057647616, 3019898880, 2969567232,
        2931818496, 2894069760, 2868903936, 2831155200, 2793406464, 2768240640,
        2730491904, 2705326080, 2667577344, 2642411520, 2617245696, 2592079872,
        2566914048, 2541748224, 2516582400, 2491416576, 2466250752, 2441084928,
        2428502016, 2403336192, 2378170368, 2365587456, 2340421632, 2327838720,
        2302672896, 2290089984, 2264924160, 2252341248, 2239758336, 2214592512,
        2202009600, 2189426688, 2164260864, 2151677952, 2139095040, 2126512128,
        2113929216, 2101346304, 2088763392, 2076180480, 2051014656, 2038431744,
        2025848832, 2013265920, 2000683008, 1988100096, 1962934272,
        1962934272, 1950351360, 1937768448, 1925185536, 1912602624, 1900019712,
        1900019712, 1887436800, 1874853888, 1862270976, 1849688064, 1849688064,
        1837105152, 1824522240, 1811939328, 1811939328, 1799356416, 1786773504,
        1786773504, 1774190592, 1761607680, 1761607680, 1749024768, 1736441856,
        1736441856, 1723858944, 1723858944, 1711276032, 1698693120, 1698693120,
        1686110208, 1686110208, 1673527296, 1660944384, 1660944384, 1648361472,
        1648361472, 1635778560, 1635778560, 1623195648, 1623195648, 1610612736
    }
    kRcubed[ 96 ] = {
        4195081216, 3999986688, 3857709056, 3673323520, 3538940928, 3364924416,
        3238224896, 3114735616, 3034196992, 2915990528, 2800922624, 2725880832,
        2615890944, 2544223232, 2439185408, 2370818048, 2303728640, 2237913088,
        2173355008, 2110061568, 2048008192, 1987165184, 1927563264, 1869150208,
        1840392192, 1783783424, 1728321536, 1701024768, 1647311872, 1620883456,
        1568898048, 1543306240, 1492993024, 1468236800, 1443762176, 1395656704,
        1372007424, 1348605952, 1302626304, 1280060416, 1257736192, 1235650560,
        1213861888, 1192294400, 1171008512, 1149979648, 1108673536, 1088379904,
        1068352512, 1048567808, 1029031936, 1029036032, 1009729536, 971888640,
        971882496, 953319424, 934993920, 916897792, 899011584, 881389568,
        881392640, 864009216, 846846976, 829900800, 813182976, 813201408,
        796721152, 780459008, 764412928, 764417024, 748601344, 732995584,
        733017088, 717624320, 702468096, 702466048, 687520768, 672786432,
        672787456, 658258944, 658256896, 643947520, 629854208, 629862400,
        615976960, 615952384, 602276864, 588779520, 588804096, 575512576,
        575526912, 562433024, 562439168, 549556224, 549564416, 536876032
    }
}
if( !a64 )
    return 0
shift = -3
while( a64 & 0xffffffff00000000 ) {
    a64 >>= 2
    shift--
}
a = a64
while( !( a & 0xc0000000 ) ) {
    a <<= 2
    shift++
}
idx = ( a >> 25 ) - 32
r = k3timesR[ idx ] - ( ( kRcubed[ idx ] * a ) >> 32 )
ar = ( r * a ) >> 32
s = 0x30000000 - ( ( r * ar ) >> 32 )
r = ( r * s ) >> 32
if( shift > 0 )

```

```

    return r << shift
else
    return r >> - shift
}

value = IntSqrt( x ) {
    if( x <= (1 << 46) )
        return 1 + ( (x * IntRecipSqrt( x ) ) >> 40 )
    else {
        x0 = ( x + 65536 ) >> 16
        return 1 + ( ( x0 * IntRecipSqrt( x0 ) ) >> 32 )
    }
}

value = popcnt( x ) {
    x = x - ( ( x >> 1 ) & 0x55555555u )
    x = ( x & 0x33333333u ) + ( ( x >> 2 ) & 0x33333333u )
    return ( ( x + ( x >> 4 ) & 0xF0F0F0Fu ) * 0x1010101u ) >> 24
}

value = ceilpow2( x ) {
    x--
    x = x | ( x >> 1 )
    x = x | ( x >> 2 )
    x = x | ( x >> 4 )
    x = x | ( x >> 8 )
    x = x | ( x >> 16 )
    return x + 1
}

value = IntLog2( x ) {
    x = ceilpow2( x + 1 ) - 1
    return popcnt( x ) - 1
}

value = RecipApprox( b , log2Scale ){
    log2ScaleOffset = 0
    log2bPlusOne = IntLog2( b ) + 1
    if ( log2bPlusOne > 31 ) {
        b = b >> ( log2bPlusOne - 31 )
        log2ScaleOffset -= log2bPlusOne - 31
    }
    if( log2bPlusOne < 31 ) {
        b = b << ( 31 - log2bPlusOne )
        log2ScaleOffset += 31 - log2bPlusOne
    }
    /* Initial approximation: 48/17 - 32/17 * b with 28 bits decimal precision */
    bRecip = ( ( 0x2d2d2d2d << 31 ) - 0x1e1e1e1e * b ) >> 28;
    for (unsigned i = 0; i < 3; ++i)
        bRecip += bRecip * ( ( 1 << 31 ) - ( b * bRecip >> 31 ) ) >> 31
    log2Scale = ( 31 << 1 ) - log2ScaleOffset
    return bRecip
}

```

The function `CompareCoherence( v, v1, v2 )` computes and compares the dot product between vertex pairs `( v, v1 )`, and `( v, v2 )` and is defined as follows:

```

value = CompareCoherence( v, v1, v2 ) {
    dotProductVV1 = Dot( v, v1 )
    dotProductVV2 = Dot( v, v2 )
    return dotProductVV1 < dotProductVV2 ? 1 : dotProductVV1 == dotProductVV2 ? 2 : 0
}

```

The function `ApplyDirectionalWeights( dispV, dispV1, dispV2, dirScale, predWeights )` is defined to compute and apply weights for `dispV1` and `dispV2` based on the coherence information between them and `dispV`, and the function is as follows:

```

dispVOut, dispV1Out, dispV2Out = ApplyDirectionalWeights( dispV, dispV1, dispV2,
    dirScale, predWeights, updateWeight1, updateWeight2, isPredStep){
    meanGlobal = dirScale[ 0 ] ÷ 100.0
    stdGlobal = dirScale[ 1 ] ÷ 100.0
}

```

```

dispVRec = isPredStep ? dispV : ( dispV + predWeights * ( dispV1 + dispV2 ) )
coherence = compareCoherence( dispVRec, dispV1, dispV2 )
zScore = abs( ( dispVRec[ 0 ] - meanGlobal ) ÷ stdGlobal )
weight = ( zScore < 1 ) ? 0.5 : ( zScore < 2 ) ? 0.75 : 0.95
weightNormalized = pow( ( weight * 0.1 + 0.9 ), isPredStep == 0 ? 2 : 1 ) - 1
if( coherence <= 1 ) {
    if( isPredStep ){
        dispVOut = dispV +
            weightNormalized * predWeights * ( coherence == 1 ? dispV1 : dispV2 )
        dispV1Out = dispV1
        dispV2Out = dispV2
    } else {
        if( coherence == 1 ) {
            dispV1Out = dispV1 - weightNormalized * updateWeight1 * dispV
            dispVOut = dispV
            dispV2Out = dispV2
        } else {
            dispV2Out = dispV2 - weightNormalized * updateWeight2 * dispV
            dispVOut = dispV
            dispV1Out = dispV1
        }
    }
} else {
    dispVOut = dispV
    dispV1Out = dispV1
    dispV2Out = dispV2
}
return dispVOut, dispV1Out, dispV2Out
}

```

5.9 Order of operation precedence

The specifications in ISO/IEC 23090-5(4E):2025:5.9 apply.

5.10 Variables, syntax elements, and tables

The specifications in ISO/IEC 23090-5(4E):2025:5.10 apply with the following addition.

A struct is a composite data type (a set) that may consist of multiple variables, arrays, or even other struct data types. Variables, arrays, and struct data types of a struct *S* are referred to as members of struct *S*. A dot operator is used as a postfix with *A* to denote an existing member. More specifically, a member variable *v* of a struct *S* is denoted as *S.v*. A member array *vArr* of a struct *S* is denoted as *S.vArr*. A member struct *vStruct* of a struct *S* is denoted as *S.vStruct*.

5.11 Text description of logical operations

The specifications in ISO/IEC 23090-5(4E):2025:5.11 apply.

5.12 Processes

The specifications in ISO/IEC 23090-5(4E):2025:5.12 apply.

5.13 Array operations

MaxValueInArray(*A*) returns the maximum value found in the array of values *A*

```

value = FindNeighbour( size, array, w ) {
    for( n = 0; n < size; n++ ) {
        if( array[ n ] == w ) {
            return 1
        }
    }
    return 0
}

index = FindIndexInArray( value, array1D, sizeofArray ) {

```

```

for( n = 0; n < sizeofArray ; n++ ) {
    if( array1D[ n ] == value ) {
        return n
    }
}
return -1
}

neighbours, neighbourCounts = ComputeNeighbours( faceCount, faces) {
    for( t = 0; t < faceCount; t++ ) {
        for( i = 0; i < 3; i++ ) {
            a = faces[ t ][ i ]
            b = faces[ t ][ ( i + 1 ) % 3 ]
            if( FindNeighbour( neighbourCounts[ a ], neighbours[ a ], b ) == 0 ) {
                neighbours[ a ][ neighbourCounts[ a ]++ ] = b
            }
            if( FindNeighbour( neighbourCounts[ b ], neighbours[ b ], a ) == 0 ) {
                neighbours[ b ][ neighbourCounts[ b ]++ ] = a
            }
        }
    }
    return neighbours, neighbourCounts
}

neighbours, neighbourCounts = ComputeTriangleNeighbours( faceCount, faces) {
    for( t = 0; t < faceCount; t++ ) {
        for( i = 0; i < 3; i++ ) {
            a = faces[ t ][ i ]
            b = faces[ t ][ ( i + 1 ) % 3 ]
            if( FindNeighbour( neighbourCounts[ a ], neighbours[ a ], t ) == 0 ) {
                neighbours[ a ][ neighbourCounts[ a ]++ ] = t
            }
            if( FindNeighbour( neighbourCounts[ b ], neighbours[ b ], t ) == 0 ) {
                neighbours[ b ][ neighbourCounts[ b ]++ ] = t
            }
        }
    }
    return neighbours, neighbourCounts
}

edgeTriNeighbours, edgeTriNeighbourCount =
    ComputeTriangleNeighboursOfEdge( vertex0, vertex1, triNeighbours, triNeighbourCounts){
    for( tadj0 = 0; tadj0 < triNeighbourCounts[ vertex0 ]; tadj0++ ) {
        for( tadj1 = 0; tadj1 < triNeighbourCounts[ vertex1 ]; tadj1++ ) {
            if( triNeighbours[ vertex0 ][ tadj0 ] == triNeighbours[ vertex1 ][ tadj1 ] )~ {
                edgeTriNeighbours[ edgeTriNeighbourCount++ ] =
                    triNeighbours[ vertex0 ][ tadj0 ]
            }
        }
    }
    return edgeTriNeighbours, edgeTriNeighbourCount
}

v = FindFaceThirdVertex( a, b, face ){
    v = -1
    for( id = 0; id < 3; id++ ){
        if( face[ id ] != a && face[ id ] != b ){
            v = face[ id ]
        }
    }
    return v
}

```

6 Overall V-DMC characteristics, decoding operations, and post-decoding processes

6.1 V-DMC characteristics

This document enables the encoding and decoding processes of media representing dynamic mesh by using V3C technology.

This is achieved through first a conversion of input dynamic mesh representation into number of V3C components: a basemesh, a set of displacements, 2D representation of the attributes, and an atlas.

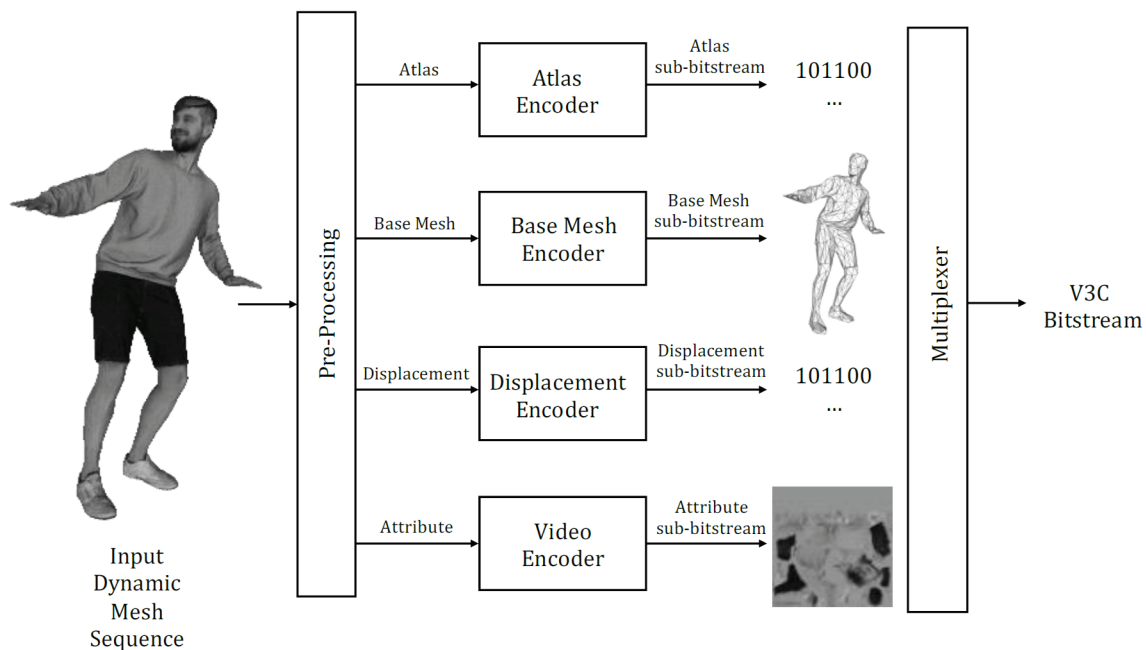
The basemesh component may be a simplified version of the input mesh. This document defines a static mesh codec in [Annex I](#) and motion codec in [Annex L](#) that can be used by the generic mechanism of the basemesh wrapper defined in [Annex H](#).

The displacements component provides displacement vectors, that can be either encoded as V3C geometry video component or encoded using arithmetic coding method defined in [Annex J](#).

The attribute components can provide additional properties, e.g. texture or material information, and are encoded as a V3C video component.

NOTE Any existing video coding specification such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video coding specification can be used.

Finally, the atlas component provides information to a V3C decoding and/or rendering system on how to perform inverse reconstruction. For example, how to perform the subdivision of basemesh and how to apply the displacement vectors to the subdivided mesh vertices and how to apply attributes to the reconstructed mesh. An example of encoder and decoder flow is presented on [Figure 1](#).



(a)

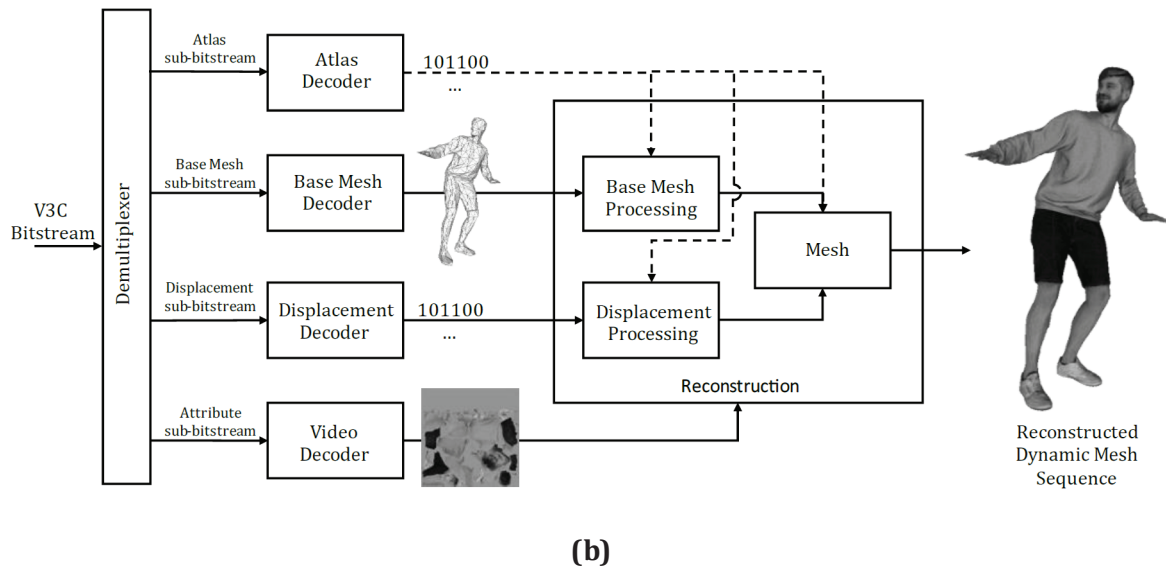


Figure 1 — Volumetric media conversion at (a) encoder and reconstruction at (b) decoder side. The 3D media is converted to a series of sub-bitstreams: basemesh, displacement, and attributes. Additional atlas information is also included in the bitstream to enable inverse reconstruction.

6.2 V-DMC 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 and reconstruction related processes needed by applications, which may include nominal format conversion, pre-reconstruction, reconstruction, post-reconstruction, and adaptation.

As mentioned in [subclause 6.1](#), a V3C bitstream contains V3C components, such as atlas, basemesh, displacement, and attributes components. Furthermore, to provide additional functionalities and similar flexibility as available in many video specifications, the atlas component may be divided into tiles and is encapsulated into NAL units. Similarly, the basemesh component may be divided into submeshes and is encapsulated into NAL units. [Clause 7](#) provides further details on the encapsulation of V3C component bitstreams into V3C units and NAL units.

The syntax elements and their semantics are specified in [Clause 8](#). Particular focus is placed on new syntax elements for VDMC as well as on the extension of V3C syntax elements defined in ISO/IEC 23090-5 to V-DMC and 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, decoded basemesh information, and if available decoded displacement information, a set of decoded video sub-bitstreams, corresponding to attribute and displacement, and information from the VPS.

The decoded V3C components may require 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 format.

With the V3C components in their nominal formats, the mesh content is obtained according to the following steps: reconstruction and post-reconstruction.

In the reconstruction stage, described in [Clause 11](#), the basemesh components in the nominal format, BasemeshFramesNF, the video components in the nominal format, GeoFramesNF, and, if present, AttrFramesNF, along with the decoded atlas data, are processed to reconstruct the mesh content and associated information as outputs.

The reconstructed mesh content can be further processed by applying informative post-reconstruction methods, as described in [Clause 12](#).

7 Bitstream format, partitioning, and scanning processes

7.1 General

The specifications in ISO/IEC 23090-5(4E):2025:7.1 apply.

7.2 V3C bitstream formats

The specifications in ISO/IEC 23090-5(4E):2025:7.2 apply.

7.3 NAL bitstream formats

The specifications in ISO/IEC 23090-5(4E):2025:7.3 apply.

7.4 Partitioning of atlas frames into tiles

The specifications in ISO/IEC 23090-5(4E):2025:7.4 apply.

7.5 Tile partition scanning processes

The specifications in ISO/IEC 23090-5(4E):2025:7.5 apply.

7.6 Extension of partitioning process for tile attribute components

The partitioning of the atlas attribute nominal frame is as follows:

- When `afve_consistent_tiling_across_attribute_video_flag` is equal to 0, tiling is not consistent across attributes. The attribute frame is first partitioned into $\text{NumPartitionColumnsAtt}[\text{attrIdx}] * \text{NumPartitionRowsAtt}[\text{attrIdx}]$ number of tile partitions, where $\text{NumPartitionColumnsAtt}[\text{attrIdx}]$, $\text{NumPartitionRowsAtt}[\text{attrIdx}]$, the width and height of each tile partition, respectively, is indicated in the atlas frame tile attribute information syntax ([8.3.6.2.4](#)).
- When `afve_consistent_tiling_across_attribute_video_flag` is equal to 1, the tiling is consistent across attributes.
 - The attribute frame for attribute with index `attrIdx` equal to `afve_reference_attribute_idx` is first partitioned into $\text{NumPartitionColumnsAtt}[\text{attrIdx}] * \text{NumPartitionRowsAtt}[\text{attrIdx}]$ number of tile partitions, where $\text{NumPartitionColumnsAtt}[\text{attrIdx}]$, $\text{NumPartitionRowsAtt}[\text{attrIdx}]$ and the width and height of the tile partitions is indicated in the atlas frame tile attribute information syntax ([8.3.6.2.4](#)).
 - The attribute frame for attribute with index `attrIdx` not equal to `afve_reference_attribute_idx`, is partitioned into $\text{NumPartitionColumnsAtt}[\text{attrIdx}] * \text{NumPartitionRowsAtt}[\text{attrIdx}]$ number of tile partitions, where $\text{NumPartitionColumnsAtt}[\text{attrIdx}]$, $\text{NumPartitionRowsAtt}[\text{attrIdx}]$ and the width and height of the tile partitions is inferred from tile partitions for attribute with `attrIdx` equal to `afve_reference_attribute_idx`. This inference process is detailed in semantics [subclause \(8.3.6.2.4\)](#).
- 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 attribute. All tile partitions within these two tile partitions collectively form a tile, which is essentially a rectangular region of the atlas attribute nominal frame.

8 Syntax and semantics

8.1 Method of specifying syntax in tabular form

The specifications in ISO/IEC 23090-5(4E):2025:8.1 apply.

8.2 Specification of syntax functions and descriptors

The specifications in ISO/IEC 23090-5(4E):2025:8.2 apply.

8.3 Syntax in tabular form

8.3.1 General

The specifications in ISO/IEC 23090-5(4E):2025:8.3.1 apply.

8.3.2 V3C unit syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.2 apply.

8.3.3 Byte alignment syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.3 apply.

8.3.4 V3C parameter set syntax

8.3.4.1 General V3C parameter set syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.1 apply.

8.3.4.2 Profile, tier, and level syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.2 apply.

8.3.4.3 Occupancy information syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.3 apply.

8.3.4.4 Geometry information syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.4 apply.

8.3.4.5 Attribute information syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.5 apply.

8.3.4.6 Profile toolset constraints information syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.6 apply with the following modification:

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)
if(ptc_num_reserved_constraint_bytes > 0)	
ptc_no_subdivision_flag	u(1)
ptc_no_displacement_flag	u(1)
ptc_displacement_ascending_flag	u(1)
ptc_displacement_dimension_constraint_flag	u(1)
ptc_displacement_video_flag	u(1)
ptc_no_packed_video_flag	u(1)
ptc_vdmc_reserved_zero_2bits	u(2)
}	
for(i = 1; i < ptc_num_reserved_constraint_bytes; i++)	
ptc_reserved_constraint_byte[i]	u(8)
}	

8.3.4.7 Packing information syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.7 apply.

8.3.4.8 VPS extension syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.8 apply.

8.3.4.9 Packed video extension syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.9 apply.

8.3.4.10 Length alignment syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.4.10 apply.

8.3.4.11 V3C parameter set V-DMC extension syntax

vps_vdmc_extension () {	Descriptor
for(k=0; k<vps_atlas_count_minus1+1; k++){	
j = vps_atlas_id[k]	
vpve_bmesh_data_substream_codec_id[j]	u(8)
vpve_bmesh_geometry_3d_bit_depth_minus1[j]	u(5)
vpve_bmesh_geometry_3d_msb_align_flag[j]	u(1)
vpve_bmesh_data_attribute_count[j]	u(8)
for(i = 0; i < vpve_bmesh_data_attribute_count[j]; i++) {	
vpve_bmesh_attribute_index[j][i]	u(7)

vpve_bmesh_attribute_bit_depth_minus1[j][i]	u(5)
vpve_bmesh_attribute_msb_align_flag[j][i]	u(1)
vpve_bmesh_attribute_type[j][i]	u(4)
}	
vpve_ac_displacement_present_flag[j]	u(1)
if(vpve_ac_displacement_present_flag[j])	
ac_displacement_information(j)	
vpve_consistent_attribute_frame_flag[j]	u(1)
for(i = 0; i < VpsAttributeNominalFrameCount[j]; i++) {	
vpve_attribute_frame_width[j][i]	ue(v)
vpve_attribute_frame_height[j][i]	ue(v)
}	
}	
}	

8.3.4.12 AC displacement information syntax

ac_displacement_information () {	Descriptor
acdi_codec_id[atlasID]	u(8)
acdi_msb_align_flag[atlasID]	u(1)
acdi_3d_bit_depth_minus1[atlasID]	u(5)
acdi_reserved_16_bits[atlasID]	u(16)
}	

8.3.5 NAL unit syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.5 apply.

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

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.1.1 apply.

8.3.6.1.2 Point local reconstruction information syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.1.2 apply.

8.3.6.1.3 Atlas sequence parameter set V-DMC extension syntax

asps_vdmc_extension() {	Descriptor
asve_subdivision_iteration_count	u(3)
AspsSubdivisionCount = asve_subdivision_iteration_count	
if(AspsSubdivisionCount > 1) {	
asve_lod_adaptive_subdivision_flag	u(1)
}	
if(AspsSubdivisionCount > 0) {	

asve_edge_based_subdivision_flag	u(1)
}	
for(i=0; i < AspsSubdivisionCount ; i++) {	
if(asve_lod_adaptive_subdivision_flag == 1 i == 0) {	
asve_subdivision_method[i]	u(3)
} else {	
asve_subdivision_method[i] = asve_subdivision_method[0]	
}	
AspsSubdivisionMethod[i] = asve_subdivision_method[i]	
}	
if(asve_edge_based_subdivision_flag == 1) {	
asve_subdivision_min_edge_length	u(16)
AspsSubdivisionMinEdgeLength = asve_subdivision_min_edge_length	
} else {	
AspsSubdivisionMinEdgeLength = 0	
}	
asve_1d_displacement_flag	u(1)
asve_interpolate_subdivided_normals_flag	u(1)
asve_displacement_reference_qp_minus49	se(v)
asve_quantization_parameters_present_flag	u(1)
if(asve_quantization_parameters_present_flag) {	
asve_inverse_quantization_offset_present_flag	u(1)
vdmc_quantization_parameters(0, AspsSubdivisionCount, 0)	
}	
if(AspsSubdivisionCount != 0) {	
asve_transform_method	u(3)
AspsTransformMethod = asve_transform_method	
}	
asve_lifting_offset_present_flag	u(1)
if(asve_transform_method == LINEAR_LIFTING && AspsSubdivisionCount != 0) {	
asve_directional_lifting_present_flag	u(1)
vdmc_lifting_transform_parameters(0, AspsSubdivisionCount)	
}	
asve_attribute_information_present_flag	u(1)
if(asve_attribute_information_present_flag) {	
asve_consistent_attribute_frame_flag	u(1)
if(!asve_consistent_attribute_frame_flag) {	
asve_attribute_frame_count	u(7)
}	
for(i=0; i < AspsAttributeNominalFrameCount; i++){	
asve_attribute_frame_width[i]	ue(v)
asve_attribute_frame_height[i]	ue(v)
asve_attribute_subtexture_enabled_flag[i]	u(1)
}	

}	
asve_displacement_id_present_flag	u(1)
asve_lod_patches_enable_flag	u(1)
asve_packing_method	u(1)
if (!asve_attribute_information_present_flag) {	
asve_projection_texcoord_enable_flag	u(1)
if(asve_projection_texcoord_enable_flag){	
asve_projection_texcoord_mapping_attribute_index_present_flag	u(1)
if(asve_projection_texcoord_mapping_attribute_index_present_flag) {	
asve_projection_texcoord_mapping_attribute_index	u(7)
}	
asve_projection_texcoord_output_bit_depth_minus1	u(5)
asve_projection_texcoord_bbox_bias_enable_flag	u(1)
asve_projection_texcoord_upscale_factor_minus1	u(40)
asve_projection_texcoord_log2_downscale_factor	u(6)
asve_projection_raw_textcoord_present_flag	u(1)
if(asve_projection_raw_textcoord_present_flag)	
asve_projection_raw_textcoord_bitdepth_minus1	u(v)
}	
}	
asve_vdmc_vui_parameters_present_flag	u(1)
if(asve_vdmc_vui_parameters_present_flag)	
vdmc_vui_parameters()	
}	

8.3.6.1.4 Quantization parameters syntax

vdmc_quantization_parameters(qpIndex, subdivisionCount, refSubdivisionCount){	Descriptor
vqp_lod_quantization_flag [qpIndex]	u(1)
if(vqp_lod_quantization_flag[qpIndex] == 0) {	
for(k = 0; k < AspsDispComponents; k++) {	
vqp_quantization_parameter [qpIndex][k]	u(7)
for(i=0 ; i < subdivisionCount + 1; i++)	
QuantizationParameter[qpIndex][i][k] =	
vqp_quantization_parameter[qpIndex][i][k]	
vqp_log2_lod_inverse_scale [qpIndex][k]	u(2)
}	
} else {	
for(i=0 ; i < subdivisionCount + 1; i++) {	
for(k = 0; k < AspsDispComponents; k++) {	
vqp_lod_delta_quantization_parameter_value [qpIndex][i][k]	ue(v)
if(vqp_lod_delta_quantization_parameter_value[qpIndex][i][k])	
vqp_lod_delta_quantization_parameter_sign [qpIndex][i][k]	u(1)

<pre> if(qpIndex = 0) { QuantizationParameter[qpIndex][i][k] = asve_displacement_reference_qp_minus49 + 49 + (1 - 2 * vqp_lod_delta_quantization_parameter_sign[qpIndex][i][k]) * vqp_lod_delta_quantization_parameter_value[qpIndex][i][k] } else if(subdivisionCount <= refSubdivisionCount) { QuantizationParameter[qpIndex][i][k] = QuantizationParameter[qpIndex - 1][i][k] + (1 - 2 * vqp_lod_delta_quantization_parameter_sign[qpIndex][i][k]) * vqp_lod_delta_quantization_parameter_value[qpIndex][i][k] } else { QuantizationParameter[qpIndex][i][k] = QuantizationParameter[qpIndex - 1][0][k] + (1 - 2 * vqp_lod_delta_quantization_parameter_sign[qpIndex][i][k]) * vqp_lod_delta_quantization_parameter_value[qpIndex][i][k] } } } } </pre>	
vqp_direct_quantization_enabled_flag [qpIndex]	u(1)
if (vqp_direct_quantization_enabled_flag[qpIndex])	
vqp_bitdepth_offset [qpIndex]	se(v)
}	

8.3.6.1.5 Lifting transform parameters syntax

vdmc_lifting_transform_parameters(ltpIndex, subdivisionCount) {	Descriptor
vltp_skip_update_flag [ltpIndex]	u(1)
for(i = 0 ; i < subdivisionCount ; i++) {	
if(vltp_skip_update_flag[ltpIndex]) {	
UpdateWeight[ltpIndex][i] = 0	
} else {	
if(i == 0) {	
vltp_adaptive_update_weight_flag [ltpIndex]	u(1)
vltp_valence_update_flag [ltpIndex]	u(1)
}	
if(vltp_adaptive_update_weight_flag[ltpIndex] == 1 i == 0) {	
vltp_lifting_update_weight_numerator [ltpIndex][i]	ue(v)
vltp_lifting_update_weight_denominator_minus1 [ltpIndex][i]	ue(v)
UpdateWeight[ltpIndex][i] =	
(vltp_lifting_update_weight_numerator[ltpIndex][i]) ÷	
(vltp_lifting_update_weight_denominator_minus1[ltpIndex][i] + 1)	
} else {	
UpdateWeight[ltpIndex][i] = UpdateWeight[ltpIndex][0]	
}	
}	
}	

vltp_adaptive_prediction_weight_flag [ltpIndex]	u(1)
if(vltp_adaptive_prediction_weight_flag[ltpIndex]) {	
for(i=0 ; i < subdivisionCount; i++) {	
vltp_lifting_prediction_weight_numerator [ltpIndex][i]	ue(v)
vltp_lifting_prediction_weight_denominator_minus1 [ltpIndex][i]	ue(v)
}	
}	
for(i=0 ; i < subdivisionCount; i++) {	
PredictionWeight[ltpIndex][i] =	
(vltp_lifting_prediction_weight_numerator[ltpIndex][i]) ÷	
(vltp_lifting_prediction_weight_denominator_minus1[ltpIndex][i] + 1)	
}	
if (asve_directional_lifting_present_flag) {	
for(i = 0; i < subdivisionCount; i++) {	
vltp_directional_lifting_lod_flag [ltpIndex][i]	u(1)
}	
}	
}	

8.3.6.2 Atlas frame parameter set RBSP syntax

8.3.6.2.1 General atlas frame parameter set RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.2.1

8.3.6.2.2 Atlas frame tile information syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.2.2

8.3.6.2.3 Atlas frame parameter set V-DMC extension syntax

afps_vdmc_extension () {	Descriptor
AfpsSubdivisionCount = AspsSubdivisionCount	
AfpsTransformMethod = AspsTransformMethod	
afve_override_flag	u(1)
if(afve_override_flag) {	
afve_subdivision_iteration_count_present_flag	u(1)
if(afve_subdivision_iteration_count_present_flag) {	
afve_subdivision_iteration_count	u(3)
AfpsSubdivisionCount = afve_subdivision_iteration_count	
}	
if(AfpsSubdivisionCount != 0)	
afve_transform_method_present_flag	u(1)
if(!afve_subdivision_iteration_count_present_flag) {	
if(AfpsSubdivisionCount != 0)	
afve_subdivision_method_present_flag	u(1)
if(asve_quantization_parameters_present_flag)	

afve_quantization_parameters_present_flag	u(1)
if((!afve_transform_method_present_flag) && AfpsSubdivisionCount != 0))	
afve_transform_parameters_present_flag	u(1)
}	
}	
if(afve_subdivision_method_present_flag) {	
if(AfpsSubdivisionCount > 1) {	
afve_lod_adaptive_subdivision_flag	u(1)
}	
for(i=0; i < AfpsSubdivisionCount ; i++) {	
if(afve_lod_adaptive_subdivision_flag == 1 i == 0) {	
afve_subdivision_method[i]	u(3)
} else {	
afve_subdivision_method[i] = afve_subdivision_method[0]	
}	
AfpsSubdivisionMethod[i] = afve_subdivision_method[i]	
}	
if(asve_edge_based_subdivision_flag == 1) {	
afve_subdivision_min_edge_length	u(16)
AfpsSubdivisionMinEdgeLength = afve_subdivision_min_edge_length	
} else {	
AfpsSubdivisionMinEdgeLength = 0	
}	
} else {	
for(i = 0; i < AfpsSubdivisionCount; i++){	
AfpsSubdivisionMethod[i] = AspsSubdivisionMethod[i]	
}	
AfpsSubdivisionMinEdgeLength = AspsSubdivisionMinEdgeLength	
}	
if(afve_quantization_parameters_present_flag)	
vdmc_quantization_parameters(1, AfpsSubdivisionCount, AspsSubdivisionCount)	
if(afve_transform_method_present_flag && (AfpsSubdivisionCount != 0)) {	
afve_transform_method	u(3)
AfpsTransformMethod = afve_transform_method	
}	
if(afve_transform_method ==	
LINEAR_LIFTING && afve_transform_parameters_present_flag) {	
vdmc_lifting_transform_parameters(1, AfpsSubdivisionCount)	
}	
if(asve_attribute_information_present_flag) {	
if(AspsAttributeNominalFrameCount > 1)	
afve_consistent_tiling_across_attribute_video_flag	u(1)
if(afve_consistent_tiling_across_attribute_video_flag) {	
if(AspsAttributeNominalFrameCount > 1)	

afve_reference_attribute_idx	ue(v)
atlas_frame_attribute_tile_information(afve_reference_attribute_idx)	
} else {	
for(attrIdx=0; attrIdx< AspsAttributeNominalFrameCount; attrIdx++) {	
atlas_frame_attribute_tile_information(attrIdx)	
}	
}	
}	
atlas_frame_mesh_information()	
if(asve_projection_texcoord_enable_flag) {	
for(i = 0; i < NumSubMeshes; i++) {	
afve_projection_texcoord_present_flag[i]	u(1)
if(afve_projection_texcoord_present_flag[i]) {	
afve_projection_texcoord_width_normalization[i]	ue(v)
afve_projection_texcoord_height_normalization[i]	ue(v)
afve_projection_texcoord_gutter[i]	ue(v)
}	
}	
}	
}	

8.3.6.2.4 Atlas frame tile attribute information syntax

atlas_frame_attribute_tile_information(attrIdx) {	Descriptor
StartIndex[attrIdx] = MaxValueInArray(TileIDToIndex) + 1	
StartId[attrIdx] = MaxValueInArray(TileIndexToID) + 1	
aftai_single_tile_in_atlas_frame_flag[attrIdx]	u(1)
if(!aftai_single_tile_in_atlas_frame_flag[attrIdx]) {	
aftai_uniform_partition_spacing_flag[attrIdx]	u(1)
if(aftai_uniform_partition_spacing_flag[attrIdx]) {	
aftai_partition_cols_width_minus1[attrIdx]	ue(v)
aftai_partition_rows_height_minus1[attrIdx]	ue(v)
} else {	
aftai_num_partition_columns_minus1[attrIdx]	ue(v)
aftai_num_partition_rows_minus1[attrIdx]	ue(v)
for(i = 0; i < aftai_num_partition_columns_minus1[attrIdx]; i++)	
aftai_partition_column_width_minus1[attrIdx][i]	ue(v)
for(i = 0; i < aftai_num_partition_rows_minus1[attrIdx]; i++)	
aftai_partition_row_height_minus1[attrIdx][i]	ue(v)
}	
aftai_single_partition_per_tile_flag[attrIdx]	u(1)
if(!aftai_single_partition_per_tile_flag[attrIdx]) {	
aftai_num_tiles_in_atlas_frame_minus1[attrIdx]	ue(v)
for(i = 0; i < aftai_num_tiles_in_atlas_frame_minus1[attrIdx] + 1; i++) {	
aftai_top_left_partition_idx[attrIdx][i]	u(v)

aftai_bottom_right_partition_column_offset [attrIdx][i]	ue(v)
aftai_bottom_right_partition_row_offset [attrIdx][i]	ue(v)
}	
}	
else	
aftai_num_tiles_in_atlas_frame_minus1[attrIdx] = NumAttributePartitionsInAtlasFrame[attrIdx] - 1	
}	
else {	
aftai_num_tiles_in_atlas_frame_minus1[attrIdx] = 0	
aftai_nal_tile_present_flag [attrIdx]	u(1)
}	
aftai_signalled_tile_id_flag [attrIdx]	u(1)
if(aftai_signalled_tile_id_flag[attrIdx]) {	
aftai_signalled_tile_id_length_minus1 [attrIdx]	ue(v)
for(i = 0; i < aftai_num_tiles_in_atlas_frame_minus1[attrIdx] + 1; i++) {	
aftai_tile_id [attrIdx][i]	u(v)
TileIDToIndex[aftai_tile_id[attrIdx][i]] = StartIndex[attrIdx] + i	
TileIndexToID[start_index + i] = aftai_tile_id[attrIdx][i]	
TileIDToAtlasAttributeIdx[aftai_tile_id[attrIdx][i]] = attrIdx	
TileIDToNALPresentFlag[aftai_tile_id[attrIdx][i]] = aftai_nal_tile_present_flag[attrIdx]	
}	
} else {	
for(i = 0; i < aftai_num_tiles_in_atlas_frame_minus1[attrIdx] + 1; i++) {	
aftai_tile_id[attrIdx][i] = StartId[attrIdx] + i	
TileIDToIndex[aftai_tile_id[attrIdx][i]] = StartIndex[attrIdx] + i	
TileIndexToID[start_index + i] = aftai_tile_id[attrIdx][i]	
TileIDToAtlasAttributeIdx[aftai_tile_id[attrIdx][i]] = attrIdx	
TileIDToNALPresentFlag[aftai_tile_id[attrIdx][i]] = aftai_nal_tile_present_flag[attrIdx]	
}	
}	
}	

8.3.6.2.5 Atlas frame mesh information syntax

atlas_frame_mesh_information () {	Descriptor
afmi_use_single_mesh_flag	u(1)
if(!afmi_use_single_mesh_flag) {	
afmi_num_submeshes_minus2	u(8)
NumSubMeshes = afmi_num_submeshes_minus2 + 2	
}	
else	
NumSubMeshes = 1	

afmi_signalled_submesh_id_flag	u(1)
if(afve_signalled_submesh_id_flag) {	
afmi_signalled_submesh_id_delta_length	ue(v)
for(i = 0; i < NumSubMeshes; i++)	
afmi_submesh_id[i]	u(v)
SubmeshIDToIndex[afmi_submesh_id[i]] = i	
SubmeshIndexToID[i] = afmi_submesh_id[i]	
}	
} else {	
for(i = 0; i < NumSubMeshes; i++) {	
afmi_submesh_id[i] = i	
SubmeshIDToIndex[i] = i	
SubmeshIndexToID[i] = i	
}	
}	
}	

8.3.6.3 Atlas adaptation parameter set RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.3 apply.

8.3.6.4 Supplemental enhancement information RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.4 apply.

8.3.6.5 Access unit delimiter RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.5 apply.

8.3.6.6 End of sequence RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.6 apply.

8.3.6.7 End of bitstream RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.7 apply.

8.3.6.8 Filler data RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.8 apply.

8.3.6.9 Atlas tile layer RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.9 apply.

8.3.6.10 RBSP trailing bit syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.10 apply.

8.3.6.11 Atlas tile header syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.11 apply.

8.3.6.12 Reference list structure syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.10 apply.

8.3.6.13 Common atlas sequence parameter set RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.13 apply.

8.3.6.14 Common atlas frame RBSP syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.6.14 apply.

8.3.7 Atlas tile data unit syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.7 apply.

8.3.8 Supplemental enhancement information message syntax

The specifications in ISO/IEC 23090-5(4E):2025:8.3.8 apply.

8.3.9 VDMC atlas tile data unit syntax**8.3.9.1 General VDMC atlas tile data unit syntax**

vmc_atlas_tile_data_unit(tileID) {	Descriptor
if(ath_type == SKIP_TILE) {	
for(p = 0; p < RefAtduTotalNumMeshpatches[tileID]; p++)	
skip_meshpatch_data_unit()	
} else {	
p = 0	
do {	
atdu_meshpatch_mode[tileID][p]	ue(v)
isEnd = (((ath_type == P_TILE) (ath_type == P_TILE_ATTR)) && (atdu_meshpatch_mode[tileID][p] == P_END)) (((ath_type == I_TILE) (ath_type == I_TILE_ATTR)) && (atdu_meshpatch_mode[tileID][p] == I_END))	
if(!isEnd) {	
meshpatch_information_data(tileID, p, atdu_meshpatch_mode[tileID][p])	
p++	
}	
} while(!isEnd)	
}	
AtduTotalNumMeshpatches[tileID] = p	
}	

8.3.9.2 Meshpatch information data syntax

meshpatch_information_data(tileID, patchIdx, meshpatchMode) {	Descriptor
if(ath_type == P_TILE) {	
if(meshpatchMode == P_SKIP)	
skip_meshpatch_data_unit()	
else if(meshpatchMode == P_MERGE)	
merge_meshpatch_data_unit(tileID, patchIdx)	
else if(meshpatchMode == P_INTRA)	
meshpatch_data_unit(tileID, patchIdx)	
else if(meshpatchMode == P_INTER)	
inter_meshpatch_data_unit(tileID, patchIdx)	
}	
else if(ath_type == I_TILE ath_type == I_TILE_ATTR) {	
if(meshpatchMode == I_INTRA)	
meshpatch_data_unit(tileID, patchIdx)	
}	
else if(ath_type == P_TILE_ATTR) {	
if(meshpatchMode == P_INTRA)	
meshpatch_data_unit(tileID, patchIdx)	
if(meshpatchMode == P_SKIP)	
skip_meshpatch_data_unit()	
}	
}	

8.3.9.3 Meshpatch data unit syntax

meshpatch_data_unit(tileID, patchIdx) {	Descriptor
mdu_submesh_id [tileID][patchIdx]	ue(v)
if(ath_type == P_TILE ath_type == I_TILE) {	
if(AspsDisplacementIdPresentFlag) {	
mdu_displ_id [tileID][patchIdx]	ue(v)
} else {	
if(AspsLodPatchesEnableFlag)	
mdu_lod_idx [tileID][patchIdx]	ue(v)
mdu_2d_pos_x [tileID][patchIdx]	ue(v)
mdu_2d_pos_y [tileID][patchIdx]	ue(v)
mdu_2d_size_x_minus1 [tileID][patchIdx]	ue(v)
mdu_2d_size_y_minus1 [tileID][patchIdx]	ue(v)
}	
if(mdu_lod_idx[tileID][patchIdx] == 0) {	
PatchSubdivisionCount[tileID][patchIdx] = AfpsSubdivisionCount	
mdu_parameters_override_flag [tileID][patchIdx]	u(1)
if(mdu_parameters_override_flag[tileID][patchIdx]) {	
mdu_subdivision_iteration_count_present_flag [tileID][patchIdx]	u(1)
if(mdu_subdivision_iteration_count_present_flag[tileID][patchIdx]) {	

mdu_subdivision_iteration_count	u(3)
PatchSubdivisionCount[tileID][patchIdx] = mdu_subdivision_iteration_count	
}	
if(PatchSubdivisionCount[tileID][patchIdx] != 0)	
mdu_transform_method_present_flag [tileID][patchIdx]	u(1)
if(!mdu_subdivision_iteration_count_present_flag[tileID][patchIdx]) {	
if(PatchSubdivisionCount[tileID][patchIdx] != 0)	
mdu_subdivision_method_present_flag [tileID][patchIdx]	u(1)
if(asve_quantization_parameters_present_flag)	
mdu_quantization_present_flag [tileID][patchIdx]	u(1)
if ((!mdu_transform_method_present_flag[tileID][patchIdx]) && PatchSubdivisionCount[tileID][patchIdx] != 0)	
mdu_transform_parameters_present_flag [tileID][patchIdx]	u(1)
}	
}	
if(mdu_subdivision_method_present_flag[tileID][patchIdx]){	
if(PatchSubdivisionCount[tileID][patchIdx] > 1) {	
mdu_lod_adaptive_subdivision_flag [tileID][patchIdx]	u(1)
}	
for(i = 0; i < PatchSubdivisionCount[tileID][patchIdx]; i++) {	
if(mdu_lod_adaptive_subdivision_flag == 1 i == 0) {	
mdu_subdivision_method [tileID][patchIdx][i]	u(3)
} else {	
mdu_subdivision_method[tileID][patchIdx][i] = mdu_subdivision_method[tileID][patchIdx][0]	
}	
PatchSubdivisionMethod[tileID][patchIdx][i] = mdu_subdivision_method[i]	
}	
if(asve_edge_based_subdivision_flag == 1) {	
mdu_subdivision_min_edge_length [tileID][patchIdx]	u(16)
PatchSubdivisionMinEdgeLength[tileID][patchIdx] = mdu_subdivision_min_edge_length[tileID][patchIdx]	
} else {	
PatchSubdivisionMinEdgeLength[tileID][patchIdx] = 0	
}	
} else {	
for(i = 0; i < PatchSubdivisionCount[tileID][patchIdx]; i++){	
PatchSubdivisionMethod[tileID][patchIdx][i] = AfpsSubdivisionMethod[i]	
}	
PatchSubdivisionMinEdgeLength[tileID][patchIdx] = AfpsSubdivisionMinEdgeLength	
}	

if(mdu_quantization_present_flag[tileID][patchIdx])	
vdmc_quantization_parameters(2, PatchSubdivisionCount[tileID][patchIdx], AfpsSubdivisionCount)	
if(AspsInvQuantOffsetPresentFlag) {	
mdu_inverse_quantization_offset_enable_flag [tileID][patchIdx]	u(1)
if(mdu_inverse_quantization_offset_enable_flag) {	
for(i = 0; i < PatchSubdivisionCount[tileID][patchIdx]; i++) {	
for(j = 0; j < AspsDispComponents; j++) {	
for(k = 0; k < 3; k++) {	
mdu_inverse_quantization_offset_sign [tileID][patchIdx][i][j][k]	u(1)
mdu_inverse_quantization_offset_value_log2_prec1_delta [tileID][patchIdx][i][j][k]	se(v)
mdu_inverse_quantization_offset_value_log2_prec2_delta [tileID][patchIdx][i][j][k]	se(v)
}	
}	
}	
}	
mdu_displacement_coordinate_system [tileID][patchIdx]	u(1)
if(mdu_transform_method_present_flag[tileID][patchIdx] && PatchSubdivisionCount[tileID][patchIdx] != 0)	
mdu_transform_method [tileID][patchIdx]	u(3)
if(mdu_transform_method[tileID][patchIdx] == LINEAR_LIFTING) {	
if(AspsLiftingOffsetPresentFlag) {	
mdu_lifting_offset_present_flag [tileID][patchIdx]	u(1)
if(mdu_lifting_offset_present_flag[tileID][patchIdx]) {	
for(i = 0; i < PatchSubdivisionCount[tileID][patchIdx]; i++) {	
mdu_lifting_offset_values_num [tileID][patchIdx][i]	se(v)
mdu_lifting_offset_values_deno_minus1 [tileID][patchIdx][i]	ue(v)
}	
}	
}	
if(AspsDirectionalLiftingPresentFlag) {	
for(i = 0; i < 2; i++) {	
mdu_directional_lifting_scale_num [tileID][patchIdx][i]	se(v)
}	
if (mdu_transform_parameters_present_flag[tileID][patchIdx]) {	
vdmc_lifting_transform_parameters(2, PatchSubdivisionCount[tileID][patchIdx])	
}	
}	
if(AspsDisplacementIdPresentFlag (AspsLodPatchesEnableFlag == 0)) {	
vertexInfoCount = PatchSubdivisionCount[tileID][patchIdx] + 1	

} else {	
vertexInfoCount = 1	
}	
for(i = 0; i < vertexInfoCount; i++) {	
mdu_block_count_minus1 [tileID][patchIdx][i];	u(v)
mdu_last_pos_in_block [tileID][patchIdx][i]	u(v)
}	
smIdx = SubmeshIDToIndex[mdu_submesh_id[tileID][patchIdx]]	
if(afve_projection_texcoord_present_flag[smIdx])	
texture_projection_information(tileID, patchIdx)	
}	
if(ath_type == P_TILE_ATT ath_type == I_TILE_ATTR) {	
i = TileIDToAtlasAttributeIdx[tileID]	
if(AspsAttributeSubtextureEnabledFlag[i]){	
mdu_attributes_2d_pos_x [tileID][patchIdx]	ue(v)
mdu_attributes_2d_pos_y [tileID][patchIdx]	ue(v)
mdu_attributes_2d_size_x_minus1 [tileID][patchIdx]	ue(v)
mdu_attributes_2d_size_y_minus1 [tileID][patchIdx]	ue(v)
}	
}	
}	

8.3.9.4 Skip meshpatch data unit syntax

skip_meshpatch_data_unit() {	Descriptor
}	

8.3.9.5 Merge meshpatch data unit syntax

merge_meshpatch_data_unit(tileID, patchIdx) {	Descriptor
if(NumRefIdxActive)	
mmdu_ref_index [tileID][patchIdx]	ue(v)
mmdu_patch_index [tileID][patchIdx]	se(v)
if(AspsLodPatchesEnableFlag)	
mmdu_lod_idx [tileID][patchIdx]	ue(v)
if(mmdu_lod_idx[tileID][patchIdx] == 0) {	
mmdu_subdivision_iteration_count_present_flag [tileID][patchIdx]	u(1)
if(mmdu_subdivision_iteration_count_present_flag[tileID][patchIdx]) {	
mmdu_subdivision_iteration_count [tileID][patchIdx]	u(3)
MergePatchSubdivisionCount[tileID][patchIdx] = mmdu_subdivision_iteration_count[tileID][patchIdx]	
} else {	
MergePatchSubdivisionCount[tileID][patchIdx] = AfpsSubdivisionCount	
}	
if(AspsInvQuantOffsetPresentFlag) {	
mmdu_inverse_quantization_offset_enable_flag [tileID][patchIdx]	u(1)

if(mmdu_inverse_quantization_offset_enable_flag) {	
for(i = 0; i < MergePatchSubdivisionCount[tileID][patchIdx]; i++) {	
for(j = 0; j < AspsDispComponents; j++) {	
for(k = 0; k < 3; k++) {	
mmdu_inverse_quantization_offset_sign [tileID][patchIdx][i][j][k]	u(1)
mmdu_inverse_quantization_offset_value_log2_prec1_delta [tileID][patchIdx][i][j][k]	se(v)
mmdu_inverse_quantization_offset_value_log2_prec2_delta [tileID][patchIdx][i][j][k]	se(v)
}	
}	
}	
}	
if(AspsLiftingOffsetPresentFlag){	
mmdu_lifting_offset_present_flag [tileID][patchIdx]	u(1)
if(mmdu_lifting_offset_present_flag[tileID][patchIdx]) {	
for(i = 0; i < MergePatchSubdivisionCount[tileID][patchIdx]; i++) {	
mmdu_lifting_offset_delta_values_num [tileID][patchIdx][i]	se(v)
mmdu_lifting_offset_delta_values_deno [tileID][patchIdx][i]	se(v)
}	
}	
}	
if(AspsDirectionalLiftingPresentFlag) {	
for(i = 0; i < 2; i++) {	
mmdu_directional_lifting_scale_num [tileID][patchIdx][i]	se(v)
}	
}	
if(asve_projection_texcoord_enable_flag){	
mmdu_texture_projection_present_flag [tileID][patchIdx]	u(1)
if(mmdu_texture_projection_present_flag[tileID][patchIdx])	
texture_projection_merge_information(tileID, patchIdx)	
}	
}	
}	
}	

8.3.9.6 Inter meshpatch data unit syntax

inter_meshpatch_data_unit(tileID, patchIdx) {	Descriptor
if(NumRefIdxActive)	
imdu_ref_index [tileID][patchIdx]	ue(v)
imdu_patch_index [tileID][patchIdx]	se(v)
if(!AspsDisplacementIdPresentFlag) {	
if(AspsLodPatchesEnableFlag)	
imdu_lod_idx [tileID][patchIdx]	ue(v)
imdu_2d_delta_pos_x [tileID][patchIdx]	se(v)

imdu_2d_delta_pos_y [tileID][patchIdx]	se(v)
imdu_2d_delta_size_x [tileID][patchIdx]	se(v)
imdu_2d_delta_size_y [tileID][patchIdx]	se(v)
}	
if(imdu_lod_idx[tileID][patchIdx] == 0) {	
imdu_subdivision_iteration_count_present_flag [tileID][patchIdx]	u(1)
if(imdu_subdivision_iteration_count_present_flag[tileID][patchIdx]){	
imdu_subdivision_iteration_count [tileID][patchIdx]	U(3)
InterPatchSubdivisionCount[tileID][patchIdx] = imdu_subdivision_iteration_count[tileID][patchIdx]	
}else	
InterPatchSubdivisionCount[tileID][patchIdx] = AfpsSubdivisionCount	
if(AspsInvQuantOffsetPresentFlag) {	
imdu_inverse_quantization_offset_enable_flag [tileID][patchIdx]	u(1)
if(mmdu_inverse_quantization_offset_enable_flag) {	
for(i = 0; i < MergePatchSubdivisionCount[tileID][patchIdx]; i++) {	
for(j = 0; j < AspsDispComponents; j++) {	
for(k = 0; k < 3; k++) {	
imdu_inverse_quantization_offset_sign [tileID][patchIdx][i][j][k]	u(1)
imdu_inverse_quantization_offset_value_log2_prec1_delta [tileID][patchIdx][i][j][k]	se(v)
imdu_inverse_quantization_offset_value_log2_prec2_delta [tileID][patchIdx][i][j][k]	se(v)
}	
}	
}	
}	
if(InterPatchSubdivisionCount[tileID][patchIdx] != 0)	
imdu_transform_parameters_present_flag [tileID][patchIdx]	u(1)
imdu_transform_method_present_flag [tileID][patchIdx]	u(1)
if(imdu_transform_method_present_flag[tileID][patchIdx]) {	
imdu_transform_method [tileID][patchIdx]	u(3)
InterPatchTransformMethod[tileID][patchIdx] = imdu_transform_method[tileID][patchIdx]	
} else {	
InterPatchTransformMethod[tileID][patchIdx] = AfpsTransformMethod	
}	
if(AspsDisplacementIdPresentFlag (AspsLodPatchesEnableFlag == 0)) {	
vertexInfoCount = InterPatchSubdivisionCount[tileID][patchIdx] + 1	
} else {	
vertexInfoCount = 1	
}	
for(i = 0; i < vertexInfoCount; i++){	
imdu_delta_block_count [tileID][patchIdx][i]	se(v)
imdu_delta_last_pos_in_block [tileID][patchIdx][i]	se(v)

}	
if (imdu_lod_idx[tileID][patchIdx] == 0) {	
if(InterPatchTransformMethod[tileID][patchIdx] == LINEAR_LIFTING && AspsLiftingOffsetPresentFlag){	
imdu_lifting_offset_present_flag [tileID][patchIdx]	u(1)
if(imdu_lifting_offset_present_flag[tileID][patchIdx]) {	
for(i = 0; i < InterPatchSubdivisionCount[tileID][patchIdx]; i++) {	
imdu_lifting_offset_delta_values_num [tileID][patchIdx][i]	se(v)
imdu_lifting_offset_delta_values_deno [tileID][patchIdx][i]	se(v)
}	
}	
}	
if(InterPatchTransformMethod[tileID][patchIdx] == LINEAR_LIFTING && AspsDirectionalLiftingPresentFlag) {	
for(i = 0; i < 2; i++) {	
imdu_directional_lifting_scale_num [tileID][patchIdx][i]	se(v)
}	
}	
if(InterPatchTransformMethod[tileID][patchIdx] == LINEAR_LIFTING && (imdu_transform_parameters_present_flag[tileID][patchIdx] imdu_subdivision_iteration_count_present_flag[tileID][patchIdx] imdu_transform_method_present_flag[tileID][patchIdx])){	
vdmc_lifting_transform_parameters(2, InterPatchSubdivisionCount[tileID][patchIdx])	
}	
if(asve_projection_texcoord_enable_flag)	
imdu_texture_projection_present_flag [tileID][patchIdx]	u(1)
if(imdu_texture_projection_present_flag[tileID][patchIdx])	
texture_projection_inter_information(tileID, patchIdx)	
}	
}	

8.3.9.7 Texture projection information syntax

texture_projection_information(tileID, patchIdx) {	Descriptor
tpi_face_id_present_flag [tileID][patchIdx]	u(1)
tpi_frame_upscale_minus1 [tileID][patchIdx]	u(40)
tpi_log2_frame_downscale [tileID][patchIdx]	u(6)
tpi_subpatch_count_minus1 [tileID][patchIdx]	ue(v)
if(asve_projection_raw_textcoord_present_flag)	
tpi_subpatch_raw_enable_flag [tileID][patchIdx]	u(1)
numSubpatches = tpi_subpatch_count_minus1[tileID][patchIdx] + 1	
for(subIdx = 0; subIdx < numSubpatches; subIdx++) {	
if(tpi_face_id_present_flag[tileID][patchIdx])	
tpi_face_id [tileID][patchIdx][subIdx]	ue(v)
if(tpi_subpatch_raw_enable_flag[tileID][patchIdx])	

tpi_subpatch_raw_present_flag [tileID][patchIdx][subIdx]	u(1)
if(tpi_subpatch_raw_present_flag[tileID][patchIdx][subIdx])	
subpatch_raw_information(tileID, patchIdx , subIdx)	
else	
subpatch_information(tileID, patchIdx, subIdx)	
}	
}	

8.3.9.8 Sub-patch information syntax

subpatch_information(tileID, patchIdx, subIdx) {	Descriptor
si_projection_id [tileID][patchIdx][subIdx]	u(v)
si_orientation_id [tileID][patchIdx][subIdx]	ue(v)
si_2d_pos_x [tileID][patchIdx][subIdx]	ue(v)
si_2d_pos_y [tileID][patchIdx][subIdx]	ue(v)
if(AspsProjectionTexcoordBboxBiasEnableFlag){	
si_2d_bias_x [tileID][patchIdx][subIdx]	ue(v)
si_2d_bias_y [tileID][patchIdx][subIdx]	ue(v)
si_2d_size_x_minus1_delta [tileID][patchIdx][subIdx]	se(v)
si_2d_size_y_minus1_delta [tileID][patchIdx][subIdx]	se(v)
}	
si_scale_present_flag [tileID][patchIdx][subdx]	u(1)
if(si_scale_present_flag[tileID][patchIdx][subIdx])	
si_scale_power_factor [tileID][patchIdx][subIdx]	ue(v)
}	

8.3.9.9 Sub-patch RAW information syntax

subpatch_raw_information(tileID, patchIdx, subIdx) {	Descriptor
sri_num_raw_uv_minus1 [tileID][patchIdx][subIdx]	ue(v)
for(i = 0; i < sri_num_raw_uv_minus1[tileID][patchIdx][subIdx] + 1; i++) {	
sri_u_coord [tileID][patchIdx][subIdx][i]	u(v)
sri_v_coord [tileID][patchIdx][subIdx][i]	u(v)
}	
}	

8.3.9.10 Texture projection inter information syntax

texture_projection_inter_information(tileID, patchIdx) {	Descriptor
tpii_face_id_present_flag [tileID][patchIdx]	u(1)
tpii_frame_upscale_minus1 [tileID][patchIdx]	u(40)
tpii_log2_frame_downscale [tileID][patchIdx]	u(6)
tpii_subpatch_count_minus1 [tileID][patchIdx]	ue(v)
tpii_subpatch_inter_enable_flag [tileID][patchIdx]	u(1)
if(asve_projection_raw_textcoord_present_flag)	
tpii_subpatch_raw_enable_flag [tileID][patchIdx]	u(1)

numSubpatches = tpii_subpatch_count_minus1[tileID][patchIdx] + 1	
for(subIdx = 0; subIdx < numSubpatches; subIdx++) {	
if(tpii_face_id_present_flag[tileID][patchIdx])	
tpii_face_id [tileID][patchIdx][subIdx]	ue(v)
if(tpii_subpatch_inter_enable_flag[tileID][patchIdx])	
tpii_subpatch_inter_present_flag [tileID][patchIdx][subIdx]	u(1)
if(tpii_subpatch_raw_enable_flag[tileID][patchIdx])	
tpii_subpatch_raw_present_flag [tileID][patchIdx][subIdx]	u(1)
if(tpii_subpatch_inter_present_flag[tileID][patchIdx][subIdx])	
subpatch_inter_information(tileID, patchIdx, subIdx)	
else {	
if(tpii_subpatch_raw_present_flag[tileID][patchIdx][subIdx])	
subpatch_raw_information(tileID, patchIdx, subIdx)	
else	
subpatch_information(tileID, patchIdx, subIdx)	
}	
}	

8.3.9.11 Sub-patch inter information syntax

subpatch_inter_information(tileID, patchIdx, subIdx) {	Descriptor
sii_subpatch_index_delta [tileID][patchIdx][subIdx]	se(v)
sii_2d_pos_x_delta [tileID][patchIdx][subIdx]	se(v)
sii_2d_pos_y_delta [tileID][patchIdx][subIdx]	se(v)
if(AspsProjectionTexcoordBboxBiasEnableFlag){	
sii_2d_bias_x_delta [tileID][patchIdx][subIdx]	se(v)
sii_2d_bias_y_delta [tileID][patchIdx][subIdx]	se(v)
sii_2d_size_x_delta [tileID][patchIdx][subIdx]	se(v)
sii_2d_size_y_delta [tileID][patchIdx][subIdx]	se(v)
}	
}	

8.3.9.12 Texture projection merge information syntax

texture_projection_merge_information(tileID, patchIdx) {	Descriptor
tpmi_subpatch_merge_present_flag [tileID][patchIdx]	u(1)
if(tpmi_submesh_merge_present_flag[tileID][patchIdx]) {	
tpmi_update_subpatch_count_minus1 [tileID][patchIdx]	ue(v)
for(subIdx = 0;	
subIdx < tpmi_update_subpatch_count_minus1[tileID][patchIdx];	
subIdx++)	
subpatch_merge_information(tileID, patchIdx, subIdx)	
}	
}	
}	

8.3.9.13 Sub-patch merge information syntax

subpatch_merge_information(tileID, patchIdx, subIdx) {	Descriptor
smi_subpatch_index [tileID][patchIdx][subIdx]	se(v)
smi_2d_pos_x_delta [tileID][patchIdx][subIdx]	se(v)
smi_2d_pos_y_delta [tileID][patchIdx][subIdx]	se(v)
if(AspsProjectionTexcoordBboxBiasEnableFlag){	
smi_2d_bias_x_delta [tileID][patchIdx][subIdx]	se(v)
smi_2d_bias_y_delta [tileID][patchIdx][subIdx]	se(v)
smi_2d_size_x_delta [tileID][patchIdx][subIdx]	se(v)
smi_2d_size_y_delta [tileID][patchIdx][subIdx]	se(v)
}	
}	

8.4 Semantics**8.4.1 General semantics**

The specifications in ISO/IEC 23090-5(4E):2025:8.4 apply.

8.4.2 V3C unit semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.2 apply.

8.4.3 Byte alignment semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.3 apply.

8.4.4 V3C parameter set semantics**8.4.4.1 General V3C parameter set semantics**

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.1 apply.

8.4.4.2 Profile, tier, and level semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.2 apply.

8.4.4.3 Occupancy information semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.3 apply.

8.4.4.4 Geometry information semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.4 apply.

8.4.4.5 Attribute information semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.5 apply.

8.4.4.6 Profile toolset constraints information semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.6 apply with the following modification:

ptc_num_reserved_constraint_bytes shall have the value of either 0 or 1 in bitstreams conforming to this version of this document. Decoders conforming to this version of this document shall ignore values of **ptc_num_reserved_constraint_bytes** other than 0 or 1.

ptc_no_subdivision_flag equal to 1 specifies that **asve_subdivision_iteration_count_present_flag**, **asve_subdivision_iteration_count**, and **asve_subdivision_method** shall be equal to 0. **ptc_no_subdivision_only_flag** equal to 0 does not impose a constraint. When not present, **ptc_no_subdivision_flag** is inferred to be equal to 0.

ptc_no_displacement_flag equal to 1 specifies that **vps_geometry_video_present_flag[j]**, **pin_geometry_present_flag[j]**, and **vpve_ac_displacement_present_flag[j]** for any atlas ID *j* shall be equal to 0. **ptc_no_displacement_flag** equal to 0 does not impose a constraint. When not present, **ptc_no_displacement_flag** is inferred to be equal to 0.

ptc_displacement_ascending_flag equal to 1 specifies that **asve_packing_method** shall be equal to 0. **ptc_displacement_ascending_flag** equal to 0 does not impose a constraint. When not present, **ptc_displacement_ascending_flag** is inferred to be equal to 0.

ptc_displacement_dimension_constraint_flag equal to 1 specifies that **asve_1d_displacement_flag** shall be equal to 1. **ptc_displacement_dimension_constraint_flag** equal to 0 does not impose a constraint. When not present, **ptc_displacement_dimension_constraint_flag** is inferred to be equal to 0.

ptc_displacement_video_flag equal to 1 specifies that **asve_displacement_id_present_flag** shall be equal to 0. **ptc_displacement_video_flag** equal to 0 does not impose a constraint. When not present, **ptc_displacement_video_flag** is inferred to be equal to 0.

ptc_no_packed_video_flag equal to 1 specifies that **vps_packed_video_present_flag** shall be equal to 0. **ptc_no_packed_video_flag** equal to 0 does not impose a constraint. When not present, **ptc_no_packed_video_flag** is inferred to be equal to 0.

ptc_vdmc_reserved_zero_2bits shall be equal to 0 in bitstreams conforming to this version of this document. Other values of **ptc_vdmc_reserved_zero_3bits** are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this version of this document. Decoders conforming to this version of this document shall ignore values of **ptc_vdmc_reserved_zero_2bits** other than 0.

8.4.4.7 Packing information semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.7 apply.

8.4.4.8 VPS extension semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.8 apply.

8.4.4.9 Packed video extension semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.9 apply.

8.4.4.10 Length alignment semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.4.10 apply.

8.4.4.11 V3C parameter set V-DMC extension semantics

vpve_bmesh_data_substream_codec_id[j] indicates the identifier of the codec used to compress the basemesh data for the atlas with atlas ID *j*. This codec may be identified through the profiles, a component codec mapping SEI message, or through means outside this document.

vpve_bmesh_geometry_3d_bit_depth_minus1[j] indicates the bit depth of vertex coordinates in the basemesh for the atlas with atlas ID *j*. **vpve_bmesh_geometry_3d_bit_depth_minus1[j]** shall be in the range of 0 to 31, inclusive.

vpve_bmesh_geometry_3d_msb_align_flag[j] indicates how the decoded vertex coordinate samples in the basemesh for the atlas with atlas ID j are converted to vertex coordinates samples at the nominal bit depth, as specified in [Annex B](#).

vpve_bmesh_data_attribute_count[j] indicates the number of attributes in the basemesh for the atlas with atlas ID j. **vpve_bmesh_data_attribute_count[j]** shall be in the range of 0 to 255, inclusive. The value indicated by **vpve_bmesh_data_attribute_count** shall be less than or equal to the number of attributes in the basemesh sub-bitstream.

vpve_bmesh_attribute_index[j][i] indicates the index of the basemesh attribute carried in the basemesh bitstream for i-th attribute for atlas with atlas ID j.

vpve_bmesh_attribute_bit_depth_minus1[j][i] indicates the bit depth of the i-th attribute in the basemesh for the atlas with atlas ID j. **vpve_bmesh_attribute_bit_depth_minus1[j]** shall be in the range of 0 to 31, inclusive.

vpve_bmesh_attribute_msb_align_flag[j][i] indicates how the decoded attribute samples in the basemesh for the atlas with atlas ID j and an attribute with index i are converted to attribute samples at the nominal attribute bit depth, as specified in [Annex B](#).

vpve_bmesh_attribute_type[j][i] indicates the attribute type of the i-th attribute signalled through the basemesh for the atlas with atlas ID j. [Table 1](#) describes the list of supported attributes and their relationship with **vpve_bmesh_attribute_type[j][i]**

Table 1 — Basemesh attribute types

vpve_bmesh_attribute_type[j][i]	Identifier	Attribute type
0	ATTR_COLOR	Color
1	ATTR_MATERIAL_ID	Material ID
2	ATTR_TRANSPARENCY	Transparency
3	ATTR_REFLECTANCE	Reflectance
4	ATTR_NORMAL	Normals
5	ATTR_TEXCOORD	Texture coordinate
6	ATTR_FACEGROUP_ID	Face Group ID
7..14	ATTR_RESERVED	Reserved
15	ATTR_UNSPECIFIED	Unspecified

It is a requirement for this version of this document that there shall be no more than one attribute of type ATTR_TEXCOORD defined in the VPS.

vpve_ac_displacement_present_flag[j] equal to 0 indicates that the atlas with atlas ID j does not have arithmetic coded displacement data associated with it. **vpve_ac_displacement_present_flag[j]** equal to 1 indicates that the atlas with atlas ID j shall have arithmetic coded displacement data associated with it.

It is a requirement of bitstream conformance that if **vps_geometry_video_present_flag[j]** is equal to 1 or **pin_geometry_present_flag[j]** is equal to 1 for an atlas with atlas ID j, **vpve_ac_displacement_present_flag[j]** shall be equal to 0 for an atlas with the same atlas ID j.

It is also a requirement of bitstream conformance that if **vpve_ac_displacement_present_flag[j]** is equal to 1 for an atlas with atlas ID j, **vps_geometry_video_present_flag[j]** and **pin_geometry_present_flag[j]** shall be equal to 0 for an atlas with the same atlas ID j.

vpve_consistent_attribute_frame_flag[j] equal to 1 indicates that **vpve_attribute_frame_width[j][i]** and **vpve_attribute_frame_height[j][i]** is associated with all V3C attribute components for the atlas with atlas ID j.

vpve_consistent_attribute_frame_flag[j] equal to 0 indicates that **vpve_attribute_frame_width[j][i]** and **vpve_attribute_frame_height[j][i]** is associated with V3C attribute components with index i for the atlas with atlas ID j.

It is a requirement of bitstream conformance that `vpve_consistent_attribute_frame_flag[ j ]` equal to 1 for the atlas with atlas ID `j` if one or more of the following conditions are true:

- `vps_attribute_video_present_flag[ j ]` equal to 0 and `vps_packed_video_present_flag[ j ]` equal to 0
- `vps_attribute_video_present_flag[ j ]` equal to 0 and `pin_attribute_present_flag[ j ]` equal to 0
- `vps_attribute_video_present_flag[ j ]` equal to 1 and `ai_attribute_count[ j ]` < 2
- `vps_packed_video_present_flag[ j ]` equal to 1 and `pin_attribute_present_flag[ j ]` equal to 1 and `pin_attribute_count[ j ]` < 2

The variable `VpsAttributeNominalFrameCount[ j ]` that represents the number of atlas attribute nominal frames signaled in V3C Parameters Set V-DMC extension for the atlas with atlas ID `j` is computed as follows:

```
if( vpve_consistent_attribute_frame_flag[ j ] ) {
    VpsAttributeNominalFrameCount[ j ] = 1
}
else if ( vps_attribute_video_present_flag[ j ] ) {
    VpsAttributeNominalFrameCount[ j ] = ai_attribute_count[ j ]
}
else if ( vps_packed_video_present_flag[ j ] && pin_attribute_present_flag[ j ] ) {
    VpsAttributeNominalFrameCount[ j ] = pin_attribute_count[ j ]
}
```

`vpve_consistent_attribute_frame_flag[ j ]` shall be equal to 1 in this version of this document.

`vpve_attribute_frame_width[ j ][ i ]` indicates the atlas attribute frame width in terms of integer luma samples for the atlas with atlas ID `j`.

When `vpve_consistent_attribute_frame_flag[ j ]` equal to 0, `vpve_attribute_frame_width[ j ][ i ]` is the atlas frame nominal width that is associated with V3C attribute components with index `i` for the atlas with atlas ID `j`.

When `vpve_consistent_attribute_frame_flag[ j ]` equal to 1, `vpve_attribute_frame_width[ j ][ i ]` is the nominal width that is associated with all V3C attribute components for the atlas with atlas ID `j`.

`vpve_attribute_frame_height[ j ][ i ]` indicates the atlas attribute frame height in terms of integer luma samples for the atlas with atlas ID `j`. This frame height is the nominal height that is associated with V3C attribute components with index `i` for the atlas with atlas ID `j`.

When `vpve_consistent_attribute_frame_flag[ j ]` equal to 0, `vpve_attribute_frame_height[ j ][ i ]` is the atlas frame nominal height that is associated with V3C attribute components with index `i` for the atlas with atlas ID `j`.

When `vpve_consistent_attribute_frame_flag[ j ]` equal to 1, `vpve_attribute_frame_height[ j ][ i ]` is the nominal height that is associated with all V3C attribute components for the atlas with atlas ID `j`.

8.4.4.12 AC displacement information semantics

`acdi_codec_id[ j ]` indicates the mapping index of a codec identifier of the arithmetic codec used to compress the displacement information for atlas ID `j`.

`acdi_msb_align_flag[ j ]` indicates how the decoded displacement vector samples in the arithmetic coded displacement data for the atlas with atlas ID `j` shall be converted to.

`acdi_3d_bit_depth_minus1[ j ]` plus 1 indicates the nominal bit depth of 3D displacement coefficients to which all arithmetic coded displacement data for the atlas with atlas ID `j` shall be converted to. `dsps_geometrydi_disp_3d_bit_depth_minus1` shall be in the range of 0 to 31, inclusive.

`acdi_reserved_16_bits[ j ]` shall be equal to 0 in bitstreams conforming to this version of this document. Other values of `acdi_reserved_16_bits[ j ]` are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this version of this document. Decoders conforming to this version of this document shall ignore values of `acdi_reserved_16_bits[ j ]` other than 0.

8.4.5 NAL unit semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.5 apply.

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

8.4.6.1 Atlas sequence parameter set RBSP semantics

8.4.6.1.1 General atlas sequence parameter set RBSP semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.6.1.1 apply.

8.4.6.1.2 Point local reconstruction information semantics

The specifications in ISO/IEC 23090-5(4E):2025:8.4.6.1.2 apply.

8.4.6.1.3 Atlas sequence parameter set V-DMC extension semantics

asve_subdivision_iteration_count indicates the number of iterations used for the subdivision.

asve_lod_adaptive_subdivision_flag equal to 1 indicates that subdivision method is signaled for each subdivision iteration. **asve_lod_adaptive_subdivision_flag** equal to 0 indicates that the same subdivision method is applied for each subdivision iterations.

asve_subdivision_method[*i*] indicates the identifier of the method to subdivide the meshes associated with the current atlas sequence parameter set at the subdivision with subdivision index equal to *i*. [Table 2](#) describes the list of supported subdivision methods and their relationship with **asve_subdivision_method**.

Table 2 — Subdivision methods list

asve_subdivision_method afve_subdivision_method mdu_subdivision_method	Name of subdivision method
0	MIDPOINT
1	LOOP
2	NORMAL
3	PYTHAG
4..7	RESERVED

asve_edge_based_subdivision_flag equal to 1 specifies that the edge length-based subdivision decision be enabled at the last iteration when midpoint subdivision is used for all iterations. **asve_edge_based_subdivision_flag** equal to 0 specifies that the edge length-based subdivision decision be disabled. When not present, the value of **asve_edge_based_subdivision_flag** is inferred to be 0.

asve_subdivision_min_edge_length indicates the value of the threshold of the edge length-based subdivision decision. When not present, the value of **asve_subdivision_min_edge_length** is inferred to be 0.

asve_1d_displacement_flag equal to 1 specifies that only a single component of the displacement is present in the geometry bitstream. The remaining two components are inferred to be 0. **asve_1d_displacement_flag** equal to 0 specifies that all three components of the displacement are present in the geometry bitstream. The following variable is defined:

$\text{AspsDispComponents} = \text{asve_1d_displacement_flag} ? 1 : 3$

asve_interpolate_subdivided_normals_flag equal to 1 indicates subdivided mesh vertex normals are interpolated on edges between LoD levels after subdivision and before applying displacements. **asve_interpolate_subdivided_normals_flag** equal to 0 indicates subdivided mesh vertex normals are estimated by averaging triangle normals of the subdivided basemesh before applying displacements.

asve_displacement_reference_qp_minus49 plus 49 specifies the initial value of QuantizationParameter for current frame. The value of **asve_displacement_reference_qp_minus49** shall be in the range of -48 to +51, inclusive. When not present **asve_displacement_reference_qp_minus49** is set to be equal to 0.

asve_quantization_parameters_present_flag equal to 1 specifies that the **vdmc_quantization_parameters()** syntax structure is present. **asve_quantization_parameters_present_flag** equal to 0 specifies that the **vdmc_quantization_parameters()** syntax structure is not present.

asve_inverse_quantization_offset_present_flag equal to 1 specifies that the inverse quantization offset is present for compensating the inverse quantized wavelet-transformed coefficients of the displacement. **asve_inverse_quantization_offset_present_flag** equal to 0 specifies that the inverse quantization offset is not present.

`AspsInvQuantOffsetPresentFlag = asve_inverse_quantization_offset_present_flag`

asve_transform_method indicates the identifier of the transform applied to the displacement. [Table 3](#) describes the list of supported transforms and their relationship with **asve_transform_method**. When not present **asve_transform_method** is inferred to be equal to 0.

Table 3 — Transform methods list

asve_transform_method afve_transform_method mdu_transform_method imdu_transform_method	Name of transform method
0	NONE
1	LINEAR_LIFTING
2..7	RESERVED

asve_lifting_offset_present_flag equal to 1 indicates that the lifting offset parameters may be present for at least one meshpatch associated with current atlas. **asve_lifting_offset_present_flag** equal to 0 indicates that the lifting offset parameters are not present for any meshpatch associated with current atlas. When not present **asve_lifting_offset_present_flag** is inferred to be equal to 0.

`AspsLiftingOffsetPresentFlag = asve_lifting_offset_present_flag`

asve_directional_lifting_present_flag equal to 1 indicates that the directional lifting parameters may be present in **vdmc_lifting_transform_parameters** syntax structure. **asve_directional_lifting_present_flag** equal to 0 indicates that the directional lifting parameters are not present in **vdmc_lifting_transform_parameters** syntax structure. When not present **asve_directional_lifting_present_flag** is inferred to be equal to 0.

`AspsDirectionalLiftingPresentFlag = asve_directional_lifting_present_flag`

asve_attribute_information_present_flag equal to 1 indicates that atlas contains information related to attributes by the presence of **asve_consistent_attribute_frame_flag**, **asve_attribute_frame_count**, **asve_attribute_frame_width[i]**, **asve_attribute_frame_height[i]**, **asve_attribute_subtexture_enabled_flag[i]** and **asve_projection_texcoord_enable_flag** parameters in the ASPs. **asve_attribute_information_present_flag** equal to 0 indicates that atlas does not contain any information related to attributes by the absence of **asve_consistent_attribute_frame_flag**, **asve_attribute_frame_count**, **asve_attribute_frame_width[i]**, **asve_attribute_frame_height[i]**, **asve_attribute_subtexture_enabled_flag[i]** and **asve_projection_texcoord_enable_flag** parameters in the ASPs.

asve_consistent_attribute_frame_flag equal to 1 indicates that only one atlas attribute nominal frame is used, and it is associated with all V3C attribute components for the current atlas.

asve_consistent_attribute_frame_flag equal to 0 indicates that there is one atlas attribute nominal frame for each V3C attribute components for the current atlas and **asve_attribute_frame_count** is present.

When **asve_consistent_attribute_frame_flag** is not present then it is inferred to be equal to 1.

It is a requirement of V3C bitstream conformance that the value of `asve_consistent_attribute_frame_flag` shall be equal to the value of `vpve_consistent_attribute_frame_flag[j]`, where `j` is the ID of the current atlas.

asve_attribute_frame_count indicates the number of the atlas attribute nominal frames. It is a requirement of V3C bitstream conformance that, when present, the value of `asve_attribute_frame_count` shall be equal to the value of `VpsAttributeNominalFrameCount[j]`, where `j` is the ID of the current atlas. When not present, the value of `asve_attribute_frame_count` is inferred to be equal to 1.

The variable `AspsAttributeNominalFrameCount` that represents the number of atlas attribute nominal frames signaled in Atlas Sequence Parameters Set V-DMC extension is computed as follows:

```
if( !asve_attribute_information_present_flag ) {
    AspsAttributeNominalFrameCount = 0
} else if ( asve_consistent_attribute_frame_flag ) {
    AspsAttributeNominalFrameCount = 1
} else {
    AspsAttributeNominalFrameCount = asve_attribute_frame_count
}
```

asve_attribute_frame_width[i] indicates the atlas attribute frame width in terms of integer luma samples for the atlas with atlas ID `j`, where `j` is the ID of the current atlas.

When `asve_consistent_attribute_frame_flag` to 0. This frame width is the nominal width that is associated with V3C attribute components with index `i` for the atlas with atlas ID `j`, where `j` is the ID of the current atlas.

When `asve_consistent_attribute_frame_flag` to 1. This frame width is the nominal width that is associated with all V3C attribute components for the atlas with atlas ID `j`.

It is a requirement of V3C bitstream conformance that the value of `asve_attribute_frame_width[i]` shall be equal to the value of `vpve_attribute_frame_width[j][i]`, where `j` is the ID of the current atlas.

asve_attribute_frame_height[i] indicates the atlas attribute frame height in terms of integer luma samples for the atlas with atlas ID `j`, where `j` is the ID of the current atlas.

When `asve_consistent_attribute_frame_flag` to 0. This frame height is the nominal height that is associated with V3C attribute components with index `i` for the atlas with atlas ID `j`, where `j` is the ID of the current atlas.

When `asve_consistent_attribute_frame_flag` to 1. This frame height is the nominal height that is associated with all V3C attribute components for the atlas with atlas ID `j`.

It is a requirement of V3C bitstream conformance that the value of `asve_attribute_frame_height[i]` shall be equal to the value of `vpve_attribute_frame_height[j][i]`, where `j` is the ID of the current atlas.

asve_attribute_subtexture_enabled_flag[i] equal to 0 specifies that the information of the area in the image corresponding to a submesh is not present for the atlas attribute nominal frame with index `i` in meshpatch data unit. `asve_attribute_subtexture_enabled_flag[i]` equal to 1 specifies the information of the area in the image corresponding to a submesh is present for the atlas attribute nominal framewith index `i` in meshpatch data unit.

```
AspsAttributeSubtextureEnabledFlag[ i ] = asve_attribute_subtexture_enabled_flag[ i ]
```

asve_displacement_id_present_flag equal to 1 specifies that the displacement data ID is signalled in the meshpatch data unit and 2D bounding box associated with displacement is not signalled in the meshpatch data unit. `asve_displacement_id_present_flag` equal to 0 specifies that the displacement data ID is not signalled in the meshpatch data unit and the 2D bounding box associated with displacement is signalled in the meshpatch data unit.

```
AspsDisplacementIdPresentFlag = asve_displacement_id_present_flag
```

asve_lod_patches_enable_flag equal to 1 specifies that patches contains data per LoD. `asve_lod_patches_enable_flag` equal to 0 specifies that patch contains data of all lods.

```
AspsLodPatchesEnableFlag = asve_lod_patches_enable_flag
```

asve_packing_method equal to 0 specifies that the displacement component samples are packed in ascending order, **asve_packing_method** equal to 1 specifies that the displacement component samples are packed in descending order.

```
AspsPackingMethod = asve_packing_method
```

asve_projection_texcoord_enable_flag equal to 0 indicates that specifies that the texture coordinates is transmitted in the basemesh. **asve_projection_texcoord_enable_flag** equal to 1 specifies that the texture coordinates may be derived using projection parameters depending on the value of **asve_projection_texcoord_present_flag[i]** corresponding to the submesh with index *i*. When **asve_projection_texcoord_enable_flag** is not present then it shall be equal to 0.

asve_projection_texcoord_mapping_attribute_index_present_flag equals to 1 indicates the **asve_projection_texcoord_mapping_attribute_index** is present. If **asve_projection_texcoord_mapping_attribute_index_present_flag** equals to 0 indicates the **asve_projection_texcoord_mapping_attribute_index** is not present, and the face ID is derived using connected components.

asve_projection_texcoord_mapping_attribute_index indicates the attribute index of the basemesh in nominal format, which will be used to map a set of faces to a sub-patch.

asve_projection_texcoord_output_bit_depth_minus1 indicates the bit depth of the texture coordinate generated by orthoAtlas. **asve_projection_texcoord_output_bit_depth_minus1** shall be in the range of 0 to 31, inclusive.

asve_projection_texcoord_bbox_bias_enable_flag equals to 1 indicates the coordinates of the projected sub-patch 3D bounding box and the sub-patch size are present in the sub-patch-related syntax structures. **asve_projection_texcoord_bbox_bias_enable_flag** equals to 0 indicates that the coordinates of the projected sub-patch 3D bounding box and the sub-patch size are not present in the sub-patch-related syntax structures.

```
AspsProjectionTexcoordBboxBiasEnableFlag =  
asve_projection_texcoord_bbox_bias_enable_flag
```

asve_projection_texcoord_upscale_factor_minus1 plus 1 indicate the upscale value of the scaling factor variable **AspsTexcoordProjectionUpscaleFactor** that is used as numerator of texture coordinate derivation from geometry projection, and computed as follows:

```
AspsTexcoordProjectionUpscaleFactor = asve_projection_texcoord_upscale_factor_minus1 + 1  
AspsTexcoordProjectionScaleFactor =
```

```
AspsTexcoordProjectionUpscaleFactor ÷ AspsTexcoordProjectionDownscaleFactor
```

asve_projection_texcoord_log2_downscale_factor indicate the logarithm of downscale to base of 2 of the scaling factor variable **AspsTexcoordProjectionDownscaleFactor** that is used as denominator of texture coordinate derivation from geometry projection, and computed as follows:

```
AspsTexcoordProjectionDownscaleFactor = 2asve_projection_texcoord_log2_downscale_factor
```

asve_projection_raw_textcoord_present_flag equal to 1 specifies that a RAW subpatch may be present in the texture_projection_information. **asve_projection_raw_textcoord_present_flag** equal to 0 specified that a RAW sub patch is not signalled in the bitstream.

asve_projection_raw_textcoord_bitdepth_minus1 indicates the bit depth of texture coordinates in sub-patch information syntax. **asve_projection_raw_textcoord_bitdepth_minus1** shall be in the range of 0 to 15, inclusive.

asve_vdmc_vui_parameters_present_flag equal to 1 specifies that the **vdmc_vui_parameters()** syntax structure, as specified in [Annex G](#), is present. **asve_vdmc_vui_parameters_present_flag** equal to 0 specifies that the **vdmc_vui_parameters()** syntax structure, as specified in [Annex G](#), is not present.

8.4.6.1.4 Quantization parameters semantics

vqp_lod_quantization_flag[qpIndex] equal to 1 indicates that the quantization parameter will be sent per level-of-detail using delta coding. **vqp_lod_quantization_flag[qpIndex]** equal to 0 indicates that the