

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

**Coding of moving pictures and audio**

**Convenorship: Japan (JISC)**

**ISO/IEC JTC 1/SC 29/WG 11 N19523**

**Document type: Approved WG 11 document**

**Title: WD of V3C and V-PCC (2nd Edition)**

**Status: Approved**

**Date of document: 2020-09-28**

**Source: Convenor, ISO/IEC JTC 1/SC 29/WG 11**

**No. of Pages: 361**

**Email of acting convenor:** ostermann@tnt.uni-hannover.de

**Committee URL:** <http://isotc.iso.org/livelink/livelink/open/jtc1sc29>

ISO/IEC 23090-5:2020(2E)

ISO/IEC JTC 1/SC 29/WG 11

Secretariat: JISC

**Information technology — Coded Representation of Immersive Media — Part 5: Visual Volumetric Video-based Coding (V3C) and Video-based Point Cloud Compression (V-PCC)**

WD stage

**Warning for WDs and CDs**

This document is not an ISO International Standard. It is distributed for review and comment. It is subject to change without notice and may not be referred to as an International Standard.

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.

© ISO/IEC 2020

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

Fax: +41 22 749 09 47

Email: copyright@iso.org

Website: www.iso.org

Published in Switzerland

Contents

[Foreword x](#_Toc52181571)

[Introduction xi](#_Toc52181572)

[1 Scope 1](#_Toc52181573)

[2 Normative references 1](#_Toc52181574)

[3 Terms and definitions 1](#_Toc52181575)

[4 Abbreviated terms 16](#_Toc52181576)

[5 Conventions 18](#_Toc52181577)

[5.1 General 18](#_Toc52181578)

[5.2 Arithmetic operators 18](#_Toc52181579)

[5.3 Logical operators 18](#_Toc52181580)

[5.4 Relational operators 18](#_Toc52181581)

[5.5 Bit-wise operators 18](#_Toc52181582)

[5.6 Assignment operators 19](#_Toc52181583)

[5.7 Other operators 19](#_Toc52181584)

[5.8 Mathematical functions 19](#_Toc52181585)

[5.9 Order of operation precedence 20](#_Toc52181586)

[5.10 Variables, syntax elements, and tables 21](#_Toc52181587)

[5.11 Text description of logical operations 23](#_Toc52181588)

[5.12 Processes 24](#_Toc52181589)

[6 Overall V3C characteristics, decoding operations, and post-decoding processes 25](#_Toc52181590)

[6.1 V3C characteristics 25](#_Toc52181591)

[6.2 V3C bitstream characteristics, decoding operations, and post-decoding processes 27](#_Toc52181592)

[7 Bitstream format, partitioning, and scanning processes 28](#_Toc52181593)

[7.1 General 28](#_Toc52181594)

[7.2 V3C bitstream formats 28](#_Toc52181595)

[7.3 NAL bitstream formats 29](#_Toc52181596)

[7.4 Partitioning of atlas frames into tiles 29](#_Toc52181597)

[7.5 Tile partition scanning process 30](#_Toc52181598)

[8 Syntax and semantics 31](#_Toc52181599)

[8.1 Method of specifying syntax in tabular form 31](#_Toc52181600)

[8.2 Specification of syntax functions and descriptors 32](#_Toc52181601)

[8.3 Syntax in tabular form 34](#_Toc52181602)

[8.3.1 General 34](#_Toc52181603)

[8.3.2 V3C unit syntax 35](#_Toc52181604)

[8.3.3 Byte alignment syntax 36](#_Toc52181605)

[8.3.4 V3C parameter set syntax 37](#_Toc52181606)

[8.3.5 NAL unit syntax 40](#_Toc52181607)

[8.3.6 Raw byte sequence payloads, trailing bits, and byte alignment syntax 41](#_Toc52181608)

[8.3.7 Atlas tile data unit syntax 47](#_Toc52181609)

[8.3.8 Supplemental enhancement information message syntax 50](#_Toc52181610)

[8.4 Semantics 51](#_Toc52181611)

[8.4.1 General 51](#_Toc52181612)

[8.4.2 V3C unit semantics 51](#_Toc52181613)

[8.4.3 Byte alignment semantics 54](#_Toc52181614)

[8.4.4 V3C parameter set semantics 54](#_Toc52181615)

[8.4.5 NAL unit semantics 62](#_Toc52181616)

[8.4.6 Raw byte sequence payloads, trailing bits, and byte alignment semantics 71](#_Toc52181617)

[8.4.7 Atlas tile data unit semantics 85](#_Toc52181618)

[8.4.8 Supplemental enhancement information message semantics 94](#_Toc52181619)

[9 Decoding process 94](#_Toc52181620)

[9.1 General decoding process 94](#_Toc52181621)

[9.2 Atlas data decoding process 95](#_Toc52181622)

[9.2.1 General atlas data decoding process 95](#_Toc52181623)

[9.2.2 Decoding process for a coded atlas frame 96](#_Toc52181624)

[9.2.3 Atlas NAL unit decoding process 97](#_Toc52181625)

[9.2.4 Atlas tile header decoding process 97](#_Toc52181626)

[9.2.5 Decoding process for patch data units 102](#_Toc52181627)

[9.2.6 Decoding process of the block to patch map 117](#_Toc52181628)

[9.2.7 Conversion of tile level patch information to atlas level patch information 117](#_Toc52181629)

[9.3 Occupancy video decoding process 120](#_Toc52181630)

[9.4 Geometry video decoding process 121](#_Toc52181631)

[9.5 Attribute video decoding process 123](#_Toc52181632)

[9.6 Packed video decoding process 126](#_Toc52181633)

[9.7 Common atlas data decoding process 126](#_Toc52181634)

[9.8 Sub-bitstream extraction process 127](#_Toc52181635)

[9.8.1 General 127](#_Toc52181636)

[9.8.2 V3C unit extraction 127](#_Toc52181637)

[9.8.3 NAL unit extraction process 128](#_Toc52181638)

[10 Pre-reconstruction process 128](#_Toc52181639)

[11 Reconstruction process 128](#_Toc52181640)

[12 Post-reconstruction process 128](#_Toc52181641)

[13 Adaptation process 128](#_Toc52181642)

[14 Parsing process 129](#_Toc52181643)

[14.1 General 129](#_Toc52181644)

[14.2 Parsing process for 0-th order Exp-Golomb codes 129](#_Toc52181645)

[14.2.1 General 129](#_Toc52181646)

[14.2.2 Mapping process for signed Exp-Golomb codes 130](#_Toc52181647)

[Annex A (normative) Profiles, tiers, and levels 132](#_Toc52181648)

[A.1 Overview of profiles, tiers, and levels 132](#_Toc52181649)

[A.2 Profile, tier, and level structure 133](#_Toc52181650)

[A.3 CodecGroup profile components 135](#_Toc52181651)

[A.4 Toolset profile components 137](#_Toc52181652)

[A.5 Reconstruction profile components 137](#_Toc52181653)

[A.6 Tiers and levels 138](#_Toc52181654)

[A.6.1 General tier and level limits 138](#_Toc52181655)

[A.6.2 Profile specific level limits for the V3C dynamic profiles 138](#_Toc52181656)

[A.7 Decoder instantiations 143](#_Toc52181657)

[Annex B (informative) Post-decoding conversion to nominal video formats 145](#_Toc52181658)

[B.1 General 145](#_Toc52181659)

[B.2 Nominal format conversion 147](#_Toc52181660)

[B.3 Conversion operations 154](#_Toc52181661)

[B.3.1 Map Extraction 154](#_Toc52181662)

[B.3.2 Bit depth conversion 156](#_Toc52181663)

[B.3.3 Resolution conversion 157](#_Toc52181664)

[B.3.4 Output composition time conversion 157](#_Toc52181665)

[B.3.4.1 Output order conversion 157](#_Toc52181666)

[B.3.4.2 Atlas composition time alignment 158](#_Toc52181667)

[B.3.5 Attribute dimension packing 160](#_Toc52181668)

[B.3.5.1 Attribute dimension packing process 160](#_Toc52181669)

[B.3.5.2 Attribute dimension packing process for an auxiliary attribute 161](#_Toc52181670)

[B.3.6 Chroma up-sampling 162](#_Toc52181671)

[B.3.7 Geometry map synthesis process 163](#_Toc52181672)

[B.3.8 Attribute map synthesis process 164](#_Toc52181673)

[B.3.9 Occupancy thresholding process 165](#_Toc52181674)

[B.4 Unpacking process of a decoded packed video 166](#_Toc52181675)

[Annex C (informative) V3C sample stream format 171](#_Toc52181676)

[C.1 General 171](#_Toc52181677)

[C.2 Sample stream V3C unit syntax and semantics 171](#_Toc52181678)

[C.2.1 Sample stream V3C header syntax 171](#_Toc52181679)

[C.2.2 Sample stream V3C unit syntax 171](#_Toc52181680)

[C.2.3 Sample stream V3C header semantics 171](#_Toc52181681)

[C.2.4 Sample stream V3C unit semantics 171](#_Toc52181682)

[Annex D (normative) NAL sample stream format 172](#_Toc52181683)

[D.1 General 172](#_Toc52181684)

[D.2 Sample stream NAL unit syntax and semantics 172](#_Toc52181685)

[D.2.1 Sample stream NAL header syntax 172](#_Toc52181686)

[D.2.2 Sample stream NAL unit syntax 172](#_Toc52181687)

[D.2.3 Sample stream NAL header semantics 172](#_Toc52181688)

[D.2.4 Sample stream NAL unit semantics 172](#_Toc52181689)

[Annex E (normative) Atlas hypothetical reference decoder 174](#_Toc52181690)

[E.1 General 174](#_Toc52181691)

[E.2 Operation of coded atlas frame buffer 179](#_Toc52181692)

[E.2.1 General 179](#_Toc52181693)

[E.2.2 Timing of decoding unit arrival 179](#_Toc52181694)

[E.2.3 Timing of decoding unit removal and decoding of decoding unit 182](#_Toc52181695)

[E.3 Operation of the decoded atlas frame buffer 184](#_Toc52181696)

[E.3.1 General 184](#_Toc52181697)

[E.3.2 Removal of atlas frames from the DAB before decoding of the current atlas frame 184](#_Toc52181698)

[E.3.3 Atlas frame output 185](#_Toc52181699)

[E.3.4 Current decoded atlas frame marking and storage 185](#_Toc52181700)

[E.3.5 Removal of atlas frames from the DAB after decoding of the current atlas frame 186](#_Toc52181701)

[E.4 Bitstream conformance 186](#_Toc52181702)

[E.5 Decoder conformance 187](#_Toc52181703)

[E.5.1 General 187](#_Toc52181704)

[E.5.2 Operation of the output order DAB 189](#_Toc52181705)

[Annex F (normative) Supplemental enhancement information 191](#_Toc52181706)

[F.1 General 191](#_Toc52181707)

[F.2 SEI payload syntax 192](#_Toc52181708)

[F.2.1 General SEI message syntax 192](#_Toc52181709)

[F.2.2 Filler payload SEI message syntax 194](#_Toc52181710)

[F.2.3 User data registered by Recommendation ITU-T T.35 SEI message syntax 194](#_Toc52181711)

[F.2.4 User data unregistered SEI message syntax 195](#_Toc52181712)

[F.2.5 Recovery point SEI message syntax 195](#_Toc52181713)

[F.2.6 No reconstruction SEI message syntax 195](#_Toc52181714)

[F.2.7 Reserved SEI message syntax 195](#_Toc52181715)

[F.2.8 SEI manifest SEI message syntax 195](#_Toc52181716)

[F.2.9 SEI prefix indication SEI message syntax 195](#_Toc52181717)

[F.2.10 Active sub-bitstreams SEI message syntax 196](#_Toc52181718)

[F.2.11 Component codec mapping SEI message syntax 196](#_Toc52181719)

[F.2.12 Volumetric annotation SEI message family syntax 196](#_Toc52181720)

[F.2.13 Buffering period SEI message syntax 201](#_Toc52181721)

[F.2.14 Atlas frame timing SEI message syntax 202](#_Toc52181722)

[F.2.15 Viewport SEI messages family syntax 202](#_Toc52181723)

[F.2.16 Decoded Atlas Information Hash SEI message syntax 203](#_Toc52181724)

[F.2.16.1 Decoded high level hash unit syntax 204](#_Toc52181725)

[F.2.16.2 Decoded atlas hash unit syntax 204](#_Toc52181726)

[F.2.16.3 Decoded atlas b2p hash unit syntax 204](#_Toc52181727)

[F.2.16.4 Decoded atlas tile hash unit syntax 204](#_Toc52181728)

[F.2.16.5 Decoded atlas tile b2p hash unit syntax 205](#_Toc52181729)

[F.2.17 Time code SEI message syntax 205](#_Toc52181730)

[F.2.18 Packed independent regions SEI payload syntax 205](#_Toc52181731)

[F.3 SEI payload semantics 207](#_Toc52181732)

[F.3.1 General SEI payload semantics 207](#_Toc52181733)

[F.3.2 Filler payload SEI message semantics 209](#_Toc52181734)

[F.3.3 User data registered by Recommendation ITU-T T.35 SEI message semantics 209](#_Toc52181735)

[F.3.4 User data unregistered SEI message semantics 209](#_Toc52181736)

[F.3.5 Recovery point SEI message semantics 210](#_Toc52181737)

[F.3.6 No reconstruction SEI message semantics 211](#_Toc52181738)

[F.3.7 Reserved SEI message semantics 211](#_Toc52181739)

[F.3.8 SEI manifest SEI message semantics 211](#_Toc52181740)

[F.3.9 SEI prefix indication SEI message semantics 213](#_Toc52181741)

[F.3.10 Active sub-bitstreams SEI message semantics 214](#_Toc52181742)

[F.3.11 Component codec mapping SEI message semantics 215](#_Toc52181743)

[F.3.12 Volumetric annotation SEI messages family syntax 215](#_Toc52181744)

[F.3.13 Buffering period SEI message semantics 228](#_Toc52181745)

[F.3.14 Atlas frame timing SEI message semantics 232](#_Toc52181746)

[F.3.15 Viewport SEI messages family semantics 234](#_Toc52181747)

[F.3.16 Decoded Atlas Information Hash SEI message semantics 236](#_Toc52181748)

[F.3.16.1 General 236](#_Toc52181749)

[F.3.17 Time code SEI message semantics 248](#_Toc52181750)

[F.3.18 Packed independent regions SEI payload semantics 250](#_Toc52181751)

[Annex G (informative) Volumetric usability information 251](#_Toc52181752)

[G.1 General 251](#_Toc52181753)

[G.2 VUI syntax 252](#_Toc52181754)

[G.2.1 VUI parameters syntax 252](#_Toc52181755)

[G.2.2 HRD parameters syntax 252](#_Toc52181756)

[G.2.3 Sub-layer HRD parameters syntax 254](#_Toc52181757)

[G.2.4 Maximum coded video resolution syntax 254](#_Toc52181758)

[G.2.5 Coordinate system parameters syntax 254](#_Toc52181759)

[G.3 VUI semantics 254](#_Toc52181760)

[G.3.1 VUI parameters semantics 254](#_Toc52181761)

[G.3.2 HRD parameters semantics 258](#_Toc52181762)

[G.3.3 HRD sub-layer parameters semantics 259](#_Toc52181763)

[G.3.4 Maximum coded video resolution semantics 260](#_Toc52181764)

[G.3.5 Coordinate system parameters semantics 261](#_Toc52181765)

[Annex H (normative) Video-based Point Cloud Coding 263](#_Toc52181766)

[H.1 Scope 263](#_Toc52181767)

[H.2 Normative References 263](#_Toc52181768)

[H.3 Terms and definitions 263](#_Toc52181769)

[H.4 Abbreviated terms 263](#_Toc52181770)

[H.5 Conventions 263](#_Toc52181771)

[H.6 Overall V-PCC characteristics, decoding operations, and post-decoding processes 264](#_Toc52181772)

[H.6.1 V3C characteristics 264](#_Toc52181773)

[H.6.2 V3C bitstream characteristics, decoding operations, and post-decoding processes 264](#_Toc52181774)

[H.7 Bitstream format, partitioning, and scanning process 265](#_Toc52181775)

[H.8 Syntax and Semantics 265](#_Toc52181776)

[H.8.1 Method of specifying syntax in tabular form 265](#_Toc52181777)

[H.8.2 Specification of syntax functions and descriptors 265](#_Toc52181778)

[H.8.3 Syntax in tabular form 265](#_Toc52181779)

[H.8.4 Semantics 267](#_Toc52181780)

[H.9 Decoding process 271](#_Toc52181781)

[H.9.1 General decoding process 271](#_Toc52181782)

[H.9.2 Atlas data decoding process 271](#_Toc52181783)

[H.9.3 Occupancy video decoding process 278](#_Toc52181784)

[H.9.4 Geometry video decoding process 278](#_Toc52181785)

[H.9.5 Attribute video decoding process 279](#_Toc52181786)

[H.9.6 Sub-bitstream extraction process 279](#_Toc52181787)

[H.10 Pre-reconstruction process 279](#_Toc52181788)

[H.10.2 Occupancy synthesis process 279](#_Toc52181789)

[H.11 Reconstruction process 287](#_Toc52181790)

[H.11.1 General 287](#_Toc52181791)

[H.11.2 Reconstruction of points for PROJECTED patches 291](#_Toc52181792)

[H.11.3 Reconstruction for EOM patches 299](#_Toc52181793)

[H.11.4 Reconstruction of RAW patches 304](#_Toc52181794)

[H.11.5 Conversion from atlas coordinates to 3D coordinates 307](#_Toc52181795)

[H.11.6 Conversion of 3D coordinates for projection planes at 45 degrees 307](#_Toc52181796)

[H.12 Post-reconstruction process 308](#_Toc52181797)

[H.12.1 General 308](#_Toc52181798)

[H.12.2 Geometry smoothing process 309](#_Toc52181799)

[H.12.3 Attribute smoothing process 312](#_Toc52181800)

[H.12.4 Identification of the geometry and attribute center grids 315](#_Toc52181801)

[H.12.5 Identification of boundary points 319](#_Toc52181802)

[H.12.6 Identification of 2 × 2 × 2 cells corresponding to a reconstructed point position 320](#_Toc52181803)

[H.12.7 Trilinear filtering 321](#_Toc52181804)

[H.13 Adaptation process 328](#_Toc52181805)

[H.14 Parsing process 328](#_Toc52181806)

[H.15 Profiles, tiers, and levels 328](#_Toc52181807)

[H.15.1 Overview of profiles, tiers, and levels 328](#_Toc52181808)

[H.15.2 Profile, tier, and level structure 328](#_Toc52181809)

[H.15.3 CodecGroup profile components 329](#_Toc52181810)

[H.15.4 Toolset profile components 329](#_Toc52181811)

[H.15.5 Reconstruction profile components 331](#_Toc52181812)

[H.15.6 Tier and Levels 334](#_Toc52181813)

[H.16 Post-decoding conversion to nominal video formats 334](#_Toc52181814)

[H.17 V3C Sample Stream Format 334](#_Toc52181815)

[H.18 NAL Sample Stream Format 334](#_Toc52181816)

[H.19 Atlas Hypothetical Reference Decoder 334](#_Toc52181817)

[H.20 Supplemental Enhancement Information 334](#_Toc52181818)

[H.20.1 General 334](#_Toc52181819)

[H.20.2 SEI payload syntax 334](#_Toc52181820)

[H.20.2.1 General SEI message syntax 334](#_Toc52181821)

[H.20.2.2 Filler payload SEI message syntax 334](#_Toc52181822)

[H.20.2.3 User data registered by Recommendation ITU-T T.35 SEI message syntax 334](#_Toc52181823)

[H.20.2.4 User data unregistered SEI message syntax 334](#_Toc52181824)

[H.20.2.5 Recovery point SEI message syntax 335](#_Toc52181825)

[H.20.2.6 No reconstruction SEI message syntax 335](#_Toc52181826)

[H.20.2.7 Reserved SEI message syntax 335](#_Toc52181827)

[H.20.2.8 SEI manifest SEI message syntax 335](#_Toc52181828)

[H.20.2.9 SEI prefix indication SEI message syntax 335](#_Toc52181829)

[H.20.2.10 Active sub-bitstreams SEI message syntax 335](#_Toc52181830)

[H.20.2.11 Component codec mapping SEI message syntax 335](#_Toc52181831)

[H.20.2.12 Volumetric annotation SEI message family syntax 335](#_Toc52181832)

[H.20.2.13 Buffering period SEI message syntax 335](#_Toc52181833)

[H.20.2.14 Atlas frame timing SEI message syntax 335](#_Toc52181834)

[H.20.2.15 Viewport SEI messages family syntax 335](#_Toc52181835)

[H.20.2.16 Decoded Atlas Information Hash SEI message syntax 335](#_Toc52181836)

[H.20.2.17 Attribute transformation parameters SEI message syntax 336](#_Toc52181837)

[H.20.2.18 Occupancy synthesis SEI message syntax 336](#_Toc52181838)

[H.20.2.19 Geometry smoothing SEI message syntax 336](#_Toc52181839)

[H.20.2.20 Attribute smoothing SEI message syntax 337](#_Toc52181840)

[H.20.3 SEI payload semantics 337](#_Toc52181841)

[H.20.3.1 General SEI payload semantics 337](#_Toc52181842)

[H.20.3.2 Filler payload SEI message semantics 338](#_Toc52181843)

[H.20.3.3 User data registered by Recommendation ITU-T T.35 SEI message semantics 338](#_Toc52181844)

[H.20.3.4 User data unregistered SEI message semantics 338](#_Toc52181845)

[H.20.3.5 Recovery point SEI message semantics 338](#_Toc52181846)

[H.20.3.6 No display SEI message semantics 338](#_Toc52181847)

[H.20.3.7 Reserved SEI message semantics 338](#_Toc52181848)

[H.20.3.8 SEI manifest SEI message semantics 338](#_Toc52181849)

[H.20.3.9 SEI prefix indication SEI message semantics 338](#_Toc52181850)

[H.20.3.10 Active sub-bitstreams SEI message semantics 338](#_Toc52181851)

[H.20.3.11 Component codec mapping SEI message semantics 338](#_Toc52181852)

[H.20.3.12 Volumetric annotation SEI messages family syntax 338](#_Toc52181853)

[H.20.3.13 Buffering period SEI message semantics 338](#_Toc52181854)

[H.20.3.14 Atlas frame timing SEI message semantics 338](#_Toc52181855)

[H.20.3.15 Viewport SEI messages family semantics 338](#_Toc52181856)

[H.20.3.16 Decoded Atlas Information Hash SEI message semantics 339](#_Toc52181857)

[H.21 Volumetric Usability Information 348](#_Toc52181858)

[Bibliography 349](#_Toc52181859)

Foreword

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

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

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

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

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

Introduction

Advances in 3D capturing and rendering technologies have unleashed a new wave of innovation in Virtual/Augmented/Mixed reality (VR/AR/MR) content creation and communication, of which visual volumetric video is an integral part.

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

Visual volumetric frames can be coded by converting the 3D volumetric information into a collection of 2D images and associated data. The converted 2D images can be coded using widely available video and image coding specifications, such as ISO/IEC 14492-10 and ISO/IEC 23008-2 and the associated data can be coded with mechanisms specified in this document. The coded images and the associated data can then be decoded and used to reconstruct the 3D volumetric information. This document specifies a generic mechanism for visual volumetric video coding, i.e. visual volumetric video-based coding. The generic mechanism may be used by applications targeting volumetric content, such as point clouds, immersive video with depth, mesh representations of visual volumetric frames, etc.

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

**Information technology — Coded Representation of Immersive Media — Part 5: Visual Volumetric Video-based Coding (V3C) and Video-based Point Cloud Compression (V-PCC)**

# Scope

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

# Normative references

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

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

IETF RFC 1321 (in force), *The MD5 Message-Digest Algorithm*

IETF RFC 5646 (in force), *Tags for Identifying Languages*

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

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

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

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

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

ISO/IEC DIS 23090-3, *Information technology —* *Coded representation of immersive media —* *Part 3: Versatile video coding[[1]](#footnote-2)*

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

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

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

# Terms and definitions

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

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

— ISO Online browsing platform: available at [https://www.iso.org/obp](https://www.iso.org/obp/ui)

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

3D bounding box

volume defined as a cuboid solid having six rectangular faces placed at right angles



associated non-ACL NAL unit

*non-ACL NAL unit* (3.86) that is associated with an *ACL NAL unit* (3.6) for the purpose of decoding or other operations specified by this document



associated ACL NAL unit

preceding *ACL NAL unit* (3.6) in decoding order for a *non-ACL NAL unit* (3.86) with nal\_unit\_type equal to NAL\_EOS, NAL\_EOB, NAL\_FD, NAL\_SUFFIX\_NSEI, or NAL\_SUFFIX\_ESEI, or in the ranges of NAL\_RSV\_NACL\_48..NAL\_RSV\_NACL\_52 or NAL\_UNSPEC\_53..NAL\_UNSPEC\_63; or otherwise the next *ACL NAL unit* (3.6) in decoding order



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



atlas bitstream

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



atlas coding layer NAL unit

ACL NAL unit

collective term for coded atlas tile layer *NAL units* (3.84) and the subset of *NAL units* (3.84) that have reserved values of nal\_unit\_type that are classified as being of type class equal to ACL in this document

atlas coordinates

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



atlas frame

2D rectangular array of atlas samples onto which *patches* (3.93) are projected and additional information related to the *patches* (3.93), corresponding to a *volumetric frame* (3.148)

atlas frame parameter set

AFPS

*syntax structure* (3.129) containing *syntax elements* (3.128) that apply to zero or more entire coded *atlas frames* (3.8) as determined by the content of a syntax element (3.128) found in each tile header



atlas sample

position on the rectangular frame onto which *patches* (3.93) that are associated with an *atlas* (3.4) are projected

atlas sequence

collection of *atlas frames* (3.8)

atlas sequence parameter set

ASPS

*syntax structure* (3.129) containing *syntax elements* (3.128) that apply to zero or more entire *coded atlas sequences* (3.35) as determined by the content of a syntax element (3.128) found in the AFPS referred to by a syntax element found in each tile header



atlas sub-bitstream

extracted *sub-bitstream* (3.117) from the *V3C bitstream* (3.138) containing a part of an atlas NAL bitstream



atlas unit

set of *NAL units* (3.84) that contain all *ACL NAL units* (3.6) of a *coded atlas* (3.34) and their *associated non-ACL NAL units* (3.2)



attribute

scalar or vector property optionally associated with each point in a *volumetric frame* (3.148) such as colour, reflectance, surface normal, time stamps, material ID, etc.



attribute access unit

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



attribute frame

2D rectangular array created through the aggregation of *patches* (3.93) containing values of a specific *attribute* (3.14)



attribute map

*attribute frame* (3.17) containing attribute *patch* (3.93) information projected at a particular depth indicated by the corresponding *geometry map* (3.65)



auxiliary attribute frame

2D rectangular array that is associated with *RAW patches* (3.111) and *EOM* *patches* (3.57), and contains values of a specific *attribute* (3.15)



auxiliary geometry frame

2D rectangular array that is associated with *RAW patches* (3.111) and contains *geometry* (3.62) values

auxiliary video component

*video component* (3.135) indicated as being of auxiliary type through an appropriate flag in the VPS (3.144) and containing data only associated with *RAW patches* (3.111) and/or *EOM* *patches* (3.57)

auxiliary video sub-bitstream

video sub-bitstream (3.137) indicated as being of auxiliary type through an appropriate flag in the VPS (3.144) and containing data only associated with *RAW patches* (3.111) and/or *EOM* *patches* (3.57)



bitstream

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



byte

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

broken link access access unit  
BLA access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *BLA coded atlas* (3.26)



broken link access coded atlas  
BLA coded atlas

*IRAP coded atlas* (3.79) frame for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_BLA\_W\_LP, NAL\_GBLA\_W\_LP, NAL\_BLA\_W\_RADL NAL\_GBLA\_W\_RADL, NAL\_BLA\_N\_LP or NAL\_GBLA\_N\_LP.

NOTE – A BLA coded atlas does not use inter prediction in its decoding process, and could be the first coded atlas in the bitstream in decoding order, or could appear later in the bitstream. Each BLA coded atlas begins a new CAS, and has the same effect on the decoding process as an instantaneous decoding refresh (IDR) coded atlas. However, a BLA coded atlas contains syntax elements that specify a non-empty DAB. When a BLA coded atlas frame for which each ACL NAL unit has nal\_unit\_type equal to NAL\_BLA\_W\_LP or NAL\_GBLA\_W\_LP, it may have associated random access skipped leading (RASL) coded atlas frames, which are not output by the decoder and may not be decodable, as they may contain references to atlas frames that are not present in the bitstream. When a BLA coded atlas frame for which each ACL NAL unit has nal\_unit\_type equal to NAL\_BLA\_W\_LP or NAL\_GBLA\_W\_LP, it may also have associated RADL coded atlas frames, which are specified to be decoded. When a BLA coded atlas frame for which each ACL NAL unit has nal\_unit\_type equal to NAL\_BLA\_W\_RADL or NAL\_GBLA\_W\_RADL, it does not have associated RASL coded atlas frames but may have associated random access decodable leading (RADL) coded atlas frames. When a BLA coded atlas frame for which each ACL NAL unit has nal\_unit\_type equal to NAL\_BLA\_N\_LP or NAL\_GBLA\_N\_LP, it does not have any associated leading coded atlas frames.



byte-aligned

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

byte-aligned position

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



byte-aligned byte

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



cartesian coordinates

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

clean random access access unit  
CRA access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *CRA coded atlas* (3.32)



clean random access coded atlas

CRA coded atlas

*IRAP coded atlas* (3.79)for which each *ACL* *NAL unit* (3.6) has nal\_unit\_type equal to NAL\_CRA or NAL\_GCRA

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

codec

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



coded atlas

coded atlas frame

coded representation of an *atlas* (3.4)



coded atlas access unit

set of atlas *NAL units* (3.84) that are associated with each other according to a specified classification rule, are consecutive in decoding order, and contain all atlas *NAL units* (3.84) pertaining to one particular output time



coded atlas sequence

CAS

sequence of *coded atlas access units* (3.35)*,* in decoding order, of an *IRAP coded atlas access unit* (3.80), followed by zero or more *coded atlas access units* (3.35)that are not *IRAP coded atlas access units* (3.80), including all subsequent *access units* (3.35)up to but not including any subsequent *coded atlas access unit* (3.35) that is an *IRAP coded atlas access unit* (3.80)



coded common atlas access unit

set of common atlas *non-ACL NAL units* (3.86) that are associated with each other according to a specified classification rule, are consecutive in decoding order, and contain all common atlas *NAL units* (3.84) pertaining to one particular output time



coded common atlas frame

coded representation of a *common* *atlas frame* (3.45)

coded common atlas sequence

CCAS

sequence of *coded common atlas access units* (3.37)*,* in decoding order, of an *IRAP coded common atlas access unit* (3.82), followed by zero or more *coded common atlas access units* (3.37)that are not *IRAP coded common atlas access units* (3.82), including all subsequent *access units* (3.37)up to but not including any subsequent *coded common atlas access unit* (3.37) that is an *IRAP coded common atlas access unit* (3.82)

coded volumetric frame

collection of coded representations of an *atlas* (3.4), *occupancy* (3.91), *geometry access unit* (3.63), and, for each available *attribute* (3.15), *attribute access unit* (3.16),pertaining to one particular time instance



coded V3C sequence

CVS

sequence of *V3C sub-bitstreams* (3.143) identified and separated by appropriate delimiters, required to start with a *VPS* (3.144), included in at least one *V3C unit* (3.145) or provided through external means, and contains one or more V3C units that can be factored into *V3C composition units* (3.141), where the first V3C composition unit is a *V3C IRAP composition unit* (3.142)



coded representation

data element as represented in its coded form

coded sub-bitstream sequence

*sub-bitstream IRAP composition unit* (3.119) followed by zero or more *sub-bitstream composition units* (3.118)

coded V3C component

coded representation of a *V3C component* (3.139)



common atlas frame

CAF

common information (e.g. projection parameters) that applies to all *atlas frames* (3.8) in a *volumetric frame* (3.148)

common atlas sequence

collection of *common atlas frames* (3.45)

common atlas sub-bitstream

extracted *sub-bitstream* (3.117) from the *V3C bitstream* (3.138) containing a part of a common atlas NAL bitstream

component bitstream

*bitstream* (3.23) representing a *V3C component* (3.139)

component sub-bitstream

portion of *component bitstream* (3.45)



composition time

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

composition time index

index to an ordered list of *composition times* (3.50)

composition unit

partition of a *bitstream* (3.23) that has a certain *composition time* (3.50)

decoder under test

DUT

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

decoding unit

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

enhanced occupancy mode

EOM

patch coding mode where a *patch* (3.93) is associated with enhanced occupancy information



EOM coded points

coded representation of 3D points located at intermediate depth positions for which *geometry* (3.62) values are stored as codewords in the *occupancy frame* (3.92) and their corresponding attributes values are stored in additional patches, referred to as *EOM patches* (3.57), in the *attribute frames* (3.17)



EOM patch

*patch* (3.93) with *patch mode* (3.95) equal to I\_EOM, P\_EOM, or P\_SKIP associated with *EOM coded points* (3.56)

EOM patch type

*patch type* (3.96) indicating an *EOM patch* (3.57)



essential supplemental enhancement information

ESEI

SEI (3.126) that is deemed as essential by the decoding process and should not be ignored or discarded



essential supplemental enhancement information NAL unit

ESEI NAL unit

*NAL unit* (3.84) corresponding to an ESEI (3.59) and has nal\_unit\_type equal to NAL\_PREFIX\_ESEI or NAL\_SUFFIX\_ESEI

flag

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



geometry

set of *cartesian coordinates* (3.30) associated with a *volumetric frame* (3.148)



geometry access unit

collection of *geometry maps* (3.65) and auxiliary geometry frames, if present, corresponding to the same time instance



geometry frame

2D array created through the aggregation of the *geometry* (3.62) information associated with each *patch* (3.93)



geometry map

*geometry frame* (3.64) containing geometry *patch* (3.93) information projected at a particular depth

global broken link access access unit  
GBLA access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *GBLA coded atlas* (3.67)



global broken link access coded atlas

GBLA coded atlas

*IRAP coded atlas* (3.79) frame for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_GBLA\_W\_LP, NAL\_GBLA\_W\_RADL, or NAL\_GBLA\_N\_LP respectively

global clean random access access unit  
GCRA access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *GCRA coded atlas* (3.69)



global clean random access coded atlas

GCRA coded atlas

*IRAP coded atlas* (3.79) frame for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_GCRA

global instantaneous decoding refresh access unit  
GIDR access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *GIDR coded atlas* (3.71)



global instantaneous decoding refresh coded atlas access unit

GIDR-coded atlas

*coded atlas* (3.34) for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_GIDR\_W\_RADL, NAL\_GBLA\_N\_LP, or NAL\_GCRA, or in the range of NAL\_GBLA\_W\_LP to NAL\_GBLA\_N\_LP, inclusive, and specify a random access association between the current *coded atlas* and its corresponding coded video frames at the same composition time

global intra random access point coded atlas

GIRAP coded atlas

*IRAP coded atlas* (3.79) for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_GIDR\_W\_RADL, NAL\_GBLA\_N\_LP, or NAL\_GCRA, or in the range of NAL\_GBLA\_W\_LP to NAL\_GBLA\_N\_LP, inclusive, and specify a random access association between the current coded atlas and the corresponding coded video frames at the same composition time



hypothetical reference decoder

HRD

hypothetical decoder model that specifies constraints on the variability of conforming atlas *NAL unit* (3.84) streams or conforming *coded atlas* (3.34) sample streams that an encoding process may produce

hypothetical stream scheduler

HSS

hypothetical delivery mechanism used for checking the conformance of an *atlas sub-bitstream* (3.13) or a decoder with regards to the timing and data flow of the input of an *atlas sub-bitstream* (3.13) into the *hypothetical reference decoder* (3.73)

instantaneous decoding refresh coded atlas access unit

IDR coded atlas access unit

access unit in which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is an *IDR coded atlas* (3.76)



instantaneous decoding refresh coded atlas

IDR coded atlas

*IRAP coded atlas* (3.79) for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_IDR\_W\_RADL, or NAL\_IDR\_N\_LP, NAL\_GIDR\_W\_RADL, or NAL\_GIDR\_N\_LP

NOTE – An IDR coded atlas does not refer to any atlases other than itself for inter prediction in its decoding process, and may be the first atlas in the bitstream in decoding order, or may appear later in the bitstream. Each IDR coded atlas is the first atlas of a CAS in decoding order. When an IDR coded atlas for which each ACL NAL unit has nal\_unit\_type equal to NAL\_IDR\_W\_RADL or NAL\_GIDR\_W\_RADL, it may have associated RADL coded atlases. When an IDR coded atlas for which each ACL NAL unit has nal\_unit\_type equal to NAL\_IDR\_N\_LP or NAL\_GIDR\_N\_LP, it does not have any associated leading coded atlases. An IDR coded atlas does not have associated RASL coded atlases.

inter atlas tile

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

intra atlas tile

atlas tile that is decoded using only intra prediction methods



intra random access point coded atlas

IRAP coded atlas

IRAP coded atlas frame

*coded atlas* (3.34) for which each *ACL NAL unit* (3.6) has nal\_unit\_type in the range of NAL\_BLA\_W\_LP to NAL\_RSV\_IRAP\_ACL\_29, inclusive

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



intra random access point coded atlas access unit

IRAP coded atlas access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is an *IRAP coded atlas* (3.79)



intra random access point coded common atlas frame

IRAP coded common atlas frame

*coded common atlas frame* (3.38) that replaces all common information (e.g. projection parameters)



intra random access point coded common atlas access unit

IRAP coded common atlas access unit

access unitin which the *coded common atlas frame* (3.38) with nal\_layer\_id equal to 0 is an *IRAP coded common atlas frame* (3.81)

multi-component collection of V3C sub-bitstreams

*V3C sub-bitstreams* (3.143) of multiple *V3C components* (3.139) that, when decoded, enable the reconstruction of volumetric content



network abstraction layer unit

NAL unit

syntax structure containing an indication of the type of data to follow and *bytes* (3.24) containing that data in the form of an *RBSP* (3.108)

network abstraction layer unit stream

NAL unit stream

sequence of *NAL units* (3.84)



non-ACL NAL unit

NAL unit (3.84) that is not an *ACL NAL unit* (3.6)

non-auxiliary video component  
regular video component

*video component* (3.135) indicated as being of non-auxiliary type through an appropriate flag in the VPS (3.144)

non-auxiliary video sub-bitstream  
regular video sub-bitstream

video sub-bitstream (3.137) indicated as being of non-auxiliary type through an appropriate flag in the VPS (3.144)



non-essential supplemental enhancement information

NSEI

SEI (3.126) that is deemed as non essential by the decoding process and may be ignored or discarded without any adverse effects



non-essential supplemental enhancement information NAL unit

NSEI NAL unit

*NAL unit* (3.84) corresponding to a *NSEI* (3.89) and has nal\_unit\_type equal to NAL\_PREFIX\_NSEI or NAL\_SUFFIX\_NSEI



occupancy

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



occupancy frame

collection of *occupancy* (3.91) values that constitute a 2D array and represents the entire occupancy (3.91) information of a single *atlas frame* (3.8)



patch

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

patch data

data in an *atlas* (3.4) associated with a *patch* (3.93) that enables the conversion of 2D projected data back to 3D space



patch mode

*syntax element* (3.128) in the *atlas bitstream* (3.5) that indicates how a *patch* (3.93) is defined and associated with other *V3C components* (3.139) and provides information of how to reconstruct such *V3C components* (3.139)



patch type

classification of *patch modes* (3.95) based on how the characteristics of a *patch* (3.93) are signalled and interpreted



prefix ESEI NAL unit

*essential SEI NAL unit* (3.60) that has nal\_unit\_type equal to NAL\_PREFIX\_ESEI



prefix NSEI NAL unit

non-essential SEI NAL unit (3.90) that has nal\_unit\_type equal to NAL\_PREFIX\_NSEI

prefix SEI message

*SEI* (3.126) message that is contained in a prefix NSEI NAL unit (3.98) or prefix ESEI NAL unit (3.97)

prefix SEI NAL unit

*SEI NAL unit* (3.127) that has nal\_unit\_type equal to NAL\_PREFIX\_NSEI or NAL\_PREFIX\_ESEI



PROJECTED patch

*patch* (3.93) with *patch mode* (3.95) equal to I\_INTRA, P\_INTRA, P\_INTER, P\_MERGE, or P\_SKIP that is associated with projected information onto a 2D image

PROJECTED patch type

*patch type* (3.96) indicatinga *PROJECTED patch* (3.101)

random access

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

random access decodable leading access unit  
RADL access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *RADL coded atlas* (3.105)



random access decodable leading coded atlas  
RADL coded atlas

*coded atlas* (3.34) for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_RADL\_R or NAL\_RADL\_N

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

random access skipped leading access unit  
RASL access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *RASL coded atlas* (3.107)



random access skipped leading coded atlas  
RASL coded atlas

*coded atlas* (3.34) for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_RASL\_R or NAL\_RASL\_N

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



raw byte sequence payload

RBSP

*syntax structure* (3.129) containing an integer number of *bytes* (3.24) that is encapsulated in a *NAL unit* (3.84) and that is either empty or has the form of a *string of data bits* (3.116) containing *syntax elements* (3.128) followed by an *RBSP stop bit* (3.109) and zero or more subsequent bits equal to 0



raw byte sequence payload stop bit

RBSP stop bit

bit equal to 1 present within a *raw byte sequence payload* (3.91)after a *string of data bits* (3.116), for which the location of the end within an *RBSP* (3.91) can be identified by searching from the end of the *RBSP* (3.91) for the for the last non-zero bit in the *RBSP* (3.91)



RAW coded point

*coded representation* (3.42) of a 3D point for which its *geometry* (3.62) value is directly stored in the corresponding region of a *RAW patch* (3.111) in either the first component of the first *geometry map* (3.65) or the *auxiliary geometry frame* (3.20), and for which its attribute values, if present, are directly stored in the corresponding region of a *RAW patch* (3.111) in either first *attribute map* (3.18) or the *auxiliary attribute frame* (3.19), without any form of projection



RAW patch

*patch* (3.93) with *patch mode* (3.95) equal to I\_RAW, P\_RAW, or P\_SKIP associated with *RAW coded points* (3.110)

RAW patch type

*patch type* (3.96) indicatinga *RAW patch* (3.111)

reserved

term that may be used to specify that some values of a particular *syntax element* (3.129) are for future use by ISO/IEC and shall not be used in *bitstreams* (3.23) or atlas *bitstreams* (3.5) conforming to this version of this document, but may be used in *bitstreams* (3.23) or atlas *bitstreams* (3.5) conforming to future extensions of this document

step-wise temporal sub-layer access access unit  
STSA access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *STSA coded atlas* (3.115)



step-wise temporal sub-layer access coded atlas  
STSA coded atlas

*coded atlas* (3.34) for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_STSA\_R or NAL\_STSA\_N



string of data bits  
SODB

sequence of some number of bits representing *syntax elements* (3.128) present within a *raw byte sequence payload* (3.108) prior to the *raw byte sequence payload stop* *bit* (3.109), where the left-most bit is considered to be the first and most significant bit, and the right-most bit is considered to be the last and least significant bit



sub-bitstream

portion of a *bitstream* (3.23)



sub-bitstream composition unit

partition of a *sub-bitstream* (3.117) that has a certain composition time



sub-bitstream IRAP composition unit

*sub-bitstream composition unit* (3.118)that forms an independent random access point for the *sub-bitstream* (3.117)

sub-layer

temporal scalable layer of a temporal scalable *atlas sub-bitstream* (3.13), consisting of *ACL NAL units* (3.6) with a particular value of the TemporalId variable and the associated *non-ACL NAL units* (3.86).

sub-layer non-reference atlas frame  
SLNR atlas frame

*atlas frames* (3.8) that contains *patches* (3.93) that cannot be used for inter prediction in the decoding process of subsequent *atlas frames* (3.8) of the same sub-layer in decoding order.



suffix ESEI NAL unit

*ESEI NAL* (3.60) unit that has nal\_unit\_type equal to NAL\_SUFFIX\_ESEI



suffix NSEI NAL unit

*NSEI NAL unit* (3.90) that has nal\_unit\_type equal to NAL\_SUFFIX\_NSEI



suffix SEI message

*SEI* (3.126) message that is contained in either a suffix NSEI NAL unit (3.123) or a suffix ESEI NAL unit (3.122)

suffix SEI NAL unit

*SEI NAL unit* (3.127) that has nal\_unit\_type equal to NAL\_SUFFIX\_NSEI or NAL\_SUFFIX\_ESEI



supplemental enhancement information

SEI

information that may be inserted into the bitstream (3.23) for various purposes such as indicating bitstream (3.23)properties or for specifying various constraints that may apply to the bitstream (3.23)



supplemental enhancement information NAL unit

SEI NAL unit

*NAL unit* (3.84), intended for providing *supplemental enhancement information* (3.126) that has nal\_unit\_type equal to NAL\_PREFIX\_NSEI, NAL\_PREFIX\_ESEI, NAL\_SUFFIX\_NSEI, or NAL\_SUFFIX\_ESEI



syntax element

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



syntax structure

zero or more *syntax elements* (3.128) present together in the *bitstream* (3.23) in a specified order

temporal sub-layer access access unit  
TSA access unit

access unitin which the *coded atlas* (3.34) with nal\_layer\_id equal to 0 is a *TSA coded atlas* (3.131)



temporal sub-layer access coded atlas  
TSA coded atlas

*coded atlas* (3.34) for which each *ACL NAL unit* (3.6) has nal\_unit\_type equal to NAL\_TSA\_R or NAL\_TSA\_N

tile

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

unspecified

term that may be used to specify some values of a particular *syntax element* (3.129) to indicate that the values have no specified meaning in this document and will not have a specified meaning in the future as an integral part of future versions of this document

video bitstream

*bitstream* (3.23) conforming to a video specification that may represent a *V3C component* (3.139) as specified by this document



video component

*V3C component* (3.139) of type *occupancy* (3.91), *geometry* (3.62), or *attribute* (3.15)

video data unit

*syntax structure* (3.129) containing video data information, such as a NAL unit (3.84) in the context of ISO/IEC 23008-2 or ISO/IEC 14496-10



video sub-bitstream

extracted *sub-bitstream* (3.117) from the *V3C bitstream* (3.138) containing a part of a *video bitstream* (3.146)



video-based visual volumetric coding bitstream

V3C bitstream

sequence of bits that forms the representation of *coded volumetric frames* (3.37) and associated data forming one or more *CVSs* (3.41)



video-based visual volumetric coding component

V3C component

*atlas* (3.4), *occupancy* (3.91), *geometry* (3.62), or *attribute* (3.15) of a particular type that is associated with a V3C volumetric content representation

video-based visual volumetric coding component frame

V3C component frame

single *V3C component* (3.139) pertaining to one particular output time



video-based visual volumetric coding composition unit

V3C composition unit

set of all *sub-bitstream composition units* (3.118)that share the same *composition time* (3.50), where one of the *sub-bitstream composition units* (3.118) is a *coded atlas access unit* (3.35)



video-based visual volumetric coding IRAP composition unit

V3C IRAP composition unit

*V3C composition unit* (3.141) for which all *sub-bitstream composition units* (3.118) are sub-bitsteam IRAP composition units (3.119), where one of the sub-bitstream composition units (3.118) is a coded atlas access unit (3.35) for which each ACL NAL unit (3.6) has nal\_unit\_type in the range of NAL\_GBLA\_W\_LP to NAL\_GBLA\_N\_LP, or is in the range of NAL\_GIDR\_W\_RADL to NAL\_GIDR\_N\_LP, or is equal to NAL\_GCRA, inclusive



video-based visual volumetric coding sub-bitstream

V3C sub-bitstream

*sub-bitstream* (3.117) of a *V3C bitstream* (3.138) corresponding to a *V3C component* (3.139)



video-based visual volumetric coding parameter set

V3C parameter set

VPS

*syntax structure* (3.129) containing *syntax elements* (3.128) that apply to zero or more entire *CVSs* (3.41) and may be referred to by *syntax elements* (3.128) found in the *V3C unit header* (3.146)



video-based visual volumetric coding unit

V3C unit

*syntax structure* (3.129) containing a *V3C unit header* (3.146) and a *V3C unit payload* (3.147)



video-based visual volumetric coding unit header

V3C unit header

*syntax structure* (3.129) containing an indication of the type of data to follow



video-based visual volumetric coding unit payload

V3C unit payload

*syntax structure* (3.129) containing the *bytes* (3.24) containing *V3C sub-bitstream* (3.143) data



volumetric frame

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

volumetric sequence

sequence of *volumetric frames* (3.148)

# Abbreviated terms

1D One-Dimensional

2D Two-Dimensional

3D Three-Dimensional

4D Four-Dimensional

5D Five-Dimensional

6D Six-Dimensional

7D Seven-Dimensional

ACL Atlas Coding Layer

AAPS Atlas Adaptation Parameter Set

AFPS Atlas Frame Parameter Set

ASPS Atlas Sequence Parameter Set

AU Access Unit

BLA Broken Link Access

CAB Coded Atlas Buffer

CRA Clean Random Access

DAB Decoded Atlas Buffer

HSS Hypothetical Stream Scheduler

I Intra

IDR Instantaneous Decoding Refresh

IRAP Intra Random Access Point

LSB Least Significant Bit

MSB Most Significant Bit

NAL Network Abstraction Layer

P Predictive

PCC Point Cloud Compression

PLR Point Local Reconstruction

RAFL Reference Atlas Frame List

RBSP Raw Byte Sequence Payload

SODB String of Data Bits

V3C Visual Volumetric Video-based Coding

V-PCC Video-based Point Cloud Compression

# Conventions

## General

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

## Arithmetic operators

|  |  |
| --- | --- |
| + | addition |
| − | subtraction (as a two-argument operator) or negation (as a unary prefix operator) |
| \* | multiplication, including matrix multiplication |
| xy | exponentiation. Specifies x to the power of y. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation. |
| / | integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1. |
| ÷ | division in mathematical equations where no truncation or rounding is intended. |
|  | division in mathematical equations where no truncation or rounding is intended. |
|  | summation of f( i ) with i taking all integer values from x up to and including y. |
| x % y | modulus. Remainder of x divided by y, defined only for integers x and y with x >= 0 and y > 0. |

## Logical operators

|  |  |
| --- | --- |
| x && y | Boolean logical "and" of x and y. |
| x || y | Boolean logical "or" of x and y. |
| ! | Boolean logical "not". |
| x ? y : z | if x is TRUE or not equal to 0, evaluates to the value of y; otherwise, evaluates to the value of z. |

## Relational operators

|  |  |
| --- | --- |
| > | Greater than. |
| >= | Greater than or equal to. |
| < | Less than. |
| <= | Less than or equal to. |
| == | Equal to. |
| != | Not equal to. |

## Bit-wise operators

|  |  |
| --- | --- |
| ~ | bit-wise "not".  When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0. |
| & | bit-wise "and".  When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0. |
| | | bit-wise "or".  When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0. |
| ^ | bit-wise "exclusive or".  When operating on integer arguments, operates on a two's complement representation of the integer value. When operating on a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding more significant bits equal to 0. |
| x >> y | arithmetic right shift of a two's complement integer representation of x by y binary digits.  This function is defined only for non-negative integer values of y. Bits shifted into the MSBs as a result of the right shift have a value equal to the MSB of x prior to the shift operation. |
| x << y | arithmetic left shift of a two's complement integer representation of x by y binary digits.  This function is defined only for non-negative integer values of y. Bits shifted into the LSBs as a result of the left shift have a value equal to 0. |

## Assignment operators

|  |  |
| --- | --- |
| = | assignment operator. |
| ++ | increment, i.e. x++ is equivalent to x = x + 1; when used in an array index, evaluates to the value of the variable prior to the increment operation. |
| −− | decrement, i.e. x−− is equivalent to x = x − 1; when used in an array index, evaluates to the value of the variable prior to the decrement operation. |
| += | increment by amount specified, i.e. x += 3 is equivalent to x = x + 3, and x += (−3) is equivalent to x = x + (−3). |
| −= | decrement by amount specified, i.e. x −= 3 is equivalent to x = x − 3, and x −= (−3) is equivalent to x = x − (−3). |

## Other operators

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

## Mathematical functions

Abs( x ) (1)

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

Clip3( x, y, z ) =  (3)

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

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

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

Max( x, y ) (7)

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

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

Min( x, y ) (10)

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

Sign( x ) =  (12)

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

Sqrt( x ) the square root of x. (14)

## Order of operation precedence

When order of precedence in an expression is not indicated explicitly by use of parentheses, the following rules apply:

– Operations of a higher precedence are evaluated before any operation of a lower precedence.

– Operations of the same precedence are evaluated sequentially from left to right.

Table 1 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

NOTE – For those operators that are also used in the C programming language, the order of precedence used in this document is the same as used in the C programming language.

Table 1 – Operation precedence from highest (at top of table) to lowest (at bottom of table)

|  |  |
| --- | --- |
| **operations (with operands x, y, and z)** | |
| "x++", "x−−" |  |
| "!x", "−x" (as a unary prefix operator) |  |
| xy |  |
| "x \* y", "x / y", "x ÷ y", "", "x % y" |  |
| "x + y", "x − y" (as a two-argument operator), |  |
| "x << y", "x >> y" |  |
| "x < y", "x <= y", "x > y", "x >= y" |  |
| "x == y", "x != y" |  |
| "x & y" |  |
| "x | y" |  |
| "x && y" |  |
| "x || y" |  |
| "x ? y : z" |  |
| "x..y" |  |
| "x = y", "x += y", "x −= y" |  |

## Variables, syntax elements, and tables

Syntax elements in the bitstream are represented in **bold** type. Each syntax element is described by its name (all lower case letters with underscore characters), and one descriptor for its method of coded representation. The decoding process behaves according to the value of the syntax element and to the values of previously decoded syntax elements. When a value of a syntax element is used in the syntax tables or the text, it appears in regular (i.e. not bold) type.

In some cases the syntax tables may use the values of other variables derived from syntax elements values. Such variables appear in the syntax tables, or text, named by a mixture of lower case and upper case letter and without any underscore characters. Variables starting with an upper case letter are derived for the decoding of the current syntax structure and all depending syntax structures. Variables starting with an upper case letter may be used in the decoding process for later syntax structures without mentioning the originating syntax structure of the variable. Variables starting with a lower case letter are only used within the subclause in which they are derived.

In some cases, "mnemonic" names for syntax element values or variable values are used interchangeably with their numerical values. Sometimes "mnemonic" names are used without any associated numerical values. The association of values and names is specified in the text. The names are constructed from one or more groups of letters separated by an underscore character. Each group starts with an upper case letter and may contain more upper case letters.

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

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

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

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

NOTE 2 – In some video specifications a reverse indexing order can be used when using square parenthesis for depicting two dimensional arrays, i.e. the first element in the square parentheses is the column (horizontal) index and the second element in the square parentheses is the row (vertical) index.

A specification of values of the entries in rows and columns of an array may be denoted by { {...} {...} }, where each inner pair of brackets specifies the values of the elements within a row in increasing column order and the rows are ordered in increasing row order. Thus, setting a matrix s equal to { { 1 } { 6 } { 4 } { 9 } } specifies that s[ 0 ][ 0 ] is set equal to 1, s[ 0 ][ 1 ] is set equal to 6, s[ 1 ][ 0 ] is set equal to 4, and s[ 1 ][ 1 ] is set equal to 9.

The symbol "×" is used to separate the individual dimensions of an array in this document. For example, a 3D array, array3D, having elements array3D[ z ][ y ][ x ], with z being in the range of 0 to ( Dim0, – 1 ), inclusive, y in the range of 0to ( Dim1 – 1 ), inclusive, and x in the range of 0 to ( Dim2 – 1 ), inclusive, can also be referred to as being of size Dim0 × Dim1 × Dim2.

Binary notation is indicated by enclosing the string of bit values by single quote marks. For example, '01000001' represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Hexadecimal notation, indicated by prefixing the hexadecimal number by "0x", may be used instead of binary notation when the number of bits is an integer multiple of 4. For example, 0x41 represents an eight-bit string having only its second and its last bits (counted from the most to the least significant bit) equal to 1.

Numerical values not enclosed in single quotes and not prefixed by "0x" are decimal values.

A value equal to 0 represents a FALSE condition in a test statement. The value TRUE is represented by any value different from zero.

## Text description of logical operations

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0 )  
 statement 0  
else if( condition 1 )  
 statement 1  
...  
else /\* informative remark on remaining condition \*/  
 statement n

may be described in the following manner:

... as follows / ... the following applies:

– If condition 0, statement 0

– Otherwise, if condition 1, statement 1

– ...

– Otherwise (informative remark on remaining condition), statement n

Each "If ... Otherwise, if ... Otherwise, ..." statement in the text is introduced with "... as follows" or "... the following applies" immediately followed by "If ... ". The last condition of the "If ... Otherwise, if ... Otherwise, ..." is always an "Otherwise, ...". Interleaved "If ... Otherwise, if ... Otherwise, ..." statements can be identified by matching "... as follows" or "... the following applies" with the ending "Otherwise, ...".

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0a && condition 0b )  
 statement 0  
else if( condition 1a || condition 1b )  
 statement 1  
...  
else  
 statement n

may be described in the following manner:

... as follows / ... the following applies:

– If all of the following conditions are true, statement 0:

– condition 0a

– condition 0b

– Otherwise, if one or more of the following conditions are true, statement 1:

– condition 1a

– condition 1b

– ...

– Otherwise, statement n

In the text, a statement of logical operations as would be described mathematically in the following form:

if( condition 0 )  
 statement 0  
if( condition 1 )  
 statement 1

may be described in the following manner:

When condition 0, statement 0

When condition 1, statement 1

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

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

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

is equivalent to the following:

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

## Processes

Processes are used to describe the decoding of syntax elements. A process has a separate specification and invoking. All syntax elements and upper case variables that pertain to the current syntax structure and depending syntax structures are available in the process specification and invoking. A process specification may also have a lower case variable explicitly specified as input. Each process specification has explicitly specified an output. The output is a variable that can either be an upper case variable or a lower case variable.

When invoking a process, the assignment of variables is specified as follows:

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

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

In the specification of a process, a specific coding block may be referred to by the variable name having a value equal to the address of the specific coding block.

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

## V3C characteristics

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

|  |
| --- |
|  |
| (a) |
|  |
| (b) |

Figure 1 – Volumetric media conversion at (a) encoder and reconstruction at (b) decoder side. The 3D media is converted to a series of 2D representations: occupancy, geometry, and attributes. Additional atlas information is also included in the bitstream to enable inverse reconstruction.

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

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



Figure 2 – An example of block to patch mapping

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

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

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

|  |  |
| --- | --- |
|  | |
| (a) | |
|  |  |
| (b) | (c) |

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

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

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

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

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

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

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

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

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

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

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

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

# Bitstream format, partitioning, and scanning processes

## General

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

## V3C bitstream formats

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

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

NOTE – The V3C unit stream format is commonly not intended to be used in any applications on its own since it requires additional information, i.e. sub-bitstream size information, for decoding its associated sub-bitstreams. One method of achieving this is through the use of the sample stream format.

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

## NAL bitstream formats

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

The bitstream can be in one of two formats: the NAL unit stream format or the sample stream format. The NAL unit stream format is conceptually the more "basic" type. It consists of a sequence of syntax structures called NAL units. This sequence is ordered in decoding order, as described in subclause 8.4.5.3. There are constraints imposed on the decoding order (and contents) of the NAL units in the NAL unit stream.

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

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

## Partitioning of atlas frames into tiles

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

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

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

* The atlas frame is first partitioned into NumPartitionColumns \* NumPartitionRows number of tile partitions, where NumPartitionColumns and NumPartitionRows are derived in subclause 7.5. The width and height of each tile partition, respectively, is indicated in the atlas frame tile information syntax (8.3.6.2.2).
* One or more tile partitions are then combined into tiles by indicating the locations of the tile partitions that correspond to the top-left and bottom-right corners of the tile. All tile partitions within these two tile partitions collectively form a tile, which is essentially a rectangular region of the atlas frame.

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

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

## Tile partition scanning process

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

if( afti\_uniform\_partition\_spacing\_flag ) {  
 partitionWidth = ( afti\_partition\_cols\_width\_minus1 + 1 ) \* 64  
 NumPartitionColumns = asps\_frame\_width / partitionWidth  
 PartitionPosX[ 0 ] = 0  
 PartitionWidth[ 0 ] = partitionWidth  
 for( i = 1; i < NumPartitionColumns – 1; i++ ) {  
 PartitionPosX[ i ] = PartitionPosX[ i – 1 ] + PartitionWidth[ i – 1 ]  
 PartitionWidth[ i ] = partitionWidth  
 }  
 } else {  
 NumPartitionColumns = afti\_num\_partition\_columns\_minus1 + 1  
 PartitionPosX[ 0 ] = 0  
 partitionWidth[ 0 ] = ( afti\_partition\_column\_width\_minus1[ 0 ] + 1 ) \* 64  
 for( i = 1; i < NumPartitionColumns – 1; i++ ) {  
 PartitionPosX[ i ] = PartitionPosX[ i – 1 ] + PartitionWidth[ i – 1 ]   
 PartitionWidth[ i ] = ( afti\_partition\_column\_width\_minus1[ i ] + 1 ) \* 64  
 }  
 }  
 if( NumPartitionColumns > 1 ) {  
 PartitionPosX[ NumPartitionColumns – 1 ] =   
 PartitionPosX[ NumPartitionColumns – 2 ] + PartitionWidth[ NumPartitionColumns – 2 ]  
 PartitionWidth[ NumPartitionColumns – 1 ] =  asps\_frame\_width – PartitionPosX[ NumPartitionColumns – 1 ]  
 }

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

if( afti\_uniform\_partition\_spacing\_flag ) {  
 partitionHeight = (afti\_partition\_rows\_height\_minus1 + 1) \* 64  
 NumPartitionRows = asps\_frame\_height / partitionHeight  
 PartitionPosY[ 0 ] = 0  
 PartitionHeight[ 0 ] = partitionHeight  
 for( j = 1; j < NumPartitionRows – 1; j++ ) {  
 PartitionPosY[ j ] = PartitionPosY[ j – 1 ] + PartitionHeight[ j – 1 ]   
 PartitionHeight[ j ] = partitionHeight  
 }  
 } else {  
 NumPartitionRows = afti\_num\_partition\_rows\_minus1 + 1  
 PartitionPosY[ 0 ] = 0  
 PartitionHeight[ 0 ] = ( afti\_partition\_row\_height\_minus1[ 0 ] + 1 ) \* 64  
 for( j = 1; j < NumPartitionRows – 1; j++ ) {  
 PartitionPosY[ j ] = PartitionPosY[ j – 1 ] + PartitionHeight[ j – 1 ]  
 PartitionHeight[ j ] = ( afti\_partition\_row\_height\_minus1[ j ] + 1 ) \* 64  
 }  
 }  
 if( NumPartitionRows > 1 ) {  
 PartitionPosY[ NumPartitionRows – 1 ] =  
 PartitionPosY[ NumPartitionRows – 2 ] + partitionHeight[ NumPartitionRows – 2 ]  
 PartitionHeight[ NumPartitionRows – 1 ] =   
 asps\_frame\_height – PartitionPosY[ NumPartitionRows – 1 ]  
 }

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

# Syntax and semantics

## Method of specifying syntax in tabular form

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

NOTE – An actual decoder should implement some means for identifying entry points into the bitstream and some means to identify and handle non-conforming bitstreams. The methods for identifying and handling errors and other such situations are not specified in this document.

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

|  |  |
| --- | --- |
|  | Descriptor |
| /\* A statement can be a syntax element with an associated descriptor or can be an expression used to specify conditions for the existence, type, and quantity of syntax elements, as in the following two examples \*/ |  |
| **syntax\_element** | ue(v) |
| conditioning statement |  |
|  |  |
| /\* A group of statements enclosed in curly brackets is a compound statement and is treated functionally as a single statement. \*/ |  |
| { |  |
| statement |  |
| statement |  |
| ... |  |
| } |  |
|  |  |
| /\* A "while" structure specifies a test of whether a condition is true, and if true, specifies evaluation of a statement (or compound statement) repeatedly until the condition is no longer true \*/ |  |
| while( condition ) |  |
| statement |  |
|  |  |
| /\* A "do ... while" structure specifies evaluation of a statement once, followed by a test of whether a condition is true, and if true, specifies repeated evaluation of the statement until the condition is no longer true \*/ |  |
| do |  |
| statement |  |
| while( condition ) |  |
|  |  |
| /\* An "if ... else" structure specifies a test of whether a condition is true and, if the condition is true, specifies evaluation of a primary statement, otherwise, specifies evaluation of an alternative statement. The "else" part of the structure and the associated alternative statement is omitted if no alternative statement evaluation is needed \*/ |  |
| if( condition ) |  |
| primary statement |  |
| else |  |
| alternative statement |  |
|  |  |
| /\* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. \*/ |  |
| for( initial statement; condition; subsequent statement ) |  |
| primary statement |  |

## Specification of syntax functions and descriptors

The functions presented in subclause 8.2 are used in the syntactical description. These functions are expressed in terms of the value of a bitstream pointer that indicates the position of the next bit to be read by the decoding process from the bitstream.

byte\_aligned( ) is specified as follows:

* If the current position in the bitstream is on a byte boundary, i.e. the next bit in the bitstream is the first bit in a byte, the return value of byte\_aligned( ) is equal to TRUE.
* Otherwise, the return value of byte\_aligned( ) is equal to FALSE.

more\_data\_in\_payload( ) is specified as follows:

* If byte\_aligned( ) is equal to TRUE and the current position in the sei\_payload( ) syntax structure is 8 \* payloadSize bits from the beginning of the sei\_payload( ) syntax structure, the return value of more\_data\_in\_payload( ) is equal to FALSE.
* Otherwise, the return value of more\_data\_in\_payload( ) is equal to TRUE.

more\_rbsp\_data( ) is specified as follows:

* If there is no more data in the raw byte sequence payload (RBSP), the return value of more\_rbsp\_data( ) is equal to FALSE.
* Otherwise, the RBSP data is searched for the last (least significant, right-most) bit equal to 1 that is present in the RBSP. Given the position of this bit, which is the first bit (rbsp\_stop\_one\_bit) of the rbsp\_trailing\_bits( ) syntax structure, the following applies:
* If there is more data in an RBSP before the rbsp\_trailing\_bits( ) syntax structure, the return value of more\_rbsp\_data( ) is equal to TRUE.
* Otherwise, the return value of more\_rbsp\_data( ) is equal to FALSE.

The method for enabling determination of whether there is more data in the RBSP is specified by the application (or in Annex D for applications that use the sample stream format).

next\_bits( n ) provides the next bits in the bitstream for comparison purposes, without advancing the bitstream pointer. Provides a look at the next n bits in the bitstream with n being its argument.

payload\_extension\_present( ) is specified as follows:

* If the current position in the sei\_payload( ) syntax structure is not the position of the last (least significant, right-most) bit that is equal to 1 that is less than 8 \* payloadSize bits from the beginning of the syntax structure (i.e. the position of the payload\_bit\_equal\_to\_one syntax element), the return value of payload\_extension\_present( ) is equal to TRUE.
* Otherwise, the return value of payload\_extension\_present( ) is equal to FALSE.

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

The following descriptors specify the parsing process of each syntax element:

* b(8): byte having any pattern of bit string (8 bits). The parsing process for this descriptor is specified by the return value of the function read\_bits( 8 ).
* f(n): fixed-pattern bit string using n bits written (from left to right) with the left bit first. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ).
* fl(n): binary floating point value using n bits. The parsing process for this descriptor is as specified in IEEE 754-2019.
* i(n): signed integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ) interpreted as a two's complement integer representation with the most significant (left) bit written first. In particular, the parsing process for this descriptor is specified as follows:

i(n) {  
 value = read\_bits( n )  
 if( value < ( 1 << ( n – 1 ) ) )  
 return value   
 else  
 return ( value | ~( ( 1 << (n – 1)) – 1 ) )  
 }

* se(v): signed integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 14.2.
* st(v): null-terminated string encoded as UTF-8 characters as specified in ISO/IEC 10646. The parsing process is specified as follows: st(v) begins at a byte-aligned position in the bitstream and reads and returns a series of bytes from the bitstream, beginning at the current position and continuing up to but not including the next byte-aligned byte that is equal to 0x00, and advances the bitstream pointer by ( stringLength + 1 ) \* 8 bit positions, where stringLength is equal to the number of bytes returned.

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

* u(n): unsigned integer using n bits. When n is "v" in the syntax table, the number of bits varies in a manner dependent on the value of other syntax elements. The parsing process for this descriptor is specified by the return value of the function read\_bits( n ) interpreted as a binary representation of an unsigned integer with the most significant bit written first. When n is equal to 0, the associated syntax element is not present in the bitstream and its corresponding value shall be inferred to be equal to 0 unless specified otherwise in the semantics.
* ue(v): unsigned integer 0-th order Exp-Golomb-coded syntax element with the left bit first. The parsing process for this descriptor is specified in subclause 14.2.

## Syntax in tabular form

### General

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

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

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



Figure 5 – Overview of V3C bitstream structure

### V3C unit syntax

#### General V3C unit syntax

|  |  |
| --- | --- |
| v3c\_unit( numBytesInV3CUnit ) { | **Descriptor** |
| v3c\_unit\_header( ) |  |
| v3c\_unit\_payload( numBytesInV3CUnit – 4 ) |  |
| } |  |

#### V3C unit header syntax

|  |  |
| --- | --- |
| v3c\_unit\_header( ) { | **Descriptor** |
| **vuh\_unit\_type** | u(5) |
| if( vuh\_unit\_type == V3C\_AVD || vuh\_unit\_type == V3C\_GVD ||  vuh\_unit\_type == V3C\_OVD || vuh\_unit\_type == V3C\_AD ||  vuh\_unit\_type == V3C\_CAD || vuh\_unit\_type == V3C\_PVD ) |  |
| **vuh\_v3c\_parameter\_set\_id** | u(4) |
| if( vuh\_unit\_type == V3C\_AVD || vuh\_unit\_type == V3C\_GVD ||  vuh\_unit\_type == V3C\_OVD || vuh\_unit\_type == V3C\_AD ||  vuh\_unit\_type == V3C\_PVD ) |  |
| **vuh\_atlas\_id** | u(6) |
| if( vuh\_unit\_type == V3C\_AVD ) { |  |
| **vuh\_attribute\_index** | u(7) |
| **vuh\_attribute\_partition\_index** | u(5) |
| **vuh\_map\_index** | u(4) |
| **vuh\_auxiliary\_video\_flag** | u(1) |
| } else if( vuh\_unit\_type == V3C\_GVD ) { |  |
| **vuh\_map\_index** | u(4) |
| **vuh\_auxiliary\_video\_flag** | u(1) |
| **vuh\_reserved\_zero\_12bits** | u(12) |
| } else if( vuh\_unit\_type == V3C\_OVD || vuh\_unit\_type == V3C\_AD ||  vuh\_unit\_type == V3C\_PVD ) |  |
| **vuh\_reserved\_zero\_17bits** | u(17) |
| else if( vuh\_unit\_type == V3C\_CAD ) |  |
| vuh\_reserved\_zero\_23bits | u(23) |
| else |  |
| **vuh\_reserved\_zero\_27bits** | u(27) |
| } |  |

#### V3C unit payload syntax

|  |  |
| --- | --- |
| v3c\_unit\_payload( numBytesInV3CPayload ) { | **Descriptor** |
| if( vuh\_unit\_type == V3C\_VPS ) |  |
| v3c\_parameter\_set( numBytesInV3CPayload ) |  |
| else if( vuh\_unit\_type == V3C\_AD  || vuh\_unit\_type == V3C\_CAD ) |  |
| atlas\_sub\_bitstream( numBytesInV3CPayload ) |  |
| else if( vuh\_unit\_type == V3C\_OVD ||  vuh\_unit\_type == V3C\_GVD ||  vuh\_unit\_type == V3C\_AVD ||  vuh\_unit\_type == V3C\_PVD ) |  |
| video\_sub\_bitstream( numBytesInV3CPayload ) |  |
| } |  |

#### Atlas sub-bitstream syntax

|  |  |
| --- | --- |
| atlas\_sub\_bitstream( numBytes ) { |  |
| sample\_stream\_nal\_header( ) |  |
| numBytes−− |  |
| while( numBytes > 0 ) { |  |
| sample\_stream\_nal\_unit( ) |  |
| numBytes −= ssnu\_nal\_unit\_size + ssnh\_unit\_size\_precision\_bytes\_minus1 + 1 |  |
| } |  |
| } |  |

### Byte alignment syntax

|  |  |
| --- | --- |
| byte\_alignment( ) { | **Descriptor** |
| **alignment\_bit\_equal\_to\_one** /\* equal to 1 \*/ | f(1) |
| while( !byte\_aligned( ) ) |  |
| **alignment\_bit\_equal\_to\_zero** /\* equal to 0 \*/ | f(1) |
| } |  |

### V3C parameter set syntax

#### General V3C parameter set syntax

|  |  |
| --- | --- |
| v3c\_parameter\_set( numBytesInV3CPayload ) { | **Descriptor** |
| profile\_tier\_level( ) |  |
| **vps\_v3c\_parameter\_set\_id** | u(4) |
| **vps\_reserved\_zero\_8bits** | u(8) |
| **vps\_atlas\_count\_minus1** | u(6) |
| for(k = 0; k < vps\_atlas\_count\_minus1 + 1; k++ ) { |  |
| **vps\_atlas\_id**[ k ] | u(6) |
| j = vps\_atlas\_id[ k ] |  |
| **vps\_frame\_width**[ j ] | ue(v) |
| **vps\_frame\_height**[ j ] | ue(v) |
| **vps\_map\_count\_minus1**[ j ] | u(4) |
| if( vps\_map\_count\_minus1[ j ] > 0 ) |  |
| **vps\_multiple\_map\_streams\_present\_flag**[ j ] | u(1) |
| vps\_map\_absolute\_coding\_enabled\_flag[ j ][ 0 ] = 1 |  |
| vps\_map\_predictor\_index\_diff[ j ][ 0 ] = 0 |  |
| for( i = 1; i <= vps\_map\_count\_minus1[ j ]; i++ ) { |  |
| if( vps\_multiple\_map\_streams\_present\_flag[ j ] ) |  |
| **vps\_map\_absolute\_coding\_enabled\_flag**[ j ][ i ] | u(1) |
| else |  |
| vps\_map\_absolute\_coding\_enabled\_flag[ j ][ i ] = 1 |  |
| if( vps\_map\_absolute\_coding\_enabled\_flag[ j ][ i ] == 0 ) { |  |
| **vps\_map\_predictor\_index\_diff**[ j ][ i ] | ue(v) |
| } |  |
| } |  |
| **vps\_auxiliary\_video\_present\_flag**[ j ] | u(1) |
| **vps\_occupancy\_video\_present\_flag**[ j ] | u(1) |
| **vps\_geometry\_video\_present\_flag**[ j ] | u(1) |
| **vps\_attribute\_video\_present\_flag**[ j ] | u(1) |
| if( vps\_occupancy\_video\_present\_flag[ j ] ) |  |
| occupancy\_information( j ) |  |
| if( vps\_geometry\_video\_present\_flag[ j ] ) |  |
| geometry\_information( j ) |  |
| if( vps\_attribute\_video\_present\_flag[ j ] ) |  |
| attribute\_information( j ) |  |
| } |  |
| **vps\_extension\_present\_flag** | u(1) |
| if( vps\_extension\_present\_flag ) { |  |
| **vps\_packing\_information\_present\_flag** | u(1) |
| **vps\_miv\_extension\_present\_flag** | u(1) |
| **vps\_extension\_6bits** | u(6) |
| } |  |
| if( vps\_packing\_information\_present\_flag ) { |  |
| for( k = 0 ; k <= vps\_atlas\_count\_minus1; k++ ) { |  |
| j = vps\_atlas\_id[ k ] |  |
| **vps\_packed\_video\_present\_flag**[ j ] |  |
| if( vps\_packed\_video\_present\_flag[ j ] ) |  |
| packing\_information( j ) |  |
| } |  |
| } |  |
| if( vps\_miv\_extension\_present\_flag ) |  |
| vps\_miv\_extension( ) /\*Specified in ISO/IEC 23090-12[[2]](#footnote-3) \*/ |  |
| if( vps\_extension\_6bits ) { |  |
| **vps\_extension\_length\_minus1** | ue(v) |
| for( j = 0; j < vps\_extension\_length\_minus1 + 1; j++ ) { |  |
| **vps\_extension\_data\_byte** | u(8) |
| } |  |
| } |  |
| byte\_alignment( ) |  |
| } |  |

#### Profile, tier, and level syntax

|  |  |
| --- | --- |
| profile\_tier\_level( ) { | **Descriptor** |
| **ptl\_tier\_flag** | u(1) |
| **ptl\_profile\_codec\_group\_idc** | u(7) |
| **ptl\_profile\_toolset\_idc** | u(8) |
| **ptl\_profile\_reconstruction\_idc** | u(8) |
| **ptl\_reserved\_zero\_16bits** | u(16) |
| **ptl\_max\_decodes\_idc** | u(4) |
| **ptl\_reserved\_0xfff\_12bits** | u(12) |
| **ptl\_level\_idc** | u(8) |
| **ptl\_num\_sub\_profiles** | u(6) |
| **ptl\_extended\_sub\_profile\_flag** | u(1) |
| for( i = 0; i < ptl\_num\_sub\_profiles; i++ ) |  |
| **ptl\_sub\_profile\_idc**[ i ] | u(v) |
| **ptl\_tool\_constraints\_present\_flag** | u(1) |
| if( ptl\_toolset\_constraints\_present\_flag ) |  |
| profile\_toolset\_constraints\_information( ) |  |
| } |  |

#### Occupancy information syntax

|  |  |
| --- | --- |
| occupancy\_information( atlasID ) { | **Descriptor** |
| **oi\_occupancy\_codec\_id**[ atlasID ] | u(8) |
| **oi\_lossy\_occupancy\_compression\_threshold**[ atlasID ] | u(8) |
| **oi\_occupancy\_2d\_bit\_depth\_minus1**[ atlasID ] | u(5) |
| **oi\_occupancy\_MSB\_align\_flag**[ atlasID ] | u(1) |
| } |  |

#### Geometry information syntax

|  |  |
| --- | --- |
| geometry\_information(atlasID ) { | **Descriptor** |
| **gi\_geometry\_codec\_id**[ atlasID ] | u(8) |
| **gi\_geometry\_2d\_bit\_depth\_minus1**[ atlasID ] | u(5) |
| **gi\_geometry\_MSB\_align\_flag**[ atlasID ] | u(1) |
| **gi\_geometry\_3d\_coordinates\_bit\_depth\_minus1**[ atlasID ] | u(5) |
| if( vps\_auxiliary\_video\_present\_flag[ atlasID ] ) |  |
| **gi\_auxiliary\_geometry\_codec\_id**[ atlasID ] | u(8) |
| } |  |

#### Attribute information syntax

|  |  |
| --- | --- |
| attribute\_information(atlasID) { | **Descriptor** |
| **ai\_attribute\_count**[ atlasID ] | u(7) |
| for( i = 0; i < ai\_attribute\_count[ atlasID ]; i++ ) { |  |
| **ai\_attribute\_type\_id**[ atlasID ][ i ] | u(4) |
| **ai\_attribute\_codec\_id**[ atlasID ][ i ] | u(8) |
| if( vps\_auxiliary\_video\_present\_flag[ atlasID ] ) |  |
| **ai\_auxiliary\_attribute\_codec\_id**[ atlasID ][ i ] | u(8) |
| if( vps\_map\_count\_minus1[ atlasID ] > 0 ) |  |
| **ai\_attribute\_map\_absolute\_coding\_persistence\_flag**[ atlasID ][ i ] | u(1) |
| **ai\_attribute\_dimension\_minus1**[ atlasID ][ i ] | u(6) |
| if( ai\_attribute\_dimension\_minus1[ atlasID ][ i ] > 0 ) { |  |
| **ai\_attribute\_dimension\_partitions\_minus1**[ atlasID ][ i ] | u(6) |
| remainingDimensions = ai\_attribute\_dimension\_minus1[ atlasID ][ i ] |  |
| k = ai\_attribute\_dimension\_partitions\_minus1[ atlasID ][ i ] |  |
| for( j = 0; j < k; j++ ) { |  |
| if( k – j == remainingDimensions ) |  |
| ai\_attribute\_partition\_channels\_minus1[ atlasID ][ i ][ j ] = 0 |  |
| else |  |
| **ai\_attribute\_partition\_channels\_minus1**[ atlasID ][ i ][ j ] | ue(v) |
| remainingDimensions –=  ai\_attribute\_partition\_channels\_minus1[ atlasID ][ i ][ j ] + 1 |  |
| } |  |
| ai\_attribute\_partition\_channels\_minus1[ atlasID ][ i ][ k ] = remainingDimensions |  |
| } |  |
| **ai\_attribute\_2d\_bit\_depth\_minus1**[ atlasID ][ i ] | u(5) |
| **ai\_attribute\_MSB\_align\_flag**[ atlasID ][ i ] | u(1) |
| } |  |
| } |  |

#### Profile toolset constraints information syntax

|  |  |
| --- | --- |
| profile\_toolset\_constraints\_information( ) { | **Descriptor** |
| **ptc\_one\_v3c\_frame\_only\_flag** | u(1) |
| **ptc\_eom\_constraint\_flag** | u(1) |
| **ptc\_max\_map\_count\_minus1** | u(4) |
| **ptc\_max\_atlas\_count\_minus1** | u(4) |
| **ptc\_multiple\_map\_streams\_constraint\_flag** | u(1) |
| **ptc\_plr\_constraint\_flag** | u(1) |
| **ptc\_attribute\_max\_dimension\_minus1** | u(6) |
| **ptc\_attribute\_max\_dimension\_partitions\_minus1** | u(6) |
| **ptc\_no\_eight\_orientations\_constraint\_flag** | u(1) |
| **ptc\_no\_45degree\_projection\_patch\_constraint\_flag** | u(1) |
| **ptc\_reserved\_zero\_6bits** | u(6) |
| **ptc\_num\_reserved\_constraint\_bytes** | u(8) |
| for( i = 0; i < ptc\_num\_reserved\_constraint\_bytes; i++ ) |  |
| **ptc\_reserved\_constraint\_byte**[ i ] | u(8) |
| } |  |

#### Packing information syntax

|  |  |
| --- | --- |
| packing\_information( j ) { | **Descriptor** |
| **pin\_codec\_id**[ j ] | u(8) |
| **pin\_regions\_count\_minus1**[ j ] | ue(v) |
| for( i = 0; i <= pin\_regions\_count\_minus1; i++ ) { |  |
| **pin\_region\_tile\_id**[ j ][ i ] | u(8) |
| **pin\_region\_type\_id\_minus2**[ j ][ i ] | u(2) |
| **pin\_region\_top\_left\_x**[ j ][ i ] | u(16) |
| **pin\_region\_top\_left\_y**[ j ][ i ] | u(16) |
| **pin\_region\_width\_minus1**[ j ][ i ] | u(16) |
| **pin\_region\_height\_minus1**[ j ][ i ] | u(16) |
| **pin\_region\_map\_index**[ j ][ i ] | u(4) |
| **pin\_region\_rotation\_flag**[ j ][ i ] | u(1) |
| if( pin\_region\_type\_id\_minus2[ j ][ i ] + 2  = =  V3C\_AVD ||  pin\_region\_type\_id\_minus2[ j ][ i ] + 2  = =  V3C\_GVD ) |  |
| **pin\_region\_auxiliary\_data\_flag**[ j ][ i ] | u(1) |
| if( pin\_region\_type\_id\_minus2[ j ][ i ] + 2  = =  V3C\_AVD ) { |  |
| **pin\_region\_attr\_type\_id**[ j ][ i ] | u(4) |
| **pin\_region\_attr\_partitions\_flag**[ j ][ i ] |  |
| if( pin\_region\_attr\_partitions\_flag[ j ][ i ] ) |  |
| **pin\_region\_attr\_partition\_index**[ j ][ i ] | u(5) |
| if( pin\_region\_attr\_partition\_index[ j ][ i ]  = =  0 ) |  |
| **pin\_region\_attr\_partitions\_minus1**[ j ][ i ] | u(6) |
| } |  |
| } |  |
| } |  |

### NAL unit syntax

#### General NAL unit syntax

|  |  |
| --- | --- |
| nal\_unit( NumBytesInNalUnit ) { | **Descriptor** |
| nal\_unit\_header( ) |  |
| NumBytesInRbsp = 0 |  |
| for( i = 2; i < NumBytesInNalUnit; i++ ) |  |
| **rbsp\_byte**[ NumBytesInRbsp++ ] | b(8) |
| } |  |

#### NAL unit header syntax

|  |  |
| --- | --- |
| nal\_unit\_header( ) { | **Descriptor** |
| **nal\_forbidden\_zero\_bit** | f(1) |
| **nal\_unit\_type** | u(6) |
| **nal\_layer\_id** | u(6) |
| **nal\_temporal\_id\_plus1** | u(3) |
| } |  |

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

#### Atlas sequence parameter set RBSP syntax

##### General atlas sequence parameter set RBSP syntax

|  |  |
| --- | --- |
| atlas\_sequence\_parameter\_set\_rbsp( ) { | **Descriptor** |
| **asps\_atlas\_sequence\_parameter\_set\_id** | ue(v) |
| **asps\_frame\_width** | ue(v) |
| **asps\_frame\_height** | ue(v) |
| **asps\_geometry\_3d\_bit\_depth\_minus1** | u(5) |
| **asps\_geometry\_2d\_bit\_depth\_minus1** | u(5) |
| **asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4** | ue(v) |
| **asps\_max\_dec\_atlas\_frame\_buffering\_minus1** | ue(v) |
| **asps\_long\_term\_ref\_atlas\_frames\_flag** | u(1) |
| **asps\_num\_ref\_atlas\_frame\_lists\_in\_asps** | ue(v) |
| for( i = 0; i < asps\_num\_ref\_atlas\_frame\_lists\_in\_asps; i++ ) |  |
| ref\_list\_struct( i ) |  |
| **asps\_use\_eight\_orientations\_flag** | u(1) |
| **asps\_extended\_projection\_enabled\_flag** | u(1) |
| if( asps\_extended\_projection\_enabled\_flag ) |  |
| **asps\_max\_number\_projections\_minus1** | ue(v) |
| **asps\_normal\_axis\_limits\_quantization\_enabled\_flag** | u(1) |
| **asps\_normal\_axis\_max\_delta\_value\_enabled\_flag** | u(1) |
| **asps\_patch\_precedence\_order\_flag** | u(1) |
| **asps\_log2\_patch\_packing\_block\_size** | u(3) |
| **asps\_patch\_size\_quantizer\_present\_flag** | u(1) |
| **asps\_map\_count\_minus1** | u(4) |
| **asps\_pixel\_deinterleaving\_enabled\_flag** | u(1) |
| if( asps\_pixel\_deinterleaving\_enabled\_flag ) |  |
| for( j = 0; j <= asps\_map\_count\_minus1; j++ ) |  |
| **asps\_map\_pixel\_deinterleaving\_flag**[ j ] | u(1) |
| **asps\_raw\_patch\_enabled\_flag** | u(1) |
| **asps\_eom\_patch\_enabled\_flag** | u(1) |
| if( asps\_eom\_patch\_enabled\_flag && asps\_map\_count\_minus1 == 0 ) |  |
| **asps\_eom\_fix\_bit\_count\_minus1** | u(4) |
| if( asps\_raw\_patch\_enabled\_flag || asps\_eom\_patch\_enabled\_flag ) |  |
| **asps\_auxiliary\_video\_enabled\_flag** | u(1) |
| **asps\_plr\_enabled\_flag** | u(1) |
| if( asps\_plr\_enabled\_flag ) |  |
| asps\_plr\_information( asps\_map\_count\_minus1 ) |  |
| **asps\_vui\_parameters\_present\_flag** | u(1) |
| if( asps\_vui\_parameters\_present\_flag ) |  |
| vui\_parameters( ) |  |
| **asps\_extension\_present\_flag** | u(1) |
| if( asps\_extension\_present\_flag ) { |  |
| **asps\_vpcc\_extension\_present\_flag** | u(1) |
| **asps\_miv\_extension\_present\_flag** | u(1) |
| **asps\_extension\_6bits** | u(6) |
| } |  |
| if( asps\_vpcc\_extension\_present\_flag ) |  |
| asps\_vpcc\_extension( ) /\* Specified in Annex H \*/ |  |
| if( asps\_miv\_extension\_present\_flag ) |  |
| asps\_miv\_extension( ) /\* Specified in ISO/IEC 23090-12 \*/ |  |
| if( asps\_extension\_6bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **asps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

##### Point local reconstruction information syntax

|  |  |
| --- | --- |
| asps\_plr\_information( mapCountMinus1 ) { | **Descriptor** |
| for( i = 0; i < mapCountMinus1 + 1; i++ ) { |  |
| **plri\_map\_present\_flag**[ i ] | u(1) |
| if( plri\_map\_present\_flag[ i ] ) { |  |
| **plri\_number\_of\_modes\_minus1**[ i ] | u(4) |
| for( j = 0; j < plri\_number\_of\_modes\_minus1[ i ] + 1; j++ ) { |  |
| **plri\_interpolate\_flag**[ i ][ j ] | u(1) |
| **plri\_filling\_flag**[ i ][ j ] | u(1) |
| **plri\_minimum\_depth**[ i ][ j ] | u(2) |
| **plri\_neighbour\_minus1**[ i ][ j ] | u(2) |
| } |  |
| **plri\_block\_threshold\_per\_patch\_minus1**[ i ] | u(6) |
| } |  |
| } |  |
| } |  |

#### Atlas frame parameter set RBSP syntax

##### General atlas frame parameter set RBSP syntax

|  |  |
| --- | --- |
| atlas\_frame\_parameter\_set\_rbsp( ) { | **Descriptor** |
| **afps\_atlas\_frame\_parameter\_set\_id** | ue(v) |
| **afps\_atlas\_sequence\_parameter\_set\_id** | ue(v) |
| atlas\_frame\_tile\_information( ) |  |
| **afps\_output\_flag\_present\_flag** | u(1) |
| **afps\_num\_ref\_idx\_default\_active\_minus1** | ue(v) |
| **afps\_additional\_lt\_afoc\_lsb\_len** | ue(v) |
| **afps\_lod\_mode\_enabled\_flag** | u(1) |
| **afps\_raw\_3d\_offset\_bit\_count\_explicit\_mode\_flag** | u(1) |
| **afps\_extension\_present\_flag** | u(1) |
| if( afps\_extension\_present\_flag ) |  |
| **afps\_extension\_8bits** | u(8) |
| if( afps\_extension\_8bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **afps\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

##### Atlas frame tile information syntax

|  |  |
| --- | --- |
| atlas\_frame\_tile\_information( ) { | **Descriptor** |
| **afti\_single\_tile\_in\_atlas\_frame\_flag** | u(1) |
| if( !afti\_single\_tile\_in\_atlas\_frame\_flag ) { |  |
| **afti\_uniform\_partition\_spacing\_flag** | u(1) |
| if( afti\_uniform\_partition\_spacing\_flag ) { |  |
| **afti\_partition\_cols\_width\_minus1** | ue(v) |
| **afti\_partition\_rows\_height\_minus1** | ue(v) |
| } else { |  |
| **afti\_num\_partition\_columns\_minus1** | ue(v) |
| **afti\_num\_partition\_rows\_minus1** | ue(v) |
| for( i = 0; i < afti\_num\_partition\_columns\_minus1; i++ ) |  |
| **afti\_partition\_column\_width\_minus1**[ i ] | ue(v) |
| for( i = 0; i < afti\_num\_partition\_rows\_minus1; i++ ) |  |
| **afti\_partition\_row\_height\_minus1**[ i ] | ue(v) |
| } |  |
| **afti\_single\_partition\_per\_tile\_flag** | u(1) |
| if( !afti\_single\_partition\_per\_tile\_flag ) { |  |
| **afti\_num\_tiles\_in\_atlas\_frame\_minus1** | ue(v) |
| for( i = 0; i < afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1; i++ ) { |  |
| **afti\_top\_left\_partition\_idx**[ i ] | u(v) |
| **afti\_bottom\_right\_partition\_column\_offset**[ i ] | ue(v) |
| **afti\_bottom\_right\_partition\_row\_offset**[ i ] | ue(v) |
| } |  |
| } |  |
| else |  |
| afti\_num\_tiles\_in\_atlas\_frame\_minus1 = NumPartitionsInAtlasFrame – 1 |  |
| } |  |
| else |  |
| afti\_num\_tiles\_in\_atlas\_frame\_minus1 = 0 |  |
| if( asps\_auxiliary\_video\_enabled\_flag ) { |  |
| **afti\_auxiliary\_video\_tile\_row\_width\_minus1** | ue(v) |
| for( i = 0; i < afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1; i++ ) |  |
| **afti\_auxiliary\_video\_tile\_row\_height**[ i ] | ue(v) |
| } |  |
| **afti\_signalled\_tile\_id\_flag** | u(1) |
| if( afti\_signalled\_tile\_id\_flag ) { |  |
| **afti\_signalled\_tile\_id\_length\_minus1** | ue(v) |
| for( i = 0; i < afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1; i++ ) { |  |
| **afti\_tile\_id**[ i ] | u(v) |
| TileIDToIndex[ afti\_tile\_id[ i ] ] = i |  |
| TileIndexToID[ i ] = afti\_tile\_id[ i ] |  |
| } |  |
| } |  |
| else |  |
| for( i = 0; i < afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1; i++ ) { |  |
| afti\_tile\_id[ i ] = i |  |
| TileIDToIndex[ i ] = i |  |
| TileIndexToID[ i ] = i |  |
| } |  |
| } |  |

#### Atlas adaptation parameter set RBSP syntax

|  |  |
| --- | --- |
| atlas\_adaptation\_parameter\_set\_rbsp( ) { | **Descriptor** |
| **aaps\_atlas\_adaptation\_parameter\_set\_id** | ue(v) |
| **aaps\_extension\_present\_flag** | u(1) |
| if( aaps\_extension\_present\_flag ) { |  |
| **aaps\_vpcc\_extension\_present\_flag** | u(1) |
| **aaps\_miv\_extension\_present\_flag** | u(1) |
| **aaps\_extension\_6bits** | u(6) |
| } |  |
| if( aaps\_vpcc\_extension\_present\_flag ) |  |
| aaps\_vpcc\_extension( ) /\* Specified in Annex H \*/ |  |
| if( aaps\_miv\_extension\_present\_flag ) { |  |
| aaps\_miv\_extension( ) /\* Specified in ISO/IEC 23090-12 \*/ |  |
| } |  |
| if( aaps\_extension\_6bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| aaps\_extension\_data\_flag | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Supplemental enhancement information RBSP syntax

|  |  |
| --- | --- |
| sei\_rbsp( ) { | **Descriptor** |
| do |  |
| sei\_message( ) |  |
| while( more\_rbsp\_data( ) ) |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Access unit delimiter RBSP syntax

|  |  |
| --- | --- |
| access\_unit\_delimiter\_rbsp( ) { | **Descriptor** |
| **aframe\_type** | u(3) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### End of sequence RBSP syntax

|  |  |
| --- | --- |
| end\_of\_sequence\_rbsp( ) { | **Descriptor** |
| } |  |

#### End of bitstream RBSP syntax

|  |  |
| --- | --- |
| end\_of\_atlas\_sub\_bitstream\_rbsp( ) { | **Descriptor** |
| } |  |

#### Filler data RBSP syntax

|  |  |
| --- | --- |
| filler\_data\_rbsp( ) { | **Descriptor** |
| while( next\_bits( 8 ) == 0xFF ) |  |
| **ff\_byte** /\* equal to 0xFF \*/ | f(8) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### Atlas tile layer RBSP syntax

|  |  |
| --- | --- |
| atlas\_tile\_layer\_rbsp( ) { | Descriptor |
| atlas\_tile\_header( ) |  |
| atlas\_tile\_data\_unit( tileID ) |  |
| rbsp\_trailing\_bits( ) |  |
| } |  |

#### RBSP trailing bit syntax

|  |  |
| --- | --- |
| rbsp\_trailing\_bits( ) { | Descriptor |
| **rbsp\_stop\_one\_bit** /\* equal to 1 \*/ | f(1) |
| while( !byte\_aligned( ) ) |  |
| **rbsp\_alignment\_zero\_bit** /\* equal to 0 \*/ | f(1) |
| } |  |

#### Atlas tile header syntax

|  |  |
| --- | --- |
| atlas\_tile\_header( ) { | **Descriptor** |
| if( nal\_unit\_type >= NAL\_BLA\_W\_LP && nal\_unit\_type <= NAL\_RSV\_IRAP\_ACL\_29 ) |  |
| **ath\_no\_output\_of\_prior\_atlas\_frames\_flag** | u(1) |
| **ath\_atlas\_frame\_parameter\_set\_id** | ue(v) |
| **ath\_atlas\_adaptation\_parameter\_set\_id** | ue(v) |
| **ath\_id** | u(v) |
| tileID = ath\_id |  |
| **ath\_type** | ue(v) |
| if( afps\_output\_flag\_present\_flag ) |  |
| **ath\_atlas\_output\_flag** | u(1) |
| **ath\_atlas\_frm\_order\_cnt\_lsb** | u(v) |
| if( asps\_num\_ref\_atlas\_frame\_lists\_in\_asps > 0 ) |  |
| **ath\_ref\_atlas\_frame\_list\_asps\_flag** | u(1) |
| if( ath\_ref\_atlas\_frame\_list\_asps\_flag == 0 ) |  |
| ref\_list\_struct( asps\_num\_ref\_atlas\_frame\_lists\_in\_asps ) |  |
| else if( asps\_num\_ref\_atlas\_frame\_lists\_in\_asps > 1 ) |  |
| **ath\_ref\_atlas\_frame\_list\_idx** | u(v) |
| for( j = 0; j < NumLtrAtlasFrmEntries; j++ ) { |  |
| **ath\_additional\_afoc\_lsb\_present\_flag**[ j ] | u(1) |
| if( ath\_additional\_afoc\_lsb\_present\_flag[ j ] ) |  |
| **ath\_additional\_afoc\_lsb\_val**[ j ] | u(v) |
| } |  |
| if( ath\_type != SKIP\_TILE ) { |  |
| if( asps\_normal\_axis\_limits\_quantization\_enabled\_flag ) { |  |
| **ath\_pos\_min\_d\_quantizer** | u(5) |
| if( asps\_normal\_axis\_max\_delta\_value\_enabled\_flag ) |  |
| **ath\_pos\_delta\_max\_d\_quantizer** | u(5) |
| } |  |
| if( asps\_patch\_size\_quantizer\_present\_flag) { |  |
| **ath\_patch\_size\_x\_info\_quantizer** | u(3) |
| **ath\_patch\_size\_y\_info\_quantizer** | u(3) |
| } |  |
| if( afps\_raw\_3d\_offset\_bit\_count\_explicit\_mode\_flag ) |  |
| **ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1** | u(v) |
| if( ath\_type == P\_TILE && num\_ref\_entries[ RlsIdx ] > 1 ) { |  |
| **ath\_num\_ref\_idx\_active\_override\_flag** | u(1) |
| if( ath\_num\_ref\_idx\_active\_override\_flag ) |  |
| **ath\_num\_ref\_idx\_active\_minus1** | ue(v) |
| } |  |
| } |  |
| byte\_alignment( ) |  |
| } |  |

#### Reference list structure syntax

|  |  |
| --- | --- |
| ref\_list\_struct( rlsIdx ) { | **Descriptor** |
| **num\_ref\_entries**[ rlsIdx ] | ue(v) |
| for( i = 0; i < num\_ref\_entries[ rlsIdx ]; i++ ) { |  |
| if( asps\_long\_term\_ref\_atlas\_frames\_flag ) |  |
| **st\_ref\_atlas\_frame\_flag**[ rlsIdx ][ i ] | u(1) |
| if( st\_ref\_atlas\_frame\_flag[ rlsIdx ][ i ] ) { |  |
| **abs\_delta\_afoc\_st**[ rlsIdx ][ i ] | ue(v) |
| if( abs\_delta\_afoc\_st[ rlsIdx ][ i ] > 0 ) |  |
| **straf\_entry\_sign\_flag**[ rlsIdx ][ i ] | u(1) |
| } else |  |
| **afoc\_lsb\_lt**[ rlsIdx ][ i ] | u(v) |
| } |  |
| } |  |

#### Common atlas frame RBSP syntax

|  |  |
| --- | --- |
| common\_atlas\_frame\_rbsp( ) { | Descriptor |
| **caf\_extension\_present\_flag** | u(1) |
| if( caf\_extension\_present\_flag ) { |  |
| **caf\_miv\_extension\_present\_flag** | u(1) |
| caf\_extension\_7bits | u(7) |
| } |  |
| if( caf\_miv\_extension\_present\_flag ) |  |
| caf\_miv\_extension( ) /\* Specified in ISO/IEC 23090-12 \*/ |  |
| if( caf\_extension\_7bits ) |  |
| while( more\_rbsp\_data( ) ) |  |
| **caf\_extension\_data\_flag** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

### Atlas tile data unit syntax

#### General atlas tile data unit syntax

|  |  |
| --- | --- |
| atlas\_tile\_data\_unit( tileID ) { | **Descriptor** |
| if( ath\_type == SKIP\_TILE ) { |  |
| for( p = 0; p < RefAtduTotalNumPatches[ tileID ]; p++ ) |  |
| skip\_patch\_data\_unit( ) |  |
| } else { |  |
| p = 0 |  |
| do { |  |
| **atdu\_patch\_mode**[ tileID ][ p ] | ue(v) |
| isEnd = ( ath\_type == P\_TILE && atdu\_patch\_mode[ tileID ][ p ] == P\_END) ||  ( ath\_type == I\_TILE && atdu\_patch\_mode[ tileID ][ p ] == I\_END ) |  |
| if( !isEnd ) { |  |
| patch\_information\_data( tileID , p , atdu\_patch\_mode[ tileID ][ p ] ) |  |
| p++ |  |
| } |  |
| } while( !isEnd ) |  |
| } |  |
| AtduTotalNumPatches[ tileID ] = p |  |
| } |  |

#### Patch information data syntax

|  |  |
| --- | --- |
| patch\_information\_data( tileID, patchIdx, patchMode ) { | **Descriptor** |
| if( ath\_type == P\_TILE ) { |  |
| if( patchMode == P\_SKIP ) |  |
| skip\_patch\_data\_unit( ) |  |
| else if( patchMode == P\_MERGE ) |  |
| merge\_patch\_data\_unit( tileID, patchIdx ) |  |
| else if( patchMode == P\_INTRA ) |  |
| patch\_data\_unit( tileID, patchIdx ) |  |
| else if( patchMode == P\_INTER ) |  |
| inter\_patch\_data\_unit( tileID, patchIdx ) |  |
| else if( patchMode == P\_RAW ) |  |
| raw\_patch\_data\_unit( tileID, patchIdx ) |  |
| else if( patchMode == P\_EOM ) |  |
| eom\_patch\_data\_unit( tileID, patchIdx ) |  |
| } |  |
| else if( ath\_type == I\_TILE ) { |  |
| if( patchMode == I\_INTRA ) |  |
| patch\_data\_unit( tileID, patchIdx ) |  |
| else if( patchMode == I\_RAW ) |  |
| raw\_patch\_data\_unit( tileID, patchIdx ) |  |
| else if( patchMode == I\_EOM ) |  |
| eom\_patch\_data\_unit( tileID, patchIdx ) |  |
| } |  |
| } |  |

#### Patch data unit syntax

|  |  |
| --- | --- |
| patch\_data\_unit( tileID, patchIdx ) { | **Descriptor** |
| **pdu\_2d\_pos\_x**[ tileID ][ patchIdx ] | ue(v) |
| **pdu\_2d\_pos\_y**[ tileID ][ patchIdx ] | ue(v) |
| **pdu\_2d\_size\_x\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| **pdu\_2d\_size\_y\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| **pdu\_3d\_offset\_u**[ tileID ][ patchIdx ] | u(v) |
| **pdu\_3d\_offset\_v**[ tileID ][ patchIdx ] | u(v) |
| **pdu\_3d\_offset\_d**[ tileID ][ patchIdx ] | u(v) |
| if( asps\_normal\_axis\_max\_delta\_value\_enabled\_flag ) |  |
| **pdu\_3d\_range\_d**[ tileID ][ patchIdx ] | u(v) |
| **pdu\_projection\_id**[ tileID ][ patchIdx ] | u(v) |
| **pdu\_orientation\_index**[ tileID ][ patchIdx ] | u(v) |
| if( afps\_lod\_mode\_enabled\_flag ) { |  |
| **pdu\_lod\_enabled\_flag**[ tileID ][ patchIdx ] | u(1) |
| if( pdu\_lod\_enabled\_flag[ tileID ][ patchIdx ] > 0 ) { |  |
| **pdu\_lod\_scale\_x\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| **pdu\_lod\_scale\_y\_idc**[ tileID ][ patchIdx ] | ue(v) |
| } |  |
| } |  |
| if( asps\_plr\_enabled\_flag ) |  |
| plr\_data( tileID, patchIdx ) |  |
| if( asps\_miv\_extension\_present\_flag ) |  |
| pdu\_miv\_extension( tileID, patchIdx ) /\* Specified in ISO/IEC 23090-12 \*/ |  |
| } |  |

#### Skip patch data unit syntax

|  |  |
| --- | --- |
| skip\_patch\_data\_unit( ) { | **Descriptor** |
| } |  |

#### Merge patch data unit syntax

|  |  |
| --- | --- |
| merge\_patch\_data\_unit( tileID, patchIdx ) { | **Descriptor** |
| if( NumRefIdxActive > 1 ) |  |
| **mpdu\_ref\_index**[ tileID ][ patchIdx ] | ue(v) |
| OverridePlrFlag = 0 |  |
| **mpdu\_override\_2d\_params\_flag**[ tileID ][ patchIdx ] | u(1) |
| if( mpdu\_override\_2d\_params\_flag[ tileID ][ patchIdx ] ) { |  |
| **mpdu\_2d\_pos\_x**[ tileID ][ patchIdx ] | se(v) |
| **mpdu\_2d\_pos\_y**[ tileID ][ patchIdx ] | se(v) |
| **mpdu\_2d\_delta\_size\_x**[ tileID ][ patchIdx ] | se(v) |
| **mpdu\_2d\_delta\_size\_y**[ tileID ][ patchIdx ] | se(v) |
| if( asps\_plr\_enabled\_flag ) |  |
| OverridePlrFlag = 1 |  |
| } else { |  |
| **mpdu\_override\_3d\_params\_flag**[ tileID ][ patchIdx ] | u(1) |
| if(mpdu\_override\_3d\_params\_flag[ tileID ][ patchIdx ] ) { |  |
| **mpdu\_3d\_offset\_u**[ tileID][ patchIdx ] | se(v) |
| **mpdu\_3d\_offset\_v**[ tileID ][ patchIdx ] | se(v) |
| **mpdu\_3d\_offset\_d**[ tileID ][ patchIdx ] | se(v) |
| if( asps\_normal\_axis\_max\_delta\_value\_enabled\_flag ) |  |
| **mpdu\_3d\_range\_d**[ tileID ][ patchIdx ] | se(v) |
| if( asps\_plr\_enabled\_flag ) { |  |
| **mpdu\_override\_plr\_flag**[ tileID ][ patchIdx ] | u(1) |
| OverridePlrFlag = mpdu\_override\_plr\_flag[ tileID ][ patchIdx ] |  |
| } |  |
| } |  |
| } |  |
| if( OverridePlrFlag && asps\_plr\_enabled\_flag ) |  |
| plr\_data( tileID, patchIdx ) |  |
| } |  |

#### Inter patch data unit syntax

|  |  |
| --- | --- |
| inter\_patch\_data\_unit( tileID, patchIdx ) { | **Descriptor** |
| if( NumRefIdxActive > 1 ) |  |
| **ipdu\_ref\_index**[ tileID ][ patchIdx ] | ue(v) |
| **ipdu\_patch\_index**[ tileID ][ patchIdx ] | se(v) |
| **ipdu\_2d\_pos\_x**[ tileID ][ patchIdx ] | se(v) |
| **ipdu\_2d\_pos\_y**[ tileID ][ patchIdx ] | se(v) |
| **ipdu\_2d\_delta\_size\_x**[ tileID ][ patchIdx ] | se(v) |
| **ipdu\_2d\_delta\_size\_y**[ tileID ][ patchIdx ] | se(v) |
| **ipdu\_3d\_offset\_u**[ tileID ][ patchIdx ] | se(v) |
| **ipdu\_3d\_offset\_v**[ tileID ][ patchIdx ] | se(v) |
| **ipdu\_3d\_offset\_d**[ tileID ][ patchIdx ] | se(v) |
| if( asps\_normal\_axis\_max\_delta\_value\_enabled\_flag ) |  |
| **ipdu\_3d\_range\_d**[ tileID ][ patchIdx ] | se(v) |
| if( asps\_plr\_enabled\_flag ) |  |
| plr\_data( tileID, patchIdx ) |  |
| } |  |

#### RAW patch data unit syntax

|  |  |
| --- | --- |
| raw\_patch\_data\_unit( tileID, patchIdx ) { | **Descriptor** |
| if( AuxTileHeight[ TileIDToIndex[ tileID ] ] > 0) |  |
| **rpdu\_patch\_in\_auxiliary\_video\_flag**[ tileID ][ patchIdx ] | u(1) |
| **rpdu\_2d\_pos\_x**[ tileID ][ patchIdx ] | ue(v) |
| **rpdu\_2d\_pos\_y**[ tileID ][ patchIdx ] | ue(v) |
| **rpdu\_2d\_size\_x\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| **rpdu\_2d\_size\_y\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| **rpdu\_3d\_offset\_u**[ tileID ][ patchIdx ] | u(v) |
| **rpdu\_3d\_offset\_v**[ tileID ][ patchIdx ] | u(v) |
| **rpdu\_3d\_offset\_d**[ tileID ][ patchIdx ] | u(v) |
| **rpdu\_points\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| } |  |

#### EOM patch data unit syntax

|  |  |
| --- | --- |
| eom\_patch\_data\_unit( tileID, patchIdx ) { | **Descriptor** |
| if( AuxTileHeight[ TileIDToIndex[ tileID ] ] > 0) |  |
| **epdu\_patch\_in\_auxiliary\_video\_flag**[ tileID ][ patchIdx ] | u(1) |
| **epdu\_2d\_pos\_x**[ tileID ][ patchIdx ] | ue(v) |
| **epdu\_2d\_pos\_y**[ tileID ][ patchIdx ] | ue(v) |
| **epdu\_2d\_size\_x\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| **epdu\_2d\_size\_y\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| **epdu\_patch\_count\_minus1**[ tileID ][ patchIdx ] | ue(v) |
| for( i = 0; i < epdu\_patch\_count\_minus1[ tileID ][ patchIdx ] + 1; i++ ) { |  |
| **epdu\_associated\_patch\_idx**[ tileID ][ patchIdx ][ i ] | ue(v) |
| **epdu\_points**[ tileID ][ patchIdx ][ i ] | ue(v) |
| } |  |
| } |  |

#### Point local reconstruction data syntax

|  |  |
| --- | --- |
| plr\_data( tileID, patchIdx ) { | **Descriptor** |
| blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] ) |  |
| for( i = 0; i < asps\_map\_count\_minus1 + 1; i++ ) { |  |
| if( plri\_map\_present\_flag[ i ] ) { |  |
| if( blockCnt > plri\_block\_threshold\_per\_patch\_minus1[ i ] + 1 ) |  |
| **plrd\_level**[ tileID ][ patchIdx ][ i ] | u(1) |
| else |  |
| plrd\_level[ tileID ][ patchIdx ][ i ] = 1 |  |
| if( plrd\_level[ tileID ][ patchIdx ][ i ] == 0 ) { |  |
| for( j = 0; j < blockCnt; j++ ) { |  |
| **plrd\_present\_block\_flag**[ tileID ][ patchIdx ][ i ][ j ] | u(1) |
| if( plrd\_present\_block\_flag[ tileID ][ patchIdx ][ i ][ j ] ) |  |
| **plrd\_block\_mode\_minus1**[ tileID ][ patchIdx ][ i ][ j ] | u(v) |
| } |  |
| } else { |  |
| **plrd\_present\_flag**[ tileID ][ patchIdx ][ i ] | u(1) |
| if( plrd\_present\_flag[ tileID ][ patchIdx ][ i ] ) |  |
| **plrd\_mode\_minus1**[ tileID ][ patchIdx ][ i ] | u(v) |
| } |  |
| } |  |
| } |  |
| } |  |

### Supplemental enhancement information message syntax

|  |  |
| --- | --- |
| sei\_message( ) { | **Descriptor** |
| payloadType = 0 |  |
| do { |  |
| **sm\_payload\_type\_byte** | u(8) |
| payloadType += sm\_payload\_type\_byte |  |
| } while( sm\_payload\_type\_byte == 0xFF ) |  |
| payloadSize = 0 |  |
| do{ |  |
| **sm\_payload\_size\_byte** | u(8) |
| payloadSize += sm\_payload\_size\_byte |  |
| } while( sm\_payload\_size\_byte == 0xFF ) |  |
| sei\_payload( payloadType, payloadSize ) |  |
| } |  |

## Semantics

### General

Semantics associated with the syntax structures and with the syntax elements within these structures are specified in this subclause. When the semantics of a syntax element are specified using a table or a set of tables, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this document.

For some applications, different encapsulation of a V3C bitstream may be used that may not utilize the V3C unit structure. As an example, in ISO/IEC DIS 23090-10[[3]](#footnote-4), the information provided by the V3C unit and V3C parameter sets may be provided through equivalent structures and the V3C unit structure may not be directly present. In such applications, all the syntax elements specified in subclauses 8.3.2 and 8.3.4, such as vuh\_unit\_type, vuh\_atlas\_id, vuh\_map\_index, vps\_atlas\_count\_minus1, etc., may be obtained through external means.

### V3C unit semantics

#### General V3C unit semantics

**trailing\_zero\_8bits** is a byte equal to 0x00.

#### V3C unit header semantics

Table 2 – V3C unit types

|  |  |  |  |
| --- | --- | --- | --- |
| **vuh\_unit\_type** | **Identifier** | **V3C unit type** | **Description** |
| 0 | V3C\_VPS | V3C parameter set | V3C level parameters |
| 1 | V3C\_AD | Atlas data | Atlas information |
| 2 | V3C\_OVD | Occupancy video data | Occupancy information |
| 3 | V3C\_GVD | Geometry video data | Geometry information |
| 4 | V3C\_AVD | Attribute video data | Attribute information |
| 5 | V3C\_PVD | Packed video data | Packing information |
| 6 | V3C\_CAD | Common atlas data | Information that is common for atlases in a CVS. Specified in ISO/IEC 23090-12 |
| 7…31 | V3C\_RSVD | Reserved | - |

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

**vuh\_v3c\_parameter\_set\_id** specifies the value of vps\_v3c\_parameter\_set\_id for the active V3C VPS. The value of vuh\_v3c\_parameter\_set\_id shall be in the range of 0 to 15, inclusive.

**vuh\_atlas\_id** specifies the ID of the atlas that corresponds to the current V3C unit. The value of vuh\_atlas\_id shall be in the range of 0 to 63, inclusive.

**vuh\_attribute\_index** indicates the index of the attribute data carried in the Attribute Video Data unit. The value of vuh\_attribute\_index shall be in the range of 0 to ( ai\_attribute\_count[ vuh\_atlas\_id ] – 1 ), inclusive.

**vuh\_attribute\_partition\_index** indicates the index of the attribute dimension group carried in the attribute video data unit. The value of vuh\_attribute\_partition\_index shall be in the range of 0 to ai\_attribute\_dimension\_partitions\_minus1[ vuh\_atlas\_id ][ vuh\_attribute\_index ], inclusive.

**vuh\_map\_index** when present, indicates the map index of the current geometry or attribute stream. When not present, the map index of the current geometry or attribute sub-bitstream is derived based on the type of the sub-bitstream and the operations described in subclauses 9.4 and 9.5 for geometry and attribute video sub-bitstreams respectively. The value of vuh\_map\_index, when present, shall be in the range of 0 to vps\_map\_count\_minus1[ vuh\_atlas\_id ], inclusive.

**vuh\_auxiliary\_video\_flag** equal to 1 indicates that the associated geometry or attribute video data unit is a RAW and/or EOM coded points video only sub-bitstream. vuh\_auxiliary\_video\_flag equal to 0 indicates that the associated geometry or attribute video data unit may contain RAW and/or EOM coded points. When vuh\_auxiliary\_video\_flag is not present, its value shall be inferred to be equal to 0.

**vuh\_reserved\_zero\_12bits**, when present, shall be equal to 0 in bitstreams conforming to this version of this document. Other values for vuh\_reserved\_zero\_12bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of vuh\_reserved\_zero\_12bits.

**vuh\_reserved\_zero\_17bits**, when present, shall be equal to 0 in bitstreams conforming to this version of this document. Other values for vuh\_reserved\_zero\_17bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of vuh\_reserved\_zero\_17bits.

**vuh\_reserved\_zero\_23bits**, when present, shall be equal to 0 in bitstreams conforming to this version of this document. Other values for vuh\_reserved\_zero\_23bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of vuh\_reserved\_zero\_23bits.

**vuh\_reserved\_zero\_27bits**, when present, shall be equal to 0 in bitstreams conforming to this version of this document. Other values for vuh\_reserved\_zero\_27bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of vuh\_reserved\_zero\_27bits.

#### V3C unit payload semantics

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

#### Atlas sub-bitstream semantics

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

#### Order of V3C units and association to coded information

##### General

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

##### Order of VPSs and their activation

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

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

A VPS includes parameters that can be referred to by one or more V3C units and their associated vuh\_v3c\_parameter\_set\_id. Each VPS is initially considered not active at the start of the operation of the decoding process. At most, one VPS is considered active at any given moment during the operation of the decoding process and the activation of any particular VPS results in the deactivation of the previously active VPS.

When a VPS (with a particular value of vps\_v3c\_parameter\_set\_id) is not already active for a particular V3C unit and it is referred to by a V3C unit (in which the associated vuh\_v3c\_parameter\_set\_id is equal to vps\_v3c\_parameter\_set\_id), it is activated for the particular V3C unit. This VPS is called the active VPS for the particular V3C unit until it is deactivated by the activation of another VPS. A VPS, with that particular value of vps\_v3c\_parameter\_set\_id, shall be available to the decoding process prior to its activation, included in at least one V3C unit or provided through external means. An activated VPS shall remain active for the entire coded V3C sequence (CVS). The activation of a new VPS starts a new CVS.

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

During operation of the decoding process (see Clause 9), the values of parameters of the active VPS are considered in effect.

##### Order of V3C units and association to CVSs

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

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

For atlas sub-bitstreams the IRAP sub-bitstream unit corresponds to a coded atlas access unit for which each ACL NAL has nal\_unit\_type in the range of NAL\_GBLA\_W\_LP to NAL\_GBLA\_N\_LP, or is in the range of NAL\_GIDR\_W\_RADL to NAL\_GIDR\_N\_LP, or is equal to NAL\_GCRA, inclusive.

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

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

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

### Byte alignment semantics

**alignment\_bit\_equal\_to\_one** shall be equal to 1.

**alignment\_bit\_equal\_to\_zero** shall be equal to 0.

### V3C parameter set semantics

#### General V3C parameter set semantics

**vps\_v3c\_parameter\_set\_id** provides an identifier for the V3C VPS for reference by other syntax elements. The value of vps\_v3c\_parameter\_set\_id shall be in the range of 0 to 15, inclusive.

**vps\_reserved\_zero\_8bits**, when present, shall be equal to 0 in bitstreams conforming to this version of this document. Other values for vps\_reserved\_zero\_8bits are reserved for future use ISO/IEC. Decoders shall ignore the value of vps\_reserved\_zero\_8bits.

**vps\_atlas\_count\_minus1** plus 1 indicates the total number of supported atlases in the current bitstream. The value of vps\_atlas\_count\_minus1 shall be in the range of 0 to 63, inclusive.

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

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

**vps\_frame\_height**[ j ] indicates the V3C frame height in terms of integer luma samples for the atlas with atlas ID j. This frame height is the nominal height that is associated with all V3C components for the atlas with atlas ID j.

**vps\_map\_count\_minus1**[ j ] plus 1 indicates the number of maps used for encoding the geometry and attribute data for the atlas with atlas ID j. vps\_map\_count\_minus1[ j ] shall be in the range of 0 to 15, inclusive.

**vps\_multiple\_map\_streams\_present\_flag**[ j ] equal to 0 indicates that all geometry or attribute maps for the atlas with atlas ID j are placed in a single geometry or attribute video stream, respectively. vps\_multiple\_map\_streams\_present\_flag[ j ] equal to 1 indicates that all geometry or attribute maps for the atlas with atlas ID j are placed in separate video streams. When vps\_multiple\_map\_streams\_present\_flag[ j ] is not present, its value shall be inferred to be equal to 0.

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

**vps\_map\_predictor\_index\_diff**[ j ][ i ] is used to compute the predictor of the geometry map with index i for the atlas with atlas ID j when vps\_map\_absolute\_coding\_enabled\_flag[ j ][ i ] is equal to 0. More specifically, the map predictor index for map i, MapPredictorIndex[ i ], shall be computed as:

MapPredictorIndex[ i ] = (i – 1) – vps\_map\_predictor\_index\_diff[ j ][ i ] (15)

The value of vps\_map\_predictor\_index\_diff[ j ][ i ] shall be in the range from 0 to i – 1, inclusive. When vps\_map\_predictor\_index\_diff[ j ][ i ] is not present, its value shall be inferred to be equal to 0.

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

**vps\_occupancy\_video\_present\_flag**[ j ]equal to 0 indicates that the atlas with atlas ID j does not have occupancy video data associated with it. vps\_occupancy**\_**video\_present\_flag[ j ] equal to 1 indicates that the atlas with atlas ID j shall have occupancy video data associated with it. When vps\_occupancy\_video\_present\_flag[ j ]is not present, it is inferred to be equal to 1.

**vps\_geometry\_video\_present\_flag**[ j ]equal to 0 indicates that the atlas with atlas ID j does not have geometry video data associated with it. vps\_geometry\_video\_present\_flag[ j ]equal to 1 indicates that the atlas with atlas ID j shall have geometry video data associated with it. When vps\_geometry\_video\_present\_flag[ j ]is not present, it is inferred to be equal to 1.

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

**vps\_extension\_present\_flag** equal to 1 specifies that the syntax elements vps\_packing\_information\_present\_flag, vps\_miv\_extension\_present\_flag, and vps\_extension\_6bits are present in the v3c\_parameter\_set( ) syntax structure. vps\_extension\_present\_flag equal to 0 specifies that these syntax elements are not present.

**vps\_packing\_information\_present\_flag** equal to 1 specifies that one or more instances of the packing\_information( j ) syntax structure are present in the v3c\_parameter\_set( ) syntax structure. vps\_packing\_information\_present\_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of vps\_packing\_information\_present\_flag is inferred to be equal to 0.

**vps\_miv\_extension\_present\_flag** equal to 1 specifies that the vps\_miv\_extension( ) syntax structure is present in the v3c\_parameter\_set( ) syntax structure. vps\_miv\_extension\_present\_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of vps\_miv\_extension\_present\_flag is inferred to be equal to 0.

**vps\_extension\_6bits** not equal to 0 specifies that the syntax element vps\_extension\_length is present in the v3c\_parameter\_set( ) syntax structure. vps\_extension\_6bits equal to 0 specifies that the syntax element vps\_extension\_length\_minus1 is not present. vps\_extension\_6bits shall be equal to 0 in bitstreams conforming to this version of this document. Other values of vps\_extension\_6bits are reserved for future use by ISO/IEC.

**vps\_extension\_length\_minus1** plus 1 specifies the number of vps\_extension\_data\_byte elements that follow this syntax element.

**vps\_extension\_data\_byte** may have any value.

#### Profile, tier, and level semantics

**ptl\_tier\_flag** specifies the tier context for the interpretation of ptl\_level\_idc as specified in Annex A.

**ptl\_profile\_codec\_group\_idc** indicates the codec group profile component to which the CVS conforms as specified in Annex A. Bitstreams shall not contain values of ptl\_profile\_codec\_group\_idc other than those specified in Annex A. Other values of ptl\_profile\_codec\_group\_idc are reserved for future use by ISO/IEC.

**ptl\_profile\_toolset\_idc** indicates the toolset combination profile component to which the CVS conforms as specified in Annex A. Bitstreams shall not contain values of ptl\_profile\_toolset\_idc other than those specified in Annex A. Other values of ptl\_profile\_toolset\_idc are reserved for future use by ISO/IEC.

**ptl\_profile\_reconstruction\_idc** indicates the reconstruction profile component to which the CVS is recommended to conform to as specified in Annex A. Decoders may select to use a different reconstruction profile than the one indicated in the bitstream. Bitstreams shall not contain values of ptl\_profile\_reconstruction\_idc other than those specified in Annex A. Other values of ptl\_profile\_reconstruction\_idc are reserved for future use by ISO/IEC.

**ptl**\_**reserved\_zero\_16bits**, when present, shall be equal to 0 in bitstreams conforming to this version of this document. Other values for ptl\_reserved\_zero\_16bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ptl\_reserved\_zero\_16bits.

**ptl\_max\_decodes\_idc** indicates on the number of sub-bitstreams requiring a video decoder instantiation to which the CVS conforms as specified in Annex A. Bitstreams shall not contain values of ptl\_max\_decodes\_idc other than those specified in Annex A. Other values of ptl\_max\_decodes\_idc are reserved for future use by ISO/IEC.

**ptl\_reserved\_0xfff\_12bits**, when present, shall be equal to 0xFFF in bitstreams conforming to this version of this document. Other values for ptl\_reserved\_0xfff\_12bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ptl\_reserved\_0xfff\_12bits.

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

**ptl\_num\_sub\_profiles** indicates the number of the ptl\_sub\_profile\_idc[ i ] syntax elements.

**ptl\_extended\_sub\_profile\_flag** equal to 1 specifies that the ptl\_sub\_profile\_idc[ i ] syntax elements, if present, should be represented using 64 bits. ptl\_extended\_sub\_profile\_flag equal to 0 specifies that the ptl\_sub\_profile\_idc[ i ] syntax elements, if present, should be represented using 32 bits.

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

**ptl\_toolset\_constraints\_present\_flag** equal to 1 specifies that an additional structure, profile\_toolset\_constraints\_information( ), is present in the bitstream. ptl\_toolset\_constraints\_present\_flag equal to 0 specifies that the structure profile\_toolset\_constraints\_information( ) is not present.

#### Occupancy information semantics

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

**oi\_lossy\_occupancy\_compression\_threshold**[ j ] indicates the threshold to be used to derive the binary occupancy from the decoded occupancy video for the atlas with atlas ID j. oi\_lossy\_occupancy\_compression\_threshold[ j ] shall be in the range of 0 to 255, inclusive.

**oi\_occupancy\_2d\_bit\_depth\_minus1**[ j ] plus 1 indicates the nominal 2D bit depth to which the occupancy video for the atlas with atlas ID j shall be converted to. oi\_occupancy\_2d\_bit\_depth\_minus1[ j ] shall be in the range of 0 to 31, inclusive.

**oi\_occupancy\_MSB\_align\_flag**[ j ] indicates how the decoded occupancy video samples associated with an atlas with atlas ID j are converted to samples at the nominal occupancy bit depth, as specified in Annex B.

#### Geometry information semantics

**gi\_geometry\_codec\_id**[ j ]indicates the identifier of the codec used to compress the geometry video data for the atlas with atlas ID j. geometry\_codec\_id[ j ] shall be in the range of 0 to 255, inclusive. This codec may be identified through the profiles defined in Annex A, a component codec mapping SEI message, or through means outside this document.

**gi\_geometry\_2d\_bit\_depth\_minus1**[ j ] plus 1 indicates the nominal 2D bit depth to which all geometry videos for the atlas with atlas ID j shall be converted to. gi\_geometry\_2d\_bit\_depth\_minus1[ j ] shall be in the range of 0 to 31, inclusive.

**gi\_geometry\_MSB\_align\_flag**[ j ] indicates how the decoded geometry video samples associated with an atlas with atlas ID j are converted to samples at the nominal geometry bit depth, as specified in Annex B.

**gi\_geometry\_3d\_coordinates\_bit\_depth\_minus1**[ j ] plus 1 indicates the bit depth of the geometry coordinates of the reconstructed volumetric content for the atlas with atlas ID j. gi\_geometry\_3d\_coordinates\_bit\_depth\_minus1[ j ]shall be in the range of 0 to 31, inclusive.

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

#### Attribute information semantics

**ai\_attribute\_count**[ j ]indicates the number of attributes associated with the atlas with atlas ID j. ai\_attribute\_count[ j ] shall be in the range of 0 to 127, inclusive.

**ai\_attribute\_type\_id**[ j ][ i ] indicates the attribute type of the Attribute Video Data unit with index i for the atlas with atlas ID j. Table 3 describes the list of supported attributes and their relationship with ai\_attribute\_type\_id[ j ][ i ].

Table 3 – V3C attribute types

|  |  |  |
| --- | --- | --- |
| **ai\_attribute\_type\_id[ j ][ i ]** | **Identifier** | **Attribute type** |
| 0 | ATTR\_TEXTURE | Texture |
| 1 | ATTR\_MATERIAL\_ID | Material ID |
| 2 | ATTR\_TRANSPARENCY | Transparency |
| 3 | ATTR\_REFLECTANCE | Reflectance |
| 4 | ATTR\_NORMAL | Normals |
| 5..14 | ATTR\_RESERVED | Reserved |
| 15 | ATTR\_UNSPECIFIED | Unspecified |

ATTR\_TEXTURE indicates an attribute that contains texture information of a volumetric frame. For example, this may indicate an attribute that contains RGB (Red, Green, Blue) colour information.

ATTR\_MATERIAL\_ID indicates an attribute that contains supplemental information that identifies the material type of a point in a volumetric frame. For example, the material type could be used as an indicator for identifying an object or the characteristic of a point within a volumetric frame. The interpretation of the values of such attribute frame type is outside the scope of this document.

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

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

ATTR\_NORMAL indicates an attribute that contains a unit vector information associated with each point in a volumetric frame. The unit vector specifies the perpendicular direction to a surface at a point (i.e. direction a point is facing). An attribute frame with this attribute type shall have ai\_attribute\_dimension\_minus1 equal to 2. Each channel of an attribute frame with this attribute type shall contain one component of the unit vector (x, y, z), where the first component contains the x coordinate, the second component contains the y coordinate, and the third component contains the z coordinate.

ATTR\_UNSPECIFIED indicates an attribute that contains values that have no specified meaning in this document and will not have a specified meaning in the future as an integral part of this document.

Values indicated as ATTR\_RESERVED are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this version of this document. Decoders conforming to this version of this document should ignore attribute sub-bitstreams with an attribute type a value ai\_attribute\_type\_id[ j ][ i ] equal to ATTR\_RESERVED.

**ai\_attribute\_codec\_id**[ j ][ i ]indicates the identifier of the codec used to compress the attribute video data with index i for the atlas with atlas ID j. ai\_attribute\_codec\_id[ j ][ i ] shall be in the range of 0 to 255, inclusive. This codec may be identified through the profiles defined in Annex A, a component codec mapping SEI message, or through means outside this document.

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

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

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

if( ai\_attribute\_map\_absolute\_coding\_persistance\_flag[ j ][ i ] == 1) {  
 for( k = 0; k < vps\_map\_count\_minus1[ j ]; k++ )  
 AttributeMapAbsoluteCodingEnabledFlag[ j ][ i ][ k ] = 1  
 }  
 else{  
 for( k = 0; k < vps\_map\_count\_minus1[ j ]; k++ )  
 AttributeMapAbsoluteCodingEnabledFlag[ j ][ i ][ k ] =  
 vps\_map\_absolute\_coding\_enabled\_flag[ j ][ i ]  
 }

**ai\_attribute\_dimension\_minus1**[ j ][ i ] plus 1 indicates the total number of dimensions (i.e., number of channels) of the attribute with index i, for the atlas with atlas ID j. ai\_attribute\_dimension\_minus1[ j ][ i ]shall be in the range of 0 to 63, inclusive.

**ai\_attribute\_dimension\_partitions\_minus1**[ j ][ i ] plus 1 indicates the number of partition groups in which the attribute channels of the attribute with index i, for the atlas with atlas ID j, should be grouped in. ai\_attribute\_dimension\_partitions\_minus1[ j ][ i ]shall be in the range of 0 to 63, inclusive.

**ai\_attribute\_partition\_channels\_minus1**[ k ][ i ][ j ] plus 1 indicates the number of channels assigned to the dimension partition group with index j, of the attribute with index i, for the atlas with atlas ID k. ai\_attribute\_partition\_channels\_minus1[ k ][ i ][ j ]shall be in the range of 0 to ai\_attribute\_dimension\_minus1[ k ][ i ], inclusive, for all dimension partition groups.

**ai\_attribute\_2d\_bit\_depth\_minus1**[ j ][ i ] plus 1 indicates the nominal 2D bit depth to which all the attribute videos with attribute index i, for the atlas with atlas ID j, shall be converted to. ai\_attribute\_2d\_bit\_depth\_minus1[ j ][ i ] shall be in the range of 0 to 31, inclusive.

**ai\_attribute\_MSB\_align\_flag**[ j ][ i ] indicates how the decoded attribute video samples with attribute index i, for the atlas with atlas ID j, are converted to samples at the nominal attribute bit depth, as specified in Annex B.

#### Profile toolset constraints information semantics

**ptc\_one\_v3c\_frame\_only\_flag,** when present, has semantics specified in Annex A, where the profile indicated by ptl\_profile\_pcc\_toolset\_idc is a profile specified in Annex A. When not present, ptc\_one\_v3c\_frame\_only\_flag is inferred to be equal to 0.

**ptc\_eom\_constraint\_flag** equal to 1 specifies that the parameter asps\_eom\_patch\_enabled\_flag shall be equal to 0. ptc\_eom\_constraint\_flag equal to 0 does not impose a constraint. When not present, ptc\_eom\_constraint\_flag is inferred to be equal to 0.

**ptc\_max\_map\_count\_minus1** specifies that vps\_map\_count\_minus1 shall be less than or equal to ptc\_max\_map\_count\_minus1. When not present, ptc\_max\_map\_count\_minus1 is inferred to be equal to 15.

**ptc\_max\_atlas\_count\_minus1** specifies that vps\_atlas\_count\_minus1 shall be less than or equal to ptc\_max\_atlas\_count\_minus1. When not present, ptc\_max\_atlas\_count\_minus1 is inferred to be equal to 63.

**ptc\_multiple\_map\_streams\_constraint\_flag** equal to 1 specifies that vps\_multiple\_map\_streams\_present\_flag shall be equal to 0. ptc\_multiple\_map\_streams\_constraint\_flag equal to 0 does not impose a constraint. When not present, ptc\_multiple\_map\_streams\_constraint\_flag is inferred to be equal to 0.

**ptc\_plr\_constraint\_flag** equal to 1 specifies that asps\_plr\_enabled\_flag shall be equal to 0. ptc\_plr\_constraint\_flag equal to 0 does not impose a constraint. When not present, ptc\_plr\_constraint\_flag is inferred to be equal to 0.

**ptc\_attribute\_max\_dimension\_minus1** specifies that ai\_attribute\_dimension\_minus1 shall be less than or equal to ptc\_attribute\_max\_dimension\_minus1. When not present, ptc\_attribute\_max\_dimension\_minus1 is inferred to be equal to 63.

**ptc\_attribute\_max\_dimension\_partitions\_minus1** specifies that ai\_attribute\_dimension\_partitions\_minus1 shall be less than or equal to ptc\_attribute\_max\_dimension\_partitions\_minus1. When not present, ptc\_attribute\_max\_dimension\_partitions\_minus1 is inferred to be equal to 63.

**ptc\_no\_eight\_orientations\_constraint\_flag** equal to 1 specifies that asps\_use\_eight\_orientations\_flag shall be equal to 0. ptc\_no\_eight\_orientations\_constraint\_flag equal to 0 does not impose a constraint. When not present, ptc\_no\_eight\_orientations\_constraint\_flag is inferred to be equal to 0.

**ptc\_45degree\_no\_projection\_patch\_constraint\_flag** equal to 1 specifies that asps\_extended\_projection\_enabled\_flag shall be equal to 0.

ptc\_45degree\_no\_projection\_patch\_constraint\_flag equal to 0 does not impose a constraint. When not present, ptc\_45degree\_no\_projection\_patch\_constraint\_flag is inferred to be equal to 0.

**ptc\_reserved\_zero\_6bits** shall be equal to 0 in bitstreams conforming to this version of this document. Other values of ptc\_reserved\_zero\_6bits are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this version of this document. Decoders conforming to this version of this document shall ignore values of ptc\_reserved\_zero\_6bitsother than 0.

**ptc\_num\_reserved\_constraint\_bytes** specifies the number of the reserved constraint bytes. The value of ptc\_num\_reserved\_constraint\_bytes shall be 0 in bitstreams conforming to this version of this document. Other values of ptc\_num\_reserved\_constraint\_bytes are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this version of this document. Decoders conforming to this version of this document shall ignore values of ptc\_num\_reserved\_constraint\_bytes other than 0.

**ptc\_reserved\_constraint\_byte**[ i ] may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this document. Decoders conforming to this version of this document shall ignore the values of all the ptc\_reserved\_constraint\_byte[ i ] syntax elements.

#### Packing information semantics

Packed video frame can be divided into one or more rectangular regions. One region shall be mapped to exactly one atlas tile. Rectangular regions of packed video frame are not allowed to overlap.

**pin\_codec\_id**[ j ] indicates the identifier of the codec used to compress the packed video data for the atlas with ID j. pin\_codec\_id shall be in the range of 0 to 255, inclusive. This codec may be identified through a component codec mapping SEI message or through means outside this document.

**pin\_regions\_count\_minus1**[ j ]plus 1 indicates the number of regions packed in one video frame for the atlas with ID j. pin\_regions\_count\_minus1 shall be in the range of 0 to 7, inclusive. When not present, the value of pin\_regions\_count\_minus1 is inferred to be equal to 0.

**pin\_region\_tile\_id**[ j ][ i ]indicates the atlas tile ID, of the atlas with ID j, corresponding to the region with index i.

**pin\_region\_type\_id\_minus2**[ j ][ i ]plus 2indicates the ID of the region with index i for the atlas with ID j. The value of pin\_region\_type\_id\_minus2[ j ][ i ]shall be in the range of 0 to 2, inclusive.

**pin\_region\_top\_left\_x**[ j ][ i ]specifies horizontal position of top left sample of the region with index i for the atlas with ID j in unit of luma samples in the packed video component frame. When not present, the value of pin\_region\_top\_left\_x[ j ][ i ] is inferred to be equal to 0.

**pin\_region\_top\_left\_y**[ j ][ i ]specifies vertical position of top left sample of the region with index i for the atlas with ID j in unit of luma samples in in the packed video component frame. When not present, the value of pin\_region\_top\_left\_y[ j ][ i ] is inferred to be equal to 0.

**pin\_region\_width\_minus1**[ j ][ i ] plus 1 specifies the width of the the region with index i for the atlas with ID j in units luma samples.

**pin\_region\_height\_minus1**[ j ][ i ] plus 1 specifies the height of the the region with index i for the atlas with ID j in units of luma samples.

**pin\_region\_map\_index**[ j ][ i ] specifies the map index of the region with index i of the atlas with ID j.

**pin\_region\_rotation\_flag**[ j ][ i ] equal to 0 indicates that no rotation on the region with index i for the atlas with ID j is performed. pin\_region\_rotation\_flag[ j ][ i ] equal to 1 indicates that the region with index i for the atlas with ID j is rotated 90 degrees.

**pin\_region\_auxiliary\_data\_flag**[ j ][ i ] equal to 1 indicates that the region with index i of the atlas with ID j contains RAW and/or EOM coded points only. pin\_region\_auxiliary\_data\_flag equal to 0 indicates that the region with index i of the atlas with ID j may contains RAW and/or EOM coded points.

**pin\_region\_attr\_type\_id**[ j ][ i ] indicates the attribute type of the region with index i of the atlas with ID j. Table 7.2 of [V3C] describes the list of supported attributes.

**pin\_region\_attr\_partitions\_flag**[ j ][ i ] equal to 1 indicates that the attribute with type pin\_region\_attr\_type\_id[ j ][ i ] contained in the region with index i of the atlas with ID j is part of an attribute with more than one partition. pin\_region\_attr\_partitions\_flag[ j ][ i ] equal to 0 indicates that the attribute with type pin\_region\_attr\_type\_id[ j ][ i ] has only one partition.

**pin\_region\_attr\_partition\_index**[ j ][ i ] indicates the attribute partition index of the region with index i of the atlas with ID j. When not present, the value of pin\_region\_attr\_partition\_index[ j ][ i ] is inferred to be equal to 0.

**pin\_attr\_partitions\_minus1**[ j ][ i ] plus 1 indicates the number of partitions for the attribute with type ID pin\_region\_attr\_type\_id[ j ][ i ] for the atlas with ID j. pin\_attr\_partitions\_minus1[ j ][ i ]shall be in the range of 0 to 63, inclusive.

### NAL unit semantics

#### General NAL unit semantics

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

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

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

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

* If the SODB is empty (i.e., zero bits in length), the RBSP is also empty.
* Otherwise, the RBSP contains the SODB as follows:

1) The first byte of the RBSP contains the first (most significant, left-most) eight bits of the SODB; the next byte of the RBSP contains the next eight bits of the SODB, etc., until fewer than eight bits of the SODB remain.

2) The rbsp\_trailing\_bits( ) syntax structure is present after the SODB as follows:

i) The first (most significant, left-most) bits of the final RBSP byte contain the remaining bits of the SODB (if any).

ii) The next bit consists of a single bit equal to 1 (i.e., rbsp\_stop\_one\_bit).

iii) When the rbsp\_stop\_one\_bit is not the last bit of a byte-aligned byte, one or more bits equal to 0 (i.e. instances of rbsp\_alignment\_zero\_bit) are present to result in byte alignment.

Syntax structures having these RBSP properties are denoted in the syntax tables using an "\_rbsp" suffix. These structures are carried within NAL units as the content of the rbsp\_byte[ i ] data bytes. The association of the RBSP syntax structures to the NAL units is as specified in Table 4.

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

#### NAL unit header semantics

**nal\_forbidden\_zero\_bit** shall be equal to 0.

**nal\_unit\_type** specifies the type of the RBSP data structure contained in the NAL unit as specified in Table 4.

NAL units that have nal\_unit\_type in the range of NAL\_UNSPEC\_53..NAL\_UNSPEC\_63, inclusive, for which semantics are not specified, shall not affect the decoding process specified in this document.

NOTE 1 – NAL unit types in the range of NAL\_UNSPEC\_53..NAL\_UNSPEC\_63 can be used as determined by the application. No decoding process for these values of nal\_unit\_type is specified in this document. Since different applications can use these NAL unit types for different purposes, particular care should be exercised in the design of encoders that generate NAL units with these nal\_unit\_type values, and in the design of decoders that interpret the content of NAL units with these nal\_unit\_type values. This document does not define any management for these values. These nal\_unit\_type values could only be suitable for use in contexts in which "collisions" of usage (i.e., different definitions of the meaning of the NAL unit content for the same nal\_unit\_type value) are unimportant, or not possible, or are managed – e.g., defined or managed in the controlling application or transport specification, or by controlling the environment in which bitstreams are distributed.

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

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

Table 4 – NAL unit type codes and NAL unit type classes

|  |  |  |  |
| --- | --- | --- | --- |
| **nal\_unit\_type** | **Name of nal\_unit\_type** | **Content of NAL unit and RBSP syntax structure** | **NAL unit type class** |
| 0  1 | NAL\_TRAIL\_N  NAL\_TRAIL\_R | Coded tile of a non-TSA, non STSA trailing atlas frame  atlas\_tile\_layer\_rbsp( ) | ACL |
| 2  3 | NAL\_TSA\_N  NAL\_TSA\_R | Coded tile of a TSA atlas frame  atlas\_tile\_layer\_rbsp( ) | ACL |
| 4  5 | NAL\_STSA\_N  NAL\_STSA\_R | Coded tile of a STSA atlas frame  atlas\_tile\_layer\_rbsp( ) | ACL |
| 6  7 | NAL\_RADL\_N  NAL\_RADL\_R | Coded tile of a RADL atlas frame  atlas\_tile\_layer\_rbsp( ) | ACL |
| 8  9 | NAL\_RASL\_N  NAL\_RASL\_R | Coded tile of a RASL atlas frame  atlas\_tile\_layer\_rbsp( ) | ACL |
| 10  11 | NAL\_SKIP\_N  NAL\_SKIP\_R | Coded tile of a skipped atlas frame  atlas\_tile\_layer\_rbsp( ) | ACL |
| 12  14 | NAL\_RSV\_ACL\_N12 NAL\_RSV\_ACL\_N14 | Reserved non-IRAP sub-layer non-reference ACL NAL unit types | ACL |
| 13  15 | NAL\_RSV\_ACL\_R13 NAL\_RSV\_ACL\_R15 | Reserved non-IRAP sub-layer reference ACL NAL unit types | ACL |
| 16 17 18 | NAL\_BLA\_W\_LP NAL\_BLA\_W\_RADL NAL\_BLA\_N\_LP | Coded tile of a BLA atlas frame that is not a GBLA atlas\_tile\_layer\_rbsp( ) | ACL |
| 19 20 21 | NAL\_GBLA\_W\_LP NAL\_GBLA\_W\_RADL NAL\_GBLA\_N\_LP | Coded tile of a GBLA atlas frame atlas\_tile\_layer\_rbsp( ) | ACL |
| 22 23 | NAL\_IDR\_W\_RADL NAL\_IDR\_N\_LP | Coded tile of an IDR atlas frame that is not a GIDR  atlas\_tile\_layer\_rbsp( ) | ACL |
| 24 25 | NAL\_GIDR\_W\_RADL NAL\_GIDR\_N\_LP | Coded tile of a GIDR atlas frame  atlas\_tile\_layer\_rbsp( ) | ACL |
| 26 | NAL\_CRA | Coded tile of a CRA atlas frame that is not a GCRA atlas\_tile\_layer\_rbsp( ) | ACL |
| 27 | NAL\_GCRA | Coded tile of a GCRA atlas frame atlas\_tile\_layer\_rbsp( ) | ACL |
| 28 29 | NAL\_RSV\_IRAP\_ACL\_28 NAL\_RSV\_IRAP\_ACL\_29 | Reserved IRAP ACL NAL unit types | ACL |
| 30..35 | NAL\_RSV\_ACL\_30.. NAL\_RSV\_ACL\_35 | Reserved non-IRAP ACL NAL unit types | ACL |
| 36 | NAL\_ASPS | Atlas sequence parameter set atlas\_sequence\_parameter\_set\_rbsp( ) | non-ACL |
| 37 | NAL\_AFPS | Atlas frame parameter set atlas\_frame\_parameter\_set\_rbsp( ) | non-ACL |
| 38 | NAL\_AUD | Access unit delimiter access\_unit\_delimiter\_rbsp( ) | non-ACL |
| 39 | NAL\_V3C\_AUD | V3C access unit delimiter access\_unit\_delimiter\_rbsp( ) | non-ACL |
| 40 | NAL\_EOS | End of sequence end\_of\_seq\_rbsp( ) | non-ACL |
| 41 | NAL\_EOB | End of bitstream end\_of\_atlas\_sub\_bitstream\_rbsp( ) | non-ACL |
| 42 | NAL\_FD | Filler filler\_data\_rbsp( ) | non-ACL |
| 43 44 | NAL\_PREFIX\_NSEI  NAL\_SUFFIX\_NSEI | Non-essential supplemental enhancement information sei\_rbsp( ) | non-ACL |
| 45 46 | NAL\_PREFIX\_ESEI NAL\_SUFFIX\_ESEI | Essential supplemental enhancement information sei\_rbsp( ) | non-ACL |
| 47 | NAL\_AAPS | Atlas adaptation parameter set atlas\_adaptation\_parameter\_set\_rbsp( ) | non-ACL |
| 48 | NAL\_CAF | Common atlas frame common\_atlas\_frame\_rbsp( ) | non-ACL |
| 49..52 | NAL\_RSV\_NACL\_48 NAL\_RSV\_NACL\_52 | Reserved non-ACL NAL unit types | non-ACL |
| 53..63 | NAL\_UNSPEC\_53 NAL\_UNSPEC\_63 | Unspecified non-ACL NAL unit types | non-ACL |

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

NOTE 4 – A broken link access (BLA) atlas frame having nal\_unit\_type equal to NAL\_BLA\_W\_LP or NAL\_GBLA\_W\_LP, can have associated RASL or RADL atlas frames present in the bitstream. A BLA atlas frame having nal\_unit\_type equal to NAL\_BLA\_W\_RADL or NAL\_GBLA\_W\_RADL, does not have associated RASL atlas frames present in the bitstream, but can have associated RADL atlas frames in the bitstream. A BLA atlas frame having nal\_unit\_type equal to NAL\_BLA\_N\_LP or NAL\_GBLA\_N\_LP, does not have associated leading atlas frames present in the bitstream.

NOTE 5 – An instantaneous decoding refresh (IDR) atlas frame having nal\_unit\_type equal to NAL\_IDR\_N\_LP or NAL\_GIDR\_N\_LP, does not have associated leading atlas frames present in the bitstream. An IDR atlas frame having nal\_unit\_type equal to NAL\_IDR\_W\_RADL or NAL\_GIDR\_W\_RADL, does not have associated RASL atlas frames present in the bitstream, but can have associated RADL atlas frames in the bitstream.

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

If an atlas frame has nal\_unit\_type equal to NAL\_TRAIL\_N, NAL\_TSA\_N, NAL\_STSA\_N, NAL\_RADL\_N, NAL\_RASL\_N, NAL\_RSV\_ACL\_N12, or NAL\_RSV\_ACL\_N14, the atlas frame is an SLNR atlas frame. Otherwise, the atlas frame is a sub-layer reference atlas frame.

If an atlas frame has nal\_unit\_type equal to NAL\_GIDR\_W\_RADL, NAL\_GBLA\_N\_LP, or NAL\_GCRA, or in the range of NAL\_GBLA\_W\_LP to NAL\_GBLA\_N\_LP, inclusive, it specifies a random access association between the current coded atlas and the corresponding coded video frames at the same composition time.

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

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

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

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

No RASL atlas frames shall be present in the bitstream that are associated with a BLA atlas frame having nal\_unit\_type equal to NAL\_BLA\_W\_RADL or NAL\_BLA\_N\_LP.

No RASL atlas frames shall be present in the bitstream that are associated with a GBLA atlas frame having nal\_unit\_type equal to NAL\_GBLA\_W\_RADL or NAL\_GBLA\_N\_LP.

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

No RADL atlas frames shall be present in the bitstream that are associated with a BLA atlas frame having nal\_unit\_type equal to NAL\_BLA\_N\_LP or NAL\_GBLA\_N\_LP, an IDR atlas frame having nal\_unit\_type equal to NAL\_IDR\_N\_LP or NAL\_GIDR\_N\_LP.

NOTE 6 – It is possible to perform random access at the position of an IRAP coded atlas access unit by discarding all access units before the IRAP coded atlas access unit (and to correctly decode the IRAP coded atlas frame and all the subsequent non-RASL atlas frames in decoding order), provided each parameter set is available (either in the bitstream or by external means not specified in this document) when it needs to be activated.

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

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

A nal\_unit\_type equal to NAL\_V3C\_AUD that is associated with a particular atlas frame j means that all output V3C component frames, if present in their corresponding sub-bitstreams, and that have the same output time order as the atlas frame j, shall have the same decoding order, while accounting for missing frames in each sub-bitstream, as that of the atlas frame.

NOTE 7 – An application can specify that all atlas frames in a V3C sequence are delimited using a nal\_unit\_type equal to NAL\_V3C\_AUD, in which case all V3C sub-bitstreams including sub-bitstreams that correspond to different maps will be aligned in both decoding order and output time.

NOTE 8 – If a frame in a sub-bitstream is missing, and a NAL\_V3C\_AUD is encountered it can be assumed that this missing frame is virtually present in its sub-bitstream and has the same decoding order as all other corresponding frames in the other sub-bitstreams. This frame can then be generated using such methods as described in Annex B.

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

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

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

NOTE 10 – The value of 63 for nal\_layer\_id can be used to indicate an extended layer identifier in a future extension of this document.

The value of nal\_layer\_id shall be the same for all ACL NAL units of a coded atlas frame. The value of nal\_layer\_id of a coded atlas frame is the value of the nal\_layer\_id of the ACL NAL units of the coded atlas frame.

When nal\_unit\_type is equal to NAL\_EOB, the value of nal\_layer\_id shall be equal to 0.

**nal\_temporal\_id\_plus1** minus 1 specifies a temporal identifier for the NAL unit. The value of nal\_temporal\_id\_plus1 shall not be equal to 0.

The variable TemporalID is specified as follows:

TemporalID = nal\_temporal\_id\_plus1 – 1 (16)

When nal\_unit\_type is in the range of NAL\_BLA\_W\_LP to NAL\_RSV\_IRAP\_ACL29, inclusive, i.e., the coded tile belongs to an IRAP coded atlas frame, TemporalID shall be equal to 0.

When nal\_unit\_type is equal to NAL\_TSA\_R or NAL\_TSA\_N, TemporalID shall not be equal to 0.

When nal\_layer\_id is equal to 0 and nal\_unit\_type is equal to NAL\_STSA\_R or NAL\_STSA\_N, TemporalID shall not be equal to 0.

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

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

– If nal\_unit\_type is equal to NAL\_ASPS, TemporalID shall be equal to 0 and the TemporalID of the access unit containing the NAL unit shall be equal to 0.

– Otherwise, if nal\_unit\_type is equal to NAL\_EOS or NAL\_EOB, TemporalID shall be equal to 0.

– Otherwise, if nal\_unit\_type is equal to NAL\_AUD, NAL\_V3C\_AUD, or NAL\_FD, TemporalID shall be equal to the TemporalID of the access unit containing the NAL unit.

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

NOTE 11 – When the NAL unit is a non-ACL NAL unit, the value of TemporalID is equal to the minimum value of the TemporalID values of all access units to which the non-ACL NAL unit applies. When nal\_unit\_type is equal to NAL\_AFPS, TemporalID can be greater than or equal to the TemporalID of the containing access unit, as all atlas frame parameter sets (AFPSs) can be included in the beginning of a bitstream, wherein the first coded atlas frame has TemporalID equal to 0. When nal\_unit\_type is equal to NAL\_PREFIX\_NSEI, NAL\_PREFIX\_ESEI, NAL\_SUFFIX\_NSEI, or NAL\_SUFFIX\_ESEI, TemporalID can be greater than or equal to the TemporalID of the containing access unit, as an SEI NAL unit can contain information, e.g., in a buffering period SEI message or an atlas frame timing SEI message, that applies to a bitstream subset that includes access units for which the TemporalID values are greater than the TemporalID of the access unit containing the SEI NAL unit.

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

##### General

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

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

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

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

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

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

When an AAPS RBSP (with a particular value of aaps\_atlas\_adaptation\_parameter\_set\_id) is not active for a particular atlas and is referred to by a coded tile NAL unit with nal\_layer\_id equal to 0 (using a value of ath\_atlas\_adaptation\_parameter\_set\_id equal to the aaps\_atlas\_adaptation\_parameter\_set\_id value), it is then activated. This AAPS RBSP is called the active AAPS RBSP until it is deactivated by the activation of another AAPS RBSP. Let the variable aapsAtlasID be either set equal to vuh\_atlas\_id of the AAPS RBSP, be set equal to -1 when the AAPS is part of a common atlas frame, or determined through external means if the V3C unit header is unavailable. Let the variable activatingAtlasID be either set equal to vuh\_atlas\_id of the coded tile NAL unit, be set equal to -1 when the NAL unit has nal\_unit\_type equal to NAL\_CAF, or determined through external means if the V3C unit header is unavailable. An AAPS RBSP, with that particular value of aaps\_atlas\_adaptation\_parameter\_set\_id, shall be available to the decoding process prior to its activation, included in at least one coded atlas access unit with TemporalID less than or equal to the TemporalID of the AAPS NAL unit or provided through external means, aapsAtlasID shall be equal to activatingAtlasID, and the AAPS NAL unit containing the AAPS RBSP shall have nal\_layer\_id equal to 0.

[Ed.(BK): Above references to common atlas data and NAL\_CAF can be removed again when MIV would avoid using the AAPS RBSP in common atlas data.]

Any AAPS NAL unit containing the value of aaps\_atlas\_adaptation\_parameter\_set\_id for the active AAPS RBSP for a coded atlas frame shall have the same content as that of the active AAPS RBSP for the coded atlas frame, unless it follows the last ACL NAL unit of the coded atlas frame and precedes the first ACL NAL unit of another coded atlas frame.

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

When an AFPS RBSP (with a particular value of afps\_atlas\_frame\_parameter\_set\_id) is not active for a particular atlas and is referred to by a coded tile NAL unit with nal\_layer\_id equal to 0 (using a value of ath\_atlas\_frame\_parameter\_set\_id equal to the afps\_atlas\_frame\_parameter\_set\_id value), it is then activated. This AFPS RBSP is called the active AFPS RBSP until it is deactivated by the activation of another AFPS RBSP. Let the variable afpsAtlasID be either set equal to vuh\_atlas\_id of the AFPS RBSP or determined through external means if the V3C unit header is unavailable. Let the variable activatingAtlasID be either set equal to vuh\_atlas\_id of the coded tile NAL unit or determined through external means if the V3C unit header is unavailable. An AFPS RBSP, with that particular value of afps\_atlas\_frame\_parameter\_set\_id, shall be available to the decoding process prior to its activation, included in at least one coded atlas access unit with TemporalID less than or equal to the TemporalID of the AFPS NAL unit or provided through external means, afpsAtlasID shall be equal to activatingAtlasID, and the AFPS NAL unit containing the AFPS RBSP shall have nal\_layer\_id equal to 0.

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

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

When an ASPS RBSP (with a particular value of asps\_atlas\_sequence\_parameter\_set\_id) is not already active for a particular atlas and it is referred to by activation of an AFPS RBSP (in which afps\_atlas\_sequence\_parameter\_set\_id is equal to the asps\_atlas\_sequence\_parameter\_set\_id value with afpsAtlasID set equal to vuh\_atlas\_id or determined through external means), it is activated for the particular atlas. This ASPS RBSP is called the active ASPS RBSP for the particular atlas until it is deactivated by the activation of another ASPS RBSP for the particular atlas. Let the variable aspsAtlasID be either set equal to vuh\_atlas\_id or determined through external means if the V3C unit header is unavailable. Let the variable activatingAtlasID either be set equal to vuh\_atlas\_id of the coded tile or determined through external means if the V3C unit header is unavailable. An ASPS RBSP, with that particular value of asps\_atlas\_sequence\_parameter\_set\_id, shall be available to the decoding process prior to its activation, included in at least one access unit with TemporalID equal to 0 or provided through external means, aspsAtlasID equal to activatingAtlasID, and the NAL unit containing the ASPS RBSP shall have nal\_layer\_id equal to 0. An activated ASPS RBSP shall remain active for the entire coded atlas sequence (CAS).

Any ASPS NAL unit with any aspsAtlasID value and nal\_layer\_id equal to 0 containing the value of asps\_atlas\_sequence\_parameter\_set\_id for the active ASPS RBSP for a CAS shall have the same content as that of the active ASPS RBSP for the CAS, unless it follows the last access unit of the CAS and precedes the first ACL NAL unit of another CAS.

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

During operation of the decoding process (see Clause 9), the values of parameters of the active AAPS RBSP, the active ASPS RBSP, and the active AFPS RBSP are considered in effect. For interpretation of SEI messages, the values of the active AAPS RBSP, the active ASPS RBSP, and the active AFPS RBSP for the operation of the decoding process for the ACL NAL units of the coded atlas frame with nal\_layer\_id equal to 0 in the same access unit are considered in effect unless otherwise specified in the SEI message semantics.

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

A bitstream conforming to this document consists of one or more CASs.

A CAS consists of one or more access units. The order of NAL units and coded atlas frames, and their association to access units are described in subclause 8.4.5.3.4.

The first access unit of a CAS is an IRAP coded atlas access unit with NoOutputBeforeRecoveryFlag equal to 1.

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

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

This subclause specifies the order of NAL units and coded atlas frames, and their association to access units for CASs that belong to a CVSs that conform to one or more of the profiles specified in Annex A and that are decoded using the decoding process specified in clauses 2 through 9.

An access unit consists of one coded atlas with nal\_layer\_id equal to 0, zero or more ACL NAL units with nal\_layer\_id greater than 0, and zero or more non-ACL NAL units. The association of ACL NAL units to coded atlases is described in subclause 8.4.5.3.5.

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

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

NOTE 1 – The last ACL NAL unit preceding firstBlAFrmNalUnit in decoding order can have nal\_layer\_id greater than 0.

* access unit delimiter or V3C access unit delimiter NAL unit with nal\_layer\_id equal to 0 (when present),
* AAPS NAL unit with nal\_layer\_id equal to 0 (when present),
* ASPS NAL unit with nal\_layer\_id equal to 0 (when present),
* AFPS NAL unit with nal\_layer\_id equal to 0 (when present),
* Prefix SEI NAL unit with nal\_layer\_id equal to 0 (when present),
* NAL units with nal\_unit\_type in the range of NAL\_RSV\_NACL\_48..NAL\_RSV\_NACL\_52 with nal\_layer\_id equal to 0 (when present),
* NAL units with nal\_unit\_type in the range of NAL\_UNSPEC\_53..NAL\_UNSPEC\_63 with nal\_layer\_id equal to 0 (when present).

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

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

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

* When an access unit delimiter or a V3C access unit delimiter NAL unit with nal\_layer\_id equal to 0 is present, it shall be the first NAL unit. There shall be at most one access unit delimiter or V3C access unit delimiter NAL unit with nal\_layer\_id equal to 0 in any access unit.
* When any ASPS NAL units, AAPS NAL units, AFPS NAL units, prefix SEI NAL units, NAL units with nal\_unit\_type in the range of NAL\_RSV\_NACL\_48..NAL\_RSV\_NACL\_50, or NAL units with nal\_unit\_type in the range of NAL\_UNSPEC\_53..NAL\_UNSPEC\_57 are present, they shall not follow the last ACL NAL unit of the access unit.
* NAL units having nal\_unit\_type equal to NAL\_FD, NAL\_SUFFIX\_NSEI, NAL\_SUFFIX\_ESEI or in the range of NAL\_RSV\_NACL\_51..NAL\_RSV\_NACL\_52 or NAL\_UNSPEC\_58..NAL\_UNSPEC\_63 shall not precede the first ACL NAL unit of the access unit.
* When an end of sequence NAL unit with nal\_layer\_id equal to 0 is present, it shall be the last NAL unit among all NAL units with nal\_layer\_id equal to 0 in the access unit other than an end of a bitstream NAL unit (when present).
* When an end of a bitstream NAL unit is present, it shall be the last NAL unit in the access unit.

NOTE 3 – Decoders conforming to profiles specified in Annex A do not use NAL units with nal\_layer\_id greater than 0, e.g., access or V3C access unit delimiter NAL units with nal\_layer\_id greater than 0, for access unit boundary detection, except for identification of a NAL unit as an ACL or non-ACL NAL unit.

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

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

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

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

* The first ACL NAL unit of the coded atlas frame shall have ath\_id equal to FirstTileID.
* The tiles of an atlas frame shall be in increasing order of their ath\_id values.

##### Order of access units (AUs) and association to CCASs

A CCAS consists of one or more non-ACL access units. The order of NAL units and coded common atlas frames, and their association to access units are described in subclause 8.4.5.3.7.

The first access unit of a CCAS is an IRAP coded common atlas access unit.

It is a requirement of bitstream conformance that, when present, the next access unit after an access unit that contains an end of sequence NAL unit or an end of a bitstream NAL unit shall be an IRAP coded common atlas access unit.

##### Order of NAL units in common atlas data and their association with access units

An access unit in a common atlas sub-bitstream consists of one non-ACL NAL unit with nal\_unit\_type NAL\_CAF with nal\_layer\_id equal to 0, and zero or more non-ACL NAL units with nal\_layer\_id equal or greater than 0.

The first access unit in a common atlas sub-bitstream starts with the first NAL unit of the sub-bitstream.

The order of non-ACL NAL units within an access unit in a common atlas bitstream shall obey the following constraints:

* There shall be at most one CAF NAL unit in any access unit in the common atlas sub-bitstream. If the CAF NAL unit is not the first NAL unit, then it shall be preceeded by an AAPS NAL unit.
* In a common atlas sub-bitstream, when any AAPS NAL units, prefix SEI NAL units, NAL units with nal\_unit\_type in the range of NAL\_RSV\_NACL\_49..NAL\_RSV\_NACL\_50, or NAL units with nal\_unit\_type in the range of NAL\_UNSPEC\_53..NAL\_UNSPEC\_57, nal\_unit\_type equal to NAL\_FDT, or in the range of NAL\_RSV\_NACL\_51..NAL\_RSV\_NACL\_52 or NAL\_UNSPEC\_58..NAL\_UNSPEC\_63 are present, they shall precede the first CAF NAL unit of the access unit.
* When any NAL units with nal\_unit\_type equal to NAL\_SUFFIX\_NSEI or NAL\_SUFFIX\_ESEI are are present in a common atlas sub-bitstream, they shall follow the last CAF NAL unit of the access unit.
* When an end of sequence NAL unit with nal\_layer\_id equal to 0 is present in a common atlas sub-bitstream, it shall be the last NAL unit among all NAL units with nal\_layer\_id equal to 0 in the access unit other than an end of an bitstream NAL unit (when present).
* When an end of an bitstream NAL unit is present in a common atlas sub-bitstream, it shall be the last NAL unit in the access unit.

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

#### Atlas sequence parameter set RBSP semantics

##### General atlas sequence parameter set RBSP semantics

**asps\_atlas\_sequence\_parameter\_set\_id** provides an identifier for the atlas sequence parameter set for reference by other syntax elements.

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

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

The variable AspsFrameSize is set equal to asps\_frame\_height \* asps\_frame\_width.

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

**asps\_geometry\_3d\_bit\_depth\_minus1** plus 1 indicates the bit depth of the geometry coordinates of the reconstructed volumetric content. asps\_geometry\_3d\_bit\_depth\_minus1 shall be in the range of 0 to 31, inclusive.

**asps\_geometry\_2d\_bit\_depth\_minus1** plus 1 indicates the bit depth of the geometry when projected onto 2D images. asps\_geometry\_2d\_bit\_depth\_minus1 shall be in the range of 0 to 31, inclusive.

NOTE 2 – For some applications, such as the application defined in Annex H, the syntax elements asps\_geometry\_3d\_bit\_depth\_minus1 and asps\_geometry\_2d\_bit\_depth\_minus1 may differ from their corresponding elements in the VPS, gi\_geometry\_3d\_coordinates\_bit\_depth\_minus1 and gi\_geometry\_2d\_bit\_depth\_minus1, respectively. This can be due to the fact that the elements asps\_geometry\_3d\_bit\_depth\_minus1 and asps\_geometry\_2d\_bit\_depth\_minus1 may potentially represent an intermediate space that may require different precision for coding purposes than the target representation space, which the elements gi\_geometry\_3d\_coordinates\_bit\_depth\_minus1 and gi\_geometry\_2d\_bit\_depth\_minus1 indicate.

**asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4** plus 4 specifies the values of the variables Log2MaxAtlasFrmOrderCntLsb and MaxAtlasFrmOrderCntLsb that are used in the decoding process for the atlas frame order count as follows:

Log2MaxAtlasFrmOrderCntLsb = asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4 + 4 (17)

MaxAtlasFrmOrderCntLsb = 2Log2MaxAtlasFrmOrderCntLsb (18)

The value of asps\_log2\_max\_atlas\_frame\_order\_cnt\_lsb\_minus4 shall be in the range of 0 to 12, inclusive.

**asps\_max\_dec\_atlas\_frame\_buffering\_minus1** plus 1 specifies the maximum required size of the decoded atlas frame buffer for the CAS in units of atlas frame storage buffers. The value of asps\_max\_dec\_atlas\_frame\_buffering\_minus1 shall be in the range of 0 to 15, inclusive.

**asps\_long\_term\_ref\_atlas\_frames\_flag** equal to 0 specifies that no long-term reference atlas frame is used for inter prediction of any coded atlas frame in the CAS. asps\_long\_term\_ref\_atlas\_frames\_flagequal to 1 specifies that long term reference atlas frames may be used for inter prediction of one or more coded atlas frames in the CAS.

**asps\_num\_ref\_atlas\_frame\_lists\_in\_asps** specifies the number of the ref\_list\_struct( rlsIdx ) syntax structures included in the atlas sequence parameter set. The value of asps\_num\_ref\_atlas\_frame\_lists\_in\_asps shall be in the range of 0 to 64, inclusive.

NOTE 3 – A decoder allocates memory for a total number of ref\_list\_struct( rlsIdx ) syntax structures equal to (asps\_num\_ref\_atlas\_frame\_lists\_in\_asps + 1) since there can be one ref\_list\_struct( rlsIdx ) syntax structure directly signalled in the atlas tile headers of the current atlas tile.

**asps\_use\_eight\_orientations\_flag** equal to 0 specifies that the patch orientation index for a patch with index j in a tile with tile ID equal to i, pdu\_orientation\_index[ i ][ j ], is in the range of 0 to 1, inclusive. asps\_use\_eight\_orientations\_flagequal to 1 specifies that pdu\_orientation\_index[ i ][ j ] is in the range of 0 to 7, inclusive.

**asps\_extended\_projection\_enabled\_flag** equal to 0 specifies that the patch projection information is not signalled for the current atlas tile. asps\_extended\_projection\_enabled\_flag equal to 1 specifies that the patch projection information is signalled for the current atlas tile. When asps\_extended\_projection\_enabled\_flag is not present, its value is inferred to be equal to 0.

**asps\_max\_number\_projections\_minus1** plus 1 specifies the maximum value that can be indicated for the patch projection ID syntax element, pdu\_projection\_id[ i ][ j ], for a patch with index j in a tile with tile ID equal to i. When asps\_max\_number\_projections\_minus1 is not present, its value is inferred to be equal to 5.

**asps\_normal\_axis\_limits\_quantization\_enabled\_flag** equal to 1 specifies that quantization parameters shall be signalled and used for quantizing the normal axis related elements of a patch data unit, a merge patch data unit, or an inter patch data unit. If asps\_normal\_axis\_limits\_quantization\_enabled\_flag is equal to 0, then no quantization is applied on any normal axis related elements of a patch data unit, a merge patch data unit, or an inter patch data unit.

**asps\_normal\_axis\_max\_delta\_value\_enabled\_flag** equal to 1 specifies that the maximum nominal delta value of the normal axis that may be present in the geometry information of a patch with index i in a frame with index j will be indicated in the bitstream for each patch data unit, a merge patch data unit, or an inter patch data unit. If asps\_normal\_axis\_max\_delta\_value\_enabled\_flagis equal to 0 then the maximum nominal delta value of the normal axis that may be present in the geometry information of a patch with index i in a frame with index j shall not be indicated in the bitstream for each patch data unit, a merge patch data unit, or an inter patch data unit.

**asps\_patch\_precedence\_order\_flag** specifies the patch precedence that is utilized for assigning atlas samples to patches as specified in subclause 9.2.6. asps\_patch\_precedence\_order\_flag equal to 1 specifies that patch precedence for the current atlas is the same as the decoding order of the patches. asps\_patch\_precedence\_order\_flag equal to 0 specifies that patch precedence for the current atlas is the reverse of the decoding order of the patches.

**asps\_log2\_patch\_packing\_block\_size** specifies the value of the variable PatchPackingBlockSize, that is used for the horizontal and vertical placement of the patches within the atlas, as follows:

PatchPackingBlockSize = 2asps\_log2\_patch\_packing\_block\_size (19)

The value of asps\_log2\_patch\_packing\_block\_size shall be in the range of 0 to 7, inclusive.

**asps\_patch\_size\_quantizer\_present\_flag** equal to 1 indicates that the patch size quantization parameters are present in an atlas tile header. If asps\_patch\_size\_quantizer\_present\_flag is equal to 0, then the patch size quantization parameters are not present.

**asps\_map\_count\_minus1** plus 1 indicates the number of maps that may be used for encoding the geometry and attribute data for the current atlas. asps\_map\_count\_minus1 shall be in the range of 0 to 15, inclusive. It is a requirement of bitstream conformance to this document that asps\_map\_count\_minus1 is equal to vps\_map\_count\_minus1[ atlasID ], where atlasID is the atlas ID of the current atlas.

**asps\_pixel\_deinterleaving\_enabled\_flag** equal to 1 indicates that the decoded geometry and attribute data may require an additional spatial interpolation process during reconstruction. asps\_pixel\_deinterleaving\_enabled\_flag equal to 0 indicates that the decoded geometry and attribute data do not require any additional spatial interpolation process.

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

NOTE 4 – In the context of Reconstruction profiles for V-PCC, an example usage is specified in subclause H.11.2.3.

**asps\_raw\_patch\_enabled\_flag** equal to 1 indicates that the decoded geometry and attribute videos for the current atlas contain information related to RAW coded points. asps\_raw\_patch\_enabled\_flag equal to 0 indicates that the decoded geometry and attribute videos do not contain information related to RAW coded points.

**asps\_eom\_patch\_enabled\_flag** equal to 1 indicates that the decoded occupancy video for the current atlas contains information related to whether intermediate depth positions between two depth maps are occupied. asps\_eom\_patch\_enabled\_flag equal to 0 indicates that the decoded occupancy video does not contain information related to whether intermediate depth positions between two depth maps are occupied.

It is a requirement of bitstream conformance that if asps\_eom\_patch\_enabled\_flag is equal to 1, oi\_lossy\_occupancy\_compression\_threshold[ atlasID ] shall be equal to 0.

**asps\_eom\_fix\_bit\_count\_minus1** plus 1 indicates the size in bits of the EOM codeword. asps\_eom\_fix\_bit\_count\_minus1 shall be in the range of 0 to Min(15, oi\_occupancy\_2d\_bit\_depth\_minus1[ atlasID ] − 1), inclusive, where atlasID is the atlas ID of the current atlas.

**asps\_auxiliary\_video\_enabled\_flag** equal to 1 indicates that information associated with RAW and EOM patch types could be placed in auxiliary video sub-bitstreams. asps\_auxiliary\_video\_enabled\_flag equal to 0 indicates that information associated with RAW and EOM patch types can only be placed in non‑auxiliary video sub-bitstreams.

It is a requirement of bitstream conformance that if vps\_auxiliary\_video\_present\_flag[ atlasID ] , where atlasID is the atlas ID of the current atlas, is equal to 0 then asps\_auxiliary\_video\_enabled\_flag is equal to 0.

**asps\_plr\_enabled\_flag** equal to 1 indicates that point local reconstruction mode information may be present in the bitstream for the current atlas. asps\_plr\_enabled\_flag equal to 0 indicates that no information related to the point local reconstruction mode is present in the bitstream for the current atlas.

It is a requirement of bitstream conformance that when asps\_pixel\_deinterleaving\_enabled\_flag is equal to 1, asps\_plr\_enabled\_flag shall be equal to 0.

**asps\_vui\_parameters\_present\_flag** equal to 1 specifies that the vui\_parameters( ) syntax structure, as specified in Annex G, is present. asps\_vui\_parameters\_present\_flag equal to 0 specifies that the vui\_parameters( ) syntax structure, as specified in Annex G, is not present.

**asps\_extension\_present\_flag** equal to 1 specifies that the syntax elements asps\_vpcc\_extension\_present\_flag and asps\_extension\_6bits are present in the atlas\_sequence\_parameter\_set\_rbsp syntax structure. asps\_extension\_present\_flag equal to 0 specifies that the syntax elements asps\_vpcc\_extension\_present\_flag and asps\_extension\_6bits are not present.

**asps\_vpcc\_extension\_present\_flag** equal to 1 specifies that the asps\_vpcc\_extension( ) syntax structure is present in the atlas\_sequence\_parameter\_set\_rbsp syntax structure. asps\_vpcc\_extension\_present\_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of asps\_vpcc\_extension\_present\_flag is inferred to be equal to 0.

**asps\_miv\_extension\_present\_flag** equal to 1 specifies that the asps\_miv\_extension( ) syntax structure is present in the atlas\_sequence\_parameter\_set\_rbsp syntax structure. asps\_miv\_extension\_present\_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of asps\_miv\_extension\_present\_flag is inferred to be equal to 0.

**asps\_extension\_6bits** equal to 0 specifies that no asps\_extension\_data\_flag syntax elements are present in the ASPS RBSP syntax structure. When present, asps\_extension\_6bits shall be equal to 0 in bitstreams conforming to this version of this document. Values of asps\_extension\_6bits not equal to 0 are reserved for future use by ISO/IEC. Decoders shall allow the value of asps\_extension\_6bits to be not equal to 0 and shall ignore all asps\_extension\_data\_flag syntax elements in an ASPS NAL unit. When not present, the value of asps\_extension\_6bits is inferred to be equal to 0.

**asps\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this document. Decoders conforming to this version of this document shall ignore all asps\_extension\_data\_flag syntax elements.

##### Point local reconstruction information semantics

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

**plri\_number\_of\_modes\_minus1**[ i ] plus 1 indicates the number of reconstruction modes specified for the point local reconstruction process associated with the map of index i of the current atlas. The value of plri\_number\_of\_modes\_minus1[ i ] shall be in the range of 0 to 15, inclusive.

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

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

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

**plri\_neighbour\_minus1**[ i ][ j ] plus 1 specifies the size of the 2D neighbourhood for point local reconstruction mode j for the map of index i of the current atlas. plri\_neighbour\_minus1[ i ][ j ] shall be in the range of 0 to 3, inclusive.

**plri\_block\_threshold\_per\_patch\_minus1**[ i ] plus 1 specifies the value representative of the threshold used to define the values contained in the array plrd\_level. plri\_block\_threshold\_per\_patch\_minus1[ i ] shall be in the range of 0 to 63, inclusive.

#### Atlas frame parameter set RBSP semantics

##### General atlas frame parameter set RBSP semantics

**afps\_atlas\_frame\_parameter\_set\_id** identifies the atlas frame parameter set for reference by other syntax elements.

**afps\_atlas\_sequence\_parameter\_set\_id** specifies the value of asps\_atlas\_sequence\_parameter\_set\_id for the active atlas sequence parameter set.

**afps\_output\_flag\_present\_flag** equal to 1 indicates that the ath\_atlas\_output\_flag syntax element is present in the associated tile headers. afps\_output\_flag\_present\_flag equal to 0 indicates that the ath\_atlas\_output\_flag syntax element is not present in the associated tile headers.

**afps\_num\_ref\_idx\_default\_active\_minus1** plus 1 specifies the inferred value of the variable NumRefIdxActive for the tile with ath\_num\_ref\_idx\_active\_override\_flag equal to 0. The value of afps\_num\_ref\_idx\_default\_active\_minus1 shall be in the range of 0 to 14, inclusive.

**afps\_additional\_lt\_afoc\_lsb\_len** specifies the value of the variable MaxLtAtlasFrmOrderCntLsb that is used in the decoding process for reference atlas frame lists as follows:

MaxLtAtlasFrmOrderCntLsb =  
 2 \* ( Log2MaxAtlasFrmOrderCntLsb + afps\_additional\_lt\_afoc\_lsb\_len) (20)

The value of afps\_additional\_lt\_afoc\_lsb\_len shall be in the range of 0 to 32 – Log2MaxAtlasFrmOrderCntLsb, inclusive.

When asps\_long\_term\_ref\_atlas\_frames\_flag is equal to 0, the value of afps\_additional\_lt\_afoc\_lsb\_len shall be equal to 0.

**afps\_lod\_mode\_enabled\_flag** equal to 1 indicates that the LOD parameters may be present in a patch. afps\_lod\_mode\_enabled\_flag equal to 0 indicates that the LOD parameters shall not be present in a patch.

**afps\_raw\_3d\_offset\_bit\_count\_explicit\_mode\_flag** equal to 1 indicates that the number of bits in the fixed-length representation of rpdu\_3d\_offset\_u[ tileID ][ p ], rpdu\_3d\_offset\_v[ tileID ][ p ], and rpdu\_3d\_offset\_d[ tileID ][ p ] for a patch with index p in a tile with tile ID equal to tileID, is explicitly coded by ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1 in the atlas tile header that refers to afps\_atlas\_frame\_parameter\_set\_id. afps\_raw\_3d\_offset\_bit\_count\_explicit\_mode\_flag equal to 0 indicates that the value of ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1 is implicitly derived.

**afps\_extension\_present\_flag** equal to 1 specifies that the syntax element afps\_extension\_8bits is present in the atlas\_frame\_parameter\_set\_rbsp syntax structure. afps\_extension\_present\_flag equal to 0 specifies that the syntax element afps\_extension\_8bits is not present. The value of afps\_extension\_present\_flag shall be 0 in this version of this document

**afps\_extension\_8bits** equal to 0 specifies that no afps\_extension\_data\_flag syntax elements are present in the AFPS RBSP syntax structure. When present, afps\_extension\_8bits shall be equal to 0 in bitstreams conforming to this version of this document. Values of afps\_extension\_8bits not equal to 0 are reserved for future use by ISO/IEC. Decoders shall allow the value of afps\_extension\_8bits to be not equal to 0 and shall ignore all afps\_extension\_data\_flag syntax elements in an AFPS NAL unit. When not present, the value of afps\_extension\_8bits is inferred to be equal to 0.

**afps\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this document. Decoders conforming to this version of this document shall ignore all afps\_extension\_data\_flag syntax elements.

##### Atlas frame tile information syntax

**afti\_single\_tile\_in\_atlas\_frame\_flag** equal to 1 specifies that there is only one tile in each atlas frame referring to the AFPS. afti\_single\_tile\_in\_atlas\_frame\_flag equal to 0 specifies that there may be more than one tile in each atlas frame referring to the AFPS.

**afti\_uniform\_partition\_spacing\_flag** equal to 1 specifies that the tile partitioning of an atlas uses a method that distributes column and row partition boundaries uniformly across the atlas frame. The information corresponding to these boundaries is signalled using the syntax elements afti\_partition\_cols\_width\_minus1 and afti\_partition\_rows\_height\_minus1, respectively. afti\_uniform\_partition\_spacing\_flag equal to 0 specifies that the tile partitioning of an atlas uses a method that may result in column and row partition boundaries that may or may not be distributed uniformly across the atlas frame. In this case, these boundaries are signalled using the syntax elements afti\_num\_partition\_columns\_minus1 and afti\_num\_partition\_rows\_minus1 and a list of syntax element pairs afti\_partition\_column\_width\_minus1[ i ] and afti\_partition\_row\_height\_minus1[ i ]. When not present, the value of afti\_uniform\_partition\_spacing\_flag is inferred to be equal to 1.

**afti\_partition\_cols\_width\_minus1** plus 1 specifies the width of the tile partition columns excluding the right-most tile partition column of the atlas frame in units of 64 samples when afti\_uniform\_partition\_spacing\_flag is equal to 1. The value of afti\_partition\_cols\_width\_minus1 shall be in the range of 0 to asps\_frame\_width / 64 − 1, inclusive. When not present, the value of afti\_partition\_cols\_width\_minus1 is inferred to be equal to asps\_frame\_width / 64 – 1**.**

**afti\_partition\_rows\_height\_minus1** plus 1 specifies the height of the tile partition rows excluding the bottom tile partition row of the atlas frame in units of 64 samples when afti\_uniform\_partition\_spacing\_flag is equal to 1. The value of afti\_partition\_rows\_height\_minus1 shall be in the range of 0 to asps\_frame\_height / 64 − 1, inclusive. When not present, the value of afti\_partition\_rows\_height\_minus1 is inferred to be equal to asps\_frame\_height / 64 – 1.

**afti\_num\_partition\_columns\_minus1** plus 1 specifies the number of tile partition columns used to partition the atlas frame when afti\_uniform\_partition\_spacing\_flag is equal to 0. The value of afti\_num\_partition\_columns\_minus1 shall be in the range of 0 to asps\_frame\_width / 64 − 1, inclusive. If afti\_single\_tile\_in\_atlas\_frame\_flag is equal to 1, the value of afti\_num\_partition\_columns\_minus1 is inferred to be equal to 0. Otherwise, when afti\_uniform\_partition\_spacing\_flag is equal to 1, the value of afti\_num\_partition\_columns\_minus1 is inferred as specified in subclause 7.5.

**afti\_num\_partition\_rows\_minus1** plus 1 specifies the number of tile partition rows used to partition the atlas frame when afti\_uniform\_partition\_spacing\_flag is equal to 0. The value of afti\_num\_partition\_rows\_minus1 shall be in the range of 0 to asps\_frame\_height / 64 − 1, inclusive. If afti\_single\_tile\_in\_atlas\_frame\_flag is equal to 1, the value of afti\_num\_partition\_rows\_minus1 is inferred to be equal to 0. Otherwise, when afti\_uniform\_partition\_spacing\_flag is equal to 1, the value of afti\_num\_partition\_rows\_minus1 is inferred as specified in subclause 7.5.

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

When afti\_single\_tile\_in\_atlas\_frame\_flag is equal to 0, NumPartitionsInAtlasFrame shall be greater than 1.

**afti\_partition\_column\_width\_minus1**[ i ] plus 1 specifies the width of the i-th tile partition column in units of 64 samples.

**afti\_partition\_row\_height\_minus1**[ i ] plus 1 specifies the height of the i-th tile partition row in units of 64 samples.

**afti\_single\_partition\_per\_tile\_flag** equal to 1 specifies that each tile that refers to this AFPS includes one tile partition. afti\_single\_partition\_per\_tile\_flag equal to 0 specifies that a tile that refers to this AFPS may include more than one tile partition. When not present, the value of afti\_single\_partition\_per\_tile\_flag is inferred to be equal to 1.

**afti\_num\_tiles\_in\_atlas\_frame\_minus1** plus 1 specifies the number of tiles in each atlas frame referring to the AFPS. The value of afti\_num\_tiles\_in\_atlas\_frame\_minus1shall be in the range of 0 to NumPartitionsInAtlasFrame − 1, inclusive. When not present and afti\_single\_partition\_per\_tile\_flag is equal to 1, the value of afti\_num\_tiles\_in\_atlas\_frame\_minus1is inferred to be equal to NumPartitionsInAtlasFrame − 1.

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

**afti\_bottom\_right\_partition\_column\_offset**[ i ] specifies the offset between the column position of the tile partition located at the bottom-right corner of the i-th tile and the column position of the tile partition with partition index equal to afti\_top\_left\_partition\_idx[ i ]. When afti\_single\_partition\_per\_tile\_flag is equal to 1, the value of afti\_bottom\_right\_partition\_column\_offset[ i ] is inferred to be equal to 0.

**afti\_bottom\_right\_partition\_row\_offset**[ i ] specifies the offset between the row position of the tile partition located at the bottom-right corner of the i-th tile and the row position of the tile partition with partition index equal to afti\_top\_left\_partition\_idx[ i ]. When afti\_single\_partition\_per\_tile\_flag is equal to 1, the value of afti\_bottom\_right\_partition\_row\_offset[ i ] is inferred to be equal to 0.

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

topLeftColumn[ i ] = afti\_top\_left\_partition\_idx[ i ] % NumPartitionColumns  
 topLeftRow[ i ] = afti\_top\_left\_partition\_idx[ i ] / NumPartitionColumns  
 bottomRightColumn[ i ] = topLeftColumn[ i ] + afti\_bottom\_right\_partition\_column\_offset[ i ]  
 bottomRightRow[ i ] = topLeftRow[ i ] + afti\_bottom\_right\_partition\_row\_offset[ i ]

It is a requirement of bitstream conformance that the values of bottomRightColumn[ i ] and bottomRightRow[ i ] shall be smaller or equal to ( asps\_frame\_width + 63 ) / 64 – 1 and ( asps\_frame\_height + 63 ) / 64 – 1, respectively.

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

* topLeftColumn[ i ] <= topLeftColumn[ j ] <= bottomRightColumn[ i ]
* topLeftRow[ i ] <= topLeftRow[ j ] <= bottomRightRow[ i ]

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

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

**afti\_auxiliary\_video\_tile\_row\_width\_minus1** plus 1 indicates the nominal width of all auxiliary video sub-bitstreams in units of 64 integer samples. When afti\_auxiliary\_video\_tile\_row\_width\_minus1 is not present, its value shall be inferred to be equal to –1.

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

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

AuxTileHeight[ i ] = afti\_auxiliary\_video\_tile\_row\_height[ i ] \* 64

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

AuxTileOffset[ 0 ] = 0

AuxTileOffset[ i ] = AuxTileOffset[ i – 1 ] + AuxTileHeight[ i – 1 ], for all i > 0.

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

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

where N is equal to afti\_num\_tiles\_in\_atlas\_frame\_minus1

**afti\_signalled\_tile\_id\_flag** equal to 1 specifies that the tile ID for each tile is signalled. afti\_signalled\_tile\_id\_flag equal to 0 specifies that tile IDs are not signalled.

**afti\_signalled\_tile\_id\_length\_minus1** plus 1 specifies the number of bits used to represent the syntax element afti\_tile\_id[ i ] when present, and the syntax element ath\_id in a tile header. The value of afti\_signalled\_tile\_id\_length\_minus1 shall be in the range of 0 to 15, inclusive. When not present, the value of afti\_signalled\_tile\_id\_length\_minus1 is inferred to be equal to Ceil( Log2( afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1 ) ) − 1.

**afti\_tile\_id**[ i ] specifies the tile ID of the i-th tile. The length of the afti\_tile\_id[ i ] syntax element is afti\_signalled\_tile\_id\_length\_minus1 + 1 bits. When not present, the value of afti\_tile\_id[ i ] is inferred to be equal to i, for each i in the range of 0 to afti\_num\_tiles\_in\_atlas\_frame\_minus1, inclusive. It is a requirement of bitstream conformance that afti\_tile\_id[ i ] shall not be equal to afti\_tile\_id[ j ] for all i != j. The length of the afti\_tile\_id[ i ] syntax element is afti\_signalled\_tile\_id\_length\_minus1 + 1 bits.

The variable FirstTileID is computed as follows:

FirstTileID = afti\_tile\_id[ 0 ]  
 for ( i = 1; i < afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1; i++ )  
 FirstTileID = Min(FirstTileID, afti\_tile\_id[ i ])

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

#### Atlas adaptation parameter set RBSP semantics

**aaps\_atlas\_adaptation\_parameter\_set\_id** identifies the atlas adaptation parameter set for reference by other syntax elements.

**aaps\_extension\_present\_flag** equal to 1 specifies that the syntax elements aaps\_vpcc\_extension\_present\_flag, aaps\_miv\_extension\_present\_flag and aaps\_extension\_6bits are present in the atlas\_adaptation\_parameter\_set\_rbsp( ) syntax structure. aaps\_extension\_present\_flag equal to 0 specifies that the syntax elements aaps\_vpcc\_extension\_present\_flag and aaps\_extension\_6bits are not present.

**aaps\_vpcc\_extension\_present\_flag** equal to 1 specifies that the aaps\_vpcc\_extension( ) syntax structure is present in the atlas\_adaptation\_parameter\_set\_rbsp( ) syntax structure. aaps\_vpcc\_extension\_present\_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of aaps\_vpcc\_extension\_present\_flag is inferred to be equal to 0.

**aaps\_miv\_extension\_present\_flag** equal to 1 specifies that the aaps\_miv\_extension( ) syntax structure is present in the atlas\_adaptation\_parameter\_set\_rbsp( ) syntax structure. aaps\_vpcc\_extension\_present\_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of aaps\_miv\_extension\_present\_flag is inferred to be equal to 0.

**aaps\_extension\_6bits** equal to 0 specifies that no aaps\_extension\_data\_flag syntax elements are present in the AAPS RBSP syntax structure. When present, aaps\_extension\_6bits shall be equal to 0 in bitstreams conforming to this version of this document. Values of aaps\_extension\_6bits not equal to 0 are reserved for future use by ISO/IEC. Decoders shall allow the value of aaps\_extension\_6bits to be not equal to 0 and shall ignore all aaps\_extension\_data\_flag syntax elements in an AAPS NAL unit. When not present, the value of aaps\_extension\_6bits is inferred to be equal to 0.

**aaps\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this document. Decoders conforming to this version of this document shall ignore all aaps\_extension\_data\_flag syntax elements.

#### Supplemental enhancement information RBSP semantics

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

#### Access unit delimiter RBSP semantics

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

**aframe\_type** indicates that the ath\_type values for all tiles of the coded atlas frame in the access unit containing the access unit delimiter or V3C access unit delimiter NAL unit are members of the set listed in Table 5 for the given value of aframe\_type. The value of aframe\_type shall be equal to 0, 1, 2, or 3 in bitstreams conforming to this version of this document. Other values of aframe\_type are reserved for future use by ISO/IEC. Decoders conforming to this version of this document shall ignore reserved values of aframe\_type.

Table 5 – Interpretation of aframe\_type

|  |  |
| --- | --- |
| **aframe\_type** | **ath\_type values that may be present in the coded atlas frame** |
| 0 | I\_TILE |
| 1 | P\_TILE, I\_TILE |
| 2 | SKIP\_TILE, P\_TILE, I\_TILE |
| 3 | SKIP\_TILE |
| 4 – 7 | RESERVED |

#### End of sequence RBSP semantics

When included in a NAL unit with nal\_layer\_id equal to 0, the end of sequence RBSP specifies that the current access unit is the last access unit in the coded atlas sequence in decoding order and the next subsequent access unit in the atlas sub-bitstream in decoding order (if any) is an IRAP coded atlas access unit with NoOutputBeforeRecoveryFlag equal to 1. The syntax content of the SODB and RBSP for the end of sequence RBSP are empty.

#### End of bitstream RBSP semantics

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

#### Filler data RBSP semantics

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

**ff\_byte** is a byte equal to 0xFF.

#### Atlas tile layer RBSP semantics

None.

#### RBSP trailing bit semantics

**rbsp\_stop\_one\_bit** shall be equal to 1.

**rbsp\_alignment\_zero\_bit** shall be equal to 0.

#### Atlas tile header semantics

When present, the value of the atlas tile header syntax elements ath\_atlas\_frame\_parameter\_set\_id, ath\_atlas\_adaptation\_parameter\_set\_id, ath\_atlas\_output\_flag, ath\_no\_output\_of\_prior\_atlas\_frames\_flag, and ath\_atlas\_frm\_order\_cnt\_lsb, shall be the same in all atlas tile headers of a coded atlas frame.

**ath\_no\_output\_of\_prior\_atlas\_frames\_flag** affects the output of previously-decoded atlas frames in the DAB after the decoding of an atlas in a CAS AU that is not the first AU in the bitstream as specified in Annex E. When ath\_no\_output\_of\_prior\_atlas\_frames\_flag is not present, its value is inferred to be equal to 0.

It is a requirement of bitstream conformance that the value of ath\_no\_output\_of\_prior\_atlas\_frames\_flag shall be the same for all atlas frames in an AU.

The value of ath\_no\_output\_of\_prior\_atlas\_frames\_flag in the atlas tile headers is also referred to as the output\_of\_prior\_atlas\_frames\_flag value of the AU.

**ath\_atlas\_frame\_parameter\_set\_id** specifies the value of afps\_atlas\_frame\_parameter\_set\_id for the active atlas frame parameter set for the current atlas tile.

**ath\_atlas\_adaptation\_parameter\_set\_id** specifies the value of aaps\_atlas\_adaptation\_parameter\_set\_id for the active atlas adaptation parameter set for the current atlas tile. It is a requirement of bitstream conformance that all tiles of an atlas frame shall have the same ath\_atlas\_adaptation\_parameter\_set\_id.

**ath\_id** specifies the tile ID associated with the current tile. When not present, the value of ath\_idis inferred to be equal to 0.

The following applies:

* The length of ath\_id is afti\_signalled\_tile\_id\_length\_minus1 + 1 bits.
* The value of ath\_id shall be in the range of values specified by the array TileIndexToID[ i ], for i in the range from 0 to afti\_num\_tiles\_in\_atlas\_frame\_minus1, inclusive.

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

* The value of ath\_id shall not be equal to the value of ath\_id of any other coded atlas tile unit of the same coded atlas frame.
* The tiles of an atlas frame shall be in increasing order of their ath\_id values.

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

Table 6 – Name association to ath\_type

|  |  |
| --- | --- |
| ath\_type | Name of ath\_type |
| 0 | P\_TILE (Inter atlas tile) |
| 1 | I\_TILE (Intra atlas tile) |
| 2 | SKIP\_TILE (SKIP atlas tile) |
| 3 - … | RESERVED |

**ath\_atlas\_output\_flag** affects the decoded atlas output and removal processes as specified in Annex E. When ath\_atlas\_output\_flag is not present, it is inferred to be equal to 1.

**ath\_atlas\_frm\_order\_cnt\_lsb** specifies the atlas frame order count modulo MaxAtlasFrmOrderCntLsb for the current atlas tile. The length of the ath\_atlas\_frm\_order\_cnt\_lsb syntax element is equal to Log2MaxAtlasFrmOrderCntLsb bits. The value of the ath\_atlas\_frm\_order\_cnt\_lsb shall be in the range of 0 to MaxAtlasFrmOrderCntLsb − 1, inclusive.

**ath\_ref\_atlas\_frame\_list\_asps\_flag** equal to 1 specifies that the reference atlas frame list of the current atlas tile is derived based on one of the ref\_list\_struct( rlsIdx ) syntax structures in the active ASPS. ath\_ref\_atlas\_frame\_list\_asps\_flag equal to 0 specifies that the reference atlas frame list of the current atlas tile list is derived based on the ref\_list\_struct( rlsIdx ) syntax structure that is directly included in the tile header of the current atlas tile. When asps\_num\_ref\_atlas\_frame\_lists\_in\_aspsis equal to 0, the value of ath\_ref\_atlas\_frame\_list\_asps\_flag is inferred to be equal to 0.

**ath\_ref\_atlas\_frame\_list\_idx** specifies the index, into the list of the ref\_list\_struct( rlsIdx ) syntax structures included in the active ASPS, of the ref\_list\_struct( rlsIdx ) syntax structure that is used for derivation of the reference atlas frame list for the current atlas tile. The syntax element ath\_ref\_atlas\_frame\_list\_idx is represented by Ceil( Log2( asps\_num\_ref\_atlas\_frame\_lists\_in\_asps ) ) bits. When not present, the value of ath\_ref\_atlas\_frame\_list\_idx is inferred to be equal to 0. The value of ath\_ref\_atlas\_frame\_list\_idx shall be in the range of 0 to asps\_num\_ref\_atlas\_frame\_lists\_in\_asps − 1, inclusive. When ath\_ref\_atlas\_frame\_list\_asps\_flag is equal to 1 and asps\_num\_ref\_atlas\_frame\_lists\_in\_asps is equal to 1, the value of ath\_ref\_atlas\_frame\_list\_idx is inferred to be equal to 0.

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

RlsIdx = ath\_ref\_atlas\_frame\_list\_asps\_flag ?  
 ath\_ref\_atlas\_frame\_list\_idx : asps\_num\_ref\_atlas\_frame\_lists\_in\_asps (21)

**ath\_additional\_afoc\_lsb\_present\_flag**[ j ] equal to 1 specifies that ath\_additional\_afoc\_lsb\_val[ j ] is present for the current atlas tile. ath\_additional\_afoc\_lsb\_present\_flag[ j ] equal to 0 specifies that ath\_additional\_afoc\_lsb\_val[ j ] is not present.

**ath\_additional\_afoc\_lsb\_val**[ j ] specifies the value of FullAtlasFrmOrderCntLsbLt[ RlsIdx ][ j ] for the current atlas tile as follows:

FullAtlasFrmOrderCntLsbLt[ RlsIdx ][ j ] =  
 ath\_additional\_afoc\_lsb\_val[ j ] \* MaxAtlasFrmOrderCntLsb +afoc\_lsb\_lt[ RlsIdx ][ j ] (22)

The syntax element ath\_additional\_afoc\_lsb\_val[ j ] is represented by afps\_additional\_lt\_afoc\_lsb\_len bits. When not present, the value of ath\_additional\_afoc\_lsb\_val[ j ] is inferred to be equal to 0.

**ath\_pos\_min\_d\_quantizer** specifies the quantizer that is to be applied to the pdu\_3d\_offset\_d[ tileID ][ p ] value of the patch with index p in the tile with tile ID equal to tileID. If ath\_pos\_min\_d\_quantizer is not present, its value shall be inferred to be equal to 0.

**ath\_pos\_delta\_max\_d\_quantizer** specifies the quantizer that is to be applied to the pdu\_3d\_range\_d[ tileID ][ p ] value of the patch with index p in the tile with tile ID equal to tileID. If ath\_pos\_delta\_max\_d\_quantizer is not present, its value shall be inferred to be equal to 0.

**ath\_patch\_size\_x\_info\_quantizer** specifies the value of the quantizer PatchSizeXQuantizer that is to be applied to the syntax elements pdu\_2d\_size\_x\_minus1[ tileID ][ p ], mpdu\_2d\_delta\_size\_x[ tileID ][ p ], ipdu\_2d\_delta\_size\_x[ tileID ][ p ], rpdu\_2d\_size\_x\_minus1[ tileID ][ p ], and epdu\_2d\_size\_x\_minus1[ tileID ][ p ] of a patch with index p in a tile with tile ID equal to tileID. If ath\_patch\_size\_x\_info\_quantizer is not present, its value shall be inferred to be equal to asps\_log2\_patch\_packing\_block\_size. The value of PatchSizeXQuantizer is computed as follows:

PatchSizeXQuantizer = 1 << ath\_patch\_size\_x\_info\_quantizer (23)

The value of ath\_patch\_size\_x\_info\_quantizer shall be in the range of 0 to asps\_log2\_patch\_packing\_block\_size, inclusive.

**ath\_patch\_size\_y\_info\_quantizer** specifies the value of the quantizer PatchSizeYQuantizer that is to be applied to the syntax elements pdu\_2d\_size\_y\_minus1[ tileID ][ p ], mpdu\_2d\_delta\_size\_y[ tileID ][ p ], ipdu\_2d\_delta\_size\_y[ tileID ][ p ], rpdu\_2d\_size\_y\_minus1[ tileID ][ p ], and epdu\_2d\_size\_y\_minus1[ tileID ][ p ] of a patch with index p in a tile with tile ID equal to tileID. If ath\_patch\_size\_y\_info\_quantizer is not present, its value shall be inferred to be equal to asps\_log2\_patch\_packing\_block\_size. The value of PatchSizeYQuantizer is computed as follows:

PatchSizeYQuantizer = 1 << ath\_patch\_size\_y\_info\_quantizer (24)

The value of ath\_patch\_size\_y\_info\_quantizer shall be in the range of 0 to asps\_log2\_patch\_packing\_block\_size, inclusive.

**ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1** plus 1 specifies the number of bits in the fixed-length representation of the syntax elements rpdu\_3d\_offset\_u[ tileID ][ p ], rpdu\_3d\_offset\_v[ tileID ][ p ], and rpdu\_3d\_offset\_d[ tileID ][ p ] of a patch with index p in a tile with tile ID equal to tileID. When present, the length of the syntax element used to represent ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1 is equal to Floor( Log2( asps\_geometry\_3d\_bit\_depth\_minus1 + 1 ) ).

When ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1 is not present, its value shall be inferred to be equal to  
Max( 0, asps\_geometry\_3d\_bit\_depth\_minus1 − asps\_geometry\_2d\_bit\_depth\_minus1 ) − 1.

A variable RawShift is defined as follows:

If afps\_raw\_3d\_offset\_bit\_count\_explicit\_mode\_flag is equal to 1,

RawShift =   
 asps\_geometry\_3d\_bit\_depth\_minus1 − ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1 (25)

Otherwise

RawShift = asps\_geometry\_2d\_bit\_depth\_minus1 + 1 (26)

**ath\_num\_ref\_idx\_active\_override\_flag** equal to 1 specifies that the syntax element ath\_num\_ref\_idx\_active\_minus1 is present for the current atlas tile. ath\_num\_ref\_idx\_active\_override\_flag equal to 0 specifies that the syntax element ath\_num\_ref\_idx\_active\_minus1 is not present. If ath\_num\_ref\_idx\_active\_override\_flag is not present, its value shall be inferred to be equal to 0.

**ath\_num\_ref\_idx\_active\_minus1** is used for the derivation of the variable NumRefIdxActive as specified by Equation 27 for the current atlas tile. The value of ath\_num\_ref\_idx\_active\_minus1 shall be in the range of 0 to 14, inclusive.

When the current atlas tile is a P\_TILE atlas tile, ath\_num\_ref\_idx\_active\_override\_flag is equal to 1, and ath\_num\_ref\_idx\_active\_minus1 is not present, ath\_num\_ref\_idx\_active\_minus1 is inferred to be equal to 0.

The variable NumRefIdxActive is derived as follows:

if( ath\_type == P\_TILE || ath\_type == SKIP\_TILE ) {  
 if( ath\_num\_ref\_idx\_active\_override\_flag == 1 )  
 NumRefIdxActive = ath\_num\_ref\_idx\_active\_minus1 + 1 (27)  
 else {  
 if( num\_ref\_entries[ RlsIdx ] >= afps\_num\_ref\_idx\_default\_active\_minus1 + 1 )  
 NumRefIdxActive = afps\_num\_ref\_idx\_default\_active\_minus1 + 1  
 else  
 NumRefIdxActive = num\_ref\_entries[ RlsIdx ]  
 }  
 }  
 else   
 NumRefIdxActive = 0

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

#### Reference list structure semantics

**num\_ref\_entries**[ rlsIdx ] specifies the number of entries in the ref\_list\_struct( rlsIdx ) syntax structure, where rlsIdx is the index of an atlas frame reference list. For P\_TILE and SKIP\_TILE, the value of num\_ref\_entries[ rlsIdx ] shall be in the range of 1 to asps\_max\_dec\_atlas\_frame\_buffering\_minus1 + 1. Otherwise, the value of num\_ref\_entries[ rlsIdx ] shall be in the range of 0 to asps\_max\_dec\_atlas\_frame\_buffering\_minus1 + 1.

**st\_ref\_atlas\_frame\_flag**[ rlsIdx ][ i ] equal to 1 specifies that the i-th entry in the ref\_list\_struct( rlsIdx ) syntax structure is a short term reference atlas frame entry. st\_ref\_atlas\_frame\_flag[ rlsIdx ][ i ] equal to 0 specifies that the i-th entry in the ref\_list\_struct( rlsIdx ) syntax structure is a long term reference atlas frame entry. When not present, the value of st\_ref\_atlas\_frame\_flag[ rlsIdx ][ i ] is inferred to be equal to 1.

The variable NumLtrAtlasFrmEntries[ rlsIdx ] is derived as follows:

NumLtrAtlasFrmEntries[ rlsIdx ] = 0  
for( i = 0; i < num\_ref\_entries[ rlsIdx ]; i++ )  
 if( !st\_ref\_atlas\_frame\_flag[ rlsIdx ][ i ] ) (28)  
 NumLtrAtlasFrmEntries[ rlsIdx ]++

**abs\_delta\_afoc\_st**[ rlsIdx ][ i ], when the i-th entry is the first short term reference atlas frame entry in ref\_list\_struct( rlsIdx ) syntax structure, specifies the absolute difference between the atlas frame order count values of the current atlas tile and the atlas frame referred to by the i-th entry, or, when the i-th entry is a short term reference atlas frame entry but not the first short term reference atlas frame entry in the ref\_list\_struct( rlsIdx ) syntax structure, specifies the absolute difference between the atlas frame order count values of the atlas frames referred to by the i-th entry and by the previous short term reference atlas frame entry in the ref\_list\_struct( rlsIdx ) syntax structure.

The value of abs\_delta\_afoc\_st[ rlsIdx ][ i ] shall be in the range of 0 to 215 − 1, inclusive.

**straf\_entry\_sign\_flag**[ rlsIdx ][ i ] equal to 1 specifies that the i-th entry in the syntax structure ref\_list\_struct( rlsIdx ) has a value greater than or equal to 0. straf\_entry\_sign\_flag[ rlsIdx ][ i ] equal to 0 specifies that the i-th entry in the syntax structure ref\_list\_struct( rlsIdx ) has a value less than 0. When not present, the value of straf\_entry\_sign\_flag[ rlsIdx ][ i ] is inferred to be equal to 1.

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

for( i = 0; i < num\_ref\_entries[ rlsIdx ]; i++ )  
 if( st\_ref\_atlas\_frame\_flag[ rlsIdx ][ i ] )   
 DeltaAfocSt[ rlsIdx ][ i ] =  
 ( 2 \* straf\_entry\_sign\_flag[ rlsIdx ][ i ] – 1 ) \* abs\_delta\_afoc\_st[ rlsIdx ][ i ] (29)  
 else  
 DeltaAfocSt[ rlsIdx ][ i ] = 0

**afoc\_lsb\_lt**[ rlsIdx ][ i ] specifies the value of the atlas frame order count modulo MaxAtlasFrmOrderCntLsb of the atlas frame referred to by the i-th entry in the ref\_list\_struct( rlsIdx ) syntax structure. The length of the afoc\_lsb\_lt[ rlsIdx ][ i ] syntax element is Log2MaxAtlasFrmOrderCntLsb bits.

#### Common atlas frame RBSP semantics

**caf\_extension\_present\_flag** equal to 1 specifies that the syntax elements caf\_miv\_extension\_present\_flag and caf\_extension\_7bits are present in the common\_atlas\_frame\_rbsp( ) syntax structure. caf\_extension\_present\_flag equal to 0 specifies that the syntax elements caf\_miv\_extension\_present\_flag and caf\_extension\_7bits are not present.

**caf\_miv\_extension\_present\_flag** equal to 1 specifies that the caf\_miv\_extension( ) syntax structure is present in the common\_atlas\_frame\_rbsp( ) syntax structure. caf\_miv\_extension\_present\_flag equal to 0 specifies that this syntax structure is not present. When not present, the value of caf\_miv\_extension\_present\_flag is inferred to be equal to 0.

**caf\_extension\_7bits** equal to 0 specifies that no caf\_extension\_data\_flag syntax elements are present in the CAF RBSP syntax structure. When present, caf\_extension\_7bits shall be equal to 0 in bitstreams conforming to this version of this document. Values of caf\_extension\_7bits not equal to 0 are reserved for future use by ISO/IEC. Decoders shall allow the value of caf\_extension\_7bits to be not equal to 0 and shall ignore all caf\_extension\_data\_flag syntax elements in a CAF NAL unit. When not present, the value of caf\_extension\_7bits is inferred to be equal to 0.

**caf\_extension\_data\_flag** may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this document. Decoders conforming to this version of this document shall ignore all caf\_extension\_data\_flag syntax elements.

### Atlas tile data unit semantics

#### General atlas tile data unit semantics

**atdu\_patch\_mode**[ tileID ][ p ] indicates the patch mode for the patch with index p in the current atlas tile with tile ID equal to tileID. The permitted values for atdu\_patch\_mode[ tileID ][ p ] are specified in Table 7 for atlas tiles with ath\_type equal to I\_TILE, Table 8 for atlas tiles with ath\_type equal to P\_TILE, and Table 9 for atlas tiles with ath\_type equal to SKIP\_TILE. When not present, the value of atdu\_patch\_mode[ tileID ][ p ] is inferred to be equal to P\_SKIP. Modes indicated as reserved shall not be present in bitstreams conforming to this version of this document.

NOTE – A tile with ath\_type equal to SKIP\_TILE implies that the entire tile information is copied directly from the tile with the same id as that of the current tile that corresponds to the first reference atlas frame in the RefAtlasFrmList.

Table 7 – Patch modes for I\_TILE type atlas tiles

|  |  |  |
| --- | --- | --- |
| **atdu\_patch\_mode[ tileID ][ p ]** | **Identifier** | **Description** |
| 0 | I\_INTRA | Non-predicted patch mode |
| 1 | I\_RAW | RAW Point Patch mode |
| 2 | I\_EOM | EOM Point Patch mode |
| 3-13 | I\_RESERVED | Reserved modes for future use by ISO/IEC |
| 14 | I\_END | Patch termination mode |

Table 8 – Patch modes for P\_TILE type atlas tiles

|  |  |  |
| --- | --- | --- |
| **atdu\_patch\_mode[ tileID ][ p ]** | **Identifier** | **Description** |
| 0 | P\_SKIP | Patch Skip mode |
| 1 | P\_MERGE | Patch Merge mode |
| 2 | P\_INTER | Inter predicted Patch mode |
| 3 | P\_INTRA | Non-predicted Patch mode |
| 4 | P\_RAW | RAW Point Patch mode |
| 5 | P\_EOM | EOM Point Patch mode |
| 6-13 | P\_RESERVED | Reserved modes for future use by ISO/IEC |
| 14 | P\_END | Patch termination mode |

Table 9 – Patch modes for SKIP\_TILE type atlas tiles

|  |  |  |
| --- | --- | --- |
| **atdu\_patch\_mode[ tileID ][ p ]** | **Identifier** | **Description** |
| 0 | P\_SKIP | Patch Skip mode |

Table – 10 Patch types

|  |  |  |
| --- | --- | --- |
| **Value** | **Identifier** | **Description** |
| 0 | PROJECTED | patch with patch mode equal to I\_INTRA, P\_INTRA, P\_INTER, P\_MERGE, or P\_SKIP that is associated with projected information onto a 2D image |
| 1 | EOM | patch with patch mode equal to I\_EOM, P\_EOM, or P\_SKIP associated with EOM coded points |
| 2 | RAW | patch with patch mode equal to I\_RAW, P\_RAW, or P\_SKIP associated with RAW coded points |

#### Patch information data semantics

None

#### Patch data unit semantics

**pdu\_2d\_pos\_x**[ tileID ][ p ] specifies the x-coordinate of the top-left corner of the patch bounding box for patch p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

**pdu\_2d\_pos\_y**[ tileID ][ p ] specifies the y-coordinate of the top-left corner of the patch bounding box for patch p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

**pdu\_2d\_size\_x\_minus1**[ tileID ][ p ] plus 1 specifies the quantized width value of the patch with index p in the current atlas tile, with tile ID equal to tileID.

**pdu\_2d\_size\_y\_minus1**[ tileID ][ p ] plus 1 specifies the quantized height value of the patch with index p in the current atlas tile, with tile ID equal to tileID.

**pdu\_3d\_offset\_u**[ tileID ][ p ] specifies the shift to be applied to the reconstructed patch points in patch with index p of the current atlas tile, with tile ID equal to tileID, along the tangent axis. The value of pdu\_3d\_offset\_u[ tileID ][ p ] shall be in the range of 0 to 2asps\_geometry\_3d\_bit\_depth\_minus1 + 1 − 1, inclusive. The number of bits used to represent pdu\_3d\_offset\_u[ tileID ][ p ] is asps\_geometry\_3d\_bit\_depth\_minus1 + 1.

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

**pdu\_3d\_offset\_d**[ tileID ][ p ] specifies the shift to be applied to the reconstructed patch points in patch with index p of the current atlas tile, with tile ID equal to tileID, along the normal axis, Pdu3dOffsetD[ tileID ][ p ], as follows:

Pdu3dOffsetD[ tileID ][ p ] = pdu\_3d\_offset\_d[ tileID ][ p ] << ath\_pos\_min\_d\_quantizer (30)

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

The number of bits used to represent pdu\_3d\_offset\_d[ tileID ][ p ] is equal to (asps\_geometry\_3d\_bit\_depth\_minus1 – ath\_pos\_min\_d\_quantizer + 1).

**pdu\_3d\_range\_d**[ tileID ][ p ], if present, specifies the nominal maximum value of the shift expected to be present in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation, in patch with index p of the current atlas tile, with tile ID equal to tileID, along the normal axis, Pdu3dRangeD[ tileID ][ p ], as follows:

if( pdu\_3d\_range\_d[ tileID ][ p ] == 0 )  
 Pdu3dRangeD[ tileID ][ p ] = 0 (31)  
 else {   
 range = pdu\_3d\_range\_d[ tileID ][ p ] << ath\_pos\_delta\_max\_d\_quantizer  
 Pdu3dRangeD[ tileID ][ p ] = range – 1 (32)  
 }

Let the variable rangeDBitDepth be computed as follows:

rangeDBitDepth =   
 Min( asps\_geometry\_2d\_bit\_depth\_minus1, asps\_geometry\_3d\_bit\_depth\_minus1 ) + 1

If pdu\_3d\_range\_d[ tileID ][ p ] is not present the value of Pdu3dRangeD[ tileID ][ p ] is assumed to be equal to 2rangeDBitDepth – 1. When present, the value of Pdu3dRangeD[ tileID ][ p ] shall be in the range of 0 to 2rangeDBitDepth – 1, inclusive.

The number of bits used to represent pdu\_3d\_range\_d[ tileID ][ p ] is equal to ( rangeDBitDepth – ath\_pos\_delta\_max\_d\_quantizer ).

**pdu\_projection\_id**[ tileID ][ p ] specifies the values of the projection mode and of the index of the normal to the projection plane for the patch with index p of the current atlas tile, with tile ID equal to tileID. The value of pdu\_projection\_id[ tileID ][ p ] shall be in range of 0 to asps\_max\_number\_projections\_minus1, inclusive.

The number of bits used to represent pdu\_projection\_id[ tileID ][ p ] is Ceil( Log2( asps\_max\_number\_projections\_minus1 + 1) ).

**pdu\_orientation\_index**[ tileID ][ p ] specifies the patch orientation index, for the patch with index p of the current atlas tile, with tile ID equal to tileID, used to determine the transformation matrices, **Ro** and **Rs**, as indicated in Table 11, that are to be used to transform atlas coordinates of a patch to a local patch coordinate system denoted by coordinates ( u, v ), before conversion to 3D space coordinates. The number of bits used to represent pdu\_orientation\_index[ tileID ][ p ] is ( asps\_use\_eight\_orientations\_flag ? 3 : 1 ).

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

**pdu\_lod\_scale\_x\_minus1**[ tileID ][ p ] specifies the LOD scaling factor to be applied to the local x‑coordinate of a point in a patch with index p of the current atlas tile, with tile ID equal to tileID, prior to its addition to the patch coordinate TilePatch3dOffsetU[ tileID ][ p ]. If pdu\_lod\_scale\_x\_minus1[ tileID ][ p ] is not present, its value shall be inferred to be equal to 0.

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

#### Skip patch data unit semantics

None

#### Merge patch data unit semantics

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

**mpdu\_override\_2d\_params\_flag**[ tileID ][ p ] specifies whether the 2D parameters for the current patch with index p of the current atlas tile, with tile ID equal to tileID, are present in the bitstream.

**mpdu\_2d\_pos\_x**[ tileID ][ p ] specifies the difference of the x-coordinate of the top-left corner of the patch bounding box of patch with index p in the current atlas tile, with tile ID equal to tileID, and of the x‑coordinate of the top-left corner of the patch bounding box of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx, expressed as a multiple of PatchPackingBlockSize. When mpdu\_2d\_pos\_x[ tileID ][ p ] is not present, it is inferred to be equal to 0.

**mpdu\_2d\_pos\_y**[ tileID ][ p ] specifies the difference of the y-coordinate of the top-left corner of the patch bounding box of patch with index p in the current atlas tile, with tile ID equal to tileID, and of the y‑coordinate of the top-left corner of the patch bounding box of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx, expressed as a multiple of PatchPackingBlockSize. When mpdu\_2d\_pos\_x[ tileID ][ p ] is not present, it is inferred to be equal to 0.

**mpdu\_2d\_delta\_size\_x**[ tileID ][ p ] specifies the difference of the width values of the patch with index p in the current atlas tile, with tile ID equal to tileID, and the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx. When mpdu\_2d\_delta\_size\_x[ tileID ][ p ] is not present, it is inferred to be equal to 0.

**mpdu\_2d\_delta\_size\_y**[ tileID ][ p ] specifies the difference of the height values of the patch with index p in the current atlas tile, with tile ID equal to tileID, and the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to the reference refIdx. When mpdu\_2d\_delta\_size\_y[ tileID ][ p ] is not present, it is inferred to be equal to 0.

**mpdu\_override\_3d\_params\_flag**[ tileID ][ p ] specifies whether the 3D parameters for the current patch of the current atlas tile, with tile ID equal to tileID, are present in the bitstream. When mpdu\_override\_3d\_params\_flag[ tileID ][ p ] is not present, it is inferred to be equal to 0.

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

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

**mpdu\_3d\_offset\_d**[ tileID ][ p ] specifies the difference between the shift to be applied to the reconstructed patch points along the normal axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the normal axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx, Mpdu3dOffsetD[ tileID ][ p ], as follows:

Mpdu3dOffsetD[ tileID ][ p ] = mpdu\_3d\_offset\_d[ tileID ][ p ] << ath\_pos\_min\_d\_quantizer (33)

The value of mpdu\_3d\_offset\_d[ tileID ][ p ] shall be in the range of ( −2asps\_geometry\_3d\_bit\_depth\_minus1+1 + 1 ) to ( 2asps\_geometry\_3d\_bit\_depth\_minus1+1 – 1 ), inclusive. When mpdu\_3d\_offset\_d[ tileID ][ p ] is not present, it is inferred to be equal to 0.

**mpdu\_3d\_range\_d**[ tileID ][ p ], if present, specifies the difference between the nominal maximum value of the shift expected to be present in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation, in the patch with index p of the current atlas tile, with tile ID equal to tileID, along the normal axis and of the nominal maximum value of the shift expected to be present in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx, Mpdu3dRangeD[ tileID ][ p ], as follows:

Mpdu3dRangeD[ tileID ][ p ] = (mpdu\_3d\_range\_d[ tileID ][ p ] == 0) ?   
 0 : (mpdu\_3d\_range\_d[ tileID ][ p ] << ath\_pos\_delta\_max\_d\_quantizer ) – 1 (34)

If mpdu\_3d\_range\_d[ tileID ][ p ] is not present the value of Mpdu3dRangeD[ tileID ][ p ] is assumed to be equal to 0.

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

#### Inter patch data unit semantics

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

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

**ipdu\_2d\_pos\_x**[ tileID ][ p ] specifies the difference of the x‑coordinate of the top-left corner of the patch bounding box of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the x‑coordinate of the top-left corner of the patch bounding box of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx, expressed as a multiple of PatchPackingBlockSize.

**ipdu\_2d\_pos\_y**[ tileID ][ p ] specifies the difference of the y‑coordinate of the top-left corner of the patch bounding box of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the y‑coordinate of the top-left corner of the patch bounding box of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx, expressed as a multiple of PatchPackingBlockSize.

**ipdu\_2d\_delta\_size\_x**[ tileID ][ p ] specifies the difference of the width values of the patch with index p in the current atlas tile, with tile ID equal to tileID, and the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that is associated with the reference refIdx.

**ipdu\_2d\_delta\_size\_y**[ tileID ][ p ] specifies the difference of the height values of the patch with index p in the current atlas tile, with tile ID equal to tileID, and the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to the reference refIdx.

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

**ipdu\_3d\_offset\_v**[ tileID ][ p ] specifies the difference between the shift to be applied to the reconstructed patch points along the bi‑tangent axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the bi‑tangent axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx. The value of ipdu\_3d\_offset\_v[ tileID ][ p ] shall be in the range of ( −2asps\_geometry\_3d\_bit\_depth\_minus1+1 + 1 ) to ( 2asps\_geometry\_3d\_bit\_depth\_minus1+1 – 1 ), inclusive.

**ipdu\_3d\_offset\_d**[ tileID ][ p ] specifies the difference between the shift to be applied to the reconstructed patch points along the normal axis of the patch with index p in the current atlas tile, with tile ID equal to tileID, and of the shift to be applied to the reconstructed patch points along the normal axis of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx, Ipdu3dOffsetD[ tileID ][ p ], as follows:

Ipdu3dOffsetD[ tileID ][ p ] = ipdu\_3d\_offset\_d[ tileID ][ p ] << ath\_pos\_min\_d\_quantizer (35)

The value of ipdu\_3d\_offset\_d[ tileID ][ p ] shall be in the range of ( −2asps\_geometry\_3d\_bit\_depth\_minus1+1 + 1 ) to ( 2asps\_geometry\_3d\_bit\_depth\_minus1+1 – 1 ), inclusive.

**ipdu\_3d\_range\_d**[ tileID ][ p ], if present, specifies the difference between the nominal maximum value of the shift expected to be present in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation, in the patch with index p of the current atlas, with tile ID equal to tileID, tile along the normal axis and of the nominal maximum value of the shift expected to be presented in the reconstructed bit depth patch geometry samples, after conversion to their nominal representation of the patch with index RefPatchIdx in the atlas tile with the same ID as the current tile in the atlas frame that corresponds to refIdx, Ipdu3dRangeD[ tileID ][ p ], as follows:

Ipdu3dRangeD[ tileID ][ p ] = ( ipdu\_3d\_range\_d[ tileID ][ p ] == 0 ) ?   
 0 : ( ipdu\_3d\_range\_d[ tileID ][ p ] << ath\_pos\_delta\_max\_d\_quantizer ) – 1 (36)

If ipdu\_3d\_range\_d[ tileID ][ p ] is not present, the value of Ipdu3dRangeD[ tileID ][ p ] is assumed to be equal to 0.

#### RAW patch data unit semantics

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

**rpdu\_2d\_pos\_x**[ tileID ][ p ] specifies the x-coordinate of the top-left corner of the patch bounding box size for the RAW coded patch p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

**rpdu\_2d\_pos\_y**[ tileID ][ p ] specifies the y-coordinate of the top-left corner of the patch bounding box size for the RAW coded patch p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

**rpdu\_2d\_size\_x\_minus1**[ tileID ][ p ] plus 1 specifies the width value of the RAW coded patch with index p in the current atlas tile, with tile ID equal to tileID.

**rpdu\_2d\_size\_y\_minus1**[ tileID ][ p ] plus 1 specifies the height value of the RAW coded patch with index p in the current atlas tile, with tile ID equal to tileID.

**rpdu\_3d\_offset\_u**[ tileID ][ p ] specifies the shift to be applied to the reconstructed RAW patch points in the patch with index p of the current atlas tile, with tile ID equal to tileID, along the tangent axis. The number of bits used to represent rpdu\_3d\_offset\_u[ tileID ][ p ] is ( ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1 + 1 ).

**rpdu\_3d\_offset\_v**[ tileID ][ p ] specifies the shift to be applied to the reconstructed RAW patch points in the patch with index p of the current atlas tile, with tile ID equal to tileID, along the bi-tangent axis. The number of bits used to represent rpdu\_3d\_offset\_v[ tileID ][ p ] is ( ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1 + 1 ).

**rpdu\_3d\_offset\_d**[ tileID ][ p ] specifies the shift to be applied to the reconstructed RAW patch points in the patch with index p of the current atlas tile, with tile ID equal to tileID, along the normal axis. The number of bits used to represent rpdu\_3d\_offset\_d[ tileID ][ p ] is ( ath\_raw\_3d\_offset\_axis\_bit\_count\_minus1 + 1 ).

**rpdu\_points\_minus1**[ tileID ][ p ] plus 1 specifies the number of RAW points present in the RAW coded patch with index p in the current atlas tile, with tile ID equal to tileID. The value of rpdu\_points\_minus1[ tileID ][ p ] shall be in the range of 0 to ( ( rpdu\_2d\_size\_x\_minus1[ tileID ][ p ] + 1) \* (rpdu\_2d\_size\_y\_minus1[ tileID ][ p ] + 1 ) ) / 3 − 1, inclusive.

#### EOM patch data unit semantics

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

**epdu\_2d\_pos\_x**[ tileID ][ p ] specifies the x-coordinate of the top-left corner of the patch bounding box size for the current EOM patch p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

**epdu\_2d\_pos\_y**[ tileID ][ p ] specifies the y-coordinate of the top-left corner of the patch bounding box size for the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID, expressed as a multiple of PatchPackingBlockSize.

**epdu\_2d\_size\_x\_minus1**[ tileID ][ p ] plus 1 specifies the width value of the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID.

**epdu\_2d\_size\_y\_minus1**[ tileID ][ p ] plus 1 specifies the height value of the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID.

**epdu\_patch\_count\_minus1**[ tileID ][ p ] plus 1 indicates the number of associated patches that contain EOM points in the decoded occupancy video for the current EOM patch with index p in the current atlas tile, with tile ID equal to tileID. When not present epdu\_patch\_count\_minus1[ tileID ][ p ] is inferred to be equal to 0.

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

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

It is a requirement of bitstream conformance that, if ath\_type is equal to I\_TILE, that the value of atdu\_patch\_mode[ tileID ][ epdu\_associated\_patch\_idx[ tileID ][ p ][ i ] ] shall not be equal to I\_RAW or I\_EOM, and, if ath\_type is equal to P\_TILE, that the value of atdu\_patch\_mode[ tileID ][ epdu\_associated\_patch\_idx[ tileID ][ p ][ i ] ] shall not be equal to P\_RAW or P\_EOM.

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

#### Point local reconstruction data semantics

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

BlockCnt( xSize, ySize ) = ( ( xSize + BSize – 1 ) / BSize ) \* ( ( ySize + BSize – 1 ) / BSize ) (37)

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

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

**plrd\_level**[ tileID ][ p ][ mapIdx ] indicates the level at which point local reconstruction data, when enabled, is present in the bitstream for a map, with index mapIdx, associated with the current patch with index p. in the current atlas tile, with tile ID equal to tileID. When plrd\_level[ tileID ][ p ][ mapIdx ] is equal to 0 this syntax element indicates that point local reconstruction data and the syntax elements plrd\_present\_block\_flag[ tileID ][ p ][ mapIdx ][ i ] and plrd\_block\_mode\_minus1[ tileID ][ p ][ mapIdx ][ i ] are present in the bitstream for each block of the map. When plrd\_level[ tileID ][ p ][ mapIdx ] is equal to 1, this syntax element indicates that point local reconstruction data and the syntax elements plrd\_present\_flag[ tileID ][ p ][ mapIdx ] and plrd\_mode\_minus1[ tileID ][ p ][ mapIdx ] are present in the bitstream for the entire map, with index mapIdx, of the patch p in the current atlas tile with tile ID equal to tileID. In this case, all blocks of the map mapIdx shall use the same point local reconstruction data.

**plrd\_present\_block\_flag**[ tileID ][ p ][ mapIdx ][ i ] equal to 1 indicates that the syntax element plrd\_block\_mode\_minus1[ tileID ][ p ][ mapIdx ][ i ] is present in the bitstream for block i of the map mapIdx of the patch p in the current atlas tile, with tile ID equal to tileID. plrd\_present\_block\_flag[ tileID ][ p ][ mapIdx ][ i ] equal to 0 indicates that the syntax element plrd\_block\_mode\_minus1[ tileID ][ p ][ mapIdx ][ i ] is not present in the bitstream.

**plrd\_block\_mode\_minus1**[ tileID ][ p ][ mapIdx ][ i ] plus 1 indicates the point local reconstruction mode, plrBlkMode, selected from the list of point local reconstruction modes of size plri\_number\_of\_modes\_minus1[ mapIdx ] plus 1 and defined by the syntax elements plri\_interpolate\_flag[ mapIdx ][ plrBlkMode ], plri\_filling\_flag[ mapIdx ][plrBlkMode ], plri\_minimum\_depth[ mapIdx ][ plrBlkMode ], and plri\_neighbour\_minus1[ mapIdx ][ plrBlkMode ] for block i of the map with index mapIdx of the patch p in the current atlas tile, with tile ID equal to tileID. plrd\_block\_mode\_minus1[ tileID ][ p ][ mapIdx ][ i ] shall be in the range of 0 to plri\_number\_of\_modes\_minus1[ mapIdx ], inclusive.

NOTE 2 – When PLR mode is equal to 0 at the block or patch level, it is not signalled in the bitstream. The value 0 corresponds to a default mode value of PLR which is implemented by the values of the syntax elements plri\_interpolate\_flag[ mapIdx ][ 0 ], plri\_filling\_flag[ mapIdx ][ 0 ], plri\_minimum\_depth[ mapIdx ][ 0 ], and plri\_neighbour\_minus1[ mapIdx ][ 0 ].

**plrd\_present\_flag**[ tileID ][ p ][ mapIdx ] equal to 1 indicates that the syntax element plrd\_mode\_minus1[ tileID ][ p ][ mapIdx ] is present in the bitstream for the map with index mapIdx of the patch p in the current atlas tile, with tile ID equal to tileID. plrd\_present\_flag[ tileID ][ p ][ mapIdx ] equal to 0 indicates that the syntax element plrd\_block\_mode\_minus1[ tileID ][ p ][ mapIdx ][ i ] is not present in the bitstream.

**plrd\_mode\_minus1**[ tileID ][ p ][ mapIdx ] plus 1 indicates the point local reconstruction mode, plrMode, selected from the list of point local reconstruction modes of size plri\_number\_of\_modes\_minus1[ mapIdx ] plus 1 and defined by the syntax elements plri\_interpolate\_flag[ mapIdx ][ plrMode ], plri\_filling\_flag[ mapIdx ][plrMode ], plri\_minimum\_depth[ mapIdx ][ plrMode ], and plri\_neighbour\_minus1[ mapIdx ][ plrMode ] for all blocks of the map with index mapIdx of the patch p in the current atlas tile, with tile ID equal to tileID. plrd\_mode\_minus1[ tileID ][ p ][ mapIdx ] shall be in the range of 0 to plri\_number\_of\_modes\_minus1[ mapIdx ], inclusive.

### Supplemental enhancement information message semantics

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

**sm\_payload\_type\_byte** is a byte of the payload type of an SEI message.

**sm\_payload\_size\_byte** is a byte of the payload size of an SEI message.

# Decoding process

## General decoding process

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

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

The decoding process is specified such that all decoders that conform to a specific image/video coding specification and, potentially, a specific profile, tier, and level that may be defined within such specification, will produce numerically identical outputs when invoking the decoding process associated with that profile for a bitstream conforming to that profile, tier, and level. Any decoding process that produces identical outputs to those produced by the process described herein (with the correct output order or output timing, as specified) conforms to the decoding process requirements of this document.

If the input to this process is a V3C bitstream then the V3C bitstream is parsed and demultiplexed into several V3C sub-bitstreams, as specified in subclause 9.8.2 or by external means, with each sub-bitstream corresponding to a different set of information. In particular, the V3C bitstream is demultiplexed into one or more atlas bitstreams, if multiple atlases are present, and their associated video bitstreams. Video bitstreams for each atlas may include occupancy, geometry, and attribute components.

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

* The decoding process for the V3C sub-bitstream component corresponding to the atlas component, which is determined through either examining if vuh\_unit\_type is equal to V3C\_AD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.2. This V3C sub-bitstream is also referred to as AtlasBitstreamToDecode in this document.
* The decoding process for the V3C sub-bitstream component corresponding to the occupancy component, which is determined through either examining if vuh\_unit\_type is equal to V3C\_OVD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.3.
* The decoding process for the V3C sub-bitstream component corresponding to the geometry components, which is determined through either examining if vuh\_unit\_type is equal to V3C\_GVD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.4.
* The decoding process for the V3C sub-bitstream component corresponding to the attribute components, which is determined through either examining if vuh\_unit\_type is equal to V3C\_AVD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.5.
* The decoding process for the V3C sub-bitstream component corresponding to the packed video component, which is determined through either examining if vuh\_unit\_type is equal to V3C\_PVD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.6.
* The decoding process for the V3C sub-bitstream component corresponding to the common atlas component, which is determined through either examining if vuh\_unit\_type is equal to V3C\_CAD or through external means if the V3C unit header is unavailable, is as specified in subclause 9.7.

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

[Ed.(BK): The terms atlas component, attribute component, etc. are not properly defined.]

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

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

## Atlas data decoding process

### General atlas data decoding process

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

Outputs of this process are:

* the number of decoded atlas frames, NumDecAtlasFrames,
* for each decoded atlas frame, the corresponding upper-case variables from subclauses 9.2.5 and/or 9.2.7,
* for each decoded atlas frame, the associated geometry, attribute, and occupancy parameters contained in non-ACL NAL units, and
* the following 1D arrays of size NumDecAtlasFrames:
* DecAtlasOutOrdIdx, indicating the atlas output order index, and
* DecAtlasCompTime, indicating the atlas composition time.

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

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

### Decoding process for a coded atlas frame

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

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

When the current atlas frame is a BLA atlas frame that has nal\_unit\_type equal to NAL\_BLA\_W\_LP or NAL\_GBLA\_W\_LP, or is a CRA atlas frame, the following applies:

* If some external means not specified in this document is available to set the variable UseAltCabParamsFlag to a value, UseAltCabParamsFlag is set equal to the value provided by the external means.
* Otherwise, the value of UseAltCabParamsFlag is set equal to 0.

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

* If the current atlas frame is an IDR atlas frame, a BLA atlas frame, the first atlas frame in the bitstream in decoding order, or the first atlas frame that follows an end of sequence NAL unit in decoding order, the variable NoOutputBeforeRecoveryFlag is set equal to 1.
* Otherwise, if some external means not specified in this document is available to set the variable HandleCraAsBlaFlag to a value for the current atlas frame, the variable HandleCraAsBlaFlag is set equal to the value provided by the external means and the variable NoOutputBeforeRecoveryFlag is set equal to HandleCraAsBlaFlag.
* Otherwise, the variable HandleCraAsBlaFlag is set equal to 0 and the variable NoOutputBeforeRecoveryFlag is set equal to 0.

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

1. The decoding of atlas NAL units is specified in subclause 9.2.3.
2. The processes in subclause 9.2.4 specify the following decoding processes using syntax elements in the atlas tile header layer and above:

* Variables and functions relating to the atlas frame order count are derived as specified in subclause 9.2.4.1.
* At the beginning of the decoding process for each atlas tile, the reference atlas frame list construction process specified in subclause 9.2.4.3 is invoked for the derivation of the reference atlas frame list, RefAtlasFrmList.
* The reference atlas frame marking process in subclause 9.2.4.4 is invoked, wherein reference atlas frames might be marked as "unused for reference" or "used for long-term reference".
* When the current atlas frame is a BLA atlas frame or is a CRA atlas frame with NoOutputBeforeRecoveryFlag equal to 1, the decoding process for generating unavailable reference atlas frames specified in subclause 9.2.4.2 is invoked, which needs to be invoked only for the first atlas tile of an atlas frame.
* The variable AtlasFrameOutputFlag is set as follows:
* If the current atlas frame is a RASL atlas frame and NoOutputBeforeRecoveryFlag of the associated IRAP coded atlas frame is equal to 1, AtlasFrameOutputFlag is set equal to 0.
* Otherwise, AtlasFrameOutputFlag is set equal to ath\_atlas\_output\_flag.

1. The processes in subclause 9.2.5 specify the patch decoding processes according to the patch mode as follows:

* Decoding of intra coded patches is specified in subclause 9.2.5.2.
* Decoding of skip mode coded patches is specified in subclause 9.2.5.3.
* Decoding of merge mode coded patches is specified in subclause 9.2.5.4.
* Decoding of inter coded patches is specified in subclause 9.2.5.5.
* Decoding of RAW coded patches is specified in subclause 9.2.5.6.
* Decoding of EOM coded patches is specified in subclause 9.2.5.7.

1. The process in subclause 9.2.6 specifies the creation of the block to patch map after the decoding of all patches present within the current atlas tile.
2. After all atlas tiles associated with the current atlas have been decoded, the current atlas is marked as "used for short-term reference".
3. An informative process is defined in subclause 9.2.7 that specifies the conversion of tile level patch information to atlas level patch information.

### Atlas NAL unit decoding process

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

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

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

### Atlas tile header decoding process

#### Atlas frame order count derivation process

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

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

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

When the current atlas frame is not an IRAP coded atlas with NoOutputBeforeRecoveryFlag equal to 1, the variables prevAtlasFrmOrderCntLsb and prevAtlasFrmOrderCntMsb are derived as follows:

* Let prevAtlasFrm be the previous atlas frame in decoding order that has TemporalID equal to 0 and that is not a RASL, RADL, or SLNR coded atlas.
* The variable prevAtlasFrmOrderCntLsb is set equal to the atlas frame order count LSB value, ath\_atlas\_frm\_order\_cnt\_lsb, of prevAtlasFrm.
* The variable prevAtlasFrmOrderCntMsb is set equal to AtlasFrmOrderCntMsb of prevAtlasFrm.

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

* If the current atlas is an IRAP coded atlas with NoOutputBeforeRecoveryFlag equal to 1, AtlasFrmOrderCntMsb is set equal to 0.
* Otherwise, AtlasFrmOrderCntMsb is derived as follows:

if( ( ath\_atlas\_frm\_order\_cnt\_lsb < prevAtlasFrmOrderCntLsb ) &&  
 ( ( prevAtlasFrmOrderCntLsb − ath\_atlas\_frm\_order\_cnt\_lsb ) >=   
 ( MaxAtlasFrmOrderCntLsb / 2 ) ) )

AtlasFrmOrderCntMsb =  
 prevAtlasFrmOrderCntMsb + MaxAtlasFrmOrderCntLsb (38)

else if( ( ath\_atlas\_frm\_order\_cnt\_lsb > prevAtlasFrmOrderCntLsb ) &&  
 ( ( ath\_atlas\_frm\_order\_cnt\_lsb – prevAtlasFrmOrderCntLsb ) >  
 ( MaxAtlasFrmOrderCntLsb / 2 ) ) )

AtlasFrmOrderCntMsb =  
 prevAtlasFrmOrderCntMsb – MaxAtlasFrmOrderCntLsb (39)

else

AtlasFrmOrderCntMsb = prevAtlasFrmOrderCntMsb (40)

AtlasFrmOrderCntVal is derived as follows:

AtlasFrmOrderCntVal = AtlasFrmOrderCntMsb + ath\_atlas\_frm\_order\_cnt\_lsb (41)

The value of AtlasFrmOrderCntVal shall be in the range of −231 to 231 − 1, inclusive. In one CAS, the AtlasFrmOrderCntVal values for any two coded atlas frames with the same value of nal\_layer\_id shall not be the same.

The function AtlasFrmOrderCnt( aFrmX ) is specified as follows:

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

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

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

The bitstream shall not contain data that result in values of DiffAtlasFrmOrderCnt( aFrmA, aFrmB) used in the decoding process that are not in the range of −215 to 215 − 1, inclusive.

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

#### Decoding process for generating unavailable reference atlas frames

##### General

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

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

When this process is invoked, the following applies:

* For each RefAtlasFrmList[ j ], with j in the range of 0 to num\_ref\_entries[ RlsIdx ] − 1, inclusive, that is equal to "no reference atlas frame", an atlas frame is generated as specified in subclause 9.2.4.2.2 and the following applies:
  + The value of nal\_layer\_id for the generated atlas frame is set equal to the nal\_layer\_id value of the current atlas frame.
  + If st\_ref\_atlas\_frame\_flag[ RlsIdx ][ j ] is equal to 1 the value of AtlasFrmOrderCntVal for the generated atlas frame is set equal to RefAtlasFrmList[ j ] and the generated atlas frame is marked as "used for short-term reference".
  + Otherwise, when st\_ref\_atlas\_frame\_flag[ RlsIdx ][ j ] is equal to 0 the value of AtlasFrmOrderCntVal for the generated atlas frame is set equal to RefAtlasFrmList[ j ], the value of ath\_atlas\_frm\_order\_cnt\_lsb for all tiles in the generated atlas frame is inferred to be equal to ( RefAtlasFrmList[ j ] & ( MaxAtlasFrmOrderCntLsb – 1 ) ), and the generated atlas frame is marked as "used for long-term reference".
  + The value of AtlasFrameOutputFlag for the generated reference atlas frame is set equal to 0.
  + RefAtlasFrmList[ j ] is set to be the generated reference atlas frame.
  + The value of TemporalID for the generated atlas frame is set equal to the TemporalID value of the current atlas frame.
  + The value of ath\_atlas\_frame\_parameter\_set\_id for the generated atlas frame is set equal to ath\_atlas\_frame\_parameter\_set\_id of the current atlas frame.
  + The value of nal\_layer\_id for the generated atlas frame is set equal to nal\_layer\_id of the current atlas frame.

##### Generation of one unavailable atlas frame

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

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

for( t = 0; t<= afti\_num\_tiles\_in\_atlas\_frame\_minus1; t++ ) {  
 tileID = TileIndexToID( t )  
 AtduTotalNumPatches[ tileID ] = MaxNumProjPatches  
 for( p = 0; p<= AtduTotalNumPatches[tileID ]; p++ ) {  
 TilePatchType[ tileID ][ p ] = PROJECTED  
 TilePatch2dPosX[ tileID ][ p ] = 0  
 TilePatch2dPosY[ tileID ][ p ] = 0  
 TilePatch2dSizeX[ tileID ][ p ] = 1  
 TilePatch2dSizeY[ tileID ][ p ] = 1  
 TilePatch3dOffsetU[ tileID ][ p ] = 0  
 TilePatch3dOffsetV[ tileID ][ p ] = 0  
 TilePatch3dOffsetD[ tileID ][ p ] = 0  
 TilePatch3dRangeD[ tileID ][ p ] = 1  
 TilePatchProjectionID[ tileID ][ p ] = 0  
 TilePatchOrientationIndex[ tileID ][ p ] = 0  
 TilePatchLoDScaleX[ tileID ][ p ] = 1  
 TilePatchLoDScaleY[ tileID ][ p ] = 1  
  
 if( asps\_plr\_enabled\_flag == 1 ) {  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ )  
 TilePatchPlrdLevel[ tileID ][ p ][ m ] = 0  
 }  
 }  
 }

#### Reference atlas frame list construction process

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

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

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

NOTE 1 – For an I\_TILE atlas tile of an atlas frame, RefAtlasFrmList could be derived for bitstream conformance checking purposes, but its derivation is not necessary for the decoding of the current atlas frame or for atlas frames that follow the current atlas frame in decoding order.

The reference atlas frame list RefAtlasFrmList is constructed as follows:

for( j = 0, afocBase = AtlasFrmOrderCntVal; j < num\_ref\_entries[ RlsIdx ]; j++) { (44)  
 if( st\_ref\_atlas\_frame\_flag[ RlsIdx ][ j ] ) {  
 RefAtlasFrmAfocList[ j ] = afocBase − DeltaAfocSt[ RlsIdx ][ j ]  
 if( there is a reference atlas frame afA in the DAB with   
 AtlasFrmOrderCntVal equal to RefAtlasFrmAfocList[ j ] )  
 RefAtlasFrmList[ j ] = afA  
 else  
 RefAtlasFrmList[ j ] = "no reference atlas frame"  
 afocBase = RefAtlasFrmAfocList[ j ]  
 } else {  
 if( there is a reference atlas frame afA in the DAB with AtlasFrmOrderCntVal & ( MaxLtAtlasFrmOrderCntLsb − 1 )  
 equal to FullAtlasFrmOrderCntLsbLt[ RlsIdx ][ j ] )  
 RefAtlasFrmList[ j ] = afA  
 else  
 RefAtlasFrmList[ j ] = "no reference atlas frame"  
 }  
 }

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

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

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

* num\_ref\_entries[ RlsIdx ] shall not be less than NumRefIdxActive.
* The atlas frame referred to by each active entry in RefAtlasFrmList shall be present in the DAB and shall have TemporalID less than or equal to that of the current atlas frame.
* The atlas frame referred to by each entry in RefAtlasFrmList shall not be the current atlas frame.
* A short term reference atlas frame entry and a long term reference atlas frame entry in RefAtlasFrmList of an atlas tile shall not refer to the same atlas frame.
* There shall be no long term reference atlas frame entry in RefAtlasFrmList for which the difference between the AtlasFrmOrderCntVal of the current atlas tile and the AtlasFrmOrderCntVal of the atlas frame referred to by the entry is greater than or equal to 224.
* Let setOfRefAtlasFrms be the set of unique atlas frames referred to by all entries in RefAtlasFrmList that have the same nal\_layer\_id as the current atlas frame. The number of atlas frames in setOfRefAtlasFrms shall be less than or equal to asps\_max\_dec\_atlas\_frame\_buffering\_minus1, and setOfRefAtlasFrms shall be the same for all atlas tiles of an atlas frame.
* The atlas frames referred to by each active entry in RefAtlasFrmList shall have exactly the same number of tiles as the current atlas frame.
* The RefAtlasFrmList of all tiles in the current atlas frame shall contain the same unique atlas frames, without, however, any restrictions in ordering of the reference atlas frames.
* When the current atlas frame, with nal\_layer\_id equal to a particular value layerID, is an IRAP coded atlas, there shall be no atlas referred to by an entry in RefAtlasFrmList that precedes, in output order or decoding order, any preceding IRAP coded atlas with nal\_layer\_id equal to layerID in decoding order (when present).
* When the current atlas frame is not a RASL coded atlas associated with a CRA coded atlas with NoOutputBeforeRecoveryFlag equal to 1, there shall be no atlas referred to by an active entry in RefAtlasFrmList that was generated by the decoding process for generating unavailable reference atlas frames for the CRA coded atlas associated with the current atlas.
* When the current atlas frame follows an IRAP coded atlas having the same value of nal\_layer\_id in both decoding order and output order, there shall be no atlas frame referred to by an active entry in RefAtlasFrmList that precedes that IRAP coded atlas in output order or decoding order.
* When the current atlas frame follows an IRAP coded atlas having the same value of nal\_layer\_id and all the leading atlas frames, if any, associated with that IRAP coded atlas in both decoding order and output order, there shall be no atlas referred to by an entry in RefAtlasFrmList that precedes that IRAP coded atlas in output order or decoding order.
* When the current atlas frame is a RADL coded atlas, there shall be no active entry in RefAtlasFrmList that is an atlas frame that precedes the associated IRAP coded atlas of the RADL coded atlas in decoding order.

#### Reference atlas frame marking process

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

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

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

For all cases, the following applies:

* For each long term reference atlas frame entry in RefAtlasFrmList, when the atlas frame is marked as "used for short-term reference" and has the same nal\_layer\_id as the current atlas frame, the atlas frame is marked as "used for long-term reference".
* Each reference atlas frame in the same nal\_layer\_id as the current atlas frame in the DAB that is not referred to by any entry in RefAtlasFrmList is marked as "unused for reference".

### Decoding process for patch data units

#### General decoding process for patch data units

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

Outputs of this process are several parameters associated with the current patch with patch index p, including its 2D and corresponding 3D location information.

More specifically the following variables are derived:

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

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

Let the variables horLimit and verLimit be computed as follows:

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

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

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

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

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

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

TilePatch2dPosX[ tileID ][ p ] + TilePatch2dSizeX[ tileID ][ p ] <= horLimit  (45)

TilePatch2dPosY[ tileID ][ p ] + TilePatch2dSizeY[ tileID ][ p ] <= verLimit  (46)

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

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

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

TilePatch3dRangeD[ tileID ][ p ] specifies for the current patch with patch index p, of the current tile with tile ID equal to tileID, the expected range of the geometry data along the normal axis in the corresponding geometry maps. Let the variable rangeDBitDepth be equal to Min( asps\_geometry\_2d\_bit\_depth\_minus1, asps\_geometry\_3d\_bit\_depth\_minus1 ) + 1. The value of TilePatch3dRangeD[ tileID ][ p ] shall be in the range of 0 to 2rangeDBitDepth – 1, inclusive.

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

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

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

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

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

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

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

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

TilePatchPlrdLevel[ tileID ][ p ][ m ] specifies the level, i.e. whether it is at the block or patch level, where PLR data information is present for the m-th map associated with the current patch with patch index p, of the current tile with tile ID equal to tileID, where m is in the range if 0 to asps\_map\_count\_minus1, inclusive.

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

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

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

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

These variables are initially set as follows:

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

If atdu\_patch\_mode[ tileID ][ p ] is equal to I\_INTRA or P\_INTRA, then the process for decoding intra coded patches in subclause 9.2.5.2 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu\_patch\_mode[ tileID ][ p ] is equal to P\_SKIP, then the process for decoding skip coded patches in subclause 9.2.5.3 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu\_patch\_mode[ tileID ][ p ] is equal to P\_MERGE, then the process for decoding merge coded patches in subclause 9.2.5.4 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu\_patch\_mode[ tileID ][ p ] is equal to P\_INTER, then the process for decoding inter coded patches in subclause 9.2.5.5 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu\_patch\_mode[ tileID ][ p ] is equal to I\_RAW or P\_RAW, then the process for decoding RAW coded patches in subclause 9.2.5.6 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

If atdu\_patch\_mode[ tileID ][ p ] is equal to I\_EOM or P\_EOM, then the process for decoding EOM coded patches in subclause 9.2.5.7 is used, with p and tileID as the inputs to that process, and the outputs of that process are used as the output.

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

For PROJECTED patches then the following applies:

* A variable oIdx is set to TilePatchOrientationIndex[ tileID ][ p ].
* Variables posX, posY, sizeX, sizeY, lodX, and lodY are assigned as follows:

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

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

(47)

where **R**o and **R**s are specified in Table 11.

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| **oIdx** | **Ro( oIdx )** | **Rs( oIdx )** | **Description** |
| 0 |  |  | No transformation |
| 1 |  |  | The x and y axes are swapped |
| 2 |  |  | The x axis is inverted first and then x and y axes are swapped |
| 3 |  |  | The x and y axes are inverted |
| 4 |  |  | The y axis is inverted first and then x and y axes are swapped |
| 5 |  |  | The x axis is inverted |
| 6 |  |  | The x and y axes are inverted first and then are swapped |
| 7 |  |  | The y axis is inverted |

#### Decoding process for patch data units coded in intra mode

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

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

TilePatch2dPosX[ tileID ][ p ] = pdu\_2d\_pos\_x[ tileID ][ p ] \* PatchPackingBlockSize (48)

TilePatch2dPosY[ tileID ][ p ] = pdu\_2d\_pos\_y[ tileID ][ p ] \* PatchPackingBlockSize (49)

TilePatch3dOffsetU[ tileID ][ p ] = pdu\_3d\_offset\_u[ tileID ][ p ] (50)

TilePatch3dOffsetV[ tileID ][ p ] = pdu\_3d\_offset\_v[ tileID ][ p ] (51)

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

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

TilePatchProjectionID[ tileID ][ p ] = pdu\_projection\_id[ tileID ][ p ] (54)

TilePatchOrientationIndex[ tileID ][ p ] = pdu\_orientation\_index[ tileID ][ p ] (55)

TilePatchLoDScaleX[ tileID ][ p ] =  
 pdu\_lod\_enabled\_flag[ p ] ? pdu\_lod\_scale\_x\_minus1[ tileID ][ p ] + 1 : 1 (56)

offsetY = ((pdu\_lod\_scale\_x\_minus1[ tileID ][ p ] > 0) ? 1 : 2)

TilePatchLoDScaleY[ tileID ][ p ] = pdu\_lod\_enabled\_flag[ tileID ][ p ] ?  
 pdu\_lod\_scale\_y\_idc[ tileID ][ p ] + offsetY : 1 (57)

TilePatch2dSizeX[ tileID ][ p ] =  
 (pdu\_2d\_size\_x\_minus1[ tileID ][ p ] + 1) \* PatchSizeXQuantizer (58)

TilePatch2dSizeY[ tileID ][ p ] =  
 (pdu\_2d\_size\_y\_minus1[ tileID ][ p ] + 1) \* PatchSizeYQuantizer (59)

If asps\_plr\_enabled\_flag is equal to 1, the following applies:

blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )

for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 TilePatchPlrdLevel[ tileID ][ p ][ m ] = plrd\_level[ tileID ][ p ][ m ]  
 if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =  
 plrd\_present\_block\_flag[ tileID ][ p ][ m ][ j ] if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )  
 TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =  
 plrd\_block\_mode\_minus1[ tileID ][ p ][ m ][ j ]  
 }  
 } else {  
 TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = plrd\_present\_flag[ tileID ][ p ][ m ]  
 if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )  
 TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] =  
 plrd\_mode\_minus1[ tileID ][ p ][ m ]  
 }  
 }  
 }

#### Decoding process for patch data units coded in skip prediction mode

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

First, refIdx is set to 0.

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the ( refIdx + 1 )-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[ refIdx ].

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

The variable RefPatchIdx, which corresponds to the predictor patch index in the tile with ID equal to tileID, in the reference atlas frame, refAtlasFrm, is computed as:

RefPatchIdx = PredictorIdx (60)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause 9.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, and refPatchEomPatchCount, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

Initially, the following parameters are derived:

TilePatchType[ tileID ][ p ] = refPatchType (61)

TilePatchInAuxVideo[ tileID ][ p ] = refPatchInAuxVideo (62)

TilePatch2dPosX[ tileID ][ p ] = refPatch2dPosX (63)

TilePatch2dPosY[ tileID ][ p ] = refPatch2dPosY (64)

TilePatch2dSizeX[ tileID ][ p ] = refPatch2dSizeX (65)

TilePatch2dSizeY[ tileID ][ p ] = refPatch2dSizeY (66)

In addition, if refPatchType is equal to RAW, the associated 2D and 3D patch parameters are derived as follows:

TilePatch3dOffsetU[ tileID ][ p ] = refPatch3dOffsetU (67)

TilePatch3dOffsetV[ tileID ][ p ] = refPatch3dOffsetV (68)

TilePatch3dOffsetD[ tileID ][ p ] = refPatch3dOffsetD (69)

TilePatch3dRangeD[ tileID ][ p ] = refPatch3dRangeD (70)

TilePatchRawPoints[ tileID ][ p ] = refPatchRawPoints (71)

Otherwise, if refPatchType is equal to EOM, the following applies:

TilePatchEomPatchCount[ tileID ][ p ] = refPatchEomPatchCount (72)

for( i = 0; i < TilePatchEomPatchCount[ tileID ][ p ]; i++ ) {

TilePatchAssociatedPatchIndex[ tileID ][ p ][ i ] = refPatchAssociatedPatchIndex[ i ] (73)

TilePatchEomPoints[ tileID ][ p ][ i ] = refPatchEomPoints[ i ] (74)

}

Otherwise, the associated 2D and 3D patch parameters are derived as follows:

TilePatch3dOffsetU[ tileID ][ p ] = refPatch3dOffsetU (75)

TilePatch3dOffsetV[ tileID ][ p ] = refPatch3dOffsetV (76)

TilePatch3dOffsetD[ tileID ][ p ] = refPatch3dOffsetD (77)

TilePatch3dRangeD[ tileID ][ p ] = refPatch3dRangeD (78)

TilePatchProjectionID[ tileID ][ p ] = refPatchProjectionID (79)

TilePatchOrientationIndex[ tileID ][ p ] = refPatchOrientationIndex (80)

TilePatchLoDScaleX[ tileID ][ p ] = refPatchLoDScaleX (81)

TilePatchLoDScaleY[ tileID ][ p ] = refPatchLoDScaleY (82)

If asps\_plr\_enabled\_flag is equal to 1, the PLR patch parameters are derived as follows:

blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) { (83)  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 TilePatchPlrdLevel[ tileID ][ p ][ m ] = refPatchPlrdLevel[ m ]  
 if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =  
 refPatchPlrdPresentBlockFlag[ m ][ j ]  
 if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )  
 TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =  
 refPatchPlrdBlockModeMinus1[ m ][ j ]  
 }  
 } else {  
 TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = refPatchPlrdPresentFlag[ m ]  
 if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )  
 TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] = refPlrdModeMinus1[ m ]  
 }  
 }  
 }

#### Decoding process for patch data units coded in merge prediction mode

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

First, the reference atlas frame index, refIdx, is derived as mpdu\_ref\_index[ tileID ][ p ].

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the ( refIdx + 1 )-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[ refIdx ].

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

Then the predictor patch index, RefPatchIdx, in the tile with ID equal to tile ID, in the reference atlas frame, refAtlasFrm, is computed as:

RefPatchIdx = PredictorIdx (84)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause 9.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, and refPatchEomPatchCount, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

It is a requirement of bitstream conformance that refPatchType is equal to PROJECTED.

First, TilePatchType[ tileID ][ p ] is assigned as follows:

TilePatchType[ tileID ][ p ] = refPatchType (85)

If mpdu\_override\_2d\_params\_flag[ tileID ][ p ] is equal to 1, the associated 2D patch parameters are derived as follows:

TilePatch2dPosX[ tileID ][ p ] =  
 refPatch2dPosX + mpdu\_2d\_pos\_x[ tileID ][ p ] \* PatchPackingBlockSize (86)

TilePatch2dPosY[ tileID ][ p ] =  
 refPatch2dPosY + mpdu\_2d\_pos\_y[ tileID ][ p ] \* PatchPackingBlockSize (87)

TilePatch2dSizeX[ tileID ][ p ] =  
 refPatch2dSizeX + mpdu\_2d\_delta\_size\_x[ tileID ][ p ] \* PatchSizeXQuantizer (88)

TilePatch2dSizeY[ tileID ][ p ] =  
 refPatch2dSizeY + mpdu\_2d\_delta\_size\_y[ tileID ][ p ] \* PatchSizeYQuantizer (89)

Otherwise,

TilePatch2dPosX[ tileID ][ p ] = refPatch2dPosX (90)

TilePatch2dPosY[ tileID ][ p ] = refPatch2dPosY (91)

TilePatch2dSizeX[ tileID ][ p ] = refPatch2dSizeX (92)

TilePatch2dSizeY[ tileID ][ p ] = refPatch2dSizeY (93)

If mpdu\_override\_3d\_params\_flag[ tileID ][ p ] is equal to 1, then the associated 3D patch parameters are derived as follows:

TilePatch3dOffsetU[ tileID ][ p ] = refPatch3dOffsetU + mpdu\_3d\_offset\_u[ tileID ][ p ] (94)

TilePatch3dOffsetV[ tileID ][ p ] = refPatch3dOffsetV + mpdu\_3d\_offset\_v[ tileID ][ p ] (95)

TilePatch3dOffsetD[ tileID ][ p ] = refPatch3dOffsetD + Mpdu3dOffsetD[ tileID ][ p ] (96)

TilePatch3dRangeD[ tileID ][ p ] = refPatch3dRangeD + Mpdu3dRangeD[ tileID ][ p ] (97)

Otherwise,

TilePatch3dOffsetU[ tileID ][ p ] = refPatch3dOffsetU (98)

TilePatch3dOffsetV[ tileID ][ p ] = refPatch3dOffsetV (99)

TilePatch3dOffsetD[ tileID ][ p ] = refPatch3dOffsetD (100)

TilePatch3dRangeD[ tileID ][ p ] = refPatch3dRangeD (101)

The following variables are set:

TilePatchProjectionID[ tileID ][ p ] = refPatchProjectionID (102)

TilePatchOrientationIndex[ tileID ][ p ] = refPatchOrientationIndex (103)

TilePatchLoDScaleX[ tileID ][ p ] = refPatchLoDScaleX (104)

TilePatchLoDScaleY[ tileID ][ p ] = refPatchLoDScaleY (105)

If asps\_plr\_enabled\_flag is equal to 1, the following applies:

blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )

If OverridePlrFlag is equal to 1, the associated PLR parameters are derived as follows:

for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 TilePatchPlrdLevel[ tileID ][ p ][ m ] = plrd\_level[ tileID ][ p ][ m ]  
 if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =  
 plrd\_present\_block\_flag[ tileID ][ p ][ m ][ j ]  
 if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )  
 TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =  
 plrd\_block\_mode\_minus1[ tileID ][ p ][ m ][ j ]  
 }  
 } else {  
 TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = plrd\_present\_flag[ tileID ][ p ][ m ]  
 if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )  
 TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] = plrd\_mode\_minus1[ tileID ][ p ][ m ]  
 }  
 }  
 }

Otherwise (OverridePlrFlag is equal to 0),

for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 TilePatchPlrdLevel[ tileID ][ p ][ m ] = refPatchPlrdLevel[ m ]  
 if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =  
 refPatchPlrdPresentBlockFlag[ m ][ j ]  
 if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )  
 TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =  
 refPatchPlrdBlockModeMinus1[ m ][ j ]  
 }  
 } else {  
 TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = refPatchPlrdPresentFlag[ m ]  
 if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )  
 TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] = refPlrdModeMinus1[ m ]  
 }  
 }

#### Decoding process for patch data units coded in inter prediction mode

##### General decoding process for patch data units coded in inter prediction mode

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

First, the reference atlas frame index, refIdx, is derived as ipdu\_ref\_index[ tileID ][ p ].

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the ( refIdx + 1 )-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[ refIdx ].

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

Then the corresponding patch index, RefPatchIdx, in the tile with ID equal to tile ID, in the reference atlas frame, refAtlasFrm, is computed as:

RefPatchIdx = PredictorIdx + ipdu\_patch\_index[ tileID ][ p ] (106)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause 9.2.5.5.2 is invoked using the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, and refPatchEomPatchCount, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

It is a requirement of bitstream conformance that refPatchType is equal to PROJECTED.

Then, the associated 2D and 3D patch parameters are derived as follows:

TilePatchType[ tileID ][ p ] = refPatchType (107)

TilePatch2dPosX[ tileID ][ p ] =  
 refPatch2dPosX + ipdu\_2d\_pos\_x[ tileID ][ p ] \* PatchPackingBlockSize (108)

TilePatch2dPosY[ tileID ][ p ] =  
 refPatch2dPosY + ipdu\_2d\_pos\_y[ tileID ][ p ] \* PatchPackingBlockSize (109)

TilePatch2dSizeX[ tileID ][ p ] =  
 refPatch2dSizeX + ipdu\_2d\_delta\_size\_x[ tileID ][ p ] \* PatchSizeXQuantizer (110)

TilePatch2dSizeY[ tileID ][ p ] =  
 refPatch2dSizeY + ipdu\_2d\_delta\_size\_y[ tileID ][ p ] \* PatchSizeYQuantizer (111)

TilePatch3dOffsetU[ tileID ][ p ] = refPatch3dOffsetU + ipdu\_3d\_offset\_u[ tileID ][ p ] (112)

TilePatch3dOffsetV[ tileID ][ p ] = refPatch3dOffsetV + ipdu\_3d\_offset\_v[ tileID ][ p ] (113)

TilePatch3dOffsetD[ tileID ][ p ] = refPatch3dOffsetD + Ipdu3dOffsetD[ tileID ][ p ] (114)

TilePatch3dRangeD[ tileID ][ p ] = refPatch3dRangeD + Ipdu3dRangeD[ tileID ][ p ] (115)

TilePatchProjectionID[ tileID ][ p ] = refPatchProjectionID (116)

TilePatchOrientationIndex[ tileID ][ p ] = refPatchOrientationIndex (117)

TilePatchLoDScaleX[ tileID ][ p ] = refPatchLoDScaleX (118)

TilePatchLoDScaleY[ tileID ][ p ] = refPatchLoDScaleY (119)

If asps\_plr\_enabled\_flag is equal to 1, the following applies:

blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) { (120)  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 TilePatchPlrdLevel[ tileID ][ p ][ m ] = plrd\_level[ tileID ][ p ][ m ]  
 if( TilePatchPlrdLevel[ tileID ][ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] =  
 plrd\_present\_block\_flag[ tileID ][ p ][ m ][ j ]  
 if( TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ] == 1 )  
 TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ] =  
 plrd\_block\_mode\_minus1[ tileID ][ p ][ m ][ j ]  
 }  
 } else {  
 TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] = plrd\_present\_flag[ tileID ][ p ][ m ]  
 if( TilePatchPlrdPresentFlag[ tileID ][ p ][ m ] == 1 )  
 TilePatchPlrdModeMinus1[ tileID ][ p ][ m ] = plrd\_mode\_minus1[ tileID ][ p ][ m ]  
 }  
 }  
 }

##### Derivation of inter reference patch parameters

###### General derivation of inter reference patch parameters

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

Outputs to this process are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, and refPatchEomPatchCount, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

First, the patch, refPatch, is determined that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the ( refIdx + 1 )-th entry of the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[ refIdx ].

Then, the outputs of this process are derived based on the associated parameters of the patch, refPatch as follows:

refPatchType = TilePatchType[ tileID ][ refPatchIdx ] (121)

refPatchInAuxVideo = TilePatchInAuxVideo[ tileID ][ refPatchIdx ] (122)

refPatch2dPosX = TilePatch2dPosX[ tileID ][ refPatchIdx ] (123)

refPatch2dPosY = TilePatch2dPosY[ tileID ][ refPatchIdx ] (124)

refPatch2dSizeX = TilePatch2dSizeX[ tileID ][ refPatchIdx ] (125)

refPatch2dSizeY = TilePatch2dSizeY[ tileID ][ refPatchIdx ] (126)

If refPatchType is equal to PROJECTED, then the process in subclause 9.2.5.5.2.2 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatchProjectionID, refPatch3dOffsetD, refPatch3dRangeD, refPatchOrientationIndex, refPatchLoDScaleX, and refPatchLoDScaleY, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

If refPatchType is equal to RAW, then the process in subclause 9.2.5.5.2.3 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, and refPatchRawPoints.

If refPatchType is equal to EOM, then the process in subclause 9.2.5.5.2.4 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variable refPatchEomPatchCount, and the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints.

###### Reference patches in PROJECTED mode

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

Outputs to this process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, and refPachLoDScaleY, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

First, the patch, refPatch, that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList is determined.

Then, the outputs of this process are derived as follows:

refPatch3dOffsetU = TilePatch3dOffsetU[ tileID ][ refPatchIdx ] (127)

refPatch3dOffsetV = TilePatch3dOffsetV[ tileID ][ refPatchIdx ] (128)

refPatch3dOffsetD = TilePatch3dOffsetD[ tileID ][ refPatchIdx ] (129)

refPatch3dRangeD = TilePatch3dRangeD[ tileID ][ refPatchIdx ] (130)

refPatchProjectionID = TilePatchProjectionID[ tileID ][ refPatchIdx ] (131)

refPatchOrientationIndex = TilePatchOrientationIndex[ tileID ][ refPatchIdx ] (132)

refPatchLoDScaleX = TilePatchLoDScaleX[ tileID ][ refPatchIdx ] (133)

refPatchLoDScaleY = TilePatchLoDScaleY[ tileID ][ refPatchIdx ] (134)

If asps\_plr\_enabled\_flag is equal to 1, the following applies:

blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ refPatchIdx ],  
  TilePatch2dSizeY[ tileID ][ refPatchIdx ] )  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 refPatchPlrdLevel[ m ] = TilePatchPlrdLevel[ tileID ][ refPatchIdx ][ m ]  
 if( refPatchPlrdLevel[ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 refPatchPlrdPresentBlockFlag[ m ][ j ] =  
 TilePatchPlrdPresentBlockFlag[ tileID ][ refPatchIdx ][ m ][ j ] if( refPatchPlrdPresentBlockFlag[ m ][ j ] == 1 )  
 refPatchPlrdBlockModeMinus1[ m ][ j ] =  
 TilePatchPlrdBlockModeMinus1[ tileID ][ refPatchIdx ][ m ][ j ]  
 }  
 } else {  
 refPatchPlrdPresentFlag[ m ] = TilePatchPlrdPresentFlag[ tileID ][ refPatchIdx ][ m ]  
 if( refPatchPlrdPresentFlag[ m ] == 1 )  
 refPlrdModeMinus1[ m ] = TilePatchPlrdModeMinus1[ tileID ][ refPatchIdx ][ m ]  
 }  
 }  
 }

where TilePatch3dOffsetU, TilePatch3dOffsetV, TilePatch3dOffsetD, TilePatch3dRangeD, TilePatchProjectionID TilePatchOrientationIndex, TilePatchLoDScaleX, TilePatchLoDScaleY, TilePatch2dSizeX, TilePatch2dSizeY, TilePatchPlrdLevel, TilePatchPlrdPresentBlockFlag, TilePatchPlrdBlockModeMinus1, TilePatchPlrdPresentFlag, and TilePatchPlrdModeMinus1 are the associated tile patch parameters from the patch with index refPatchIdx in the tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList.

###### Reference Patches in RAW mode

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

Outputs to this process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, and refPatchRawPoints.

First, the patch, refPatch, that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList is determined.

Then, the outputs of this process are derived as follows:

refPatch3dOffsetU = TilePatch3dOffsetU[ tileID ][ refPatchIdx ] (135)

refPatch3dOffsetV = TilePatch3dOffsetV[ tileID ][ refPatchIdx ] (136)

refPatch3dOffsetD = TilePatch3dOffsetD[ tileID ][ refPatchIdx ] (137)

refPatchRawPoints = TilePatchRawPoints[ tileID ][ refPatchIdx ] (138)

where TilePatch3dOffsetU, TilePatch3dOffsetV, TilePatch3dOffsetD, and TilePatchRawPoints are the associated tile patch parameters from the patch with index refPatchIdx in the tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList.

###### Reference Patches in EOM mode

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

Outputs to this process are the variable refPatchEomPatchCount and the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints.

First, the patch, refPatch, that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the ( refIdx + 1 )-th entry of the reference atlas frame list struct, RefAtlasFrmList is determined.

Then, the outputs of this process are derived as follows:

refPatchEomPatchCount = TilePatchEomPatchCount[ tileID ][ refPatchIdx ]  (139)  
 for( i = 0; i < refPatchEomPatchCount; i++ ) {  
 refPatchAssociatedPatchIndex[ i ]  
  = TilePatchAssociatedPatchIndex[ tileID ][ refPatchIdx ][ i ] (140)  
 refPatchEomPoints[ i ] = TilePatchEomPoints[ tileID ][ refPatchIdx ][ i ] (141)  
 }

where TilePatchEomPatchCount, TilePatchAssociatedPatchIndex, and TilePatchEomPoints are the associated tile patch parameters from the patch with index refPatchIdx in the tile ID equal to tileID of the refIdx-th entry of the reference atlas frame list struct, RefAtlasFrmList.

#### Decoding process for patch data units coded in RAW mode

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

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

TilePatchType[ tileID ][ p ] = RAW (142)

TilePatchInAuxVideo[ tileID ][ p ] = rpdu\_patch\_in\_auxiliary\_video\_flag[ tileID ][ p ] (143)

TilePatch2dPosX[ tileID ][ p ] = rpdu\_2d\_pos\_x[ tileID ][ p ] \* PatchPackingBlockSize (144)

TilePatch2dPosY[ tileID ][ p ] = rpdu\_2d\_pos\_y[ tileID ][ p ] \* PatchPackingBlockSize (145)

Then the variables TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ], and TilePatchRawPoints[ tileID ][ p ] are derived as follows:

TilePatch2dSizeX[ tileID ][ p ] =  
 (rpdu\_2d\_size\_x\_minus1[ tileID ][ p ] + 1) \* PatchSizeXQuantizer (146)

TilePatch2dSizeY[ tileID ][ p ] =  
 (rpdu\_2d\_size\_y\_minus1[ tileID ][ p ] + 1) \* PatchSizeYQuantizer (147)

TilePatchRawPoints[ tileID ][ p ] = rpdu\_points\_minus1[ tileID ][ p ] + 1 (148)

Then, the 3D patch variables TilePatch3dOffsetU[ tileID ][ p ], TilePatch3dOffsetV[ tileID ][ p ], and TilePatch3dOffsetD[ tileID ][ p ] are derived and set as output of this process as follows:

TilePatchRawPoints[ tileID ][ refPatchIdx ] = rpdu\_points[ tileID ][ p ] (149)

TilePatch3dOffsetU[ tileID ][ p ] = rpdu\_3d\_offset\_u[ tileID ][ p ] << RawShift (150)

TilePatch3dOffsetV[ tileID ][ p ] = rpdu\_3d\_offset\_v[ tileID ][ p ] << RawShift (151)

TilePatch3dOffsetD[ tileID ][ p ] = rpdu\_3d\_offset\_d[ tileID ][ p ] << RawShift (152)

#### Decoding process for patch data units coded in EOM mode

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

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

TilePatchType[ tileID ][ p ] = EOM (153)

TilePatchInAuxVideo[ tileID ][ p ] = epdu\_patch\_in\_auxiliary\_video\_flag[ tileID ][ p ] (154)

TilePatch2dPosX[ tileID ][ p ] = epdu\_2d\_pos\_x[ tileID ][ p ] \* PatchPackingBlockSize (155)

TilePatch2dPosY[ tileID ][ p ] = epdu\_2d\_pos\_y[ tileID ][ p ] \* PatchPackingBlockSize (156)

TilePatch2dSizeX[ tileID ][ p ] =  
 (epdu\_2d\_size\_x\_minus1[ tileID ][ p ] + 1) \* PatchSizeXQuantizer (157)

TilePatch2dSizeY[ tileID ][ p ] =  
 (epdu\_2d\_size\_y\_minus1[ tileID ][ p ] + 1) \* PatchSizeYQuantizer (158)

TilePatchEomPatchCount[ tileID ][ p ] = ( epdu\_patch\_count\_minus1[ tileID ][ p ] + 1 ) (159)

for( i = 0; i < TilePatchEomPatchCount[ tileID ][ p ]; i++ ) {

TilePatchAssociatedPatchIndex[ tileID ][ p ][ i ] =  
 epdu\_associated\_patch\_idx[ tileID ][ p ] (160)

TilePatchEomPoints[ tileID ][ p ][ i ] = epdu\_points[ tileID ][ p ] (161)

}

### Decoding process of the block to patch map

Input to this process is the current tile ID, tileID.

Then the variables TileBlockToPatchMapWidth[ tileID ] and TileBlockToPatchMapHeight[ tileID ], and the two-dimensional array TileBlockToPatchMap[ tileID ] are computed as follows:

offset = PatchPatchPackingSize – 1

TileBlockToPatchMapWidth[ tileID ] =  
 ( TileWidth[ TileIDToIndex[ tileID ] ] + offset ) / PatchPackingBlockSize (162)

TileBlockToPatchMapHeight[ tileID ]=  
 ( TileHeight[ TileIDToIndex[ tileID ] ] + offset ) / PatchPackingBlockSize  (163)

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

for( y = 0; y < TileBlockToPatchMapHeight[ tileID ]; y++ )  
 for( x = 0; x < TileBlockToPatchMapWidth[ tileID ]; x++ )  
 TileBlockToPatchMap[ tileID ][ y ][ x ] = −1

Then the TileBlockToPatchMap array is updated as follows:

for( p = 0; p < AtduTotalNumPatches[tileID ]; p++ ) {  
 patchType = TilePatchType[tileID ][ p ]  
 if( patchType == PROJECTED ) {  
 xOrg = TilePatch2dPosX[tileID ][ p ] / PatchPackingBlockSize  
 yOrg = TilePatch2dPosY[tileID ][ p ] / PatchPackingBlockSize

tilePatchWidthBlk = TilePatch2dSizeX[tileID ][ p ] / PatchPackingBlockSize  
 tilePatchHeightBlk = TilePatch2dSizeY[tileID ][ p ] / PatchPackingBlockSize  
 for( y = 0; y < tilePatchHeightBlk ; y++)  
 for( x = 0; x < tilePatchWidthBlk; x++ ) {

if(( asps\_patch\_precedence\_order\_flag == 0 ) ||  
 ( TileBlockToPatchMap[ tileID ][ yOrg + y ][ xOrg + x ] == −1 ) )  
 TileBlockToPatchMap[ tileID ][ yOrg + y ][ xOrg + x ] = p  
 }  
 }  
 }  
 else if(TilePatchInAuxVideo[ tileID ][ p ] == 0 ) {  
 xOrg = TilePatch2dPosX[tileID ][ p ] / PatchPackingBlockSize  
 yOrg = TilePatch2dPosY[tileID ][ p ] / PatchPackingBlockSize  
 tilePatchWidthBlk = TilePatch2dSizeX[tileID ][ p ] / PatchPackingBlockSize  
 tilePatchHeightBlk = TilePatch2dSizeY[tileID ][ p ] / PatchPackingBlockSize  
 for( y = 0; y < tilePatchHeightBlk ; y++)  
 for( x = 0; x < tilePatchWidthBlk; x++ )  
 TileBlockToPatchMap[ tileID ][ yOrg + y ][ xOrg + x ] = p  
 }   
 }

### Conversion of tile level patch information to atlas level patch information

#### General

In some implementations it is possible that the reconstruction process may be performed using tile information directly, i.e. the reconstruction process is tile based. In other implementations the reconstruction process may be performed using all tile information, i.e. is atlas based. In that particular case, a conversion of tile level patch information to atlas level patch information may be necessary. This conversion is performed in this subclause.

#### Conversion of tile level blockToPatch information to atlas level blockToPatch information

In this subclause, the block to patch maps from each tile are combined into a single two-dimensional array, AtlasBlockToPatchMap. This array and the variables AtlasBlockToPatchMapWidth, and AtlasBlockToPatchMapHeight are computed as follows:

offset = PatchPatchPackingSize – 1

AtlasBlockToPatchMapWidth = ( asps\_frame\_width + offset ) / PatchPackingBlockSize (164)

AtlasBlockToPatchMapHeight = ( asps\_frame\_height + offset ) / PatchPackingBlockSize (165)

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

for( y = 0; y < AtlasBlockToPatchMapHeight; y++ )  
 for( x = 0; x < AtlasBlockToPatchMapWidth; x++ )  
 AtlasBlockToPatchMap[ y ][ x ] = −1

Then the AtlasBlockToPatchMap array is updated as follows:

offsetPatch = 0  
 for( t = 0; t <= afti\_num\_tiles\_in\_atlas\_frame\_minus1; t++ ) {  
 tileID = TileIndexToID[ t ]  
 tileOffsetX = TileOffsetX[ t ]  
 tileOffsetY = TileOffsetY[ t ]  
 for( blkY = 0; blkY < TileBlockToPatchMapHeight[ tileID ]; blkY++ ) {  
 for( blkX = 0; blkX < TileBlockToPatchMapWidth[ tileID ]; blkX++ ) {  
 y = tileOffsetY / PatchPackingBlockSize + blkY  
 x = tileOffsetX / PatchPackingBlockSize + blkX  
 offset = ( TileBlockToPatchMap[ tileID ][ blkY ][ blkX ] == −1 ) ? 0 : offsetPatch  
 AtlasBlockToPatchMap[ y ][ x ] = TileBlockToPatchMap[ tileID ][ blkY ][ blkX ] + offset }  
 }  
 offsetPatch += AtduTotalNumPatches[ tileID]  
 }  
 AtlasTotalNumPatches = offsetPatch

#### Conversion of tile level patch information to atlas level patch information

##### General

In this subclause tile level patch information is converted to atlas level patch information, using also the processes CommonTilePatchParamsToAtlas(tileID, p) and ApplicationTilePatchParamsToAtlas(tileID, p) defined in subclauses 9.2.7.3.2 and 9.2.7.3.3, respectively, as follows:

NumRawPoints = 0  
 NumEOMPoints = 0  
 AtlasTotalNumPatches = 0  
 AtlasTotalNumProjPatches = 0  
 AtlasTotalNumEOMPatches = 0  
 AtlasTotalNumRawPatches = 0  
 atlasPatchIdx = 0

offsetPatch = 0  
  
 for( t = 0; t <= afti\_num\_tiles\_in\_atlas\_frame\_minus1; t++ ) {  
 tileID = TileIndexToID[ t ]

tileOffsetX = TilePatchInAuxVideo[ tileID ][ p ] ? 0 : TileOffsetX[ t ]  
 tileOffsetY = TilePatchInAuxVideo[ tileID ][ p ] ? AuxTileOffset[ t ] : TileOffsetY[ t ]  
 for( p = 0; p < AtduTotalNumPatches[tileID ]; p++ ) {  
 CommonTilePatchParamsToAtlas( atlasPatchIdx, tileID, p)  
 ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch )  
 atlasPatchIdx += 1  
 }  
 offsetPatch += AtduTotalNumPatches[ tileID]  
 }  
 AtlasTotalNumPatches = atlasPatchIdx

##### Process of copying common patch parameters from a tile to an atlas representation

The process CommonTilePatchParamsToAtlas( atlasPatchIdx, tileID, p ) is defined as follows:

CommonTilePatchParamsToAtlas( atlasPatchIdx, tileID, p ) {  
 AtlasPatchInAuxVideo[ atlasPatchIdx ] = TilePatchInAuxVideo[ tileID ][ p ]  
 AtlasPatchType[ atlasPatchIdx ] = TilePatchType[ tileID ][ p ]  
 AtlasPatch2dSizeX[ atlasPatchIdx ] = TilePatch2dSizeX[ tileID ][ p ]  
 AtlasPatch2dSizeY[ atlasPatchIdx ] = TilePatch2dSizeY[ tileID ][ p ]  
 AtlasPatch2dPosX[ atlasPatchIdx ] = TilePatch2dPosX[ tileID ][ p ] + tileOffsetX  
 AtlasPatch2dPosY[ atlasPatchIdx ] = TilePatch2dPosY[ tileID ][ p ] + tileOffsetY  
 AtlasPatch3dOffsetU[ atlasPatchIdx ] = TilePatch3dOffsetU[ tileID ][ p ]  
 AtlasPatch3dOffsetV[ atlasPatchIdx ] = TilePatch3dOffsetV[ tileID ][ p ]  
 AtlasPatch3dOffsetD[ atlasPatchIdx ] = TilePatch3dOffsetD[ tileID ][ p ]  
 AtlasPatch3dRangeD[ atlasPatchIdx ] = TilePatch3dRangeD[ tileID ][ p ]  
 AtlasPatchProjectionID[ atlasPatchIdx ] = TilePatchProjectionID[ tileID ][ p ]  
 AtlasPatchOrientationIndex[ atlasPatchIdx ] = TilePatchOrientationIndex[ tileID ][ p ]  
 AtlasPatchLoDScaleX[ atlasPatchIdx ] = TilePatchLoDScaleX[ tileID ][ p ]  
 AtlasPatchLoDScaleY[ atlasPatchIdx ] = TilePatchLoDScaleY[ tileID ][ p ]  
 AtlasPatchRawPoints[ atlasPatchIdx ] = TilePatchRawPoints[ tileID ][ p ]  
 AtlasEpduPatchCount[ atlasPatchIdx ] = TilePatchEomPatchCount[ tileID ][ p ]  
}

##### Process of copying application specific patch parameters from a tile to an atlas representation

The process ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch ) is defined as follows:

ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch ) {  
 if( AtlasPatchType[ atlasPatchIdx ] == PROJECTED ) {  
 AtlasTotalNumProjPatches += 1 blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 AtlasPlrdLevel[ atlasPatchIdx ][ m ] = TilePatchPlrdLevel[ tileID ][ p ][ m ]  
 if( AtlasPlrdLevel[ atlasPatchIdx ][ m ] == 0 ) {   
 for( j = 0; j < blockCnt; j++ ) {  
 AtlasPlrdPresentBlockFlag[ atlasPatchIdx ][ m ][ j ] =  
 TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ]  
 if( AtlasPlrdPresentBlockFlag[ atlasPatchIdx ][ m ][ j ] == 1 )  
 AtlasPlrdBlockModeMinus1[ atlasPatchIdx ][ m ][ j ] =  
 TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ]  
 }  
 } else {  
 AtlasPlrdPresentFlag[ atlasPatchIdx ][ m ] =  
 TilePatchPlrdPresentFlag[ tileID ][ p ][ m ]  
 if( AtlasPlrdPresentFlag[ atlasPatchIdx ][ m ] == 1 )  
 AtlasPlrdModeMinus1[ atlasPatchIdx ][ m ] =  
 TilePatchPlrdModeMinus1[ tileID ][ p ][ m ]  
 }  
 }  
 }  
 }  
 if( AtlasPatchType[ atlasPatchIdx ] == EOM ) {  
 AtlasTotalNumEOMPatches += 1  
 for( i = 0; i < AtlasEpduPatchCount[ atlasPatchIdx ]; i++ ) {  
 NumEOMPoints += TilePatchEomPoints[ tileID ][ p ][ i ]  
 AtlasEpduPoints[ atlasPatchIdx ][ i ] = TilePatchEomPoints[ tileID ][ p ][ i ]  
 AtlasEpduAssociatedPatchIndex[ atlasPatchIdx ][ i ] =  
 TilePatchAssociatedPatchIndex[ tileID ][ p ][ i ] + offsetPatch  
 }  
 }  
 if( AtlasPatchType[ atlasPatchIdx ] == RAW ) {  
 AtlasTotalNumRawPatches += 1  
 NumRawPoints += TilePatchRawPoints[ tileID ][ p ]  
 }  
}

## Occupancy video decoding process

The decoding process of an occupancy video component associated with the atlas, with atlas ID DecAtlasID, is performed as follows.

For the occupancy video component, the codec is first determined using either the profiles defined in Annex A or the value of oi\_occupancy\_codec\_id[ DecAtlasID ] and a component codec mapping SEI message specified in subclause F.2.11, if present. Then, the occupancy video decoding process, according to the corresponding coding specification, is invoked using the occupancy video sub-bitstreams present in the V3C bitstream as the input.

Outputs of this process are:

* NumDecOccFrames, indicating the number of decoded occupancy video frames,
* a 4D array DecOccFrames, the decoded occupancy video frames, where the dimensions correspond to the decoded occupancy video frame index, the component index, the row index, and the column index, respectively, and
* the following 1D arrays:
* DecOccBitDepth, indicating the occupancy video bit depth,
* DecOccHeight, indicating the occupancy video height,
* DecOccWidth, indicating the occupancy video width ,
* DecOccChromaFormat, indicating the attribute chroma format,
* DecOccChromaSamplingPosition, indicating, if present, the video chroma sampling position as specified in ISO/IEC 23091-2,
* DecOccFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,
* DecOccColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,
* DecOccTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,
* DecOccMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,
* DecOccOutOrdIdx, indicating the occupancy video output order index, and
* DecOccCompTime, indicating the occupancy video composition time.

where the dimension corresponds to the decoded occupancy video frame index.

If the array DecOccFullRange is missing, all of its elements shall be set to 1.

If any of the elements of the array DecOccTransferCharacteristics are missing or are set to a value of 2, i.e. unspecified, those elements shall be set to 8, i.e. linear.

If the array DecOccChromaSamplingPosition is missing, all of its elements shall be set to 0.

If the array DecOccColourPrimaries is missing, all of its elements shall be set to 2.

If the array DecOccMatrixCoeffs is missing, all of its elements shall be set to 2.

The values of the arrays DecOccChromaSamplingPosition, DecOccColourPrimaries, and DecOccMatrixCoeffs, shall not be used for any further processing of the decoded occupancy frames.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

NOTE – Any existing video coding specification such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video coding specification can be used if included in oi\_occupancy\_codec\_id.

## Geometry video decoding process

The decoding process of a geometry video component associated with the atlas, with atlas ID DecAtlasID, is performed as follows.

For a geometry video component associated with syntax elements vuh\_map\_index, and vuh\_auxiliary\_video\_flag, the codec is first determined using either the profiles defined in Annex A or the value of gi\_geometry\_codec\_id[ DecAtlasID ] and gi\_geometry\_auxiliary\_codec\_id[ DecAtlasID ], if vuh\_auxiliary\_video\_flag is equal to 0 or 1, respectively, and the component codec mapping SEI specified in subclause F.2.11, if present. Then, the geometry video decoding process, according to the corresponding coding specification, is invoked using the geometry video sub-bitstreams present in the V3C bitstream as the input.

If the geometry video sub-bitstream is not an auxiliary geometry video, either identified by vuh\_auxiliary\_video\_flag equal to 0 or determined through external means if the V3C unit header is unavailable, the outputs of this process are:

* the 1D array NumDecGeoFrames, indicating the number of decoded geometry video frames, where the dimension corresponds to vuh\_map\_index ,
* the 5D array DecGeoFrames, the decoded geometry video frames, where the dimensions correspond to vuh\_map\_index, the decoded geometry video frame index, the component index, the row index, and the column index, respectively,
* the following 2D arrays:
* DecGeoBitDepth, indicating the geometry video bit depth,
* DecGeoHeight, indicating the geometry video height,
* DecGeoWidth, indicating the geometry video width ,
* DecGeoChromaFormat, indicating the attribute chroma format,
* DecGeoChromaSamplingPosition, indicating, if present, the geometry chroma sampling as specified in ISO/IEC 23091-2,
* DecGeoFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,
* DecGeoColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,
* DecGeoTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,
* DecGeoMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,
* DecGeoOutOrdIdx, indicating the geometry video output order index, and
* DecGeoCompTime, indicating the geometry video composition time,

where the dimensions correspond to vuh\_map\_index and the decoded geometry video frame index, respectively.

If the array DecGeoFullRange is missing, all of its elements shall be set to 1.

If any of the elements of the array DecGeoTransferCharacteristics are missing or are set to a value of 2, i.e. unspecified, those elements shall be set to 8, i.e. linear.

If the array DecGeoChromaSamplingPosition is missing, all of its elements shall be set to 0.

If the array DecGeoColourPrimaries is missing, all of its elements shall be set to 2.

If the array DecGeoMatrixCoeffs is missing, all of its elements shall be set to 2.

The values of the arrays DecGeoChromaSamplingPosition, DecGeoColourPrimaries, and DecGeoMatrixCoeffs, shall not be used for any further processing of the decoded geometry frames.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

Otherwise, if the geometry video sub-bitstream is an auxiliary geometry video either identified by vuh\_auxiliary\_video\_flag equal to 1 or determined through external means if the V3C unit header is unavailable, the outputs of this process are:

* NumDecGeoAuxFrames, indicating the number of decoded auxiliary geometry video frames,
* the 4D array DecGeoAuxFrames, the decoded auxiliary geometry video frames, where the dimensions correspond to the decoded auxiliary geometry video frame index, the component index, the row index, and the column index, respectively, and
* the following 1D arrays:
* DecGeoAuxBitDepth, indicating the auxiliary geometry video bit depth,
* DecGeoAuxHeight, indicating the auxiliary geometry video height,
* DecGeoAuxWidth, indicating the auxiliary geometry video width ,
* DecGeoAuxChromaFormat, indicating the attribute chroma format,
* DecGeoAuxChromaSamplingPosition, indicating, if present, the geometry chroma sampling as specified in ISO/IEC 23091-2,
* DecGeoAuxFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,
* DecGeoAuxColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,
* DecGeoAuxTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,
* DecGeoAuxMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,
* DecGeoAuxOutOrdIdx, indicating the auxiliary geometry video output order index, and
* DecGeoAuxCompTime, indicating the auxiliary geometry video composition time,

where the dimension corresponds to the decoded auxiliary geometry video frame index.

If the array DecGeoAuxFullRange is missing, all of its elements shall be set to 1.

If any of the elements of the array DecGeoAuxTransferCharacteristics are missing or are set to a value of 2, i.e. unspecified, those elements shall be set to 8, i.e. linear.

If the array DecGeoAuxChromaSamplingPosition is missing, all of its elements shall be set to 0.

If the array DecGeoAuxColourPrimaries is missing, all of its elements shall be set to 2.

If the array DecGeoAuxMatrixCoeffs is missing, all of its elements shall be set to 2.

The values of the arrays DecGeoAuxChromaSamplingPosition, DecGeoAuxColourPrimaries, and DecGeoAuxMatrixCoeffs, shall not be used for any further processing of the decoded auxiliary geometry frames.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

NOTE – Any existing video coding specification such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video coding specification can be used if included in gi\_geometry\_codec\_id.

## Attribute video decoding process

The decoding process of an attribute video component associated with the atlas with atlas ID DecAtlasID, is performed as follows:

* If ai\_attribute\_count[ DecAtlasID ] is equal to 0, no attribute video frames are decoded and no attribute information is associated with the final, reconstructed volumetric frame.
* Otherwise (if ai\_attribute\_count[ DecAtlasID ] is not equal to 0), and for each attribute with index attrIdx, for the attribute video component associated with the syntax elements vuh\_map\_index, vuh\_attribute\_index equal to attrIdx, vuh\_attribute\_partition\_index, and vuh\_auxiliary\_video\_flag, the codec is first determined using either the profiles defined in Annex A or the value of ai\_attribute\_codec\_id[ DecAtlasID ][ attrIdx ] or ai\_attribute\_auxiliary\_codec\_id[ DecAtlasID ][ attrIdx ], if vuh\_auxiliary\_video\_flag is equal to 0 or 1, respectively and the component codec mapping SEI specified in subclause F.2.11, if present. Then, the attribute video decoding process, according to the corresponding coding specification, is invoked using the attribute video sub-bitstreams present in the V3C bitstream as the input.

If the attribute video sub-bitstream is not an auxiliary attribute video, either identified by vuh\_auxiliary\_video\_flag equal to 0 or determined through external means if the V3C unit header is unavailable, the outputs of this process are:

* the 1D array DecAttrType, indicating the attribute type, where the dimension corresponds to the attribute index,
* the 3D array NumDecAttrFrames, indicating the number of decoded attribute video frames, where the dimensions correspond to the attribute index, the attribute partition index, and vuh\_map\_index,
* the 7D array DecAttrFrames, the decoded attribute video frames, where the dimensions correspond to the attribute index, the attribute partition index, vuh\_map\_index, the decoded attribute video frame index, the component index, the row index, and the column index, respectively, and
* the following 4D arrays:
* DecAttrBitDepth, indicating the attribute bit depth,
* DecAttrHeight, indicating the attribute height,
* DecAttrWidth, indicating the attribute width,
* DecAttrNumComp, indicating the attribute number of components,
* DecAttrChromaFormat, indicating the attribute chroma format,
* DecAttrChromaSamplingPosition, indicating, if present, the attribute chroma sampling as specified in ISO/IEC 23091-2,
* DecAttrFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,
* DecAttrColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,
* DecAttrTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,
* DecAttrMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,
* DecAttrOutOrdIdx, indicating the attribute output order index, and
* DecAttrCompTime, indicating the attribute composition time,

where the dimensions correspond to the attribute index, the attribute partition index, vuh\_map\_index, and the decoded attribute frame index, respectively.

If the array DecAttrChromaSamplingPosition, is missing, all of its elements shall be set to 0.

If the array DecAttrFullRange, is missing, all of its elements shall be set to 1.

If the array DecAttrColourPrimaries, is missing, all of its elements shall be set to 1.

If the array DecAttrTransferCharacteristics, is missing, all of its elements shall be set to 1.

If the array DecAttrMatrixCoeffs, is missing, all of its elements shall be set to 1.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

Otherwise, if the attribute video sub-bitstream is an auxiliary attribute video, either identified by vuh\_auxiliary\_video\_flag equal to 1 or determined through external means if the V3C unit header is unavailable, the outputs of this process are:

* the 1D array DecAttrAuxType, indicating the attribute type, where the dimension corresponds to the attribute index,
* the 2D array NumDecAttrAuxFrames, indicating the number of decoded auxiliary attribute video frames, where the dimensions correspond to the attribute index and the attribute partition index,
* the 6D array DecAttrAuxFrames, the decoded auxiliary attribute video frames, where the dimensions correspond to the attribute index, the attribute partition index, the decoded attribute video frame index, the component index, the row index, and the column index, respectively, and
* the following 3D arrays:
  + DecAttrAuxBitDepth, indicating the auxiliary attribute bit depth,
  + DecAttrAuxHeight, indicating the auxiliary attribute height,
  + DecAttrAuxWidth, indicating the auxiliary attribute width,
  + DecAttrAuxNumComp, indicating the attribute number of components,
  + DecAttrAuxChromaFormat, indicating the auxiliary attribute chroma format,
  + DecAttrAuxChromaSamplingPosition, indicating, if present, the attribute chroma sampling as specified in ISO/IEC 23091-2,
  + DecAttrAuxFullRange, indicating, if present, the video full range code point as specified in ISO/IEC 23091-2,
  + DecAttrAuxColourPrimaries, indicating, if present, the chromaticity coordinates of the source primaries as specified in ISO/IEC 23091-2,
  + DecAttrAuxTransferCharacteristics, indicating, if present, the transfer characteristics as specified in ISO/IEC 23091-2,
  + DecAttrAuxMatrixCoeffs, indicating, if present, the matrix coefficients as specified in ISO/IEC 23091-2,
  + DecAttrAuxOutOrdIdx, indicating the auxiliary attribute output order index, and
  + DecAttrAuxCompTime, indicating the auxiliary attribute composition time,

where the dimensions correspond to the attribute index, the attribute partition index, and the decoded attribute frame index, respectively.

If the array DecAttrAuxChromaSamplingPosition, is missing, all of its elements shall be set to 0.

If the array DecAttrAuxFullRange, is missing, all of its elements shall be set to 1.

If the array DecAttrAuxColourPrimaries, is missing, all of its elements shall be set to 1.

If the array DecAttrAuxTransferCharacteristics, is missing, all of its elements shall be set to 1.

If the array DecAttrAuxMatrixCoeffs, is missing, all of its elements shall be set to 1.

These values shall be interpreted according to their corresponding coding points in ISO/IEC 23091-2.

NOTE – Any existing video coding specification such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video coding specification can be used if included in ai\_attribute\_codec\_id.

## Packed video decoding process

[Ed.(BK): This clause is copied verbatim from ISO/IEC CD 23090-12 and needs to be aligned to the format used in previous sections.]

The codec for the packed video component associated with the current atlas, with atlas index atlasIdx, is first determined using pin\_pack\_codec\_id[ atlasIdx ] and the codec mapping SEI message, if present. Then, the packed video decoding process, according to the corresponding codec specification, is invoked using the packed video sub-bitstreams present in the V3C bitstream as the input.

Outputs of this process are:

* the decoded packed video frames, decPackedFrame[ frameIdx ][ compIdx ][ y ][ x ], and
* for each decoded packed video frame, its corresponding coded representation parameters:
  + bit depth, decPackedBitDepth[ frameIdx ],
  + width, decPackedWidth[ frameIdx ],
  + height, decPackedHeight[ frameIdx ],
  + output order index, decPackedOutOrdIdx[ frameIdx ], and
  + composition time, decPackedCompTime[ frameIdx ],

where frameIdx is the decoder output index of the decoded packed video frames, compIdx corresponds to the colour component index and is in the range of 0 to 2, y is the row index in the decoded frame and is in the range of 0 to decPackedHeight[ frameIdx ] − 1, inclusive, and x is the column index in the decoded frame and is in the range of 0 to decPackedWidth[ frameIdx ] − 1, inclusive.

NOTE – Any existing video codec such as ISO/IEC 14496-10 or ISO/IEC 23008-2 or any future defined video codec can be used if included in pin\_packed\_codec\_id.

## Common atlas data decoding process

Decoding of common atlas data is outside the scope of this version of this document.

NOTE 1: Common atlas data (V3C\_CAD) and common atlas frame (NAL\_CAF) form an extension mechanism to share information (e.g. projection information) that applies to multiple atlases.

## Sub-bitstream extraction process

### General

In order to perform the decoding processes identified in subclauses 9.2, 9.3, 9.4, and 9.5, it may be necessary to extract the appropriate V3C units, e.g. atlas or occupancy data, from the V3C bitstream, or ACL and non-ACL NAL units of specific type from an atlas sub-bitstream. Subclause 9.8.2 describes the extraction of V3C units. Subclause 9.8.3 describes the extraction of NAL units.

### V3C unit extraction

Inputs to this process are a V3C bitstream, such as a bitstream in the V3C sample stream format defined in Annex C, a target atlas identifier, targetAtlasId, and a set of parameters specifying the component or components to be extracted.

First the locations of the V3C unit boundaries in a V3C bitstream are identified. For example, for the case of the V3C sample stream format specified in Annex C, the location of each boundary can be identified using the syntax element ssvu\_v3c\_unit\_size.

Then, the extraction process for each V3C unit is performed as follows:

If vuh\_unit\_type in the V3C unit headers is equal to V3C\_AVD, V3C\_GVD, V3C\_OVD, or V3C\_AD, and the vuh\_atlas\_id is equal to targetAtlasId, then the following applies based on the input extraction parameters:

* If extraction of atlas sub-bitstreams is indicated, then if vuh\_unit\_type is equal to V3C\_AD in the current V3C unit, then the V3C unit payload is extracted as an output.
* If extraction of occupancy sub-bitstreams is indicated, then if vuh\_unit\_type is equal to V3C\_OVD in the current V3C unit, then the V3C unit payload is extracted as an output.
* If extraction of geometry sub-bitstreams with map index mapIdx is indicated, then if vuh\_unit\_type is equal to V3C\_GVD, vuh\_map\_index is equal to mapIdx, and vuh\_auxiliary\_video\_flag is equal to 0 in the current V3C unit, then the V3C unit payload is extracted as an output.
* If extraction of auxiliary geometry sub-bitstreams is indicated, then if vuh\_unit\_type is equal to V3C\_GVD, and vuh\_auxiliary\_video\_flag is equal to 1 in the current V3C unit, then the V3C unit payload is extracted as an output.
* If extraction of attribute sub-bitstreams with attribute index attrIdx, map index mapIdx, and partition index partIdx, is indicated, then if vuh\_unit\_type is equal to V3C\_AVD, vuh\_attribute\_index is equal to attrIdx, vuh\_map\_index is equal to mapIdx, vuh\_attribute\_partition\_index is equal to partIdx, and vuh\_auxiliary\_video\_flag is equal to 0 in the current V3C unit, then the V3C unit payload is extracted as an output.
* If extraction of auxiliary attribute sub-bitstreams with attribute index attrIdx and partition index partIdx, is indicated, then if vuh\_unit\_type is equal to V3C\_AVD, vuh\_attribute\_index is equal to attrIdx, vuh\_attribute\_partition\_index is equal to partIdx, and vuh\_auxiliary\_video\_flag is equal to 1 in the current V3C unit, then the V3C unit payload is extracted as an output.

It is a requirement of bitstream conformance for the input bitstream that any output sub-bitstream that is the output of the process specified in this clause shall be a conforming bitstream according to the codec specification identified by either the CodecGroup profile component or, if present, the corresponding codec indicated by the component codec mapping SEI message.

### NAL unit extraction process

Inputs to this process are an atlas sub-bitstream, such as a bitstream in the NAL sample stream format defined in Annex D, and a set of parameters specifying whether it is an ACL or non-ACL NAL unit and when it is an ACL NAL unit, optionally, the tile ID to be extracted.

First the locations of the NAL unit boundaries in a NAL bitstream are identified. For example, for the case of the NAL sample stream format specified in Annex D, the location of each boundary can be identified using the syntax element ssnu\_nal\_unit\_size.

Then, the extraction process for each NAL unit is performed as follows:

* If extraction of ACL NAL units with a specific tile ID, tileID, is indicated, then all ACL NAL units with ath\_id equal to tileID and their associated non-ACL NAL units are extracted.
* If extraction of ACL NAL units is indicated without specifying a specific tile ID, then all ACL NAL units and their associated non-ACL NAL units are extracted.
* If extraction of a non-ACL NAL unit of a specific NAL type, nalType, is indicated, then all NAL units with nal\_unit\_type equal to nalType, and their associated non-ACL NAL units are extracted.

# Pre-reconstruction process

The pre-reconstruction process depends on the profiles defined in Annex A.

Some applications within the scope of this document define explicit pre-reconstruction processing steps. However, some other applications may not, in which case the pre-reconstruction process is outside the scope of this document. For example, in the context of the video-based point cloud decoding applications, as defined in Annex H, and the associated profiles, a pre-reconstruction process, which performs occupancy synthesis operations, is specified in subclause H.10.

# Reconstruction process

The reconstruction process depends on the profiles defined in Annex A.

Some applications within the scope of this document define explicit reconstruction processing steps. However, some other applications may not, in which case the reconstruction process is outside the scope of this document. For example, in the context of the video-based point cloud decoding applications, as defined in Annex H, and the associated profiles, a nominal reconstruction process is specified in subclause H.11.

# Post-reconstruction process

The post-reconstruction process depends on the profiles defined in Annex A.

Some applications within the scope of this document define explicit post-reconstruction processing steps. However, some other applications may not, in which case the post-reconstruction process is outside the scope of this document. For example, in the context of the video-based point cloud decoding applications, as defined in Annex H, and the associated profiles, a post-reconstruction process, which performs a number of smoothing operations on the reconstructed point cloud data, is specified in subclause H.12.

# Adaptation process

The adaptation process depends on the profiles defined in Annex A

Some applications within the scope of this document define explicit adaptation process steps. However, some other applications may not, in which case the adaptation process is outside the scope of this document. For example, in the context of the video-based point cloud decoding applications, as defined in Annex H, and the associated profiles, an adaptation process is specified in subclause H.13.

# Parsing process

## General

Inputs to this process are bits from the bitstream

Outputs of this process are syntax element values.

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v) or se(v) (see subclause 14.2).

## Parsing process for 0-th order Exp-Golomb codes

### General

This process is invoked when the descriptor of a syntax element in the syntax tables is equal to ue(v) or se(v).

Inputs to this process are bits from the bitstream.

Outputs of this process are syntax element values.

Syntax elements coded as ue(v) or se(v) are Exp-Golomb-coded. The parsing process for these syntax elements begins with reading the bits starting at the current location in the bitstream up to and including the first non-zero bit and counting the number of leading bits that are equal to 0. This process is specified as follows:

leadingZeroBits = −1  
 for( b = 0; !b; leadingZeroBits++ ) (166)  
 b = read\_bits( 1 )

The variable codeNum is then assigned as follows:

codeNum = 2leadingZeroBits – 1 + read\_bits( leadingZeroBits ) (167)

where the value returned from read\_bits( leadingZeroBits ) is interpreted as a binary representation of an unsigned integer with the most significant bit written first.

Table 12 illustrates the structure of the Exp-Golomb code by separating the bit string into "prefix" and "suffix" bits. The "prefix" bits are those bits that are parsed as specified above for the computation of leadingZeroBits and are shown as either 0 or 1 in the bit string column of Table 12. The "suffix" bits are those bits that are parsed in the computation of codeNum and are shown as xi in Table 12, with i in the range of 0 to leadingZeroBits − 1, inclusive. Each xi is equal to either 0 or 1.

Table 12 – Bit strings with "prefix" and "suffix" bits and assignment to codeNum ranges

|  |  |
| --- | --- |
| **Bit string form** | **Range of codeNum** |
| 1 | 0 |
| 0 1 x0 | 1..2 |
| 0 0 1 x1 x0 | 3..6 |
| 0 0 0 1 x2 x1 x0 | 7..14 |
| 0 0 0 0 1 x3 x2 x1 x0 | 15..30 |
| 0 0 0 0 0 1 x4 x3 x2 x1 x0 | 31..62 |
| ... | ... |

Table 13 illustrates explicitly the assignment of bit strings to codeNum values.

Table 13 – Exp-Golomb bit strings and codeNum in explicit form and used as ue(v)

|  |  |
| --- | --- |
| **Bit string** | **codeNum** |
| 1 | 0 |
| 0 1 0 | 1 |
| 0 1 1 | 2 |
| 0 0 1 0 0 | 3 |
| 0 0 1 0 1 | 4 |
| 0 0 1 1 0 | 5 |
| 0 0 1 1 1 | 6 |
| 0 0 0 1 0 0 0 | 7 |
| 0 0 0 1 0 0 1 | 8 |
| 0 0 0 1 0 1 0 | 9 |
| ... | ... |

Depending on the descriptor, the value of a syntax element is derived as follows:

* If the syntax element is coded as ue(v), the value of the syntax element is equal to codeNum.
* Otherwise (the syntax element is coded as se(v)), the value of the syntax element is derived by invoking the mapping process for signed Exp-Golomb codes as specified in subclause 14.2.2 with codeNum as input.

### Mapping process for signed Exp-Golomb codes

Input to this process is codeNum as specified in subclause 14.1.

Output of this process is a value of a syntax element coded as se(v).

The syntax element is assigned to the codeNum by ordering the syntax element by its absolute value in increasing order and representing the positive value for a given absolute value with the lower codeNum. Table 14 provides the assignment rule.

Table 14 – Assignment of syntax element to codeNum

|  |  |
| --- | --- |
| **codeNum** | **syntax element value** |
| 0 | 0 |
| 1 | 1 |
| 2 | −1 |
| 3 | 2 |
| 4 | −2 |
| 5 | 3 |
| 6 | −3 |
| k | ( (k + 1) / 2 ) \* (−1)k + 1 |

1. (normative)  
   Profiles, tiers, and levels
   1. Overview of profiles, tiers, and levels

Profiles, tiers, and levels specify restrictions on the bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles, tiers, and levels may also be used to indicate interoperability points between individual decoder implementations.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

Each level of a tier specifies a set of limits on the values that may be taken by the syntax elements of this document. The same set of tier and level definitions is used with all profiles, but individual implementations may support a different tier and within a tier a different level for each supported profile. For any given profile, a level of a tier generally corresponds to a particular decoder processing load and memory capability.

Capabilities of V3C decoders conforming to this version of this document are specified in terms of the ability to decode V3C bitstreams or collection of V3C sub-bitstreams conforming to the constraints of profiles, tiers, and levels specified in this annex and other annexes. When expressing the capabilities of a decoder for a specified profile, the tier and level supported for that profile should also be expressed.

NOTE – The term “decode” in this context can include the reconstruction in 3D space, i.e. “decode” can refer to only decoding the 2D video sub-bitstreams (i.e. geometry, occupancy, attribute(s)) and the associated atlas sub-bitstream, or it can include further reconstruction into 3D space. The respective steps included in the “decoding” are specified for each profile.

In this annex, phrases like "the bitstream" should be intepreted as "the V3C bitstream of the operation point", and phrases like "coded atlas access unit n" should be interpreted as "coded atlas access unit n in the bitstream of the operation point”, where "the operation point" is the operation point with which the profile, tier, or level is associated with.

The variables HtidOP and TemporalID, which indicate the highest temporal layer present in an atlas sub‑bitstream and the temporal ID of an atlas sub-bitstream, respectively, shall be equal to 0 in this version of the document.

For each operation point identified by HtidOP, the profile, tier, and level information are indicated by a combination of syntax elements, including ptl\_profile\_codec\_group\_idc and ptl\_profile\_toolset\_idc, that are found in or derived from the profile\_tier\_level( ) syntax structure in the VPS, if available, or determined through external means.

* 1. Profile, tier, and level structure

V3C profiles follow a structured and flexible definition to allow for clearly identifying two distinct conformance points. These conformance points are illustrated in the decoding block diagram shown in Figure A-1.



Figure A-1 – V3C decoding block diagram with decoding conformance points A and B

The first conformance point, point A, covers the decoded video sub-bitstreams and atlas sub-bitstream. It also covers the derived block to patch map information. It does not, however, cover the reconstruction process.

The second conformance point, point B, covers the reconstruction process.

A V3C profile decoding capability is defined by a combination of syntax elements ptl\_profile\_codec\_group\_idc and ptl\_profile\_toolset\_idc that are present in the VPS or obtained through external means if the VPS is unavailable. A V3C profile reconstruction capability is optionally defined by ptl\_profile\_reconstruction\_idc, as shown in Figure A-2.

The first two profile components together describe conformance point A. More specifically, they describe the supported:

* video decoding specifications and their profiles (e.g., Progressive High as specified in ISO/IEC 14496-10:2014:Annex A, Main or Main 10 as specified in ISO/IEC 23008-2:2020:Annex A), referred to as the **CodecGroup** profile component, and
* V3C specific tools (e.g., use of EOM and PLR etc), referred to as the **Toolset** profile component. The **Toolset** profile component describes the bitstream syntax structure.

The third profile component describes conformance point B, specifying the pre-reconstruction, reconstruction, post-reconstruction, and adaptation tools supported or recommended to achieve conformance in terms of 3D reconstruction. This is referred to as the **Reconstruction** profile component.

Together, these profile components form a V3C profile, which follows the naming convention **CodecGroup Toolset Reconstruction**. For example, taking the first blocks of Figure A-2, the resulting V3C profile is named as **AVC V-PCC Basic Rec0*.***



Figure A-2 – V3C profile, tier, and level structure

Conformance can be specified for a single V3C bitstream, as defined in Annex C, or for a collection of V3C sub-bitstreams that are associated with each other and provide information for the reconstruction of a sequence of volumetric frames.

A V3C bitstream may consist of one or more atlas sub‑bitstreams, and each atlas may include additional sub-bitstreams, such as an occupancy sub-bitstream, 0 or more geometry map sub-bitstreams, and 0 or more attributes of different type.

Any V3C bitstream, collection of V3C sub-bitstreams, or V3C decoder can claim support of both conformance point A and conformance point B. However, indicating conformance point A is mandatory, while conformance point B is optional. For example:

* **AVC V-PCC Basic**, is a valid profile that clearly indicates its CodecGroup and Toolset profile components while providing no indication of a Reconstruction profile component.
* **AVC V-PCC Basic Rec0**, is a valid profile that clearly indicates its CodecGroup and Toolset profile components while also indicating support of a Reconstruction profile component that provides for elementary 3D reconstruction.
* **AVC Rec0**, is NOT a valid profile as the Toolset information is missing.
* **V-PCC Basic Rec0**, is NOT a valid profile as the CodecGroup information is missing.

To indicate profile conformance of a bitstream or decoder at conformance point A, it is only needed to consider the CodecGroup and Toolset profile components. Decoders conforming to a V3C profile at conformance point A (identified by syntax elements ptl\_profile\_codec\_group\_idc and ptl\_profile\_toolset\_idc) at a specific level (identified by a specific value of syntax element ptl\_level\_idc) of a specific tier (identified by a specific value of syntax element ptl\_tier\_flag) shall be capable of decoding all V3C bitstreams or collection of V3C sub-bitstreams, according to Clause 9, for which all of the following conditions apply:

* The V3C bitstream or the collection of V3C sub-bitstreams are indicated to conform to the supported CodecGroup and Toolset profile components, as indicated in subclauses A.3 (CodecGroup) and A.4 (Toolset).
* The V3C bitstream or the collection of V3C sub-bitstreams are indicated to conform to a level that is lower than or equal to the specified level, as indicated in subclause A.6.
* The V3C bitstream or the collection of V3C sub-bitstreams are indicated to conform to a tier that is lower than or equal to the specified tier, as indicated in subclause A.6.

A decoder may support and may be able to conform to multiple reconstruction profiles as specified in subclause A.5. However, the reconstruction profile a decoder selects to operate in, is outside the scope of this document. When a decoder claims to be operating at a particular conformance point B, it shall be capable of reconstructing a volumetric frame according to the reconstruction recommendations specified by that profile.

* 1. CodecGroup profile components

Table A-1 provides a list of defined CodecGroup profile components for V3C and respectively the allowed values for syntax element ptl\_profile\_codec\_group\_idc.

**Table A-1** – **Available CodecGroup profile components**

|  |  |  |
| --- | --- | --- |
| **CodecGroup** | **ptl\_profile\_codec\_group\_idc** | **4CC code** |
| AVC Progressive High | 0 | 'avc3' |
| HEVC Main10 | 1 | 'hev1' |
| HEVC444 | 2 | 'hev1' |
| VVC Main10 | 3 | 'vvc1' |
| Reserved | 4..126 | – |
| MP4RA | 127 | provided by component codec mapping SEI message (F.2.11) |

Table A-2 provides a list of the supported codec functionalities for each CodecGroup profile component.

**Table A-2** – **CodecGroup profile component supported functionality**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | AVC Progressive High | | | HEVC Main10 | | | HEVC444 | | | VVC Main 10 | | | MP4RA | | |
|  | Capability | Occupancy | Geometry | Attributes | Occupancy | Geometry | Attributes | Occupancy | Geometry | Attributes | Occupancy | Geometry | Attributes | Occupancy | Geometry | Attributes |
| Chroma format | Mono |  |  |  |  |  |  | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ‒ | ‒ | ‒ |
| 4:2:0 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ‒ | ‒ | ‒ |
| 4:4:4 |  |  |  |  |  |  |  |  | ✓ |  |  |  | ‒ | ‒ | ‒ |
| Bit depth | 8 bit | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ‒ | ‒ | ‒ |
| 10 bit |  |  |  | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ‒ | ‒ | ‒ |
| Video coding specification | name | ISO/IEC 14496-10 | | | ISO/IEC 23008-2 | | | | | | ISO/IEC DIS 23090-3[[4]](#footnote-5) | | | As defined by a component codec mapping SEI (F.2.11) | | |
| profile | Progressive High as specified in ISO/IEC 14496‑10 | | | Main 10 as specified in ISO/IEC 23008‑2 | | | Main 4:4:4 10 as specified in ISO/IEC 23008‑2 | | | VVC Main 10 as specified in ISO/IEC DIS 23090‑3 | | | ‒ | | |

A tick mark in the Table A-2 indicates that a particular feature is supported by the defined profile. A dash symbol, i.e. "‒", indicates that support of a particular feature is specified by means outside this document.

All video sub-bitstreams of a V3C bitstream or a collection of V3C sub-bitstreams conform to the stream format indicated in the component codec mapping SEI (F.2.11).

For ptl\_profile\_codec\_group\_idc equal to 0, all video sub-bitstreams shall conform to the stream format identified by 4CC code equal to 'avc3'. The codec ids associated with all video sub-bitstreams in the active VPS are inferred to be mapped to 'avc3'. The component codec mapping SEI message (F.2.11), when present, shall also indicate the value of ‘avc3’ for all instances j of ccm\_codec\_4cc[ j ] present in this SEI message.

For ptl\_profile\_codec\_group\_idc equal to 1 or 2, all video sub-bitstreams shall conform to the stream format identified by 4CC code equal to 'hev1'. The codec ids associated with all video sub-bitstreams in the active VPS are inferred to be mapped to 'hev1'. The component codec mapping SEI message (F.2.11), when present, shall also indicate the value of ‘hev1' for all instances j of ccm\_codec\_4cc[ j ] present in this SEI message.

For ptl\_profile\_codec\_group\_idc equal to 3, all video sub-bitstreams shall conform to the stream format identified by 4CC code equal to 'vvc1'. The codec ids associated with all video sub-bitstreams in the active VPS are inferred to be mapped to 'vvc1'. The component codec mapping SEI message (F.2.11), when present, shall also indicate the value of ‘vvc1' for all instances j of ccm\_codec\_4cc[ j ] present in this SEI message.

For ptl\_profile\_codec\_group\_idc equal to 127, a video sub-bitstream of an atlas with atlas ID equal to atlasID that is associated with a particular codec id codecID, where codecID may correspond to the value of oi\_occupancy\_codec\_id[ atlasID ], gi\_geometry\_codec\_id[ atlasID ], gi\_auxiliary\_geometry\_codec\_id[ atlasID ], ai\_attribute\_codec\_id[ atlasID ][ i ], and ai\_auxiliary\_attribute\_codec\_id[ atlasID ][ i ] depending on the type of the video, i.e. occupancy, geometry, auxiliary geometry, attribute with index i, or auxiliary attribute with index i, shall conform to the stream format indicated by the value of ccm\_codec\_4cc[ codecID ] as defined in the component codec mapping SEI message (F.2.11).

When stream format is identified by a 4CC code defined in ISO/IEC 14496-15, the number of bytes that indicate the NAL unit length, NALUnitLength, defined in ISO/IEC 14496-15 as being equal to the field LengthSizeMinusOne + 1, shall be equal to 4.

The use of the MP4RA CodecGroup profile component is not meant as an interoperability point. Conformance to this profile can be specified by means outside this document.

* 1. Toolset profile components

Table A-3 provides a list of defined toolset profile components for V3C and their corresponding identifying syntax element values, e.g. ptl\_profile\_toolset\_idc and ptc\_one\_v3c\_frame\_only flag, that are defined for use by this document. The syntax element ptl\_profile\_toolset\_idc provides the primary definition of a toolset profile. However, additional syntax elements, such as ptc\_one\_v3c\_frame\_only\_flag, can indicate additional characteristics or restrictions of a defined profile. In the case of ptc\_one\_v3c\_frame\_only\_flag, this syntax element indicates that such a profile can only support a single V3C frame. Reserved values of ptl\_profile\_toolset\_idc are reserved for future use by ISO/IEC and shall not be present in V3C bitstreams conforming to this version of this document. The defined profiles in Table A-3 are also characterized as either dynamic or static, depending on whether they are targeting dynamic or static, i.e. single, volumetric frame applications.

**Table A-3** – **Available toolset profile components**

|  |  |  |  |
| --- | --- | --- | --- |
| **ptl\_profile\_toolset\_idc** | **ptc\_one\_v3c\_frame\_only\_flag** | **Toolset profile component** | **Type** |
| 0 | 0 | V-PCC Basic  /\* Specified in Annex H\*/ | Dynamic |
| 1 | V-PCC Basic Still /\* Specified in Annex H\*/ | Static |
| 1 | 0 | V-PCC Extended /\* Specified in Annex H\*/ | Dynamic |
| 1 | V-PCC Extended Still /\* Specified in Annex H\*/ | Static |
| 64 | 0 | MIV Main /\* Specified in ISO/IEC 23090-12 \*/ | Dynamic |
| 2..255 | - | Reserved | - |

* 1. Reconstruction profile components

Table A-4 provides a list of defined reconstruction profile components for V3C and of the allowed values for syntax element ptl\_profile\_reconstruction\_idc. Values of ptl\_profile\_reconstruction\_idc in the range of 0 to 254, inclusive, may be specified by a V3C application and may depend on the value of ptl\_profile\_toolset\_idc. For example, for the case of ptl\_profile\_toolset\_idc equal to 0 or 1, which indicates a V-PCC application, subclause H.15.5 specifies reconstruction profiles appropriate for that application.

For the Rec Unconstrained reconstruction profile, the reconstruction process is outside the scope of this document.

**Table A-4** – **Defined reconstruction profile components**

|  |  |
| --- | --- |
| **ptl\_profile\_reconstruction\_idc** | **Reconstruction profile component** |
| 0..254 | Application dependent |
| 255 | Rec Unconstrained |

* 1. Tiers and levels
     1. General tier and level limits

For purposes of comparison of tier capabilities, the tier with ptl\_tier\_flag equal to 0 is considered to be a lower tier than the tier with ptl\_tier\_flag equal to 1.

For purposes of comparison of level capabilities, a particular level of a specific tier is considered to be a lower level than some other level of the same tier when the value of the ptl\_level\_idc of the particular level is less than that of the other level.

The following is specified for expressing the constraints in this annex:

* Let atlas coded access unit n be the n-th access unit in decoding order, with the first access unit being access unit 0 (i.e., the 0-th access unit).

Let atlas n be the coded atlas or the corresponding decoded atlas of access unit n.

When a specified level is not level 8.5, V3C bitstreams conforming to all profiles specified in this document at a specified tier and level shall obey the following constraints for each atlas sub-bitstream conformance test as specified in Annex E:

1. AspsFrameSize shall be less than or equal to MaxAtlasSize, where MaxAtlasSize is specified in Table A-6.
2. vps\_map\_count\_minus1 + 1 shall be less than or equal to LevelMapCount, where LevelMapCount is specified in Table A-5.
3. ai\_attribute\_count shall be less than or equal to MaxNumAttributeCount, where MaxNumAttributeCount is specified in Table A-5.
4. ai\_attribute\_dimension\_minus1 + 1 shall be less than or equal to MaxNumAttributeDims, where MaxNumAttributeDims is specified in Table A-5.
5. afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1 shall be less than MaxNumTiles is specified in Table A-6.
6. When the current atlas frame is an IDR coded atlas ath\_atlas\_frm\_order\_cnt\_lsb shall be equal to 0.
   * 1. Profile specific level limits for the V3C dynamic profiles

A tier and level to which V3C bitstream conforms are indicated by the syntax elements ptl\_tier\_flag and ptl\_level\_idc. ptl\_level\_idc shall be set equal to a value of 30 times the level number specified in Table A-5.

Table A-5 – General V3C or VPS related level limits

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Level** | **Max # of PROJECTED points per second**  **MaxNumProjPointsPerSec** | **Max # of EOM points per second**  **MaxNumEOMPointsPerSec** | **Max # of RAW points per second**  **MaxNumRawPointsPerSec** | **Max # of PROJECTED points per atlas**  **MaxNumProjPoints** | **Max # EOM points per atlas**  **MaxNumEOMPoints** | **Max # RAW points per atlas**  **MaxNumRawPoints** | **Max # of maps**  **LevelMapCount** | **Max # of attributes**  **MaxNumAttributeCount** | **Max # of attribute dimensions**  **MaxNumAttributeDims** |
| 1.0 | 30 000 000 | 1 500 000 | 1 500 000 | 1 000 000 | 50 000 | 50 000 | 2 | 1 | 3 |
| 1.5 | 60 000 000 | 3 000 000 | 3 000 000 | 2 000 000 | 100 000 | 100 000 | 2 | 3 | 3 |
| 2.0 | 120 000 000 | 6 000 000 | 6 000 000 | 4 000 000 | 200 000 | 200 000 | 4 | 4 | 3 |
| 2.5 | 240 000 000 | 12 000 000 | 12 000 000 | 8 000 000 | 400 000 | 400 000 | 4 | 8 | 4 |
| 3.0 | 480 000 000 | 24 000 000 | 24 000 000 | 16 000 000 | 800 000 | 800 000 | 8 | 16 | 5 |
| 3.5 | 960 000 000 | 48 000 000 | 48 000 000 | 32 000 000 | 1 600 000 | 1 600 000 | 8 | 24 | 6 |

Table A-5 specifies the general V3C and VPS related limits for each level of each tier.

Let the variable t0 denote the time instance in milliseconds at which the first bit of the coded atlas access unit 0 begins to enter the CAB.

For any time t1 greater than or equal to t0, the variables NumProjPointsPerSec, NumRawPointsPerSec, and NumEOMPointsPerSec, are defined and set equal to the total number of NumProjPoints, NumRawPoints, and NumEOMPoints, respectively, of the volumetric frames that are output for all values of t, with t1 <= t < t1 + 1 000. NumRawPoints and NumEOMPoints are specified in subclause 9.2.7, and NumProjPoints is specified in subclause B.2.2.

For any time t1 greater than or equal to t0, the variables NumProjPatchesPerSec, NumRawPatchesPerSec, and NumEOMPatchesPerSec, are defined and set equal to the total number of AtlasTotalNumProjPatches, AtlasTotalNumRawPatches, and AtlasTotalNumEOMPatches, respectively, of the atlas frames that are decoded for all values of t, with t1 <= t < t1 + 1 000. AtlasTotalNumProjPatches, AtlasTotalNumRawPatches, and AtlasTotalNumEOMPatches are specified in subclause 9.2.7.

V3C bitstreams conforming to dynamic profiles at a specified tier and level shall obey the following constraints for each atlas bitstream conformance test as specified in Annex E.

* NumProjPoints shall be less than or equal to MaxNumProjPoints,
* NumRawPoints shall be less than or equal to MaxNumRawPoints,
* NumEOMPoints shall be less than or equal to MaxNumEOMPoints,
* NumProjPointsPerSec shall be less than or equal to MaxNumProjPointsPerSec,
* NumRawPointsPerSec shall be less than or equal to MaxNumRawPointsPerSec, and
* NumEOMPointsPerSec shall be less than or equal to MaxNumEOMPointsPerSec

where MaxNumProjPoints, MaxNumRawPoints, MaxNumEOMPoints, MaxNumProjPointsPerSec, MaxNumRawPointsPerSec, and MaxNumEOMPointsPerSec are specified in Table A-5.

Table A-6 – General atlas ASPS and tile related level limits

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Level** | **Max # PROJECTED Patches per atlas**  **MaxNumProjPatches** | **Max # RAW patches per atlas**  **MaxNumRawPatches** | **Max # EOM patches per atlas**  **MaxNumEOMPatches** | **Max CAB size**  **MaxCABSize in 1 000 bits** | **Max atlas bit rate**  **MaxAtlasBR**  **in 1 000 bits/s** | **Max # number of tiles per atlas**  **MaxNumTiles** | **Max atlas size**  **MaxAtlasSize** | **Max PROJECTED patch rate**  **MaxProjectedPatchRate (patches/s)** | **Max RAW patch rate**  **MaxRawPatchRate (patches/s)** | **Max EOM patch rate**  **MaxEOMPatchRate (patches/s)** |
| 1.0 | 2 048 | 32 | 32 | 15 000 | 15 000 | 50 | 2 228 224 | 65 536 | 1 024 | 1 024 |
| 1.5 | 4 096 | 64 | 32 | 30 000 | 30 000 | 50 | 2 228 224 | 131 072 | 2 048 | 1 024 |
| 2.0 | 16 384 | 128 | 64 | 120 000 | 120 000 | 200 | 8 912 896 | 524 288 | 4 096 | 2 048 |
| 2.5 | 32 384 | 128 | 64 | 240 000 | 240 000 | 200 | 8 912 896 | 1 036 288 | 4 096 | 2 048 |
| 3.0 | 65 536 | 512 | 128 | 480 000 | 480 000 | 500 | 35 651 584 | 2 097 152 | 16 384 | 4 096 |
| 3.5 | 65 536 | 512 | 512 | 480 000 | 480 000 | 500 | 35 651 584 | 4 194 304 | 32 768 | 32 768 |

Table A-6 specifies the atlas ASPS and tile related level limits.

The specified limits apply to all atlas tiles forming the V3C bitstream.

* AtlasTotalNumProjPatches shall be less than or equal to MaxNumProjPatches,
* AtlasTotalNumRawPatches shall be less than or equal to MaxNumRawPatches,
* AtlasTotalNumEOMPatches shall be less than or equal to MaxNumEOMPatches,
* AtlasTotalNumProjPatchesPerSec shall be less than or equal to MaxProjPatchesPerSec,
* AtlasTotalNumRawPatchesPerSec shall be less than or equal to MaxRawPatchesPerSec,
* AtlasTotalNumEOMPatchesPerSec shall be less than or equal to MaxProjectedEOMPerSec, and
* afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1 shall be less than or equal to MaxNumTiles,

where MaxNumProjPatches, MaxNumRawPatches, MaxNumEOMPatches, MaxRawPatchesPerSec, MaxProjectedEOMPerSec, and MaxNumTiles are specified in Table A-6.

Table A-7 – General video bitstream level limits

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Level** | **Max luma picture size in**  **MaxPictureSize (samples)** | **Max aggregate luma sample rate MaxAggregateLumaSr**  **(samples/sec)** | **Max # 1000 bits/s per video stream**  **MaxBitRatePerStream** | | **Max aggregate # 1000 bits/s**  **MaxAggregateBitRate** | |
| MT | HT | MT | HT |
| 1.0 | 2 228 224 | 133 693 440 | 12 000 | 30 000 | 25 000 | 100 000 |
| 1.5 | 2 228 224 | 267 386 880 | 20 000 | 50 000 | 40 000 | 160 000 |
| 2.0 | 8 912 896 | 534 773 760 | 25 000 | 100 000 | 60 000 | 240 000 |
| 2.5 | 8 912 896 | 1 069 547 520 | 60 000 | 240 000 | 120 000 | 480 000 |
| 3.0 | 35 651 584 | 2 139 095 040 | 60 000 | 240 000 | 120 000 | 480 000 |
| 3.5 | 35 651 584 | 4 278 190 080 | 240 000 | 800 000 | 480 000 | 1 600 000 |

Table A-7 specifies the general video bitstream level limits.

For example, when ptl\_profile\_codec\_group\_idc is equal to 0, the maximum sample rate of a single video sub-bitstream is specified in ISO/IEC 14496-10:2014:Table A-1 as MaxMBPS \* MaxFS. When ptl\_profile\_codec\_group\_idc is equal to 1 or equal to 2, the maximum sample rate of a single video sub-bitstream is specified as MaxLumaSr in ISO/IEC 23008-2:2020:Table A-7. When ptl\_profile\_codec\_group\_idc is equal to 127, the definition of the maximum sample rate of a single video sub-bitstream is outside the scope of this document.

Let the variable t0 to denote the time instance in milliseconds when the first bit across all V3C video sub-bitstreams is encountered.

For any time t1 greater than or equal to t0, the bit rate for each individual V3C video sub-bitstream, V3CVideoBitRate, is defined and set equal to the total number of decoded bits for the individual V3C video sub-bitstream for all values of t, with t1 <= t < t1 + 1 000.

It is a requirement of V3C bitstream conformance that V3CVideoBitrate is less than or equal to MaxBitRatePerStream, as specified in Table A-7, for all individual V3C video sub-bitstreams.

For any time t1 greater than or equal to t0, the aggregate sample rate, AggregateLumaSr, and the aggregate bit rate, AggregateBitRate, are defined and set equal to the total number of decoded luma samples and the total number of decoded bits, respectively, across all video components for all values of t, with t1 <= t < t1 + 1 000.

It is a requirement of V3C bitstream conformance that AggregateLumaSr and AggregateBitRate are less than or equal to MaxAggregateLumaSr and MaxAggregateBitRate, respectively, as specified in Table A-7.

For each video sub-bistream, the coded picture buffer size shall be less than or equal to the maximum coded picture buffer size, as specified by the corresponding video codec specification. For example, when ptl\_profile\_codec\_group\_idc is equal to 0, the maximum coded picture buffer size is identified by a variable MaxCPB specified in ISO/IEC 14496-10:2014:Table A-1. When ptl\_profile\_codec\_group\_idc is equal to 1 or equal to 2, the maximum coded picture buffer size is also identified by a variable MaxCPB specified in ISO/IEC 23008-2:2020:Annex A. When ptl\_profile\_codec\_group\_idc is equal to 127, the derivation of the maximum coded picture buffer size is outside the scope of this document.

The following variables are specified:

* Let the variable fR be set equal to 1 ÷ 1 000.
* Let the variable MaxPatchSize, which represents the maximum total number of bytes that is expected to be associated with all decoded elements in a patch, be equal to 40 bytes.
* Let the variable aR = AspsFrameSize ÷ MaxAggregateLumaSr.
* Let access unit n be the n-th access unit in decoding order, with the first access unit being access unit 0 (i.e., the 0-th access unit).
* Let atlas n be the coded atlas or the corresponding decoded atlas of access unit n.
* Let the variable MaxNumSubLayersMinus1 be equal to 0.

Then the following constraints apply:

1. The value of asps\_max\_dec\_atlas\_frame\_buffering\_minus1 + 1 shall be less than or equal to 16.
2. For the ACL HRD parameters, CabSize[ 1 ][ HtidOP ][ i ] shall be less than or equal to MaxCABSize for at least one value of i in the range of 0 to hrd\_cab\_cnt\_minus1[ HtidOP ], inclusive, where CabSize[ 1 ][ HtidOP ][ i ] is specified in subclause G.3.3 based on parameters selected as specified in subclause E.1 and MaxCABSize is specified in Table A-5 in bits.
3. For the NAL HRD parameters, CabSize[ 0 ][ HtidOP ][ i ] shall be less than or equal to MaxCABSize for at least one value of i in the range of 0 to hrd\_cab\_cnt\_minus1[ HtidOP ], inclusive, where CabSize[ 0 ][ HtidOP ][ i ] is specified in subclause G.3.3 based on parameters selected as specified in subclause E.1 and MaxCABSize is specified in Table A-1 in bits.
4. The nominal removal time of a coded atlas access unit n (with n greater than 0) from the CAB, as specified in subclause E.2.3, shall satisfy the constraint that AuNominalRemovalTime[ n ] − AuCabRemovalTime[ n − 1 ] is greater than or equal to Max( aR, fR ).
5. The difference between consecutive output times of atlases from the DAB, as specified in subclause E.3.3, shall satisfy the constraint that DabOutputInterval[ n ] is greater than or equal to Max( aR, fR ) provided that atlas is not the last atlas of the atlas bitstream that is output.
6. For the ACL HRD parameters, BitRate[ 1 ][ HtidOP ][ i ] shall be less than or equal to MaxAtlasBR for at least one value of i in the range of 0 to hrd\_cab\_cnt\_minus1[ HtidOP ], inclusive, where BitRate[ 1 ][ HtidOP ][ i ] is specified in subclause G.3.3 based on parameters selected as specified in subclause E.1 and MaxAtlasBR is specified in Table A-6.
7. For the NAL HRD parameters, BitRate[ 0 ][ HtidOP ][ i ] shall be less than or equal to MaxAtlasBR for at least one value of i in the range of 0 to hrd\_cab\_cnt\_minus1[ HtidOP ], inclusive, where BitRate[ 0 ][ HtidOP ][ i ] is specified in subclause G.3.3 based on parameters selected as specified in subclause E.1 and MaxAtlasBR is specified in Table A-7.
8. For coded atlas access unit 0, let the variables maxNumPatches, maxPatchBytes, and nB be defined as follows:

maxNumPatches = MaxNumProjPatches + MaxNumRawPatches + MaxNumEOMPatches  
 maxPatchBytes = MaxPatchRate \* MaxPatchSize  
 nB = Max( maxNumPatches, fR \* MaxPatchRate ) +  
 maxPatchBytes \* ( AuCabRemovalTime[ 0 ] − AuNominalRemovalTime[ 0 ] )

where the values of MaxNumProjPatches, MaxNumRawPatches, MaxNumEOMPatches, and MaxPatchRate are specified in Table A-6.

The sum of the NumBytesInNalUnit variables for coded atlas access unit 0 shall be less than or equal to nB.

1. For coded atlas access unit n, with n greater than 0, let the variables maxPatchBytes and nB be defined as follows:

maxPatchBytes = MaxPatchRate \* MaxPatchSize  
nB = maxPatchBytes \* ( AuCabRemovalTime[ n ] − AuCabRemovalTime[ n − 1 ] )

where MaxPatchRate is specified in Table A-6.

The sum of the NumBytesInNalUnit variables for access unit n shall be less than or equal to nB.

* 1. Decoder instantiations

Conforming bitstreams shall obey the following constraint:

NumDecodes shall be less than or equal to MaxDecodes specified in Table A-8, for values of ptl\_max\_decodes\_idc in the range of 0 .. 9.

ptl\_max\_decodes\_idc equal to 15 does not indicate a constraint. ptl\_max\_decodes\_idc values in the range of 10..14 are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this version of this specification.

Variable NumDecodes is derived as follows:

NumDecodes = 0  
for( j = 0; j < vps\_atlas\_count\_minus1 + 1; j++ ) {  
 NumDecodes  +=  vps\_auxiliary\_video\_present\_flag[ j ]  
 NumDecodes  +=  vps\_occupancy\_video\_present\_flag[ j ]  
 NumDecodes  +=  vps\_geometry\_video\_present\_flag[ j ] \* (vps\_map\_count\_minus1[ j ] + 1)  
 if ( vps\_attribute\_video\_present\_flag[ j ] )  
 NumDecodes  +=  ai\_attribute\_count[ j ] \* ( vps\_map\_count\_minus1[ j ] + 1 )  
 NumDecodes  −=  vps\_packed\_video\_present\_flag[ j ] ? pin\_regions\_count\_minus1[ j ] : 0  
}

Table A-8: Mapping of ptl\_max\_decodes\_idc to MaxDecodes

|  |  |
| --- | --- |
| **ptl\_max\_decodes\_idc** | **MaxDecodes** |
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | 4 |
| 4 | 6 |
| 5 | 8 |
| 6 | 12 |
| 7 | 16 |
| 8 | 24 |
| 9 | 32 |
| 10..14 | Reserved |
| 15 | Unconstrained |

1. (informative)  
   Post-decoding conversion to nominal video formats
   1. General

The video frames provided by the decoder may require additional processing steps before being input to the reconstruction process. Such processing steps may include conversion of the decoded video frames to a nominal format (e.g. a nominal resolution, bit depth, chroma format, etc.) or a conversion operation (e.g. map extraction, multi-dimensional attribute packing, etc.), as described in subclause B.2. This clause provides informative methods of how to perform such processing steps. These may be substituted by alternative processes, which are outside of the scope of this document, that may produce similar visual quality. The definition of similar visual quality is outside the scope of this document. However, for purposes of conformance to point B and as an informative example, a post‑decoding operation flow is presented in Figure B-1. The order of processing steps may not necessarily be the same as presented in Figure B-1.

The processes described in Annex B are invoked for decoded video components associated with the same atlas ID, identified by variable ConvAtlasID, which is set equal to vuh\_atlas\_id or determined through external means if the V3C unit header is unavailable.

The video frames provided by the decoder may require additional processing steps before the reconstruction process. Such processing steps may include unpacking of the decoded video frames to a seperate geometry, attribute and/or occupancy frames, as described in Annex B.4.

[Ed.(BK): Above paragraph is copied verbatim from ISO/IEC CD 23090-12 and needs to be integrated in Annex B.1 or Annex B.2.]

This clause takes as inputs the syntax elements and upper-case variables from clauses 8 and 9.



Figure B-1 – Post-processing conversion for a V3C bitstream with a single atlas

* 1. Nominal format conversion
     1. General

The nominal format refers collectively to the nominal bit depth, resolution, chroma format, and composition time index that the decoded videos should be converted to.

Each video sub-bitstream is associated with a nominal bit depth, which is the target bit depth that all operations for reconstruction are expected to be performed in. The nominal bit depth for the occupancy component, OccBitDepthNF, is set equal to oi\_occupancy\_2d\_bit\_depth\_minus1[ ConvAtlasID ] + 1. The nominal bit depth for each geometry video component, GeoBitDepthNF, is set equal to gi\_geometry\_2d\_bit\_depth\_minus1[ ConvAtlasID ] + 1. Finally, the nominal bit depth for each attribute video component with attribute index attrIdx, AttrBitDepthNF[ attrIdx ], is set equal to ai\_attribute\_2d\_bit\_depth\_minus1[ ConvAtlasID ][ attrIdx ] + 1.

The nominal frame resolution for non-auxiliary video components is defined by the nominal width, VideoWidthNF, set equal to asps\_frame\_width, and the nominal height, VideoHeightNF, set equal to asps\_frame\_height. The nominal frame resolution for auxiliary video components is defined by the nominal width and height specified by the variables AuxVideoWidthNF and AuxVideoHeightNF, respectively.

The nominal chroma format is defined to be 4:4:4.

The nominal composition time is defined based on the timing information of the atlas sub-bitstream and specified by the decoded atlas composition time, DecAtlasCompTime. The timing information of the atlas sub-bitstream may be provided either in the bitstream or by external means not specified in this document. For example, for V3C bitstreams encapsulated in the ISO media file format, the composition time for the atlas sub-bitstream is indicated according to ISO/IEC DIS 23090-10.

The conversion processes to the nominal format are defined as follows:

* for the occupancy video component subclause B.2.2 is invoked,
* for the geometry video component subclause B.2.3 is invoked,
* if the auxiliary geometry video component is present, for the auxiliary geometry video component subclause B.2.4 is invoked, and
* when the number of attributes is greater than zero, i.e. ai\_attribute\_count[ ConvAtlasID ] is greater than zero or determined through external means if the VPS is not available, then:
* for each attribute video component subclause B.2.5 is invoked and
* if auxiliary attribute video component is present, for each auxiliary attribute video component, subclause B.2.6 is invoked.

NOTE – For checking conformance point B it is recommended to follow the order of processes as described in this subclause. The output of this process can also be achieved by a different combination of processes or variations of the processes described in this subclause. The specification and order described in this subclause is informative only.

* + 1. Occupancy nominal format conversion

This process converts the decoded occupancy frames, DecOccFrames, to the nominal format.

Let the variables occNumComp, occMSBAlignFlag, and occThreshold be set as follows:

occNumComp = 1  
 occMSBAlignFlag = oi\_occupancy\_MSB\_align\_flag[ ConvAtlasID ]  
 occThreshold =  oi\_lossy\_occupancy\_compression\_threshold[ ConvAtlasID ]

Output of this process is:

* the 4D array OccFramesNF, indicating the decoded occupancy frames in the nominal format, where the dimensions correspond to the occupancy video frame index, the component index, the row index, and the column index, respectively.

To convert the decoded occupancy video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

* To convert the bit depth of the decoded occupancy video frames to the required nominal bit depth, subclause B.3.2 is invoked for each frame index frameIdx, with variables DecOccBitDepth[ frameIdx ], DecOccHeight[ frameIdx ], DecOccWidth[ frameIdx ], OccBitDepthNF, and occMSBAlignFlag, and array DecOccFrames[ frameIdx ][ 0 ] as inputs. The output of this process is the decoded occupancy frame at the nominal bit depth, decOccFramesNBD[ frameIdx ][ 0 ].
* To convert the occupancy frames to the nominal frame resolution, subclause B.3.3 is invoked for each frame index frameIdx, with variables DecOccHeight[ frameIdx ], DecOccWidth[ frameIdx ], VideoHeightNF, and VideoWidthNF, and array decOccFramesNBD[ frameIdx ][ 0 ] as inputs. The output of this process is the decoded occupancy frame at the nominal resolution, decOccFramesNR[ frameIdx ][ 0 ].
* To identify missing frames in the sequence and re-order the input, subclause B.3.4.1 is invoked with variables NumDecOccFrames, occNumComp, VideoHeightNF, and VideoWidthNF, and arrays decOccFramesNR, DecOccOutOrdIdx, and DecOccCompTime as inputs. The outputs of this process are the number of frames being output, numOutOrdOccFrames, the decoded occupancy frames ordered according to the output order index, decOccFramesONF, a 1D array, framePresent, indicating the presence of a frame, and a 1D array, decOccCompTimeONF, indicating the composition time of each frame ordered according to the output order index.
* To align the composition times of the decoded occupancy frames in output order and the decoded atlas frames, subclause B.3.4.2 is invoked with variables NumDecAtlasFrames, numOutOrdOccFrames, occNumComp, VideoHeightNF, and VideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decOccFramesONF, decOccCompTimeONF, and framePresent as inputs. The outputs of this process are the number of frames being output, rangeCompTimeIdx, the output composition time of each frame, occCompTime, and a sequence of frames in the nominal format aligned with the decoded atlas composition time, OccFramesNF.
* If asps\_eom\_patch\_enabled\_flag is equal to 0, to convert the sample values in the decoded occupancy frames to binary values, subclause B.3.9 is invoked for each frame index frameIdx, with variables occThreshold, OccBitDepthNF, VideoHeightNF, and VideoWidthNF, and array OccFramesNF[ frameIdx ][ 0 ] as inputs. The output of this process is the updated occupancy frame, OccFramesNF[ frameIdx ][ 0 ], containing binary values only.

NOTE 1 – In an alternative implementation, a decoder may perform the thresholding process before the occupancy was upscaled to the nominal resolution.

Given OccFramesNF, the variable NumProjPoints is derived as follows:

NumProjPoints = 0  
 for ( y = 0; y < asps\_frame\_height; y++ )  
 for( x = 0; x < asps\_frame\_width; x++ )  
 if( OccFramesNF[ compTimeIdx ][ 0 ][ y ][ x ] != 0 )  
 NumProjPoints++  
  
 projectedPointsScale = asps\_map\_count\_minus1 + 1  
 for ( i = 0; i < asps\_map\_count\_minus1+1; i++ ) {  
 addPoints = ( asps\_map\_pixel\_deinterleaving\_flag[ i ] !=0 ) || ( plri\_map\_present\_flag[ i ] !=0 )  
 projectedPointsScale += addPoints  
 }  
 NumProjPoints = NumProjPoints \* projectedPointsScale

* + 1. Geometry nominal format conversion

This process converts the decoded geometry frames, DecGeoFrames, to the nominal format.

Let the variables geoNumComp, geoMultipleMapsPresentFlag, geoMapCount, and geoMSBAlignFlag, be set as follows:

geoNumComp = 1  
 geoMultipleMapsPresentFlag = vps\_multiple\_map\_streams\_present\_flag[ ConvAtlasID ]  
 geoMapCount = vps\_map\_count\_minus1[ ConvAtlasID ] + 1  
 geoMSBAlignFlag = gi\_geometry\_MSB\_align\_flag[ ConvAtlasID ]

Let the 1D array geoMapAbsCodingFlag be set as follows:

for( c = 0; c < geoMapCount; c++ ) {  
 geoMapAbsCodingFlag[ c ] = vps\_map\_absolute\_coding\_enabled\_flag[ ConvAtlasID ][ c ]  
 }

Output of this process is:

* the 5D array GeoFramesNF, the decoded geometry frames in the nominal format, where the dimensions correspond to the map index, the geometry video frame index, the components, the column index, and the row index of the frames.

NOTE 1 – The order of processes or variations of the processes described in this Annex is informative only. The maps are first separated and then converted. The conversion may happen before the map separation, right after decoding.

To convert the decoded geometry video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

* To convert the bit depth of the geometry frame to the required nominal bit depth resolution, subclause B.3.2 is invoked for each map index, mapIdx, and each frame index, frameIdx, with variables DecGeoBitDepth[ mapIdx ][ frameIdx ], DecGeoHeight[ mapIdx ][ frameIdx ], DecGeoWidth[ mapIdx ][ frameIdx ], GeoBitDepthNF, and geoMSBAlignFlag, and array DecGeoFrames[ mapIdx ][ frameIdx ][ 0 ] as inputs. Output of this process is the decoded geometry frame at the nominal bit depth, decGeoFramesNBD[ mapIdx ][ frameIdx ][ 0 ].
* To convert the frame resolution of the geometry frame to the required nominal frame resolution, subclause B.3.3 is invoked for each map index, mapIdx, and each frame index, frameIdx, with variables DecGeoHeight[ mapIdx ][ frameIdx ], DecGeoWidth[ mapIdx ][ frameIdx ], VideoHeightNF, and VideoWidthNF, and array decGeoFramesNBD[ mapIdx ][ frameIdx ][ 0 ] as inputs. Output of this process is the decoded geometry frame at the nominal resolution, decGeoFramesNR[ mapIdx ][ frameIdx ][ 0 ].
* To extract the maps, subclause B.3.1 is invoked, with variables geoMultipleMapsPresentFlag, geoMapCount, geoNumComp, VideoHeightNF, and VideoWidthNF, and arrays NumDecGeoFrames, decGeoFramesNR, DecGeoOutOrdIdx, and DecGeoCompTime as inputs. Outputs of this process are the number of frames being output, rangeOutOrdIdx, the decoded nominal resolution geometry frames ordered according to the output order index, decGeoFramesONF, a 1D array, framePresent, indicating if a frame is present, and a 1D array, decGeoCompTime, indicating the composition time of each frame ordered according to the output order index.
* To align the composition times of the decoded geometry and the decoded atlas frames, subclause B.3.4.2 is invoked for each map index, mapIdx, with variables NumDecAtlasFrames, rangeOutOrdIdx[ mapIdx ], geoNumComp, VideoHeightNF, and VideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decGeoFramesONF[ mapIdx ], decGeoCompTimeONF[ mapIdx ], and framePresent[ mapIdx ] as inputs. The outputs of this process are the number of frames being output, rangeCompTimeIdx, the output composition time of each frame, geoCompTime[ mapIdx ], and a sequence of frames in the nominal format aligned with the decoded atlas composition time, GeoFramesNF[ mapIdx ].
* To synthesize the maps of the geometry, subclause B.3.7 is invoked with variables geoMultipleMapsPresentFlag, geoMapCount, rangeCompTimeIdx, GeoBitDepthNF, geoNumComp, VideoHeightNF, and VideoWidthNF, and arrays GeoFramesNF and geoMapAbsCodingFlag as inputs. The output of this process is the updated decoded geometry frame in the nominal format, GeoFramesNF, containing synthesized geometry values.
  + 1. Auxiliary geometry nominal format conversion

This process converts the decoded auxiliary geometry frames, DecGeoAuxFrames, to the nominal format.

Let the variables geoAuxNumComp and geoMSBAlignFlag, be set as follows:

geoAuxNumComp = 1  
 geoMSBAlignFlag = gi\_geometry\_MSB\_align\_flag[ ConvAtlasID ]

Output of this process is:

* the 4D array GeoAuxFramesNF, the decoded geometry frames in the nominal format, where the dimensions correspond to the geometry video frame index, the components, the column index, and the row index of the frames.

NOTE 2 – The output of this process can also be achieved by a different combination of processes or variations of the processes described in this Annex. The specification and order described in this Annex is informative only.

To convert the decoded auxiliary geometry video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

* To convert the bit depth of the auxiliary geometry to the required nominal bit depth resolution, subclause B.3.2 is invoked for each frame index frameIdx, with variables DecGeoAuxBitDepth[ frameIdx ], DecGeoAuxHeight[ frameIdx ], DecGeoAuxWidth[ frameIdx ], GeoBitDepthNF, and geoMSBAlignFlag, and array DecGeoAuxFrames[ frameIdx ][ 0 ] as inputs. The output of this process is the decoded auxiliary geometry frame at the nominal bit depth, decGeoAuxFramesNBD[ frameIdx ][ 0 ].
* To convert the frame resolution of the auxiliary geometry to the required nominal frame resolution, subclause B.3.3 is invoked for each frame index frameIdx, with variables DecGeoAuxHeight[ frameIdx ], DecGeoAuxWidth[ frameIdx ], AuxVideoWidthNF, and AuxVideoHeightNF, and array decGeoAuxFramesNBD[ frameIdx ][ 0 ] as inputs. The output of this process is the decoded auxiliary geometry frame at the nominal resolution, decGeoAuxFramesNR[ frameIdx ][ 0 ].
* To convert the auxiliary geometry frame order defined by frameIdx, to the required output order defined by ordIdx, subclause B.3.4.1 is invoked with variables NumDecGeoAuxFrames, geoAuxNumComp, AuxVideoHeightNF, and AuxVideoWidthNF, and arrays decGeoAuxFramesNR, DecGeoAuxOutOrdIdx, and DecGeoAuxCompTime as inputs. The outputs of this process are the number of frames being output, numOutOrdGeoAuxFrames, the decoded auxiliary geometry frames ordered according to the output order index, decGeoAuxFramesONF, a 1D array, framePresent, indicating the presence of a frame, and a 1D array, decGeoAuxCompTimeONF, indicating the composition time of each frame ordered according to the output order index.
* To align the composition times of the decoded auxiliary geometry frames in output order and the decoded atlas frames, subclause B.3.4.2 is invoked with variables NumDecAtlasFrames, numOutOrdGeoAuxFrames, geoAuxNumComp, AuxVideoHeightNF, and AuxVideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decGeoAuxFramesONF, decGeoAuxCompTimeONF, and framePresent as inputs. The outputs of this process are the number of frames, rangeCompTimeIdx, the output composition time of each frame, auxGeoCompTime, and a sequence of frames in the nominal format aligned with the decoded atlas composition time, GeoAuxFramesNF.
  + 1. Attribute nominal format conversion

This process converts the decoded attribute frames DecAttrFrames to the nominal format.

The processes described below are invoked for decoded attribute frames associated with the same attribute index, identified by variable attrIdx, which is set equal to vuh\_attribute\_index or determined through external means if the V3C unit header is unavailable.

Let the variables, attrMultipleMapsPresentFlag, attrMapCount, attrMSBAlignFlag, attrDim, and attrNumPart be set as follows:

attrMultipleMapsPresentFlag = vps\_multiple\_map\_streams\_present\_flag[ ConvAtlasID ]  
 attrMapCount = vps\_map\_count\_minus1[ ConvAtlasID ] + 1  
 attrMSBAlignFlag = ai\_attribute\_MSB\_align\_flag[ ConvAtlasID ][ attrIdx ]  
 attrDim = ai\_attribute\_dimension\_minus1[ ConvAtlasID ][ attrIdx ] + 1  
 attrNumPart = ai\_attribute\_dimension\_partitions\_minus1[ ConvAtlasID ][ attrIdx ] + 1

Let the 1D array attrPartChannelsMinus1, of size attrNumPart, be set as follows:

for( c = 0; c < attrNumPart; c++ ) {  
 attrPartChannelsMinus1[ c ] =   
 ai\_attribute\_partition\_channels\_minus1[ ConvAtlasID ][ attrIdx ][ c ]  
 }

Let the 1D array attrMapAbsCodingFlag, of size attrMapCount, be set as follows:

for( c = 0; c < attrMapCount; c++ ) {  
 attrMapAbsCodingFlag[ c ] =   
 AttributeMapAbsoluteCodingEnabledFlag[ ConvAtlasID ][ attrIdx ][ c ]  
 }

Output of this process is:

* the 7D array AttrAuxFramesNF, the decoded attribute frames in the nominal format, where the dimensions correspond to the attribute index, the attribute partition index, the map index, the decoded attribute video frame index, the component index, the row index, and the column index, respectively.

NOTE 3 – The order of processes or variations of the processes described in this Annex is informative only. The maps are first separated and then converted. The conversion may happen before the map separation, right after decoding.

To convert the decoded attribute video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

* To convert the bit depth of the attribute frame to the required nominal bit depth resolution, subclause B.3.2 is invoked for each attribute partition index, partIdx, each map index, mapIdx, each frame index, frameIdx, and each component index, compIdx, with variables DecAttrBitDepth[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], DecAttrHeight[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], DecAttrWidth[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], AttrBitDepthNF[ attrIdx ], and attrMSBAlignFlag, and array DecAttrFrames[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ][ compIdx ] as inputs. Output of this process is the decoded attribute frame at the nominal bit depth, decAttrFramesNBD[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ][ compIdx ].
* To convert the chroma format of the attribute frame with attributes type DecAttrType[ attrIdx ] equal to ATTR\_TEXTURE, to the required nominal chroma format, subclause B.3.6 is invoked for each attribute partition index, partIdx, each map index, mapIdx, and each frame index, frameIdx, with variables DecAttrHeight[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], DecAttrWidth[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], AttrBitDepthNF[ attrIdx ], DecAttrChromaFormat[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], and DecAttrChromaSamplingPosition[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], and array decAttrFramesNBD[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ] as inputs. The output of this process is the decoded attribute frame with a 4:4:4 colour format, decAttrFramesNCF[ attrIdx ][ mapIdx ][ frameIdx ]
* To convert the frame resolution of the attribute frame to the required nominal frame resolution, subclause B.3.3 is invoked for each attribute partition index, partIdx, each map index, mapIdx, each frame index frameIdx, and each component index, compIdx, with variables DecAttrHeight[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], DecAttrWidth[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ], VideoHeightNF, and VideoWidthNF, and array decAttrFramesNCF[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ][ compIdx ] as inputs. Output of this process is the decoded attribute frame with nominal resolution, decAttrFramesNR[ attrIdx ][ partIdx ][ mapIdx ][ frameIdx ][ compIdx ].
* To extract the maps, subclause B.3.1 is invoked for each attribute partition index, partIdx, with variables attrMultipleMapsPresentFlag, attrMapCount, DecAttrNumComp[ attrIdx ][ partIdx ], VideoHeightNF, and VideoWidthNF, and arrays NumDecAttrFrames[ attrIdx ][ partIdx ], decAttrFramesNR[ attrIdx ][ partIdx ], DecAttrOutOrderIdx[ attrIdx ][ partIdx ], and DecAttrCompTime[ attrIdx ][ partIdx ] as inputs. Outputs of this process are the number of frames being output, rangeOutOrdIdx[ attrIdx ][ partIdx ], the decoded attribute frames ordered according to the output order index, decAttrFramesONF[ attrIdx ][ partIdx ], a 1D array, framePresent[attrIdx][ partIdx ], indicating if a frame is present, and a 1D array, decAttrCompTime[ attrIdx ][ partIdx ], indicating the composition time of each frame ordered according to the output order index.
* To align the composition times of the decoded attributes and the decoded atlas frames, subclause B.3.4.2 is invoked for each attribute partition index, partIdx, and each map index, mapIdx, with variables NumDecAtlasFrames, rangeOutOrdIdx[ attrIdx ][ partIdx ][ mapIdx ], DecAttrNumComp[ attrIdx ][ partIdx ][ mapIdx ], VideoHeightNF, and VideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decAttrFramesONF[ attrIdx ][ partIdx ][ mapIdx ], decAttrCompTimeONF[ attrIdx ][ partIdx ][ mapIdx ], and framePresent[ attrIdx ][ partIdx ][ mapIdx ] as inputs. The outputs of this process are the number of frames being output, rangeCompTimeIdx[ attrIdx ], the output composition time of each frame, attrCompTime[ attrIdx ][ partIdx ][ mapIdx ], and a sequence of frames in the nominal format aligned with the decoded atlas composition time, decAttrFramesANF[ attrIdx ][ partIdx ][ mapIdx ].
* To perform dimension packing of attributes, subclause B.3.5.1 is invoked, with variables attrNumPart, attrMapCount, rangeCompTimeIdx[ attrIdx ], (attrDim-(attrNumPart-1)), VideoHeightNF, VideoWidthNF, and attrDim, and arrays decAttrFramesANF[ attrIdx ], and attrPartChannelsMinus1 as input. The output of this process is a sequence of attribute frames with merged partitions at the nominal format, AttrFramesNF[ attrIdx ].
* To synthesize the maps of the attribute, subclause B.3.8 is invoked with variables attrMultipleMapsPresentFlag, attrMapCount, rangeCompTimeIdx[ attrIdx ], AttrBitDepthNF[ attrIdx ], attrDim[ attrIdx ], VideoHeightNF, and VideoWidthNF, and arrays AttrFramesNF[ attrIdx ] and AttributeMapAbsoluteCodingEnabledFlag[ ConvAtlasID ][ attrIdx ] as inputs. The output of this process is the updated decoded attribute frame in the nominal format, AttrFramesNF[ attrIdx ], containing synthesized attribute values.
  + 1. Auxiliary attribute nominal format conversion

This process converts the decoded auxiliary attribute frames DecAttrAuxFrames to the nominal format.

The processes described below are invoked for decoded auxiliary attribute frames associated with the same attribute index, identified by variable attrIdx, which is set equal to vuh\_attribute\_index or determined through external means if the V3C unit header is unavailable.

Let the variable attrMSBAlignFlag, attrDim, and attrNumPart be set as follows:

attrMSBAlignFlag = ai\_attribute\_MSB\_align\_flag[ ConvAtlasID ][ attrIdx ]  
 attrDim = ai\_attribute\_dimension\_minus1[ ConvAtlasID ][ attrIdx ] +1  
 attrNumPart = ai\_attribute\_dimension\_partitions\_minus1[ ConvAtlasID ][ attrIdx ] +1

Let the 1D array attrPartChannelsMinus1, of size attrNumPart, be set as follows:

for( c = 0; c < attrNumPart; c++ ) {  
 attrPartChannelsMinus1[ c ] =   
 ai\_attribute\_partition\_channels\_minus1[ ConvAtlasID ][ attrIdx ][ c ]  
 }

Output of this process is:

* the 6D array AttrAuxFramesNF, the decoded auxiliary attribute frames in the nominal format, where the dimensions correspond to the attribute index, the attribute partition index, the decoded attribute video frame index, the component index, the row index, and the column index, respectively.

NOTE 4 – The output of this process can also be achieved by a different combination of processes or variations of the processes described in this Annex. The specification and order described in this Annex is informative only.

To convert the decoded auxiliary attribute video to the nominal format, several processing steps may need to be performed depending on the original format of the decoded video. In particular:

* To convert the bit depth of the decoded auxiliary attribute video frames to the required nominal bit depth, subclause B.3.2 is invoked for each partition index partIdx, each frame index frameIdx, and each component index compIdx, with variables DecAttrAuxBitDepth[ attrIdx ][ partIdx ][ frameIdx ], DecAttrAuxHeight[ attrIdx ][ partIdx ][ frameIdx ], DecAttrAuxWidth[ attrIdx ][ partIdx ][ frameIdx ], AttrBitDepthNF[ attrIdx ], and attrMSBAlignFlag, and array DecAttrAuxFrames[ attrIdx ][ partIdx ][ frameIdx ][ compIdx ] as inputs. The output of this process is the decoded auxiliary attribute frame at the nominal bit depth, decAttrAuxFrameNBD[ attrIdx ][ partIdx ][ frameIdx ][ frameIdx ].
* To convert the chroma format of the auxiliary attribute frame with attribute type DecAttrAuxType[ attrIdx ] equal to ATTR\_TEXTURE, to the required nominal chroma format, subclause B.3.6 is invoked for each attribute partition index, partIdx, and each frame index, frameIdx, with variables DecAttrAuxHeight[ attrIdx ][ partIdx ][ frameIdx ], DecAttrAuxWidth[ attrIdx ][ partIdx ][ frameIdx ], AttrBitDepthNF[ attrIdx ], DecAttrAuxChromaFormat[ attrIdx ][ partIdx ][ frameIdx ], and DecAttrAuxChromaSamplingPosition[ attrIdx ][ partIdx ][ frameIdx ], and array decAttrAuxFrameNBD[ attrIdx ][ partIdx ][ frameIdx ] as inputs. The output of this process is a decoded auxiliary attribute frame with 4:4:4 colour format, decAttrAuxFrameNCF[ attrIdx ][ mapIdx ][ compTimeIdx ].

NOTE 5 – When asps\_pixel\_deinterleaving\_enabled\_flag is equal to 1, the chroma format conversion process described in subclause B.3.6 may not work as intended or produce the desired visual quality.

* To convert the frame resolution of the auxiliary attribute to the required nominal frame resolution, subclause B.3.3 is invoked for each partition index, partIdx, each frame index, frameIdx, and each component index, compIdx with variables DecAttrAuxHeight[ attrIdx ][ partIdx ][ frameIdx ], DecAttrAuxWidth[ attrIdx ][ partIdx ][ frameIdx ], AuxVideoHeightNF, and AuxVideoWidthNF, and array decAttrAuxFrameNCF[ attrIdx ][ partIdx ][ frameIdx ][ compIdx ] as inputs. The output of this process is the decoded auxiliary attribute frame at the nominal resolution, decAttrAuxFrameNR[ attrIdx ][ partIdx ][ frameIdx ][ compIdx ].
* To identify missing frames in the sequence and re-order the input, subclause B.3.4.1 is invoked for each partition index partIdx, with variables NumDecAttrAuxFrames[ attrIdx ][ partIdx ], DecAttrAuxNumComp[ attrIdx ][ partIdx ], AuxVideoWidthNF, and AuxVideoHeightNF, and arrays decAttrAuxFrameNR, DecAttrAuxOutOrdIdx[ attrIdx ][ partIdx ], and DecAttrAuxCompTime[ attrIdx ][ partIdx ] as inputs. The outputs of this process are the number of frames being output, numOutOrdAttrAuxFrames[ attrIdx ][ partIdx ], the decoded auxiliary attribute frames ordered according to the output order index, decAttrAuxFrameONF[ attrIdx ][ partIdx ], a 1D array, framePresent[ attrIdx ][ partIdx ], indicating the presence of a frame, and a 1D array, decAttrAuxCompTimeONF[ attrIdx ][ partIdx ], indicating the composition times of each frame ordered according to the output order index.
* To align the composition times of the decoded auxiliary attribute and the decoded atlas frames, subclause B.3.4.2 is invoked for each partition index partIdx with variables NumDecAtlasFrames, numOutOrdAttrAuxFrames[ attrIdx ][ partIdx ], AuxVideoHeightNF, and AuxVideoWidthNF, and arrays DecAtlasOutOrdIdx, DecAtlasCompTime, decAttrAuxFrameONF[ attrIdx ][ partIdx ], decAttrAuxCompTimeONF[ attrIdx ][ partIdx ], and framePresent[ attrIdx ][ partIdx ] as inputs. The outputs of this process are the number of frames being output, rangeCompTimeIdx[ attrIdx ], the output composition time of each frame, auxAttrCompTime[ attrIdx ][ partIdx ], and a sequence of frames in the nominal format aligned with the decoded atlas composition time, decAttrAuxFrameANF[ attrIdx ][ partIdx ].
* To perform dimension packing of auxiliary attributes, subclause B.3.5.2 is invoked with variables attrNumPart, rangeCompTimeIdx[ attrIdx ], (attrDim-(attrNumPart-1)), VideoHeightNF, VideoWidthNF, and attrDim, and arrays decAttrAuxFrameANF[ attrIdx ] and attrPartChannels as input. The output of this process is a decoded auxiliary attribute frame with merged partitions, AttrAuxFramesNF[ attrIdx ].
  1. Conversion operations
     1. Map Extraction

Inputs to this process are:

* a variable multipleMapStreamsPresentFlag, indicating the presence of multiple streams,
* a variable mapCount, indicating the number of maps
* a variable iNumComp, indicating the number of components for each frame,
* a variable iHeight, indicating the height of each frame,
* a variable iWidth, indicating the width of each frame,
* a 1D array numFrames, of size mapCount, indicating the number of input frames,
* a 5D array inputFrame of size mapCount × numInputFrames × iNumComp × iHeight × iWidth,
* a 2D array iOutputOrderIdx, of size mapCount × numInputFrames, indicating the output order index, and
* a 2D array iCompTime, of size mapCount × numInputFrames , indicating the composition time,

where numInputFrames is the maximum value of the numFrames array.

Outputs of this process are:

* a 1D array rangeOrdIdx, of size mapCount, indicating the number of frames of each map,
* a 5D array outputSamples, of size mapCount × numOutputFrames × iNumComp × iHeight × iWidth,
* a 2D array framePresent, of size mapCount × numOutputFrames, indicating the presence of a frame,
* a 2D array outputCompTime, of size mapCount × numOutputFrames, indicating the composition time of each frame,

where numOutputFrames is the maximum value of the array rangeOrdIdx.

The map extraction process is performed as follows:

for( i = 0; i < mapCount; i++ )  
 rangeOrdIdx[ i ] = 0  
 if( mapCount – 1 == 0 ) {  
 for( f = 0; f < numFrames[ 0 ]; f++ ) {  
 g = iOutputOrderIdx[ 0 ][ f ]  
 for( i = rangeOrdIdx[ 0 ]; i < g; i++ )  
 framePresent[ 0 ][ i ] = 0  
 framePresent[ 0 ][ g ] = 1  
 rangeOrdIdx[ 0 ] = g + 1  
 outputCompTime[ 0 ][ g ] = inputCompTime[ 0 ][ f ]  
 for( c = 0; c < iNumComp; c++ )  
 for( y = 0; y < iHeight; y++ )  
 for( x = 0; x < iWidth; x++ )  
 outputSamples[ 0 ][ g ][ c ][ y ][ x ] = inputSamples[ 0 ][ f ][ c ][ y ][ x ]  
 }  
 } else {  
 if( multipleMapStreamsPresentFlag == 0 ) {  
 for( f = 0; f < numFrames[ 0 ]; f++ ) {  
 g = iOutputOrderIdx[ 0 ][ f ] / ( vps\_map\_count\_minus1[ ConvAtlasID ] + 1 )  
 m = iOutputOrderIdx[ 0 ][ f ] % ( vps\_map\_count\_minus1[ ConvAtlasID ] + 1 )  
 for( i = rangeOrdIdx[ m ]; i < g; i++ )  
 framePresent[ m ][ i ] = 0  
 framePresent[ m ][ g ] = 1  
 rangeOrdIdx[ m ] = g + 1  
 outputCompTime[ m ][ g ] = iCompTime[ 0 ][ f ]  
 for( c = 0; c < iNumComp; c++ )  
 for( y = 0; y < iHeight; y++ )  
 for( x = 0; x < iWidth; x++ )  
 outputSamples[ m ][ g ][ c ][ y ][ x ] =  
 inputSamples[0][ f ][ c ][ y ][ x ]  
 }  
 } else {  
 for( m = 0; m <= vps\_map\_count\_minus1[ ConvAtlasID ]; m++ ) {  
 for( f = 0; f < numFrames[ m ]; f++ ) {  
 g = inputOrderIdx[ m ][ f]  
 for( i = rangeOrdIdx[ m ]; i < g; i++ )  
 framePresent[ m ][ i ] = 0  
 framePresent[ m ][ g ] = 1  
 rangeOrdIdx[ m ] = g + 1  
 outputCompTime[ m ][ g ] = iCompTime[ i ][ f ]  
 for( c = 0; c < iNumComp; c++ )  
 for( y = 0; y < iHeight; y++ )  
 for( x = 0; x < iWidth; x++ )  
 outputSamples[ m ][ g ][ c ][ y ][ x ] =  
 inputSamples[ m ][ f ][ c ][ y ][ x ]  
 }  
 }  
 }  
 }

* + 1. Bit depth conversion

Inputs to this process are:

* a variable inputBitDepth, indicating the bit depth of inputSamples,
* a variable inputHeight, indicating the height of inputSamples,
* a variable inputWidth, indicating the width of inputSamples,
* a variable nominalBitDepth, indicating the nominal bit depth,
* a variable alignmentFlag, indicating the MSB alignment, and
* a 2D array inputSamples, of size inputHeight × inputWidth.

Output of this process is:

* a 2D array outputSamples of size inputWidth × inputHeight.

The variable bitDepthDifference is derived as follows:

* bitDepthDifference = inputBitDepth – nominalBitDepth

Then the array outputSamples is generated in the following manner:

if( ( bitDepthDifference > 0 ) && ( alignmentFlag == 1 ) ) {  
 for( y=0; y < inputHeight; y++ )  
 for( x=0; x < inputWidth; x++ )  
 outputSamples[ y ][ x ] = ( inputSamples[ y ][ x ] >> bitDepthDifference)  
 }  
 else if( ( bitDepthDifference > 0 ) ) {  
 for( y=0; y < inputHeight; y++ )  
 for( x=0; x < inputWidth; x++ )  
 outputSamples[ y ][ x ] = Min( inputSamples[ y ][ x ], ( 1 << nominalBitDepth ) – 1 )  
 } else if( alignmentFlag == 1 ) {  
 for( y=0; y < inputHeight; y++ )  
 for( x=0; x < inputWidth; x++ )  
 outputSamples[ y ][ x ] = ( inputSamples[ y ][ x ] << (-bitDepthDifference) )  
 } else {  
 for( y=0; y < inputHeight; y++ )  
 for( x=0; x < inputWidth; x++ )  
 outputSamples[ y ][ x ] = inputSamples[ y ][ x ]  
 }

* + 1. Resolution conversion

Inputs to this process are:

* a variable inputHeight, indicating the height of inputSamples
* a variable inputWidth, indicating the width of inputSamples
* a variable nominalHeight, indicating the nominal height, and
* a variable nominalWidth, indicating the nominal width,
* a 2D array inputSamples, of size inputHeight × inputWidth.

Output of this process is:

* a 2D array outputSamples of size nominalHeight × nominalWidth.

The variables scaleY and scaleX are derived as follows:

scaleY = inputHeight ÷ nominalHeight  
 scaleX = inputWidth ÷ nominalWidth

Then the array outputSamples is generated as follows:

for( y=0; y < nominalHeight; y++ )  
 for( x=0; x < nominalWidth; x++ )  
 outputSamples[ y ][ x ] = inputSamples[ Floor( y \* scaleY ) ][ Floor( x \* scaleX ) ]

NOTE – In the above process pixel replication is used. A more sophisticated resampling process may be used. For example, the upsampling process using the filters specified in ISO/IEC 23008-2:2020:H.8.1.4.2.2 can be used. Other filters such as Lanczos can also be used.

* + 1. Output composition time conversion
       1. Output order conversion

Inputs to this process are:

* a variable numInputFrames, indicating the number of input frames,
* a variable iNumComp, indicating the number of components for each frame,
* a variable iHeight, indicating the height of each frame,
* a variable iWidth, indicating the width of each frame,
* a 4D array inputFrames, of size numInputFrames × iNumComp × iHeight × iWidth
* a 1D array outOrdIdx, of size numInputFrames, indicating the output order index of each frame, and
* a 1D array iCompTime, of size numInputFrames, indicating the composition time of each frame.

Outputs of this process are:

* a variable rangeOrdIdx, indicating the number of output frames,
* a 4D array outputFrames, of size rangeOrdIdx × iNumComp × iHeight × iWidth
* a 1D array framePresent, of size rangeOrdIdx, indicating the presence of a frame, and
* a 1D array outputCompTime, of size rangeOrdIdx, indicating the composition time of each frame.

This process copies the input frames to an output structure according to their output order index and assigns to them a composition time. It also generates a list identifying any missing frames. The following applies:

rangeOrdIdx = 0  
 for( f = 0; f < numFrames; f++ ) {  
 ordIdx = outOrdIdx[ 0 ][ f ]  
  
 for( i = rangeOrdIdx ; i < ordIdx; i++ )  
 framePresent[ i ] = 0  
  
 framePresent[ ordIdx ] = 1  
 outputCompTime[ ordIdx ] = iCompTime[ f ]  
 for( c=0; c < iNumComp; c++ ) {  
 for( y=0; y < iHeight; y++ ) {  
 for( x=0; x < iWidth; x++ ) {  
 outputFrames[ ordIdx ][ c ][ y ][ x ] = inputFrames[ f ][ c ][ y ][ x ]  
 }  
 }  
 }  
  
 rangeOrdIdx = ordIdx + 1  
 }

* + - 1. Atlas composition time alignment

Inputs to this process are:

* a variable numAtlFrm, indicating the number of atlas frames,
* a variable numInFrames, indicating the number of input frames,
* a variable iNumComp, indicating the number of components for each frame,
* a variable iHeight, indicating the height of each frame,
* a variable iWidth, indicating the width of each frame,
* a 1D array atlasOutOrdIdx, of size numAtlFrm, indicating the atlas output order index,
* a 1D array atlasCompTime, of size numAtlFrm, indicating the atlas composition time,
* a 4D array inputFrames, of size numInFrames × iNumComp × iHeight × iWidth
* a 1D array iCompTime, of size numInFrames, indicating the composition time of each frame, and
* a 1D array framePresent, of size numInFrames, indicating the presence of a frame.

Outputs of this process are:

* a variable rangeCompTimeIdx, indicating the number of output frames the number of frames,
* a 4D array outputFrames, of size rangeCompTimeIdx × iNumComp × iHeight × iWidth
* a 1D array outputCompTime, of size rangeCompTimeIdx, indicating the composition time of each frame.

This process copies the input frames to an output structure according to the relationship of their composition time to that of the atlas. It then re-assigns to them the composition time of the atlas. In particular, the following applies:

rangeCompTimeIdx = 0  
 inputOrdIdx = 0  
 for( i=0; i < numAtlFrm; i++ ) {   
 compTimeIdx = atlasOutOrdIdx[ i ]  
 compTime = atlasCompTime[ i ]  
  
 foundCompositionTime = 0  
 while( ( inputOrdIdx <numInFrames ) && ( foundCompositionTime == 0 ) ) {  
 skipFrames = 0  
 while( ( ( inputOrdIdx+ skipFrames + 1 ) < numInFrames ) &&  
 ( framePresent[ inputOrdIdx + skipFrames ] == 0 ) )  
 skipFrames = skipFrames + 1

inputOrdIdx = inputOrdIdx + skipFrames  
 if( iCompTime[ inputOrdIdx ] == compTime ) {  
 foundCompositionTime = 1  
 savedInputOrdIdx = inputOrdIdx   
 } else {  
 if( iCompTime[ inputOrdIdx ] > compTime )  
 foundCompositionTime = 1   
 else {  
 savedInputOrdIdx = inputOrdIdx  
 inputOrdIdx = inputOrdIdx +1  
 }  
 }  
 }  
  
 for( i = rangeCompTimeIdx; i <= compTimeIdx; i++ ) {  
 outputCompTime[ i ] = compTime  
 for( c=0; c < iNumComp; c++ ) {  
 for( y=0; y < iHeight; y++ ) {  
 for( x=0; x < iWidth; x++ ) {  
 outputSamples[ i ][ c ][ y ][ x ] =  
 inputSamples[ savedInputOrdIdx ][ c ][ y ][ x ]  
 }  
 }  
 }  
 }  
  
 rangeCompTimeIdx = compTimeIdx + 1  
 }

Two informative examples of alignment of the composition times of the decoded video component are presented in Figure B-2 and Figure B-3.



Figure B-2 – An example of alignment of the composition times of the decoded video component with different frame rate and the decoded atlas frames



Figure B-3 – An example of alignment of the composition times of the decoded video component containing pixel interleaved maps and the decoded atlas frames

* + 1. Attribute dimension packing
       1. Attribute dimension packing process

Inputs to this process are:

* a variable iNumPart, indicating the number of dimension partitions in which an attribute has been partitioned
* a variable iNumMaps, indicating the number of maps
* a variable iNumFrames, indicating the number of frames
* a variable iNumComp, indicating the number of components for each frame,
* a variable iHeight, indicating the height of each frame,
* a variable iWidth, indicating the width of each frame,
* a variable iAttrDim, indicating the dimensions of the attribute,
* a 6D array, iFrames, of size iNumPart × iNumMaps × iNumFrames × iNumComp × iHeight × iWidth, and
* a 1D array iNumPartChannelsMinus1, of size iNumPart, indicating the number of channels used in each partition.

Output of this process is:

* a 5D array, oFrames, of size iNumMaps × iNumFrames × iAttrDim × inputHeight × inputWidth.

The array oFrames is generated as follows:

outComp = 0  
 for( m = 0; m < iNumMap; m++) {  
 for( f = 0; f < iNumFrames; f++) {  
 oCompIdx = 0  
 for( p = 0; p < iNumPart; p++) {  
 for( c = 0; c < iNumPartChannelsMinus1[ p ] + 1; c++ ) {  
 for( h = 0; h < iHeight; h++ ) {  
 for( w = 0; w < iWidth; w++ ) {  
 oFrames[ m ][ f ][ oCompIdx ][ h ][ w ] =  
 iFrames[ p ][ m ][ f ][ c ][ j ][ i ]  
 }  
 }  
 oCompIdx++  
 }  
 }  
 }  
 }

* + - 1. Attribute dimension packing process for an auxiliary attribute

Inputs to this process are:

* a variable iNumPart, indicating the number of dimension partitions in which an attribute has been partitioned
* a variable iNumFrames, indicating the number of frames
* a variable iNumComp, indicating the number of components for each frame,
* a variable iHeight, indicating the height of each frame,
* a variable iWidth, indicating the width of each frame,
* a variable iAttrDim, indicating the dimensions of the attribute,
* a 5D array, iFrames, of size iNumPart × iNumFrames × iNumComp × iHeight × iWidth, and
* a 1D array iNumPartChannelsMinus1, of size iNumPart, indicating the number of channels used in each partition.

Output of this process is:

* a 4D array, oFrames, of size iNumFrames × iAttrDim × inputHeight × inputWidth.

The array oFrames is generated as follows:

outComp =  0  
 for( f = 0; f < iNumFrames; f++) {  
 oCompIdx = 0  
 for( p = 0; p < iNumPart; p++) {  
 for( c = 0; c < iNumPartChannelsMinus1[ p ] + 1; c++ ) {  
 for( h = 0; h < iHeight; h++ ) {  
 for( w = 0; w < iWidth; w++ ) {  
 oFrames[ f ][ oCompIdx ][ h ][ w ] =  
 iFrames[ p ][ f ][ c ][ j ][ i ]  
 }  
 }  
 oCompIdx++  
 }  
 }  
 }

* + 1. Chroma up-sampling

This process can be invoked when an attribute with identifier ATTR\_TEXTURE has three components with a chroma format indicated by decAttrChromaFormat other than 4:4:4, and the reconstruction system has selected to perform chroma format conversion to a 4:4:4 chroma format.

NOTE 1 – This process may also be invoked for an individual partition of an attribute with identifier ATTR\_TEXTURE when that partition has three components with a chroma format indicated by decAttrChromaFormat other than 4:4:4.

Inputs to this process are:

* a variable inputHeight, indicating the height of inputChromaSamples,
* a variable inputWidth, indicating the width of inputChromaSamples,
* a variable nominalBitDepth, indicating the chroma bit depth,
* a variable chromaFormat,
* a variable chromaSamplingPosition, and
* a 3D array inputChromaSamples, of size iNumComp × inputHeight × inputWidth.

Output of this process is:

* a 3D array outputChromaSamples, of size iNumComp × inputHeight × inputWidth, of up‑sampled chroma values.

Assuming that the chromaFormat indicates the 4:2:0 chroma format and the chromaSamplingPosition is equal to 0, an example chroma conversion process is specified as follows:

Table B-1 specifies the 4-tap filter coefficients fC[ p, x ] with p = 0..1 and x = 0..3 that is used for up‑sampling by a factor of 2. The same filter is applied both vertically and horizontally.

**Table B-1** – **4-phase chroma resampling filter**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Phase p** | **Interpolation coefficients** | | | |
| **fc[ p, 0 ]** | **fc[ p, 1 ]** | **fc[ p, 2 ]** | **fc[ p, 3 ]** |
| 0 | 0 | 16 | 0 | 0 |
| 1 | −1 | 9 | 9 | −1 |

NOTE 2 – This is essentially a "Lanczos 2" filter. Higher-precision and higher-order filters could potentially be used, especially when up-sampling content of very high quality or with no compression.

The chroma values are up-sampled as follows:

shift = 8  
 offset = 1 << ( shift − 1 )  
 for( c = 1; c < 3 ; c++ ) { for( y = 0; y < inputHeight ; y++ ) {  
 yRef = ( y >> 1 )  
 yPhase = y % 2  
 for( x = 0; x < inputWidth ; x++ ) {  
 xRef = ( x >> 1 )  
 xPhase = x % 2  
 usChromaSample = 0  
 for( n = 0; n < 4; n++ ) {  
 yPos = Clip3( 0, ( inputHeight >> 2 ) − 1, yRef + n – 1 )  
 temp = 0  
 for( m = 0; m < 4; m++ ) {  
 xPos = Clip3( 0, ( inputWidth >> 2 ) − 1, xRef + m – 1 )  
 temp += fC[ xPhase, m ] \* inputChromaSample[ c ][ yPos ][ xPos ]  
 }  
 usChromaSample += fC[ yPhase, 0 ] \* temp  
 }  
 usChromaSample = ( usChromaSample + offset ) >> shift  
 outputChromaSample[ c ][ y ][ x ] =  
 Clip3( 0, ( 1 << nominalBitDepth ) − 1, usChromaSample )  
 }  
 }  
 }

NOTE 3 – Instead of clipping the chroma output to the nominal bit depth for the attribute, a point cloud reconstruction system may choose to store the attribute in 4:4:4 chroma format at a higher bit depth.

For other chroma formats or chroma sampling positions different processes could be used that are outside the scope of this document.

* + 1. Geometry map synthesis process

Inputs to this process are:

* a variable multipleMapStreamsPresentFlag, indicating the presence of multiple streams,
* a variable mapCount, indicating the number of maps
* a variable numInputFrames, indicating the number of input frames,
* a variable iBitDepth, indicating the decoded geometry frames bit depth,
* a variable iNumComp, indicating the number of components for each frame,
* a variable iHeight, indicating the height of each frame,
* a variable iWidth, indicating the width of each frame,
* a 5D array iFrames of size mapCount × numInputFrames × iNumComp × iHeight × iWidth, and
* a 1D array mapAbsoluteCodingFlag, of size mapCount.

Output of this process is:

* a 5D array oFrames, of size mapCount × numInputFrames × iNumComp × iHeight × iWidth.

The geometry video map synthesis process is performed as follows:

for( f = 0; f < numInputFrames; f++ ) {  
 if( multipleMapStreamsPresentFlag == 0 ) {  
 for( i = 0; i < mapCount; i++ ) {  
 for( c = 0; c < iNumComp ; y++ ) {  
 for( y = 0; y < iHeight ; y++ ) {  
 for( x = 0; x < iWidth ; x++ ) {  
 oFrames[ i ][ f ][ c ][ y ][ x ] = iFrames[ i ][ f ][ c ][ y ][ x ]  
 }  
 }  
 }  
 }  
 } else {  
 geoMaxValue = ( 1 << iBitDepth ) – 1  
 for( i = 0; i < mapCount; i++ ) {  
 if( mapAbsoluteCodingFlag[ i ] == 0) {  
 j = MapPredictorIndex[ i ]  
 for( c = 0; c < iNumComp ; y++ ) {  
 for( y = 0; y < iHeight ; y++ ) {  
 for( x = 0; x < iWidth ; x++ ) {  
 oFrames[ i ][ f ][ c ][ y ][ x ] =  
 Clip3( 0, geoMaxValue, iFrames[ i ][ f ][ c ][ y ][ x ] +  
 oFrames[ j ][ f ][ c ][ y ][ x ] )  
 }  
 }  
 }  
 } else {  
 for( c = 0; c < iNumComp ; y++ ) {  
 for( y = 0; y < iHeight ; y++ ) {  
 for( x = 0; x < iWidth ; x++ ) {  
 oFrames[ i ][ f ][ 0 ][ y ][ x ] = iFrames[ i ][ f ][ 0 ][ y ][ x ]  
 }  
 }  
 }  
 }  
 }  
 }  
 }

* + 1. Attribute map synthesis process

Inputs to this process are:

* a variable multipleMapStreamsPresentFlag, indicating the presence of multiple streams,
* a variable mapCount, indicating the number of maps
* a variable numInputFrames, indicating the number of input frames,
* a variable iBitDepth, indicating the decoded geometry frames bit depth,
* a variable iNumComp, indicating the number of components for each frame,
* a variable iHeight, indicating the height of each frame,
* a variable iWidth, indicating the width of each frame,
* a 5D array iFrames of size mapCount × numInputFrames × iNumComp × iHeight × iWidth, and
* a 1D array mapAbsoluteCodingFlag, of size mapCount.

Outputs of this process are:

* a 5D array oFrames, of size mapCount × numInputFrames × iNumComp × iHeight × iWidth.

The attribute video map synthesis process is performed as follows:

for( f = 0; f < numInputFrames; f++ ) {  
 if( multipleMapStreamsPresentFlag == 0 ) {  
 for( m = 0; m < mapCount; m++ ) {  
 for( c = 0; c < iNumComp ; c++ ) {  
 for( y = 0; y < iHeight ; y++ ) {  
 for( x = 0; x < iWidth ; x++ ) {  
 oFrames[ m ][ f ][ c ][ y ][ x ] =iFrames[ m ][ f ][ c ][ y ][ x ]  
 }  
 }  
 }  
 }  
 } else {  
 attrMaxValue = ( 1 << iBitDepth ) – 1  
 attrOffset = 1 << (iBitDepth – 1 )  
 for( i = 0; i < mapCount; i++ ) {  
 if( mapAbsoluteCodingFlag[ i ] == 0 ) {  
 j = MapPredictorIndex[ i ]  
 for( c = 0; c < iNumComp; c++ )  
 for( y = 0; y < iHeight ; y++ )  
 for( x = 0; x < iWidth ; x++ )  
 oFrames[ i ][ f ][ c ][ y ][ x ] =  
 Clip3( 0, attrMaxValue, – attrOffset +  
 iFrames[ i ][ f ][ c ][ y ][ x ] +  
 oFrames[ j ][ f ][ c ][ y ][ x ] )  
 } else {  
 for( c = 0; c < iNumComp; c++ ) {  
 for( y = 0; y < iHeight ; y++ ) {  
 for( x = 0; x < iWidth ; x++ ) {  
 oFrames[ i ][ f ][ c ][ y ][ x ] = iFrames[ i ][ f ][ c ][ y ][ x ]  
 }  
 }  
 }  
 }  
 }  
 }  
 }

* + 1. Occupancy thresholding process

Inputs to this process are:

* a variable inputThreshold, indicating the threshold value,
* a variable inputBitDepth, indicating the bit depth of inputSamples,
* a variable inputHeight, indicating the height of inputSamples,
* a variable inputWidth, indicating the width of inputSamples, and
* a 2D array inputSamples, of size inputHeight × inputWidth.

Output of this process is:

* a 2D array outputSamples, of size inputHeight × inputWidth.

The occupancy thresholding process is performed as follows:

lossyThreshold = inputThreshold << ( inputBitDepth – 8 )  
  
 for( y=0; y < inputHeight; y++ )  
 for( x=0; x < inputWidth; x++ )  
 if( inputSamples[ y ][ x ] <= lossyThreshold )  
 outputSamples[ y ][ x ] = 0  
 else  
 outputSamples[ y ][ x ] = 1

* 1. Unpacking process of a decoded packed video

[Ed.(BK): This text is copied verbatim from ISO/IEC CD 23090-12 and needs to be aligned and improved.]

[Ed.(BK): The process has to be split in sub-processes for readability.]

[Ed. (JB): check the derivation of the scaling factors and the use of TileOffsetX & Y to determine the width and height]

For an atlas with ID equal to k and frame index equal to frameIdx, inputs to this process are:

* packed information syntax elements, as specified in clause 8.3.4.7
* tile information TileWidth, TileHeight, TileOffsetX, TileOffsetY,
* the outputs of the packed video decoding process, as specified in clause 9.6

Outputs of this process are:

* the decoded occupancy frame decOccFrame[ frameIdx ][ compIdx ][  ][  ], if present, and its associated information:
* bit depth, decOccBitDepth[ frameIdx ]
* width, decOccWidth[ frameIdx ],
* height, decOccHeight[ frameIdx ],
* output order index, decOccOutOrdIdx[ frameIdx ], and
* composition time, decOccCompTime[ frameIdx ]
* the decoded geometry frame decGeoFrame[ mapIdx ][ frameIdx ][ compIdx ][  ][  ], if present, and its associated information
* bitdepth, decGeoBitdepth[ mapIdx ][ frameIdx ],
* width, decGeoWidth[ mapIdx ][ frameIdx ],
* height, decGeoHeight[ mapIdx ][ frameIdx ],
* output order index, decGeoOutOrdIdx[ frameIdx ], and
* composition time, decGeoCompTime[ frameIdx ]
* the decoded auxiliary geometry frame decGeoAuxFrame[ frameIdx ][ compIdx ][  ][  ], if present, and its associated information
* bitdepth, decGeoAuxBitdepth[ frameIdx ],
* width, decGeoAuxWidth[ frameIdx ],
* height, decGeoAuxHeight[ frameIdx ],
* output order index decGeoAuxOutOrdIdx[ frameIdx ], and
* composition time, decGeoAuxCompTime[ frameIdx ]
* the decoded attribute frame decAttrFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ][  ][  ], if present, and its associated information
* bit depth, decAttrBitDepth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ],
* width, decAttrWidth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ],
* height, decAttrHeight[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ],
* output order index, decAttrOutOrdIdx[ frameIdx ], and
* composition time, decAttrCompTime[ frameIdx ]
* the decoded auxiliary attribute frame decAttrAuxFrame[ attrIdx ][ partIdx ][ frameIdx ][ compIdx ][  ][  ], if present, and its associated information
* bit depth, decAttrAuxBitDepth[ attrIdx ][ partIdx ][ frameIdx ],
* width, decAttrAuxWidth[ attrIdx ][ partIdx ][ frameIdx ],
* height, decAttrAuxHeight[ attrIdx ][ partIdx ][ frameIdx ],
* output order index, decAttrAuxOutOrdIdx[ frameIdx ], and
* composition time, decAttrAuxCompTime[ frameIdx ],

To unpacked the decoded packed video frame the following process apply.

decOccWidth[ frameIdx ] = 0   
decOccHeight[ frameIdx ] = 0  
decGeoWidth[ mapIdx ][ frameIdx ] = 0  
decGeoHeight[ mapIdx ][ frameIdx ] = 0  
decGeoAuxWidth[ frameIdx ] = 0  
decGeoAuxHeight[ frameIdx ] = 0  
decAttrWidth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = 0  
decAttrHeight[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = 0  
decAttrAuxWidth[ frameIdx ] = 0  
decAttrAuxHeight[ frameIdx ] = 0

for( i = 0; i <= pin\_num\_region\_minus1[ i ]; i++ ) {  
 mapIdx = pin\_region\_map\_index[ k ][ i ]  
 attrIdx = pin\_region\_attr\_type\_id[ k ][ i ]  
 partIdx = pin\_region\_attr\_partition\_index[ k ][ i ]  
 auxDataFlag = pin\_region\_auxiliary\_data\_flag[ k ][ i ]  
 regionOffsetX = pin\_region\_top\_left\_x[ k ][ i ]  
 regionOffsetY = pin\_region\_top\_left\_y[ k ][ i ]  
 regionWidth = pin\_region\_width\_minus1[ k ][ i ] + 1  
 regionHeight = pin\_region\_height\_minus1[ k ][ i ] + 1  
 tileID = pin\_region\_tile\_id[ k ][ i ]  
 tileWidth = TileWidth[ TileIdToIndex[ tileID ] ]  
 tileHeight = TileHeight[ TileIdToIndex[ tileID ] ]  
   
 if( pin\_region\_rotation\_flag[ k ][ i] = = 0 ) {  
 scaleX = tileWidth / regionWidth   
 scaleY = tileHeight / regionHeight   
 } else {  
 scaleX = tileWidth / regionHeight   
 scaleY = tileHeight / regionWidth   
 }  
   
 offsetX = TileOffsetX[ TileIdToIndex[ tileID ] ] / scaleX   
 offsetY = TileOffsetY[ TileIdToIndex[ tileID ] ] / scaleY   
   
 if( pin\_region\_rotation\_flag[ k ][ i ] = = 0 ) {  
 tempWidth = offsetX + regionWidth   
 tempHeight = offsetY + regionHeight  
 } else {  
 tempWidth = offsetX + regionHeight  
 tempHeight = offsetY + regionWidth  
 }  
   
 if( pin\_region\_type\_id\_minus2[ k ][ i ] + 2 = = V3C\_OVD ) {  
 if( tempWidth > decOccWidth[ frameIdx ] )  
 decOccWidth[ frameIdx ] = tempWidth   
 if( tempHeight > decOccHeight[ frameIdx ] )  
 decOccHeight[ frameIdx ] = tempHeight   
 }  else if (pin\_region\_type\_id\_minus2[ k ][ i ] +2 = = V3C\_GVD && auxDataFlag = = 0 ) {  
 if( tempWidth > decGeoWidth[ mapIdx ][ frameIdx ] )  
 decGeoWidth[ mapIdx ][ frameIdx ] = tempWidth   
 if( tempHeight > decGeoHeight[ mapIdx ][ frameIdx ] )  
 decGeoHeight[ mapIdx ][ frameIdx ] = tempHeight   
 }  else if (pin\_region\_type\_id\_minus2[ k ][ i ] +2 = = V3C\_GVD && auxDataFlag = = 1 ) {  
 if( tempWidth > decGeoAuxWidth[ frameIdx ] )  
 decGeoAuxWidth[ frameIdx ] = tempWidth   
 if( tempHeight > decGeoAuxHeight[ frameIdx ] )  
 decGeoAuxHeight[ frameIdx ] = tempHeight   
 }  else if (pin\_region\_type\_id\_minus2[ k ][ i ] +2 = = V3C\_AVD && auxDataFlag = = 0 ) {  
 if( tempWidth > decAttrWidth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ]  )  
 decAttrWidth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = tempWidth   
 if( tempHeight > decAttrHeight[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] )  
 decAttrHeight[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = tempHeight   
 }  else if ( pin\_region\_type\_id\_minus2[ k ][ i ] +2 = = V3C\_AVD && auxDataFlag = = 1 ){  
 if( tempWidth > decAttrAuxWidth[ frameIdx ] )  
 decAttrAuxWidth[ frameIdx ] = tempWidth   
 if( tempHeight > decAttrAuxHeight[ frameIdx ] )  
 decAttrAuxHeight[ frameIdx ] = tempHeight   
 }  
  
 for( y = 0; y < regionHeight; y++ ) {  
 for( x = 0; x < regionWidth ; x++ ) {  
 for( compIdx = 0; compIdx < 3 ; compIdx ++ ) {  
 sample = decPackedFrame[ frameIdx ][ compIdx ][ regionOffsetY + y ][ regionOffsetX + x ]   
  
 if( pin\_region\_type\_id\_minus2[ k ][ i ] + 2 =  V3C\_OVD && compIdx = = 0 ) {  
 if( pin\_region\_rotation\_flag[ k ][ i] = = 0) {  
 decOccFrame[ mapIdx ][ frameIdx ][ 0 ][ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decOccFrame[ mapIdx ][ frameIdx ][ 0 ][ offsetY + x ][ offsetX + y ] = sample  
 }  
  
 } else if ( pin\_region\_type\_id\_minus2[ k ][ i ] +2 = = V3C\_GVD &&   
 compIdx = = 0 && auxDataFlag = = 0 ) {  
 if( pin\_region\_rotation\_flag[ k ][ i] = = 0) {  
 decGeoFrame [ atlasIdx ][ mapIdx ][ frameIdx ][ 0 ][ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decGeoFrame[ atlasIdx ][ mapIdx ][ frameIdx ][ 0 ][ offsetY + x ][ offsetX + y ] = sample  
 }  
  
 } else if ( pin\_region\_type\_id\_minus2[ k ][ i ] +2 = = V3C\_GVD &&   
 compIdx = = 0  && auxDataFlag = = 1 ){  
 if( pin\_region\_rotation\_flag[ k ][ i] = = 0 ) {  
 decGeoAuxFrame[ atlasIdx ][ mapIdx ][ frameIdx ][ 0 ][ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decGeoAuxFrame[ atlasIdx ][ mapIdx ][ frameIdx ][ 0 ][ offsetY + x ][ offsetX + y ] = sample  
 }  
  
 } else if (pin\_region\_type\_id\_minus2[ k ][ i ] + 2 = = V3C\_AVD && auxDataFlag = = 0 ) {  
 attrIdx = pin\_region\_attr\_type\_id[ k ][ i ]  
 decAttrType[ attrIdx ] = pin\_region\_attr\_type\_id[ k ][ i ]  
 partIdx = pin\_region\_attr\_partition\_index[ k ][ i ]  
 if( pin\_region\_rotation\_flag[ k ][ i] = = 0 ) {  
 decAttrFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ]  
 [ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decAttrFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ]  
 [ offsetY + x ][ offsetX + y ] = sample  
 }  
  
 } else if (pin\_region\_type\_id\_minus2[ k ][ i ] + 2 = = V3C\_AVD && auxDataFlag = = 0 ) {  
 attrIdx = pin\_region\_attr\_type\_id[ k ][ i ]  
 decAttrAuxType[ attrIdx ] = pin\_region\_attr\_type\_id[ k ][ i ]  
 partIdx = pin\_region\_attr\_partition\_index[ k ][ i ]  
 if( pin\_region\_rotation\_flag[ k ][ i] = = 0) {  
 decAttrAuxFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ]  
 [ offsetY + y ][ offsetX + x ] = sample  
 } else {  
 decAttrAuxFrame[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ][ compIdx ]   
 [ offsetY + x ][ offsetX + y ] = sample  
 }  
 }  
 }  
 }  
 }  
}

decOccBitdepth[ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decOccCompTime[ frameIdx ] = decPackedCompTime[ frameIdx ]   
decOccOutOrdIdx[ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]   
  
decGeoBitdepth[ mapIdx ][ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decGeoCompTime[ mapIdx ][ [ frameIdx ] = decPackedCompTime[ frameIdx ]   
decGeoOutOrdIdx[ mapIdx ][ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]   
  
decGeoAuxBitdepth[ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decGeoAuxCompTime[ frameIdx ] = decPackedCompTime[ frameIdx ]   
decGeoAuxOutOrdIdx[ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]   
  
decAttrBitdepth[ attrIdx ][ mapIdx ][ partIdx ][ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decAttrCompTime[ frameIdx ] = decPackedCompTime[ frameIdx ]   
decAttrOutOrdIdx[ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]   
  
decAttrAuxBitdepth[ attrIdx ][ partIdx ][ frameIdx ] = decPackedBitdepth[ frameIdx ]   
decAttrAuxCompTime[ frameIdx ] = decPackedCompTime[ frameIdx ]   
decAttrOutOrdIdx[ frameIdx ] = decPackedOutOrdIdx[ frameIdx ]

1. (informative)  
   V3C sample stream format
   1. General

This annex specifies syntax and semantics of a sample stream format specified for use by applications that deliver some or all of the V3C unit stream as an ordered stream of bytes or bits within which the locations of V3C unit boundaries need to be identifiable from patterns in the data, such as Rec. ITU-T H.222.0 | ISO/IEC 13818-1 systems or Recommendation ITU‑T H.320 systems. For bit-oriented delivery, the bit order for the sample stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The sample stream format starts with a sample stream header and consists of a sequence of sample stream V3C unit syntax structures. Each sample stream V3C unit syntax structure contains one element, named ssvu\_v3c\_unit\_size,that specifies a size, followed by one v3c\_unit( ssvu\_v3c\_unit\_size ) syntax structure. This v3c\_unit is essentially of size ssvu\_v3c\_unit\_size. The number of bits used to encode the element ssvu\_v3c\_unit\_size is specified in the sample stream header.

* 1. Sample stream V3C unit syntax and semantics
     1. Sample stream V3C header syntax

|  |  |
| --- | --- |
| sample\_stream\_v3c\_header( ) { | **Descriptor** |
| **ssvh\_unit\_size\_precision\_bytes\_minus1** | u(3) |
| **ssvh\_reserved\_zero\_5bits** | u(5) |
| } |  |

* + 1. Sample stream V3C unit syntax

|  |  |
| --- | --- |
| sample\_stream\_v3c\_unit( ) { | **Descriptor** |
| **ssvu\_v3c\_unit\_size** | u(v) |
| v3c\_unit(ssvu\_v3c\_unit\_size) |  |
| } |  |

* + 1. Sample stream V3C header semantics

The sample stream V3C header shall always be at the beginning of the sample stream V3C stream.

**ssvh\_unit\_size\_precision\_bytes\_minus1** plus 1 specifies the precision, in bytes, of the ssvu\_v3c\_unit\_size element in all sample stream V3C units. ssvh\_unit\_size\_precision\_bytes\_minus1 shall be in the range of 0 to 7.

**ssvh\_reserved\_zero\_5bits** shall be equal to 0 in bitstreams conforming to this version of this document. Other values for sshv\_reserved\_zero\_5bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of sshv\_reserved\_zero\_5bits.

* + 1. Sample stream V3C unit semantics

The order of sample stream V3C units in the sample stream shall follow the decoding order of the V3C units contained in the sample stream V3C units (see subclause 8.4.2.5.3).

**ssvu\_v3c\_unit\_size** specifies the size, in bytes, of the subsequent v3c\_unit. The number of bits used to represent ssvu\_v3c\_unit\_size is equal to ( ssvh\_unit\_size\_precision\_bytes\_minus1 + 1 ) \* 8.

1. (normative)  
   NAL sample stream format
   1. General

This annex specifies syntax and semantics of a sample stream format specified for use by applications that deliver some or all of the atlas NAL unit stream as an ordered stream of bytes or bits within which the locations of atlas NAL unit boundaries need to be identifiable from patterns in the data, such as Rec. ITU-T H.222.0 | ISO/IEC 13818-1 systems or Recommendation ITU‑T H.320 systems. For bit-oriented delivery, the bit order for the sample stream format is specified to start with the MSB of the first byte, proceed to the LSB of the first byte, followed by the MSB of the second byte, etc.

The sample stream format starts with a sample stream header and consists of a sequence of sample stream NAL unit syntax structures. Each sample stream NAL unit syntax structure contains one element, named ssnu\_nal\_unit\_size,that specifies a size, followed by one nal\_unit( ssnu\_nal\_unit\_size ) syntax structure. This nal\_unit is essentially of size ssnu\_nal\_unit\_size. The number of bits used to encode the element ssnu\_nal\_unit\_size is specified in the sample stream header.

* 1. Sample stream NAL unit syntax and semantics
     1. Sample stream NAL header syntax

|  |  |
| --- | --- |
| sample\_stream\_nal\_header( ) { | **Descriptor** |
| **ssnh\_unit\_size\_precision\_bytes\_minus1** | u(3) |
| **ssnh\_reserved\_zero\_5bits** | u(5) |
| } |  |

* + 1. Sample stream NAL unit syntax

|  |  |
| --- | --- |
| sample\_stream\_nal\_unit( ) { | **Descriptor** |
| **ssnu\_nal\_unit\_size** | u(v) |
| nal\_unit(ssnu\_nal\_unit\_size) |  |
| } |  |

* + 1. Sample stream NAL header semantics

The sample stream NAL header shall always be at the beginning of the NAL stream.

**ssnh\_unit\_size\_precision\_bytes\_minus1** plus 1 specifies the precision, in bytes, of the ssnu\_nal\_unit\_size element in all sample stream NAL units. ssnh\_unit\_size\_precision\_bytes\_minus1 shall be in the range of 0 to 7.

**ssnh\_reserved\_zero\_5bits** shall be equal to 0 in bitstreams conforming to this version of this document. Other values for ssnh\_reserved\_zero\_5bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of ssnh\_reserved\_zero\_5bits.

* + 1. Sample stream NAL unit semantics

The order of sample stream NAL units in the sample stream shall follow the decoding order of the NAL units contained in the sample stream NAL units (see subclause 8.4.5.3). The content of each sample stream NAL unit is associated with the same coded atlas access unit as the NAL unit contained in the sample stream NAL unit (see subclause 8.4.5.3).

**ssnu\_nal\_unit\_size** specifies the size, in bytes, of the subsequent NAL\_unit. The number of bits used to represent ssnu\_nal\_unit\_size is equal to ( ssnh\_unit\_size\_precision\_bytes\_minus1 + 1 ) \* 8.

1. (normative)  
   Atlas hypothetical reference decoder
   1. General

This annex specifies the hypothetical reference decoder (HRD) and its use to check atlas sub-bitstream and decoder conformance. In this section, a bitstream refers to an atlas sub-bitstream unless otherwise indicated.

Two types of bitstreams or bitstream subsets are subject to HRD conformance checking for this document. The first type, called a Type I bitstream, is a NAL unit stream containing the ACL NAL units and non-ACL NAL units with nal\_unit\_type equal to NAL\_FD (filler data NAL units) for all coded atlas access units in the atlas sub-bitstream. The second type, called a Type II bitstream, is a NAL unit stream contains, in addition to the ACL NAL units and and filler data NAL units for all coded atlas access units in the atlas sub‑bitstream, at least one of the following:

* additional non-ACL NAL units other than filler data NAL units,
* the sample stream nal header structure syntax elements, ssnh\_unit\_size\_precision\_bytes\_minus1

and ssnh\_reserved\_zero\_5bits, and all ssnu\_nal\_unit\_size syntax elements that form a NAL sample stream from the NAL unit stream (as specified in Annex D).

Figure E-1 shows the types of bitstream conformance points checked by the atlas sub-bitstream HRD.



Figure E-1 – Structure of atlas NAL unit streams for HRD conformance checks

The syntax elements of non-ACL NAL units and their default values, when applicable, required for the HRD, are specified in Clause 8, Annex F, and Annex G.

Two types of HRD parameter sets (NAL HRD parameters and non-ACL HRD parameters) are used. The HRD parameter sets are signalled through the hrd\_parameters( ) syntax structure, which is part of the VUI syntax structure specified in Annex G and the buffering period SEI message specified in subclauses F.2.13 and F.3.13. The first set, named as the entireBitstreamTestSet, is to test conformance of the entire bitstream. The second set, named as ACLTestSet, is to test conformance of the bitstream without including non-ACL units since they may not impact the decoding process or may be provided through external means.

Multiple tests may be needed for checking the conformance of an atlas sub-bitstream, AtlasBitstreamToDecode, which is referred to as the bitstream under test. For each test, the following steps apply in the order listed:

1. An operation point under test, denoted as TargetOp, is set by selecting a target highest TemporalID value OpTid. The value of OpTid is in the range of 0 to bp\_max\_sub\_layers\_minus1, inclusive. The value of OpTid is such that the sub-bitstream BitstreamToDecode satisfy both of the following conditions:
   * There is at least one ACL NAL unit in BitstreamToDecode with nal\_layer\_id equal to 0.
   * There is at least one ACL NAL unit with TemporalId equal to OpTid in BitstreamToDecode.
2. The value of Htid is set equal to opTid of TargetOp.
3. The hrd\_parameters( ) syntax structure in the VUI of the active ASPS (or provided through some external means not specified in this document) that applies to TargetOp is selected.

Within the selected hrd\_parameters( ) syntax structure, if AtlasBitstreamToDecode is a Type I bitstream, the hrd\_sub\_layer\_parameters( 1, Htid ) syntax structure that immediately follows the condition "if( hrd\_acl\_parameters\_present\_flag )" is selected and the variable NalHrdModeFlag is set equal to 0.

Otherwise (AtlasBitstreamToDecode is a Type II bitstream), when using the ACLTestSet, the variable NalHrdModeFlag is first set to 0. Then the parameters corresponding to the hrd\_sub\_layer\_parameters( 1, Htid ) syntax structure that immediately follows the condition "if( hrd\_acl\_parameters\_present\_flag )" are selected and used. Otherwise, when using the entireBitstreamTestSet, the variable NalHrdModeFlag is set equal to 1 and the parameters corresponding to the hrd\_sub\_layer\_parameters( 0, Htid ) syntax structure that immediately follows the condition "if( hrd\_nal\_parameters\_present\_flag )" are selected and used.

When AtlasBitstreamToDecode is a Type II bitstream and NalHrdModeFlag is equal to 0, all non-ACL NAL units except filler data NAL units and all the syntax elements that form a sample stream from the NAL unit stream, as specified in Annex D, when present, are discarded from AtlasBitstreamToDecode and the remaining atlas bitstream is assigned to AtlasBitstreamToDecode.

1. A coded atlas access unit associated with a buffering period SEI message, present in AtlasBitstreamToDecode or available through external means not specified in this document, applicable to TargetOp, is selected as the HRD initialization point and referred to as the coded atlas access unit 0.
2. For each coded atlas access unit in AtlasBitstreamToDecode starting from coded atlas access unit 0, the buffering period SEI message and the atlas frame timing SEI message, present in AtlasBitstreamToDecode or available through external means not specified in this document, that are associated with the coded atlas access unit and applies to TargetOp, are selected.
3. A value of SchedSelIdx is selected. The selected value of SchedSelIdx shall be in the range of 0 to hrd\_cab\_cnt\_minus1[ Htid ], inclusive, where hrd\_cab\_cnt\_minus1[ Htid ] is found in the buffering period SEI syntax structure, as specified in F.2.13.
4. When the coded atlas frame in access unit 0 has nal\_unit\_type equal to NAL\_CRA, NAL\_GCRA, NAL\_BLA\_W\_LP, or NAL\_GBLA\_W\_LP, and bp\_irap\_cab\_params\_present\_flag in the selected buffering period SEI message is equal to 1, either of the following applies for the selection of the initial CAB removal delay and delay offset:

* If NalHrdModeFlag is equal to 1, the default initial CAB removal delay and the delay offset represented by bp\_nal\_initial\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_nal\_initial\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, in the selected buffering period SEI message are selected. Otherwise, the default initial CAB removal delay and delay offset represented by bp\_acl\_initial\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_acl\_initial\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, in the selected buffering period SEI message are selected. The variable DefaultInitCabParamsFlag is set equal to 1.
* If NalHrdModeFlag is equal to 1, the alternative initial CAB removal delay and delay offset represented by bp\_nal\_initial\_alt\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_nal\_initial\_alt\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, in the selected buffering period SEI message are selected. Otherwise, the alternative initial CAB removal delay and delay offset represented by bp\_acl\_initial\_alt\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_acl\_initial\_alt\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, in the selected buffering period SEI message are selected. The variable DefaultInitCabParamsFlag is set equal to 0, and the RASL coded atlas access units associated with the coded atlas access unit 0 are discarded from AtlasBitstreamToDecode and the remaining bitstream is assigned to AtlasBitstreamToDecode.

Each conformance test consists of a combination of one option in each of the above steps. When there is more than one option for a step, for any conformance test, only one option is chosen. All possible combinations of all the steps form the entire set of conformance tests. For each operation point under test, the number of bitstream conformance tests to be performed is equal to n0 \* n1 \* ( n2 \* 2 + n3 ) \* n4, where the values of n0, n1, n2, n3, and n4 are specified as follows:

* n0 is derived as follows:
* If AtlasBitstreamToDecode is a Type I bitstream, n0 is equal to 1.
* Otherwise (AtlasBitstreamToDecode is a Type II bitstream), n0 is equal to 2.
* n1 is equal to hrd\_cab\_cnt\_minus1[ Htid ] + 1.
* n2 is the number of coded atlas access units in AtlasBitstreamToDecode that each is associated with a buffering period SEI message applicable to TargetOp and for each of which both of the following conditions are true:
* nal\_unit\_type is equal to NAL\_CRA, NAL\_BLA\_W\_LP, NAL\_GCRA, or NAL\_GBLA\_W\_LP for the ACL NAL units.
* The associated buffering period SEI message applicable to TargetOp has bp\_irap\_cab\_params\_present\_flag equal to 1.
* n3 is the number of coded atlas access units in AtlasBitstreamToDecode that each is associated with a buffering period SEI message applicable to TargetOp and for each of which one or both of the following conditions are true:
* nal\_unit\_type is not equal to NAL\_CRA, NAL\_BLA\_W\_LP, NAL\_GCRA, or NAL\_GBLA\_W\_LP, or for the ACL NAL units.
* The associated buffering period SEI message applicable to TargetOp has bp\_irap\_cab\_params\_present\_flag equal to 0.
* n4 is equal to 1.

NOTE 1 – n0 corresponds to conformance tests for Type I bitstream conformance and Type II bitstream conformance. n1 corresponds to conformance tests for each CAB delivery schedule. n2 corresponds to conformance tests for bitstreams starting at each CRA coded atlas frames with associated RASL coded atlas frames and alternative initial CAB removal delay and delay offset present. These test are carried twice: once for bitstream keeping and once for bitstreams removing RASL coded atlas associated with the CRA. n3 corresponds to conformance tests for bitstreams starting at each IRAP coded atlas that is not a CRA with associated RASL coded atlas frames and alternative initial CAB removal delay and delay offset present. n4 corresponds to conformance tests for access unit based conformance.

When AtlasBitstreamToDecode is a Type II patch bitstream, the following applies:

* If the hrd\_sub\_layer\_parameters( 1, Htid ) syntax structure that immediately follows the condition "if( hrd\_acl\_parameters\_present\_flag )" is selected, the test is conducted at the Type I conformance point shown in Figure E-1, and only ACL and filler data NAL units are counted for the input bit rate and CAB storage.
* Otherwise (the hrd\_sub\_layer\_parameters( 0, Htid ) syntax structure that immediately follows the condition "if( hrd\_nal\_parameters\_present\_flag )" is selected), the test is conducted at the Type II conformance point shown in Figure E-1, and all bytes of the Type II bitstream, which is a NAL unit stream, are counted for the input bit rate and CAB storage.

NOTE 2 – NAL HRD parameters established by a value of SchedSelIdx for the Type II conformance point shown in   
Figure E-1 are sufficient to also establish ACL HRD conformance for the Type I conformance point shown in Figure E-1 for the same values of InitCabRemovalDelay[ Htid ][ SchedSelIdx ], BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] and CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] for the variable bit rate (VBR) case (hrd\_cbr\_flag[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] equal to 0). This is because the data flow into the Type I conformance point is a subset of the data flow into the Type II conformance point and because, for the VBR case, the CAB is allowed to become empty and stay empty until the time a next atlas frame is scheduled to begin to arrive. For example, when decoding a CAS that is associated with a CVS conforming to one or more of the profiles specified in Annex A using the decoding process specified in subclause 9.2.

All AAPS, ASPSs, and AFPSs referred to in the ACL NAL units and the corresponding buffering period and atlas frame timing SEI messages shall be conveyed to the HRD, in a timely manner, either in the bitstream (by non-ACL NAL units), or by other means not specified in this document.

In Annex E and Annex F, the specification for "presence" of non-ACL NAL units that contain AAPSs, ASPSs, AFPSs~~,~~ buffering period SEI messages, and atlas frame timing SEI messages is also satisfied when those NAL units (or just some of them) are conveyed to the atlas frame decoders (or to the HRD) by other means not specified in this document. For the purpose of counting bits, only the appropriate bits that are actually present in the atlas sub-bitstream are counted.

NOTE 3 – As an example, synchronization of such a non-ACL NAL unit, conveyed by means other than presence in the bitstream, with the NAL units that are present in the bitstream, can be achieved by indicating two points in the bitstream, between which the non‑ACL NAL unit would have been present in the bitstream, had the atlas frame encoder decided to convey it in the bitstream.

When the content of such a non-ACL NAL unit is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the non-ACL NAL unit is not required to use the same syntax as specified in this document.

NOTE 4 – When HRD information is contained within the bitstream, it is possible to verify the conformance of a bitstream to the requirements of this subclause based solely on information contained in the bitstream. When the HRD information is not present in the bitstream, as is the case for all "stand-alone" Type I bitstreams, conformance can only be verified when the HRD data is supplied by some other means not specified in this document.

The HRD contains a coded atlas buffer (CAB), an instantaneous decoding process, a decoded atlas buffer (DAB), and the normative output widow as shown in Figure E-2. The size of the normative output window is asps\_frame\_height × asps\_frame\_width.



Figure E-2 – Atlas frame HRD buffer model

For each atlas bitstream conformance test, the CAB size (number of bits) is CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] as specified in subclause G.3.3, where SchedSelIdx and the HRD parameters are specified in this subclause. The DAB size (number of atlas frames storage buffers) is asps\_max\_dec\_atlas\_frame\_buffering\_minus1 + 1.

The following is specified for expressing the constraints in this annex:

* Each coded atlas access unit is referred to as coded atlas access unit n, where the number n identifies the particular coded atlas access unit. Coded atlas access unit 0 is selected per step 4 as defined in this subclause. The value of n is incremented by 1 for each subsequent coded atlas access unit in decoding order.
* Atlas frame n refers to the coded atlas frame or the decoded atlas frame of coded atlas access unit n.

The HRD operates as follows:

* The HRD is initialized at decoding unit 0, with both the CAB and the DAB being set to be empty (the DAB fullness is set equal to 0).

NOTE 5 – After initialization, the HRD is not initialized again by subsequent buffering period SEI messages.

* Data associated with decoding units that flow into the CAB according to a specified arrival schedule are delivered by the hypothetical stream scheduler (HSS).
* The data associated with each decoding unit are removed and decoded instantaneously by the instantaneous decoding process at the CAB removal time of the decoding unit.
* Each decoded atlas frame is placed in the DAB.
* A decoded atlas frame is removed from the DAB when it becomes no longer needed for inter prediction reference and no longer needed for output.

For each bitstream conformance test, the operation of the CAB is specified in subclause E.2, the instantaneous decoder operation is specified in clauses 2 through 14, and the operation of the DAB is specified in subclause E.3.

HSS and HRD information concerning the number of enumerated delivery schedules and their associated bit rates and buffer sizes is specified in subclauses G.2.2 and G.3.2. The HRD is initialized as specified by the buffering period SEI message specified in subclauses F.2.13 and F.3.13. The removal timing of decoding units from the CAB and the output timing of decoded atlas frames from the DAB is specified using information in atlas frame timing SEI messages, as specified in subclauses F.2.14 and F.3.14. All timing information relating to a specific decoding unit shall arrive prior to the CAB removal time of the decoding unit.

The requirements for atlas sub-bitstream conformance are specified in subclause E.4, and the HRD is used to check conformance of atlas sub-bitstreams as specified in subclause E.1 and to check conformance of decoders as specified in subclause E.5.

NOTE 6 – While conformance is guaranteed under the assumption that all “atlas frame”-rates and clocks used to generate the atlas sub-bitstream match exactly the values signalled in the atlas sub-bitstream, in a real system each of these can vary from the signalled or specified value.

All the arithmetic in this annex is performed with real values, so that no rounding errors can propagate. For example, the number of bits in a CAB just prior to or after removal of a decoding unit is not necessarily an integer.

The variable ClockTick is derived as follows and is called a clock tick:

ClockTick = vui\_num\_units\_in\_tick ÷ vui\_time\_scale (E-1)

* 1. Operation of coded atlas frame buffer
     1. General

The specifications in this subclause apply independently to each set of coded atlas frame buffer (CAB) parameters that is present and to both the Type I and Type II conformance points shown in Figure E-1. The set of CAB parameters is selected as specified in subclause E.1.

* + 1. Timing of decoding unit arrival

The decoding unit is considered as a coded atlas access unit, for derivation of the initial and final CAB arrival times for coded atlas access unit n.

The variables InitCabRemovalDelay[ Htid ][ SchedSelIdx ] and InitCabRemovalDelayOffset[ Htid ][ SchedSelIdx ] are derived as follows:

* If one or more of the following conditions are true, InitCabRemovalDelay[ Htid ][ SchedSelIdx ] and InitCabRemovalDelayOffset[ Htid ][ SchedSelIdx ] are set equal to the values of the buffering period SEI message syntax elements bp\_nal\_initial\_alt\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_nal\_initial\_alt\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, when NalHrdModeFlag is equal to 1 or bp\_acl\_initial\_alt\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_acl\_initial\_alt\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message syntax elements are selected as specified in subclause E.1:
* Coded atlas access unit 0 is a BLA access unit, for which the coded atlas frame has nal\_unit\_type equal to NAL\_BLA\_W\_RADL, NAL\_BLA\_N\_LP, NAL\_GBLA\_W\_RADL, or NAL\_GBLA\_N\_LP and the value of bp\_irap\_cab\_params\_present\_flag of the buffering period SEI message is equal to 1.
* Coded atlas access unit 0 is a BLA access unit, for which the coded picture has nal\_unit\_type equal to NAL\_BLA\_W\_LP or NAL\_GBLA\_W\_LP, or is a CRA coded atlas access unit, and the value of bp\_irap\_cab\_params\_present\_flag of the buffering period SEI message is equal to 1 and one or more of the following conditions are true:
* UseAltCabParamsFlag for coded atlas access unit 0 is equal to 1.
* DefaultInitCabParamsFlag is equal to 0.
* Otherwise, InitCabRemovalDelay[ Htid ][ SchedSelIdx ] and InitCabRemovalDelayOffset[ Htid ][ SchedSelIdx ] are set equal to the values of the buffering period SEI message syntax elements bp\_nal\_initial\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_nal\_initial\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, when NalHrdModeFlag is equal to 1, or bp\_acl\_initial\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_acl\_initial\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message syntax elements are selected as specified in subclause E.1.

The time at which the first bit of decoding unit m begins to enter the CAB is referred to as the initial arrival time initArrivalTime[ m ].

The initial arrival time of decoding unit m is derived as follows:

* If the decoding unit is decoding unit 0 (i.e., when m is equal to 0), initArrivalTime[ 0 ] is set equal to 0.
* Otherwise (the decoding unit is decoding unit m with m > 0), the following applies:

If hrd\_cbr\_flag[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] is equal to 1, the initial arrival time for decoding unit m is equal to the final arrival time (which is derived below) of decoding unit m – 1, i.e.,

initArrivalTime[ m ] = AuFinalArrivalTime[ m − 1 ] (E‑2)

Otherwise (hrd\_cbr\_flag[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] is equal to 0), the initial arrival time for decoding unit m is derived as follows:

* initArrivalTime[ m ] =   
   Max( AuFinalArrivalTime[ m − 1 ], initArrivalEarliestTime[ m ] ) (E‑3)

where initArrivalEarliestTime[ m ] is derived as follows:

* The variable tmpNominalRemovalTime is derived as follows:
* tmpNominalRemovalTime = AuNominalRemovalTime[ m ] (E‑4)

where AuNominalRemovalTime[ m ] is the nominal CAB removal time of coded atlas access unit m respectively, as specified in subclause E.2.3.

* + - * If decoding unit m is not the first decoding unit of a subsequent buffering period, initArrivalEarliestTime[ m ] is derived as follows:

initArrivalEarliestTime[ m ] = tmpNominalRemovalTime −  
 ( InitCabRemovalDelay[ Htid ][ SchedSelIdx ] +  
 InitCabRemovalDelayOffset[ Htid ][ SchedSelIdx ] ) ÷ 90000 (E‑5)

* + - * Otherwise (decoding unit m is the first decoding unit of a subsequent buffering period), initArrivalEarliestTime[ m ] is derived as follows:

initArrivalEarliestTime[ m ] = tmpNominalRemovalTime −  
 ( InitCabRemovalDelay[ Htid ][ SchedSelIdx ] ÷ 90000 ) (E‑6)

The final arrival time for decoding unit m is derived as follows:

AuFinalArrivalTime[ m ] =  
 initArrivalTime[ m ] + sizeInBits[ m ] ÷ BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] (E‑7)

where sizeInBits[ m ] is the size in bits of decoding unit m, counting the bits of the ACL NAL units and the filler data NAL units for the Type I conformance point or all bits of the Type II bitstream for the Type II conformance point, where the Type I and Type II conformance points are as shown in Figure E-1.

The values of SchedSelIdx, BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ], and CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] are constrained as follows:

* If the content of the selected hrd\_parameters( ) syntax structures for the coded atlas access unit containing decoding unit m and the previous coded atlas access unit differ, the HSS selects a value SchedSelIdx1 of SchedSelIdx from among the values of SchedSelIdx provided in the selected hrd\_parameters( ) syntax structures for the coded atlas access unit containing decoding unit m that results in a BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx1 ] or CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx1 ] for the access unit containing decoding unit m. The value of BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx1 ] or CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx1 ] may differ from the value of BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx0 ] or CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx0 ] for the value SchedSelIdx0 of SchedSelIdx that was in use for the previous coded atlas access unit.
* Otherwise, the HSS continues to operate with the previous values of SchedSelIdx, BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ], and CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ].

When the HSS selects values of BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] or CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] that differ from those of the previous access unit, the following applies:

* The variable BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] comes into effect at the initial CAB arrival time of the current coded atlas access unit.
* The variable CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] comes into effect as follows:
* If the new value of CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] is greater than the old CAB size, it comes into effect at the initial CAB arrival time of the current access unit.
* Otherwise, the new value of CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] comes into effect at the CAB removal time of the current access unit.
  + 1. Timing of decoding unit removal and decoding of decoding unit

The variables InitCabRemovalDelay[ Htid ][ SchedSelIdx ], InitCabRemovalDelayOffset[ Htid ][ SchedSelIdx ], CabDelayOffset, and DabDelayOffset are derived as follows:

* If one or more of the following conditions are true, CabDelayOffset is set equal to the value of the buffering period SEI message syntax element bp\_cab\_delay\_offset, DabDelayOffset is set equal to the value of the buffering period SEI message syntax element bp\_dab\_delay\_offset, and InitCabRemovalDelay[ Htid ][ SchedSelIdx ] and InitCabRemovalDelayOffset[ Htid ][ SchedSelIdx ] are set equal to the values of the buffering period SEI message syntax elements bp\_nal\_initial\_alt\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_nal\_initial\_alt\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, when NalHrdModeFlag is equal to 1, or bp\_acl\_initial\_alt\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_acl\_initial\_alt\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message containing the syntax elements is selected as specified in subclause E.1.
* Coded atlas access unit 0 is a BLA access unit for which the coded atlas has nal\_unit\_type is equal to NAL\_BLA\_W\_RADL, NAL\_BLA\_N\_LP, NAL\_GBLA\_W\_RADL, or NAL\_GBLA\_N\_LP and the value of bp\_irap\_cab\_params\_present\_flag of the buffering period SEI message is equal to 1.
* Coded atlas access unit 0 is a BLA access unit for which the coded atlas access unit has nal\_unit\_type equal to NAL\_BLA\_W\_LP or NAL\_GBLA\_W\_LP, or is a CRA coded atlas access unit and the value of bp\_irap\_cab\_params\_present\_flag of the buffering period SEI message is equal to 1, and one or more of the following conditions are true:
* UseAltCabParamsFlag for coded atlas access unit 0 is equal to 1.
* DefaultInitCabParamsFlag is equal to 0.
* Otherwise, InitCabRemovalDelay[ Htid ][ SchedSelIdx ] and InitCabRemovalDelayOffset[ Htid ][ SchedSelIdx ] are set equal to the values of the buffering period SEI message syntax elements bp\_nal\_initial\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_nal\_initial\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, when NalHrdModeFlag is equal to 1, or bp\_acl\_initial\_cab\_removal\_delay[ Htid ][ SchedSelIdx ] and bp\_acl\_initial\_cab\_removal\_offset[ Htid ][ SchedSelIdx ], respectively, when NalHrdModeFlag is equal to 0, where the buffering period SEI message containing the syntax elements is selected as specified in subclause E.1, and CabDelayOffset and DabDelayOffset are both set equal to 0.

The nominal removal time of the coded atlas access unit n from the CAB is specified as follows:

* If coded atlas access unit n is the access unit with n equal to 0 (the access unit that initializes the HRD), the nominal removal time of the access unit from the CAB is specified by:

AuNominalRemovalTime[ 0 ] = InitCabRemovalDelay[ Htid ][ SchedSelIdx ] ÷ 90000 (E‑8)

* Otherwise, the following applies:

When coded atlas access unit n is the first coded atlas access unit of a buffering period that does not initialize the HRD, the following applies:

The nominal removal time of the coded atlas access unit n from the CAB is specified by:

if( !concatenationFlag ) {  
 baseTime = AuNominalRemovalTime[ firstAtlasInPrevBuffPeriod ]  
 removalDelay = AuCabRemovalDelayVal  
 } else {  
 offsetTime = AuFinalArrivalTime[ n – 1 ] − AuNominalRemovalTime[ n – 1 ]  
 baseTime = AuNominalRemovalTime[ PrevNonDiscardableAtlasFrame ]  
 removalDelay = Max( auCabRemovalDelayDelta, (E‑9)  
 Ceil( ( InitCabRemovalDelay[ Htid ][ SchedSelIdx ] ÷ 90000 + offsetTime )   
 ÷ ClockTick ) )  
 }  
 AuNominalRemovalTime( n ) = ClockTick \* ( removalDelay − CabDelayOffset ) + baseTime

where AuNominalRemovalTime[ firstAtlasInPrevBuffPeriod ] is the nominal removal time of the first coded atlas access unit of the previous buffering period, AuNominalRemovalTime[ PrevNonDiscardableAtlasFrame ] is the nominal removal time of the preceding coded atlas access unit in decoding order with TemporalID equal to 0 that is not a RASL, RADL, or SLNR atlas frame, AuCabRemovalDelayVal is the value of AuCabRemovalDelayVal derived according to aft\_au\_cab\_removal\_delay\_minus1[ Htid ] in the atlas frame timing SEI message, selected as specified in subclause E.1, associated with coded atlas access unit n, and concatenationFlag and auCabRemovalDelayDelta are equal to bp\_concatenation\_flag and ( bp\_atlas\_cab\_removal\_delay\_delta\_minus1 + 1 ), respectively, selected as specified in subclause E.1, and associated with coded atlas access unit n.

After the derivation of the nominal CAB removal time and before the derivation of the DAB output time of coded atlas access unit n, the values of CabDelayOffset and DabDelayOffset are updated as follows:

* + - * If one or more of the following conditions are true, CabDelayOffset is set equal to the value of the buffering period SEI message syntax element bp\_cab\_delay\_offset, and DabDelayOffset is set equal to the value of the buffering period SEI message syntax element bp\_dab\_delay\_offset, where the buffering period SEI message containing the syntax elements is selected as specified in subclause E.1:
      * Coded atlas access unit n is a BLA access unit for which the nal\_unit\_type is equal to NAL\_BLA\_W\_RADL, NAL\_GBLA\_W\_RADL, NAL\_BLA\_N\_LP, or NAL\_GBLA\_N\_LP and the value of bp\_irap\_cab\_params\_present\_flag of the buffering period SEI message is equal to 1.
      * Coded atlas access unit n is a BLA access unit for which the nal\_unit\_type is equal to NAL\_BLA\_W\_LP or NAL\_GBLA\_W\_LP, or is a CRA access unit, and the value of bp\_irap\_cab\_params\_present\_flag of the buffering period SEI message is equal to 1, and UseAltCabParamsFlag for access unit n is equal to 1.
* Otherwise, CabDelayOffset and DabDelayOffset are both set equal to 0.
  + When coded atlas access unit n is not the first coded atlas access unit of a buffering period, the nominal removal time of the coded atlas access unit n from the CAB is specified by:

AuNominalRemovalTime[ n ] =  
 AuNominalRemovalTime[ firstAtlasInCurrBuffPeriod ] +  
 ClockTick \* ( AuCabRemovalDelayVal – CabDelayOffset ) (E‑10)

where AuNominalRemovalTime[ firstAtlasInCurrBuffPeriod ] is the nominal removal time of the first access unit of the current buffering period, and AuCabRemovalDelayVal is the value of AuCabRemovalDelayVal derived according to aft\_au\_cab\_removal\_delay\_minus1[ Htid ] in the atlas timing SEI message, selected as specified in subclause E.1, and associated with access unit n.

* 1. Operation of the decoded atlas frame buffer
     1. General

The specifications in this subclause apply independently to each set of decoded atlas frame buffer (DAB) parameters selected as specified in subclause E.1.

The DAB contains atlas frame storage buffers. Each of the atlas frame storage buffers may contain a decoded atlas frame that is marked as "used for reference" or is held for future output. The processes specified in subclauses E.3.2 and E.3.3 are sequentially applied.

* + 1. Removal of atlas frames from the DAB before decoding of the current atlas frame

The removal of atlas frames from the DAB before decoding of the current atlas frame (but after parsing the tile header of the first tile of the current atlas frame) happens instantaneously at the CAB removal time of the first decoding unit of coded atlas access unit n (containing the current atlas frame) and proceeds as follows:

* The reference atlas list (RAFL) construction process, as specified in subclause 9.2.4.3, followed by the reference atlas frame marking process, as specified in subclause 9.2.4.4, are invoked.
* When the current atlas frame is an IRAP coded atlas frame with NoOutputBeforeRecoveryFlag equal to 1 that is not atlas frame 0, the following ordered steps are applied:
  + - The variable NoOutputOfPriorAtlasFramesFlag is derived for the decoder under test as follows:
* If the current atlas frame is a CRA atlas frame, NoOutputOfPriorAtlasFramesFlag is set equal to 1 (regardless of the value of ath\_no\_output\_of\_prior\_atlas\_frames\_flag).
* Otherwise, if the value of asps\_frame\_width, asps\_frame\_height, or asps\_max\_dec\_atlas\_frame\_buffering\_minus1 derived for the current atlas frame is different from the value of asps\_frame\_width, asps\_frame\_height, or asps\_max\_dec\_atlas\_frame\_buffering\_minus1, respectively, derived for the preceding atlas frame in decoding order, NoOutputOfPriorAtlasFramesFlag may (but should not) be set equal to 1 by the decoder under test, regardless of the value of ath\_no\_output\_of\_prior\_atlas\_frames\_flag of the current coded atlas access.

NOTE – Although setting NoOutputOfPriorAtlasFramesFlag equal to ath\_no\_output\_of\_prior\_atlas\_frames\_flag is preferred under these conditions, the decoder under test is allowed to set NoOutputOfPriorAtlasFramesFlag to 1 in this case.

* Otherwise, NoOutputOfPriorAtlasFramesFlag is set equal to ath\_no\_output\_of\_prior\_atlas\_frames\_flag.

2. The value of NoOutputOfPriorAtlasFramesFlag derived for the decoder under test is applied for the HRD, such that when the value of NoOutputOfPriorAtlasFramesFlag is equal to 1, all atlas frame storage buffers in the DAB are emptied without output of the atlas frames they contain, and the DAB fullness is set equal to 0.

– When both of the following conditions are true for any atlas frame k in the DAB, all such atlas frames k in the DAB are removed from the DAB:

– atlas frame k is marked as "unused for reference".

– atlas frame k has AtlasFrameOutputFlag equal to 0 or its DAB output time is less than or equal to the CAB removal time of the first decoding unit (denoted as decoding unit m) of the current atlas frame n; i.e., DabOutputTime[ k ] is less than or equal to AuCabRemovalTime[ m ].

– For each atlas frame that is removed from the DAB, the DAB fullness is decremented by one.

* + 1. Atlas frame output

The processes specified in this subclause happen instantaneously at the CAB removal time of the coded atlas access unit n, AuCabRemovalTime[ n ].

When atlas frame n has AtlasFrameOutputFlag equal to 1, its DAB output time DabOutputTime[ n ]is derived as follows, where the variable firstAtlasFrameInBufferingPeriodFlag is equal to 1 if coded atlas access unit n is the first access unit of a buffering period and 0 otherwise:

DabOutputTime[ n ] = AuCabRemovalTime[ n ] + ClockTick \* atlasFrameDabOutputDelay  
 if( firstAtlasFrameInBufferingPeriodFlag ) (E‑11)  
 DabOutputTime[ n ] −= ClockTick \* DabDelayOffset

where atlasFrameDabOutputDelay is the value of aft\_dab\_output\_delay[ Htid ] in the atlas frame timing SEI message associated with coded atlas access unit n.

The output of the current atlas is specified as follows:

* If the ath\_atlas\_output\_flag is equal to 1 for at least one atlas tile with tile ID equal to tileID, then the variable AtlasFrameOutputFlag is set to be equal to 1. Otherwise the variable AtlasFrameOutputFlag is set to be equal to 0.
* If AtlasFrameOutputFlag is equal to 1 and DabOutputTime[ n ]is equal to AuCabRemovalTime[ n ], the current atlas frame is output.
* Otherwise, if AtlasFrameOutputFlag is equal to 0, the current atlas frame is not output, but will be stored in the DAB as specified in subclause E.3.4
* Otherwise (AtlasFrameOutputFlag is equal to 1 and DabOutputTime[ n ]is greater than AuCabRemovalTime[ n ]), the current atlas frame is output later and will be stored in the DAB (as specified in subclause E.3.4) and is output at time DabOutputTime[ n ]unless indicated not to be output at a time that precedes DabOutputTime[ n ].

When atlas frame n is an atlas frame that is output and is not the last atlas frame of the bitstream that is output, the value of the variable DabOutputInterval[ n ]is derived as follows:

DabOutputInterval[ n ] =  
 DabOutputTime[ nextAtlasFrameInOutputOrder ] − DabOutputTime[ n ] (E‑12)

where nextAtlasFrameInOutputOrder is the atlas frame that follows atlas frame n in output order and has AtlasFrameOutputFlag equal to 1.

* + 1. Current decoded atlas frame marking and storage

The current decoded atlas frame is stored in the DAB in an empty atlas frame storage buffer, and the DAB fullness is incremented by one. When asps\_long\_term\_ref\_atlas\_frames\_flag is equal to 1, this atlas frame is marked as "used for long-term reference". After all the tiles of the current atlas frame have been decoded, this atlas frame is marked as "used for short-term reference".

NOTE – Unless more memory than required by the level limit is available for storage of decoded atlas frames, decoders should start storing decoded parts of the current atlas frames into the DAB when the first tile is decoded and continue storing more decoded samples as the decoding process proceeds.

* + 1. Removal of atlas frames from the DAB after decoding of the current atlas frame

Immediately after decoding of the current atlas frame, at the CAB removal time of the coded atlas access unit n (containing the current atlas frame), the current decoded atlas frame is removed from the DAB, and the DAB fullness is decremented by one.

* 1. Bitstream conformance

A bitstream of coded data conforming to this document shall fulfil all requirements specified in this subclause.

The bitstream shall be constructed according to the syntax, semantics, and constraints specified in this document outside this annex.

The first coded atlas access unit in a bitstream shall be an IRAP coded atlas frame, i.e., an IDR atlas frame, a CRA atlas frame, or a BLA atlas frame.

The atlas sub-bitstream is tested by the HRD for conformance as specified in subclause E.1.

For each current atlas frame, let the variables maxAtlasFrameOrderCnt and minAtlasFrameOrderCnt be set equal to the maximum and the minimum, respectively, of the AtlasFrmOrderCntVal values of the following atlas frames:

* The current atlas frame.
* The previous atlas frame in decoding order that has TemporalId equal to 0 and that is not a RASL, RADL, or SLNR atlas frame
* The short-term reference atlas frames in the reference atlas list (RAFL) of the current atlas frame.
* All atlas frames n that have AtlasFrameOutputFlag equal to 1, AuCabRemovalTime[ n ] less than AuCabRemovalTime[ currAtlasFrame ], and DabOutputTime[ n ] greater than or equal to AuCabRemovalTime[ currAtlasFrame ], where currAtlasFrame is the current atlas frame.

All of the following conditions shall be fulfilled for each of the bitstream conformance tests:

1. For each coded atlas access unit n, with n greater than 0, associated with a buffering period SEI message, let the variable deltaTime90k[ n ]be specified as follows:

deltaTime90k[ n ] = 90 000 \* ( AuNominalRemovalTime[ n ] − AuFinalArrivalTime[ n − 1 ] ) (E‑13)

The value of InitCabRemovalDelay[ Htid ][ SchedSelIdx ] is constrained as follows:

* If hrd\_cbr\_flag[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] is equal to 0, the following condition shall be true:

InitCabRemovalDelay[ Htid ][ SchedSelIdx ] <= Ceil( deltaTime90k[ n ] ) (E‑14)

* Otherwise ( hrd\_cbr\_flag[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] is equal to 1 ), the following condition shall be true:

Floor( deltaTime90k[ n ]) <=  
 InitCabRemovalDelay[ Htid ][ SchedSelIdx ] <= Ceil( deltaTime90k[ n ] ) (E‑15)

NOTE 1 – The exact number of bits in the CAB at the removal time of each atlas frame can depend on which buffering period SEI message is selected to initialize the HRD. Encoders shall take this into account to ensure that all specified constraints shall be obeyed regardless of which buffering period SEI message is selected to initialize the HRD, as the HRD can be initialized at any one of the buffering period SEI messages.

1. A CAB overflow is specified as the condition in which the total number of bits in the CAB is greater than the CAB size. The CAB shall never overflow.
2. When hrd\_low\_delay\_flag[ Htid ] is equal to 0, the CAB shall never underflow. A CAB underflow is specified as follows:

– A CAB underflow is specified as the condition in which the nominal CAB removal time of the coded atlas access unit n, AuNominalRemovalTime[ n ], is less than the final CAB arrival time of the coded atlas access unit n, AuFinalArrivalTime[ n ], for at least one value of n.

1. The nominal removal times of atlas frames from the CAB (starting from the second atlas in decoding order) shall satisfy the constraints on AuNominalRemovalTime[ n ] and AuCabRemovalTime[ n ] expressed in Annex A.
2. For each current atlas frame, after invocation of the process for removal of atlases from the DAB as specified in subclause E.3.2, the number of decoded atlas frames in the DAB, including all atlas frames n that are marked as "used for reference", or that have AtlasFrameOutputFlag equal to 1 and AuCabRemovalTime[ n ]less than AuCabRemovalTime[ currAtlasFrame ], where currAtlasFrame is the current atlas frame, shall be less than or equal to asps\_max\_dec\_atlas\_frame\_buffering\_minus1.
3. All reference atlas frames shall be present in the DAB when needed for prediction. Each atlas frame that has AtlasFrameOutputFlag equal to 1 shall be present in the DAB at its DAB output time unless it is removed from the DAB before its output time by one of the processes specified in subclause E.3
4. For each current atlas frame that is not an IRAP coded atlas frame with NoOutputBeforeRecoveryFlag equal to 1, the value of maxAtlasFrameOrderCnt − minAtlasFrameOrderCnt shall be less than MaxAtlasFrmOrderCntLsb / 2.
5. The value of DabOutputInterval[ n ] as given by Equation E‑12, which is the difference between the output time of an atlas frame and that of the first atlas frame following it in output order and having AtlasFrameOutputFlag equal to 1, shall satisfy the constraint expressed in Annex A for the profile, tier, and level specified in the bitstream using the decoding process specified in subclause 9.2.
6. For any two atlas frames m and n in the same CAS, when DabOutputTime[ m ] is greater than DabOutputTime[ n ], the AtlasFrmOrderCntVal of atlas frame m shall be greater than the AtlasFrmOrderCntVal of atlas frame n.

NOTE 2 – All atlas frames of an earlier CAS in decoding order that are output are output before any atlas frames of a later CAS in decoding order. Within any particular CAS, the atlas frames that are output are output in increasing AtlasFrmOrderCntVal order.

* 1. Decoder conformance
     1. General

A decoder conforming to this document shall fulfil all requirements specified in this subclause.

A decoder claiming conformance to a specific profile, tier, and level shall be able to successfully decode all atlas frame bitstreams that conform to the atlas frame bitstream conformance requirements specified in subclause E.4, in the manner specified in Annex A, provided that all AAPSs, ASPSs, and AFPSs referred to in the ACL NAL units and appropriate buffering period and atlas frame timing SEI messages are conveyed to the decoder, in a timely manner, either in the bitstream (by non-ACL NAL units), or by external means not specified in this document.

When an atlas sub-bitstream contains syntax elements that have values that are specified as reserved and it is specified that decoders shall ignore values of the syntax elements or NAL units containing the syntax elements having the reserved values, and the atlas sub-bitstream is otherwise conforming to this document, a conforming decoder shall decode the atlas sub-bitstream in the same manner as it would decode a conforming atlas sub-bitstream and shall ignore the syntax elements or the NAL units containing the syntax elements having the reserved values as specified.

There are two types of conformance that can be claimed by an atlas frame decoder: output timing conformance and output order conformance.

To check conformance of an atlas frame decoder, test bitstreams conforming to the claimed profile, tier and level, as specified in subclause E.4 are delivered by a hypothetical atlas frame stream scheduler (HSS) both to the HRD and to the atlas frame decoder under test (DUT). All decoded atlas frames output by the HRD shall also be output by the DUT, each decoded atlas frame output by the DUT shall be an atlas frame with AtlasFrameOutputFlag equal to 1, and, for each such decoded atlas frame output by the DUT, the information associated with all atlas tiles and patches that are output shall be equivalent to the information associated with the atlas tiles and patches produced by the specified atlas frame decoding process.

For output timing atlas frame decoder conformance, the HSS operates as described above, with delivery schedules selected only from the subset of values of SchedSelIdx for which the bit rate and CAB size are restricted as specified in Annex A for the specified profile, tier, and level or with "interpolated" delivery schedules as specified below for which the bit rate and CAB size are restricted as specified in Annex A. The same delivery schedule is used for both the HRD and the DUT.

When the HRD parameters and the buffering period SEI messages are present with hrd\_cab\_cnt\_minus1[ Htid ] greater than 0, the atlas frame decoder shall be capable of decoding the atlas sub-bitstream as delivered from the HSS operating using an "interpolated" delivery schedule specified as having peak bit rate r, CAB size c( r ), and initial CAB removal delay ( f( r ) ÷ r ) as follows:

α = ( r – BitRate[[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx – 1 ] ) ÷  
 ( BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] –   
 BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx – 1 ] ) (E‑16)

c( r ) = α \* CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] +  
  ( 1 – α ) \* CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx – 1 ] (E‑17)

f( r ) = α \* InitCabRemovalDelay[ Htid ][ SchedSelIdx ] \*   
 BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ] +  
 ( 1 – α ) \* InitCabRemovalDelay[ Htid ][ SchedSelIdx – 1 ] \*  
 BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx – 1 ] (E‑18)

for any SchedSelIdx > 0 such that:

BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx – 1 ] <= r   
 <= BitRate[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ]

such that r and c( r ) are within the limits as specified in Annex A for the maximum bit rate and buffer size for the specified profile, tier, and level.

NOTE 1 – InitCabRemovalDelay[ Htid ][ SchedSelIdx ] can be different from one buffering period to another and has to be re-calculated.

For output timing atlas frame decoder conformance, an HRD as described above is used and the timing (relative to the delivery time of the first bit) of atlas frame output is the same for both the HRD and the DUT up to a fixed delay.

For output order atlas frame decoder conformance, the following applies:

* The HSS delivers the bitstream AtlasBitstreamToDecode to the DUT "by demand" from the DUT, meaning that the HSS delivers bits (in decoding order) only when the DUT requires more bits to proceed with its processing.

NOTE 2 – This means that for this test, the coded atlas buffer of the DUT could be as small as the size of the largest decoding unit.

* A modified HRD, as described in subclause E.5.2, is used, and the HSS delivers the bitstream to the HRD by one of the schedules specified in the atlas frame bitstream, AtlasBitstreamToDecode, such that the bit rate and CAB size are restricted as specified in Annex A. The output order of atlas frames shall be the same for both the HRD and the DUT.
* The HRD CAB size is given by CabSize[ !NalHrdModeFlag ][ Htid ][ SchedSelIdx ], as specified in subclause G.3.3, where SchedSelIdx and the HRD parameters are selected as specified in subclause E.1. The DAB size is given by asps\_max\_dec\_atlas\_frame\_buffering\_minus1 + 1. Removal time from the CAB for the HRD is the final bit arrival time and decoding is immediate. The operation of the DAB of this HRD is as described in subclauses E.5.2 through E.5.2.3.
  + 1. Operation of the output order DAB
       1. General

The decoded atlas frame buffer contains atlas frame storage buffers. Each of the atlas frame storage buffers contains a decoded atlas frame that is marked as "used for reference" or is held for future output. The process for output and removal of atlas frames from the DAB before decoding of the current atlas frame as specified in subclause E.5.2.2 is invoked, the invocation of the process for current decoded atlas frame marking and storage as specified in subclause E.3.4, further followed by the invocation of the process for removal of the atlas frame from the DAB after decoding of the current atlas frame as specified in subclause E.3.5, and finally followed by the invocation of the process for additional bumping as specified in subclause E.5.2.3. The "bumping" process is specified in subclause E.5.2.4 and is invoked as specified in subclauses E.5.2.2 and E.5.2.3.

* + - 1. Output and removal of atlas frames from the DAB

The output and removal of atlas frames from the DAB before the decoding of the current atlas frame (but after parsing the tile header of the first tile of the current atlas frame) happens instantaneously when the first decoding unit of the access unit containing the current atlas frame is removed from the CAB and proceeds as follows:

* The decoding process for RAFL as specified in subclause 9.2.4.3, is invoked.
* If the current picture is an IRAP coded atlas frame with NoOutputOfPriorAtlasFramesFlag equal to 1 that is not atlas frame 0, the following ordered steps are applied:

1. The variable NoOutputOfPriorAtlasFrameFlag is derived for the decoder under test as follows:

* If the current atlas frame is CRA atlas frame, NoOutputOfPriorAtlasFramesFlag is set equal to 1 (regardless of the value of ath\_no\_output\_of\_prior\_atlas\_frames\_flag).
* Otherwise, if the value of asps\_frame\_width, asps\_frame\_height, or asps\_max\_dec\_atlas\_frame\_buffering\_minus1 derived for the current atlas frame is different from the value of asps\_frame\_width, asps\_frame\_height, or asps\_max\_dec\_atlas\_frame\_buffering\_minus1, respectively, derived from the ASPS active for the preceding atlas, NoOutputOfPriorAtlasFramesFlag may (but should not) be set equal to 1 by the decoder under test, regardless of the value of ath\_no\_output\_of\_prior\_atlas\_frames\_flag of the current atlas frame.
* Otherwise, NoOutputOfPriorAtlasFramesFlag is set equal to ath\_no\_output\_of\_prior\_atlas\_frames\_flag if the current coded atlas access unit.

NOTE – Although, under these conditions, setting NoOutputOfPriorAtlasFramesFlag equal to ath\_no\_output\_of\_prior\_atlas\_frames\_flag of the current coded atlas frame is preferred, the decoder under test is allowed to set, in this case, NoOutputOfPriorAtlasFramesFlag equal to 1.

1. The value of NoOutputOfPriorAtlasFramesFlag derived for the decoder under test is applied for the HRD as follows:

* If NoOutputOfPriorAtlasFramesFlag is equal to 1, all atlas frame storage buffers in the DAB are emptied without output of the atlas frames they contain, and the DAB fullness is set equal to 0.
* Otherwise (NoOutputOfPriorAtlasFramesFlag is equal to 0), all atlas frame storage buffers containing atlas frames that are marked as "not needed for output" and "unused for reference" are emptied (without output) and all non-empty atlas frame storage buffers in the DAB are emptied by repeatedly invoking the "bumping" process specified in subclause E.5.2.4 and the DAB fullness is set equal to 0.
* Otherwise (the current atlas frame is not an IRAP coded atlas frame with NoOutputBeforeRecoveryFlag equal to 1), all atlas frame storage buffers containing atlas frames which are marked as "not needed for output" and "unused for reference" are emptied (without output). For each atlas frame storage buffer that is emptied, the DAB fullness is decremented by one. The "bumping" process specified in subclause E.5.2.4 is invoked repeatedly for each additional atlas frame storage buffer that is emptied, until the number of atlas frames in the DAB is less than asps\_max\_dec\_atlas\_frame\_buffering\_minus1 + 1.
  + - 1. Additional bumping

The processes specified in this subclause happen instantaneously when the last decoding unit of access unit n containing the current atlas frame is removed from the CAB.

When the current atlas frame has AtlasFrameOutputFlag equal to 1, for each atlas frame in the DAB that is marked as "needed for output" and follows the current atlas frame in output order, the associated variable AtlasFrameLatencyCount is set equal to AtlasFrameLatencyCount + 1.

The following applies:

* If the current decoded atlas frame has AtlasFrameOutputFlag equal to 1, it is marked as "needed for output" and its associated variable AtlasFrameLatencyCount is set equal to 0.
* Otherwise (the current decoded atlas frame has AtlasFrameOutputFlag equal to 0), it is marked as "not needed for output".

When the following condition is true, the "bumping" process specified in subclause E.5.2.4 is invoked repeatedly until the following conditions is not true:

* The number of atlas frames in the DAB that are marked as "needed for output" is greater than asps\_max\_dec\_atlas\_frame\_buffering\_minus1 + 1.
  + - 1. "Bumping" process

The "bumping" process consists of the following ordered steps:

1. The atlas frame that is first for output is selected as the one having the smallest value of AtlasFrmOrderCntVal of all atlas frames in the DAB is marked as "needed for output".
2. The atlas frame is output, and the atlas frame is marked as "not needed for output".
3. When the atlas frame storage buffer that included the atlas frame that was output contains an atlas frame marked as "unused for reference", the atlas frame storage buffer is emptied.

NOTE – For any two atlas frames atlasFrameA and atlasFrameB that belong to the same CAS and are output by the "bumping process", when atlasFrameA is output earlier than atlasFrameB, the value of AtlasFrmOrderCntVal of atlasFrameA is less than the value of AtlasFrmOrderCntVal of atlasFrameB.

1. (normative)  
   Supplemental enhancement information
   1. General

This annex specifies syntax and semantics for SEI message payloads associated with an atlas, identified with an atlas ID, SeiAtlasID, which is set equal to vuh\_atlas\_id or determined through external means if the V3C unit header is unavailable.

SEI messages assist in processes related to decoding, reconstruction, display, or other purposes. This annex defines two types of SEI messages: essential and non-essential.

Non-essential SEI messages are not required by the decoding process. Conforming decoders are not required to process this information for output order conformance to this document (see Annex A for the specification of conformance).

In subclause E.5.2 including its subclauses, specification for presence of non-essential SEI messages is also satisfied when those messages (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified in this document. When present in the bitstream, non-essential SEI messages shall obey the syntax and semantics specified in subclause 8.3.8 and this annex. When the content of a non-essential SEI message is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the SEI message is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

Essential SEI messages are an integral part of the V3C bitstream and should not be removed from the bitstream. The essential SEI messages are categorized into two types, based on the conformance point they are related to, as specified in Annex A.

1. Conformance point A, named as Type-A, essential SEI messages: These SEIs contain information required to check bitstream conformance and for output timing decoder conformance. Every V3C decoder conforming to point A should not discard any relevant Type-A essential SEI messages and shall consider them for bitstream conformance and for output timing decoder conformance.
2. Conformance point B, named as Type-B, essential SEI messages: V3C decoders that wish to conform to a particular reconstruction profile as specified in Annex A should not discard any relevant Type-B essential SEI messages and should consider them for volumetric frame reconstruction and conformance purposes.

Table F-1 lists the essential and non-essential SEI messages. In case of essential SEI messages, the type is also specified.

Table F-1 – The essential and non-essential SEI messages

|  |  |  |
| --- | --- | --- |
| **SEI message** | **NAL Type** | **Conformance Type** |
| Filler payload SEI | NAL\_PREFIX\_NSEI/ NAL\_SUFFIX\_NSEI | N/A |
| User data registered by Recommendation ITU-T T.35 | NAL\_PREFIX\_NSEI/ NAL\_SUFFIX\_NSEI | N/A |
| User data unregistered | NAL\_PREFIX\_NSEI/ NAL\_SUFFIX\_NSEI | N/A |
| Recovery point | NAL\_PREFIX\_NSEI | N/A |
| No reconstruction | NAL\_PREFIX\_NSEI | N/A |
| Reserved | N/A | N/A |
| SEI manifest | NAL\_PREFIX\_ESEI | Type-A, Type-B |
| SEI prefix indication | NAL\_PREFIX\_ESEI | Type-A, Type-B |
| Active sub-bitstreams | NAL\_PREFIX\_ESEI | Type-A |
| Component codec mapping | NAL\_PREFIX\_ESEI | Type-A |
| Scene object information | NAL\_PREFIX\_NSEI | N/A |
| Object label information | NAL\_PREFIX\_NSEI | N/A |
| Patch information | NAL\_PREFIX\_NSEI | N/A |
| Volumetric rectangle information | NAL\_PREFIX\_NSEI | N/A |
| Atlas object association | NAL\_PREFIX\_NSEI | N/A |
| Buffering period | NAL\_PREFIX\_ESEI | Type-A |
| Atlas frame timing | NAL\_PREFIX\_ESEI | Type-A |
| Viewport camera parameters | NAL\_PREFIX\_NSEI | N/A |
| Viewport position | NAL\_PREFIX\_NSEI | N/A |
| Decoded atlas information hash | NAL\_SUFFIX\_NSEI | N/A |
| Time code | NAL\_PREFIX\_NSEI | N/A |
| Attribute transformation parameters | NAL\_PREFIX\_NSEI | Type-B |
| Occupancy synthesis | NAL\_PREFIX\_ESEI | Type-B |
| Geometry smoothing | NAL\_PREFIX\_ESEI | Type-B |
| Attribute smoothing | NAL\_PREFIX\_ESEI | Type-B |

* 1. SEI payload syntax
     1. General SEI message syntax

|  |  |
| --- | --- |
| sei\_payload( payloadType, payloadSize ) { | **Descriptor** |
| if(( nal\_unit\_type == NAL\_PREFIX\_NSEI ) || ( nal\_unit\_type == NAL\_PREFIX\_ESEI )) { |  |
| if( payloadType == 0 ) |  |
| buffering\_period( payloadSize ) |  |
| else if( payloadType == 1 ) |  |
| atlas\_frame\_timing( payloadSize ) |  |
| else if( payloadType == 2 ) |  |
| filler\_payload( payloadSize ) |  |
| else if( payloadType == 3 ) |  |
| user\_data\_registered\_itu\_t\_t35( payloadSize ) |  |
| else if( payloadType == 4 ) |  |
| user\_data\_unregistered( payloadSize ) |  |
| else if( payloadType == 5 ) |  |
| recovery\_point( payloadSize ) |  |
| else if( payloadType == 6 ) |  |
| no\_display( payloadSize ) |  |
| else if( payloadType == 7 ) |  |
| time\_code( payloadSize ) |  |
| else if( payloadType == 8 ) |  |
| sei\_manifest( payloadSize ) |  |
| else if( payloadType == 9 ) |  |
| sei\_prefix\_indication( payloadSize ) |  |
| else if( payloadType == 10 ) |  |
| active\_sub\_bitstreams( payloadSize ) |  |
| else if( payloadType == 11 ) |  |
| component\_codec\_mapping( payloadSize ) |  |
| else if( payloadType == 12 ) |  |
| scene\_object\_information( payloadSize ) |  |
| else if( payloadType == 13 ) |  |
| object\_label\_information( payloadSize ) |  |
| else if( payloadType == 14 ) |  |
| patch\_information( payloadSize ) |  |
| else if( payloadType == 15 ) |  |
| volumetric\_rectangle\_information( payloadSize ) |  |
| else if( payloadType == 16 ) |  |
| atlas\_object\_association( payloadSize ) |  |
| else if( payloadType == 17 ) |  |
| viewport\_camera\_parameters( payloadSize ) |  |
| else if( payloadType == 18 ) |  |
| viewport\_position( payloadSize ) |  |
| else if( payloadType == 19 ) |  |
| packed\_independent\_regions( payloadSize ) |  |
| else if( payloadType == 64 ) |  |
| attribute\_transformation\_params( payloadSize ) /\* Specified in Annex H \*/ |  |
| else if( payloadType == 65 ) |  |
| occupancy\_synthesis( payloadSize ) /\* Specified in Annex H \*/ |  |
| else if( payloadType == 66 ) |  |
| geometry\_smoothing( payloadSize ) /\* Specified in Annex H \*/ |  |
| else if( payloadType == 67 ) |  |
| attribute\_smoothing( payloadSize ) /\* Specified in Annex H \*/ |  |
| else if( payloadType == 128 ) |  |
| viewing\_space( payloadSize ) /\* Specified in ISO/IEC 23090-12 \*/ |  |
| else if( payloadType == 129 ) |  |
| viewing\_space\_handling( payloadSize ) /\* Specified in ISO/IEC 2309-12 \*/ |  |
| else if( payloadType == 130 ) |  |
| geometry\_upscaling\_parameters( payloadSize ) /\* Specified in ISO/IEC 23090-12 \*/ |  |
| else |  |
| reserved\_sei\_message( payloadSize ) |  |
| } |  |
| else { /\*( nal\_unit\_type == NAL\_SUFFIX\_NSEI ) || ( nal\_unit\_type == NAL\_SUFFIX\_ESEI )\*/ |  |
| if( payloadType == 2 ) |  |
| filler\_payload( payloadSize ) |  |
| else if( payloadType == 3 ) |  |
| user\_data\_registered\_itu\_t\_t35( payloadSize ) |  |
| else if( payloadType == 4 ) |  |
| user\_data\_unregistered( payloadSize ) |  |
| else if( payloadType == 19 ) |  |
| decoded\_atlas\_information\_hash( payloadSize ) |  |
| else |  |
| reserved\_sei\_message( payloadSize ) |  |
| } |  |
| if( more\_data\_in\_payload( ) ) { |  |
| if( payload\_extension\_present( ) ) |  |
| **sp\_reserved\_payload\_extension\_data** | u(v) |
| byte\_alignment( ) |  |
| } |  |
| } |  |

* + 1. Filler payload SEI message syntax

|  |  |
| --- | --- |
| filler\_payload( payloadSize ) { | Descriptor |
| for( k = 0; k < payloadSize; k++ ) |  |
| **ff\_byte** /\* equal to 0xFF \*/ | f(8) |
| } |  |

* + 1. User data registered by Recommendation ITU-T T.35 SEI message syntax

|  |  |
| --- | --- |
| user\_data\_registered\_itu\_t\_t35( payloadSize ) { | Descriptor |
| **itu\_t\_t35\_country\_code** | b(8) |
| if( itu\_t\_t35\_country\_code != 0xFF ) |  |
| i = 1 |  |
| else { |  |
| **itu\_t\_t35\_country\_code\_extension\_byte** | b(8) |
| i = 2 |  |
| } |  |
| do { |  |
| **itu\_t\_t35\_payload\_byte** | b(8) |
| i++ |  |
| } while( i < payloadSize ) |  |
| } |  |

* + 1. User data unregistered SEI message syntax

|  |  |
| --- | --- |
| user\_data\_unregistered( payloadSize ) { | Descriptor |
| **uuid\_iso\_iec\_11578** | u(128) |
| for( i = 16; i < payloadSize; i++ ) |  |
| **user\_data\_payload\_byte** | b(8) |
| } |  |

* + 1. Recovery point SEI message syntax

|  |  |
| --- | --- |
| recovery\_point( payloadSize ) { | Descriptor |
| **recovery\_afoc\_cnt** | se(v) |
| **exact\_match\_flag** | u(1) |
| **broken\_link\_flag** | u(1) |
| } |  |

* + 1. No reconstruction SEI message syntax

|  |  |
| --- | --- |
| no\_reconstruction( payloadSize ) { | Descriptor |
| } |  |

* + 1. Reserved SEI message syntax

|  |  |
| --- | --- |
| reserved\_sei\_message( payloadSize ) { | Descriptor |
| for( i = 0; i < payloadSize; i++ ) |  |
| **reserved\_sei\_message\_payload\_byte** | b(8) |
| } |  |

* + 1. SEI manifest SEI message syntax

|  |  |
| --- | --- |
| sei\_manifest( payloadSize ) { | **Descriptor** |
| **manifest\_num\_sei\_msg\_types** | u(16) |
| for( i = 0; i < manifest\_num\_sei\_msg\_types; i++ ) { |  |
| **manifest\_sei\_payload\_type**[ i ] | u(16) |
| **manifest\_sei\_description**[ i ] | u(8) |
| } |  |
| } |  |

* + 1. SEI prefix indication SEI message syntax

|  |  |
| --- | --- |
| sei\_prefix\_indication( payloadSize ) { | **Descriptor** |
| **prefix\_sei\_payload\_type** | u(16) |
| **num\_sei\_prefix\_indications\_minus1** | u(8) |
| for( i = 0; i <= num\_sei\_prefix\_indications\_minus1; i++ ) { |  |
| **num\_bits\_in\_prefix\_indication\_minus1**[ i ] | u(16) |
| for( j = 0; j <= num\_bits\_in\_prefix\_indication\_minus1[ i ]; j++ ) |  |
| **sei\_prefix\_data\_bit**[ i ][ j ] | u(1) |
| while( !byte\_aligned( ) ) |  |
| **byte\_alignment\_bit\_equal\_to\_one** /\* equal to 1 \*/ | f(1) |
| } |  |
| } |  |

* + 1. Active sub-bitstreams SEI message syntax

|  |  |
| --- | --- |
| active\_sub\_bitstreams( payloadSize ) { | **Descriptor** |
| **asb\_active\_sub\_bitstreams\_cancel\_flag** | u(1) |
| if( !asb\_active\_sub\_bitstreams\_cancel\_flag ) { |  |
| **asb\_active\_attributes\_changes\_flag** | u(1) |
| **asb\_active\_maps\_changes\_flag** | u(1) |
| **asb\_auxiliary\_sub\_bitstreams\_active\_flag** | u(1) |
| if( asb\_active\_attributes\_changes\_flag ) { |  |
| **asb\_all\_attributes\_active\_flag** | u(1) |
| if( !asb\_all\_attributes\_active\_flag ) { |  |
| **asb\_active\_attribute\_count\_minus1** | u(7) |
| for( i = 0; i <= asb\_active\_attribute\_count\_minus1; i++ ) |  |
| **asb\_active\_attribute\_idx**[ i ] | u(7) |
| } |  |
| } |  |
| if( asb\_active\_maps\_changes\_flag ) { |  |
| **asb\_all\_maps\_active\_flag** | u(1) |
| if( !asb\_all\_maps\_active\_flag ) { |  |
| **asb\_active\_map\_count\_minus1** | u(4) |
| for( i = 0; i <= asb\_active\_map\_count\_minus1 ) |  |
| **asb\_active\_map\_idx**[ i ] | u(4) |
| } |  |
| } |  |
| } |  |
| } |  |

* + 1. Component codec mapping SEI message syntax

|  |  |
| --- | --- |
| component\_codec\_mapping( payloadSize ) { | **Descriptor** |
| **ccm\_component\_codec\_cancel\_flag** | u(1) |
| if( !ccm\_component\_codec\_cancel\_flag ) { |  |
| **ccm\_codec\_mappings\_count\_minus1** | u(8) |
| for( i = 0; i <= ccm\_codec\_mappings\_count\_minus1; i++ ) { |  |
| **ccm\_codec\_id** | u(8) |
| **ccm\_codec\_4cc**[ ccm\_codec\_id ] | st(v) |
| } |  |
| } |  |
| } |  |

* + 1. Volumetric annotation SEI message family syntax
       1. Scene object information SEI message syntax

|  |  |
| --- | --- |
| scene\_object\_information( payloadSize ) { |  |
| **soi\_persistence\_flag** | u(1) |
| **soi\_reset\_flag** | u(1) |
| **soi\_num\_object\_updates** | ue(v) |
| if( soi\_num\_object\_updates > 0 ) { |  |
| **soi\_simple\_objects\_flag** | u(1) |
| if( soi\_simple\_objects\_flag == 0) { |  |
| **soi\_object\_label\_present\_flag** | u(1) |
| **soi\_priority\_present\_flag** | u(1) |
| **soi\_object\_hidden\_present\_flag** | u(1) |
| **soi\_object\_dependency\_present\_flag** | u(1) |
| **soi\_visibility\_cones\_present\_flag** | u(1) |
| **soi\_3d\_bounding\_box\_present\_flag** | u(1) |
| **soi\_collision\_shape\_present\_flag** | u(1) |
| **soi\_point\_style\_present\_flag** | u(1) |
| **soi\_material\_id\_present\_flag** | u(1) |
| **soi\_extension\_present\_flag** | u(1) |
| } |  |
| else { |  |
| soi\_object\_label\_present\_flag = 0 |  |
| soi\_priority\_present\_flag = 0 |  |
| soi\_object\_hidden\_present\_flag = 0 |  |
| soi\_object\_dependency\_present\_flag = 0 |  |
| soi\_visibility\_cones\_present\_flag = 0 |  |
| soi\_3d\_bounding\_box\_present\_flag = 0 |  |
| soi\_collision\_shape\_present\_flag = 0 |  |
| soi\_point\_style\_present\_flag = 0 |  |
| soi\_material\_id\_present\_flag = 0 |  |
| soi\_extension\_present\_flag = 0 |  |
| } |  |
| if( soi\_3d\_bounding\_box\_present\_flag ) { |  |
| **soi\_3d\_bounding\_box\_scale\_log2** | u(5) |
| } |  |
| **soi\_log2\_max\_object\_idx\_updated\_minus1** | u(5) |
| if( soi\_object\_dependency\_present\_flag ) |  |
| **soi\_log2\_max\_object\_dependency\_idx** | u(5) |
| for( i = 0; i < soi\_num\_object\_updates; i++ ) { |  |
| **soi\_object\_idx**[ i ] | u(v) |
| k = soi\_object\_idx[ i ] |  |
| **soi\_object\_cancel\_flag**[ k ] | u(1) |
| ObjectTracked[ k ] = !soi\_object\_cancel\_flag[ k ] |  |
| if( !soi\_object\_cancel\_flag[ k ] ) { |  |
| if( soi\_object\_label\_present\_flag ) { |  |
| **soi\_object\_label\_update\_flag**[ k ] | u(1) |
| if( soi\_object\_label\_update\_flag[ k ] ) |  |
| **soi\_object\_label\_idx**[ k ] | ue(v) |
| } |  |
| if( soi\_priority\_present\_flag ) { |  |
| **soi\_priority\_update\_flag**[ k ] | u(1) |
| if( soi\_priority\_update\_flag[ k ] ) |  |
| **soi\_priority\_value**[ k ] | u(4) |
| } |  |
| if( soi\_object\_hidden\_present\_flag ) |  |
| **soi\_object\_hidden\_flag**[ k ] | u(1) |
| if( soi\_object\_dependency\_present\_flag ) { |  |
| **soi\_object\_dependency\_update\_flag**[ k ] | u(1) |
| if( soi\_object\_dependency\_update\_flag[ k ] ) { |  |
| **soi\_object\_num\_dependencies**[ k ] | u(4) |
| for( j = 0; j < soi\_object\_num\_dependencies[ k ]; j++ ) |  |
| **soi\_object\_dependency\_idx**[ k ][ j ] | u(v) |
| } |  |
| } |  |
| if( soi\_visibility\_cones\_present\_flag ) { |  |
| **soi\_visibility\_cones\_update\_flag**[ k ] | u(1) |
| if( soi\_visibility\_cones\_update\_flag[ k ]) { |  |
| **soi\_direction\_x**[ k ] | i(16) |
| **soi\_direction\_y**[ k ] | i(16) |
| **soi\_direction\_z**[ k ] | i(16) |
| **soi\_angle**[ k ] | u(16) |
| } |  |
| } |  |
| if( soi\_3d\_bounding\_box\_present\_flag ) { |  |
| **soi\_3d\_bounding\_box\_update\_flag**[ k ] | u(1) |
| if( soi\_3d\_bounding\_box\_update\_flag[ k ]) { |  |
| **soi\_3d\_bounding\_box\_x**[ k ] | ue(v) |
| **soi\_3d\_bounding\_box\_y**[ k ] | ue(v) |
| **soi\_3d\_bounding\_box\_z**[ k ] | ue(v) |
| **soi\_3d\_bounding\_box\_size\_x**[ k ] | ue(v) |
| **soi\_3d\_bounding\_box\_size\_y**[ k ] | ue(v) |
| **soi\_3d\_bounding\_box\_size\_z**[ k ] | ue(v) |
| } |  |
| } |  |
| if( soi\_collision\_shape\_present\_flag ) { |  |
| **soi\_collision\_shape\_update\_flag**[ k ] | u(1) |
| if(soi\_collision\_shape\_update\_flag[ k ]) |  |
| **soi\_collision\_shape\_id**[ k ] | u(16) |
| } |  |
| if( soi\_point\_style\_present\_flag ) { |  |
| **soi\_point\_style\_update\_flag**[ k ] | u(1) |
| if( soi\_point\_style\_update\_flag[ k ] ) { |  |
| **soi\_point\_shape\_id**[ k ] | u(8) |
| **soi\_point\_size**[ k ] | u(16) |
| } |  |
| } |  |
| if( soi\_material\_id\_present\_flag ) { |  |
| **soi\_material\_id\_update\_flag**[ k ] | u(1) |
| if( soi\_material\_id\_update\_flag[ k ] ) |  |
| **soi\_material\_id**[ k ] | u(16) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

* + - 1. Object label information SEI message syntax

|  |  |
| --- | --- |
| object\_label\_information( payloadSize ) { | **Descriptor** |
| **oli\_cancel\_flag** | u(1) |
| if( !oli\_cancel\_flag ) { |  |
| **oli\_label\_language\_present\_flag** | u(1) |
| if( oli\_label\_language\_present\_flag ) { |  |
| while( !byte\_aligned( ) ) |  |
| **oli\_bit\_equal\_to\_zero** /\* equal to 0 \*/ | f(1) |
| **oli\_label\_language** | st(v) |
| } |  |
| **oli\_num\_label\_updates** | ue(v) |
| for( i = 0; i < oli\_num\_label\_updates; i++ ) { |  |
| **oli\_label\_idx**[ i ] | ue(v) |
| k = oli\_label\_idx[ i ] |  |
| **oli\_label\_cancel\_flag**[ k ] | u(1) |
| LabelSetFlag[ k ] = !oli\_label\_cancel\_flag[ k ] |  |
| if( !oli\_label\_cancel\_flag[ k ] ) { |  |
| while( !byte\_aligned( ) ) |  |
| **oli\_bit\_equal\_to\_zero** /\* equal to 0 \*/ | f(1) |
| **oli\_label**[ k ] | st(v) |
| } |  |
| } |  |
| **oli\_persistence\_flag** | u(1) |
| } |  |
| } |  |

* + - 1. Patch information SEI message syntax

|  |  |
| --- | --- |
| patch\_information( payloadSize ) { | **Descriptor** |
| **pi\_persistence\_flag** | u(1) |
| **pi\_reset\_flag** | u(1) |
| **pi\_num\_tile\_updates** | ue(v) |
| if( pi\_num\_tile\_updates > 0 ) { |  | |
| **pi\_log2\_max\_object\_idx\_tracked\_minus1** | u(5) | |
| **pi\_log2\_max\_patch\_idx\_updated\_minus1** | u(4) | |
| } |  | |
| for( i = 0; i < pi\_num\_tile\_updates; i++ ) { |  |
| **pi\_tile\_id**[ i ] | ue(v) |
| j = pi\_tile\_id[ i ] |  |
| **pi\_tile\_cancel\_flag**[ j ] | u(1) |
| **pi\_num\_patch\_updates**[ j ] | ue(v) |
| for( k = 0; k < pi\_num\_patch\_updates[ j ]; k++ ) { |  |
| **pi\_patch\_idx**[ j ][ k ] | u(v) |
| p = pi\_patch\_idx[ j ][ k ] |  |
| **pi\_patch\_cancel\_flag**[ j ][ p ] | u(1) |
| if( !pi\_patch\_cancel\_flag[ j ][ p ] ) { |  |
| **pi\_patch\_number\_of\_objects\_minus1**[ j ][ p ] | ue(v) |
| m = pi\_patch\_number\_of\_objects\_minus1[ j ][ p ] + 1 |  |
| for( n = 0; n < m; n++ ) |  |
| **pi\_patch\_object\_idx**[ j ][ p ][ n ] | u(v) |
| } |  |
| } |  |
| } |  |
| } |  |

* + - 1. Volumetric rectangle information SEI message syntax

|  |  |
| --- | --- |
| volumetric\_rectangle\_information( payloadSize ) { | **Descriptor** |
| **vri\_persistence\_flag** | u(1) |
| **vri\_reset\_flag** | u(1) |
| **vri\_num\_rectangles\_updates** | ue(v) |
| if( vri\_num\_rectangles\_updates > 0 ) { |  |
| **vri\_log2\_max\_object\_idx\_tracked\_minus1** | u(5) |
| **vri\_log2\_max\_rectangle\_idx\_updated\_minus1** | u(4) |
| } |  |
| for( k = 0; k < vri\_num\_rectangles\_updates; k++ ) { |  |
| **vri\_rectangle\_idx**[ k ] | u(v) |
| p = vri\_rectangle\_idx[ k ] |  |
| **vri\_rectangle\_cancel\_flag**[ p ] | u(1) |
| if( !vri\_rectangle\_cancel\_flag[ p ] ) { |  |
| **vri\_bounding\_box\_update\_flag**[ p ] | u(1) |
| if( vri\_bounding\_box\_update\_flag[ p ] ) { |  |
| **vri\_bounding\_box\_top**[ p ] | u(v) |
| **vri\_bounding\_box\_left**[ p ] | u(v) |
| **vri\_bounding\_box\_width**[ p ] | u(v) |
| **vri\_bounding\_box\_height**[ p ] | u(v) |
| } |  |
| **vri\_rectangle\_number\_of\_objects\_minus1**[ p ] | ue(v) |
| m = vri\_rectangle\_number\_of\_objects\_minus1[ p ] + 1 |  |
| for( n = 0; n < m; n++ ) |  |
| **vri\_rectangle\_object\_idx**[ p ][ n ] | u(v) |
| } |  |
| } |  |
| } |  |

* + - 1. Atlas object association SEI message syntax

|  |  |
| --- | --- |
| atlas\_object\_association( payloadSize ) { | **Descriptor** |
| **aoa\_persistence\_flag** | u(1) |
| **aoa\_reset\_flag** | u(1) |
| **aoa\_num\_atlases\_minus1** | u(6) |
| **aoa\_num\_updates** | ue(v) |
| if( aoa\_num\_updates > 0 ) { |  |
| **aoa\_log2\_max\_object\_idx\_tracked\_minus1** | u(5) |
| for( j = 0; j < aoa\_num\_atlases\_minus1 + 1; j++ ) |  |
| **aoa\_atlas\_id**[ j ] | u(6) |
| for( i = 0; i < aoa\_num\_updates ; i++ ) { |  |
| **aoa\_object\_idx**[ i ] | u(v) |
| k = aoa\_object\_idx[ i ] |  |
| for( j = 0; j < aoa\_num\_atlases\_minus1 + 1; j++) |  |
| **aoa\_object\_in\_atlas**[ k ][ aoa\_atlas\_id[ j ] ] | u(1) |
| } |  |
| } |  |
| } |  |

* + 1. Buffering period SEI message syntax

|  |  |
| --- | --- |
| buffering\_period( payloadSize ) { | **Descriptor** |
| **bp\_nal\_hrd\_params\_present\_flag** | u(1) |
| **bp\_acl\_hrd\_params\_present\_flag** | u(1) |
| **bp\_initial\_cab\_removal\_delay\_length\_minus1** | u(5) |
| **bp\_au\_cab\_removal\_delay\_length\_minus1** | u(5) |
| **bp\_dab\_output\_delay\_length\_minus1** | u(5) |
| **bp\_irap\_cab\_params\_present\_flag** | u(1) |
| if( bp\_irap\_cab\_params\_present\_flag ) { |  |
| **bp\_cab\_delay\_offset** | u(v) |
| **bp\_dab\_delay\_offset** | u(v) |
| } |  |
| **bp\_concatenation\_flag** | u(1) |
| **bp\_atlas\_cab\_removal\_delay\_delta\_minus1** | u(v) |
| **bp\_max\_sub\_layers\_minus1** | u(3) |
| for( i = 0; i <= bp\_max\_sub\_layers\_minus1; i++ ) { |  |
| **bp\_hrd\_cab\_cnt\_minus1**[ i ] | ue(v) |
| if( bp\_nal\_hrd\_params\_present\_flag ) { |  |
| for( j = 0; j < bp\_hrd\_cab\_cnt\_minus1[ i ] + 1; j++ ) { |  |
| **bp\_nal\_initial\_cab\_removal\_delay**[ i ][ j ] | u(v) |
| **bp\_nal\_initial\_cab\_removal\_offset**[ i ][ j ] | u(v) |
| if( bp\_irap\_cab\_params\_present\_flag ) { |  |
| **bp\_nal\_initial\_alt\_cab\_removal\_delay**[ i ][ j ] | u(v) |
| **bp\_nal\_initial\_alt\_cab\_removal\_offset**[ i ][ j ] | u(v) |
| } |  |
| } |  |
| } |  |
| if( bp\_acl\_hrd\_params\_present\_flag ) { |  |
| for( j = 0; j < bp\_hrd\_cab\_cnt\_minus1[ i ] + 1; j++ ) { |  |
| **bp\_acl\_initial\_cab\_removal\_delay**[ i ][ j ] | u(v) |
| **bp\_acl\_initial\_cab\_removal\_offset**[ i ][ j ] | u(v) |
| if( bp\_irap\_cab\_params\_present\_flag ) { |  |
| **bp\_acl\_initial\_alt\_cab\_removal\_delay**[ i ][ j ] | u(v) |
| **bp\_acl\_initial\_alt\_cab\_removal\_offset**[ i ][ j ] | u(v) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

* + 1. Atlas frame timing SEI message syntax

|  |  |
| --- | --- |
| atlas\_frame\_timing( payloadSize ) { | Descriptor |
| if( CabDabDelaysPresentFlag ) { |  |
| for( i = 0; i <= bp\_max\_sub\_layers\_minus1; i++ ) { |  |
| **aft\_au\_cab\_removal\_delay\_minus1**[ i ] | u(v) |
| **aft\_dab\_output\_delay**[ i ] | u(v) |
| } |  |
| } |  |
| } |  |

* + 1. Viewport SEI messages family syntax
       1. Viewport camera parameters SEI messages syntax

|  |  |
| --- | --- |
| viewport\_camera\_parameters( payloadSize ) { | **Descriptor** |
| **vcp\_camera\_id** | u(10) |
| **vcp\_cancel\_flag** | u(1) |
| if( vcp\_camera\_id > 0 && !vcp\_cancel\_flag ) { |  |
| **vcp\_persistence\_flag** | u(1) |
| **vcp\_camera\_type** | u(3) |
| if( vcp\_camera\_type == 0 ) { /\* equirectangular \*/ |  |
| **vcp\_erp\_horizontal\_fov** | u(32) |
| **vcp\_erp\_vertical\_fov** | u(32) |
| } else if( vcp\_camera\_type == 1 ) { /\* perspective \*/ |  |
| **vcp\_perspective\_aspect\_ratio** | fl(32) |
| **vcp\_perspective\_horizontal\_fov** | u(32) |
| } else if( vcp\_camera\_type == 2 ) { /\* orthographic \*/ |  |
| **vcp\_ortho\_aspect\_ratio** | fl(32) |
| **vcp\_ortho\_horizontal\_size** | fl(32) |
| } |  |
| **vcp\_clipping\_near\_plane** | fl(32) |
| **vcp\_clipping\_far\_plane** | fl(32) |
| } |  |
| } |  |

* + - 1. Viewport position SEI messages syntax

|  |  |
| --- | --- |
| viewport\_position( payloadSize ) { | **Descriptor** |
| **vp\_viewport\_id** | ue(v) |
| **vp\_camera\_parameters\_present\_flag** | u(1) |
| if( vp\_camera\_parameters\_present\_flag ) |  |
| **vp\_vcp\_camera\_id** | u(10) |
| **vp\_cancel\_flag** | u(1) |
| if( !vp\_cancel\_flag ) { |  |
| **vp\_persistence\_flag** | u(1) |
| for( d = 0 ; d < 3; d++) |  |
| **vp\_position**[d] | fl(32) |
| **vp\_rotation\_qx** | i(16) |
| **vp\_rotation\_qy** | i(16) |
| **vp\_rotation\_qz** | i(16) |
| **vp\_center\_view\_flag** | u(1) |
| if( !vp\_center\_view\_flag ) |  |
| **vp\_left\_view\_flag** | u(1) |
| } |  |
| } |  |

* + 1. Decoded Atlas Information Hash SEI message syntax

|  |  |
| --- | --- |
| decoded\_atlas\_information\_hash( payloadSize ) { | Descriptor |
| **daih\_cancel\_flag** | u(1) |
| if( !daih\_cancel\_flag ) { |  |
| **daih\_persistence\_flag** | u(1) |
| **daih\_hash\_type** | u(8) |
| **daih\_decoded\_high\_level\_hash\_present\_flag** | u(1) |
| **daih\_decoded\_atlas\_hash\_present\_flag** | u(1) |
| **daih\_decoded\_atlas\_b2p\_hash\_present\_flag** | u(1) |
| **daih\_decoded\_atlas\_tiles\_hash\_present\_flag** | u(1) |
| **daih\_decoded\_atlas\_tiles\_b2p\_hash\_present\_flag** | u(1) |
| **daih\_reserved\_zero\_1bit** | u(1) |
| if( daih\_decoded\_high\_level\_hash\_present\_flag ) |  |
| decoded\_high\_level\_hash( daih\_hash\_type ) |  |
| if( daih\_decoded\_atlas\_hash\_present\_flag ) |  |
| decoded\_atlas\_hash( daih\_hash\_type ) |  |
| if( daih\_decoded\_atlas\_b2p\_hash\_present\_flag ) |  |
| decoded\_atlas\_b2p\_hash( daih\_hash\_type ) |  |
| if( daih\_decoded\_atlas\_tiles\_hash\_present\_flag ||   daih\_decoded\_atlas\_tiles\_b2p\_hash\_present\_flag ) { |  |
| **daih\_num\_tiles\_minus1** | ue(v) |
| **daih\_tile\_id\_len\_minus1** | ue(v) |
| for( t = 0; t <= daih**\_**num\_tiles\_minus1; t++ ) |  |
| **daih\_tile\_id**[ t ] | u(v) |
| byte\_align( ) |  |
| for( t = 0; t <= daih**\_**num\_tiles\_minus1; t++ ) { |  |
| j = daih\_tile\_id[ t] |  |
| if( daih\_decoded\_atlas\_tiles\_hash\_present\_flag ) |  |
| decoded\_atlas\_tile\_hash( daih\_hash\_type, j ) |  |
| if( daih\_decoded\_atlas\_tiles\_b2p\_hash\_present\_flag ) |  |
| decoded\_atlas\_tile\_b2p\_hash( daih\_hash\_type, j ) |  |
| } |  |
| } |  |
| } |  |
| } |  |

* + - 1. Decoded high level hash unit syntax

|  |  |
| --- | --- |
| decoded\_high\_level\_hash( hashType ) { |  |
| if( hashType == 0 ) |  |
| for( i = 0; i < 16; i++ ) |  |
| **daih\_high\_level\_md5**[ i ] | b(8) |
| else if( hashType == 1 ) |  |
| **daih\_high\_level\_crc** | u(16) |
| else if( hashType == 2 ) |  |
| **daih\_high\_level\_checksum** | u(32) |
| } |  |

* + - 1. Decoded atlas hash unit syntax

|  |  |
| --- | --- |
| decoded\_atlas\_tile\_hash( hashType ) { |  |
| if( hashType == 0 ) |  |
| for( i = 0; i < 16; i++ ) |  |
| **daih\_atlas\_md5**[ i ] | b(8) |
| else if( hashType == 1 ) |  |
| **daih\_atlas\_crc** | u(16) |
| else if( hashType == 2 ) |  |
| **daih\_atlas\_checksum** | u(32) |
| } |  |

* + - 1. Decoded atlas b2p hash unit syntax

|  |  |
| --- | --- |
| decoded\_atlas\_tile\_b2p\_hash( hashType ) { |  |
| if( hashType == 0 ) |  |
| for( i = 0; i < 16; i++ ) |  |
| **daih\_atlas\_b2p\_md5**[ i ] | b(8) |
| else if( hashType == 1 ) |  |
| **daih\_atlas\_b2p\_crc** | u(16) |
| else if( hashType == 2 ) |  |
| **daih\_atlas\_b2p\_checksum** | u(32) |
| } |  |

* + - 1. Decoded atlas tile hash unit syntax

|  |  |
| --- | --- |
| decoded\_atlas\_tile\_hash( hashType, j ) { |  |
| if( hashType == 0 ) |  |
| for( i = 0; i < 16; i++ ) |  |
| **daih\_atlas\_tile\_md5**[ j ][ i ] | b(8) |
| else if( hashType == 1 ) |  |
| **daih\_atlas\_tile\_crc**[ j ] | u(16) |
| else if( hashType == 2 ) |  |
| **daih\_atlas\_tile\_checksum**[ j ] | u(32) |
| } |  |

* + - 1. Decoded atlas tile b2p hash unit syntax

|  |  |
| --- | --- |
| decoded\_atlas\_b2p\_hash( hashType, j ) { |  |
| if( hashType == 0 ) |  |
| for( i = 0; i < 16; i++ ) |  |
| **daih\_atlas\_tile\_b2p\_md5**[ j ][ i ] | b(8) |
| else if( hashType == 1 ) |  |
| **daih\_atlas\_tile\_b2p\_crc**[ j ] | u(16) |
| else if( hashType == 2 ) |  |
| **daih\_atlas\_tile\_b2p\_checksum**[ j ] | u(32) |
| } |  |

* + 1. Time code SEI message syntax

|  |  |
| --- | --- |
| time\_code( payloadSize ) { | **Descriptor** |
| **num\_units\_in\_tick** | u(32) |
| **time\_scale** | u(32) |
| **counting\_type** | u(5) |
| **full\_timestamp\_flag** | u(1) |
| **discontinuity\_flag** | u(1) |
| **cnt\_dropped\_flag** | u(1) |
| **n\_frames** | u(9) |
| if( full\_timestamp\_flag ) { |  |
| **seconds\_value** /\* 0..59 \*/ | u(6) |
| **minutes\_value** /\* 0..59 \*/ | u(6) |
| **hours\_value** /\* 0..23 \*/ | u(5) |
| } else { |  |
| **seconds\_flag** | u(1) |
| if( seconds\_flag ) { |  |
| **seconds\_value** /\* range 0..59 \*/ | u(6) |
| **minutes\_flag** | u(1) |
| if( minutes\_flag ) { |  |
| **minutes\_value** /\* 0..59 \*/ | u(6) |
| **hours\_flag** | u(1) |
| if( hours\_flag ) |  |
| **hours\_value** /\* 0..23 \*/ | u(5) |
| } |  |
| } |  |
| } |  |
| **time\_offset\_length** | u(5) |
| if( time\_offset\_length > 0 ) |  |
| **time\_offset\_value** | i(v) |
| } |  |

* + 1. Packed independent regions SEI payload syntax

|  |  |
| --- | --- |
| packed\_independent\_regions( ) { | Descriptor |
| **pir\_num\_packed\_frames\_minus1** | u(5) |
| for( j = 0; j <= pir\_num\_packed\_frames\_minus1; j++ ) { |  |
| **pir\_packed\_frame\_id**[ j ] | u(5) |
| k = pir\_packed\_frame\_id[ j ] |  |
| **pir\_description\_type\_idc**[ k ] | u(2) |
| **pir\_num\_regions\_minus1**[ k ] | u(8) |
| for( i = 0;  i <= pir\_num\_regions\_minus1[ k ]; i++) { |  |
| if( pir\_description\_type\_idc[ k ]  = =  0 ) { |  |
| **pir\_top\_left\_tile\_idx**[ k ][ i ] | ue(v) |
| **pir\_bottom\_right\_tile\_idx**[ k ][ i ] | ue(v) |
| } |  |
| else if( pir\_description\_type\_idc  = =  1 ) { |  |
| **pir\_subpic\_id**[ k ][ i ] | ue(v) |
| } |  |
| } |  |
| } |  |
| } |  |

* 1. SEI payload semantics
     1. General SEI payload semantics

**sp\_reserved\_payload\_extension\_data** shall not be present in bitstreams conforming to this version of this document. However, decoders conforming to this version of this document shall ignore the presence and value of sp\_reserved\_payload\_extension\_data. When present, the length, in bits, of sp\_reserved\_payload\_extension\_data is equal to 8 \* payloadSize − nEarlierBits − nPayloadZeroBits − 1, where nEarlierBits is the number of bits in the sei\_payload( ) syntax structure that precede the sp\_reserved\_payload\_extension\_data syntax element and nPayloadZeroBits is the number of payload\_bit\_equal\_to\_zero syntax elements at the end of the sei\_payload( ) syntax structure.

NOTE 1 – SEI messages with the same value of payloadType are conceptually the same SEI message regardless of whether they are contained in prefix or suffix SEI NAL units.

The semantics and persistence scope for each SEI message are specified in the semantics specification for each particular SEI message. Unless indicated otherwise, all defined SEI messages and their persistence scope are associated only with the current atlas with atlas ID SeiAtlasID, or the common atlas.

NOTE 2 – Persistence information for SEI messages is informatively summarized in Table F-2.

|  |  |
| --- | --- |
| Table F-2 – Persistence scope of SEI messages | |
| SEI message | Persistence scope |
| Buffering period | The remainder of the bitstream |
| Atlas frame timing | The access unit containing the SEI message |
| Filler payload | The access unit containing the SEI message |
| User data registered by  Rec. ITU-T T.35 | Unspecified |
| User data unregistered | Unspecified |
| Recovery point | Specified by the syntax of the SEI message |
| Decoded atlas information hash | Specified by the syntax of the SEI message |
| No display | The coded atlas access unit containing the SEI message |
| Time code | The coded atlas access unit containing the SEI message |
| SEI manifest | The remainder of the bitstream |
| SEI prefix indication | The remainder of the bitstream |
| Active sub-bitstreams | The remainder of the bitstream or until a new active attributes SEI message |
| Component codec mapping | The remainder of the bitstream or until a new active attributes SEI message |
| Scene object information | Specified by the syntax of the SEI message |
| Object label information | Specified by the syntax of the SEI message |
| Patch information | Specified by the syntax of the SEI message |
| Volumetric rectangle information | Specified by the syntax of the SEI message |
| Atlas object association | Specified by the syntax of the SEI message |
| Viewport camera parameters | Specified by the syntax of the SEI message |
| Viewport position | Specified by the syntax of the SEI message |
| Attribute transformation parameters | Specified by the syntax of the SEI message |
| Occupancy synthesis | Specified by the syntax of the SEI message |
| Geometry smoothing | Specified by the syntax of the SEI message |
| Attribute smoothing | Specified by the syntax of the SEI message |
| Packed independent regions | The remainder of the bitstream |

The values of some SEI message syntax elements are split into two sets of value ranges, where the first set is specified as "may be used as determined by the application", and the second set is specified as "reserved for future use by ISO/IEC". Applications should be cautious of potential “collisions” of the interpretation for values of these syntax elements belonging to the first set of value ranges. Since different applications might use these IDs having values in the first set of value ranges for different purposes, particular care should be exercised in the design of encoders that generate SEI messages with these IDs having values in the first set of value ranges, and in the design of decoders that interpret SEI messages with these IDs. This document does not define any management for these values. These IDs having values in the first set of value ranges might only be suitable for use in contexts in which "collisions" of usage (i.e., different definitions of the syntax and semantics of an SEI message with one of these IDs having the same value in the first set of value ranges) are unimportant, or not possible, or are managed – e.g., defined or managed in the controlling application or transport specification, or by controlling the environment in which bitstreams are distributed.

It is a requirement of bitstream conformance that the following restrictions apply on containing of SEI messages in SEI NAL units:

* When an SEI NAL unit contains a buffering period SEI message or an atlas frame timing SEI message, the SEI NAL unit shall not contain any other SEI message with payloadType not equal to 0 (buffering period) or 1 (atlas frame timing).

Let prevAclNalUnitInAu of an SEI NAL unit or an SEI message be the preceding ACL NAL unit in decoding order, if any, in the same access unit, and nextAclNalUnitInAu of an SEI NAL unit or an SEI message be the next ACL NAL unit in decoding order, if any, in the same access unit.

It is a requirement of bitstream conformance that the following restrictions apply on decoding order of SEI messages:

* When a buffering period SEI message is present in an access unit, it shall not follow any other SEI message that follows the prevAclNalUnitInAu of the buffering period SEI message and precedes the nextAclNalUnitInAu of the buffering period SEI message.
* When an atlas frame timing SEI message is present in an access unit, it shall not follow any other SEI message that follows the prevAclNalUnitInAu of the atlas frame timing SEI message and precedes the nextAclNalUnitInAu of the atlas frame timing SEI message.
* When payloadType is equal to 0 (buffering period) or 1 (atlas frame timing), the SEI NAL unit containing the SEI message shall precede all NAL units of any atlas unit that has nal\_layer\_id greater than highestAppLayerId, where highestAppLayerId is the greatest value of nal\_layer\_id of all the layers in all the operation points that the SEI message applies to.

The following applies on the applicable operation points or layers of SEI messages:

* For an SEI message when payloadType is equal to 0 (buffering period) the SEI message applies to the operation point that has OpTid equal to the greatest value of nal\_temporal\_id\_plus1 among all ACL NAL units in the bitstream, and that has OpLayerIdList containing all values of nal\_layer\_id in all ACL units in the bitstream..
* For an SEI message, when payloadType is equal to 1 (atlas frame timing), the SEI message applies to the operation point that has OpTid equal to the greatest value of nal\_temporal\_id\_plus1 among all ACL NAL units in the bitstream, and that has OpLayerIdList containing all values of nal\_layer\_id in all ACL units in the bitstream.

It is a requirement of bitstream conformance that the following restrictions apply on the values of nal\_layer\_id and TemporalId of SEI NAL units:

* When an SEI message has payloadType equal to 0 or 1 the SEI NAL unit containing the SEI message shall have nal\_layer\_id equal to 0.

In the following subclauses of this annex, the following applies:

* The current SEI message refers to the particular SEI message.
* The current access unit refers to the access unit containing the current SEI message.

In the following subclauses of this annex, when a particular SEI message applies to a set of one or more maps (instead of a set of operation points), i.e., when the payloadType value is not equal to one of 0 (buffering period) and 1 (atlas frame timing), the following applies:

* The semantics apply independently to each unit the particular SEI message applies.

In the following subclauses of this annex, when a particular SEI message applies to a set of one or more operation points (instead of a set of one or more maps), i.e., when the payloadType value is equal to 0 (buffering period) or 1 (atlas frame timing), the following applies:

* The semantics apply independently to each particular operation point of the set of operation points to which the particular SEI message applies.
* The current operation point refers to the particular operation point.
* The terms "access unit" and "CAS" apply to the bitstream BitstreamToDecode that is the sub-bitstream of the particular operation point.
  + 1. Filler payload SEI message semantics

This SEI message contains a series of payloadSize bytes of value 0xFF, which can be discarded.

**ff\_byte** shall be a byte having the value 0xFF.

* + 1. User data registered by Recommendation ITU-T T.35 SEI message semantics

This SEI message contains user data registered as specified in Recommendation ITU-T T.35, the contents of which are not specified in this document.

**itu\_t\_t35\_country\_code** shall be a byte having a value specified as a country code by Recommendation ITU-T T.35:2000, Annex A.

**itu\_t\_t35\_country\_code\_extension\_byte** shall be a byte having a value specified as a country code by Recommendation ITU-T T.35:2000, Annex B.

**itu\_t\_t35\_payload\_byte** shall be a byte containing data registered as specified in Recommendation ITU-T T.35.

The ITU-T T.35 terminal provider code and terminal provider oriented code shall be contained in the first one or more bytes of the itu\_t\_t35\_payload\_byte, in the format specified by the Administration that issued the terminal provider code. Any remaining itu\_t\_t35\_payload\_byte data shall be data having syntax and semantics as specified by the entity identified by the ITU-T T.35 country code and terminal provider code.

* + 1. User data unregistered SEI message semantics

This SEI message contains unregistered user data identified by a universal unique identifier (UUID), the contents of which are not specified in this document.

**uuid\_iso\_iec\_11578** shall have a value specified as a UUID according to the procedures of ISO/IEC 11578:1996, Annex A.

**user\_data\_payload\_byte** shall be a byte containing data having syntax and semantics as specified by the UUID generator.

* + 1. Recovery point SEI message semantics

The recovery point SEI message assists a decoder in determining when the decoding process will produce acceptable atlas frames for reconstruction and display after the decoder initiates random access or after the encoder indicates a broken link in the CAS. When the decoding process is started with the access unit in decoding order associated with the recovery point SEI message, all decoded atlas frames at or subsequent to the recovery point in output order specified in this SEI message are indicated to be correct or approximately correct in content. Decoded atlas frames produced by random access at or before the atlas frame associated with the recovery point SEI message need not be correct in content until the indicated recovery point, and the operation of the decoding process starting at the atlas frame associated with the recovery point SEI message may contain references to atlas frames and related video data that might be unavailable for prediction.

In addition, by use of the broken\_link\_flag, the recovery point SEI message can indicate to the decoder the location of some atlas frames in the bitstream that can result in serious visual artefacts when reconstructed and displayed, because of potentially missing references.

NOTE 1 – The broken\_link\_flag can be used by encoders to indicate the location of a point after which the decoding process for the decoding of some atlas frames can cause references to information that, though available for use in the decoding process, is not the information that was used for reference when the bitstream was originally encoded (e.g., due to a splicing operation performed during the generation of the bitstream).

When random access is performed to start decoding from the access unit associated with the recovery point SEI message, the decoder operates as if the associated atlas frame was the first atlas frame in the bitstream in decoding order, and the variables prevAtlasFrmOrderCntLsb and prevAtlasFrmOrderCntMsb used in the derivation of AtlasFrmOrderCntVal are both set equal to 0.

NOTE 2 – When HRD information is present in the bitstream, a buffering period SEI message could be associated with the access unit associated with the recovery point SEI message in order to establish initialization of the HRD buffer model after a random access.

Any ASPS or AFPS RBSP that is referred to by an atlas frame associated with a recovery point SEI message or by any atlas frame following such an atlas frame in decoding order shall be available to the decoding process prior to its activation, regardless of whether or not the decoding process is started at the beginning of the bitstream or with the access unit, in decoding order, that is associated with the recovery point SEI message.

**recovery\_afoc\_cnt** specifies the recovery point of decoded atlas frames in output order. If there is an atlas frame aFrmA that follows the current atlas frame (i.e., the atlas frame associated with the current SEI message) in decoding order in the CAS and that has AtlasFrmOrderCntVal equal to the AtlasFrmOrderCntVal of the current atlas frame plus the value of recovery\_afoc\_cnt, the atlas frame aFrmA is referred to as the recovery point atlas frame. Otherwise, the first atlas frame in output order that has AtlasFrmOrderCntVal greater than the AtlasFrmOrderCntVal of the current atlas frame plus the value of recovery\_afoc\_cnt is referred to as the recovery point atlas frame. The recovery point atlas frame shall not precede the current atlas frame in decoding order. All decoded atlas frames in output order are indicated to be correct or approximately correct in content starting at the output order position of the recovery point atlas frame. The value of recovery\_afoc\_cnt shall be in the range of −MaxAtlasFrmOrderCntLsb / 2 to MaxAtlasFrmOrderCntLsb / 2 − 1, inclusive.

**exact\_match\_flag** indicates whether decoded atlas frames at and subsequent to the specified recovery point in output order derived by starting the decoding process at the access unit associated with the recovery point SEI message will be an exact match to the atlas frames that would be produced by starting the decoding process at the location of a previous IRAP coded atlas access unit, if any, in the bitstream. The value 0 indicates that the match may not be exact and the value 1 indicates that the match will be exact. When exact\_match\_flag is equal to 1, it is a requirement of bitstream conformance that the decoded atlas frames at and subsequent to the specified recovery point in output order derived by starting the decoding process at the access unit associated with the recovery point SEI message shall be an exact match to the atlas frames that would be produced by starting the decoding process at the location of a previous IRAP coded atlas access unit, if any, in the bitstream.

When exact\_match\_flag is equal to 0, the quality of the approximation at the recovery point is chosen by the encoding process and is not specified in this document.

**broken\_link\_flag** indicates the presence or absence of a broken link in the atlas data unit stream at the location of the recovery point SEI message and is assigned further semantics as follows:

* If broken\_link\_flag is equal to 1, atlas frames produced by starting the decoding process at the location of a previous IRAP coded atlas access unit may contain undesirable visual artefacts to the extent that decoded atlas frames at and subsequent to the access unit associated with the recovery point SEI message in decoding order should not be reconstructed and displayed until the specified recovery point in output order.
* Otherwise (broken\_link\_flag is equal to 0), no indication is given regarding any potential presence of visual artefacts.

Regardless of the value of the broken\_link\_flag, atlas frames subsequent to the specified recovery point in output order are specified to be correct or approximately correct in content.

* + 1. No reconstruction SEI message semantics

The no reconstruction SEI message indicates that the current atlas frame should not be used for reconstruction and that no V3C volumetric frame should be reconstructed and displayed using this atlas frame.

* + 1. Reserved SEI message semantics

The reserved SEI message consists of data reserved for future backward-compatible use by ISO/IEC. It is a requirement of bitstream conformance that bitstreams shall not contain reserved SEI messages until and unless the use of such messages has been specified by ISO/IEC. Decoders shall ignore reserved SEI messages.

**reserved\_sei\_message\_payload\_byte** is a byte reserved for future use by ISO/IEC.

* + 1. SEI manifest SEI message semantics

The SEI manifest SEI message conveys information on SEI messages that are indicated as expected (i.e., likely) to be present or not present. Such information may include:

1. The indication that certain types of SEI messages are expected (i.e., likely) to be present (although not guaranteed to be present) in the CAS.
2. For each type of SEI message that is indicated as expected (i.e., likely) to be present in the CAS, the degree of expressed necessity of interpretation of the SEI messages of this type.

The degree of necessity of interpretation of an SEI message type may be indicated as "necessary", "unnecessary", or "undetermined".

An SEI message is indicated by the encoder (i.e., the content producer) as being "necessary" when the information conveyed by the SEI message is considered as necessary for interpretation by the decoder or receiving system in order to properly process the content and enable an adequate user experience; it does not mean that the bitstream is required to contain the SEI message in order to be a conforming bitstream. It is at the discretion of the encoder to determine which SEI messages are to be considered as necessary in a particular CAS. However, it is suggested that some SEI messages should typically be considered as necessary.

1. The indication that certain types of SEI messages are expected (i.e., likely) not to be present (although not guaranteed not to be present) in the CAS.

The content of an SEI manifest SEI message may, for example, be used by transport-layer or systems-layer processing elements to determine whether the CAS is suitable for delivery to a receiving and decoding system, based on whether the receiving system can properly process the CAS to enable an adequate user experience or whether the CAS satisfies the application needs.

When an SEI manifest SEI message is present in any access unit of a CAS, an SEI manifest SEI message shall be present in the first access unit of the CAS. The SEI manifest SEI message persists in decoding order from the current access unit until the end of the CAS. When there are multiple SEI manifest SEI messages present in a CAS, they shall have the same content.

**manifest\_num\_sei\_msg\_types** specifies the number of types of SEI messages for which information is provided in the SEI manifest SEI message.

**manifest\_sei\_payload\_type**[ i ] indicates the payloadType value of the i-th type of SEI message for which information is provided in the SEI manifest SEI message. The values of manifest\_sei\_payload\_type[ m ] and manifest\_sei\_payload\_type[ n ]shall not be identical when m is not equal to n.

**manifest\_sei\_description**[ i ] provides information on SEI messages with payloadType equal to manifest\_sei\_payload\_type[ i ] as specified in Table F-3.

Table F-3 – manifest\_sei\_description[ i ] values

|  |  |
| --- | --- |
| **Value** | **Description** |
| 0 | Indicates that there is no SEI message with payloadType equal to manifest\_sei\_payload\_type[ i ] expected to be present in the CAS. |
| 1 | Indicates that there are SEI messages with payloadType equal to manifest\_sei\_payload\_type[ i ] expected to be present in the CAS, and these SEI messages are considered as necessary. |
| 2 | Indicates that there are SEI messages with payloadType equal to manifest\_sei\_payload\_type[ i ] expected to be present in the CAS, and these SEI messages are considered as unnecessary. |
| 3 | Indicates that there are SEI messages with payloadType equal to manifest\_sei\_payload\_type[ i ] expected to be present in the CAS, and the necessity of these SEI messages is undetermined. |
| 4-255 | Reserved |

The value of manifest\_sei\_description[ i ] shall be in the range of 0 to 3, inclusive, in bitstreams conforming to this version of this document. Other values for manifest\_sei\_description[ i ] are reserved for future use by ISO/IEC. Decoders shall allow the value of manifest\_sei\_description[ i ] greater than or equal to 4 to appear in the syntax and shall ignore all information for payloadType equal to manifest\_sei\_payload\_type[ i ] signalled in the SEI manifest SEI message and shall ignore all SEI prefix indication SEI messages with prefix\_sei\_payload\_type equal to manifest\_sei\_payload\_type[ i ] when manifest\_sei\_description[ i ] is greater than or equal to 4.

* + 1. SEI prefix indication SEI message semantics

The SEI prefix indication SEI message carries one or more SEI prefix indications for SEI messages of a particular value of payloadType. Each SEI prefix indication is a bit string that follows the SEI payload syntax of that value of payloadType and contains a number of complete syntax elements starting from the first syntax element in the SEI payload.

Each SEI prefix indication for an SEI message of a particular value of payloadType indicates that one or more SEI messages of this value of payloadType are expected (i.e., likely) to be present in the CAS and to start with the provided bit string. A starting bit string would typically contain only a true subset of an SEI payload of the type of SEI message indicated by the payloadType, may contain a complete SEI payload, and shall not contain more than a complete SEI payload. It is not prohibited for SEI messages of the indicated value of payloadType to be present that do not start with any of the indicated bit strings.

These SEI prefix indications should provide sufficient information for indicating what type of processing is needed or what type of content is included. The former (type of processing) indicates decoder-side processing capability, e.g., whether some type of post-filtering process is needed. The latter (type of content) indicates, for example, whether the bitstream contains subtitle captions in a particular language.

The content of an SEI prefix indication SEI message may, for example, be used by transport-layer or systems-layer processing elements to determine whether the CAS is suitable for delivery to a receiving and decoding system, based on whether the receiving system can properly process the CAS to enable an adequate user experience or whether the CAS satisfies the application needs (as determined in some manner by external means outside the scope of this document).

In one example, for user data registered SEI messages that are used to carry captioning information, an SEI prefix indication should include up to at least the language code; and for user data unregistered SEI messages extended for private use, an SEI prefix indication should include up to at least the UUID.

When an SEI prefix indication SEI message is present in any access unit of a CAS, an SEI prefix indication SEI message shall be present in the first access unit of the CAS. The SEI prefix indication SEI message persists in decoding order from the current access unit until the end of the CAS. When there are multiple SEI prefix indication SEI messages present in a CAS for a particular value of payloadType, they shall have the same content.

**prefix\_sei\_payload\_type** indicates the payloadType value of the SEI messages for which one or more SEI prefix indications are provided in the SEI prefix indication SEI message. When an SEI manifest SEI message is also present for the CAS, the value of prefix\_sei\_payload\_type shall be equal to one of the manifest\_sei\_payload\_type[ m ] values for which manifest\_sei\_description[ m ] is equal to 1 to 3, inclusive, as indicated by an SEI manifest SEI message that applies to the CAS.

**num\_sei\_prefix\_indications\_minus1** plus 1 specifies the number of SEI prefix indications.

**num\_bits\_in\_prefix\_indication\_minus1**[ i ] plus 1 specifies the number of bits in the i-th SEI prefix indication.

**sei\_prefix\_data\_bit**[ i ][ j ] specifies the j-th bit of the i-th SEI prefix indication.

The bits sei\_prefix\_data\_bit[ i ][ j ] for j ranging from 0 to num\_bits\_in\_prefix\_indication\_minus1[ i ], inclusive, follow the syntax of the SEI payload with payloadType equal to prefix\_sei\_payload\_type, and contain a number of complete syntax elements starting from the first syntax element in the SEI payload syntax, and may or may not contain all the syntax elements in the SEI payload syntax. The last bit of these bits (i.e., the bit sei\_prefix\_data\_bit[ i ][ num\_bits\_in\_prefix\_indication\_minus1[ i ] ]) shall be the last bit of a syntax element in the SEI payload syntax, unless it is a bit within an itu\_t\_t35\_payload\_byte or user\_data\_payload\_byte.

NOTE – The exception for itu\_t\_t35\_payload\_byte and user\_data\_payload\_byte is provided because these syntax elements can contain externally-specified syntax elements, and the determination of the boundaries of such externally-specified syntax elements is a matter outside the scope of this document.

**byte\_alignment\_bit\_equal\_to\_one** shall be equal to 1.

* + 1. Active sub-bitstreams SEI message semantics

This SEI message informs the V3C decoder which attributes and/or maps are not available and can therefore be skipped in the decoding process. Similarly, it can also inform the decoder that auxiliary sub-bitstream are not available. This is done by signalling changes to active attributes/maps and whether the auxiliary sub-bitstreams are active or not. An active sub-bitstreams SEI message references a specific V3C parameter set. When only a sub-set of the attributes/maps shall be active, the SEI message contains the attribute/map indices for these attributes/maps. A V3C decoder shall consider any other attributes/maps in the referenced VPS that are not listed in the SEI message as inactive and skip them in the decoding and/or rendering process.

NOTE – The active sub-bitstreams SEI message allows the possibility of enabling only a subset of attributes and subset of maps to be active at any given time.

The persistence scope for this SEI message is the remainder of the bitstream (i.e., the signalled active attributes persist until the end of the stream) or when a new active attributes SEI message is encountered. Previously defined parameters from an earlier SEI message shall persist if not modified and if the value of asb\_active\_sub\_bitstream\_cancel\_flag is not equal to 1. This SEI message should not be ignored by the decoder.

The semantics of the fields of the active sub-bitstreams SEI message are as follows:

**asb\_active\_sub\_bitstream\_cancel\_flag** equal to 1 indicates that all sub-bitstreams are active. asb\_active\_sub\_bitstream\_cancel\_flag equal to 0 indicates that some sub-bitstreams may not be active.

**asb\_active\_attributes\_changes\_flag** indicates whether there are activation changes for any of the attribute sub-bitstreams of the V3C stream. asb\_active\_attributes\_changes\_flag equal to 1 indicates an activation change for at least one attribute and its associated sub-bitstreams. asb\_active\_attributes\_changes\_flag equal to 0 indicates no changes.

**asb\_active\_maps\_changes\_flag** indicates whether there are activation changes for any of the V3C map sub-bitstreams. asb\_active\_maps\_changes\_flag equal to 1 indicates an activation change for at least one map and its associated sub-bitstreams. asb\_active\_maps\_changes\_flag equal to 0 indicates no changes.

**asb\_auxiliary\_sub\_bitstreams\_active\_flag** indicates whether the auxiliary sub-bitstreams for the geometry and all attributes are active or not. asb\_auxiliary\_sub\_bitstreams\_active\_flag equal to 1 indicates that such sub-bitstreams are active. asb\_auxiliary\_sub\_bitstreams\_active\_flag equal to 0 indicates that such sub-bitstreams are not active.

**asb\_all\_attributes\_active\_flag** indicates whether all the attributes signalled in the referenced VPS shall be active. asb\_all\_attributes\_active\_flag equal to 1 indicates that all attributes shall be active. asb\_all\_attributes\_active\_flag equal to 0 indicates that only a sub-set of the attributes may be active.

**asb\_active\_attribute\_count\_minus1** plus 1 indicates the number of active attributes signalled in the active sub-bitstreams SEI message.

**asb\_active\_attribute\_idx**[ i ] indicates the attribute index in the VPS for the active attribute at index i in the associated SEI message.

**asb\_all\_maps\_active\_flag** indicates whether all the maps signalled in the referenced VPS shall be active. asb\_all\_maps\_active\_flag equal to 1 indicates that all maps shall be active. asb\_all\_maps\_active\_flag equal to 0 indicates that only a sub-set of the maps may be active.

**asb\_active\_map\_count\_minus1** plus 1 indicates the number of active maps signalled in the active sub-bitstreams SEI message.

**asb\_active\_map\_idx**[ i ] indicates the map index in the VPS for the active map at index i in the associated SEI message.

* + 1. Component codec mapping SEI message semantics

This SEI message informs the V3C decoder of the codec mapping for the codec ids of the component sub-bitstreams signalled in the VPS. Each component sub-bitstream codec id is mapped to a specific codec index in a codec lookup table. The codec ids for the component sub-bitstreams in the VPS shall be unique. The component codec mapping SEI message shall be used to signal the initial codec mapping to the decoder at the beginning of the V3C bitstream as well as signalling updated mappings when the codec of one or more of the V3C components sub-bitstreams changes. A V3C decoder receiving a component codec mapping SEI message should instantiate new video decoders for the respective component sub-bitstreams signalled in the message.

The persistence scope for this SEI message is the remainder of the bitstream (i.e., the codec changes for the signalled components persist until the end of the stream) or until a new component codec change SEI message is encountered. Only the codec mapping for codec ids specified in the SEI message shall be updated. Previously defined mappings for other codec ids from an earlier SEI message shall persist if not modified and if the value of ccm\_component\_codec\_cancel\_flag is not equal to 1. This SEI message shall not be ignored by the decoder.

When a component codec mapping SEI message is present in any access unit of a CVS, a component codec mapping SEI message shall be present in the first access unit of the CVS. The component codec mapping SEI message persists in decoding order from the current access unit until the end of the CVS. When there are multiple component codec mapping SEI messages present in a CVS, they shall have the same content.

The semantics of the fields of the component codec mapping SEI message are as follows:

**ccm\_component\_codec\_cancel\_flag** indicates whether the component codec mapping should be reset to the default mapping defined by the CodecGroup profile type for the bitstream.ccm\_component\_codec\_cancel\_flag equal to 1 indicates that the component codec mapping should be reset to the default mapping defined by the CodecGroup profile type for the bitstream. ccm\_component\_codec\_cancel\_flag equal to 0 indicates that the component codec mapping for some components may be updated.

**ccm\_codec\_mappings\_count\_minus1** plus 1 indicates the number of codec mappings that are listed in this SEI message.

**ccm\_codec\_id** is the codec id that is to be mapped to a particular 4CC codec. This codec id may be associated with one or more of the sub-bitstreams in a V3C bitstream, as specified within the active VPS.

**ccm\_codec\_4cc**[ j ] is the four-character code (4CC) for the codec mapped to the codec id of value j. The codec code shall be an MP4RA registered code according to ISO/IEC 14496-12.

* + 1. Volumetric annotation SEI messages family syntax
       1. Scene object information SEI message semantics
          1. General

This SEI message defines a set of objects that may be present in a volumetric scene, and optionally assigns different properties to these objects. These objects could then potentially be associated with different types of information, including patches and 2D volumetric rectangles that may be defined using the patch information and volumetric rectangle information SEI messages.

Let the variable MaxNumObjectsTracked be set equal to either 216 – 1 or a value determined by the application.

Let the variable MaxObjectNumDependencies be set equal to 15.

NOTE – An application may wish to limit the value of the variable MaxNumObjectsTracked so as to constrain the required memory.

At the start of each sequence,

* the 1D arrays ObjectIndex, ObjectTracked, ObjectLabelIndex, ObjectPriorityValue, ObjectHiddenFlag, ObjectNumDependencies, ObjectVisibilityDirectionX, ObjectVisibilityDirectionY, ObjectVisibilityDirectionZ, ObjectVisibilityAngle, Object3DBoundingBoxX, Object3DBoundingBoxY, Object3DBoundingBoxZ, Object3DBoundingBoxSizeX, Object3DBoundingBoxSizeY, Object3DBoundingBoxSizeZ, ObjectCollisionShapeID, ObjectPointShapeID, ObjectPointSize, and ObjectMaterialID, of size MaxNumObjectsTracked, and
* the 2D array ObjectDependencyIndex, of size MaxNumObjectsTracked × MaxObjectNumDependencies,

are initialized as follows:

for ( i = 0; i < MaxNumObjectsTracked; i++ ) {  
 ObjectIndex[ i ] = i  
 ObjectInit( ObjectIndex[ i ] )  
 }

where the function ObjectInit( ) is defined in the subclause F.3.12.1.2.

**soi\_persistence\_flag** specifies the persistence of the scene object information SEI message for the current layer. soi\_persistence\_flag equal to 0 specifies that the scene object information SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. soi\_persistence\_flag equal to 1 specifies that the scene object information SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing a scene object information SEI message with the same value of soi\_persistence\_flag and applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**soi\_reset\_flag** indicates that all arrays associated with the scene object information SEI message are reset as follows:

for ( i = 0; i < MaxNumObjectsTracked; i++ ) {  
 ObjectIndex[ i ] = i  
 ObjectInit( ObjectIndex[ i ] )  
 }

**soi\_num\_object\_updates** indicates the number of objects that are to be updated by the current SEI message. The value of soi\_num\_object\_updates shall be in the range from 0 to MaxNumObjectsTracked, inclusive. The default value of soi\_num\_object\_updates is equal to 0.

**soi\_simple\_objects\_flag** equal to 1 indicates that no additional information for an updated or newly introduced object will be signalled. soi\_simple\_objects\_flag equal to 0 indicates that additional information for an updated or newly introduced object may be signalled.

**soi\_object\_label\_present\_flag** equal to 1 indicates that object label information is present in the current scene object information SEI message. soi\_object\_label\_present\_flag equal to 0 indicates that object label information is not present.

**soi\_priority\_present\_flag** equal to 1 indicates that priority information is present in the current scene object information SEI message. soi\_priority\_present\_flag equal to 0 indicates that priority information is not present.

**soi\_object\_hidden\_present\_flag** equal to 1 indicates that hidden object information is present in the current scene object information SEI message. soi\_object\_hidden\_present\_flag equal to 0 indicates that hidden object information is not present.

**soi\_object\_dependency\_present\_flag** equal to 1 indicates that object dependency information is present in the current scene object information SEI message. soi\_object\_dependency\_present\_flag equal to 0 indicates that object dependency information is not present.

**soi\_visibility\_cones\_present\_flag** equal to 1 indicates that visibility cones information is present in the current scene object information SEI message. soi\_visibility\_cones\_present\_flag equal to 0 indicates that visibility cones information is not present.

**soi\_3d\_bounding\_box\_present\_flag** equal to 1 indicates that 3D bounding box information is present in the current scene object information SEI message. soi\_3d\_bounding\_box\_present\_flag equal to 0 indicates that 3D bounding box information is not present.

**soi\_collision\_shape\_present\_flag** equal to 1 indicates that collision information is present in the current scene object information SEI message. soi\_collision\_shape\_present\_flag equal to 0 indicates that collision shape information is not present.

**soi\_point\_style\_present\_flag** equal to 1 indicates that point style information is present in the current scene object information SEI message. soi\_point\_style\_present\_flag equal to 0 indicates that point style information is not present.

**soi\_material\_id\_present\_flag** equal to 1 indicates that material ID information is present in the current scene object information SEI message. soi\_material\_id\_present\_flag equal to 0 indicates that material ID information is not present.

**soi\_extension\_present\_flag** equal to 1 indicates that additional extension information shall be present in the current scene object information SEI message. soi\_extension\_present\_flag equal to 0 indicates that additional extension information is not present. It is a requirement of bitstream conformance to this version of this document that soi\_extension\_present\_flag shall be equal to 0.

**soi\_3d\_bounding\_box\_scale\_log2** indicates the scale to be applied to the 3D bounding box parameters that may be specified for an object.

**soi\_log2\_max\_object\_idx\_updated\_minus1** plus 1specifies the number of bitsused to signal the value of an object index in the current scene object information SEI message.

**soi\_log2\_max\_object\_dependency\_idx** specifies the number of bitsused to signal the value of a dependency object index in the current scene object information SEI message. The default value of soi\_log2\_max\_object\_dependency\_idx is equal to 0.

**soi\_object\_idx**[ i ] indicates the object index of the i-th object to be updated. The number of bits used to represent soi\_object\_idx[ i ] is equal to soi\_log2\_max\_object\_idx\_updated\_minus1 + 1.

**soi\_object\_cancel\_flag**[ i ] equal to 1 indicates that the object with index equal to i shall be canceled and that the variable ObjectTracked[ i ] shall be set to 0. Furthermore, all of its associated parameters, including the object label, 3D bounding box parameters, priority information, hidden flag, dependency information, visibility cones, collision shapes, point style, and material id, shall be reset by invoking the function ObjectInit( i ). soi\_object\_cancel\_flag[ i ] equal to 0 indicates that the object with index equal to soi\_object\_idx[ i ] shall be updated with information that follows this element and that that the variable ObjectTracked[ i ] shall be set to 1.

**soi\_object\_label\_update\_flag**[ i ] equal to 1 indicates that object label update information is present for an object with object index i. soi\_object\_label\_update\_flag[ i ] equal to 0 indicates that object label update information is not present.

**soi\_object\_label\_idx**[ i ] indicates the label index, ObjectLabelIndex[ i ], of an object with index i. The value of soi\_object\_label\_idx[ i ] shall be in the range from 0 to MaxNumObjectsTracked – 1, inclusive.

**soi\_priority\_update\_flag**[ i ] equal to 1 indicates that priority update information is present for an object with object index i. soi\_priority\_update\_flag[ i ] equal to 0 indicates that object priority information is not present.

**soi\_priority\_value**[ i ] indicates the priority, ObjectPriorityValue[ i ], of an object with index i. The lower the priority value, the higher the priority. The default value of soi\_priority\_value[ i ] is equal to 0.

**soi\_object\_hidden\_flag**[ i ] indicates the value of ObjectHiddenFlag[ i ] of an object with index i. soi\_object\_hidden\_flag[ i ] equal to 1 indicates that the object with index i shall be hidden. soi\_object\_hidden\_flag[ i ] equal to 0 indicates that the object with index i shall become present.

**soi\_object\_dependency\_update\_flag**[ i ] equal to 1 indicates that object dependency update information is present for an object with object index i. soi\_object\_dependency\_update\_flag[ i ] equal to 0 indicates that object dependency update information is not present.

**soi\_object\_num\_dependencies**[ i ] indicates the value of ObjectNumDependencies[ i ] of an object with index i, which specifies the number of object dependencies for the object with index i.

**soi\_object\_dependency\_idx**[ i ][ j ] indicates the value of ObjectDependencyIndex[ i ][ j ] of the j-th object that has a dependency with the object with object index i.

**soi\_visibility\_cones\_update\_flag**[ i ] equal to 1 indicates that visibility cones update information is present for an object with object index i. soi\_visibility\_cones\_update\_flag[ i ] equal to 0 indicates that visibility cones update information is not present.

**soi\_direction\_x**[ i ] specifies the normalized x-component value, ObjectVisibilityDirectionX[ i ], of the direction vector for the visibility cone of an object with object index i. The value of soi\_direction\_x[ i ] shall be in the range of − 214 to 214, inclusive. The default value of soi\_direction\_x[ i ] is equal to 214.

The value of ObjectVisibilityDirectionX[ i ] is computed as follows:

ObjectVisibilityDirectionX[ i ] = soi\_direction\_x[ i ] ÷ 214

**soi\_direction\_y**[ i ] specifies the normalized y-component value, ObjectVisibilityDirectionY[ i ], of the direction vector for the visibility cone of an object with object index i. The value of soi\_direction\_y[ i ] shall be in the range of − 214 to 214, inclusive. The default value of soi\_direction\_y[ i ] is equal to 214.

The value of ObjectVisibilityDirectionY[ i ] is computed as follows:

ObjectVisibilityDirectionY[ i ] = soi\_direction\_y[ i ] ÷ 214

**soi\_direction\_z**[ i ] specifies the normalized z-component value, ObjectVisibilityDirectionZ[ i ], of the direction vector for the visibility cone of an object with object index i. The value of soi\_direction\_z[ i ] shall be in the range of −214 to 214, inclusive. The default value of soi\_direction\_z[ i ] is equal to 214.

The value of ObjectVisibilityDirectionZ[ i ] is computed as follows:

ObjectVisibilityDirectionZ[ i ] = soi\_direction\_z[ i ] ÷ 214

**soi\_angle**[ i ] indicates the angle, ObjectVisibilityAngle[ i ], of the visibility cone along the direction vector in degrees. The value of soi\_angle[ i ] shall be in the range of 0 to 216 − 1, inclusive. The default value of soi\_angle[ i ] is equal to 216 − 1.

The value of ObjectVisibilityAngle[ i ] is computed as follows:

ObjectVisibilityAngle[ i ] = ( soi\_angle[ i ] ÷ ( 216 – 1 ) ) \* 180

**soi\_3d\_bounding\_box\_update\_flag**[ i ] equal to 1 indicates that 3D bounding box information is present for an object with object index i. soi\_3d\_bounding\_box\_update\_flag[ i ] equal to 0 indicates that 3D bounding box information is not present.

**soi\_3d\_bounding\_box\_x**[ i ] specifies the quantized x coordinate value of the origin position of the 3D bounding box of an object with index i. The default value of soi\_3d\_bounding\_box\_x[ i ] is equal to 0.

**soi\_3d\_bounding\_box\_y**[ i ] specifies the quantized y coordinate value of the origin position of the 3D bounding box of an object with index i. The default value of soi\_3d\_bounding\_box\_y[ i ] is equal to 0.

**soi\_3d\_bounding\_box\_z**[ i ] specifies the quantized z coordinate value of the origin position of the 3D bounding box of an object with index i. The default value of soi\_3d\_bounding\_box\_z[ i ] is equal to 0.

**soi\_3d\_bounding\_box\_size\_x**[ i ]specifies the quantized size of the bounding box on the x axis of an object with index i. The default value of soi\_3d\_bounding\_box\_size\_x[ i ] is equal to 0.

**soi\_3d\_bounding\_box\_size\_y**[ i ]specifies the quantized size of the bounding box on the y axis of an object with index i. The default value of soi\_3d\_bounding\_box\_size\_y[ i ] is equal to 0.

**soi\_3d\_bounding\_box\_size\_z**[ i ]specifies the quantized size of the bounding box on the z axis of an object with index i. The default value of soi\_3d\_bounding\_box\_size\_z[ i ] is equal to 0.

The bounding box origin and size for an object with index i are computed as follows:

Object3DBoundingBoxX[ i ] = soi\_3d\_bounding\_box\_x[ i ] <<soi\_3d\_bounding\_box\_scale\_log2  
Object3DBoundingBoxY[ i ] = soi\_3d\_bounding\_box\_y[ i ] <<soi\_3d\_bounding\_box\_scale\_log2  
Object3DBoundingBoxZ[ i ] = soi\_3d\_bounding\_box\_z[ i ] <<soi\_3d\_bounding\_box\_scale\_log2  
Object3DBoundingBoxSizeX[ i ] = soi\_3d\_bounding\_box\_size\_x[ i ] <<soi\_3d\_bounding\_box\_scale\_log2  
Object3DBoundingBoxSizeY[ i ] = soi\_3d\_bounding\_box\_size\_y[ i ] <<soi\_3d\_bounding\_box\_scale\_log2  
Object3DBoundingBoxSizeZ[ i ] = soi\_3d\_bounding\_box\_size\_z[ i ] <<soi\_3d\_bounding\_box\_scale\_log2

**soi\_collision\_shape\_update\_flag**[ i ] equal to 1 indicates that collision shape update information is present for an object with object index i. soi\_collision\_shape\_update\_flag[ i ] equal to 0 indicates that collision shape update information is not present.

**soi\_collision\_shape\_id**[ i ] indicates the collision shape ID, ObjectCollisionShapeID[ i ], of an object with index i. The collision shape ID is identified through means outside this document. The default value of soi\_collision\_shape\_id[ i ] is equal to 0.

**soi\_point\_style\_update\_flag**[ i ] equal to 1 indicates that point style update information is present for an object with object index i. soi\_point\_style\_update\_flag[ i ] equal to 0 indicates that point style update information is not present.

**soi\_point\_shape\_id**[ i ] indicates the point shape ID, ObjectPointShapeID[ i ], of an object with index i. The default value of soi\_point\_shape\_id[ i ] is equal to 0. The value of soi\_point\_shape\_id[ i ] shall be in the range of 0 to 2, inclusive in bitstreams conforming to this version of this document. Other values of soi\_point\_shape\_id[ i ] are reserved for future use by ISO/IEC. Decoders conforming to this version of this document shall ignore reserved values of soi\_point\_shape\_id[ i ].

Table F-4 – soi\_point\_shape\_id[ i ] values

|  |  |
| --- | --- |
| **value** | **Description** |
| 0 | Circle |
| 1 | Square |
| 2 | Diamond |
| 3..255 | Reserved |

**soi\_point\_size**[ i ] indicates the point size, ObjectPointSize[ i ], of an object with index i. The default value of soi\_point\_size[ i ] is equal to 1.

**soi\_material\_id\_update\_flag**[ i ] equal to 1 indicates that material ID update information is present for an object with object index i. soi\_point\_style\_update\_flag[ i ] equal to 0 indicates that point style update information is not present.

**soi\_material\_id**[ i ] indicates the material ID, ObjectMaterialID[ i ] of an object with index i. The default value of soi\_material\_id[ i ] is equal to 0. The material ID is identified through means outside this document.

* + - * 1. Object initialization

Let the function ObjectInit( k ) be defined as follows:

ObjectInit( k ) {  
 ObjectTracked[ k ] = 0  
 ObjectLabelIndex[ k ] = 0  
 ObjectPriorityValue[ k ] = 0  
 ObjectHiddenFlag[ k ] = 0  
 ObjectNumDependencies[ k ] = 0  
 for ( j = 0; j < MaxObjectNumDependencies; j++ ) {  
 ObjectDependencyIndex[ k ][ j ] = 0  
 }  
 ObjectVisibilityDirectionX[ k ] = 1  
 ObjectVisibilityDirectionY[ k ] = 1  
 ObjectVisibilityDirectionZ[ k ] = 1  
 ObjectVisibilityAngle[ k ] = 180  
 Object3DBoundingBoxX[ k ] = 0  
 Object3DBoundingBoxY[ k ] = 0  
 Object3DBoundingBoxZ[ k ] = 0  
 Object3DBoundingBoxSizeX[ k ] = 0  
 Object3DBoundingBoxSizeY[ k ] = 0  
 Object3DBoundingBoxSizeZ[ k ] = 0  
 ObjectCollisionShapeID[ k ] = 0  
 ObjectPointShapeID[ k ] = 0  
 ObjectPointSize[ k ] = 0  
 ObjectMaterialID[ k ] = 0  
 }

* + - 1. Object label information SEI message semantics
         1. General

This SEI message defines a set of labels that could be associated with objects in a volumetric scene.

Let the variable MaxNumLabels be set equal to either 216 – 1 or a value determined by the application.

NOTE – An application may wish to limit the value of the variable MaxNumLabels so as to constrain the required memory.

At the start of each sequence,

* the variable LabelLanguage, and
* the 1D arrays LabelIndex, LabelSetFlag, and Label,

are initialized as follows:

LabelLanguage = ""  
 for ( i = 0; i < MaxNumLabels; i++ ) {  
 LabelIndex[ i ] = i  
 LabelInit( LabelIndex[ i ] )  
 }

where the function LabelInit( ) is defined in the subclause F.3.12.2.2.

**oli\_cancel\_flag** equal to 1 indicates that the object label information SEI message cancels the persistence of any previous object label information SEI message in output order and sets all variables and arrays associated with the object label SEI message, as follows:

LabelLanguage = ""  
 for ( i = 0; i < MaxNumLabels; i++ ) {  
 LabelIndex[ i ] = i  
 LabelInit( LabelIndex[ i ] )  
 }

**oli\_label\_language\_present\_flag** equal to 1 indicates that label language information is present in the object label information SEI message. oli\_label\_language\_present\_flag equal to 0 indicates that label language information is not present.

**oli\_bit\_equal\_to\_zero** shall be equal to 0.

**oli\_label\_language** contains a language tag, LabelLanguage, as specified by IETF RFC 5646 followed by a null termination byte equal to 0x00. The length of the oli\_label\_language syntax element shall be less than or equal to 255 bytes, not including the null termination byte.

**oli\_num\_label\_updates** indicates the number of labels that are to be updated by the current SEI message. The value of oli\_num\_label\_updates shall be in the range from 0 to MaxNumLabels, inclusive.

**oli\_label\_idx**[ i ] indicates the label index, LabelIndex[ i ], of the i-th label to be updated. The value of oli\_label\_idx[ i ] shall be in the range from 0 to MaxNumLabels – 1, inclusive.

**oli\_label\_cancel\_flag**[ i ] equal to 1 indicates that all label information, for label with index i, shall be reset by invoking the function LabelInit( i ). oli\_label\_cancel\_flag[ i ] equal to 0 indicates that the label with index i shall be updated with information that follows this element and that the variable LabelSetFlag[ i ] shall be set to 1.

**oli\_bit\_equal\_to\_zero** shall be equal to 0.

**oli\_label**[ i ] indicates the label, Label[ i ], of the i-th label. The length of the oli\_label[ i ] syntax element shall be less than or equal to 255 bytes, not including the null termination byte. The default value of oli\_label[ i ] is equal to "".

**oli\_persistence\_flag** specifies the persistence of the scene object information SEI message for the current layer. oli\_persistence\_flag equal to 0 specifies that the object label information SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. oli\_persistence\_flag equal to 1 specifies that the object label information SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing an object label information SEI message, with either an oli\_cancel\_flag equal to 1 or the same value of oli\_persistence\_flag, applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.
  + - * 1. Label initialization

Let the function LabelInit( k ) be defined as follows:

LabelInit( k ) {  
 LabelSetFlag[ k ] = 0  
 Label[ k ] = ""  
 }

* + - 1. Patch information SEI message semantics
         1. General

This SEI message defines a table that indicates the set of tiles and corresponding patches that may be present in the current frame. These patches can be assigned to one or more objects that may be present in a volumetric scene and which were previously defined through one or more scene object information SEI messages.

Let the variables MaxNumTiles, MaxNumPatches, and MaxNumObjectsPerPatch be set equal to either 1024, 216 – 1, and 216 – 1, respectively, or a value determined by the application in bitstreams conforming to this document.

NOTE – An application may wish to limit the value of the variable MaxNumTiles, MaxNumPatches and MaxNumObjectsPerPatch so as to constrain the required memory.

At the start of each sequence,

* the 2D array PatchNumObjects, of size MaxNumTiles × MaxNumPatches, and
* the 3D array PatchObjectIndex, of size MaxNumTiles × MaxNumPatches × MaxNumObjectsPerPatch,

are initialized as follows:

for ( i = 0; i < MaxNumTiles; i++ ) {   
 TileInit( i )  
 }

where the function TileInit( ) is defined in the subclause F.3.12.3.2.

**pi\_persistence\_flag** specifies the persistence of the patch information SEI message for the current layer. pi\_persistence\_flag equal to 0 specifies that the patch information SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. pi\_persistence\_flag equal to 1 specifies that the patch information SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing a patch information SEI message with the same value of pi\_persistence\_flag and applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**pi\_reset\_flag** equal to 1 indicates that all entries in the patch information table shall be removed as follows:

for ( i = 0; i < MaxNumTiles; i++ ) {   
 TileInit( i )  
 }

**pi\_num\_tile\_updates** indicates the number of tiles that are to be updated in the patch information table by the current SEI message. The value of pi\_num\_tile\_updates shall be in the range from 0 to MaxNumTiles, inclusive.

**pi\_log2\_max\_object\_idx\_tracked\_minus1** plus 1 specifies the number of bitsused to signal the value of a tracked object index in the current patch information SEI message.

**pi\_log2\_max\_patch\_idx\_updated\_minus1** plus 1 specifies the number of bitsused to signal the value of an updated patch index in the current patch information SEI message.

**pi\_tile\_id**[ i ] specifies the tile ID for the i-th updated tile in the current SEI message.

**pi\_tile\_cancel\_flag**[ i ] equal to 1 indicates that the tile with index i shall be reset by invoking the function TileInit( i ). pi\_tile\_cancel\_flag[ i ] equal to 0 indicates that all patches previously assigned to the tile with index i will be retained.

**pi\_num\_patch\_updates**[ i ] indicates the number of patches that are to be updated by the current SEI message within the tile with index i in the patch information table. The value of pi\_num\_patch\_updates[ i ] shall be in the range from 0 to MaxNumPatches, inclusive.

**pi\_patch\_idx**[ tileID ][ i ] indicates the patch index of the i-th patch in tile with tile ID equal to tileID that is to be updated in the patch information table. The number of bits used to represent pi\_patch\_idx[ tileID ][ i ] is equal to pi\_log2\_max\_patch\_idx\_updated\_minus1 + 1.

**pi\_patch\_cancel\_flag**[ tileID ][ i ] equal to 1 indicates that the information related to the patch with index i in tile with tile ID equal to tileID shall be reset from the patch information table by invoking the function PatchInit( tileID, i ).

**pi\_patch\_number\_of\_objects\_minus1**[ tileID ][ i ] plus 1 indicates the number of objects, PatchNumObjects[ tileID ][ i ], that are to be associated with the patch with index i in tile with tile ID equal to tileID. The value of pi\_patch\_number\_of\_objects\_minus1[ tileID ][ i ] shall be in the range from 0 to MaxNumObjectsPerPatch, inclusive.

**pi\_patch\_object\_idx**[ tileID ][ i ][ k ]indicates the k-th object index, PatchObjectIndex[ tileID ][ i ][ k ], that is associated with the i-th patch in tile with tile ID equal to tileID. The number of bits used to represent pi\_patch\_object\_idx[ tileID ][ i ][ k ] is equal to pi\_log2\_max\_object\_idx\_tracked\_minus1 + 1.

* + - * 1. Tile and Patch Initialization

Let the function PatchInit( i, j ) be defined as follows:

PatchInit( i, j ) {  
 PatchNumObjects[ i ][ j ] = 0  
 for ( k = 0; k < MaxNumObjectsPerPatch; k++ ) {  
 PatchObjectIndex[ i ][ j ][ k ] = 0  
 }  
 }

Then, also the function TileInit( i ) be defined as follows:

TileInit( i ) {  
 for ( j = 0; j < MaxNumPatches; j++ ) {   
 PatchInit( i, j )  
 }  
 }

* + - 1. Volumetric rectangle information SEI message semantics
         1. General

This SEI message defines a table that indicates a set of volumetric rectangles within an atlas. These volumetric rectangles can be assigned to one or more objects that may be present in a volumetric scene and that were previously defined through one or more scene object information SEI messages.

Let the variables MaxNumRectangles and MaxNumObjectsPerRectangle be set equal to either 216 – 1 and 216 – 1, respectively, or a value determined by the application in bitstreams conforming to this document.

NOTE – An application may wish to limit the value of the variable MaxNumRectangles and MaxNumObjectsPerRectangle so as to constrain the required memory.

At the start of each sequence,

* the 1D arrays VriBoundingBoxTop, VriBoundingBoxLeft, VriBoundingBoxWidth, VriBoundingBoxHeight, and VriNumberOfObjects, of size MaxNumRectangles,
* the 2D array VriObjectIndex, of size MaxNumRectangles × MaxNumObjectsPerRectangle,

are initialized as follows:

for ( i = 0; i < MaxNumRectangles; i++ ) {   
 RectangleInit( i )  
 }

where the function RectangleInit( ) is defined in the subclause F.3.12.4.2.

**vri\_persistence\_flag** specifies the persistence of the volumetric rectangle information SEI message for the current layer. vri\_persistence\_flag equal to 0 specifies that the volumetric rectangle information SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. vri\_persistence\_flag equal to 1 specifies that the volumetric rectangle information SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing a volumetric rectangle information SEI message with the same value of vri\_persistence\_flag and applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**vri\_reset\_flag** equal to 1 indicates that all entries in the volumetric rectangle information table of the current atlas with atlas ID SeiAtlasID shall be initialized as follows:

for ( i = 0; i < MaxNumRectangles; i++ ) {   
 RectangleInit( i )  
 }

**vri\_num\_rectangles\_updates** indicates the number of volumetric rectangles that are to be updated by the current SEI message. The value of vri\_num\_rectangles\_updates shall be in the range from 0 to MaxNumRectangles , inclusive.

**vri\_log2\_max\_object\_idx\_tracked\_minus1** plus 1specifies the number of bits used to signal the value of a tracked object index in the current volumetric rectangle information SEI message.

**vri\_log2\_max\_rectangle\_idx\_updated\_minus1** plus 1specifies the number of bits used to signal the value of an updated volumetric rectangle index in the current volumetric rectangle information SEI message.

**vri\_rectangle\_idx**[ i ]indicates the i-th volumetric rectangle index that is to be updated in the volumetric rectangle information table. The number of bits used to represent vri\_rectangle\_idx[ i ] is equal to ( vri\_log2\_max\_rectangle\_idx\_updated\_minus1 + 1 ).

**vri\_rectangle\_cancel\_flag**[ i ]equal to 1 indicates that the volumetric rectangle with index i shall be removed from the volumetric rectangle information table by invoking the function RectangleInit( i ).

**vri\_bounding\_box\_update\_flag**[ i ] equal to 1 indicates that 2D bounding box information for the volumetric rectangle with index i should be updated. vri\_bounding\_box\_update\_flag[ i ] equal to 0 indicates that 2D bounding box information for the volumetric rectangle with index i should not be updated.

**vri\_bounding\_box\_top**[ i ] indicates the vertical coordinate value, VriBoundingBoxTop[ i ], of the top-left position of the bounding box of the i-th volumetric rectangle within the current atlas frame. The default value of vri\_bounding\_box\_top[ i ] is equal to 0.

**vri\_bounding\_box\_left**[ i ]indicates the horizonal coordinate value, VriBoundingBoxLeft[ i ], of the top-left position of the bounding box of the i-th volumetric rectangle within the current atlas frame. The default value of vri\_bounding\_box\_left[ i ] is equal to 0.

**vri\_bounding\_box\_width**[ i ]indicates the width of the bounding box, VriBoundingBoxWidth[ i ], of the i-th volumetric rectangle. The default value of vri\_bounding\_box\_width[ i ] is equal to 0.

**vri\_bounding\_box\_height**[ i ]indicates the height of the bounding box, VriBoundingBoxHeight[ i ], of the i-th volumetric rectangle. The default value of vri\_bounding\_box\_height[ i ] is equal to 0.

**vri\_rectangle\_number\_of\_objects\_minus1**[ i ]plus 1 indicates the number of objects, VriNumberOfObjects[ i ], that are to be associated with the i-th volumetric rectangle. The value of vri\_rectangle\_number\_of\_objects\_minus1[ i ] shall be in the range from 0 to MaxNumObjectsPerRectangle – 1, inclusive.

**vri\_rectangle\_object\_idx**[ i ][ j ] indicates the j-th object index that is associated with the i-th volumetric rectangle, VriObjectIndex[ i ][ j ]. The number of bits used to represent vri\_rectangle\_object\_idx[ i ] is equal to vri\_log2\_max\_object\_idx\_tracked\_minus1 + 1.

* + - * 1. Rectangle initialization

Let the function RectangleInit( i ) be defined as follows:

RectangleInit( i ) {  
 VriBoundingBoxTop[ i ] = 0  
 VriBoundingBoxLeft[ i ] = 0  
 VriBoundingBoxWidth[ i ] = 0  
 VriBoundingBoxHeight[ i ] = 0  
 VriNumberOfObjects[ i ] = 0  
 for ( j = 0; j < MaxNumObjectsPerRectangle; j++ ) {  
 VriObjectIndex[ i ][ j ] = 0  
 }  
 }

* + - 1. Atlas object association SEI message semantics

This SEI message defines a mask that indicates the mapping between objects and atlases. Objects can be contained in one or more atlases that describe a volumetric scene. Objects are defined through scene object information SEI messages.

Let the variable MaxNumObjects be set equal to 216 – 1 or a value determined by the application in bitstreams conforming to this document.

NOTE – An application may wish to limit the value of the variable MaxNumObjects so as to constrain the required memory.

At the start of each sequence,

* the 2D array ObjectInAtlas, of size MaxNumObjects × 64,

are initialized as follows:

for ( i = 0; i < MaxNumObjects; i++ ) {   
 for ( j = 0; j < 64; i++ ) {   
 ObjectInAtlas[ i ][ j ] = 0  
 }  
 }

**aoa\_persistence\_flag** specifies the persistence of the atlas object association SEI message for the current layer. aoa\_persistence\_flag equal to 0 specifies that the atlas object association SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. aoa\_persistence\_flag equal to 1 specifies that the atlas object association SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing an atlas object association SEI message with the same value of aoa\_persistence\_flag and applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**aoa\_reset\_flag** equal to 1 indicates that all entries in the atlas object association table shall be removed as follows:

for ( i = 0; i < MaxNumObjects; i++ ) {   
 for ( j = 0; j < 64; i++ ) {   
 ObjectInAtlas[ i ][ j ] = 0  
 }  
 }

**aoa\_num\_atlases\_minus1** plus 1 indicates the number of atlases present in the CVS. aoa\_num\_atlases\_minus1 shall be equal to vps\_atlas\_count\_minus1.

**aoa\_num\_updates** indicates the number of objects that are to be updated in the object to atlas table by the current SEI message.

**aoa\_log2\_max\_object\_idx\_tracked\_minus1** plus 1 specifies the number of bits used to signal the value of an object index in the current atlas object association SEI message.

**aoa\_atlas\_id**[ j ]specifies the atlas ID of the j-th atlas**.**

**aoa\_object\_idx**[ i ] indicates the i-th object index. The number of bits used to represent aoa\_object\_idx[ i ] is equal to aoa\_log2\_max\_object\_idx\_tracked\_minus1 + 1.

**aoa\_object\_in\_atlas**[ i ][ j ] indicates the value of ObjectInAtlas[ i ][ j ] for atlas ID equal to j and object index equal to i. aoa\_object\_in\_atlas[ i ][ j ] equal to 1 indicates that the object with index i is present in atlas with atlas ID equal to j. aoa\_object\_in\_atlas[ i ][ j ] equal to 0 indicates that the object with index i is not present in atlas with atlas ID equal to j.

* + 1. Buffering period SEI message semantics

A buffering period SEI message provides initial CAB removal delay and initial CAB removal delay offset information for initialization of the HRD at the position of the associated coded atlas access unit in decoding order.

The following applies for the buffering period SEI message syntax and semantics:

* The variables CabSize[ !NalHrdModeFlag ][ Htid ][ i ], BitRate[ !NalHrdModeFlag ][ Htid ][ i ] and CabCnt are derived from syntax elements found in the hrd\_sub\_layer\_parameters( !NalHrdModeFlag, Htid ) syntax structure that is applicable to at least one of the operation points to which the buffering period SEI message applies.
* Any two operation points that the buffering period SEI message applies to having different OpTid values tIdA and tIdB indicate that the values of bp\_hrd\_cab\_cnt\_minus1[ tIdA ] and bp\_hrd\_cab\_cnt\_minus1[ tIdB ] applicable to the two operation points are identical.
* The bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with any of the operation points to which the buffering period SEI message applies.

The presence of buffering period SEI messages for an operation point is specified as follows:

* If NalHrdBpPresentFlag is equal to 1 or AclHrdBpPresentFlag is equal to 1, the following applies for each access unit in the CAS:
* If the coded atlas access unit is an IRAP coded atlas access unit, a buffering period SEI message applicable to the operation point shall be associated with the access unit.
* Otherwise, if both of the following conditions apply, a buffering period SEI message applicable to the operation point may or may not be present for the access unit:
* The atlas frame has TemporalID equal to 0.
* The atlas frame is not a RASL, RADL or SLNR atlas frame.
* Otherwise, the coded atlas access unit shall not be associated with a buffering period SEI message applicable to the operation point.
* Otherwise (NalHrdBpPresentFlag and AclHrdBpPresentFlag are both equal to 0), no coded atlas access unit in the CAS shall be associated with a buffering period SEI message applicable to the operation point.

NOTE 1 – For some applications, frequent presence of buffering period SEI messages can be desirable (e.g., for random access at an IRAP coded atlas frame or a non-IRAP coded atlas frame or for bitstream splicing).

**bp\_nal\_hrd\_params\_present\_flag** equal to 1 specifies that a list of syntax element pairs bp\_nal\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_nal\_initial\_cab\_removal\_offset[ i ][ j ] are present in the buffering period SEI message. bp\_nal\_hrd\_params\_present\_flag equal to 0 specifies that no syntax element pairs bp\_nal\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_nal\_initial\_cab\_removal\_offset[ i ][ j ] are present in the buffering period SEI message.

It is a requirement of bitstream conformance that the value of bp\_nal\_hrd\_params\_present\_flag shall be equal to hrd\_nal\_parameters\_present\_flag.

The variable NalHrdBpPresentFlag is derived as follows:

* If one or more of the following conditions are true, then the value of NalHrdBpPresentFlag is set equal to 1:
* bp\_nal\_hrd\_params\_present\_flag is present in the bitstream and is equal to 1.
* The need for presence of buffering periods for NAL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this document.
* Otherwise, the value of NalHrdBpPresentFlag is set equal to 0.

**bp\_acl\_hrd\_params\_present\_flag** equal to 1 specifies that a list of syntax element pairs bp\_acl\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_acl\_initial\_cab\_removal\_offset[ i ][ j ] are present in the buffering period SEI message. bp\_acl\_hrd\_params\_present\_flag equal to 0 specifies that no syntax element pairs bp\_acl\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_acl\_initial\_cab\_removal\_offset[ i ][ j ] are present in the buffering period SEI message.

It is a requirement of bitstream conformance that the value of bp\_acl\_hrd\_params\_present\_flag shall be equal to hrd\_acl\_parameters\_present\_flag.

It is a requirement of bitstream conformance that the values of bp\_acl\_hrd\_params\_present\_flag and bp\_nal\_hrd\_params\_present\_flag in a buffering period SEI message shall not be both equal to 0.

The variable AclHrdBpPresentFlag is derived as follows:

* If one or more of the following conditions are true, then the value of AclHrdBpPresentFlag is set equal to 1:
* bp\_acl\_hrd\_params\_present\_flag is present in the bitstream and is equal to 1.
* The need for presence of buffering periods for ACL HRD operation to be present in the bitstream in buffering period SEI messages is determined by the application, by some means not specified in this document.
* Otherwise, the value of AclHrdBpPresentFlag is set equal to 0.

The variable CabDabDelaysPresentFlag is derived as follows:

* If one or more of the following conditions are true, then the value of CabDabDelaysPresentFlag is set equal to 1:
* bp\_nal\_hrd\_params\_present\_flag is present in the bitstream and is equal to 1.
* bp\_acl\_hrd\_params\_present\_flag is present in the bitstream and is equal to 1.
* The need for presence of CAB and DAB output delays to be present in the bitstream in atlas timing SEI messages is determined by the application, by some means not specified in this document.
* Otherwise, the value of CabDabDelaysPresentFlag is set equal to 0.

**bp\_initial\_cab\_removal\_delay\_length\_minus1** plus 1 specifies the length, in bits, of the j-th default and alternative initial CAB removal delays and offsets, respectively, for the i-th temporal sub-layer.of the buffering period SEI messages. When not present, the value of bp\_initial\_cab\_removal\_delay\_length\_minus1 is inferred to be equal to 23.

**bp\_au\_cab\_removal\_delay\_length\_minus1** plus 1 specifies the length, in bits, of the syntax elements bp\_cab\_delay\_offset in the buffering period SEI message and the syntax elements aft\_au\_cab\_removal\_delay\_minus1[ i ] in the atlas frame timing SEI messages in the current buffering period. When not present, the value of bp\_au\_cab\_removal\_delay\_length\_minus1 is inferred to be equal to 23.

**bp\_dab\_output\_delay\_length\_minus1** plus 1 specifies the length, in bits, of the syntax elements bp\_dab\_delay\_offset syntax element in the buffering period SEI message and aft\_dab\_output\_delay[ i ] in the atlas frame timing SEI message. When not present, the value of bp\_dab\_output\_delay\_length\_minus1 is inferred to be equal to 23.

**bp\_irap\_cab\_params\_present\_flag** equal to 1 specifies the presence of the bp\_nal\_initial\_alt\_cab\_removal\_delay[ Htid ][ i ], bp\_nal\_initial\_alt\_cab\_removal\_delay[ Htid ][ i ], bp\_acl\_initial\_alt\_cab\_removal\_offset[ Htid ][ i ] and bp\_acl\_initial\_alt\_cab\_removal\_offset[ Htid ][ i ] syntax elements. When not present, the value of bp\_irap\_cab\_params\_present\_flag is inferred to be equal to 0. When the associated atlas frame is neither a CRA atlas frame nor a BLA atlas fame, the value of bp\_irap\_cab\_params\_present\_flag shall be equal to 0.

**bp\_cab\_delay\_offset** specifies an offset to be used in the derivation of the nominal CAB removal times of access units following, in decoding order, the CRA or BLA coded atlas frame associated with the buffering period SEI message when the RASL coded atlas access units associated with the CRA or BLA access unit are not present. The syntax element has a length in bits given by bp\_au\_cab\_removal\_delay\_length\_minus1 + 1. When not present, the value of bp\_cab\_delay\_offset is inferred to be equal to 0.

**bp\_dab\_delay\_offset** specifies an offset to be used in the derivation of the DAB output times of the CRA or BLA coded atlas access unit associated with the buffering period SEI message when the RASL coded atlas access units associated with the CRA or BLA coded atlas access unit are not present. The syntax element has a length in bits given by bp\_dab\_output\_delay\_length\_minus1 + 1. When not present, the value of bp\_dab\_delay\_offset is inferred to be equal to 0.

When the current atlas frame is not the first atlas frame in the bitstream in decoding order, let PrevNonDiscardableAtlasFrame be the preceding atlas in decoding order with TemporalID equal to 0 that is not a RASL, RADL or SLNR atlas frame.

**bp\_concatenation\_flag** indicates, when the current atlas is not the first atlas in the bitstream in decoding order, whether the nominal CAB removal time of the current atlas is determined relative to the nominal CAB removal time of the preceding atlas with a buffering period SEI message or relative to the nominal CAB removal time of the atlas frame PrevNonDiscardableAtlasFrame.

**bp\_atlas\_cab\_removal\_delay\_delta\_minus1** plus 1, when the current atlas is not the first atlas frame in the bitstream in decoding order, specifies a CAB removal delay increment value relative to the nominal CAB removal time of the atlas frame PrevNonDiscardableAtlasFrame. This syntax element has a length in bits given by bp\_atlas\_cab\_removal\_delay\_length\_minus1 + 1.

When the current atlas contains a buffering period SEI message and bp\_concatenation\_flag is equal to 0 and the current atlas is not the first atlas in the bitstream in decoding order, it is a requirement of bitstream conformance that the following constraint applies:

* If the atlas frame PrevNonDiscardableAtlasFrame is not associated with a buffering period SEI message, the aft\_au\_cab\_removal\_delay\_minus1[ i ] of the current atlas shall be equal to the aft\_au\_cab\_removal\_delay\_minus1[ i ] of PrevNonDiscardableAtlasFrame plus bp\_atlas\_cab\_removal\_delay\_delta\_minus1+ 1.
* Otherwise, aft\_au\_cab\_removal\_delay\_minus1[ i ] shall be equal to bp\_atlas\_cab\_removal\_delay\_delta\_minus1.

NOTE 2 – When the current atlas contains a buffering period SEI message and bp\_concatenation\_flag is equal to 1, the aft\_au\_cab\_removal\_delay\_minus1[ i ] for the current atlas frame is not used. The above-specified constraint can, under some circumstances, make it possible to splice bitstreams (that use suitably-designed referencing structures) by simply changing the value of bp\_concatenation\_flag from 0 to 1 in the buffering period SEI message for an IRAP coded atlas frame at the splicing point. When bp\_concatenation\_flag is equal to 0, the above-specified constraint enables the decoder to check whether the constraint is satisfied as a way to detect the loss of the atlas frame PrevNonDiscardableAtlasFrame.

**bp\_max\_sub\_layers\_minus1** plus 1 specifies the maximum number of temporal sub-layers. It is a condition of bitstream conformance that for this version of this document the value of bp\_max\_sub\_layers\_minus1 shall be equal to 0.

**bp\_hrd\_cab\_cnt\_minus1**[ i ] plus 1 specifies the number of alternative CAB specifications in the bitstream of the CAS when Htid is equal to i. The value of bp\_hrd\_cab\_cnt\_minus1[ i ] shall be in the range of 0 to 31, inclusive. When not present, the value of bp\_hrd\_cab\_cnt\_minus1[ i ] is inferred to be equal to 0.

It is a requirement of bitstream conformance that the value of bp\_hrd\_cab\_cnt\_minus1[ i ] shall be the same as the value of hrd\_cab\_cnt\_minus1[ i ].

**bp\_nal\_initial\_cab\_removal\_delay**[ i ][ j ] and **bp\_nal\_initial\_alt\_cab\_removal\_delay**[ i ][ j ] specify respectively the j-th default and alternative initial CAB removal delay for the NAL HRD in units of a 90 kHz clock of the i-th temporal sub-layer. The length of bp\_nal\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_nal\_initial\_alt\_cab\_removal\_delay[ i ][ j ] is bp\_cab\_initial\_removal\_delay\_length\_minus1 + 1 bits. The value of bp\_nal\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_nal\_initial\_alt\_cab\_removal\_delay[ i ][ j ] shall not be equal to 0 and shall be less than or equal to 90000 \* ( CabSize[ !NalHrdModeFlag ][ i ][ j ] ÷ BitRate[ !NalHrdModeFlag ][ i ][ j ] ), the time-equivalent of the CAB size in 90 kHz clock units. When not present, the values of bp\_nal\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_nal\_initial\_alt\_cab\_removal\_delay[ i ][ j ] are inferred to be equal to 90000 \* ( CabSize[ !NalHrdModeFlag ][ i ][ j ] ÷ BitRate[ !NalHrdModeFlag ][ i ][ j ] ).

**bp\_nal\_initial\_cab\_removal\_offset**[ i ][ j ] and **bp\_nal\_initial\_alt\_cab\_removal\_offset**[ i ][ j ] specify the default and the alternative initial CAB removal offsets, respectively, for the i-th CAB when the NAL HRD parameters are in use. The syntax elements have a length in bits given by hrd\_initial\_cab\_removal\_delay\_length\_minus1 + 1 and are in units of a 90 kHz clock.

**bp\_acl\_initial\_cab\_removal\_delay**[ i ][ j ] and **bp\_acl\_initial\_alt\_cab\_removal\_delay**[ i ][ j ] specify respectively the j-th default and alternative initial CAB removal delay of the i-th temporal sub-layer for the ACL HRD in units of a 90 kHz clock. The length of bp\_acl\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_acl\_initial\_alt\_cab\_removal\_delay[ i ][ j ] is bp\_cab\_initial\_removal\_delay\_length\_minus1 + 1 bits. The value of bp\_acl\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_acl\_initial\_alt\_cab\_removal\_delay[ i ][ j ] shall not be equal to 0 and shall be less than or equal to 90000 \* ( CabSize[ !NalHrdModeFlag ][ i ][ j ] ÷ BitRate[ !NalHrdModeFlag ][ i ][ j ] ), the time-equivalent of the CAB size in 90 kHz clock units. When not present, the values of bp\_acl\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_acl\_initial\_alt\_cab\_removal\_delay[ i ][ j ] are inferred to be equal to 90000 \* ( CabSize[ !NalHrdModeFlag ][ i ][ j ] ÷ BitRate[ !NalHrdModeFlag ][ i ][ j ] ).

**bp\_nal\_initial\_cab\_removal\_offset**[ i ][ j ] and **bp\_nal\_initial\_alt\_cab\_removal\_offset**[ i ][ j ] specify the j-th default and alternative initial CAB removal offset of the i-th temporal sub-layer for the NAL HRD in units of a 90 kHz clock. The length of bp\_nal\_initial\_cab\_removal\_offset[ i ][ j ] and bp\_nal\_initial\_alt\_cab\_removal\_offset[ i ][ j ] is bp\_cab\_initial\_removal\_delay\_length\_minus1 + 1 bits. When not present, the values of bp\_nal\_initial\_cab\_removal\_offset[ i ][ j ] and bp\_nal\_initial\_alt\_cab\_removal\_offset[ i ][ j ] are inferred to be equal to 0.

Over the entire CAS, for each value pair of i and j, the sum of bp\_nal\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_nal\_initial\_cab\_removal\_offset[ i ][ j ] shall be constant, and the sum of bp\_nal\_initial\_alt\_cab\_removal\_delay[ i ][ j ] and bp\_nal\_initial\_alt\_cab\_removal\_offset[ i ][ j ] shall be constant.

**bp\_acl\_initial\_cab\_removal\_offset**[ i ][ j ] and **bp\_acl\_initial\_alt\_cab\_removal\_offset**[ i ][ j ] specify the j-th default and alternative initial CAB removal offset of the i-th temporal sub-layer for the ACL HRD in units of a 90 kHz clock. The length of bp\_acl\_initial\_cab\_removal\_offset[ i ] and bp\_acl\_initial\_alt\_cab\_removal\_offset[ i ][ j ] is bp\_cab\_initial\_removal\_delay\_length\_minus1 + 1 bits. When not present, the values of bp\_acl\_initial\_cab\_removal\_offset[ i ][ j ] and bp\_acl\_initial\_alt\_cab\_removal\_offset[ i ][ j ] are inferred to be equal to 0.

Over the entire CAS, for each value pair of i and j the sum of bp\_acl\_initial\_cab\_removal\_delay[ i ][ j ] and bp\_acl\_initial\_cab\_removal\_offset[ i ][ j ] shall be constant, and the sum of bp\_acl\_initial\_alt\_cab\_removal\_delay[ i ][ j ] and bp\_acl\_initial\_alt\_cab\_removal\_offset[ i ][ j ] shall be constant.

Encoders should not include irap\_cab\_params\_present\_flag equal to 1 in buffering period SEI messages associated with a CRA or BLA atlas for which at least one of its associated RASL atlases follows one or more of its associated RADL atlas in decoding order.

* + 1. Atlas frame timing SEI message semantics

The atlas frame timing SEI message provides CAB removal delay and DAB output delay information for the coded atlas access unit associated with the SEI message.

The following applies for the atlas timing SEI message syntax and semantics:

* The variable CabDabDelaysPresentFlag is derived from syntax elements found in the buffering period SEI message that are applicable to at least one of the operation points to which the atlas frame timing SEI message applies.
* The bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with any of the operation points to which the atlas timing SEI message applies.

NOTE 1 – The syntax of the atlas frame timing SEI message is dependent on the content of the hrd\_parameters( ) syntax structures applicable to the operation points to which the atlas frame timing SEI message applies. These hrd\_parameters( ) syntax structures are in the ASPS that are active for the coded atlas access unit associated with the atlas frame timing SEI message. When the atlas frame timing SEI message is associated with an IRAP coded atlas access unit with NoOutputBeforeRecoveryFlag equal to 1, unless it is preceded by a buffering period SEI message within the same access unit, the activation of the ASPS (and, for IRAP coded atlas frames with NoOutputBeforeRecoveryFlag equal to 1 that are not the first atlas in the bitstream in decoding order, the determination that the coded atlas is an IRAP NoOutputBeforeRecoveryFlag equal to 1) does not occur until the decoding of the first coded tile NAL unit of the coded atlas. Since the coded tile NAL unit of the coded atlas follows the atlas frame timing SEI message in NAL unit order, there can be cases in which it is necessary for a decoder to store the RBSP containing the atlas frame timing SEI message until determining the activeASPS for the coded atlas, and then perform the parsing of the atlas frame timing SEI message.

**aft\_au\_cab\_removal\_delay\_minus1**[ i ] plus 1 specifies, for the i-th sub-layer, the number clock ticks between the nominal CAB removal time of the coded atlas access unit associated with the atlas timing SEI message and the preceding coded atlas access unit in decoding order that contained a buffering period SEI message. This value is also used to calculate an earliest possible time of arrival of access unit atlas data into the CAB for the HSS. The syntax element is a fixed length code whose length in bits is given by bp\_au\_cab\_removal\_delay\_length\_minus1 + 1.

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

* If the current atlas frame is associated with a buffering period SEI message that is applicable to at least one of the operation points to which the atlas frame timing SEI message applies, AuCabRemovalDelayMsb is set equal to 0.
* Otherwise, the following applies:
* Let maxCabRemovalDelay be equal to 2bp\_au\_cab\_removal\_delay\_length\_minus1 + 1.
* Let prevAuCabRemovalDelayMinus1 and prevAuCabRemovalDelayMsb be set equal to aft\_au\_cab\_removal\_delay\_minus1[ i ] and AuCabRemovalDelayMsb, respectively, of the previous atlas frame in decoding order that has TemporalID equal to 0, that is not a RASL, RADL or SLNR atlas frame, and that is within the same buffering period as the current atlas frame.
* AuCabRemovalDelayMsb is derived as follows:

if( aft\_au\_cab\_removal\_delay\_minus1[ i ] <= prevAuCabRemovalDelayMinus1 )  
 AuCabRemovalDelayMsb = prevAuCabRemovalDelayMsb + maxCabRemovalDelay (F‑1)  
else  
 AuCabRemovalDelayMsb = prevAuCabRemovalDelayMsb

The variable AuCabRemovalDelayVal is derived as follows:

AuCabRemovalDelayVal = AuCabRemovalDelayMsb +  
 aft\_au\_cab\_removal\_delay\_minus1[ i ] + 1 (F‑2)

The value of AuCabRemovalDelayVal shall be in the range of 1 to 232, inclusive. Within one buffering period, the AuCabRemovalDelayVal values for any two access units shall not be the same.

**aft\_dab\_output\_delay**[ i ] is used to compute the DAB output time of the atlas frame, for the i-th sub-layer. It specifies how many clock ticks to wait after removal of the access unit from the CAB before the decoded atlas is output from the DAB.

NOTE 2 – An atlas frame is not removed from the DAB at its output time when it is still marked as "used for short-term reference" or "used for long-term reference".

The length of the syntax element aft\_dab\_output\_delay[ i ] is given in bits by bp\_dab\_output\_delay\_length\_minus1 + 1. When asps\_max\_dec\_atlas\_frame\_buffering\_minus1 associated with minTid, is equal to 0, where minTid is the minimum of the OpTid values of all operation points the atlas frame timing SEI message applies to, aft\_dab\_output\_delay[ i ] shall be equal to 0.

The output time derived from the aft\_dab\_output\_delay[ i ] of any atlas frame that is output from an output timing conforming decoder shall precede the output time derived from the aft\_dab\_output\_delay[ i ] of all atlas frames in any subsequent CAS in decoding order.

The atlas output order established by the values of this syntax element shall be the same order as established by the values of AtlasFrmOrderCntVal.

For atlas frames that are not output by the "bumping" process because they precede, in decoding order, an IRAP coded atlas frame with NoOutputBeforeRecoveryFlag equal to 1, the output times derived from aft\_dab\_output\_delay[ i ] shall be increasing with increasing value of AtlasFrmOrderCntVal relative to all atlas frames within the same CAS.

* + 1. Viewport SEI messages family semantics
       1. Viewport camera parameters SEI message semantics

This SEI message defines viewport camera parameters that can be assigned to a viewport signalled by the viewport position SEI message.

The persistence scope for this SEI message is the remainder of the bitstream or until a new viewport camera parameters SEI message is encountered. Only the corresponding parameters specified in the SEI message shall be updated. Previously defined parameters from an earlier SEI message shall persist if not modified and if the value of vcp\_cancel\_flag is not equal to 1.

**vcp\_camera\_id** containsan identifying number that may be used to identify viewport camera parameters.

**vcp\_cancel\_flag** equal to 1 indicates that the SEI message cancels the persistence of any previous viewport camera parameters SEI message in output order with the same value of vcp\_camera\_id. vcp\_cancel\_flag equal to 0 indicates that viewport camera parameters information follows.

**vcp\_persistence\_flag** specifies the persistence of the viewport camera parameters SEI message for the current layer. vcp\_persistence\_flag equal to 0 specifies that the viewport camera parameters SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. vcp\_persistence\_flag equal to 1 specifies that the viewport camera parameters SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing a viewport camera parameters SEI message, with either an vcp\_cancel\_flag equal to 1 or the same value of vcp\_persistence\_flag, applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**vcp\_camera\_type** indicates the projection method of the viewport camera. vcp\_camera\_type equal to 0 specifies equirectangular projection. vcp\_camera\_type equal to 1 specifies a perspective projection. vcp\_camera\_type equal to 2 specifies an orthographic projection. vcp\_camera\_type values in tbe range of 3 to 7, inclusive, are reserved for future use by ISO/IEC.

**vcp\_erp\_horizontal\_fov** indicates the horizontal size of the viewport region, in units of 2−16  degrees. The value of vcp\_erp\_horizontal\_fov shall be in the range of 0 to 360 \* 216 − 1, inclusive.

**vcp\_erp\_vertical\_fov** indicates the vertical size of the viewport region, in units of 2−16  degrees. The value of vcp\_erp\_vertical\_fov shall be in the range of 0 to 180 \* 216 − 1, inclusive.

**vcp\_perspective\_aspect\_ratio** indicates the relative aspect ratio of the viewport for perspective projection (horizontal/vertical).

**vcp\_perspective\_horizontal\_fov** indicates the horizontal field of view for perspective projection in units of 2−16  degrees. The value of vcp\_perspective\_horizontal\_fov shall be in the range of 0 to 180 \* 216 − 1, inclusive.

**vcp\_ortho\_aspect\_ratio** indicates the relative aspect ratio of the viewport for orthogonal projection (horizontal/vertical).

**vcp\_ortho\_horizontal\_size** indicates the horizontal size of an orthogonal viewport in metres.

**vcp\_clipping\_near\_plane** indicates the distance of the near clipping plane for a given viewport in metres.

**vcp\_clipping\_far\_plane** indicates the distance of the far clipping plane for a given viewport in metres.

* + - 1. Viewport position SEI message semantics

This SEI message defines a viewport position and orientation. The viewport could then be associated with different types of information such as viewport camera parameters indicated by a viewport camera parameters SEI message.

The persistence scope for this SEI message is the remainder of the bitstream or until a new viewport position SEI message is encountered. Only the corresponding parameters specified in the SEI message shall be updated. Previously defined parameters from an earlier SEI message shall persist if not modified and if the value of vp\_cancel\_flag is not equal to 1.

**vp\_viewport\_id** specifiesan identifying number that may be used to identify a viewport.

**vp\_camera\_parameters\_present\_flag** equal to 1 indicates that vp\_vcp\_camera\_id is present in the viewport position SEI message. vp\_camera\_parameters\_present\_flag equal to 0 indicates that vp\_vcp\_camera\_id is not present in viewport position SEI message

**vp\_vcp\_camera\_id** specifies the value of vcp\_camera\_id of viewport parameter SEI message that should provide the camera information for this viewport. When vp\_vcp\_camera\_id is not present, it shall be inferred to be equal to 0.

**vp\_cancel\_flag** equal to 1 indicates that the SEI message cancels the persistence of any previous viewport position SEI message in output order with the same value of vp\_view\_id. vp\_cancel\_flag equal to 0 indicates that viewport position information follows.

**vp\_persistence\_flag** specifies the persistence of the viewport position SEI message for the current layer. vp\_persistence\_flag equal to 0 specifies that the viewport position SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. vp\_persistence\_flag equal to 1 specifies that the viewport position SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing a viewport position SEI message, with either an vp\_cancel\_flag equal to 1 or the same value of vp\_persistence\_flag, applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**vp\_position**[ d ]indicates a position of a viewport along the d axis. When vp\_position[ d ] is not present, it shall be inferred to be equal to represent a view position equal to (0.0, 0.0, 0.0). The values of d equal to 0, 1, and 2 correspond to the X, Y, and Z axis, respectively.

**vp\_rotation\_qx** specifies the x component, qX, for the rotation of the viewport region using the quaternion representation. The value of vp\_rotation\_qx shall be in the range of − 214 to 214, inclusive. When vp\_rotation\_qx is not present, its value shall be inferred to be equal to 0. The value of qX is computed as follows:

qX = vp\_rotation\_qx ÷ 214

**vp\_rotation\_qy** specifies the y component, qY, for the rotation of the viewport region using the quaternion representation. The value of vp\_rotation\_qy shall be in the range of 0 to − 214 to 214, inclusive. When vp\_rotation\_qy is not present, its value shall be inferred to be equal to 0. The value of qY is computed as follows:

qY = vp\_rotation\_qy ÷ 214

**vp\_rotation\_qz** specifies the z component, qZ, for the rotation of the viewport region using the quaternion representation. The value of vp\_rotation\_qz shall be in the range of 0 to − 214 to 214, inclusive. When vp\_rotation\_qz is not present, its value shall be inferred to be equal to 0. The value of qZ is computed as follows:

qZ = vp\_rotation\_qz ÷ 214

The fourth component, qW, for the rotation of the current camera model using the quaternion representation is calculated as follows:

qW = Sqrt( 1 – ( qX2 + qY2 + qZ2 ) )

NOTE – In the context of this document qW is always positive. If a negative qW is desired, one may signal all three syntax elements, vp\_rotation\_qx, vp\_rotation\_qy, and vp\_rotation\_qz, with an opposite sign, which is equivalent.

A unit quaternion can be represented as a rotation matrix vpRotMatrix as follows:

It is a requirement of bitstream conformance that qX2 + qY2 + qZ2 <= 1.

**vp\_center\_view\_flag** equal to 1 indicates that the viewport position signalled correspond to the center of the viewport. vp\_center\_view\_flag equal to 0 indicates that the viewport position signalled correspond to one of two stereo positions of the viewport.

**vp\_left\_view\_flag** equal to 1 indicates that the viewport parameters signalled correspond to the left stereo position of the viewport. vp\_left\_view\_flag equal to 0 indicates that the viewport position signalled correspond to the right stereo positions of the viewport.

* + 1. Decoded Atlas Information Hash SEI message semantics
       1. General

This SEI message defines hash values for decoded atlas information that are associated with the current atlas SeiAtlasID. Prior to computing the hash, the decoded atlas information and corresponding derived data, such as the block to patch information, are arranged into strings of bytes as defined in F.3.16.2, and the hash values are calculated as specified in the semantics of the corresponding syntax elements.

The persistence scope for this SEI message is the remainder of the bitstream or until a new decoded atlas information hash SEI message is encountered. Only the corresponding parameters specified in the SEI message shall be updated. Previously defined parameters from an earlier SEI message shall persist if not modified.

**daih\_cancel\_flag** equal to 1 indicates that the SEI message cancels the persistence of any previous decoded atlas information hash SEI message in coding order that applies to the atlas. daih\_cancel\_flag equal to 0 indicates that decoded atlas information hash follows.

**daih\_persistence\_flag** specifies the persistence of the decoded atlas information hash SEI message. daih\_persistence\_flag equal to 0 specifies that the decoded atlas information hash SEI message applies to the current decoded access unit only.

Let aFrmA be the current atlas frame. daih\_persistence\_flag equal to 1 specifies that the decoded atlas information hash SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing a decoded atlas information hash SEI message, with either an daih\_cancel\_flag equal to 1 or the same value of daih\_persistence\_flag, applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**daih\_hash\_type** indicates the method used to calculate the checksum according to Table F-5. Values of daih\_hash\_type that are not listed in Table F-5 are reserved for future use by ISO/IEC and shall not be present in bitstreams conforming to this version of this document. Decoders shall ignore decoded atlas information hash SEI messages that contain reserved values of hash\_type.

Table F-5 – Atlas hash type

|  |  |
| --- | --- |
| **hash\_type** | **Method** |
| 0 | MD5 (RFC 1321) |
| 1 | CRC |
| 2 | Checksum |

**daih\_decoded\_high\_level\_hash\_present\_flag** equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the high-level syntax elements associated with the atlas, with atlas ID equal to SeiAtlasID, is present. daih\_decoded\_high\_level\_hash\_present\_flag equal to 0 specifies the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the high-level syntax elements associated with the atlas, with atlas ID equal to SeiAtlasID, will not be present.

**daih\_decoded\_atlas\_hash\_present\_flag** equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the patch information associated with the atlas, with atlas ID equal to SeiAtlasID, is present. daih\_decoded\_atlas\_hash\_present\_flag equal to 0 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the patch information associated with the atlas, with atlas ID equal to SeiAtlasID, will not be present.

**daih\_decoded\_atlas\_b2p\_hash\_present\_flag** equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the block to patch map associated with the atlas, with atlas ID equal to SeiAtlasID, is present. daih\_decoded\_atlas\_b2p\_hash\_present\_flag equal to 0 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum value for the block to patch map associated with the atlas, with atlas ID equal to SeiAtlasID, will not be present.

**daih\_decoded\_atlas\_tiles\_hash\_present\_flag** equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum values for the patch information associated with the tiles in the atlas, with atlas ID equal to SeiAtlasID, are present. daih\_decoded\_atlas\_tiles\_hash\_present\_flag equal to 0 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum values for the patch information associated with the tiles in the atlas, with atlas ID equal to SeiAtlasID, will not be present.

**daih\_decoded\_atlas\_tiles\_b2p\_hash\_present\_flag** equal to 1 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum values for the block to patch map associated with the tiles in the atlas, with atlas ID SeiAtlasID, are present. daih\_atlas\_tiles\_b2p\_hash\_present\_flag equal to 0 specifies that the MD5 hash, the cyclic redundancy check (CRC), or the checksum values for the block to patch map associated with the tiles in the atlas, with atlas ID SeiAtlasID, will not be present.

**daih\_reserved\_1bit** shall be equal to 0 in bitstreams conforming to this version of this document. Other values for daih\_reserved\_1bit are reserved for future use by ISO/IEC. Decoders shall ignore the value of daih\_reserved\_1bit.

**daih\_high\_level\_md5**[ i ] is the i-th byte of the 16-byte MD5 hash of the high-level syntax elements of the decoded atlas associated with SeiAtlasID. The value of daih\_high\_level\_md5[ i ] shall be equal to the value digestVal[ i ], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes HighLevelAtlasData with length HighLevelAtlasDataBytes, as specified in subclause F.3.16.2.2:

MD5Init( context )  
 MD5Update( context, HighLevelAtlasData, HighLevelAtlasDataBytes )  
 MD5Final( digestVal, context )

**daih\_high\_level\_crc** is the cyclic redundancy check (CRC) of the high-level syntax elements of the decoded atlas associated with SeiAtlasID. The value of daih\_high\_level\_crc shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes HighLevelAtlasData with length HighLevelAtlasDataBytes, as specified in subclause F.3.16.2.2:

crc = 0xFFFF  
 HighLevelAtlasData[ HighLevelAtlasDataBytes ] = 0  
 HighLevelAtlasData[ HighLevelAtlasDataBytes + 1 ] = 0  
 for( bitIdx = 0; bitIdx < ( HighLevelAtlasDataBytes + 2 ) \* 8; bitIdx++ ) {  
 dataByte = HighLevelAtlasData[ bitIdx >> 3 ]  
 crcMsb = ( crc >> 15 ) & 1  
 bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1  
 crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb \* 0x1021 )  
 }  
 crcVal = crc

**daih\_high\_level\_checksum** is the checksum of the high-level syntax elements of the decoded atlas associated with SeiAtlasID. The value of daih\_high\_level\_checksum shall be equal to the value of checksumVal, obtained using the string of bytes HighLevelAtlasData with length HighLevelAtlasDataBytes, as specified in subclause F.3.16.2.2:

checksum = 0  
 for( i = 0; i < HighLevelAtlasDataBytes ; i++ ) {  
 xorMask = ( i & 0xFF ) ^ ( i >> 8 )  
 checksum = ( checksum + ( HighLevelAtlasData[ i ] & 0xFF) ^ xorMask) & 0xFFFFFFFF  
 checksum = ( checksum + ( HighLevelAtlasData[ i ] >> 8) ^ xorMask) & 0xFFFFFFFF  
 }

**daih\_atlas\_md5**[ i ] is the i-th byte of the 16-byte MD5 hash of the decoded atlas associated with SeiAtlasID. The value of daih\_atlas\_md5[ i ] shall be equal to the value of digestVal[ i ], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes AtlasData with length AtlasDataBytes, as specified in subclause F.3.16.2.3:

MD5Init( context )  
 MD5Update( context, AtlasData, AtlasDataBytes )  
 MD5Final( digestVal, context )

**daih\_atlas\_crc** is the cyclic redundancy check (CRC) of the decoded atlas associated with SeiAtlasID. The value of daih\_atlas\_crc shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes AtlasData with length AtlasDataBytes, as specified in subclause F.3.16.2.3:

crc = 0xFFFF  
 AtlasData[ AtlasDataBytes ] = 0  
 AtlasData[ AtlasDataBytes + 1 ] = 0  
 for( bitIdx = 0; bitIdx < ( AtlasDataBytes + 2 ) \* 8; bitIdx++ ) {  
 dataByte = AtlasData[ bitIdx >> 3 ]  
 crcMsb = ( crc >> 15 ) & 1  
 bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1  
 crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb \* 0x1021 )  
 }  
 crcVal = crc

**daih\_atlas\_checksum** is the checksum of the decoded atlas associated with SeiAtlasID. The value of daih\_atlas\_checksum shall be equal to the value of checksumVal, obtained using the string of bytes AtlasData with length AtlasDataBytes, as specified in subclause F.3.16.2.3:

checksum = 0  
 for( i = 0; i < AtlasDataBytes ; i++ ) {  
 xorMask = ( i & 0xFF ) ^ ( i >> 8 )  
 checksum = ( checksum + ( AtlasData[ i ] & 0xFF ) ^ xorMask ) & 0xFFFFFFFF  
 checksum = ( checksum + ( AtlasData[ i ] >> 8 ) ^ xorMask ) & 0xFFFFFFFF  
 }

**daih\_atlas\_b2p\_md5**[ i ] is the i-thbyte of the 16-byte MD5 hash of the block to patch map of the atlas associated with SeiAtlasID. The value of daih\_atlas\_b2p\_md5[ i ] shall be equal to the value digestVal[ i ], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes AtlasB2PData with length AtlasB2PDataBytes, as specified in subclause F.3.16.2.5:

MD5Init( context )  
 MD5Update( context, AtlasB2PData, AtlasB2PDataBytes )  
 MD5Final( digestVal, context )

**daih\_atlas\_b2p\_crc** is the cyclic redundancy check (CRC) of the block to patch map of the atlas associated with SeiAtlasID. The value of daih\_atlas\_b2p\_crc shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes AtlasB2PData with length AtlasB2PDataBytes, as specified in subclause F.3.16.2.5:

crc = 0xFFFF  
 AtlasB2PData[ AtlasB2PDataBytes ] = 0  
 AtlasB2PData[ AtlasB2PDataBytes + 1 ] = 0  
 for( bitIdx = 0; bitIdx < ( AtlasB2PDataBytes + 2 ) \* 8; bitIdx++ ) {  
 dataByte = AtlasB2PData[ bitIdx >> 3 ]  
 crcMsb = ( crc >> 15 ) & 1  
 bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1  
 crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb \* 0x1021 )  
 }  
 crcVal = crc

**daih\_atlas\_b2p\_checksum** is the checksum of the block to patch map of the atlas associated with SeiAtlasID. The value of daih\_atlas\_b2p\_checksum shall be equal to the value of checksumVal, obtained using the string of bytes AtlasB2PData with length AtlasB2PDataBytes , as specified in subclause F.3.16.2.5:

checksum =0  
 for( i = 0; i < AtlasB2PDataBytes ; i++ ) {  
 xorMask = ( i & 0xFF ) ^ ( i >> 8 )  
 checksum = ( checksum + ( AtlasB2PData[ i ] & 0xFF ) ^ xorMask ) & 0xFFFFFFFF  
 checksum = ( checksum + ( AtlasB2PData[ i ] >> 8) ^ xorMask ) & 0xFFFFFFFF  
 }

**daih\_num\_tiles\_minus1** plus 1 specifies the number of tiles for which hash values will be signalled. The value of daih\_num\_tiles\_minus1 should be in the range of 0 to afti\_num\_tiles\_in\_atlas\_frame\_minus1, inclusive.

**daih\_tile\_id\_len\_minus1** plus 1 specifies the number of bits used to represent the syntax element daih\_tile\_id[ t ]. The value of daih\_tile\_id\_len\_minus1 shall be in the range of 0 to 15, inclusive.

**daih\_tile\_id**[ t ] specifies the tile ID of the t-th tile. The length of the daih\_tile\_id[ t ] syntax element is daih\_tile\_id\_len\_minus1 + 1 bits. When not present, the value of daih\_tile\_id[ t ] is inferred to be equal to t. It is a requirement of bitstream conformance that daih\_tile\_id[ t ] should not be equal to daih\_tile\_id[ j ] for all t != j.

**daih\_atlas\_tiles\_md5**[ t ][ i ] is the i-thbyte of the 16-byte MD5 hash of the decoded atlas tile with tile ID equal to t, associated with SeiAtlasID. The value of daih\_atlas\_tiles\_md5[ t ][ i ] shall be equal to the value digestVal[ i ], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes TileData with length TileDataBytes, as specified in subclause F.3.16.2.4:

MD5Init( context )  
 MD5Update( context, TileData, TileDataBytes )  
 MD5Final( digestVal, context )

**daih\_atlas\_tiles\_crc**[ t ] is the cyclic redundancy check (CRC) of the decoded atlas tile with tile ID equal to t, associated with SeiAtlasID. The value of daih\_atlas\_tiles\_crc[ t ] shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes TileData with length TileDataBytes, as specified in subclause F.3.16.2.4:

crc = 0xFFFF  
 TileData[ TileDataBytes ] = 0  
 TileData[ TileDataBytes + 1 ] = 0  
 for( bitIdx = 0; bitIdx < ( TileDataBytes + 2 ) \* 8; bitIdx++ ) {  
 dataByte = TileData[ bitIdx >> 3 ]  
 crcMsb = ( crc >> 15 ) & 1  
 bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1  
 crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb \* 0x1021 )  
 }  
 crcVal = crc

**daih\_atlas\_tiles\_checksum**[ t ] is the checksum of the decoded atlas tile with tile ID equal to t, associated with SeiAtlasID. The value of daih\_atlas\_tile\_checksum[ t ] shall be equal to the value of checksumVal, obtained using the string of bytes TileData with length TileDataBytes, as specified in subclause F.3.16.2.4:

checksum = 0  
 for( i = 0; i < TileDataBytes ; i++ ) {  
 xorMask = ( i & 0xFF ) ^ ( i >> 8 )  
 checksum = ( checksum + ( TileData[ i ] & 0xFF) ^ xorMask ) & 0xFFFFFFFF  
 checksum = ( checksum + ( TileData[ i ] >> 8 ) ^ xorMask ) & 0xFFFFFFFF  
 }

**daih\_atlas\_tiles\_b2p\_md5**[ t ][ i ] is the i-thbyte of the 16-byte MD5 hash of the block to patch map of the decoded atlas tile, with tile ID equal to t, associated with SeiAtlasID. The value of daih\_atlas\_tiles\_b2p\_md5[ t ][ i ] shall be equal to the value digestVal[ i ], obtained using the MD5 functions defined in IETF RFC 1321 and the string of bytes TileB2PData with length TileB2PDataBytes, as specified in subclause F.3.16.2.6:

MD5Init( context )  
 MD5Update( context, TileB2PData, TileB2PDataBytes )  
 MD5Final( digestVal, context )

**daih\_atlas\_tiles\_b2p\_crc**[ t ] is the cyclic redundancy check (CRC) of the block to patch map of the decoded atlas tile, with tile ID equal to t, associated with SeiAtlasID. The value of daih\_atlas\_tiles\_b2p\_crc[ t ] shall be equal to the value of crcVal, obtained using the CRC specification defined in Rec. ITU-T H.271 and the string of bytes TileB2PData with length TileB2PDataBytes, as specified in subclause F.3.16.2.6:

crc = 0xFFFF  
 TileB2PData[ TileB2PDataBytes ] = 0  
 TileB2pData[ TileB2PDataBytes + 1 ] = 0  
 for( bitIdx = 0; bitIdx < ( TileB2PDataBytes + 2 ) \* 8; bitIdx++ ) {  
 dataByte = TileB2PData[ bitIdx >> 3 ]  
 crcMsb = ( crc >> 15 ) & 1  
 bitVal = ( dataByte >> ( 7 − ( bitIdx & 7 ) ) ) & 1  
 crc = ( ( ( crc << 1 ) + bitVal ) & 0xFFFF ) ^ ( crcMsb \* 0x1021 )  
 }  
 crcVal = crc

**daih\_atlas\_tiles\_b2p\_checksum**[ t ] is the checksum of the block to patch map of the decoded atlas tile, with tile ID equal to t, associated with SeiAtlasID. The value of daih\_atlas\_tile\_checksum[ t ] shall be equal to the value of checksumVal, obtained using the string of bytes TileB2PData with length TileB2PDataBytes, as specified in subclause F.3.16.2.6:

checksum = 0  
 for( i = 0; i < TileB2PDataBytes ; i++ ) {  
 xorMask = ( i & 0xFF ) ^ ( i >> 8 )  
 checksum = ( checksum + ( TileB2PData[ i ] & 0xFF) ^ xorMask ) & 0xFFFFFFFF  
 checksum = ( checksum + ( TileB2PData[ i ] >> 8) ^ xorMask ) & 0xFFFFFFFF  
 }

* + - 1. Strings of bytes arrangement for hash calculation
         1. General

The string of bytes for the high-level syntax, the atlas patch data and for each tile patch data that is associated with the atlas are generated as a string of unsigned 16 bit data as specified in subclauses F.3.16.2.2, F.3.16.2.3, and F.3.16.2.4. Similarly, the block to patch string of bytes for the atlas and for each tile that is associated with the atlas are generated as string of unsigned 16 bit data as specified in subclauses F.3.16.2.5 and F.3.16.2.6, respectively.

* + - * 1. High-level atlas information string of bytes

In this subclause, using ASPS and AFPS parameters, as described in subclauses 8.4.6.1 and 8.4.6.2, a 1D array, HighLevelAtlasData, and a variable, HighLevelAtlasDataBytes, providing the length of the array in bytes, are derived and set as output of this process as follows:

HighLevelAtlasDataBytes = 0  
 HighLevelAtlasDataBytes =  
 ASPSCommonByteString( HighLevelAtlasData, HighLevelAtlasDataBytes)  
 HighLevelAtlasDataBytes =  
 ASPSApplicationByteString( HighLevelAtlasData, HighLevelAtlasDataBytes)  
 HighLevelAtlasDataBytes =  
 AFPSCommonByteString( HighLevelAtlasData, HighLevelAtlasDataBytes)  
 HighLevelAtlasDataBytes =  
 AFPSApplicationByteString( HighLevelAtlasData, HighLevelAtlasDataBytes)

where the functions ASPSCommonByteString, ASPSApplicationByteString, AFPSCommonByteString, and AFPSApplicationByteString are defined in subclauses F.3.16.2.2.1, F.3.16.2.2.2, F.3.16.2.2.3, and F.3.16.2.2.4, respectively.

Common ASPS level string of bytes

Let the function ASPSCommonByteString( stringByte, posByte ) be defined as follows:

ASPSCommonByteString( stringByte, posByte ) {  
 stringByte[ posByte++ ] = asps\_frame\_width && 0xFF  
 stringByte[ posByte++ ] = (asps\_frame\_width >> 8 ) && 0xFF  
 stringByte[ posByte++ ] = (asps\_frame\_width >> 16 ) && 0xFF  
 stringByte[ posByte++ ] = (asps\_frame\_width >> 24 ) && 0xFF  
 stringByte[ posByte++ ] = asps\_frame\_height && 0xFF  
 stringByte[ posByte++ ] = (asps\_frame\_height >> 8 ) && 0xFF  
 stringByte[ posByte++ ] = (asps\_frame\_height >> 16 ) && 0xFF  
 stringByte[ posByte++ ] = (asps\_frame\_height >> 24 ) && 0xFF  
 stringByte[ posByte++ ] = asps\_geometry\_3d\_bit\_depth\_minus1 && 0xFF  
 stringByte[ posByte++ ] = asps\_geometry\_2d\_bit\_depth\_minus1 && 0xFF  
 stringByte[ posByte++ ] = asps\_map\_count\_minus1 && 0xFF  
 stringByte[ posByte++ ] = asps\_max\_number\_projections\_minus1 && 0xFF  
 stringByte[ posByte++ ] = asps\_patch\_precedence\_order\_flag && 0xFF  
 return posByte  
 }

Application-based ASPS level string of bytes

Let the function ASPSApplicationByteString( stringByte, posByte ) be defined as follows:

ASPSApplicationByteString( stringByte, posByte ) {  
 if( asps\_pixel\_deinterleaving\_enabled\_flag ) {  
 for( j = 0; j <= asps\_map\_count\_minus1; j++ )  
 stringByte[ posByte++ ] = asps\_map\_pixel\_deinterleaving\_flag[ j ] && 0xFF  
 }  
  
 stringByte[ posByte++ ] = asps\_raw\_patch\_enabled\_flag && 0xFF  
 stringByte[ posByte++ ] = asps\_eom\_patch\_enabled\_flag && 0xFF  
 if( asps\_eom\_patch\_enabled\_flag && asps\_map\_count\_minus1 == 0 ) {  
 stringByte[ posByte++ ] = asps\_eom\_fix\_bit\_count\_minus1 && 0xFF  
 }  
  
 if( ( asps\_auxiliary\_video\_enabled\_flag ) &&  
 ( asps\_raw\_patch\_enabled\_flag || asps\_eom\_patch\_enabled\_flag ) ) {  
 stringByte[ posByte++ ] = AuxVideoWidthNF && 0xFF  
 stringByte[ posByte++ ] = (AuxVideoWidthNF >> 8 ) && 0xFF  
 stringByte[ posByte++ ] = AuxVideoHeightNF && 0xFF  
 stringByte[ posByte++ ] = ( AuxVideoHeightNF >> 8 ) && 0xFF  
 }  
  
 stringByte[ posByte++ ] = asps\_plr\_enabled\_flag && 0xFF  
 if( asps\_plr\_enabled\_flag ) {   
 for( i = 0; i < asps\_map\_count\_minus1 + 1; i++) {  
 stringByte[ posByte++ ] = plri\_map\_present\_flag[ i ] && 0xFF  
 if( plri\_map\_present\_flag[ i ] ) {  
 stringByte[ posByte++ ] = plri\_number\_of\_modes\_minus1[ i ] && 0xFF  
 for( j = 0; j < plri\_number\_of\_modes\_minus1[ i ] + 1; j++ ) {  
 stringByte[ posByte++ ] = plri\_interpolate\_flag[ i ][ j ] && 0xFF  
 stringByte[ posByte++ ] = plri\_filling\_flag[ i ][ j ] && 0xFF  
 stringByte[ posByte++ ] = plri\_minimum\_depth[ i ][ j ] && 0xFF  
 stringByte[ posByte++ ] = plri\_neighbour\_minus1[ i ][ j ] && 0xFF  
 }  
 stringByte[ posByte++ ] = plri\_block\_threshold\_per\_patch\_minus1[ i ] && 0xFF  
 }  
 }  
 }  
 return posByte  
 }

Common AFPS level string of bytes

Let the function AFPSCommonByteString( stringByte, posByte ) be defined as follows:

AFPSCommonByteString( stringByte, posByte ) {  
 stringByte[ posByte++ ] = afti\_num\_tiles\_in\_atlas\_frame\_minus1 && 0xFF  
 for ( i = 1; i < afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1; i++ ) {  
 stringByte[ posByte++ ] = TileOffsetX[ i ] && 0xFF  
 stringByte[ posByte++ ] = (TileOffsetX[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = TileOffsetY[ i ] && 0xFF  
 stringByte[ posByte++ ] = (TileOffsetY[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = TileWidth[ i ] && 0xFF  
 stringByte[ posByte++ ] = (TileWidth[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = TileHeight[ i ] && 0xFF  
 stringByte[ posByte++ ] = (TileHeight[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = AuxTileOffset[ i ] && 0xFF  
 stringByte[ posByte++ ] = (AuxTileOffset[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = AuxTileWidth[ i ] && 0xFF  
 stringByte[ posByte++ ] = (AuxTileWidth[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = AuxTileHeight[ i ] && 0xFF  
 stringByte[ posByte++ ] = (AuxTileHeight[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = afti\_tile\_id[ i ] && 0xFF  
 stringByte[ posByte++ ] = (afti\_tile\_id[ i ] >> 8 )&& 0xFF  
 }  
 return posByte  
 }

Application-based AFPS level string of bytes

Let the function AFPSApplicationByteString( stringByte, posByte ) be defined as follows:

AFPSApplicationByteString( stringByte, posByte ) {  
 return posByte  
 }

* + - * 1. Atlas information string of bytes

In this subclause, using the atlas patch level information parameters, as described in subclause 9.2.7, an array, AtlasData, and a variable, AtlasDataBytes, providing the length of the array in bytes, are derived and set as output of this process as follows:

AtlasDataBytes = 0  
 for( atlasPatchIdx = 0; atlasPatchIdx < AtlasTotalNumPatches ; atlasPatchIdx++ ) {  
 AtlasDataBytes = AtlasPatchCommonByteString( AtlasData, AtlasDataBytes, atlasPatchIdx )  
 AtlasDataBytes = AtlasPatchApplicationByteString( AtlasData, AtlasDataBytes, atlasPatchIdx )  
 }

where the functions AtlasPatchCommonByteString and AtlasPatchApplicationByteString are defined in subclauses F.3.16.2.3.1 and F.3.16.2.3.2, respectively.

Common atlas patch string of bytes

[Ed.(BK): Rename this clause to avoid ambiguity with common atlas data.]

Let the function AtlasPatchCommonByteString( stringByte, posByte, p ) be defined as follows:

AtlasPatchCommonByteString( stringByte, posByte, p ) {  
 stringByte[ posByte++ ] = AtlasPatchType[ p ] & 0xFF  
  
 stringByte[ posByte++ ] = AtlasPatch2dPosX[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch2dPosX[ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = AtlasPatch2dPosY[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch2dPosY[ p ] >> 8 ) & 0xFF  
  
 stringByte[ posByte++ ] = AtlasPatch2dSizeX[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch2dSizeX[ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = AtlasPatch2dSizeY[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch2dSizeY[ p ] >> 8 ) & 0xFF  
  
 stringByte[ posByte++ ] = AtlasPatch3dOffsetU[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch3dOffsetU[ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = AtlasPatch3dOffsetV[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch3dOffsetV[ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = AtlasPatch3dOffsetV[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch3dOffsetV[ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = AtlasPatch3dOffsetD[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch3dOffsetD[ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = AtlasPatch3dRangeD[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatch3dRangeD[ p ] >> 8 ) & 0xFF  
  
 stringByte[ posByte++ ] = AtlasPatchProjectionID[ p ] & 0xFF  
 stringByte[ posByte++ ] = AtlasPatchOrientationIndex[ p ] & 0xFF  
  
 stringByte[ posByte++ ] = AtlasPatchLoDScaleX[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatchLoDScaleX[ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = AtlasPatchLoDScaleY[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatchLoDScaleY[ p ] >> 8 ) & 0xFF  
  
 return posByte  
 }

Application-based atlas patch string of bytes

Let the function AtlasPatchApplicationByteString( stringByte, posByte, p ) be defined as follows:

AtlasPatchApplicationByteString( stringByte, posByte, p ) {  
 stringByte[ posByte++ ] = AtlasPatchInAuxVideo[ p ] & 0xFF  
 if( AtlasPatchType[ p ] == RAW ) {  
 stringByte[ posByte++ ] = AtlasPatchRawPoints[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatchRawPoints[ p ] >> 8 ) & 0xFF  
 } else if( AtlasPatchType[ p ] == PROJECTED ) {  
 blockCnt = BlockCnt( AtlasPatch2dSizeX[ p ], AtlasPatch2dSizeY[ p ] )  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 if(AtlasPlrdLevel[ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 stringByte[ posByte++ ] =  
 AtlasPlrdBlockModeMinus1[ p ][ m ][ j ] & 0xFF  
 stringByte[ posByte++ ] =  
 (AtlasPlrdBlockModeMinus1[ p ][ m ][ j ] >> 8 ) & 0xFF  
 }  
 } else {  
 stringByte[ posByte++ ] = AtlasPlrdModeMinus1[ p ][ m ] & 0xFF  
 stringByte[ posByte++ ] =  
 (AtlasPlrdBlockModeMinus1[ p ][ m ] >> 8 ) & 0xFF  
 }  
 }  
 }  
 } else if( AtlasPatchType[ p ] == EOM ) {  
 stringByte[ posByte++ ] = AtlasPatchEOMPatchCount[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatchEOMPatchCount[ p ] >> 8 ) & 0xFF  
 for( i = 0; i < AtlasEpduPatchCount[ p ]; i++ ) {  
 stringByte[ posByte++ ] = AtlasEpduPoints[ p ][ i ] & 0xFF  
 stringByte[ posByte++ ] = (AtlasEpduPoints[ p ][ i ] >> 8 ) & 0xFF  
 }  
 }  
 return posByte  
 }

* + - * 1. Tile information string of bytes

In this subclause, using tile patch information variables, as defined in subclause 9.2.5, an array of string of bytes, TileData, and a variable providing the length in bytes of the array, TileDataBytes, for the tile with tile ID, tileID are generated as the output of this process and the following applies:

TileDataBytes = 0  
 for( tilePatchIdx = 0 ; tilePatchIdx < AtduTotalNumPatches[ tileID ] ; tilePatchIdx++ ) {  
 TileDataBytes = TilePatchCommonByteString( TileData, TileDataBytes, tileID, atlasPatchIdx )  
 TileDataBytes = TilePatchApplicationByteString( TileData, TileDataBytes, tileID, atlasPatchIdx )  
 }

where the functions TilePatchCommonByteString and TilePatchApplicationByteString are defined in subclauses F.3.16.2.4.1 and F.3.16.2.4.2, respectively.

Common tile patch string of bytes

Let the function TilePatchCommonByteString( stringByte, posByte, t, p ) be defined as follows:

TilePatchCommonByteString( stringByte, posByte, tileID, p ) {  
 stringByte[ posByte++ ] = TilePatchType[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatchType[ t ][ p ] >> 8 ) & 0xFF  
  
 stringByte[ posByte++ ] = TilePatch2dPosX[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch2dPosX[ t ][ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = TilePatch2dPosY[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch2dPosY[ t ][ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = TilePatch2dSizeX[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch2dSizeX[ t ][ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = TilePatch2dSizeY[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch2dSizeY[ t ][ p ] >> 8 ) & 0xFF  
  
 stringByte[ posByte++ ] = TilePatch3dOffsetU[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch3dOffsetU[ t ][ p ] >> 8) & 0xFF  
 stringByte[ posByte++ ] = TilePatch3dOffsetV[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch3dOffsetV[ t ][ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = TilePatch3dOffsetV[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch3dOffsetV[ t ][ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = TilePatch3dOffsetD[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch3dOffsetD[ t ][ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = TilePatch3dRangeD[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatch3dRangeD[ t ][ p ] >> 8 ) & 0xFF  
  
 stringByte[ posByte++ ] = TilePatchProjectionID[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = TilePatchOrientationIndex[ t ][ p ] & 0xFF  
  
 stringByte[ posByte++ ] = TilePatchLoDScaleX[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatchLoDScaleX[ t ][ p ] >> 8 ) & 0xFF  
 stringByte[ posByte++ ] = TilePatchLoDScaleY[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatchLoDScaleY[ t ][ p ] >> 8) & 0xFF  
  
 return posByte  
 }

Application based tile patch string of bytes

Let the function TilePatchApplicationByteString( stringByte, posByte, t, p ) be defined as follows:

TilePatchApplicationByteString( stringByte, posByte, t, p) {  
 stringByte[ posByte++ ] = TilePatchInAuxVideo[ p ] & 0xFF  
 if( TilePatchType[ t ][ p ] == RAW ) {  
 stringByte[ posByte++ ] = TilePatchRawPoints[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( TilePatchRawPoints[ p ] >> 8 ) & 0xFF  
 } else if( TilePatchType[ t ][ p ] == PROJECTED ) {  
 blockCnt = BlockCnt( AtlasPatch2dSizeX[ t ][ p ], AtlasPatch2dSizeY[ t ][ p ] )  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 if(TilePlrdLevel[ t ][ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 stringByte[ posByte++ ] =  
 TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ][ j ] & 0xFF  
 stringByte[ posByte++ ]=  
 (TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ][ j ] >> 8 ) & 0xFF  
 }  
 } else {  
 stringByte[ posByte++ ] = TilePatchPlrdModeMinus1[ t ][ p ][ m ] & 0xFF  
 stringByte[ posByte++ ] =  
 (TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ] >> 8 ) & 0xFF  
 }  
 }  
 }  
 } else if( TilePatchType[ tileID ][ p ] == EOM ) {  
 stringByte[ posByte++ ] = TilePatchEOMPatchCount[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatchEOMPatchCount[ t ][ p ] >> 8 ) & 0xFF  
 for( i = 0; i < TilePatchEomPatchCount[ t ][ p ]; i++ ) {  
 stringByte[ posByte++ ] = TileEpduPoints[ t ][ p ][ i ] & 0xFF  
 stringByte[ posByte++ ] = (TileEpduPoints[ t ][ p ][ i ] >> 8 ) & 0xFF  
 }  
 }  
 return posByte  
 }

* + - * 1. Atlas block to patch string of bytes

In this subclause, using the block to patch maps from each atlas, as defined in subclause 9.2.7.2, an array, AtlasB2PData and a variable AtlasB2PDataBytes,providing the length of the array in bytes, are derived and set as output of this process as follows:

AtlasB2PDataBytes = 0  
 for( y = 0; y < AtlasBlockToPatchMapHeight; y++ ) {  
 for( x = 0; x < AtlasBlockToPatchMapWidth; x++ ) {  
 b2pVal = ( AtlasBlockToPatchMap[ y ][ x ] == −1 ) ?  
 0xFFFF : AtlasBlockToPatchMap[ y ][ x ]  
 AtlasB2pData[ AtlasB2PDataBytes++ ] = b2pVal & 0xFF  
 AtlasB2pData[ AtlasB2PDataBytes++ ] = (b2pVal >> 8 ) & 0xFF  
 }  
 }

* + - * 1. Tile block to patch string of bytes

In this subclause, using the block to patch maps for a tile with tile ID equal to t, as defined in subclause 9.2.6, an array, TileB2PData and the variable TileB2PDataBytes, providing the length in bytes of the array, are derived and set as output of this process as follows:

TileB2PDataBytes = 0  
 for( y = 0; y < TileBlockToPatchMapHeight[ t ]; y++ ) {  
 for( x = 0; x < TileBlockToPatchMapWidth[ t ]; x++ ) {  
 b2pVal = ( TileBlockToPatchMap[ t ][ y ][ x ] == −1 ) ?  
  0xFFFF : TileBlockToPatchMap[ t ][ y ][ x ]  
 TileB2pData[ TileB2PDataBytes++ ] = b2pVal & 0xFF  
 TileB2pData[ TileB2PDataBytes++ ] = (b2pVal >> 8 ) & 0xFF  
 }  
 }

* + 1. Time code SEI message semantics

The time code SEI message provides time code information similar to that defined by SMPTE ST 12-1 (2014) for atlas frame(s).

The contents of the clock timestamp syntax elements indicate a time of origin, capture, or alternative ideal display. This indicated time is computed as

clockTimestamp = ( ( hH \* 60 + mM ) \* 60 + sS ) \* time\_scale +  
 nFrames \* num\_units\_in\_tick + tOffset (F‑3)

in units of clock ticks of a clock with clock frequency equal to time\_scale Hz, relative to some unspecified point in time for which clockTimestamp would be equal to 0. Output order and DAB output timing are not affected by the value of clockTimestamp.

NOTE 1 – clockTimestamp time indications may aid display on devices with refresh rates other than those well-matched to DAB output times. However, the time code SEI message does not affect the output order and DPB output timing defined in this document.

**num\_units\_in\_tick** is the number of time units of a clock operating at the frequency time\_scale Hz that corresponds to one increment (called a clock tick) of a clock tick counter. num\_units\_in\_tick shall be greater than 0. A clock tick, in units of seconds, is equal to the quotient of num\_units\_in\_tick divided by time\_scale. For example, when the frame rate of an atlas signal is 25 Hz, time\_scale may be equal to 27 000 000 and num\_units\_in\_tick may be equal to 1 080 000 and consequently a clock tick may be equal to 0.04 seconds. When vui\_timing\_info\_present\_flag is equal to 1, it is a requirement of V3C bitstream conformance that the value of num\_units\_in\_tick shall be equal to the value of vui\_num\_units\_in\_tick.

**time\_scale** is the number of time units that pass in one second. For example, a time coordinate system that measures time using a 27 MHz clock has a time\_scale of 27 000 000. The value of time\_scale shall be greater than 0. When vui\_timing\_info\_present\_flag is equal to 1, it is a requirement of V3C bitstream conformance that the value of time\_scale shall be equal to the value of vui\_time\_scale.

**counting\_type** specifies the method of dropping values of the n\_frames syntax element as specified in Table F-6.

Table F-6 – Definition of counting\_type values

|  |  |
| --- | --- |
| Value | Interpretation |
| 0 | no dropping of n\_frames count values and no use of time\_offset\_value |
| 1 | no dropping of n\_frames count values |
| 2 | dropping of individual zero values of n\_frames count |
| 3 | dropping of individual values of n\_frames count equal to MaxFPS − 1 |
| 4 | dropping of unspecified individual n\_frames count values |
| 5 | dropping of unspecified numbers of unspecified n\_frames count values |
| 6..31 | reserved |

**full\_timestamp\_flag** equal to 1 specifies that the n\_frames syntax element is followed by seconds\_value, minutes\_value, and hours\_value. full\_timestamp\_flag equal to 0 specifies that the n\_frames syntax element is followed by seconds\_flag.

**discontinuity\_flag** equal to 0 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous set of clock timestamp syntax elements in output order can be interpreted as the time difference between the times of origin or capture of the associated frames. discontinuity\_flag equal to 1 indicates that the difference between the current value of clockTimestamp and the value of clockTimestamp computed from the previous set of clock timestamp syntax elements in output order should not be interpreted as the time difference between the times of origin or capture of the associated frames. When discontinuity\_flag is equal to 0, the value of clockTimestamp shall be greater than or equal to the value of clockTimestamp for the previous set of clock timestamp syntax elements in output order (when present).

**cnt\_dropped\_flag** specifies the skipping of one or more values of n\_frames using the counting method specified by counting\_type.

**n\_frames** specifies the value of nFrames used to compute clockTimestamp. n\_frames shall be less than MaxFPS, where MaxFPS is specified by

MaxFPS = Ceil(time\_scale ÷ num\_units\_in\_tick ) (F‑4)

When counting\_type is equal to 2 and cnt\_dropped\_flag is equal to 1, n\_frames shall be equal to 1 and the value of n\_frames for the previous set of clock timestamp syntax elements in output order (when present) shall not be equal to 0 unless discontinuity\_flag is equal to 1.

NOTE 2 – When counting\_type[ i ] is equal to 2, the need for increasingly large magnitudes of tOffset in Equation F‑3 when using fixed non-integer frame rates (e.g., 12.5 frames per second with time\_scale equal to 50 and num\_units\_in\_tick equal to 2) can be avoided by occasionally skipping over the value n\_frames equal to 0 when counting (e.g., counting n\_frames from 0 to 12, then incrementing seconds\_value and counting n\_frames from 1 to 12, then incrementing seconds\_value and counting n\_frames from 0 to 12, etc.).

When counting\_type is equal to 3, and cnt\_dropped\_flag is equal to 1, n\_frames shall be equal to 0 and the value of n\_frames for the previous set of clock timestamp syntax elements in output order (when present) shall not be equal to MaxFPS − 1 unless discontinuity\_flag is equal to 1.

NOTE 3 – When counting\_type is equal to 3, the need for increasingly large magnitudes of tOffset in Equation F‑3 when using fixed non-integer frame rates (e.g., 12.5 frames per second with time\_scale equal to 50 and num\_units\_in\_tick equal to 2) can be avoided by occasionally skipping over the value n\_frames equal to MaxFPS − 1 when counting (e.g., counting n\_frames from 0 to 12, then incrementing seconds\_value and counting n\_frames from 0 to 11, then incrementing seconds\_value and counting n\_frames from 0 to 12, etc.).

When counting\_type is equal to 4 or 5, and cnt\_dropped\_flag is equal to 1, n\_frames shall not be equal to 1 plus the value of n\_frames for the previous set of clock timestamp syntax elements in output order (when present) modulo MaxFPS unless discontinuity\_flag is equal to 1.

NOTE 4 – When counting\_type is equal to 4 or 5, the need for increasingly large magnitudes of tOffset in Equation F‑3 when using fixed non-integer frame rates can be avoided by occasionally skipping over some values of n\_frames when counting. The specific values of n\_frames that are skipped are not specified when counting\_type is equal to 4 or 5.

**seconds\_flag** equal to 1 specifies that seconds\_value and minutes\_flag are present when full\_timestamp\_flag is equal to 0. seconds\_flag equal to 0 specifies that seconds\_value and minutes\_flag are not present.

**seconds\_value** specifies the value of sS used to compute clockTimestamp. The value of seconds\_value shall be in the range of 0 to 59, inclusive. When seconds\_value is not present, its value is inferred to be equal to the value of seconds\_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous seconds\_value shall have been present.

**minutes\_flag** equal to 1 specifies that minutes\_value and hours\_flag are present when full\_timestamp\_flag is equal to 0 and seconds\_flag is equal to 1. minutes\_flag equal to 0 specifies that minutes\_value and hours\_flag are not present.

**minutes\_value** specifies the value of mM used to compute clockTimestamp. The value of minutes\_value shall be in the range of 0 to 59, inclusive. When minutes\_value is not present, its value is inferred to be equal to the value of minutes\_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous minutes\_value shall have been present.

**hours\_flag** equal to 1 specifies that hours\_value is present when full\_timestamp\_flag is equal to 0 and seconds\_flag is equal to 1 and minutes\_flag is equal to 1.

**hours\_value** specifies the value of hH used to compute clockTimestamp. The value of hours\_value shall be in the range of 0 to 23, inclusive. When hours\_value is not present, its value is inferred to be equal to the value of hours\_value for the previous set of clock timestamp syntax elements in decoding order, and it is required that such a previous hours\_value shall have been present.

**time\_offset\_length** greater than 0 specifies the length in bits of the time\_offset\_value syntax element. time\_offset\_length equal to 0 specifies that the time\_offset\_value syntax element is not present. When counting\_type is equal to 0, time\_offset\_length shall be equal to 0.

NOTE 5 – time\_offset\_length should be the same for all atlas frames in the CAS.

**time\_offset\_value** specifies the value of tOffset used to compute clockTimestamp. The number of bits used to represent time\_offset\_value is equal to time\_offset\_length. When time\_offset\_value is not present, its value is inferred to be equal to 0.

* + 1. Packed independent regions SEI payload semantics

**pir\_num\_packed\_frames\_minus1** plus 1 specifies the number of packed frames for which independently decodable region information is signaled.

**pir\_packed\_frame\_id**[ j ] specifies the ID of the pack with index j. The value of pir\_packed\_frame\_id[ j ] shall be in the range of 0 to 15, inclusive.

**pir\_description\_type\_idc**[ k ] equal to 0 indicates that tile indices of the top left and bottom right tiles of an independently decodable rectangular region (such as a temporal motion-constrained tile set specified in ISO/IEC 23008-2) are signaled for the rectangles of the k-th packed frame. pir\_description\_type\_idc[ k ] equal to 1 indicates that sub pictures IDs, as specified in ISO/IEC 23090-3, are signaled for the regions of the k-th packed frame The value of pir\_description\_type\_idc[ k ] shall be in the range of 0 to 1. The values in the range of 2 .. 3 are reserved for future use by ISO/IEC.

**pir\_num\_regions\_minus1**[ k ] plus 1 specifies the number of rectangular regions of the packed frame with ID k for which independently decodable region information is signaled.

**pir\_top\_left\_tile\_idx**[ k ][ i ] and **pir\_bottom\_right\_tile\_idx**[ k ][ i ], when present, identify the tile indices of the top-left tile and the bottom-right tile of an independently decodable region (such as a temporal motion-constrained tile set specified in ISO/IEC 23008-2), respectively, in tile raster scan order, corresponding to the i-th region of the video sub-bitstream of the packed frame with ID k.

**pir\_subpic\_id**[ k ][ i ], when present, identifies the subpicture ID, as specified in ISO/IEC 23090-3, corresponding to the i-th rectangle of the video sub-bitstream of the pack with ID k.

1. (informative)  
   Volumetric usability information
   1. General

This annex specifies syntax and semantics of the VUI parameters of the ASPSs associated with an atlas, identified with an atlas ID, VUIAtlasID, which is set equal to vuh\_atlas\_id or determined through external means if the V3C unit header is unavailable.

VUI parameters are not required for constructing the atlas by the decoding process. Conforming decoders are not required to process this information for output order conformance to this document (see Annex E for the specification of output order conformance). Some VUI parameters are required to check bitstream conformance and for output timing decoder conformance.

In Annex G, specification for presence of VUI parameters is also satisfied when those parameters (or some subset of them) are conveyed to decoders (or to the HRD) by other means not specified in this document. When present in the bitstream, VUI parameters shall follow the syntax and semantics specified in this annex. When the content of VUI parameters is conveyed for the application by some means other than presence within the bitstream, the representation of the content of the VUI parameters is not required to use the same syntax specified in this annex. For the purpose of counting bits, only the appropriate bits that are actually present in the bitstream are counted.

* 1. VUI syntax
     1. VUI parameters syntax

|  |  |  |
| --- | --- | --- |
| vui\_parameters( ) { | **Descriptor** | |
| **vui\_timing\_info\_present\_flag** | u(1) | |
| if( vui\_timing\_info\_present\_flag ) { |  | |
| **vui\_num\_units\_in\_tick** | u(32) | |
| **vui\_time\_scale** | u(32) | |
| **vui\_poc\_proportional\_to\_timing\_flag** | u(1) | |
| if( vui\_poc\_proportional\_to\_timing\_flag ) |  | |
| **vui\_num\_ticks\_poc\_diff\_one\_minus1** | ue(v) | |
| **vui\_hrd\_parameters\_present\_flag** | u(1) | |
| if( vui\_hrd\_parameters\_present\_flag ) |  | |
| hrd\_parameters( 0 ) |  | |
| } |  | |
| **vui\_tile\_restrictions\_present\_flag** | u(1) | |
| if( vui\_tile\_restrictions\_present\_flag ) { |  | |
| **vui\_fixed\_atlas\_tile\_structure\_flag** | u(1) | |
| **vui\_fixed\_video\_tile\_structure\_flag** | u(1) | |
| **vui\_constrained\_tiles\_across\_v3c\_components\_idc** | ue(v) | |
| **vui\_max\_num\_tiles\_per\_atlas\_minus1** | ue(v) | |
| } |  | |
| **vui\_max\_coded\_video\_resolution\_present\_flag** | u(1) | |
| if( vui\_max\_coded\_video\_resolution\_present\_flag ) |  | |
| max\_coded\_video\_resolution( ) |  | |
| **vui\_coordinate\_system\_parameters\_present\_flag** | u(1) |
| if( vui\_coordinate\_system\_parameters\_present\_flag ) |  |
| coordinate\_system\_parameters( ) |  |
| **vui\_unit\_in\_metres\_flag** | u(1) | |
| **vui\_display\_box\_info\_present\_flag** | u(1) |
| if( **vui\_display\_box\_info\_present\_flag**) { |  |
| for( d = 0; d < 3; d++ ) { |  |
| **vui\_display\_box\_origin**[ d ] | u(v) |
| **vui\_display\_box\_size**[ d ] | u(v) |
| } |  |
| } |  |
| **vui\_anchor\_point\_present\_flag** | u(1) |
| if( vui\_anchor\_point\_present\_flag ) |  |
| for( d = 0; d < 3; d++ ) |  |
| **vui\_anchor\_point**[ d ] | u(v) |
| } |  | |

* + 1. HRD parameters syntax

|  |  |
| --- | --- |
| hrd\_parameters( maxNumSubLayersMinus1) { | Descriptor |
| **hrd\_nal\_parameters\_present\_flag** | u(1) |
| **hrd\_acl\_parameters\_present\_flag** | u(1) |
| if( hrd\_nal\_parameters\_present\_flag || hrd\_acl\_parameters\_present\_flag ) { |  |
| **hrd\_bit\_rate\_scale** | u(4) |
| **hrd\_cab\_size\_scale** | u(4) |
| } |  |
| for( i = 0; i <= maxNumSubLayersMinus1; i++ ) { |  |
| **hrd\_fixed\_atlas\_rate\_general\_flag**[ i ] | u(1) |
| if( !hrd\_fixed\_atlas\_rate\_general\_flag[ i ] ) |  |
| **hrd\_fixed\_atlas\_rate\_within\_cas\_flag**[ i ] | u(1) |
| else |  |
| hrd\_fixed\_atlas\_rate\_within\_cas\_flag[ i ] = 1 |  |
| if( hrd\_fixed\_atlas\_rate\_within\_cas\_flag[ i ] ) { |  |
| **hrd\_elemental\_duration\_in\_tc\_minus1**[ i ] | ue(v) |
| hrd\_low\_delay\_flag[ i ] = 0 |  |
| } else |  |
| **hrd\_low\_delay\_flag**[ i ] | u(1) |
| if( !hrd\_low\_delay\_flag[ i ] ) |  |
| **hrd\_cab\_cnt\_minus1**[ i ] | ue(v) |
| else |  |
| hrd\_cab\_cnt\_minus1[ i ] = 0 |  |
| if( hrd\_nal\_parameters\_present\_flag ) |  |
| hrd\_sub\_layer\_parameters( 0, i ) |  |
| if( hrd\_acl\_parameters\_present\_flag ) |  |
| hrd\_sub\_layer\_parameters( 1, i ) |  |
| } |  |
| } |  |

* + 1. Sub-layer HRD parameters syntax

|  |  |
| --- | --- |
| hrd\_sub\_layer\_parameters( hrdMode, subLayerID ) { | **Descriptor** |
| for( j = 0; j <= CabCnt; j++ ) { |  |
| **hrd\_bit\_rate\_value\_minus1**[ hrdMode ][ subLayerID ][ j ] | ue(v) |
| **hrd\_cab\_size\_value\_minus1**[ hrdMode ][ subLayerID ][ j ] | ue(v) |
| **hrd\_cbr\_flag**[ hrdMode ][ subLayerID ][ j ] | u(1) |
| } |  |
| } |  |

* + 1. Maximum coded video resolution syntax

|  |  |
| --- | --- |
| max\_coded\_video\_resolution( ) { | **Descriptor** |
| **mcv\_occupancy\_resolution\_present\_flag** | u(1) |
| **mcv\_geometry\_resolution\_present\_flag** | u(1) |
| **mcv\_attribute\_resolution\_present\_flag** | u(1) |
| if( mcv\_occupancy\_resolution\_present\_flag ) { |  |
| **mcv\_occupancy\_width** | ue(v) |
| **mcv\_occupancy\_height** | ue(v) |
| } |  |
| if( mcv\_geometry\_resolution\_present\_flag ) { |  |
| **mcv\_geometry\_width** | ue(v) |
| **mcv\_geometry\_height** | ue(v) |
| } |  |
| if( mcv\_attribute\_resolution\_present\_flag ) { |  |
| **mcv\_attribute\_width** | ue(v) |
| **mcv\_attribute\_height** | ue(v) |
| } |  |
| } |  |

* + 1. Coordinate system parameters syntax

|  |  |
| --- | --- |
| coordinate\_system\_parameters( ) { | **Descriptor** |
| **csp\_forward\_axis** | u(2) |
| **csp\_delta\_left\_axis\_minus1** | u(1) |
| **csp\_forward\_sign** | u(1) |
| **csp\_left\_sign** | u(1) |
| **csp\_up\_sign** | u(1) |
| } |  |

* 1. VUI semantics
     1. VUI parameters semantics

**vui\_timing\_info\_present\_flag** equal to 1 specifies that vui\_num\_units\_in\_tick, vui\_time\_scale, vui\_poc\_proportional\_to\_timing\_flag, and vui\_hrd\_parameters\_present\_flag are present in the vui\_parameters( ) syntax structure. vui\_timing\_info\_present\_flag equal to 0 specifies that vui\_num\_units\_in\_tick, vui\_time\_scale, vui\_poc\_proportional\_to\_timing\_flag, and vui\_hrd\_parameters\_present\_flag are not present in the vui\_parameters( ) syntax structure.

**vui\_num\_units\_in\_tick** is the number of time units of a clock operating at the frequency vui\_time\_scale Hz that corresponds to one increment (called a clock tick) of a clock tick counter. vui\_num\_units\_in\_tick shall be greater than 0. A clock tick, in units of seconds, is equal to the quotient of vui\_num\_units\_in\_tick divided by vui\_time\_scale. For example, when the frame rate of an atlas signal is 25 Hz, vui\_time\_scale may be equal to 27 000 000 and vui\_num\_units\_in\_tick may be equal to 1 080 000 and consequently a clock tick may be equal to 0.04 seconds.

**vui\_time\_scale** is the number of time units that pass in one second. For example, a time coordinate system that measures time using a 27 MHz clock has a vui\_time\_scale of 27 000 000. The value of vui\_time\_scale shall be greater than 0.

**vui\_poc\_proportional\_to\_timing\_flag** equal to 1 indicates that the atlas frame order count value for each atlas frame in the CAS that is not the first atlas frame in the CAS, in decoding order, is proportional to the output time of the atlas frame relative to the output time of the first atlas frame in the CAS. vui\_poc\_proportional\_to\_timing\_flag equal to 0 indicates that the atlas frame order count value for each atlas frame in the CAS that is not the first atlas frame in the CAS, in decoding order, may or may not be proportional to the output time of the atlas frame relative to the output time of the first atlas frame in the CAS.

**vui\_num\_ticks\_poc\_diff\_one\_minus1** plus 1 specifies the number of clock ticks corresponding to a difference of atlas frame order count values equal to 1. The value of vui\_num\_ticks\_poc\_diff\_one\_minus1 shall be in the range of 0 to 232 − 2, inclusive.

**vui\_hrd\_parameters\_present\_flag** equal to 1 specifies that the syntax structure hrd\_parameters( ) is present in the vui\_parameters( ) syntax structure. vui\_hrd\_parameters\_present\_flag equal to 0 specifies that the syntax structure hrd\_parameters( ) is not present in the vui\_parameters( ) syntax structure.

**vui\_tile\_restrictions\_present\_flag** equal to 1 specifies that the syntax elements vui\_fixed\_atlas\_tile\_structure\_flag, vui\_fixed\_video\_tile\_structure\_flag, vui\_constrained\_tiles\_across\_v3c\_components\_idc, and vui\_max\_num\_tiles\_per\_atlas\_minus1 are present in the vui\_parameters( ) syntax structure. vui\_tile\_restrictions\_present\_flag equal to 0 specifies that the syntax elements vui\_fixed\_atlas\_tile\_structure\_flag, vui\_fixed\_video\_tile\_structure\_flag, vui\_constrained\_tiles\_across\_v3c\_components\_idc, and vui\_max\_num\_tiles\_per\_atlas\_minus1 are not present in the vui\_parameters( ) syntax structure.

**vui\_fixed\_atlas\_tile\_structure\_flag** equal to 1 indicates that all the atlas frames of the current atlas shall have the same tiling structure. vui\_fixed\_atlas\_tile\_structure\_flag equal to 0 indicates that atlas frames of the current atlas may or may not have the same tiling structure. When the vui\_fixed\_atlas\_tile\_structure\_flag syntax element is not present, it is inferred to be equal to 0.

**vui\_fixed\_video\_tile\_structure\_flag** equal to 1 indicates that for each video sub-bitstream associated with the current atlas, all of its frames shall have the same tiling structure. vui\_fixed\_video\_tile\_structure\_flag equal to 0 indicates that for each video sub-bitstream associated with the current atlas, frames may or may not have the same tiling structure. When the vui\_fixed\_video\_tile\_structure\_flag syntax element is not present, it is inferred to be equal to 0.

**vui\_constrained\_tiles\_across\_v3c\_components\_idc** indicates whether any constraints apply to the sizes of tiles in the atlas sub-bitstream and the video tiles in the video sub-bitstreams, as specified in Table G.7.

Table G.7 – Definition of vui\_constrained\_tiles\_across\_v3c\_components\_idc

|  |  |
| --- | --- |
| **Value** | **Interpretation** |
| 0 | Unconstrained |
| 1 | Proportionally constrained video tiles |
| 2 | Atlas based constrained video tiles with exact match |
| 3 | Atlas based constrained video tiles |
| 4 | Edge based constrained video tiles |

Values outside the 0 to 4 range, inclusive, are reserved for future use by ISO/IEC. It is a requirement of bitstream conformance that bitstreams conforming to this version of this document shall not contain such values of vui\_constrained\_tiles\_across\_v3c\_components\_idc.

Video tiles can be independent coding units defined by the video or image coding specification and may vary in name according to such specification. Names could include slices, partitions, tiles, or sub-pictures, among others. Some specifications may not support multiple tiles, i.e. are limited to a single video tile. The specification of such tiles is outside the scope of this document.

vui\_constrained\_tiles\_across\_v3c\_components\_idc equal to 0 means that the tile sizes of the video are constrained only by the video coding specification used. The tile sizes of the atlas sub-bitstream are constrained by this document.

vui\_constrained\_tiles\_across\_v3c\_components\_idc equal to 1 means that the tile sizes of the video and atlas sub-bitstreams are constrained as follows:

For each tile in the atlas sub-bitstream, there is a corresponding video tile for each video sub-bitstream present (attributes, geometry, and occupancy) such that the video tile, when scaled to the nominal format, as defined in Annex B.2, represents exactly the same area on the atlas as the corresponding atlas tile.

NOTE 1 – In this version of this document the nominal format has the same resolution as the atlas.

Let the width and height of the video be videoWidth and videoHeight, respectively. Let the width and height of the video tile be videoTileWidth and videoTileHeight, respectively. Let the width and height of the atlas tile be atlasTileWidth and atlasTileHeight, respectively. Then,

videoTileWidth = atlasTileWidth \* videoWidth / asps\_frame\_width and

videoTileHeight = atlasTileHeight \* videoHeight / asps\_frame\_height.

It shall be possible to decode each video tile that corresponds to a tile in the atlas sub-bitstream without reference to any information from other video tiles in that sub-bitstream.

vui\_constrained\_tiles\_across\_v3c\_components\_idc equal to 2 means that the tile sizes of the video and atlas sub-bitstreams are constrained as follows:

For each tile in the atlas sub-bitstream, there is a set of video tiles for each video sub-bitstream present (occupancy, geometry, and attributes) such that the set of video tiles, when scaled to the nominal format, as defined in Annex B.2, together, represent exactly the same area on the atlas as the atlas tile.

NOTE 2 – In this version of this document the nominal format has the same resolution as the atlas.

It shall be possible to decode each set of video tiles that corresponds to a tile in the atlas sub-bitstream without reference to any information from video tiles from that sub-bitstream that are outside that set.

vui\_constrained\_tiles\_across\_v3c\_components\_idc equal to 3 means that the tile sizes of the video and atlas sub-bitstreams are constrained as follows:

For each tile in the atlas sub-bitstream, there is a corresponding video tile for each video sub-bitstream present (occupancy, geometry, and attributes) such that the video tile, when scaled to the nominal format, as defined in Annex B.2, represents an area on the atlas that is greater than or equal to the area represented by the atlas tile.

NOTE 3 – In this version of this document the nominal format has the same resolution as the atlas.

The number of luma samples in the video tile shall be less than or equal to the number of samples in the atlas tile.

It shall be possible to decode each video tile that corresponds to a tile in the atlas sub-bitstream without reference to any information from other video tiles in that sub-bitstream.

vui\_constrained\_tiles\_across\_video\_components\_idc equal to 4 means that for all video sub-bitstreams, the tiling pattern, after scaling to the nominal format, as defined in Annex B.2, is considered, is such that one of the following must hold for any pair of tiles, X from one sub-bitstream and Y from another sub‑bitstream:

* X and Y are non-overlapping
* X is completely contained in Y
* Y is completely contained in X.

It shall be possible to decode each tile in the video sub-bitstreams without reference to any information from other tiles in that video bitstream.

NOTE 4 – In this version of this document the nominal format has the same resolution as the atlas.

The sizes of the tiles in the auxiliary atlas sub-bitstream and the video tiles in the auxiliary video sub-bitstreams are also constrained depending on the value of vui\_constrained\_tiles\_across\_video\_components\_idc.

**vui\_max\_num\_tiles\_per\_atlas\_minus1** plus 1indicates the maximum number of tiles present in the CAS.

**vui\_max\_coded\_video\_resolution\_present\_flag** equal to 1 specifies that the syntax structure max\_coded\_video\_resolution( ) is present in the vui\_parameters( ) syntax structure. vui\_max\_coded\_video\_resolution\_present\_flag equal to 0 specifies that the syntax structure max\_coded\_video\_resolution( ) is not present in the vui\_parameters( ) syntax structure.

**vui\_coordinate\_system\_parameters\_present\_flag** equal to 1 specifies that the syntax structure coordinate\_system\_parameters( ) is present in the vui\_parameters( ) syntax structure. vui\_coordinate\_system\_parameters\_present\_flag equal to 0 specifies that the syntax structure coordinate\_system\_parameters( ) is not present in the vui\_parameters( ) syntax structure.

**vui\_unit\_in\_metres\_flag** equal to 1 specifies that the real-world coordinates information is expressed in metres. vui\_unit\_in\_metres\_flag equal to 0 specifies that the world coordinates are unitless. If vui\_unit\_in\_metres\_flag is not present, it shall be inferred to be equal to 0.

**vui\_display\_box\_info\_present\_flag** equal to 1 specifies that the syntax elements vui\_display\_box\_origin[ d ] and vui\_display\_box\_size[ d ] are present in the vui\_parameters( ) syntax structure. vui\_display\_box\_info\_present\_flag equal to 0 specifies that the syntax elements vui\_display\_box\_origin[ d ] and vui\_display\_box\_size[ d ] elements are not present in the vui\_parameters( ) syntax structure. The values of d equal to 0, 1, and 2 correspond to the X, Y, and Z axis, respectively.

**vui\_display\_box\_origin**[ d ] specifies an offset with respect to the coordinate system origin point along the axis d. When elements of the vui\_display\_box\_origin[ d ] are not present, its values shall be inferred to be equal to 0. The number of bits used to represent vui\_display\_box\_origin[ d ] is asps\_geometry\_3d\_bit\_depth\_minus1 +1. The values of d equal to 0, 1, and 2 correspond to the X, Y, and Z axis, respectively.

**vui\_display\_box\_size**[ d ] specifies the size of the display box in terms of samples in the direction of the axis d. When elements of the vui\_display\_box\_size[ d ] are not present, its values are unknown. The number of bits used to represent vui\_display\_box\_origin[ d ] is asps\_geometry\_3d\_bit\_depth\_minus1 + 1.

The following variables are derived from the display box parameters:

minOffset[ d ] = vui\_display\_box\_origin[ d ] (G‑1)

maxOffset[ d ] = vui\_display\_box\_origin[ d ] + vui\_display\_box\_size[ d ] (G‑2)

where the values of d equal to 0, 1, and 2 correspond to the X, Y, and Z axis, respectively.

**vui\_anchor\_point\_present\_flag** equal to 1 indicates that vui\_anchor\_point[ d ] syntax elements are present in the vui\_parameters( ) syntax structure. vui\_anchor\_point\_present\_flag equal to 0 indicates that vui\_anchor\_point[ d ] elements are not present.

**vui\_anchor\_point**[ d ] indicates a position of an anchor point along the d axis. The value of vui\_anchor\_point[ d ] shall be in range of 0 to 2asps\_geometry\_3d\_bit\_depth\_minus1+1 − 1. If vui\_anchor\_point[ d ] is not present it shall be inferred to be equal to 0. The number of bits used to represent vui\_anchor\_point[ d ] is asps\_geometry\_3d\_bit\_depth\_minus1 + 1. The values of d equal to 0, 1, and 2 correspond to the X, Y, and Z axis, respectively.

* + 1. HRD parameters semantics

The hrd\_parameters( ) syntax structure provides HRD parameters used in the HRD operations for a layer set. It is a requirement for bitstream conformance to this version of this document that only one layer shall be supported, i.e. nal\_layer\_id and nal\_temporal\_id shall be equal to 0. When the hrd\_parameters( ) syntax structure is included in an ASPS, the layer set to which the hrd\_parameters( ) syntax structure applies is the layer set for which the associated layer identifier list contains all nal\_layer\_id values present in the CAS.

For interpretation of the following semantics, the bitstream (or a part thereof) refers to the bitstream subset (or a part thereof) associated with the layer set to which the hrd\_parameters( ) syntax structure applies.

**hrd\_nal\_parameters\_present\_flag** equal to 1 specifies that NAL HRD parameters (pertaining to Type II bitstream conformance) are present in the hrd\_parameters( ) syntax structure. hrd\_nal\_parameters\_present\_flag equal to 0 specifies that NAL HRD parameters are not present in the hrd\_parameters( ) syntax structure.

NOTE 1 – When hrd\_nal\_parameters\_present\_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the NAL HRD parameters and all buffering period and atlas timing SEI messages, by some means not specified in this document.

**hrd\_acl\_parameters\_present\_flag** equal to 1 specifies that ACL HRD parameters (pertaining to all bitstream conformance types) are present in the hrd\_parameters( ) syntax structure. hrd\_acl\_parameters\_present\_flag equal to 0 specifies that ACL HRD parameters are not present in the hrd\_parameters( ) syntax structure.

NOTE 2 – When hrd\_acl\_parameters\_present\_flag is equal to 0, the conformance of the bitstream cannot be verified without provision of the ACL HRD parameters and all buffering period and atlas timing SEI messages, by some means not specified in this document.

**hrd\_bit\_rate\_scale**, together with hrd\_bit\_rate\_value\_minus1[ h ][ i ][ j ], specifies the maximum input bit rate of the j-th CAB when Htid is equal to i. When hrd\_bit\_rate\_scale is not present, its value is inferred to be equal to 1.

**hrd\_cab\_size\_scale**, together with hrd\_cab\_size\_value\_minus1[ h ][ i ][ j ], specifies the CAB size of the j-th CAB when Htid is equal to i and when the CAB operates at the coded atlas access unit level. When hrd\_cab\_size\_scale is not present, its value is inferred to be equal to 1.

**hrd\_fixed\_atlas\_rate\_general\_flag**[ i ] equal to 1 indicates that, when Htid is equal to i, the temporal distance between the HRD output times of consecutive atlas frames in output order is constrained using additional syntax elements. hrd\_fixed\_atlas\_rate\_general\_flag[ i ] equal to 0 indicates that this constraint may not apply. When hrd\_fixed\_atlas\_rate\_general\_flag[ i ] is not present, it is inferred to be equal to 0.

**hrd\_fixed\_atlas\_rate\_within\_cas\_flag**[ i ] equal to 1 indicates that, when Htid is equal to i, the temporal distance between the HRD output times of consecutive atlases in output order is constrained using additional syntax elements. hrd\_fixed\_atlas\_rate\_within\_cas\_flag[ i ] equal to 0 indicates that this constraint may not apply. When hrd\_fixed\_atlas\_rate\_within\_cas\_flag[ i ] is not present (hrd\_fixed\_atlas\_rate\_general\_flag[ i ] is equal to 1), the value of hrd\_fixed\_atlas\_rate\_within\_cas\_flag[ i ] is inferred to be equal to 1.

**hrd\_elemental\_duration\_in\_tc\_minus1**[ i ] plus 1, when present, specifies, when Htid is equal to i, the temporal distance, in clock ticks, between the elemental units that specify the HRD output times of consecutive atlases in output order as specified below. The value of hrd\_elemental\_duration\_in\_tc\_minus1[ i ] shall be in the range of 0 to 2047, inclusive.

When Htid is equal to i and hrd\_fixed\_atlas\_rate\_general\_flag[ i ] is equal to 1 for a CAS containing atlas frame n, the value computed for DabOutputInterval[ n ] shall be equal to ClockTick \* ( hrd\_elemental\_duration\_in\_tc\_minus1[ i ] + 1 ), wherein ClockTick is as specified in Equation E-1 (using the value of ClockTick for the CAS containing atlas frame n) when one of the following conditions is true for the following atlas in output order nextAtlasFrameInOutputOrder that is specified for use in Equation E‑12 :

* the atlas frame nextAtlasFrameInOutputOrder is in the same CAS as the atlas frame n.
* the atlas frame nextAtlasFrameInOutputOrder is in a different CAS and hrd\_fixed\_atlas\_rate\_general\_flag[ i ] is equal to 1 in the CAS containing the atlas frame nextAtlasFrameInOutputOrder, the value of ClockTick is the same for both CASs, and the value of hrd\_elemental\_duration\_in\_tc\_minus1[ i ] is the same for both CASs.

When Htid is equal to i and hrd\_fixed\_atlas\_rate\_within\_cas\_flag[ i ] is equal to 1 for a CAS containing atlas frame n, the value computed for DabOutputInterval[ n ] shall be equal to ClockTick \* ( hrd\_elemental\_duration\_in\_tc\_minus1[ i ] + 1 ), wherein ClockTick is as specified in Equation E-1 (using the value of ClockTick for the CAS containing atlas frame n) when the following atlas frame in output order nextAtlasFrameInOutputOrder that is specified for use in Equation E‑12 is in the same CAS as atlas frame n.

**hrd\_low\_delay\_flag**[ i ] specifies the HRD operational mode, when Htid is equal to i, as specified in Annex E. When not present, the value of hrd\_low\_delay\_flag[ i ] is inferred to be equal to 0.

NOTE 3 – When hrd\_low\_delay\_flag[ i ] is equal to 1, "big atlases" that violate the nominal CAB removal times due to the number of bits used by an access unit are permitted. It is expected, but not required, that such "big atlases" occur only occasionally.

**hrd\_cab\_cnt\_minus1**[ i ] plus 1 specifies the number of alternative CAB specifications in the bitstream of the CAS when Htid is equal to i. The value of hrd\_cab\_cnt\_minus1[ i ] shall be in the range of 0 to 31, inclusive. When not present, the value of hrd\_cab\_cnt\_minus1[ i ] is inferred to be equal to 0.

* + 1. HRD sub-layer parameters semantics

The variable CabCnt is set equal to hrd\_cab\_cnt\_minus1[ subLayerID ].

**hrd\_bit\_rate\_value\_minus1**[ h ][ i ][ j ] (together with hrd\_bit\_rate\_scale) specifies the maximum input bit rate for the j-th CAB, for the i-th sub-layer, when the CAB operates at the access unit level, with h indicating the type of bitstream conformance. hrd\_bit\_rate\_value\_minus1[ h ][ i ][ j ] shall be in the range of 0 to 232 − 2, inclusive. For any j greater than 0 and any particular value of i, hrd\_bit\_rate\_value\_minus1[ h ][ i ][ j ] shall be greater than hrd\_bit\_rate\_value\_minus1[ h ][ i ][ j – 1 ].

The bit rate in bits per second is given by:

BitRate[ h ][ i ][ j ] = ( hrd\_bit\_rate\_value\_minus1[ h ][ i ][ j ] + 1 ) \* 2( 6 + hrd\_bit\_rate\_scale ) (G‑3)

When the hrd\_bit\_rate\_value\_minus1[ h ][ i ][ j ] syntax element is not present, the value of BitRate[ h ][ i ][ j ] is inferred to be equal to MaxAtlasBR, where MaxAtlasBR is specified in Annex A.

**hrd\_cab\_size\_value\_minus1**[ h ][ i ][ j ] is used together with cpb\_size\_scale to specify the j-th CAB size, for the i-th sub-layer, when the CAB operates at the access unit level, with h indicating the type of bitstream conformance. hrd\_cab\_size\_value\_minus1[ h ][ i ][ j ] shall be in the range of 0 to 232 − 2, inclusive. For any j greater than 0 and any particular value of i, hrd\_cab\_size\_value\_minus1[ h ][ i ][ j ] shall be less than or equal to hrd\_cab\_size\_value\_minus1[ h ][ i ][ j – 1 ].

The CAB size in bits is given by:

CabSize[ h ][ i ][ j ] = ( hrd\_cab\_size\_value\_minus1[ h ][ i ][ j ] + 1 ) \* 2( 4 + hrd\_cab\_size\_scale ) (G‑4)

When the hrd\_cab\_size\_value\_minus1[ h ][ i ][ j ] syntax element is not present, the value of CabSize[ h ][ i ][ j ] is inferred to be equal to MaxCABSize, where MaxCABSize is specified in Annex A.

**hrd\_cbr\_flag**[ h ][ i ][ j ] equal to 0 specifies that to decode this CAS bitstream by the HRD using the j-th CAB specification, for the i-th sub-layer and with h indicating the type of bitstream conformance, the hypothetical stream scheduler (HSS) operates in an intermittent bit rate mode. hrd\_cbr\_flag[ h ][ i ][ j ] equal to 1 specifies that the HSS operates in a constant bit rate (CBR) mode. When not present, the value of hrd\_cbr\_flag[ h ][ i ][ j ] is inferred to be equal to 0.

* + 1. Maximum coded video resolution semantics

**mcv\_occupancy\_resolution\_present\_flag** equal to 1 specifies that the syntax elements mcv\_occupancy\_width and mcv\_occupancy\_height are present in the max\_coded\_video\_resolution( ) syntax structure. mcv\_occupancy\_resolution\_present\_flag equal to 0 specifies that the syntax elements mcv\_occupancy\_width and mcv\_occupancy\_height are not present. When mcv\_occupancy\_resolution\_present\_flag is not present, its value is inferred to be equal to 0.

**mcv\_geometry\_resolution\_present\_flag** equal to 1 specifies that the syntax elements mcv\_geometry\_width and mcv\_geometry\_height are present in the max\_coded\_video\_resolution( ) syntax structure. mcv\_geometry\_resolution\_present\_flag equal to 0 specifies that the syntax elements mcv\_geometry\_width and mcv\_geometry\_height are not present. When mcv\_geometry\_resolution\_present\_flag is not present, its value is inferred to be equal to 0.

**mcv\_attribute\_resolution\_present\_flag** equal to 1 specifies that the syntax elements mcv\_attribute\_width and mcv\_attribute\_height are present in max\_coded\_video\_resolution( ) syntax structure. mcv\_attribute\_resolution\_present\_flag equal to 0 specifies that the syntax elements mcv\_attribute\_width and mcv\_attribute\_height are not present. When mcv\_attribute\_resolution\_present\_flag is not present, its value is inferred to be equal to 0.

**mcv\_occupancy\_width** specifies the maximum width value allowed for any frame in the occupancy sub‑bitstream of the current atlas, VUIAtlasID. mcv\_occupancy\_width shall be larger than or equal toDecOccWidth[ i ], as defined in subclause 9.3, for i in the range of 0 to NumDecOccFrames – 1, inclusive. If mcv\_occupancy\_width is not present, then no constraint to the width of the occupancy sub-bitstream frames is specified.

**mcv\_occupancy\_height** specifies the maximum height value allowed for any frame in the occupancy sub‑bitstream of the current atlas, VUIAtlasID. mcv\_occupancy\_height shall be larger than or equal toDecOccHeight[ i ], as defined in subclause 9.3, for i in the range of 0 to NumDecOccFrames – 1, inclusive. If mcv\_occupancy\_height is not present, then no constraint to the height of the occupancy sub-bitstream frames is specified.

**mcv\_geometry\_width** specifies the maximum width value allowed for any frame in the geometry auxiliary and non-auxiliary sub-bitstreams of the current atlas, VUIAtlasID. mcv\_geometry\_width shall be larger than or equal toDecGeoWidth[ geoBitstreamIdx ][ i ] and DecGeoAuxWidth[ j ], as defined in subclause 9.4, with geoBitstreamIdx in the range of 0 to vps\_multiple\_map\_streams\_present\_flag[ VUIAtlasID ] ? vps\_map\_count\_minus1[ j ] : 0, inclusive, i in the range of 0 to NumDecGeoFrames – 1, inclusive, and j in the range of 0 to NumDecGeoAuxFrames – 1, inclusive. If mcv\_geometry\_width is not present, then no constraint to the width of the geometry auxiliary and non-auxiliary sub-bitstream frames is specified.

**mcv\_geometry\_height** specifies the maximum height value allowed for any frame in the geometry auxiliary and non-auxiliary sub-bitstreams of the current atlas, VUIAtlasID. mcv\_geometry\_height shall be larger than or equal toDecGeoHeight[ geoBitstreamIdx ][ i ] and DecGeoAuxHeight[ j ], as defined in subclause 9.4, with geoBitstreamIdx in the range of 0 to vps\_multiple\_map\_streams\_present\_flag[ VUIAtlasID ] ? vps\_map\_count\_minus1[ j ] : 0, inclusive, i in the range of 0 to NumDecGeoFrames – 1, inclusive, and j in the range of 0 to NumDecGeoAuxFrames – 1, inclusive. If mcv\_geometry\_height is not present, then no constraint to the height of the geometry auxiliary and non-auxiliary sub-bitstream frames is specified.

**mcv\_attribute\_width** specifies the maximum width value allowed for any frame in the attribute auxiliary and non-auxiliary sub-bitstreams of the current atlas, VUIAtlasID. mcv\_attribute\_width shall be larger than or equal toDecAttrWidth[ attrIdx ][ partIdx ][ attrBitstreamIdx ][ i ] and DecAttrAuxWidth[ attrIdx ][ partIdx ][ j ], as defined in subclause 9.5, with attrIdx in the range of 0 to ai\_attribute\_count[ VUIAtlasID ] – 1, inclusive, partIdx in the range of 0 to ai\_attribute\_dimension\_partitions\_minus1[ VUIAtlasID ][ attrIdx ], inclusive, attrBitstreamIdx in the range of 0 to vps\_multiple\_map\_streams\_present\_flag[ VUIAtlasID ] ? vps\_map\_count\_minus1[ j ] : 0, inclusive, , i in the range of 0 to NumDecAttrFrames – 1, inclusive, and j in the range of 0 to NumDecAttrAuxFrames – 1, inclusive. If mcv\_attribute\_width is not present, then no constraint to the width of the attribute auxiliary and non-auxiliary sub-bitstream frames is specified.

**mcv\_attribute\_height** specifies the maximum height value allowed for any frame in the attribute auxiliary and non-auxiliary sub-bitstreams of the current atlas, VUIAtlasID. mcv\_attribute\_height shall be larger than or equal toDecAttrHeight[ attrIdx ][ partIdx ][ attrBitstreamIdx ][ i ] and DecAttrAuxHeight[ attrIdx ][ partIdx ][ j ], as defined in subclause 9.5, with attrIdx in the range of 0 to ai\_attribute\_count[ VUIAtlasID ] – 1, inclusive, partIdx in the range of 0 to ai\_attribute\_dimension\_partitions\_minus1[ VUIAtlasID ][ attrIdx ], inclusive, attrBitstreamIdx in the range of 0 to vps\_multiple\_map\_streams\_present\_flag[ VUIAtlasID ] ? vps\_map\_count\_minus1[ j ] : 0, inclusive, , i in the range of 0 to NumDecAttrFrames – 1, inclusive, and j in the range of 0 to NumDecAttrAuxFrames – 1, inclusive. If mcv\_attribute\_height is not present, then no constraint to the height of the attribute auxiliary and non-auxiliary sub-bitstream frames is specified.

* + 1. Coordinate system parameters semantics

The coordinate system parameters assign the orthogonal directions forward, left and up to the axis indices 0..2 corresponding to abstract dimensions x, y, and z.

**csp\_forward\_axis** specifies the axis index of the forward direction. The variable CspForwardAxisIndex is derived as follows:

CspForwardAxisIndex = csp\_forward\_axis (G‑5)

**csp\_delta\_left\_axis\_minus1** plus 1 specifies the difference between the left axis index, CspLeftAxisIndex, and the forward axis index, CspForwardAxisIndex. The variable CspLeftAxisIndex is derived as follows:

CspLeftAxisIndex = (CspForwardAxisIndex + csp\_delta\_left\_axis\_minus1 + 1) % 3 (G‑6)

The variable CspUpAxisIndex is derived from the CspForwardAxisIndex and CspLeftAxisIndex variables:

if( CspForwardAxisIndex != (CspLeftAxisIndex + 1) % 3)

CspUpAxisIndex = (CspLeftAxisIndex + 1) % 3 (G‑7)

else

CspUpAxisIndex = (CspLeftAxisIndex + 2) % 3 (G‑8)

**csp\_forward\_sign** specifies the forward direction in relation to the forward axis. csp\_forward\_sign equal to 1 specifies that the forward direction is equal to the forward axis direction. csp\_forward\_sign equal to 0 specifies that the forward direction is opposite to the forward axis direction.

The variable array CspForwardVector[ i ] is the forward direction as a unit vector with i the axis number and is derived as follows:

for( i = 0; i < 3; i++ )

CspForwardVector[ i ] = (i == CspForwardAxisIndex ) ? (2 \* csp\_forward\_sign – 1 ) : 0 (G‑9)

**csp\_left\_sign** specifies the left direction in relation to the left axis. csp\_left\_sign equal to 1 specifies that the left direction is equal to the left axis direction. csp\_left\_sign equal to 0 specifies that the left direction is opposite to the left axis direction.

The variable array CspLeftVector[ i ] is the left direction as a unit vector with i the axis number and is derived as follows:

for( i = 0; i < 3; i++ )

CspLeftVector[ i ] = ( i == CspLeftAxisIndex ) ? (2 \* csp\_left\_sign – 1 ) : 0 (G‑10)

**csp\_up\_sign** specifies the up direction in relation to the up axis. csp\_up\_sign equal to 1 specifies that the up direction is equal to the up axis direction. csp\_up\_sign equal to 0 specifies that the up direction is opposite to the up axis direction.

The variable array CspUpVector[ i ] is the up direction as a unit vector with i the axis number and is derived as follows:

for( i = 0; i < 3; i++ )

CspUpVector[ i ] = ( i == CspUpAxisIndex ) ? ( 2 \* csp\_up\_sign – 1 ) : 0 (G‑11)

NOTE – The three vectors CspForwardVector, CspLeftVector and CspUpVector can be combined into a coordinate transformation matrix for rendering purposes. The layout of the coordinate transformation matrix is out of scope of this document.

1. (normative)  
   Video-based Point Cloud Coding
   1. Scope

This annex specifies the syntax, semantics, decoding, post-decoding, pre-reconstruction, reconstruction, post-reconstructions, and adaptation processes for video-based point cloud compression that use the syntax, semantics, decoding, post-decoding, pre-reconstruction, reconstruction, post-reconstructions, and adaptation processes specified in clauses 2 through 14, inclusive and Annex A through Annex F, inclusive. This annex also specifies profiles, tiers, and levels for video-based point cloud compression.

* 1. Normative References

The list of normative references in Clause 2 applies.

* 1. Terms and definitions

For the purpose of this annex, the following definitions apply in addition to the definitions in Clause 3. These definitions are either not present in Clause 3 or replace definitions in Clause 3.



point

data specified by single set of *cartesian coordinates* (3.30) and zero or more corresponding *attributes* (3.15)



point cloud

set of *points* (H.3.1) at a particular time instance



point cloud frame

see *point cloud* (H.3.2)



point cloud sequence

sequence of *point cloud frames* (H.3.3)

* 1. Abbreviated terms

The specifications in Clause 4 apply with the following additions.

PBF Patch Border Filtering

* 1. Conventions

The specifications in Clause 5 and its subclauses apply.

* 1. Overall V-PCC characteristics, decoding operations, and post-decoding processes
     1. V3C characteristics

The specifications in subclause 6.1 apply.

* + 1. V3C bitstream characteristics, decoding operations, and post-decoding processes

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

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

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

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

The decoded video frames may require the application of additional transformations, as described in Annex B, before any reconstruction operations. For example, the different components may need to be time-aligned and converted to a nominal video format. The outputs of Annex B are the following videos in the nominal format: OccFramesNF and GeoFramesNF, and, if present, GeoAuxFramesNF, AttrFramesNF, and AttrAuxFramesNF.

With the V3C components in the nominal format, the point cloud content is obtained by invoking several additional steps, including pre-reconstruction, reconstruction, post-reconstruction, and adaptation.

The pre-reconstruction process, which is specified in Clause H.10, may specify additional processes to be applied on the reconstructed video frames, such as certain forms of post-filtering, which may result in improved visual quality or other benefits. Currently a process that allows for the modification of the occupancy frames, OccFramesNF, is specified. In particular, the technique named as the occupancy synthesis process is specified in subclause H.10.2.

In the reconstruction stage, described in Clause H.11, the video components in the nominal format, OccFramesNF and GeoFramesNF, and, if present, GeoAuxFramesNF, AttrFramesNF, and AttrAuxFramesNF, along with the decoded atlas data, are processed to reconstruct the point cloud content.

The reconstructed point cloud content can be further processed by applying informative post-reconstruction methods, as described in Clause H.12. For example, point cloud post-reconstruction in the form of applying a geometry smoothing, attribute smoothing, and/or attribute transfer process, may be performed on the reconstructed point cloud, as specified in subclauses H.12.2, H.12.3, and H.12.8, respectively.

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

* 1. Bitstream format, partitioning, and scanning process

The specifications in Clause 7 and its subclauses apply.

* 1. Syntax and Semantics
     1. Method of specifying syntax in tabular form

The specifications in subclause 8.1 apply.

* + 1. Specification of syntax functions and descriptors

The specifications in subclause 8.2 apply

* + 1. Syntax in tabular form
       1. General

The specifications in subclause 8.3.1 and its subclauses apply.

* + - 1. V3C unit syntax

The specifications in subclause 8.3.2 and its subclauses apply.

* + - 1. Byte alignment syntax

The specifications in subclause 8.3.3 and its subclauses apply.

* + - 1. V3C parameter set syntax

The specifications in subclause 8.3.4 and its subclauses apply.

* + - 1. NAL unit syntax

The specifications in subclause 8.3.4.7 and its subclauses apply.

* + - 1. Raw byte sequence payloads, trailing bits, and byte alignment syntax
         1. Atlas sequence parameter set RBSP syntax

General atlas sequence parameter set RBSP syntax

The specifications in subclause 8.3.6.1.1 apply.

Point local reconstruction information syntax

The specifications in subclause 8.3.6.1.2 apply.

ASPS V-PCC extension syntax

|  |  |
| --- | --- |
| asps\_vpcc\_extension( ) { | **Descriptor** |
| **asps\_vpcc\_remove\_duplicate\_point\_enabled\_flag** | u(1) |
| if( asps\_pixel\_deinterleaving\_enabled\_flag || asps\_plr\_enabled\_flag ) |  |
| **asps\_vpcc\_surface\_thickness\_minus1** | ue(v) |
| } |  |

* + - * 1. Atlas frame parameter set RBSP syntax

The specifications in subclause 8.3.6.2 and its subclauses apply.

* + - * 1. Atlas adaptation parameter set RBSP syntax

General atlas adaptation parameter set RBSP syntax

The specifications in subclause 8.3.6.3 and its subclauses apply.

AAPS V-PCC extension syntax

|  |  |
| --- | --- |
| aaps\_vpcc\_extension ( ) { | **Descriptor** |
| **aaps\_vpcc\_camera\_parameters\_present\_flag** | u(1) |
| if( aaps\_vpcc\_camera\_parameters\_present\_flag ) |  |
| atlas\_camera\_parameters( ) |  |
| } |  |

Atlas camera parameters syntax

|  |  |
| --- | --- |
| atlas\_camera\_parameters( ) { | **Descriptor** |
| **acp\_camera\_model** | u(8) |
| if( acp\_camera\_model == 1) { |  |
| **acp\_scale\_enabled\_flag** | u(1) |
| **acp\_offset\_enabled\_flag** | u(1) |
| **acp\_rotation\_enabled\_flag** | u(1) |
| if( acp\_scale\_enabled\_flag ) |  |
| for( d = 0; d < 3; d++ ) |  |
| **acp\_scale\_on\_axis**[ d ] | u(32) |
| if( acp\_offset\_enabled\_flag ) |  |
| for( d = 0; d < 3; d++ ) |  |
| **acp\_offset\_on\_axis**[ d ] | i(32) |
| if( acp\_rotation\_enabled\_flag ) { |  |
| **acp\_rotation\_qx** | i(16) |
| **acp\_rotation\_qy** | i(16) |
| **acp\_rotation\_qz** | i(16) |
| } |  |
| } |  |
| } |  |

* + - * 1. Supplemental enhancement information RBSP syntax

The specifications in subclause 8.3.6.4 and its subclauses apply.

* + - * 1. Access unit delimiter RBSP syntax

The specifications in subclause 8.3.6.5 and its subclauses apply.

* + - * 1. End of sequence RBSP syntax

The specifications in subclause 8.3.6.6 and its subclauses apply.

* + - * 1. End of bitstream RBSP syntax

The specifications in subclause 8.3.6.7 and its subclauses apply.

* + - * 1. Filler data RBSP syntax

The specifications in subclause 8.3.6.8 and its subclauses apply.

* + - * 1. Atlas tile layer RBSP syntax

The specifications in subclause 8.3.6.9 and its subclauses apply.

* + - * 1. RBSP trailing bit syntax

The specifications in subclause 8.3.6.10 and its subclauses apply.

* + - * 1. Atlas tile header syntax

The specifications in subclause 8.3.6.11 and its subclauses apply.

* + - * 1. Reference list structure syntax

The specifications in subclause 8.3.6.12 and its subclauses apply.

* + - 1. Atlas tile data unit syntax

The specifications in subclause 8.3.6.13 and its subclauses apply.

* + - 1. **Supplemental enhancement information message syntax**

The specifications in subclause 8.3.8 and its subclauses apply.

* + 1. Semantics
       1. General

The specifications in subclause 8.4.1 and its subclauses apply.

* + - 1. V3C unit semantics

The specifications in subclause 8.4.2 and its subclauses apply.

* + - 1. Byte alignment semantics

The specifications in subclause 8.4.3 and its subclauses apply.

* + - 1. V3C parameter set semantics

The specifications in subclause 8.4.4 and its subclauses apply.

* + - 1. NAL unit semantics

The specifications in subclause 8.4.4.7 and its subclauses apply, with the following modification:

nal\_unit\_type shall not be equal to NAL\_CAF.

* + - 1. Raw byte sequence payloads, trailing bits, and byte alignment semantics
         1. Atlas sequence parameter set RBSP semantics

General atlas sequence parameter set RBSP semantics

The specifications in subclause 8.4.6.1.1 and its subclauses apply, with the following modifications:

**asps\_max\_number\_projections\_minus1** plus 1 specifies the maximum value of pdu\_projection\_id[ tileID ][ p ] in the patch\_data\_unit( ) syntax structure for a patch with index p in a tile with tile ID equal to tileID. When asps\_extended\_projection\_enabled\_flag is set to 1, asps\_max\_number\_projections\_minus1 shall be equal to 17. When asps\_max\_number\_projections\_minus1 is not present, it shall be inferred to be equal to 5.

Point local reconstruction information syntax

The specifications in subclause 8.4.6.1.2 apply.

ASPS V-PCC extension semantics

**asps\_vpcc\_remove\_duplicate\_point\_enabled\_flag** equal to 1 indicates that duplicated points shall not be reconstructed for the current atlas, where a duplicated point is a point with the same 2D and 3D geometry coordinates as another point from a lower indexed map associated with the same patch. asps\_vpcc\_remove\_duplicate\_point\_enabled\_flag equal to 0 indicates that all points shall be reconstructed.

NOTE – This process only caters for duplicated points that may be associated with a single patch because of the presence of multiple maps. Duplicated points due to other PROJECTED patches or RAW and EOM patches are not accounted for in this process.

**asps\_vpcc\_surface\_thickness\_minus1** plus 1 specifies the maximum absolute difference between an explicitly coded depth value and interpolated depth value when asps\_pixel\_deinterleaving\_enabled\_flag or asps\_plr\_enabled\_flag is equal to 1.

* + - * 1. Atlas frame parameter set RBSP semantics

The specifications in subclause 8.4.6.2 and its subclauses apply.

* + - * 1. Atlas adaptation parameter set RBSP semantics

General atlas adaptation parameter set RBSP semantics

The specifications in subclause 8.4.6.3 apply.

AAPS V-PCC extension semantics

**aaps\_vpcc\_camera\_parameters\_present\_flag** equal to 1 specifies that camera parameters shall be present in the current atlas adaptation parameter set. aaps\_vpcc\_camera\_parameters\_present\_flag equal to 0 specifies that camera parameters for the current adaptation parameter set shall not be present.

Atlas camera parameters semantics

This subclause describes camera parameters related to scale, offset, and rotation that may be used for reconstruction operations after decoding as described in subclause H.13.

**acp\_camera\_model** indicates the camera model for point cloud frames that are associated with the current adaptation parameter set, as listed in Table H‑1. The value of acp\_camera\_model shall be equal to 0 or 1 in bitstreams conforming to this version of this document. Other values of acp\_camera\_model are reserved for future use by ISO/IEC. Decoders conforming to this version of this document shall ignore reserved values of acp\_camera\_model.

Table H‑1 – Interpretation of acp\_camera\_model

|  |  |
| --- | --- |
| **acp\_camera\_model** | **Name of acp\_camera\_model** |
| 0 | UNSPECIFIED |
| 1 | Orthographic camera model |
| 2 - 255 | RESERVED |

**acp\_scale\_enabled\_flag** equal to 1 indicates that scale parameters for the current camera model are present. acp\_scale\_enabled\_flagequal to 0 indicates that scale parameters for the current camera model are not present. When acp\_scale\_enabled\_flagis not present, it shall be inferred to be equal to 0.

**acp\_offset\_enabled\_flag** equal to 1 indicates that offset parameters for the current camera model are present. acp\_offset\_enabled\_flagequal to 0 indicates that offset parameters for the current camera model are not present. When acp\_offset\_enabled\_flagis not present, it shall be inferred to be equal to 0.

**acp\_rotation\_enabled\_flag** equal to 1 indicates that rotation parameters for the current camera model are present. acp\_rotation\_enabled\_flagequal to 0 indicates that rotation parameters for the current camera model are not present. When acp\_rotation\_enabled\_flagis not present, it shall be inferred to be equal to 0.

**acp\_scale\_on\_axis**[ d ] specifies the value of the scale, Scale[ d ], along the d axis for the current camera model in increments of 2–16. The value of d is in the range of 0 to 2, inclusive, with the values of 0, 1, and 2 corresponding to the X, Y, and Z axis, respectively. The value of acp\_scale\_on\_axis[ d ], shall be in the range of 1 to 232 – 1, inclusive. When acp\_scale\_on\_axis[ d ] is not present, it shall be inferred to be equal to 216. The value of Scale[ d ] is computed as follows:

Scale[ d ] = acp\_scale\_on\_axis[ d ] ÷ 216

A ScaleMatrix can be represented as follows:

**acp\_offset\_on\_axis**[ d ]indicates the value of the offset, Offset[d], along the d axis for the current camera model in increments of 2–16. The value of acp\_offset\_on\_axis[ d ] shall be in the range of −231 to 231 − 1, inclusive, where d is in the range of 0 to 2, inclusive. The values of d equal to 0, 1, and 2 correspond to the X, Y, and Z axis, respectively. When acp\_offset\_on\_axis[ d ] is not present, it shall be inferred to be equal to 0.

Offset[ d ] = acp\_offset\_on\_axis[ d ] ÷ 216

An offset matrix can be represented as follows:

**acp\_rotation\_qx** specifies the x component, qX, for the rotation of the current camera model using the quaternion representation. The value of acp\_rotation\_qx shall be in the range of −214 to 214, inclusive. When acp\_rotation\_qx is not present, its value shall be inferred to be equal to 0. The value of qX is computed as follows:

qX = acp\_rotation\_qx ÷ 214

**acp\_rotation\_qy** specifies the y component, qY, for the rotation of the current camera model using the quaternion representation. The value of acp\_rotation\_qy shall be in the range of −214 to 214, inclusive. When acp\_rotation\_qy is not present, its value shall be inferred to be equal to 0. The value of qY is computed as follows:

qY = acp\_rotation\_qy ÷ 214

**acp\_rotation\_qz** specifies the z component, qZ, for the rotation of the current camera model using the quaternion representation. The value of acp\_rotation\_qz shall be in the range of −214 to 214, inclusive. When acp\_rotation\_qz is not present, its value shall be inferred to be equal to 0. The value of qZ is computed as follows:

qZ = acp\_rotation\_qz ÷ 214

The fourth component, qW, for the rotation of the current camera model using the quaternion representation is calculated as follows:

qW = Sqrt( 1 – ( qX2 + qY2 + qZ2 ) )

NOTE – In the context of this document qW is always positive. If a negative qW is desired, one may signal all three syntax elements, acp\_rotation\_qx, acp\_rotation\_qy, and acp\_rotation\_qz, with an opposite sign, which is equivalent.

A unit quaternion can be represented as a rotation matrix R as follows:

It is a requirement of bitstream conformance that qX2 + qY2 + qZ2 <= 1.

* + - * 1. Supplemental enhancement information RBSP semantics

The specifications in subclause 8.4.6.4 and its subclauses apply.

* + - * 1. Access unit delimiter RBSP semantics

The specifications in subclause 8.4.6.5 and its subclauses apply.

* + - * 1. End of sequence RBSP semantics

The specifications in subclause 8.4.6.6 and its subclauses apply.

* + - * 1. End of bitstream RBSP semantics

The specifications in subclause 8.4.6.7 and its subclauses apply.

* + - * 1. Filler data RBSP semantics

The specifications in subclause 8.4.6.8 and its subclauses apply.

* + - * 1. Atlas tile layer RBSP semantics

The specifications in subclause 8.4.6.9 and its subclauses apply.

* + - * 1. RBSP trailing bit semantics

The specifications in subclause 8.4.6.10 and its subclauses apply.

* + - * 1. Atlas tile header semantics

The specifications in subclause 8.4.6.11 and its subclauses apply.

* + - * 1. Reference list structure semantics

The specifications in subclause 8.4.6.12 and its subclauses apply.

* + - 1. Atlas tile data unit semantics

The specifications in subclause 8.4.6.13 and its subclauses apply.

* + - 1. **Supplemental enhancement information message semantics**

The specifications in subclause 8.4.8 and its subclauses apply.

* 1. Decoding process

The specifications in Clause 9 and its subclauses apply.

* + 1. General decoding process

The specifications in subclause 9.1 apply.

* + 1. Atlas data decoding process
       1. General atlas data decoding process

The specifications in subclause 9.2.1 and its subclauses apply

* + - 1. Decoding process for a coded atlas frame

The specifications in subclause 9.2.2 and its subclauses apply

* + - 1. Atlas NAL unit decoding process

The specifications in subclause 9.2.3 and its subclauses apply

* + - 1. Atlas tile header decoding process

The specifications in subclause 9.2.4 and its subclauses apply

* + - 1. Decoding process for patch data units
         1. General decoding process for patch data units

The specifications in subclause 9.2.5.1 apply with the following modifications and additions:

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

TilePatchAxisV[ tileID ][ p ] is the index of the bi-tangent to the projection plane for the current patch with patch index p. The value of TilePatchAxisV[ tileID ][ p ] shall be in range of 0 to 2, inclusive.

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

TilePatchProjectionFlag[ tileID ][ p ] specifies on which one of two projection planes, indicated by the TilePatchAxisD[ tileID ][ p ] plane, the current patch with patch index p is to be projected.

TilePatch45DegreeMode[ tileID ][ p ] specifies the mode that indicates the axis around which the patch projection rotation is applied for the patch with index p.

If atdu\_patch\_mode[ tileID ][ p ] is equal to I\_INTRA or P\_INTRA, then the process for decoding intra coded patches in subclause H.9.2.5.2 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu\_patch\_mode[ tileID ][ p ] is equal to P\_SKIP, then the process for decoding skip coded patches in subclause H.9.2.5.3 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu\_patch\_mode[ tileID ][ p ] is equal to P\_MERGE, then the process for decoding merge coded patches in H.9.2.5.4 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu\_patch\_mode[ tileID ][ p ] is equal to P\_INTER, then the process for decoding inter coded patches in subclause H.9.2.5.5 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu\_patch\_mode[ tileID ][ p ] is equal to I\_RAW or P\_RAW, then the process for decoding RAW coded patches in subclause H.9.2.5.6 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

If atdu\_patch\_mode[ tileID ][ p ] is equal to I\_EOM or P\_EOM, then the process for decoding EOM coded patches in subclause H.9.2.5.7 is invoked, with p and tileID as the input to that process, and the outputs of that process are used as the outputs of the current process.

These additional variables are initially set as follows:

TilePatchAxisU[ tileID ][ p ] = 0  
 TilePatchAxisV[ tileID ][ p ] = 0  
 TilePatchAxisD[ tileID ][ p ] = 0  
 TilePatchProjectionFlag[ tileID ][ p ] = 0  
 TilePatch45DegreeMode[ tileID ][ p ] = 0

* + - * 1. Decoding process for patch data units coded in intra mode

The specifications in subclause 9.2.5.2 apply with the following modifications and additions:

In addition to the variables derived in subclause 9.2.5.2 the variables TilePatchAxisU[ tileID ][ p ], TilePatchAxisV[ tileID ][ p ], TilePatchAxisD[ tileID ][ p ], TilePatch45DegreeMode[ tileID ][ p ], and TilePatchProjectionFlag[ tileID ][ p ], for patch with index p in the atlas tile with index tileID, are derived as follows:

projectionPlane = pdu\_projection\_id[ tileID ][ p ] (H‑1)

rotationAxisMode = Max( 0, ( projectionPlane − 2)/4 ) (H‑2)

If rotationAxisMode is equal to 0 then the following applies:

TilePatchProjectionFlag[ tileID ][ p ] = Clip3( 0, 1, projectionPlane / 3) (H‑3)

TilePatchAxisD[ tileID ][ p ] = projectionPlane % 3 (H‑4)

Otherwise:

TilePatchProjectionFlag[ tileID ][ p ] = Min( 1, ( ( projectionPlane – 6 ) % 4) / 2 ) (H‑5)

if( rotationAxisMode == 1 )  
 TilePatchAxisD[ tileID ][ p ] = 2 \* ( projectionPlane % 2 )  
 else if( rotationAxisMode == 2 )  
 TilePatchAxisD[ tileID ][ p ] = 2 − ( projectionPlane % 2 ) (H‑6)  
 else  
 TilePatchAxisD[ tileID ][ p ] = 1 − ( projectionPlane % 2 )

TilePatchAxisU[ tileID ][ p ] = ( TilePatchAxisD[ tileID ][ p ] == 2 ) ? 0 : 2 (H‑7)

TilePatchAxisV[ tileID ][ p ] = ( TilePatchAxisD[ tileID ][ p ] == 1 ) ? 0 : 1 (H‑8)

The values of the variables TilePatchAxisU[ tileID ][ p ], TilePatchAxisV[ tileID ][ p ], TilePatchAxisD[ tileID ][ p ], TilePatch45DegreeMode[ tileID ][ p ], and TilePatchProjectionFlag[ tileID ][ p ] that are derived from Equations H‑1 to H‑8 are also presented in tabular form in Table H‑2.

Table H‑2 – Derivation of patch projection variables based on pdu\_projection\_id[ tileID ][ p ] values

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **pdu\_projection\_id[ tileID ][ p ] values** | | | | | | | | | | | | | | | | | |
| **Projection Variables** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** |
| TilePatchAxisU[ tileID ][ p ] | 2 | 2 | 0 | 2 | 2 | 0 | 2 | 0 | 2 | 0 | 0 | 2 | 0 | 2 | 2 | 2 | 2 | 2 |
| TilePatchAxisV[ tileID ][ p ] | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| TilePatchAxisD[ tileID ][ p ] | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 2 | 0 | 2 | 2 | 1 | 2 | 1 | 1 | 0 | 1 | 0 |
| TilePatchProjectionFlag[ tileID ][ p ] | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| TilePatch45DegreeMode[ tileID ][ p ] | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 |

* + - * 1. Decoding process for patch data units coded in skip prediction mode

The specifications in subclause 9.2.5.3 apply with the following modifications and additions:

First, refIdx is set to 0.

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

Then the corresponding patch index, RefPatchIdx, in the atlas frame corresponding to the first entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[ refIdx ] for the current tile is computed as:

RefPatchIdx = PredictorIdx (H‑9)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause H.9.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, refPatchEomPatchCount, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatch45DegreeMode, and refPatchProjectionFlag, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

If refPatchType is equal to PROJECTED then, these additional variables are derived:

TilePatchAxisU[ tileID ][ p ] = refPatchAxisU (H‑10)

TilePatchAxisV[ tileID ][ p ] = refPatchAxisV (H‑11)

TilePatchAxisD[ tileID ][ p ] = refPatchAxisD (H‑12)

TilePatch45DegreeMode[ tileID ][ p ] = refPatch45DegreeMode (H‑13)

TilePatchProjectionFlag[ tileID ][ p ] = refPatchProjectionFlag (H‑14)

* + - * 1. Decoding process for patch data units coded in merge prediction mode

The specifications in subclause 9.2.5.4 apply with the following modifications and additions:

First, the reference atlas frame index, refIdx, is derived as mpdu\_ref\_index[ tileID ][ p ].

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the ( refIdx + 1 )-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[ refIdx ].

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

Then the corresponding patch index, RefPatchIdx, in the reference atlas frame, refAtlasFrm, for the current tile is computed as:

RefPatchIdx = PredictorIdx (H‑15)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause H.9.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, refPatchEomPatchCount, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatch45DegreeMode, and refPatchProjectionFlag, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

It is a requirement of bitstream conformance that refPatchType is equal to PROJECTED.

Then, these additional variables are derived:

TilePatchAxisU[ tileID ][ p ] = refPatchAxisU (H‑16)

TilePatchAxisV[ tileID ][ p ] = refPatchAxisV (H‑17)

TilePatchAxisD[ tileID ][ p ] = refPatchAxisD (H‑18)

TilePatch45DegreeMode[ tileID ][ p ] = refPatch45DegreeMode (H‑19)

TilePatchProjectionFlag[ tileID ][ p ] = refPatchProjectionFlag (H‑20)

* + - * 1. Decoding process for patch data units coded in inter prediction mode

General

The specifications in subclause 9.2.5.5.1 apply with the following modifications and additions:

First, the reference atlas frame index, refIdx, is derived as ipdu\_ref\_index[ tileID ][ p ].

Then the reference atlas frame, refAtlasFrm, is selected as the atlas frame corresponding to the ( refIdx + 1 )-th entry in the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[ refIdx ].

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

Then the corresponding patch index, RefPatchIdx, in the reference atlas frame, refAtlasFrm, for the current tile is computed as:

RefPatchIdx = PredictorIdx + ipdu\_patch\_index[ tileID ][ p ] (H‑21)

and PredictorIdx is set to RefPatchIdx + 1.

The process described in subclause H.9.2.5.5.2 is invoked with the variables refIdx, RefPatchIdx, and tileID as inputs, and the outputs are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, refPatchEomPatchCount, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatch45DegreeMode, and refPatchProjectionFlag, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

It is a requirement of bitstream conformance that refPatchType is equal to PROJECTED.

Then, these additional variables are derived:

TilePatchAxisU[ tileID ][ p ] = refPatchAxisU (H‑22)

TilePatchAxisV[ tileID ][ p ] = refPatchAxisV (H‑23)

TilePatchAxisD[ tileID ][ p ] = refPatchAxisD (H‑24)

TilePatch45DegreeMode[ tileID ][ p ] = refPatch45DegreeMode (H‑25)

TilePatchProjectionFlag[ tileID ][ p ] = refPatchProjectionFlag (H‑26)

Derivation of inter reference patch parameters

General derivation of inter reference patch parameters

The specifications in subclause 9.2.5.5.2.1 apply with the following modifications and additions:

Outputs to this process are the variables refPatchType, refPatch2dPosX, refPatch2dPosY, refPatch2dSizeX, refPatch2dSizeY, refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchRawPoints, refPatchEomPatchCount, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatchProjectionFlag, and refPatch45DegreeMode, the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

First, the patch, refPatch, is determined that has an index equal to refPatchIdx in the tile with tile ID equal to tileID of the ( refIdx + 1 )-th entry of the reference atlas frame list RefAtlasFrmList, RefAtlasFrmList[ refIdx ].

Then, the outputs of this process are derived based on the associated parameters of the patch, refPatch, as follows:

refPatchType = TilePatchType[ tileID ][ refPatchIdx ] (H‑27)

refPatchInAuxVideo = TilePatchInAuxVideo[ tileID ][ refPatchIdx ] (H‑28)

refPatch2dPosX = TilePatch2dPosX[ tileID ][ refPatchIdx ] (H‑29)

refPatch2dPosY = TilePatch2dPosY[ tileID ][ refPatchIdx ] (H‑30)

refPatch2dSizeX = TilePatch2dSizeX[ tileID ][ refPatchIdx ] (H‑31)

refPatch2dSizeY = TilePatch2dSizeY[ tileID ][ refPatchIdx ] (H‑32)

If refPatchType is equal to PROJECTED, then the process in subclause H.9.2.5.5.2.2 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatchProjectionID, refPatch3dOffsetD, refPatch3dRangeD, refPatchOrientationIndex, refPatchLoDScaleX, refPatchLoDScaleY, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatchProjectionFlag, and refPatch45DegreeMode, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

If refPatchType is equal to RAW, then the process in subclause H.9.2.5.5.2.3 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, and refPatchRawPoints.

If refPatchType is equal to EOM, then the process in subclause H.9.2.5.5.2.4 is invoked with refIdx, tileID, and refPatchIdx as inputs. The outputs of the process are the variable refPatchEomPatchCount, and the 1D arrays refPatchAssociatedPatchIndex and refPatchEomPoints.

Reference patches in PROJECTED mode

The specifications in subclause 9.2.5.5.2.2 apply with the following modifications and additions:

Outputs to this process are the variables refPatch3dOffsetU, refPatch3dOffsetV, refPatch3dOffsetD, refPatch3dRangeD, refPatchProjectionID, refPatchOrientationIndex, refPatchLoDScaleX, refPachLoDScaleY, refPatchAxisU, refPatchAxisV, refPatchAxisD, refPatchProjectionFlag, and refPatch45DegreeMode, and, if asps\_plr\_enabled\_flag is equal to 1, the 1D arrays refPatchPlrdLevel, refPatchPlrdPresentFlag, and refPlrdModeMinus1, and the 2D arrays refPatchPlrdPresentBlockFlag and refPatchPlrdBlockModeMinus1.

These additional output variables are derived:

refPatchAxisU = TilePatchAxisU[ tileID ][ refPatchIdx ] (H‑33)

refPatchAxisV = TilePatchAxisV[ tileID ][ refPatchIdx ] (H‑34)

refPatchAxisD = TilePatchAxisD[ tileID ][ refPatchIdx ] (H‑35)

refPatch45DegreeMode = TilePatch45DegreeMode[ tileID ][ refPatchIdx ] (H‑36)

refPatchProjectionFlag = TilePatchProjectionFlag[ tileID ][ refPatchIdx ] (H‑37)

Reference Patches in RAW mode

The specifications in subclause 9.2.5.5.2.3 apply

Reference Patches in EOM mode

The specifications in subclause 9.2.5.5.2.4 apply

* + - * 1. Decoding process for patch data units coded in RAW mode

The specifications in subclause 9.2.5.6 apply

* + - * 1. Decoding process for patch data units coded in EOM mode

The specifications in subclause 9.2.5.7 apply

* + - 1. Decoding process of the block to patch map

The specifications in subclause 9.2.6 and its subclauses apply

* + - 1. Conversion of tile level information to atlas level information
         1. General

The specifications in subclause 9.2.7.1 apply

* + - * 1. Conversion of tile level blockToPatch information to atlas level blockToPatch information

The specifications in subclause 9.2.7.2 apply.

* + - * 1. Conversion of tile level patch information to atlas level patch information

General

The specifications in subclause 9.2.7.3.1 apply.

**Process of copying common patch parameters from a tile to an atlas representation**

The specifications in subclause 9.2.7.3.2 apply.

Process of copying application specific patch parameters from a tile to an atlas representation

The specifications in subclause 9.2.7.3.3 apply with the following modifications and additions:

The process ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch ) is defined as follows:

ApplicationTilePatchParamsToAtlas( atlasPatchIdx, tileID, p, offsetPatch ) {  
 blockCnt = BlockCnt( TilePatch2dSizeX[ tileID ][ p ], TilePatch2dSizeY[ tileID ][ p ] )  
 if( AtlasPatchType[ atlasPatchIdx ] == PROJECTED ) {  
 AtlasTotalNumProjPatches += 1  
 AtlasPatchProjectionFlag[ atlasPatchIdx ] = TilePatchProjectionFlag[ tileID ][ p ]  
 AtlasPatchAxisU[ atlasPatchIdx ] = TilePatchAxisU[ tileID ][ p ]  
 AtlasPatchAxisV[ atlasPatchIdx ] = TilePatchAxisV[ tileID ][ p ]  
 AtlasPatchAxisD[ atlasPatchIdx ] = TilePatchAxisD[ tileID ][ p ]  
 AtlasPatch45DegreeMode[ atlasPatchIdx ] = TilePatch45DegreeMode[ tileID ][ p ]  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 AtlasPlrdLevel[ atlasPatchIdx ][ m ] = TilePatchPlrdLevel[ tileID ][ p ][ m ]  
 if( AtlasPlrdLevel[ atlasPatchIdx ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 AtlasPlrdPresentBlockFlag[ atlasPatchIdx ][ m ][ j ] =  
 TilePatchPlrdPresentBlockFlag[ tileID ][ p ][ m ][ j ]  
 if( AtlasPlrdPresentBlockFlag[ atlasPatchIdx ][ m ][ j ] == 1 )  
 AtlasPlrdBlockModeMinus1[ atlasPatchIdx ][ m ][ j ] =  
 TilePatchPlrdBlockModeMinus1[ tileID ][ p ][ m ][ j ]  
 }  
 } else {  
 AtlasPlrdPresentFlag[ atlasPatchIdx ][ m ] =  
 TilePatchPlrdPresentFlag[ tileID ][ p ][ m ]  
 if( AtlasPlrdPresentFlag[ atlasPatchIdx ][ m ] == 1 )  
 AtlasPlrdModeMinus1[ atlasPatchIdx ][ m ] =  
 TilePatchPlrdModeMinus1[ tileID ][ p ][ m ]  
 }  
 }  
 }  
 }  
 if( AtlasPatchType[ atlasPatchIdx ] == EOM ) {  
 AtlasTotalNumEOMPatches += 1  
 for( i = 0; i < AtlasEpduPatchCount[ atlasPatchIdx ]; i++ ) {  
 NumEOMPoints += TilePatchEomPoints[ tileID ][ p ][ i ]  
 AtlasEpduPoints[ atlasPatchIdx ][ i ] = TilePatchEomPoints[ tileID ][ p ][ i ]  
 AtlasEpduAssociatedPatchIndex[ atlasPatchIdx ][ i ] =  
 TilePatchAssociatedPatchIndex[ tileID ][ p ][ i ] + offsetPatch  
 }  
 }  
 if( AtlasPatchType[ atlasPatchIdx ] == RAW ) {  
 AtlasTotalNumRawPatches += 1  
 NumRawPoints += TilePatchRawPoints[ tileID ][ p ]  
 }  
}

* + 1. Occupancy video decoding process

The specifications in subclause 9.3 apply.

* + 1. Geometry video decoding process

The specifications in subclause 9.4 apply.

* + 1. Attribute video decoding process

The specifications in subclause 9.5 apply.

* + 1. Sub-bitstream extraction process

The specifications in subclause 9.6 apply.

* 1. Pre-reconstruction process
     1. General

The pre-reconstruction process depends on the profiles defined in subclause H.15.

The processes described in this subclause are invoked for decoded occupancy frames at nominal resolution, and syntax elements associated with the same atlas ID, identified by a variable OccSynAtlasID.

The pre-reconstruction process, for the current point cloud frame with a composition time index compTimeIdx, takes as inputs the syntax elements and upper-case variables from subclauses H.8, H.9, and Annex B.

If a bitstream contains an occupancy synthesis SEI message with a value of os\_method\_type[ k ] larger than zero at time instance compTimeIdx, and the point cloud reconstruction system has selected to use this k-th occupancy synthesis instance, subclause H.10.2.2 is invoked.

* + 1. Occupancy synthesis process
       1. General

If os\_method\_type[ k ] is equal to 1, which indicates the patch border filtering method, then subclause H.10.2.2 is invoked with OccFramesNF[ compTimeIdx ][ 0 ] and GeoFramesNF[ 0 ][ compTimeIdx ][ 0 ] as inputs and the modified array OccFramesNF[ compTimeIdx ][ 0 ] as output.

NOTE – If subclause H.10.2.2 is not invoked, OccFramesNF[ compTimeIdx ][ 0 ] is not modified.

* + - 1. Patch Border Filtering (PBF)
         1. General

Inputs to this process are:

* a variable k, indicating the selected occupancy synthesis filter instance,
* a 2D array oFrame, of size asps\_frame\_height× asps\_frame\_width, and
* a 2D array gFrame, of size asps\_frame\_height× asps\_frame\_width.

Output of this process is the modified array oFrame.

NOTE – This process may not be effective when the occupancy video is coded at the nominal resolution.

The following applies:

* Let the variable boxExtension be set to 8.
* If asps\_eom\_patch\_enabled\_flag is equal to 1, the array oFrame is returned without modification.
* Otherwise ( asps\_eom\_patch\_enabled\_flag is equal to 0 ), the following process is performed:
* First, the following arrays are defined:
* borderPointCnt, a 1D array of size AtlasTotalNumPatches,
* neighPatchCnt, a 1D array of size AtlasTotalNumPatches,
* neighbouringPatches, a 2D array of size AtlasTotalNumPatches × AtlasTotalNumPatches,
* borderPoints, a 3D array of size AtlasTotalNumPatches × AspsFrameSize × 3,
* bBox, a 3D array of size AtlasTotalNumPatches × 2 × 3, and
* neighbourDepth, a 2D array of size asps\_frame\_height × asps\_frame\_width.

NOTE – The allocated sizes of neighbouringPatches and borderPoints use the maximum possible values according to this document. Different, more efficient implementations may be used.

* For pIdx = 0 .. AtlasTotalNumPatches − 1, the following applies:
* subclause H.10.2.2.2 is invoked with pIdx, oFrame, and gFrame, as inputs, and borderPointCnt[ pIdx ], borderPoints[ pIdx ], and bBox[ pIdx ] as outputs.
* For pIdx = 0 .. AtlasTotalNumPatches − 1, the following applies:
* subclause H.10.2.2.3 is invoked with pIdx and bBox as inputs, and neighPatchCnt[ pIdx ] and neighbouringPatches[ pIdx ] as outputs.
* subclause H.10.2.2.4 is invoked with variables PbfThreshold[ k ], boxExtension, and borderPointCnt, and arrays borderPoints, bBox, neighPatchCnt, and neighbouringPatches, as inputs, and the array neighbourDepth as output.
* subclause H.10.2.2.5 is invoked with variables PbfPasses[ k ], PbfSizeX[ k ], and PbfSizeY[ k ], and arrays oFrame, gFrame, and neighbourDepth as inputs, and the modified array oFrame as output.
  + - * 1. Bounding box and border point determination

Inputs to this process are:

* a variable pIdx, indicating the patch index,
* a 2D array oFrame, of size asps\_frame\_height × asps\_frame\_width, corresponding to the occupancy frame in the nominal format, and
* a 2D array gFrame, of size asps\_frame\_height × asps\_frame\_width, corresponding to the geometry frame in the nominal format.

Outputs of this process are:

* a variable borderPointCnt, indicating the number of border points,
* a 2D array borderPoints, of size AspsFrameSize × 3, and
* a 2D array bBox, of size 2 × 3.

Let borderPointCnt and bBox be initialized as follows:

borderPointCnt = 0.  
 for( k = 0; k < 3; k++ ) {  
 bBox[ 0 ][ k ] = 0xFFFF  
 bBox[ 1 ][ k ] = 0  
 }

Let yStart, yEnd, xStart, and xEnd be computed as follows:

yStart = AtlasPatch2dPosY[ pIdx ]  
 yEnd = AtlasPatch2dPosY[ pIdx ] + AtlasPatch2dSizeY[ pIdx ]  
 xStart = AtlasPatch2dPosX[ pIdx ]  
 xEnd = AtlasPatch2dPosX[ pIdx ] + AtlasPatch2dSizeX[ pIdx ]

Then the following applies:

for( y = yStart; y < yEnd; y++ ) {  
 yBlock = y / PatchPackingBlockSize  
 for( x = xStart; x < xEnd; x++ ) {  
 xBlock = x / PatchPackingBlockSize  
 isBoundaryPoint = 0

If oFrame[ y ][ x ] is not equal to zero and AtlasBlockToPatchMap[ yBlock ][ xBlock ] is equal to pIdx, then subclause H.12.5 is invoked with array oFrame, and variables y and x as inputs. The output is the modified variable isBoundaryPoint.

If isBoundaryPoint is equal to one, then the following applies:

The 3D coordinates corresponding to the point with atlas coordinates ( x, y ) are derived by invoking subclause H.11.5 with variables pIdx, gFrame[ y ][ x ], x, and y as inputs. The output is an array, pos3D, containing the 3D coordinates of the point.

The 3D coordinates corresponding to the point are added to the 3D boundary points array as follows:

for( k = 0; k < 3; k++ )  
 borderPoints[ borderPointCnt ][ k ] = pos3D[ k ]  
 borderPointCnt++

The 3D coordinates corresponding to the point are added to the border bounding box array as follows:

for( k = 0; k < 3; k++ ) {  
 bBox[ 0 ][ k ] = Min( pos3D[ k ], bBox[ 0 ][ k ] )   
 bBox[ 1 ][ k ] = Max( pos3D[ k ], bBox[ 1 ][ k ] )  
 }  
 }  
 }

* + - * 1. Neighbouring patches determination

Inputs to this process are:

* a variable pIdx, indicating the patch index, and
* a 3D array bBox, of size AtlasTotalNumPatches × 2 × 3.

Outputs of this process are:

* a variable neighPatchCnt, indicating the number of neighbouring patches for the patch with index pIdx, and
* a 1D array neighbouringPatches, of size AtlasTotalNumPatches.

Let neighPatchCnt be initialized as follows:

neighPatchCnt = 0

Then the following applies:

for( p = 0; p < AtlasTotalNumPatches; p++ ) {  
 isIntersect = 0  
 if( p != pIdx) {  
 isIntersect =  
 bBox[ pIdx ][ 1 ][ 0 ] >= bBox[ p ][ 0 ][ 0 ] &&  
 bBox[ pIdx ][ 0 ][ 0 ] <= bBox[ p ][ 1 ][ 0 ] &&  
 bBox[ pIdx ][ 1 ][ 1 ] >= bBox[ p ][ 0 ][ 1 ] &&  
 bBox[ pIdx ][ 0 ][ 1 ] <= bBox[ p ][ 1 ][ 1 ] &&  
 bBox[ pIdx ][ 1 ][ 2 ] >= bBox[ p ][ 0 ][ 2 ] &&  
 bBox[ pIdx ][ 0 ][ 2 ] <= bBox[ p ][ 1 ][ 2 ]  
 if( isIntersect ) {  
 neighbouringPatches[ neighPatchCnt ] = p  
 neighPatchCnt++  
 }  
 }  
 }

* + - * 1. Neighbouring depth determination

Inputs to this process are:

* a variable depthThreshold, indicating the threshold for determining the neighbouring depth value,
* a variable boxExtension, indicating by how much the bounding box should be extended,
* a 1D array borderPointCnt, of size AtlasTotalNumPatches,
* a 3D array borderPoints, of size AtlasTotalNumPatches× AspsFrameSize × 3,
* a 3D array bBox, of size AtlasTotalNumPatches × 2 × 3,
* a 1D array neighPatchCnt, of size AtlasTotalNumPatches, and
* a 1D array neighbouringPatches, of size AtlasTotalNumPatches.

Output of this process is:

* a 2D array neighbourDepth, of size asps\_frame\_height × asps\_frame\_width.

The neighbouring depth values are initialized as follows:

for ( y = 0; y < asps\_frame\_height, y++ )   
 for ( x = 0; x < asps\_frame\_width, x++ )  
 neighbourDepth[ y ][ x ] = −1

Then the values are determined as follows

for( pIdx = 0; pIdx < AtlasTotalNumPatches, pIdx++ ) {  
 for( q = 0; q < neighPatchCnt[ pIdx ], q++ ) {  
 nIdx = neighbouringPatches[ q ]  
 for ( b = 0; b < borderPointCnt[ q ], b++ ) {  
 if( !( ( borderPoints[ nIdx ][ b ][ 0 ] < bBox[ pIdx ][ 0 ][ 0 ] – boxExtension ) ||  
 ( borderPoints[ nIdx ][ b ][ 0 ] > bBox[ pIdx ][ 1 ][ 0 ] + boxExtension ) ||  
 ( borderPoints[ nIdx ][ b ][ 1 ] < bBox[ pIdx ][ 0 ][ 1 ] – boxExtension ) ||  
 ( borderPoints[ nIdx ][ b ][ 1 ] > bBox[ pIdx ][ 1 ][ 1 ] + boxExtension ) ||  
 ( borderPoints[ nIdx ][ b ][ 2 ] < bBox[ pIdx ][ 0 ][ 2 ] – boxExtension ) ||  
 ( borderPoints[ nIdx ][ b ][ 2 ] > bBox[ pIdx ][ 1 ][ 2 ] + boxExtension ) ) ) {

The atlas coordinates ( x, y ) and variable borderPointDepth are derived by invoking subclause H.10.2.2.6 with variable pIdx and array borderPoints[ pIdx ][ b ] as inputs and the variables x, y, and borderPointDepth as outputs.

absDistance = Abs( borderPointDepth − gFrame[ y ][ x ] )  
 if( absDistance <= depthThreshold ) {  
 if( absDistance < Abs( neighbourDepth[ y ][ x ] − gFrame[ y ][ x ] ) ) {  
 neighbourDepth[ y ][ x ] = borderPointDepth  
 }  
 }  
 }  
 }  
 }  
 }

* + - * 1. Occupancy filtering

Inputs to this process are:

* a variable numPasses, indicating the number of passes
* a variable filterSizeX, indicating the size of the filter in the x-dimension,
* a variable filterSizeY, indicating the size of the filter in the y-dimension,
* a 2D array oFrame, of size asps\_frame\_height× asps\_frame\_width,
* a 2D array gFrame, of size asps\_frame\_height× asps\_frame\_width, and
* a 2D array neighbourDepth, of size asps\_frame\_height × asps\_frame\_width.

Output of this process is:

* the modified array oFrame.

Let the 2D array oPbfFrame, of size asps\_frame\_height × asps\_frame\_width, be used as a temporary array in this process.

The occupancy is pruned as follows:

for ( pIdx = 0; pIdx < AtlasTotalNumPatches, pIdx++ ) {

for ( y = 0; y < asps\_frame\_height, y++ )   
 for ( x = 0; x < asps\_frame\_width, x++ )  
 oPbfFrame[ y ][ x ] = 0

yStart = AtlasPatch2dPosY[ pIdx ]  
 yEnd = AtlasPatch2dPosY[ pIdx ] + AtlasPatch2dSizeY[ pIdx ]  
 xStart = AtlasPatch2dPosX[ pIdx ]

xEnd = AtlasPatch2dPosX[ pIdx ] + AtlasPatch2dSizeX[ pIdx ]

for ( iter = 0; iter < numPasses; iter++ ) {  
 if( iter & 0x1 ) {

Let the pSrc and pDst arrays correspond to the arrays oFrame and oPbfFrame, respectively.

} else {

Let the pSrc and pDst arrays correspond to the arrays oPbfFrame and oFrame, respectively.

}  
 for( y = yStart; y < yEnd; y++ ) {  
 subBlkY = y / PatchPackingBlockSize  
 for( x = xStart; x < xEnd; x++ ) {  
 subBlkX = x / PatchPackingBlockSize  
 if( AtlasBlockToPatchMap[ subBlkY ][ subBlkX ] == pIdx ) {  
 if( pSrc[ y ][ x ] == 0 ) {  
 pDst[ y ][ x ] = 0  
 } else {  
 countNeighbour = ( x – 1 > 0 ) ? pSrc[ y ][ x − 1 ] : 0  
 countNeighbour += ( x + 1 < asps\_frame\_width ) ? pSrc[ y ][ x + 1 ] : 0  
 countNeighbour += ( y – 1 > 0 ) ? pSrc[ y − 1 ][ x ] : 0  
 countNeighbour += ( y + 1 < asps\_frame\_height ) ? pSrc[ y + 1 ][ x ] : 0  
 if( countNeighbour == 0 || countNeighbour == 4 ) {  
 pDst[ y ][ x ] = 0  
 } else {

Subclause H.10.2.2.7 is invoked with variables filterSizeX, filterSizeY, x, and y, and the arrays pSrc, gFrame, and neighbourDepth[ pIdx ] as inputs and pDst[ y ][ x ] as output.

}

}  
 }  
 }  
 }  
 }  
 if( numPasses & 0x1 ) {  
 for( y = yStart; y < yEnd; y++ ) {  
 subBlkY = y / PatchPackingBlockSize  
 for( x = xStart; x < xEnd; x++ ) {  
 subBlkX = x / PatchPackingBlockSize  
 if( AtlasBlockToPatchMap[ subBlkY ][ subBlkX ] == pIdx )  
 oFrame[ y ][ x ] = oPbfFrame[ y ][ x ]  
 }  
 }  
 }  
 }

* + - * 1. Conversion from a 3D point position to atlas coordinates and depth

Inputs to this process are:

* a variable pIdx, indicating the patch index and
* a 1D array pos3D, of size 3, indicating the 3D coordinates of a point.

Outputs of this process are:

* a variable x, indicating the x atlas coordinate,
* a variable y, indicating the y atlas coordinate, and
* a variable depthValue, indicating the depth of a point.

The following applies:

* The 3D coordinates are converted to a local patch coordinate pair ( u, v ) and the variable depthValue is computed as follows:

u = (pos3D[ AtlasPatchAxisU[ pIdx ]] – AtlasPatch3dOffsetU[ pIdx ]) (H‑38)

v = (pos3D[ AtlasPatchAxisV[ pIdx ]] – AtlasPatch3dOffsetV[ pIdx ]) (H‑39)

projScale = (1 – 2\* AtlasPatchProjectionFlag[ pIdx ]) (H‑40)

depthValue = (H‑41)  
 Max( 0, (pos3D[ AtlasPatchAxisD[ pIdx ] ] − AtlasPatch3dOffsetD[ pIdx ]) \* projScale)

* A variable oIdx is set to AtlasPatchOrientationIndex[ pIdx ].
* Variables posX, posY, sizeX, sizeY, lodX, and lodY are assigned as follows:

posX = AtlasPatch2dPosX[ pIdx ]  
 posY = AtlasPatch2dPosY[ pIdx ]  
 sizeX = AtlasPatch2dSizeX[ pIdx ]  
 sizeY = AtlasPatch2dSizeY[ pIdx ]  
 lodX = AtlasPatchLoDScaleX[ pIdx ]  
 lodY = AtlasPatchLoDScaleY[ pIdx ]

* The local patch coordinate pair ( u, v ) are converted to atlas coordinates (x, y) as follows:

(H‑42)

where **R**o and **R**s are specified in Table 11.

* + - * 1. Occupancy pruning

Inputs to this process are:

* a variable filterSizeX, indicating the size of the filter in the x-dimension,
* a variable filterSizeY, indicating the size of the filter in the y-dimension,
* a variable x, specifying the x coordinate of a point on the atlas frame,
* a variable y, specifying the y coordinate of a point on the atlas frame,
* a 2D array oFrame, of size asps\_frame\_height × asps\_frame\_width, corresponding to the occupancy frame in the nominal format, and
* a 2D array gFrame, of size asps\_frame\_height × asps\_frame\_width, corresponding to the geometry frame in the nominal format.
* a 2D array neighbourDepth, of size AtlasTotalNumPatches × asps\_frame\_height × asps\_frame\_width.

Output of this process is:

* a variable oPbfFrameValue, indicating the occupancy value modified by the PBF process

The following steps apply:

* The variable boundaryOrientation is derived as follows:

top = Max( 0, y − 1 )  
 bottom = Min( asps\_frame\_height – 1, y + 1 )  
 left = Max( 0, x − 1 )  
 right = Min( asps\_frame\_width– 1, x + 1 )  
 boundaryOrientation = orientArr[  
 (oFrame[ top ][ left ] << 7) | (oFrame[ top ][ x ] << 6) |  
 (oFrame[ top ][ right ] << 5) | (oFrame[ y ][ left ] << 4) |  
 (oFrame[ y ][ right ] << 3) | (oFrame[ bottom ][ left ] << 2) |  
 (oFrame[ bottom ][ x ] << 1) | (oFrame[ bottom ][ right ]) ]

where the array orientArr is defined as follows:

orientArr[ 256 ] = {

0, 0, 6, 0, 0, 0, 0, 6, 4, 0, 0, 5, 0, 0, 0, 5,

0, 0, 0, 0, 0, 0, 7, 7, 0, 0, 0, 0, 0, 0, 0, 6,

0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 5, 0, 0, 0, 5,

0, 0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 5,

2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 0, 4, 3, 0, 0, 5,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 7, 7, 0, 0, 0, 0, 0, 0, 0, 7,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 1, 0, 0, 1, 0, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0,

2, 0, 0, 0, 0, 0, 0, 0, 3, 3, 0, 3, 0, 0, 0, 4,

1, 0, 0, 0, 1, 0, 1, 0, 2, 3, 0, 0, 1, 2, 0, 0 }

* The arrays vX and vY of size 2 are derived as follows:

vX[ 0 ] = vectorOrientation[ boundaryOrientation ][ 0 ]  
 vX[ 1 ] = vectorOrientation[ boundaryOrientation ][ 1 ]  
 vY[ 0 ] = vectorOrientation[ ( boundaryOrientation + 2 ) % 8 ][ 0 ]  
 vY[ 1 ] = vectorOrientation[ ( boundaryOrientation + 2 ) % 8 ][ 1 ]

where vectorOrientation is defined as follows:

vectorOrientation[ 8 ][ 2 ] =  
 { {1, 0}, {1, 1}, {0, 1}, {−1, 1}, {−1, 0}, {−1, −1}, {0, −1}, {1, −1} }

* Then the following applies:

sumP = 0  
 sumE = 0  
 windowCount = 0  
 for( xW = −filterSizeX; xW <= filterSizeX; xW++ ) {  
 for( yW = −filterSizeY; yW <= filterSizeY; yW++ ) {  
 yPos = Clip3( 0, asps\_frame\_height − 1, y + vX[ 1 ] \* xW + vY[ 1 ] \* yW )  
 xPos = Clip3( 0, asps\_frame\_width − 1, x + vX[ 0 ] \* xW + vY[ 0 ] \* yW )  
 yPosG = Clip3( 0, asps\_frame\_height − 1, y − vX[ 1] )  
 xPosG = Clip3( 0, asps\_frame\_width − 1, x − vX[ 0 ] )  
 if( neighbourDepth[ yPos ][ xPos ] > 0) {  
 p = ( xW \* vX[ 0 ] + yW \* vY[ 0 ] )2  
 p += ( xW \* vX[ 1 ] + yW \* vY[ 1 ] )2  
 p += ( neighbourDepth[ yPos][ xPos ] − gFrame[ y ][ x ] )2  
 sumP += Sqrt( p )  
 e = ( ( xW + 1 ) \* vX[ 0 ] + yW \* vY[ 0 ] )2  
 e += ( ( xW + 1 ) \* vX[ 1 ] + yW \* vY[ 1 ] )2  
 e += ( neighbourDepth[ yPos ][ xPos ] − gFrame[ yPosG ][ xPosG ] )2  
 sumE += Sqrt( e )  
 windowCount++  
 }  
 }  
 }  
 oPbfFrameValue = ( windowCount == 0 ) || ( sumE >= sumP )

* 1. Reconstruction process
     1. General

The reconstruction process depends on the profiles defined in subclause H.15.

The processes described in this subclause are invoked for decoded atlas frames, decoded video frames, and syntax elements associated with the same atlas ID, identified by a variable RecAtlasID.

The reconstruction process for the current point cloud frame with a composition time index compTimeIdx, takes as inputs the syntax elements and upper-case variables from subclauses H.8 and H.9, and the following arrays:

* a 5D array geoFramesNF[ mapIdx ][ compTimeIdx ][ 0 ][ y ][ x ] specifying the decoded geometry frames in the nominal format, where mapIdx is in the range of 0 to asps\_map\_count\_minus1, inclusive, y is in the range of 0 to asps\_frame\_height – 1, inclusive, and x is in the range of 0 to asps\_frame\_width – 1, inclusive,
* a 4D array occFramesNF[ compTimeIdx ][ 0 ][ y ][ x ] specifying the decoded occupancy frames in the nominal format, where y is in the range of 0 to asps\_frame\_height – 1, inclusive, and x is in the range of 0 to asps\_frame\_width – 1, inclusive,
* when ai\_attribute\_count[ RecAtlasID ] is greater than 0, a 6D array attrFramesNF[ attrIdx ][ mapIdx ][ compTimeIdx ][ compIdx ][ y ][ x ] specifying the decoded attribute frames in the nominal format, where attrIdx is in the range of 0 to ai\_attribute\_count[ RecAtlasID ] − 1, inclusive, mapIdx is in the range of 0 to asps\_map\_count\_minus1, inclusive, compIdx is in the range of 0 to ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ], inclusive, y is in the range of 0 to asps\_frame\_height – 1, inclusive, and x is in the range of 0 to asps\_frame\_width – 1, inclusive.
* when ( asps\_auxiliary\_video\_enabled\_flag && asps\_raw\_patch\_enabled\_flag ), a 4D array geoAuxFramesNF[ compTimeIdx ][ 0 ][ y ][ x ] specifying decoded geometry auxiliary frames in the nominal format, where y is in the range of 0 to AuxVideoHeightNF – 1, inclusive, and x is in the range of 0 to AuxVideoWidthNF – 1, inclusive, and
* when ( asps\_auxiliary\_video\_enabled\_flag && ( ai\_attribute\_count[ RecAtlasID ] > 0 ) ), a 5D array attrAuxFramesNF[ attrIdx ][ compTimeIdx ][ compIdx ][ y ][ x ] specifying the decoded attribute auxiliary frames in the nominal format, where attrIdx is in the range of 0 to ai\_attribute\_count[ RecAtlasID ] − 1, inclusive, compIdx is in the range of 0 to ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ], inclusive, y is in the range of 0 to AuxVideoHeightNF – 1, inclusive, and x is in the range of 0 to AuxVideoWidthNF – 1, inclusive.

Output of this process is a sequence of reconstructed point cloud frames before the post-reconstruction step. The following information is output for each reconstructed point cloud frame:

* a variable pointCnt, specifying the number of points in the reconstructed point cloud frame,
* a 1D array pointToPatch, of size pointCnt, specifying the patch index corresponding to each reconstructed point,
* a 2D array pointToPixel, of size pointCnt × 2, specifying the atlas coordinates corresponding to each reconstructed point,
* a 2D array recPcGeo, of size pointCnt × 3, specifying the list of coordinates corresponding to each reconstructed point, and,
* when ai\_attribute\_count[ RecAtlasID ] is greater than 0, a 3D array recPcAttr, specifying the attributes associated with the points in the reconstructed point cloud frame, where the three dimensions correspond to the index of the reconstructed point, the attribute index, and the attribute dimension, respectively.

The arrays gFrame[ mapIdx ][ y ][ x ], aFrame[ mapIdx ][ attrIdx ][ compIdx ][ y ][ x ], and oFrame[ y ][ x ] are derived as follows:

for( j = 0; j < asps\_frame\_height; j++ )  
 for( i = 0; i < asps\_frame\_width; i++ ) {  
 oFrame[ j ][ i ] = occFramesNF[ compTimeIdx ][ 0 ][ j ][ i ]  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 gFrame[ m ][ j ][ i ] = geoFramesNF[ m ][ compTimeIdx ][ 0 ][ j ][ i ]  
 for( a = 0; a < ai\_attribute\_count[ RecAtlasID ]; a++ )  
 for( c = 0; c < ai\_attribute\_dimension\_minus1[ RecAtlasID ][ a ] + 1; c++ )  
 aFrame[ m ][ a ][ c ][ j ][ i ] = attrFramesNF[ a ][ m ][ compTimeIdx ][ c ][ j ][ i ]  
 }  
 }

When asps\_auxiliary\_video\_enabled\_flag is equal to 1, the arrays gAuxFrame[ y ][ x ] and aAuxFrame[ attrIdx ][ compIdx ][ y ][ x ] are derived as follows:

if( asps\_raw\_patch\_enabled\_flag )  
 for( j = 0; j < AuxVideoHeightNF; j++ )  
 for( i = 0; i < AuxVideoWidthNF; i++ )  
 gAuxFrame[ j ][ i ] = geoAuxFramesNF[ compTimeIdx ][ 0 ][ j ][ i ]

for( a = 0; a < ai\_attribute\_count[ RecAtlasID ]; a++ )  
 for( c = 0; c < ai\_attribute\_dimension\_minus1[ RecAtlasID ][ a ] + 1; c++ )  
 for( j = 0; j < AuxVideoHeightNF; j++ )  
 for( i = 0; i < AuxVideoWidthNF; i++ )  
 aAuxFrame[ a ][ c ][ j ][ i ] = attrAuxFramesNF[ a ][ compTimeIdx ][ c ][ j ][ i ]

When asps\_plr\_enabled\_flag is equal to 1 the following applies:

* Arrays plrMode[ p ][ mapIdx ][ bIdxY ][ bIdxX ], plrInterpolateFlag[ mapIdx ][ bIdxY ][ bIdxX ], plrFillingFlag[ mapIdx ][ bIdxY ][ bIdxX ], plrMinimumDepth[ mapIdx ][ bIdxY ][ bIdxX ], and plrNeighbour[ mapIdx ][ bIdxY ][ bIdxX ] are derived for each block with coordinates (bIdxX, bIdxY ), for the map with index mapIdx of the patch with index p, as follows:
* For m = 0.. asps\_map\_count\_minus1 and p = 0.. AtlasTotalNumPatches – 1 the following applies:
* Variables blockCnt and numCols are derived as follows:

blockCnt = BlockCnt( AtlasPatch2dSizeX[ p ], AtlasPatch2dSizeY[ p ] )  
 numCols =  
 (AtlasPatch2dSizeX[ p ] + PatchPackingBlockSize – 1 ) / PatchPackingBlockSize

* If plri\_map\_present\_flag[ m ] is equal to 1, the following applies:

for ( n = 0; n < blockCnt; n++ ) {  
 bIdxY = n / numCols  
 bIdxX = n % numCols

if( AtlasPlrdLevel[ p ][ m ] == 0 )  
 if( AtlasPlrdPresentBlockFlag[ p ][ m ][ n ] == 0 )  
 plrMode[ p ][ m ][ bIdxY ][ bIdxX ] = 0  
 else  
 plrMode[ p ][ m ][ bIdxY ][ bIdxX ] =  
 AtlasPlrdBlockModeMinus1[ p ][ m ][ n ] + 1  
 else if( AtlasPlrdPresentFlag[ p ][ m ] == 0 )  
 plrMode[ p ][ m ][ bIdxY ][ bIdxX ] = 0  
 else  
 plrMode[ p ][ m ][ bIdxY ][ bIdxX ] = AtlasPlrdModeMinus1[ p ][ m ] + 1  
  
 pMode = plrMode[ p ][ m ][ bIdxY ][ bIdxX ]  
 plrInterpolateFlag[ m ][ bIdxY ][ bIdxX ] = plri\_interpolate\_flag[ m ][ pMode ]  
 plrFillingFlag[ m ][ bIdxY ][ bIdxX ] = plri\_filling\_flag[ m ][ pMode ]  
 plrMinimumDepth[ m ][ bIdxY ][ bIdxX ] = plri\_minimum\_depth[ m ][ pMode ]  
 plrNeighbour[ m ][ bIdxY ][ bIdxX ] = plri\_neighbour\_minus1[ m ][ pMode ] + 1  
 }

Depending on the type of a patch, the reconstruction of points in the patch is performed as follows:

* A variable pointCnt is initialized to 0. Arrays recPcGeo, recPcAttr, pointToPatch, pointToPixel, and attrPresent are initially empty. A 2D array pos3D of size ( asps\_map\_count\_minus1 + 1 ) × 3, specifying the 3D coordinates of a reconstructed point corresponding to a particular atlas position, is initialized to 0.
* For p = 0..AtlasTotalNumPatches − 1, the following applies:
* If AtlasPatchType[ p ] is equal to EOM, the following applies:
* If AtlasPatchInAuxVideo[ p ] is equal to 1, subclause H.11.3 is invoked with variables pointCnt, patch index p, AuxVideoHeightNF, and AuxVideoWidthNF, arrays pointToPatch, pointToPixel, attrPresent, oFrame, gFrame, recPcGeo, and, when ai\_attribute\_count[ RecAtlasID ] is greater than 0, arrays aAuxFrame and recPcAttr, as inputs. The outputs are the modified variable pointCnt and the modified arrays pointToPatch, pointToPixel, attrPresent, recPcGeo, and recPcAttr.
* Otherwise ( AtlasPatchInAuxVideo[ p ] is equal to 0 ), subclause H.11.3 is invoked with variables pointCnt, patch index p, asps\_frame\_height, and asps\_frame\_width, arrays pointToPatch, pointToPixel, attrPresent, oFrame, gFrame, recPcGeo, and, when ai\_attribute\_count[ RecAtlasID ] is greater than 0, arrays aFrame[ 0 ] and recPcAttr, as inputs. The outputs are the modified variable pointCnt and the modified arrays pointToPatch, pointToPixel, attrPresent, recPcGeo, and recPcAttr.
* Otherwise, if AtlasPatchType[ p ] is RAW, the following applies:
* If AtlasPatchInAuxVideo[ p ] is equal to 1, subclause H.11.4 is invoked with variables pointCnt, patch index p, AuxVideoHeightNF, and AuxVideoWidthNF, arrays pointToPatch, pointToPixel, attrPresent, gAuxFrame, recPcGeo, and, when ai\_attribute\_count[ RecAtlasID ] is greater than 0, arrays aAuxFrame and recPcAttr, as inputs. The outputs are the modified variable pointCnt and the modified arrays pointToPatch, pointToPixel, attrPresent, recPcGeo, and recPcAttr.
* Otherwise ( AtlasPatchInAuxVideo[ p ] is equal to 0 ), subclause H.11.4 is invoked with variables pointCnt, patch index p, asps\_frame\_height, and asps\_frame\_width, arrays pointToPatch, pointToPixel, attrPresent, gFrame[ 0 ], recPcGeo, and, when ai\_attribute\_count[ RecAtlasID ] is greater than 0, arrays aFrame[ 0 ] and recPcAttr, as inputs. The outputs are the modified variable pointCnt and the modified arrays pointToPatch, pointToPixel, attrPresent, recPcGeo, and recPcAttr.
* Otherwise (AtlasPatchType[ p ] is PROJECTED), the following applies:
* A variable blockShift is set to asps\_log2\_patch\_packing\_block\_size.
* For s = 0..AtlasPatch2dSizeX[ p ] – 1 and t = 0.. AtlasPatch2dSizeY[ p ] – 1, the following applies:
* The atlas coordinates ( x, y ) are derived as follows:

x = s + AtlasPatch2dPosX[ pProjIdx ]  
 y = t + AtlasPatch2dPosY[ pProjIdx ]

* The variables subBlkX and subBlkY are calculated as follows:

subBlkY = y / PatchPackingBlockSize  
 subBlkX = x / PatchPackingBlockSize

* If oFrame[ y ][ x ] is not equal to 0 and AtlasBlockToPatchMap[ subBlkY ][ subBlkX ] is equal to p, the following applies:
* For mapIdx = 0 .. asps\_map\_count\_minus1, the following applies:
* Subclause H.11.2.1 is invoked with variables pointCnt, patch index p, mapIdx, x, and y, and arrays pos3D[ mapIdx ], pointToPatch, pointToPixel, attrPresent, gFrame, recPcGeo, and, when ai\_attribute\_count[ RecAtlasID ] is greater than 0, arrays aFrame[ mapIdx ] and recPcAttr, as inputs. The outputs are the modified variable pointCnt, and the modified arrays pos3D[ mapIdx ], pointToPatch, pointToPixel, attrPresent, recPcGeo, and recPcAttr.
* If asps\_map\_pixel\_deinterleaving\_flag[ mapIdx ] is equal to 1, subclause H.11.2.3 is invoked with variables pointCnt, patch index p, x, and y, and arrays pos3D[ mapIdx ], pointToPatch, pointToPixel, attrPresent, oFrame, gFrame[ mapIdx ], and recPcGeo as inputs. The outputs are the modified variable pointCnt and the modified arrays pointToPatch, pointToPixel, attrPresent, and recPcGeo.
* If asps\_plr\_enabled\_flag is equal to 1 and plri\_map\_present\_flag[ mapIdx ] is equal to 1, subclause H.11.2.4 is invoked with variables pointCnt, patch index p, x, y, and blockShift, and arrays plrMode[ p ][ mapIdx ], plrInterpolateFlag[ mapIdx ], plrFillingFlag[ mapIdx ], plrMinimumDepth[ mapIdx ], plrNeighbour[ mapIdx ], pos3D[ mapIdx ], pointToPatch, pointToPixel, attrPresent, gFrame[ mapIdx ], and recPcGeo. The outputs are the modified variable pointCnt and the modified arrays pointToPatch, pointToPixel, attrPresent, and recPcGeo.
* For n = 0 .. ( pointCnt – 1 ), the following applies:
* If AtlasPatch45DegreeMode[ pointToPatch[ n ] ] is greater than 0, subclause H.11.6 is invoked with patch index p and the three-dimensional array recPcGeo[ n ] as inputs. The output of this process is the modified array recPcGeo[ n ].
* If ai\_attribute\_count[ RecAtlasID ] is greater than 0 and attrPresent[ n ] is equal to 0, the following applies:
* Each reconstructed point with index q, where q != n, that satisfies pointToPatch[ q ] == pointToPatch[ n ] and attrPresent[ q ] is equal to 1 is added to a set, neighbourSet.
* If neighbourSet is not empty, the following applies:
* A point with index m from neighbourSet that is at the minimum distance to the reconstructed point with index n, where the distance is the Euclidean distance, is found. The attributes corresponding to the point with index m are assigned to the attributes for the reconstructed point with index n as follows:

for( a = 0; a < ai\_attribute\_count[ RecAtlasID ]; a++ )  
 for( c = 0; c < ai\_attribute\_dimension\_minus1[ RecAtlasID ][ a ] + 1; c++ )  
 recPcAttr[ n ][ a ][ c ] = recPcAttr[ m ][ a ][ c ]

If multiple reconstruction points from neighbourSet are at the same minimum distance from reconstructed point n, the attribute values for an attribute with index a, are derived based on its corresponding ai\_attribute\_type\_id[ RecAtlasID ][ a ]. In particular, for ai\_attribute\_type\_id[ RecAtlasID ][ a ] equal to ATTR\_TEXTURE, ATTR\_TRANSPARENCY, ATTR\_REFLECTANCE, and ATTR\_NORMAL the average of the corresponding attribute values, rounded to the nearest integer, of the reconstruction points from neighbourSet that are at the same minimum distance is assigned to the corresponding attributes for the reconstructed point with index n. Otherwise, for ai\_attribute\_type\_id[ RecAtlasID ][ a ] equal to ATTR\_MATERIAL\_ID the median value is assigned.

* + 1. Reconstruction of points for PROJECTED patches
       1. Common reconstruction process of points for PROJECTED patches

The inputs to this process are:

* a variable pointCnt, specifying the current number of points in the reconstructed point cloud frame,
* a variable pIdx, specifying the patch index,
* a variable mapIdx, specifying the map index,
* a variable x, specifying the x coordinate of a point on the atlas frame,
* a variable y, specifying the y coordinate of a point on the atlas frame,
* a 1D array pos3D, of size 3, specifying the 3D coordinates of a point,
* a 1D array pointToPatch, of size pointCnt, specifying the patch index for each reconstructed point in the reconstructed point cloud frame,
* a 2D array pointToPixel, of size pointCnt × 2, specifying the atlas coordinates for each reconstructed point in the reconstructed point cloud frame,
* a 1D array attrPresent, of size pointCnt, specifying whether the attribute values for a reconstructed point are present in the attribute video bitstream,
* a 3D array gFrame specifying the decoded geometry frames in the nominal format, where the three dimensions correspond to the map index, row index, and column index, respectively, and
* a 2D array recPcGeo, of size pointCnt × 3, specifying the 3D coordinates of each reconstructed point in the reconstructed point cloud frame.

Additionally, when ai\_attribute\_count[ RecAtlasID ] is greater than 0, the inputs to this process are:

* a 4D array aFrame, specifying the decoded attribute frames in the nominal format, where the four dimensions correspond to the attribute index, attribute dimension, row index, and column index, respectively, and
* a 3D array recPcAttr, specifying the attributes associated with the points in the reconstructed point cloud frame, where the three dimensions correspond to the index of the reconstructed point, the attribute index, and the attribute dimension, respectively.

The outputs of this process are:

* the modified variable pointCnt,
* the modified array pos3D,
* the modified array pointToPatch,
* the modified array pointToPixel,
* the modified array attrPresent,
* the modified array recPcGeo, and
* the modified array recPcAttr.

The following applies:

* The 3D coordinates corresponding to the point with atlas coordinates ( x, y ) are derived by invoking subclause H.11.5 with variables pIdx, gFrame[ mapIdx ][ y ][ x ], x, and y as inputs. The output is used to update the array pos3D.
* Subclause H.11.2.2 is invoked with variables x, y, mapIdx, and array gFrame as inputs. The output is assigned to a variable isDuplicate.
* If isDuplicate is equal to 0, the following applies:
* The array pos3D is copied over to a new entry in the array recPcGeo as follows:

for( k=0; k < 3; k++ )  
 recPcGeo[ pointCnt ][ k ] = pos3D[ k ]

* Then, pointToPixel[ pointCnt ] and pointToPatch[ pointCnt ] are set as follows:

pointToPixel[ pointCnt ][ 0 ] = x  
 pointToPixel[ pointCnt ][ 1 ] = y  
 pointToPatch[ pointCnt ] = pIdx

* If ai\_attribute\_count[ RecAtlasID ] is greater than 0 the following applies:
* For attrIdx = 0 .. ai\_attribute\_count[ RecAtlasID ] − 1, attributes are assigned to the newly added point as follows:

numComps = ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ] + 1  
 for( compIdx = 0; compIdx < numComps; compIdx++ )  
 recPcAttr[ pointCnt ][ attrIdx ][ compIdx ] = aFrame[ attrIdx ][ compIdx ][ y ][ x ]

* attrPresent[ pointCnt ] is set to 1.
* Finally, pointCnt is incremented by 1.
  + - 1. Process to check for duplicated points across maps

This process finds duplicated points that may be associated with a single patch because of the presence of multiple maps. Duplicated points due to other PROJECTED patches or RAW and EOM patches are not accounted for in this process.

Inputs to this process are:

* a variable xC, specifying the x atlas coordinate,
* a variable yC, specifying the y atlas coordinate,
* a variable mapIdx, specifying a map index, and
* a 3D array gFrame, specifying the decoded geometry frames in the nominal format, where the three dimensions correspond to the map index, the row index, and the column index, respectively.

Output of this process is a variable isDuplicate, indicating whether the point corresponding to mapIdx is a duplicate of a point from the set of points with map indices in the range of 0 to mapIdx – 1, inclusive.

The variable isDuplicate is first initialized to 0.

If asps\_vpcc\_remove\_duplicate\_point\_enabled\_flag is equal to 1, the following applies:

for( m = 0; ( m < mapIdx && !isDuplicate ); m++ )  
 if( gFrame[ m ][ yC ][ xC ] == gFrame[ mapIdx ][ yC ][ xC ] )  
 isDuplicate = 1

* + - 1. Reconstruction of points for PROJECTED patches when asps\_pixel\_deinterleaving\_enabled\_flag is equal to 1

The pixel deinterleaving process consists of two stages, a deinterleaving stage and a filling stage. During the deinterleaving stage, two surfaces are constructed, an inner and an outer surface. In the filling stage, points between the inner and outer surface are generated. An example is shown in Figure H-1.



Figure H-1 – Example of pixel deinterleaving process

Inputs to this process are:

* a variable pointCnt, specifying the number of reconstructed points in the reconstructed point cloud frame,
* a variable pIdx, specifying the index of the patch,
* a variable x, specifying the x coordinate of a point on the atlas frame,
* a variable y, specifying the y coordinate of a point on the atlas frame,
* a 1D array pos3D, of size 3, specifying the 3D coordinates of a point,
* a 1D array pointToPatch, of size pointCnt, specifying the patch index for each reconstructed point in the reconstructed point cloud frame,
* a 2D array pointToPixel, of size pointCnt × 2, specifying the atlas coordinates for each reconstructed point in the reconstructed point cloud frame,
* a 1D array attrPresent, of size pointCnt, specifying whether the attribute values for each reconstructed point are present in the attribute video bitstream,
* a 2D array oFrame, of size asps\_frame\_height × asps\_frame\_width, specifying the decoded occupancy frames in the nominal format,
* a 2D array gFrame, of size asps\_frame\_height × asps\_frame\_width, specifying the decoded geometry frames in the nominal format, and
* a 2D array recPcGeo, of size pointCnt × 3, specifying coordinates of points in the reconstructed point cloud frame.

Outputs of this process are:

* the modified variable pointCnt,
* the modified array pointToPatch,
* the modified array pointToPixel,
* the modified array attrPresent, and
* the modified array recPcGeo.

The following applies:

* A variable validNeighbourCount and array depthNeighbours are initialized as follows:

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

* The neighbours of the pixel ( x, y ) that belong to the patch with index pIdx, are identified and their minimum and maximum geometry values are derived as follows:

minDepth = pos3D[ AtlasPatchAxisD[ pIdx ] ]  
maxDepth = pos3D[ AtlasPatchAxisD[ pIdx ] ]

if( x > 0 ) {  
 subBlkY = y / PatchPackingBlockSize  
 subBlkX = (x – 1) / PatchPackingBlockSize  
 if( ( oFrame[ y ][ x – 1 ] != 0 ) && ( AtlasBlockToPatchMap[ subBlkY ][ subBlkX ] == pIdx ) ) {  
 depthNeighbours[ 0 ] = AtlasPatch3dOffsetD[ pIdx ] +  
 ( 1 – 2 \* AtlasPatchProjectionFlag[ pIdx ] ) \* gFrame[ y ][ x – 1 ]  
 validNeighbourCount++  
 minDepth = Min( minDepth, depthNeighbour[ 0 ] )  
 maxDepth = Max( maxDepth, depthNeighbour[ 0 ] )  
 }  
}

if( x < ( asps\_frame\_width – 1 ) ) {  
 subBlkY = y / PatchPackingBlockSize  
 subBlkX = (x + 1) / PatchPackingBlockSize  
 if( ( oFrame[ y ][ x + 1 ] != 0 ) &&  
 ( AtlasBlockToPatchMap[ subBlkY ][ subBlkX ] == pIdx ) ) {  
 depthNeighbours[ 1 ] = AtlasPatch3dOffsetD[ pIdx ] +  
 ( 1 – 2 \* AtlasPatchProjectionFlag[ pIdx ] ) \* gFrame[ y ][ x + 1 ]  
 validNeighbourCount++  
 minDepth = Min( minDepth, depthNeighbour[ 1 ] )  
 maxDepth = Max( maxDepth, depthNeighbour[ 1 ] )  
 }  
}

if( y > 0 ) {  
 subBlkY = (y − 1) / PatchPackingBlockSize  
 subBlkX = x / PatchPackingBlockSize

if( ( oFrame[ y− 1 ][ x ] != 0 ) &&  
 ( AtlasBlockToPatchMap[ subBlkY ][ subBlkX ] == pIdx ) ) {  
 depthNeighbours[ 2 ] = AtlasPatch3dOffsetD[ pIdx ] +  
 ( 1 – 2\* AtlasPatchProjectionFlag[ pIdx ] ) \* gFrame[ y – 1 ][ x ]  
 validNeighbourCount++  
 minDepth = Min( minDepth, depthNeighbour[ 2 ] )  
 maxDepth = Max( maxDepth, depthNeighbour[ 2 ] )  
 }  
}

if( y < ( asps\_frame\_height – 1 ) ) {  
 subBlkY = (y + 1) / PatchPackingBlockSize  
 subBlkX = x / PatchPackingBlockSize  
 if( ( oFrame[ y+ 1 ][ x ] != 0 ) &&  
 ( AtlasBlockToPatchMap[ subBlkY ][ subBlkX ] == pIdx ) ) {  
 depthNeighbours[ 3 ] = AtlasPatch3dOffsetD[ pIdx ] +  
 ( 1 – 2 \* AtlasPatchProjectionFlag[ pIdx ] ) \* gFrame[ y + 1 ][ x ]  
 validNeighbourCount++  
 minDepth = Min( minDepth, depthNeighbour[ 3 ] )  
 maxDepth = Max( maxDepth, depthNeighbour[ 3 ] )  
 }  
}

* If validNeighbourCount is greater than 0, the following applies:
* If( ( x + y ) % 2 ) is equal to 1 the following applies:
* An additional depth value is estimated as follows:

depth1 = pos3D[ AtlasPatchAxisD[ pIdx ] ]  
depthLowerBound = depth1 – asps\_vpcc\_surface\_thickness\_minus1 – 1  
depthUpperBound = depth1 + asps\_vpcc\_surface\_thickness\_minus1 + 1  
for( k = 0; k < 3; k++ )  
 point3D[ k ] = pos3D[ k ]  
if( AtlasPatchProjectionFlag[ pIdx ] == 0 )  
 depth0 = Clip3(depthLowerBound, depth1, minDepth )   
else  
 depth0 = Clip3( depth1, depthUpperBound, maxDepth )

* If depth0 is not equal to depth1, the following applies:

point3D[ AtlasPatchAxisD[ pIdx ] ] = depth0  
for( k = 0; k < 3; k++ )  
 recPcGeo[ PointCnt ][ k ] = point3D[ k ]  
attrPresent[ pointCnt ] = 0  
pointToPatch[ pointCnt ] = pIdx  
pointToPixel[ pointCnt ][ 0 ] = x  
pointToPixel[ pointCnt ][ 1 ] = y  
pointCnt++

* Otherwise ( ( ( x + y ) % 2 ) is equal to 0 ) the following applies:
* An additional depth value is estimated as follows:

depth0 = pos3D[ AtlasPatchAxisD[ pIdx ] ]  
for( k = 0; k < 3; k++ )  
 point3D[ k ] = pos3D[ k ]  
averageDepth = 0  
for( d = 0; d < 4; d++ )  
 averageDepth += depthNeighbours[ d ]  
averageDepth = averageDepth / validNeighbourCount  
depthLowerBound = depth0 – asps\_vpcc\_surface\_thickness\_minus1 – 1  
depthUpperBound = depth0 + asps\_vpcc\_surface\_thickness\_minus1 + 1  
if( AtlasPatchProjectionFlag[ pIdx ] == 0 )  
 depth1 = Clip3( depth0, depthUpperBound, averageDepth )  
else  
 depth1 = Clip3( depthLowerBound, depth0, averageDepth )

* If depth1 is not equal to depth0, the following applies :

point3D[ AtlasPatchAxisD[ pIdx ] ] = depth1  
for( k = 0; k < 3; k++ )  
 recPcGeo[ pointCnt ][ k ] = point3D[ k ]  
attrPresent[ pointCnt ] = 0  
pointToPatch[ pointCnt ] = pIdx  
pointToPixel[ pointCnt ][ 0 ] = x  
pointToPixel[ pointCnt ][ 1 ] = y  
pointCnt++

* The points between depth0 and depth1 are filled as follows:

maxDepth = Max( depth0, depth1 )  
minDepth = Min( depth0, depth1 )

* For d = 1..( maxDepth – minDepth – 1), the following applies:

point3D[ AtlasPatchAxisD[ pIdx ] ] = minDepth + d  
for( k = 0; k < 3; k++ )  
 recPcGeo[ pointCnt ][ k ] = point3D[ k ]  
attrPresent[ pointCnt ] = 0  
pointToPatch[ pointCnt ] = pIdx  
pointToPixel[ pointCnt ][ 0 ] = x  
pointToPixel[ pointCnt ][ 1 ] = y  
pointCnt++

* + - 1. Reconstruction of points for PROJECTED patches when asps\_plr\_enabled\_flag is equal to 1

Inputs to this process are:

* a variable pointCnt, the number of points in the reconstructed point cloud frame,
* a variable pIdx, the index of the patch,
* a variable x, specifying the x coordinate of a point on the atlas frame,
* a variable y, specifying the y coordinate of a point on the atlas frame,
* a variable blockShift, the base-2 logarithm of PatchPackingBlockSize,
* a 2D array plrMode, of size atlasBlockSizeY × atlasBlockSizeX, specifying the PLR mode for a patch packing block in the current atlas frame,
* a 2D array plrInterpolateFlag, of size atlasBlockSizeY × atlasBlockSizeX, specifying the PLR interpolate flag for a patch packing block in the current atlas frame,
* a 2D array, plrFillingFlag, of size atlasBlockSizeY × atlasBlockSizeX, specifying the PLR filling flag for a patch packing block in the current atlas frame,
* a 2D array plrMinimumDepth, of size atlasBlockSizeY × atlasBlockSizeX, specifying the PLR minimum depth for a patch packing block in the current atlas frame,
* a 2D array plrNeighbour, of size atlasBlockSizeY × atlasBlockSizeX, specifying the PLR neighbour for a patch packing block in the current atlas frame,
* a 1D array pos3D, of size 3, specifying the 3D coordinates of a reconstructed point in the reconstructed point cloud frame,
* a 1D array pointToPatch, of size pointCnt, specifying the patch index of each reconstructed point in the reconstructed point cloud frame,
* a 2D array pointToPixel, of size pointCnt × 2, specifying the atlas coordinates for each reconstructed point in the reconstructed point cloud frame,
* a 1D array attrPresent, of size pointCnt, specifying whether the attribute values for a reconstructed point are present in the attribute video bitstream,
* a 2D array gFrame, of size asps\_frame\_height × asps\_frame\_width, specifying the decoded geometry frames in the nominal format, and
* a 2D array recPcGeo, of size pointCnt × 3, specifying the coordinates of points in the reconstructed point cloud frame,

where the variables atlasBlockSizeY  and atlasBlockSizeX are equal to:

atlasBlockSizeY = ( ( AtlasPatch2dSizeY[ pIdx ] − 1 ) >> blockShift ) + 1

atlasBlockSizeX = ( ( AtlasPatch2dSizeY[ pIdx ] − 1 ) >> blockShift ) + 1

Outputs of this process are:

* the modified variable pointCnt,
* the modified array pointToPatch,
* the modified array pointToPixel,
* the modified array attrPresent, and
* the modified array recPcGeo.

The following applies:

* A variable deltaDepth is initialized to 0.
* Set variables yN and xB as follows:

yB = ( y >> blockShift )

* xB = ( x >> blockShift )
* If plrInterpolateFlag[ yB ][ xB ] is equal to 1, then
* A variable deltaLimit is determined, which identifies the maximum or minimum depth values in a neighbourhood depending on the value of the current patch’s projection flag, AtlasPatchProjectionFlag[ pIdx ], as follows:

surfaceThickness = asps\_vpcc\_surface\_thickness\_minus1 + 1  
 deltaLimit = 0  
 startCol = Max( 0, x – plrNeighbour[ yB ][ xB ] )  
 endCol = Min( x + plrNeighbour[ yB ][ xB ], asps\_frame\_width – 1 )  
 startRow = Max( 0, y – plrNeighbour[ yB ][ xB ] )  
 endRow = Min( y + plrNeighbour[ yB ][ xB ], asps\_frame\_height – 1 )  
 for( s = startCol; s <= endCol; s++ ) {  
 sB = s >> blockShift  
 for( t = startRow; t <= endRow; t++ ) {  
 tB = t >> blockShift  
   
 if( ( oFrame[ t ][ s ] != 0 ) && ( AtlasBlockToPatchMap[ tB ][ sB ] == pIdx ) ) {  
 diff = gFrame[ t ][ s ] – gFrame[ y ][ x ]  
 if( AtlasPatchProjectionFlag[ pIdx ] == 0 ) {  
 if( ( diff <= surfaceThickness ) && (diff > deltaLimit ) )  
 deltaLimit = diff  
 } else if( ( diff >= −surfaceThickness ) && (diff < deltaLimit ) ) {  
 deltaLimit = diff  
 }  
 }  
 }  
 }  
  
 if( deltaLimit != 0 ) {  
 if( AtlasPatchProjectionFlag[ pIdx ] == 0 ) {  
 deltaLimit = deltaLimit – 1  
 }  
 else {  
 deltaLimit = deltaLimit + 1  
 }  
 }

Then the variable deltaDepth is derived as follows:

* If AtlasPatchProjectionFlag[ pIdx ] is equal to 0 then,

deltaDepth = Max( deltaLimit, plrMinimumDepth[ yB ][ xB ] )

* Otherwise ( AtlasPatchProjectionFlag[ pIdx ] is equal to 1 ),

deltaDepth = Min( deltaLimit, – ( plrMinimumDepth[ yB ][ xB ]) )

* If deltaDepth is different from 0, the following applies:
* A variable plrDepth is set equal to gFrame[ y ][ x ] + deltaDepth
* A variable orgDepth is set equal to gFrame[ y ][ x ]
* Then, a number of points sharing the same x and y coordinates in the atlas coordinate space are generated with different depth values. First, a point at depth plrDepth, is generated.

pos3D[ AtlasPatchAxisD[ pIdx ] ] = plrDepth  
 for( k = 0; k < 3; k++ )  
 recPcGeo[ pointCnt ][ k ] = pos3D[ k ]  
 attrPresent[ pointCnt ] = 0  
  
 pointToPatch[ pointCnt ] = pIdx  
 pointToPixel[ pointCnt ][ 0 ] = x  
 pointToPixel[ pointCnt ][ 1 ] = y  
 pointCnt++

* If plrFillingFlag[ yB ][ xB ] is equal to 1, then the points corresponding to depth values between orgDepth and plrDepth are generated as follows:

maxDepth = Max( orgDepth, plrDepth )  
 minDepth = Min( orgDepth, plrDepth )  
  
 for( s = minDepth + 1; s < maxDepth; s++ ) {  
 pos3D[ AtlasPatchAxisD[ pIdx ] ] = s  
 for( k = 0; k < 3; k++ )  
 recPcGeo[ pointCnt ][ k ] = pos3D[ k ]  
 attrPresent[ pointCnt ] = 0  
  
 pointToPatch[ pointCnt ] = pIdx  
 pointToPixel[ pointCnt ][ 0 ] = x  
 pointToPixel[ pointCnt ][ 1 ] = y  
 pointCnt++  
 }

* + 1. Reconstruction for EOM patches

Inputs to this process are:

* a variable pointCnt, specifying the number of points in the reconstructed point cloud frame,
* a variable pIdx, specifying the index of the patch,
* a variable aFrameHeight, specifying the height of the decoded attributes frames,
* a variable aFrameWidth, specifying the width of the decoded attributes frames,
* an array pointToPatch, of size pointCnt, specifying the patch index for the reconstructed point,
* an array pointToPixel, of size pointCnt × 2, specifying the atlas coordinates for the reconstructed points,
* a 1D array attrPresent, of size pointCnt, specifying whether the attribute values for a reconstructed point are present in the attribute video bitstream,
* a 2D array oFrame, of size asps\_frame\_height × asps\_frame\_width, specifying the decoded occupancy frames in the nominal format,
* a 3D array gFrame, of size ( asps\_map\_count\_minus1 + 1 ) × asps\_frame\_height × asps\_frame\_width, indicating the decoded geometry frames in the nominal format and
* a 2D array recPcGeo, of size pointCnt × 2, specifying a list of coordinates of the points in the reconstructed point cloud frame.

Additionally, when ai\_attribute\_count[ RecAtlasID ] is greater than 0, the inputs to this process are:

* a 4D array aFrame, of size ai\_attribute\_count[ RecAtlasID ] × ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ] × aFrameHeight × aFrameWidth , specifying the decoded attribute frames in the nominal format, and
* a 3D array recPcAttr, off size pointCnt × ai\_attribute\_count[ RecAtlasID ] × ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ], specifying a list of attributes associated with the points in the reconstructed point cloud frame.

Outputs of this process are:

* the modified variable pointCnt
* the modified array pointToPatch,
* the modified array pointToPixel,
* the modified array attrPresent,
* the modified array recPcGeo, and
* the modified array recPcAttr.

An array eomPointOffset, of size AtlasEpduPatchCount[ pIdx ], is derived as follows:

eomPointOffset[ 0 ] = 0  
patchCnt = AtlasEpduPatchCount[ pIdx ]  
for( i = 1; i < patchCnt; i++ )  
 eomPointOffset[ i ] = eomPointOffset[ i – 1 ] + AtlasEpduPoints[ pIdx ][ i – 1 ]

When ai\_attribute\_count[ RecAtlasID ] is greater than 0, it is a requirement of bitstream conformance that the following applies:

eomPointOffset[ patchCnt – 1 ] + AtlasEpduPoints[ pIdx ][ patchCnt – 1 ] <= (H‑43)  
 AtlasPatch2dSizeX[ pIdx ] \* AtlasPatch2dSizeY[ pIdx ]

When ai\_attribute\_count[ RecAtlasID ] is equal to 0, a decoder can ignore the information in the array eomPointOffset, and only needs to consider information in arrays AtlasEpduPatchCount and AtlasEpduAssociatedPatchIndex to determine the PROJECTED patches that contain geometry information corresponding to the current EOM patch.

A variable eomPointCnt, which indicates the current number of processed EOM points in the current EOM patch, is initialized to 0. Then, the following applies:

* For i = 0..AtlasEpduPatchCount[ pIdx ] − 1, the following applies:
* A variable pProjIdx is set equal to AtlasEpduAssociatedPatchIndex[ pIdx ][ i ]
* For s = 0..AtlasPatch2dSizeX[ pProjIdx ] − 1, and t = 0..AtlasPatch2dSizeY[ pProjIdx ] – 1, the following applies:
* The atlas coordinates x and y are calculated as follows:

x = s + AtlasPatch2dPosX[ pProjIdx ]  
y = t + AtlasPatch2dPosY[ pProjIdx ]

* The variables subBlkX and subBlkY are calculated as follows:

subBlkX = x / PatchPackingBlockSize  
subBlkY = y / PatchPackingBlockSize

* If oFrame[ y ][ x ] is not equal to 0 and AtlasBlockToPatchMap[ subBlkY ][ subBlkX ] is equal to pProjIdx, the following applies:
* A variable d0 is set equal to gFrame[ 0 ][ y ][ x ].
* A variable d1 is derived as follows:
* If asps\_map\_count\_minus1 is equal to 0, a variable d1 is set equal to d0 + asps\_eom\_fix\_bit\_count\_minus1 + 1.
* Otherwise ( asps\_map\_count\_minus1 is greater than 0 ) d1 is set equal to gFrame[ 1 ][ y ][ x ].
* If ( d1 – d0 ) is greater than 1, the following applies:
* A variable eomCode is initialized as follows:

eomCode = ( 1 << (d1 – d0 – 1 ) ) − oFrame[ y ][ x ]

* For b = 0 .. (d1 – d0 – 2 ), and if ( ( ( eomCode >> b ) & 0x01 ) == 1 ), the following applies:
* The 3D coordinates corresponding to the point with atlas coordinates ( x, y ) are derived by invoking subclause H.11.5 with variables pIdx, ( d0 + b + 1 ), x, and y as inputs. The output is an array, pos3D.
* The array pos3D, is copied over to a new entry in recPcGeo, as follows:

for( n=0; n < 3; n++ )

recPcGeo[ pointCnt ][ n ] = pos3D[ n ]

* If ai\_attribute\_count[ RecAtlasID ] is greater than 0, attributes are assigned to the newly added point, as follows:

eomPos = eomPointOffset[ i ] + eomPointCnt  
inBlockPos = eomPos % ( PatchPackingBlockSize \* PatchPackingBlockSize )  
nbBlock = eomPos / ( PatchPackingBlockSize \* PatchPackingBlockSize )  
xBlock = nbBlock % ( AtlasPatch2dSizeX[ pIdx ]/ PatchPackingBlockSize )  
yBlock = nbBlock / ( AtlasPatch2dSizeY[ pIdx ]/ PatchPackingBlockSize )

xx = xBlock \* PatchPackingBlockSize + inBlockPos % PatchPackingBlockSize   
 + AtlasPatch2dPosX[ pIdx ]  
yy = yBlock \* PatchPackingBlockSize + inBlockPos / PatchPackingBlockSize  
 + AtlasPatch2dPosY[ pIdx ]

for( attrIdx = 0; attrIdx < ai\_attribute\_count[ RecAtlasID ]; attrIdx++ ) {  
 attrDim = ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ] + 1  
 for( compIdx = 0; compIdx < attrDim; compIdx++ ) {  
 recPcAttr[ pointCnt ][ attrIdx ][ compIdx ] =  
 aFrame[ attrIdx ][ compIdx ][ yy ][ xx ]  
 }  
}  
attrPresent[ pointCnt ] = 1

* Otherwise ( ai\_attribute\_count[ RecAtlasID ] is equal to 0 ),

attrPresent[ pointCnt ] = 0

* The following applies:

pointToPixel[ pointCnt ][ 0 ] = x  
pointToPixel[ pointCnt ][ 1 ] = y  
pointToPatch[ pointCnt ] = pProjIdx  
eomPointCnt++  
pointCnt++

Figure H-2 shows an example EOM patch and an EOM coded points attribute layout in an attribute frame. The EOM attribute values are stored in the EOM patch contiguously in raster scan order with a stride of patch packing block size, for each component. Patch packing blocks filled with EOM coded points are stored in raster scan order in the EOM patch.

Figure H-2 shows an example of an occupancy frame with EOM coded points.

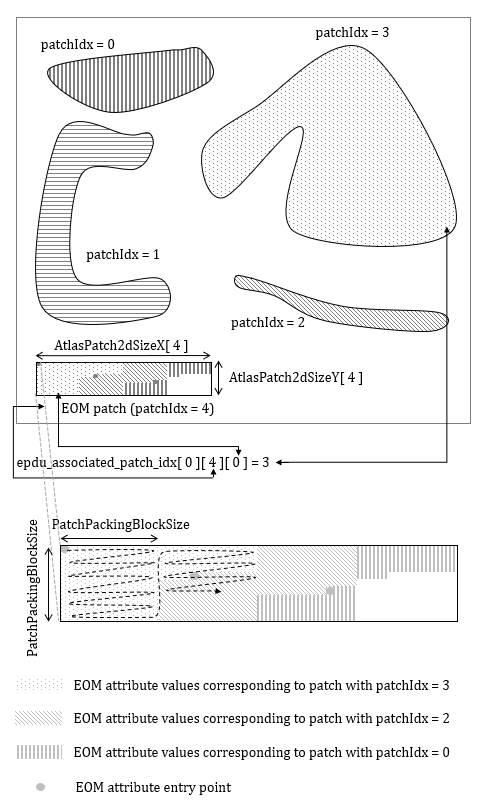


Figure H-2 – An example of an EOM patch, its associated EOM coded points, and   
the layout of the corresponding values in an attribute frame

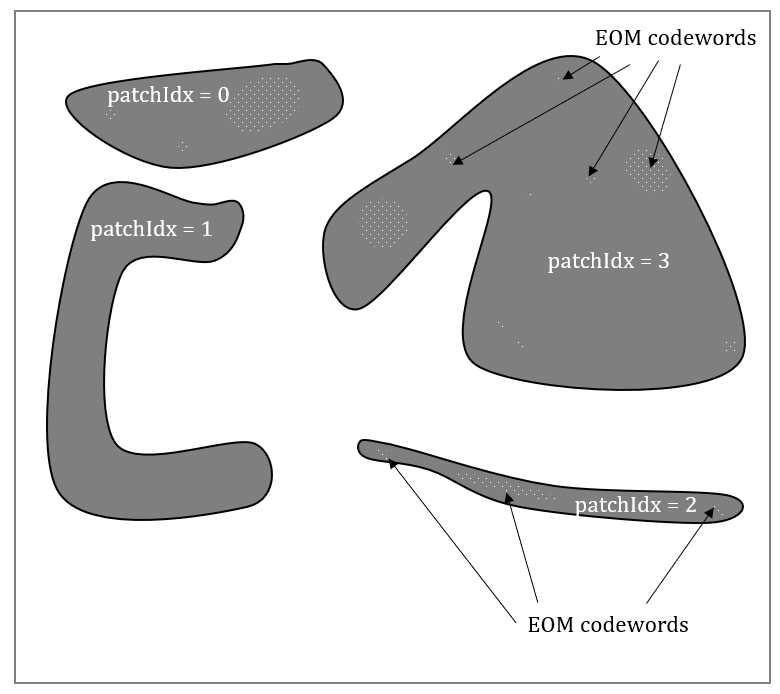


Figure H-3 – An example of an occupancy frame with EOM coded points

* + 1. Reconstruction of RAW patches

Inputs to this process are:

* a variable pointCnt, the number of points in the reconstructed point cloud frame,
* a variable pIdx, the index of the patch,
* a variable frameHeight, the height of the decoded geometry and possibly decoded attributes frames,
* a variable frameWidth, the width of the decoded geometry and possibly decoded attributes frames,
* a 1D array pointToPatch, of size pointCnt, specifying the patch index for the reconstructed point,
* a 2D array pointToPixel, of size pointCnt × 2, specifying the atlas coordinates for the reconstructed point,
* a 1D array attrPresent, of size pointCnt, specifying whether the attribute values for a reconstructed point are present in the attribute video bitstream,
* a 2D array gFrame, of size frameHeight × frameWidth, specifying the decoded geometry frames in the nominal format, and
* a 2D array recPcGeo, of size pointCnt × 3, specifying a list of coordinates of the points in the reconstructed point cloud frame.

Additionally, when ai\_attribute\_count[ RecAtlasID ] is greater than 0, the inputs to this process are:

* a 4D array aFrame, of size attribute\_count[ RecAtlasID ] × ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ] × frameHeight × frameWidth, specifying the decoded attribute frames in the nominal format, and
* a 3D array recPcAttr, of size pointCnt × ai\_attribute\_count[ RecAtlasID ] × ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ], specifying a list of attributes associated with the points in the reconstructed point cloud frame.

Outputs of this process are:

* the modified variable pointCnt,
* the modified array pointToPatch,
* the modified array pointToPixel,
* the modified array attrPresent,
* the modified array recPcGeo, and
* the modified array recPcAttr.

The 3D coordinates of a RAW coded point are retrieved from the first map of the first component of the geometry frame containing the RAW patch corresponding to the RAW coded point, and the point is added to recPcGeo. The attribute information for the RAW coded point is retrieved from the attribute frame containing the RAW patch corresponding to the RAW coded point, across the different attribute dimensions at the same location and added to recPcAttr. This is accomplished as follows:

An array rawPos1D[ m ] where m is in the range of 0 to 3 \* AtlasPatchRawPoints[ pIdx ] – 1, inclusive, is initialized to 0.

m = 0  
for( k = 0; k<3; k++)  
 for( pointIdx = 0; pointIdx < AtlasPatchRawPoints[ pIdx ]; pointIdx++ ) {

y = AtlasPatch2dPosY[ pIdx ] + ( pointIdx / AtlasPatch2dSizeX[ pIdx ] )

x = AtlasPatch2dPosX[ pIdx ] + ( pointIdx % AtlasPatch2dSizeX[ pIdx ] )

rawPos1D[ m ] = gFrame[ y ][ x ]

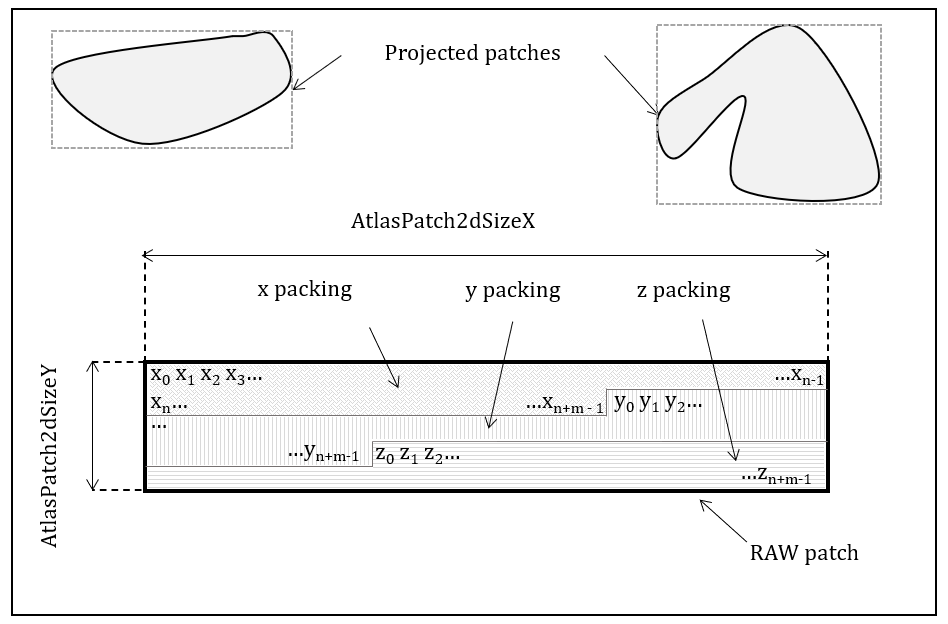
m++

}

for( pointIdx = 0; n < AtlasPatchRawPoints[ pIdx ]; n++ ) {

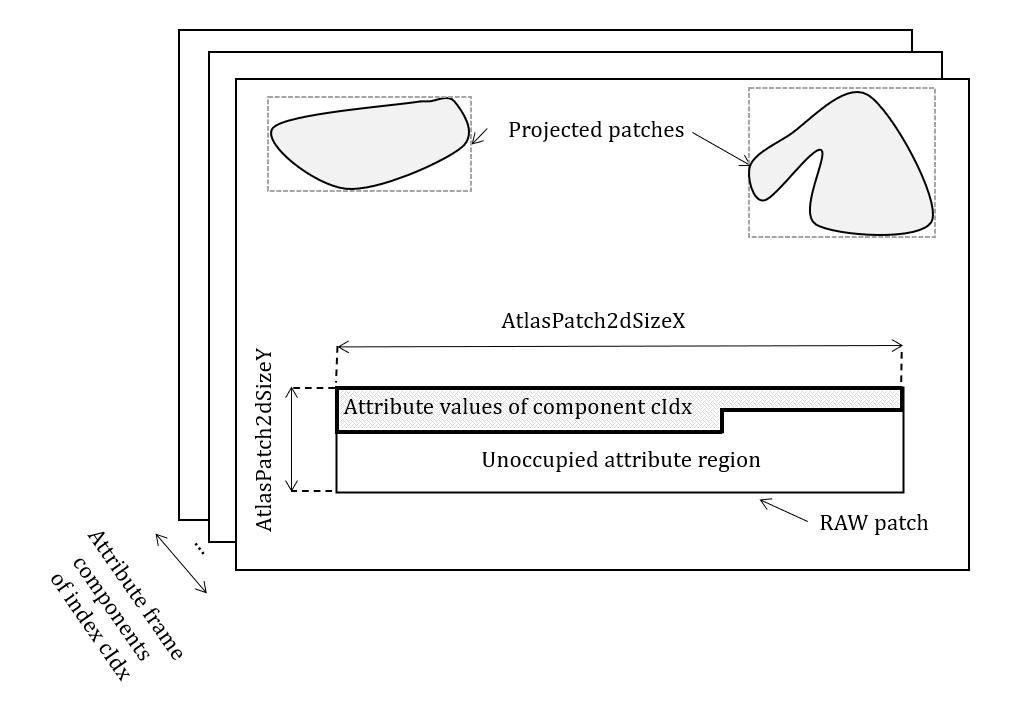
recPcGeo[ pointCnt ][ 0 ] = rawPos1D[ pointIdx ] +  
 AtlasPatch3dOffsetU[ pIdx ] recPcGeo[ pointCnt ][ 1 ] = rawPos1D[ pointIdx + AtlasPatchRawPoints[ pIdx ] ] +  
 AtlasPatch3dOffsetV[ pIdx ] recPcGeo[ pointCnt ][ 2 ] = rawPos1D[ pointIdx + 2 \* AtlasPatchRawPoints[ pIdx ] ] +  
 AtlasPatch3dOffsetD[ pIdx ]  
  
 y = AtlasPatch2dPosY[ pIdx ] + ( pointIdx / AtlasPatch2dSizeX[ pIdx ] )  
 x = AtlasPatch2dPosX[ pIdx ] + ( pointIdx % AtlasPatch2dSizeX[ pIdx ] )  
  
 for( attrIdx = 0; attrIdx < ai\_attribute\_count[ RecAtlasID ]; attrIdx++ ) {  
 attrDim = ai\_attribute\_dimension\_minus1[ RecAtlasID ][ attrIdx ] + 1  
 for( compIdx = 0; compIdx < attrDim; compIdx++ )  
 recPcAttr[ pointCnt ][ attrIdx ][ compIdx ] = aFrame[ attrIdx ][ compIdx ][ y ][ x ]  
 }  
  
 if( ai\_attribute\_count[ RecAtlasID ] > 0 )  
 attrPresent[ pointCnt ] = 1   
  
 pointToPixel[ pointCnt ][ 0 ] = −1  
 pointToPixel[ pointCnt ][ 1 ] = −1  
 pointToPatch[ pointCnt ] = pIdx  
 pointCnt++  
}

Figure H-4 shows an example RAW patch in a geometry frame, where the coordinates of 3D points represented by RAW coded points are each packed contiguously in the first component of the geometry frame.



**Figure H-4 –** **An example of a RAW patch and RAW coded points geometry layout in a geometry frame**

Figure H-5 shows an example RAW patch in an attribute frame where RAW attribute values are stored in raster scan order for each component of index cIdx.



**Figure H-5 –** **An example of a RAW patch and RAW coded points attribute layout in an attribute frame**

* + 1. Conversion from atlas coordinates to 3D coordinates

Inputs to this process are:

* a variable pIdx, the patch index,
* a variable depthValue, the depth of a point,
* a variable x, the x atlas coordinate,
* a variable y, the y atlas coordinate.

Output of this process is a 2D array, pos3D, of size 3, specifying the 3D coordinates of a point.

The following applies:

* A variable oIdx is set to AtlasPatchOrientationIndex[ pIdx ].
* Variables posX, posY, sizeX, sizeY, lodX, and lodY are assigned as follows:

posX = AtlasPatch2dPosX[ pIdx ]  
 posY = AtlasPatch2dPosY[ pIdx ]  
 sizeX = AtlasPatch2dSizeX[ pIdx ]  
 sizeY = AtlasPatch2dSizeY[ pIdx ]  
 lodX = AtlasPatchLoDScaleX[ pIdx ]  
 lodY = AtlasPatchLoDScaleY[ pIdx ]

* The atlas coordinates (x, y) are converted to a local patch coordinate pair ( u, v ) as follows:

(H‑44)

where **R**o and **R**s are specified in Table 11.

* The local patch coordinate pair ( u, v ) is converted to 3D coordinates as follows:

pos3D[ AtlasPatchAxisU[ pIdx ] ] = AtlasPatch3dOffsetU[ pIdx ] + u (H‑45)

pos3D[ AtlasPatchAxisV[ pIdx ] ] = AtlasPatch3dOffsetV[ pIdx ] + v (H‑46)

tempD = ( 1 – 2\* AtlasPatchProjectionFlag[ pIdx ] ) \* depthValue (H‑47)

pos3D[ AtlasPatchAxisD[ pIdx ] ] = Max(0, AtlasPatch3dOffsetD[ pIdx ] + tempD ) (H‑48)

Some V-PCC decoder implementations may choose to clip the reconstructed 3D coordinates to the range of 0 to ( 1 << ( asps\_geometry\_3d\_bit\_depth\_minus1 + 1 ) ) − 1, inclusive. Other decoder implementations may apply the clipping operation after the 45 degrees transformation or the post-reconstruction process.

* + 1. Conversion of 3D coordinates for projection planes at 45 degrees

Inputs to this process are:

* a variable pIdx, specifying the patch index, and
* an array point3D, of size 3, specifying the 3D coordinates of a point.

Output of this process is the modified array point3D.

The 3D coordinates of the array point3D are adjusted based on the value of the variable AtlasPatch45DegreeMode[ pIdx ] as follows:

* The variable rotBias is initialized as follows:

rotBias = ( 1 << ( asps\_geometry\_3d\_bit\_depth\_minus1 ) – 1 ) (H‑49)

* The variables rotAxis1 and rotAxis2 are derived as follows:

rotAxis1 = ( AtlasPatch45DegreeMode[ pIdx ] – 1 ) (H‑50)

rotAxis2 = ( AtlasPatch45DegreeMode[ pIdx ] + 1 ) % 3. (H‑51)

* The 3D coordinates of the array point3D are modified as follows:

temp1 = point3D[ rotAxis1 ]

temp2 = point3D[ rotAxis2 ]

point3D[ rotAxis1 ] = (temp1 – temp2 + rotBias ) / 2 (H‑52)

point3D[ rotAxis2 ] = (temp1 + temp2 – rotBias ) / 2 (H‑53)

NOTE – Projection planes at 45 degree temporarily increase the geometry range to perform invertible rotation. It is recommended that the encoder increases by 1 the bit count for geometry, defined by the syntax element asps\_geometry\_3d\_bit\_depth\_minus1.

* 1. Post-reconstruction process
     1. General

The post-reconstruction process depends on the profiles defined in subclause H.15.

The processes described in this subclause are invoked for reconstructed point cloud frames and syntax elements associated with the same atlas ID, identified by a variable postRecAtlasID.

This process is invoked after reconstructing the geometry and attributes for the current point cloud frame with a composition time index compTimeIdx. This process takes as inputs the syntax elements and upper-case variables from subclauses H.8, H.9, H.10, and H.11, and the following variables and arrays:

* a variable pointCnt, specifying the number of reconstructed points for the current point cloud frame associated with the current atlas,
* a 1D array attrBitDepth, of size ai\_attribute\_count[ PostRecAtlasID ], specifying the nominal bit depth,
* a 2D array oFrame, of size asps\_frame\_height × asps\_frame\_width, specifying the occupancy frame, at the nominal resolution, corresponding to the current point cloud frame,
* a 2D array recPcGeo, of size pointCnt × 3, specifying the geometry positions of the points in the current reconstructed point cloud frame, and
* a 3D array recPcAttr, of size poinCnt × ai\_attribute\_count[ PostRecAtlasID ] × ( ai\_attribute\_dimension\_minus1[ PostRecAtlasID ] + 1 ), specifying the reconstructed attributes for the current reconstructed point cloud frame.

The outputs of this process are:

* the 2D array recPcGeo, possibly modified by application of geometry smoothing, and
* the 3D array recPcAttr, possibly modified by the application of attribute smoothing.

The following ordered steps apply:

1. Two 1D arrays, isBoundaryPoint and isModifiedPoint, of size pointCnt, are initialized to 0. isBoundaryPoint specifies whether the reconstructed points lie on or near a patch boundary on the atlas. isModifiedPoint specifies whether the reconstructed points were modified by geometry smoothing.
2. If the point cloud reconstruction system has selected to use the m-th geometry smoothing instance for performing geometry smoothing operations during point cloud reconstruction and the bitstream contains a geometry smoothing SEI message with gs\_method\_type[ m ] equal to 1 at the time instance compTimeIdx, subclause H.12.2 is invoked with variables pointCnt, gs\_method\_type\_index, m, and arrays oFrame, recPcGeo, and pointToPatch as inputs. The outputs of this process are the array recPcGeoSmooth, which contains the updated positions of the points in the current reconstructed point cloud frame after geometry smoothing, and the updated arrays isBoundaryPoint, and isModifiedPoint.
3. If the point cloud reconstruction system has selected to use the m-th geometry smoothing instance for which gs\_method\_type[ m ] is greater than 0 at the time instance compTimeIdx, the attribute transfer process as specified in subclause H.12.8 may be invoked with variable pointCnt, and arrays isModifiedPoint, recPcGeo, recPcGeoSmooth, and recPcAttr as inputs. The output of this process is an updated array recPcAttrMod, which contains the updated attribute values of the points in the current reconstructed point cloud frame after attribute transfer. Subclause H.12.8 may be substituted by an alternate process that produces similar visual quality. The definition of the same visual quality is outside the scope of this document.
4. For each attribute aIdx, aIdx =0..ai\_attribute\_count[ PostRecAtlasID ] – 1, if the attribute identifier is ATTR\_TEXTURE, ai\_attribute\_dimension\_minus1[ PostRecAtlasID ][ aIdx ] is equal to 0 or 2, the point cloud reconstruction system has selected to use the q-th attribute smoothing instance for performing attribute smoothing, and the bitstream contains an attribute smoothing SEI message with as\_filter\_instance\_type[ aIdx ][ q ] equal to 1 at the time instance compTimeIdx, subclause H.12.3 is invoked with pointCnt, aIdx, q, ( ai\_attribute\_dimension\_minus1[ PostRecAtlasID ][ aIdx ] + 1 ), attrBitDepth[ aIdx ], and arrays oFrame, recPcGeo, recPcAttr, and pointToPatch as inputs, and the output of the process is the updated array recPcAttr, the current reconstructed point cloud attribute frame for attribute aIdx after attribute smoothing.

NOTE – A bitstream may contain multiple geometry and attribute smoothing filter instances. The point cloud reconstruction system may choose an appropriate filter instance for geometry and/or attribute smoothing based on considerations such as computational complexity, power, and visual quality requirements, etc. These considerations are outside the scope of this document.

* + 1. Geometry smoothing process

This process is invoked when a bitstream contains a geometry smoothing SEI message with a value of gs\_method\_type[ r ] equal to 1, and the point cloud reconstruction system has selected to use this r-th geometry smoothing instance for performing smoothing during point cloud reconstruction.

Inputs to this process are:

* pointCnt, the number of points in the current reconstructed point cloud frame,
* m, a geometry smoothing filter instance,
* oFrame[ y ][ x ], y = 0..asps\_frame\_height – 1, x = 0..asps\_frame\_width – 1, the occupancy frame, at the nominal resolution, corresponding to the current point cloud frame,
* recPcGeo[ n ][ k ], n = 0..pointCnt – 1, k = 0..2, an array containing the geometry positions of points in the current reconstructed point cloud frame, and
* pointToPatch[ n ], n = 0..pointCnt – 1, the patch index corresponding to each reconstructed point in the current point cloud frame.

Outputs of this process are:

* recPcGeoSmoothed[ n ][ k ], n = 0..pointCnt – 1, k = 0..2, an array containing the geometry positions of points in the reconstructed point cloud frame after the geometry smoothing process,
* isModifiedPoint[ n ], n = 0..pointCnt – 1, an array specifying whether the geometry smoothing process modified the position of the n-th reconstructed point, and
* isBoundaryPoint[ n ], n = 0..pointCnt – 1, an array specifying whether the n-th reconstructed point is near a patch boundary on the atlas.

The following applies:

* A variable geoGridSize is set to ( gs\_grid\_size\_minus2[ m ] + 2 ).
* Variable geoSmoothingThreshold is initialized to gs\_threshold[ m ].
* An array isBoundaryPoint[ n ], n =0..pointCnt – 1, is initialized to 0.
* An array isModifiedPoint[ n ], n =0..pointCnt – 1, is initialized to 0.
* An array recPcGeoSmoothed[ n ][ k ], n = 0..pointCnt – 1, k = 0..2, is initialized using the array recPcGeo[ n ][ k ] as follows:

for( n = 0; n < pointCnt; n++ )  
 for( k = 0; n < 3; k++ )  
 recPcGeoSmoothed[ n ][ k ] = recPcGeo[ n ][ k ]

* For each reconstructed point with index n, n =0..pointCnt – 1, the following applies:
* The position of the point on the atlas is derived as follows:

y = pointToPixel[ n ][ 1 ]  
 x = pointToPixel[ n ][ 0 ]

* A variable p is set equal to pointToPatch[ n ].
* If all of the conditions:
* oFrame[ y ][ x ] != 0
* isBoundaryPoint[ n ] == 0
* ( AtlasPatchType[ p ] != EOM ) || ( gs\_filter\_eom\_points\_flag[ m ] == 1 )

are satisfied, then subclause H.12.5 is invoked with oFrame, y, and x as inputs and the output is assigned to isBoundaryPoint[ n ].

* Subclause H.12.4.1 is invoked with variables geoGridSize, pointCnt, and arrays isBoundaryPoint, recPcGeo, and pointToPatch as inputs and the outputs are variables numCells1D, numBoundaryCells, and arrays cellIdxLut, geoCenterGrid, cellCnt, and cellDoSmoothing.

Then the following process applies:

for( n =0; n < pointCnt ; n++ ) {

qx = recPcGeo[ n ][ 0 ]  
 qy = recPcGeo[ n ][ 1 ]  
 qz = recPcGeo[ n ][ 2 ]

for( k =0; k < 3; k++ )  
 recPos3D[ k ] = recPcGeo[ n ][ k ]

halfGridSize = Max( geoGridSize / 2, 1 );  
 maxCoord = geoGridSize \* numCells1D;  
 if( qx < halfGridSize || maxCoord <= qx + halfGridSize ||  
 qy < halfGridSize || maxCoord <= qy + halfGridSize ||  
 qz < halfGridSize || maxCoord <= qz + halfGridSize )  
 continue

If isBoundaryPoint[ n ] is greater than 0, subclause H.12.6 is invoked with variables numCells1D,numBoundaryCells, and arrays recPos3D, cellIdxLut, cellCnt, and cellDoSmoothing as inputs and the outputs are variables otherClusterPtCnt, currentCell1DIdx, and arrays idxArr, cellAnchor.

When otherClusterPointCnt is equal to 1, the following applies:

* A 4D array, centroid, of size 2 × 2 × 2 × 3 is calculated as follows:

for( dx = 0; dx < 2; dx++ ) {  
 for( dy = 0; dy < 2; dy++ ) {  
 for( dz = 0; dz < 2; dz++ ) {  
 tmpIdx = idxArr[ dx ][ dy ][ dz ]  
 if( cellCnt[ tmpIdx ] > 0 )  
 for( k = 0; k < 3; k++ )  
 centroid[ dx ][ dy ][ dz ][ k ] = geoCenterGrid[ tmpIdx ][ k ]  
 else  
 for( k = 0; k < 3; k++ )  
 centroid[ dx ][ dy ][ dz ][ k ] = recPcGeo[ n ][ k ]  
 }  
 }  
 }

Subclause H.12.7.1 is invoked with variables geoGridSize, numBoundaryCells, and arrays recPcGeo, centroid, cellAnchor, idxArr, and cellCnt as inputs and the outputs are pos3DFilt, a filtered version of the current reconstructed point, and weightedCnt, a weighted average of the number of points from the 2 × 2 × 2 grid of cells.

The following applies:

sqrNorm = 0  
 for( k = 0; k < 3; k++ ) {  
 diff = recPcGeo[ n ][ k ] \* weightedCnt – pos3DFilt[ k ]  
 sqrNorm += diff \* diff  
 }  
 dist2 = sqrNorm ÷ weightedCnt  
 if( dist2 >= Max( geoSmoothingThreshold, weightedCnt ) \* 2 ) {  
 for( k = 0; k < 3; k++ )  
 recPcGeoSmoothed[ n ][ k ] = Round( pos3DFilt[ k ] )  
 isModifiedPoint[ n ] = 1  
 }  
 }

* + 1. Attribute smoothing process

This process is invoked when the attribute identifier is ATTR\_TEXTURE, the attribute dimension is equal to 1 or 3, the point cloud reconstruction system has selected to use the m-th attribute filtering instance for performing attribute smoothing during point cloud reconstruction of the attribute with index k, and the bitstream contains an attribute smoothing SEI message with a value of AttributeSmoothingMethod[ k ][ m ] equal to 1.

When the attribute to be smoothed consists of red, green, and blue components, it is recommended that the components be ordered as green, blue, and red.

Inputs to this process are:

* a variable pointCnt, specifying the number of points in the current reconstructed point cloud frame,
* a variable aIdx, specifying an attribute index,
* a variable m, specifying an attribute smoothing filter instance,
* a variable numComps, specifying the number of components corresponding to the attribute with index aIdx,
* a variable attrBitDepth, specifying the bit depth of the reconstructed attribute values,
* a 2D array oFrame, of size asps\_frame\_height × asps\_frame\_width, specifying the occupancy frame, at the nominal resolution, corresponding to the current point cloud frame,
* a 2D array recPcGeo, of size pointCnt × 3, specifying (possibly smoothed) reconstructed positions,
* a 3D array recPcAttr[ aIdx ][ n ][ j ], specifying the reconstructed attribute values for the attribute with index aIdx, n = 0..pointCnt – 1, and j = 0..numComps – 1, and
* a 1D array pointToPatch, of size pointCnt, specifying the patch index corresponding to each reconstructed point in the current point cloud frame.

Output of this process is:

* a 3D array recPcAttrSmoothed[ aIdx ][ n ][ j ], specifying the reconstructed attribute values for the attribute with index aIdx, n = 0..pointCnt – 1, and j = 0..numComps – 1.

The following applies:

* A variable attrGridSize is set to AttributeSmoothingGridSize[ aIdx ][ m ].
* A variable bitDepthDiff is set to ( attrBitDepth – 8 ).
* Variables attrSmoothingThreshold, attrThresholdVariation, and attrThresholdDiff are initialized as follows:
* If bitDepthDiff is greater than or equal to 0,

attrSmoothingThreshold = as\_threshold[ aIdx ][ m ] << bitDepthDiff  
attrThresholdVariation = as\_threshold\_variation[ aIdx ][ m ] << bitDepthDiff  
attrThresholdDiff = as\_threshold\_difference[ aIdx ][ m ] << bitDepthDiff

* Otherwise ( bitDepthDiff is less than 0 )

attrSmoothingThreshold = as\_threshold[ aIdx ][ m ] >> ( −bitDepthDiff )  
attrThresholdVariation = as\_threshold\_variation[ aIdx ][ m ] >> ( −bitDepthDiff )  
attrThresholdDiff = as\_threshold\_difference[ aIdx ][ m ] >> ( −bitDepthDiff )

* An array isBoundaryPoint[ n ], n =0..pointCnt – 1, is initialized to 0.
* An array recPcAttrSmoothed[ aIdx ][ n ][ j ], n = 0..pointCnt – 1, j = 0..numComps – 1, is initialized using the array recPcAttr[ aIdx ][ n ][ j ] as follows:

for( n = 0; n < pointCnt; n++ )  
 for( j = 0; j < numComps; j++ )  
 recPcAttrSmoothed[ aIdx ][ n ][ j ] = recPcAttr[ aIdx ][ n ][ j ]

* For each reconstructed point with index n, n =0..pointCnt – 1, the following applies:
* The position of the point on the atlas is derived as follows:

y = pointToPixel[ n ][ 1 ]  
 x = pointToPixel[ n ][ 0 ]

* A variable p is set equal to pointToPatch[ n ].
* Subclause H.12.5 is invoked with oFrame, y, and x as inputs if all of the conditions below are satisfied. The output is assigned to isBoundaryPoint[ n ].
* oFrame[ y ][ x ] != 0
* isBoundaryPoint[ n ] == 0
* ( AtlasPatchType[ p ] != EOM ) || ( as\_filter\_eom\_points\_flag[ aIdx ][ m ] == 1 )

NOTE 1 – If geometry smoothing process in subclause H.12.2 has already been invoked, instead of repeating the calculation of isBoundaryPoint[ n ], n =0..pointCnt – 1, it may be possible to use isBoundaryPoint[ n ] obtained as an output of subclause H.12.2. For this to be feasible, the grid sizes for geometry smoothing and attribute smoothing should be the same. In addition, the values of gs\_filter\_eom\_points\_flag and as\_filter\_eom\_points\_flag for the chosen filter instances should also match.

* Subclause H.12.4.2 is invoked with variables numComps, attrGridSize, and pointCnt, and arrays isBoundaryPoint, recPcGeo, recPcAttr[ aIdx ], and pointToPatch as inputs. The outputs are variables numCells1D and numBoundaryCells, and arrays cellIdxLut, attrCenterGrid, cellCnt, cellDoSmoothing, and cellLum.

for( n =0; n < pointCnt ; n++ ) {

x = recPcGeo[ n ][ 0 ]  
y = recPcGeo[ n ][ 1 ]  
z = recPcGeo[ n ][ 2 ]

for( k =0; k < 3; k++ )  
 recPos3D[ k ] = recPcGeo[ n ][ k ]

halfGridSize = Max( attrGridSize / 2, 1 )  
maxCoord = attrGridSize \* numCells1D  
if( x < halfGridSize || maxCoord <= x + halfGridSize ||  
 y < halfGridSize || maxCoord <= y + halfGridSize ||   
 z < halfGridSize || maxCoord <= z + halfGridSize )  
 continue

* If isBoundaryPoint[ n ] is greater than 0, subclause H.12.6 is invoked with variables numCells1D, numBoundaryCells, and arrays recPos3D, cellIdxLut, cellCnt, and cellDoSmoothing as inputs and the outputs are variables otherClusterPtCnt, currentCell1DIdx and arrays idxArr and cellAnchor.
* When otherClusterPointCnt is equal to 1, the following applies:
* An array currAttr[ j ], j = 0..numComps – 1 is initialized as follows:

for( j = 0; j < numComps; j++ )  
 currAttr[ j ] = recPcAttr[ aIdx ][ n ][ j ]

* A 4D array centroid, of size 2 × 2 × 2 × numComps, is initialized to 0. The following applies:

exitFlag = 0  
 for( dx = 0; dx < 2; dx++ ) {  
 for( dy = 0; dy < 2; dy++ ) {  
 for( dz = 0; dz < 2; dz++ ) {  
 tmpIdx = idxArr[ dx ][ dy ][ dz ]  
 if( tmpIdx == currentCell1DIdx ) {  
 for( m = 0; m < cellCnt[ tmpIdx ]; m++ )  
 lum[ m ] = cellLum[ tmpIdx ][ m ]  
 if( cellCnt[ tmpIdx ] > 0 ) {  
 for( j = 0; j < numComps; j++ )  
 centroid[ dx ][ dy ][ dz ][ j ]  
  = attrCenterGrid[ tmpIdx ][ j ] ÷ cellCnt[ tmpIdx ]  
 if( cellCnt[ tmpIdx ] > 1 ) {  
 meanLum = Mean( lum)  
 medianLum = Median( lum)  
 if( Abs( meanLum – medianLum ) > attrThreshDiff ) {  
 exitFlag = 1  
 for( j = 0; j < numComps; j++ )  
 attrCentroid[ j ] = currAttr[ j ]  
 }  
 }  
 }  
 lum0 = centroid[ dx ][ dy ][ dz ][ 0 ]  
 }  
 }  
 }  
 }

* If exitFlag is equal to 0, the following applies:

for( dx = 0; dx < 2; dx++ ) {  
 for( dy = 0; dy < 2; dy++ ) {  
 for( dz = 0; dz < 2; dz++ ) {  
 tmpIdx = idxArr[ dx ][ dy ][ dz ]  
 if( tmpIdx != currentCell1DIdx ) {  
 for( m = 0; m < cellCnt[ tmpIdx ]; m++ )  
 lum[ m ] = cellLum[ tmpIdx ][ m ]  
 if( cellCnt[ tmpIdx ] > 0 ) {  
 for( j = 0; j < numComps; j++ )  
 centroid[ dx ][ dy ][ dz ][ j ]  
  = attrCenterGrid[ tmpIdx ][ j ] ÷ cellCnt[ tmpIdx ]  
 centroid0 = centroid[ dx ][ dy ][ dz ][ 0 ]  
 if( Abs( lum0 − centroid0 ) > attrThresholdVariation )  
 for( j = 0; j < numComps; j++ )  
 centroid[ dx ][ dy ][ dz ][ j ] = currAttr[ j ]  
 if( cellCnt[ tmpIdx ] > 1 ) {  
 meanLum = Mean( lum )  
 medianLum = Median( lum )  
 if( Abs( meanLum – medianLum ) > attrThreshDiff )  
 for( j = 0; j < numComps; j++ )  
 centroid[ dx ][ dy ][ dz ][ j ] = currAttr[ j ]  
 }  
 } else {  
 for( j = 0; j < numComps; j++ )  
 centroid[ dx ][ dy ][ dz ][ j ] = currAttr[ j ]  
 }  
 }  
 }  
 }  
 }

* If exitFlag is equal to 0, subclause H.12.7.2 is invoked with attrGridSize, numBoundaryCells, recPcGeo, centroid, cellAnchor, idxArr, and cellCnt and the outputs are attrCentroid, a filtered version of the current reconstructed attribute value for attribute with index attrIdx.
* The following applies:

distToCentroid = Abs( attrCentroid[ 0 ] – currAttr[ 0 ] )  
 if( distToCentroid > attrSmoothingThreshold )  
 for( j = 0; j < numComps; j++ )  
 recPcAttrSmoothed[ aIdx ][ n ][ j ] = attrCentroid[ j ]

}

* + 1. Identification of the geometry and attribute center grids
       1. Identification of the geometry center grid

This process divides the 3D point cloud space into cells of uniform size and identifies cells containing boundary points and cells which are adjacent to such cells.

Inputs to this process are:

* geoGridSize, indicating the geometry smoothing grid size,
* pointCnt, the number of points in the current reconstructed point cloud frame,
* isBoundaryPoint[ n ], n = 0..pointCnt – 1, indicating whether the n-th reconstructed point lies on or near a patch boundary,
* recPcGeo[ n ][ k ], i = 0..pointCnt – 1, k = 0..2, an array containing the reconstructed point positions for the current point cloud frame, and
* pointToPatch[ n ], n = 0..pointCnt – 1, an array containing the index of the patch that the n-th reconstructed point, recPcGeo[ n ], belongs to.

Outputs of this process are:

* numCells1D, the number of cells in x, y, or z direction,
* numBoundaryCells, the number of geometry smoothing grid cells containing a boundary point or belonging to a 2 × 2 × 2 neighbourhood of the cells containing boundary points,
* cellIdxLut[ x ][ y ][ z ], for x, y, and z in the range of 0 to numCells1D − 1, inclusive, an array mapping a 3D cell index to a 1D index in the range of 0..numBoundaryCells – 1,
* centerGrid[ i ][ k ], i = 0..numBoundaryCells – 1 and k = 0..2, an array containing reconstructed geometry center grid positions,
* cellCnt[ i ], i = 0..numBoundaryCells – 1, an array containing number of reconstructed points in each cell, and
* cellDoSmoothing[ i ], i = 0..numBoundaryCells – 1, a boolean array indicating whether smoothing should be performed for a cell.

The following applies:

* Subclause H.12.4.3 is invoked with variables geoGridSize and pointCnt, and arrays isBoundaryPoint and recPcGeo as inputs and the outputs are variables numCells1D and numBoundaryCells, and an array cellIdxLut.
* The arrays cellDoSmoothing, cellCnt, cellPatchIdx, centerGrid, and cellPatchIdx are initialized as follows:

for( i = 0; i < numBoundaryCells; i ++ ) {  
 cellDoSmoothing[ i ] = 0  
 cellCnt[ i ] = 0  
 cellPatchIdx[ i ] = 0  
 centerGrid[ i ][ 0 ] = 0  
 centerGrid[ i ][ 1 ] = 0  
 cellPatchIdx[ i ] = −1  
}

* For n = 0..pointCnt – 1, the following applies:
* Variables x, y, z, and cellIndex are derived as follows:

x = recPcGeo[ n ][ 0 ] / geoGridSize  
y = recPcGeo[ n ][ 1 ] / geoGridSize  
z = recPcGeo[ n ][ 2 ] / geoGridSize  
cellIndex = cellIdxLut[ x ][ y ][ z]

* If cellIndex is not equal to −1, the following applies:
* If cellCnt[ cellIndex ] is equal to 0, cellPatchIdx[ cellIndex ] is set to pointToPatch[ n ].
* Otherwise, if cellDoSmoothing[ cellIndex ] is equal to 0 and cellPatchIdx[ cellIndex ] is not equal to pointToPatch[ n ], cellDoSmoothing[ cellIndex ] is set to 1.
* The following applies:

for( k = 0; k < 3 ; k++ )  
 centerGrid[ cellIndex ][ k ] += recPcGeo[ n ][ k ]  
cellCnt[ cellIndex ]++

* For i = 0..numBoundaryCells – 1 the following applies:

if( cellCnt[ i ]> 0 )  
 for( k = 0; k < 3 ; k++ )  
 centerGrid[ i ][ k] = centerGrid[ i ][ k ] ÷ cellCnt[ i ]

* + - 1. Identification of the attribute center grid

Inputs to this process are:

* a variable numComps, indicating the number of components for the current attribute
* a variable attrGridSize, indicating the attribute smoothing grid size,
* a variable pointCnt, indicating the number of points in the current reconstructed point cloud frame,
* a 1D array isBoundaryPoint, of size pointCnt, indicating whether the n-th reconstructed point lies on or near a patch boundary,
* a 2D array recPcGeo, of size pointCnt × 3, containing the reconstructed, possibly smoothed, point positions,
* a 2D array recPcAttr, of size pointCnt × numComps, containing the reconstructed attribute values of the current attribute
* a 1D array pointToPatch of size pointCnt, indicating the index of the patch that each reconstructed point in recPcGeo belongs to.

Outputs of this process are:

* a variable numCells1D, indicating the number of attribute smoothing grid cells in x, y, or z direction,
* a variable numBoundaryCells, indicating the number of attribute smoothing grid cells containing a boundary point or belonging to a 2 × 2 × 2 neighbourhood of the cells containing boundary points,
* a 3D array cellIdxLut of size numCells1D × numCells1D × numCells1D, mapping a 3D cell index to a 1D index,
* centerGrid[ i ][ j ], i = 0..numBoundaryCells – 1 and j = 0..numComps – 1, an array containing reconstructed attribute center grid positions,
* cellCnt[ i ], i = 0..numBoundaryCells – 1, an array containing the number of reconstructed points in each cell,
* cellDoSmoothing[ i ], i = 0..numBoundaryCells – 1, a boolean array indicating whether smoothing should be performed for a cell, and
* cellLum[ i ][ m ], i = 0..numBoundaryCells – 1 and m = 0..cellCnt[ i ] – 1, the first attribute component, for attribute with index attrIdx, of each point belonging to the boundary cell.

The following applies:

* Subclause H.12.4.3 is invoked with variables attrGridSize and pointCnt, and arrays isBoundaryPoint and recPcGeo as inputs and the outputs are variables numCells1D and numBoundaryCells, and array cellIdxLut.
* The arrays cellDoSmoothing, cellCnt, cellPatchIdx,centerGrid, and cellPatchIdx are initialized as follows:

for( i = 0; i < numBoundaryCells; i ++ ) {  
 cellDoSmoothing[ i ] = 0  
 cellCnt[ i ] = 0  
 cellPatchIdx[ i ] = 0  
 centerGrid[ i ][ 0 ] = 0  
 centerGrid[ i ][ 1 ] = 0  
 cellPatchIdx[ i ] = −1  
}

* For n = 0..pointCnt – 1, the following applies:
* Variables x, y, z, and cellIndex are derived as follows:

x = recPcGeo[ n ][ 0 ] / attrGridSize  
y = recPcGeo[ n ][ 1 ] / attrGridSize  
z = recPcGeo[ n ][ 2 ] / attrGridSize  
cellIndex = cellIdxLut[ x ][ y ][ z ]

* If cellIndex is not equal to −1, the following applies:
* If cellCnt[ cellIndex ] is equal to 0, cellPatchIdx[ cellIndex ] is set equal to pointToPatch[ n ].
* Otherwise, if cellDoSmoothing[ cellIndex ] is equal to 0 and cellPatchIdx[ cellIndex ] is not equal to pointToPatch[ n ], cellDoSmoothing[ cellIndex ] is set to 1.
* The following applies:

for( j = 0; j < numComps ; j++ )  
 centerGrid[ cellIndex ][ j ] += recPcAttr[ n ][ j ]  
 cellLum[ cellIndex ][ cellCnt[ cellIndex ] ] = recPcAttr[ n ][ 0 ]  
 cellCnt[ cellIndex ]++

* For i = 0..numBoundaryCells – 1 the following applies:

if( cellCnt[ i ]> 0 )  
 for( j = 0; j < numComps ; j++ )  
 centerGrid[ i ][ j ] = centerGrid[ i ][ j ] ÷ cellCnt[ i ]

* + - 1. Generation of an 1D index lookup table

Inputs to this process are:

* a variable gridSize, indicating the size of the cell in the x, y, or z direction,
* a variable pointCnt, indicating the number of reconstructed points in the current point cloud frame,
* isBoundaryPoint[ n ], n = 0..pointCnt – 1, indicating whether the n-th reconstructed point lies on or near a patch boundary, and
* recPcGeo[ n ][ k ], n = 0..pointCnt – 1, k = 0..2, an array containing the reconstructed, possibly smoothed, point positions.

Outputs of this process are:

* numCells1D, the number of cells in x, y or z direction,
* numBoundaryCells, the number of smoothing grid cells containing a boundary point or belonging to a 2 × 2 × 2 neighbourhood of the cells containing boundary points, and
* cellIdxLut[ x ][ y ][ z ], for x, y, and z in the range of 0..numCells1D − 1, an array mapping a 3D cell index to a 1D index in the range of 0..numBoundaryCells – 1.

The following applies:

* The three-dimensional geometry coordinate space is divided into cells of size gridSize × gridSize × gridSize. The variable numCells1D is derived as follows:

for( k = 0; k < 3; k++ )  
 maxG[ k ] = recPcGeo[ 0 ][ k ]

for( i = 1; i < pointCnt ; i++ )  
 for( k = 0; k < 3 ; k++ )  
 maxG[ k ] = Max(maxG[ k ], recPcGeo[ i ][ k ] )  
 numCells1D = ( Max( maxG[ 0 ], Max( maxG[ 1 ], maxG[ 2 ] ) ) / gridSize ) + 1

Only some of the cells in the numCells1D × numCells1D × numCells1D grid affect geometry or attribute smoothing. So, for storing information specific to such cells, this process creates a lookup table from the 3D cell index into a smaller 1D cell structure which only stores information about cells that affect geometry or attribute smoothing.

An array cellIdxLut of size numCells1D × numCells1D × numCells1D is initialized to −1 and a variable numBoundaryCells is initialized to 0.

The following applies:

halfGridSize = Max( gridSize / 2, 1 )  
 maxCoord = gridSize \* numCells1D  
 for( n = 0; n < pointCnt; n++ ) {  
 if( isBoundaryPoint[ n ] > 0 ) {  
 x = recPcGeo[ n ][ 0 ]  
 y = recPcGeo[ n ][ 1 ]  
 z = recPcGeo[ n ][ 2 ]  
  
 if(!( x < halfGridSize || maxCoord <= x + halfGridSize ||  
 y < halfGridSize || maxCoord <= y + halfGridSize ||  
 z < halfGridSize || maxCoord <= z + halfGridSize ) ) {  
 x2 = x / gridSize  
 y2 = y / gridSize  
 z2 = z / gridSize  
  
 x3 = x % gridSize  
 y3 = y % gridSize  
 z3 = z % gridSize  
  
 qx = x2 + ( ( x3 < gridSize / 2 ) ? −1 : 0 )  
 qy = y2 + ( ( y3 < gridSize / 2 ) ? −1 : 0 )  
 qz = z2 + ( ( z3 < gridSize / 2 ) ? −1 : 0 )  
  
 for( ix = qx; ix < qx + 2; ix++ ) {  
 for( iy = qy; iy < qy + 2; iy++ ) {  
 for( iz = qz; iz < qz + 2; iz++ ) {  
 if( cellIdxLut[ ix ][ iy ][ iz ] == −1 ) {  
 cellIdxLut[ ix ][ iy ][ iz ] = numBoundaryCells  
 numBoundaryCells++  
 }  
 }  
 }  
 }  
 }  
 }  
 }

NOTE – This process allows various arrays such as cell centroids and smoothing flags to be stored with reduced memory requirements. However, this process is not normative. A conformant decoder may choose to store the full three dimensional arrays or use a different technique for reducing the memory requirements.

* + 1. Identification of boundary points

This process identifies a point on the atlas as a boundary point if it lies near the boundary of the patches. Geometry or attribute smoothing should be applied only to points which correspond to such boundary points.

Inputs to this process are:

* a 2D array oFrame, of size asps\_frame\_height × asps\_frame\_width, corresponding to an occupancy frame,
* y, the row index of a reconstructed point on the current atlas, and
* x, the column index of a reconstructed point on the current atlas.

Output of this process is:

* boundaryPoint, a boolean variable indicating whether geometry or attribute smoothing should be applied to the reconstructed point corresponding to atlas coordinates ( y, x ).

A variable boundaryPoint identifies whether a point on the atlas with coordinates ( y, x ) is located near a patch boundary.

The following applies:

boundaryPoint = 0  
 if( ( x == 0 ) || ( x == asps\_frame\_width – 1 ) || ( y == 0 ) || ( y == asps\_frame\_height – 1 ) )  
 boundaryPoint = 1  
 else {  
 if( ( oFrame[ y ][ x − 1 ] == 0 ) || ( oFrame[ y ][ x + 1 ] == 0 ) ||  
 ( oFrame[ y – 1 ][ x ] == 0 ) || ( oFrame[ y + 1 ][ x ] == 0 ) ||  
 ( oFrame[ y – 1 ][ x − 1 ] == 0 ) || ( oFrame[ y – 1 ][ x + 1 ] == 0 ) ||  
 ( oFrame[ y + 1 ][ x − 1 ] == 0 ) || ( oFrame[ y + 1 ][ x + 1 ] == 0 ) )  
 boundaryPoint = 1

if( boundaryPoint != 1 ) {  
 if( ( x == 1 ) || ( x == asps\_frame\_width – 2 ) ||( y == 1 ) || ( y == asps\_frame\_height – 2 ) )  
 boundaryPoint = 1  
 else if( ( oFrame[ y – 2 ][ x − 2 ] == 0 ) || ( oFrame[ y – 2 ][ x − 1 ] == 0 ) ||  
 ( oFrame[ y – 2 ][ x ] == 0 ) || ( oFrame[ y – 2 ][ x + 1 ] == 0 ) ||  
 ( oFrame[ y – 2 ][ x + 2 ] == 0 ) || ( oFrame[ y – 1 ][ x − 2 ] == 0 ) ||  
 ( oFrame[ y – 1 ][ x + 2 ] == 0 ) || ( oFrame[ y ][ x − 2 ] == 0 ) ||  
 ( oFrame[ y ][ x + 2 ] == 0 ) || ( oFrame[ y + 1 ][ x − 2 ] == 0 ) ||  
 ( oFrame[ y + 1 ][ x + 2 ] == 0 ) || ( oFrame[ y + 2 ][ x − 2 ] == 0 ) ||  
 ( oFrame[ y + 2 ][ x − 1 ] == 0 ) || ( oFrame[ y + 2 ][ x ] == 0 ) ||  
 ( oFrame[ y + 2 ][ x + 1 ] == 0 ) || ( oFrame[ y + 2 ][ x + 2 ] == 0 ) )  
 boundaryPoint = 1  
 }  
 }

* + 1. Identification of 2 × 2 × 2 cells corresponding to a reconstructed point position

Inputs to this process are:

* numCells1D, the number of cells in x, y, or z direction,
* numBoundaryCells, the number of cells containing a boundary point or a cell neighbouring such a cell,
* recPos3D[ k ], k = 0..2, an array containing the position of the current reconstructed point,
* cellIdxLut[ x ][ y ][ z ], for x, y, and z in the range of 0..numCells1D − 1, an array mapping a 3D cell index to a 1D index in the range of 0..numBoundaryCells – 1,
* cellCnt[ n ], n = 0..numBoundaryCells – 1, an array containing the number of reconstructed points in a cell, and
* cellDoSmoothing[ n ], n = 0..numBoundaryCells – 1, a boolean array indicating whether smoothing should be performed for a cell.

Outputs of this process are:

* otherClusterPtCnt, a boolean variable indicating whether tri-linear filtering should be performed for the cell to which the current reconstructed point belongs to,
* currentCell1DIdx, a variable specifying the 1D index for the cell to which the reconstructed point recPos3D belongs to,
* a 3D array idxArr, of size 2 × 2 × 2, specifying the 1D indices for the 2 × 2 × 2 grid of cells, and
* a 1D array, cellAnchor, of size 3, specifying the grid indices corresponding to the origin of 2 × 2 × 2 grid of cells in x, y, and z direction, respectively.

The 2 × 2 × 2 grid of cells neighbouring the reconstructed point recPos3D[ k ], k = 0..2, is identified as follows:

for( k = 0; k < 3 ; k++ ) {  
 idxCell[ k ] = ( recPos3D[ k ] / gridSize )  
 cellAnchor[ k ] = idxCell[ k ] + ( ( ( recPos3D[ k ] % gridSize ) < ( gridSize / 2 ) ) ? –1 : 0 )  
 }  
 currentCell1DIdx = cellIdxLut[ idxCell[ 0 ] ][ idxCell[ 1] ][ idxCell[ 2 ] ]  
  
 otherClusterPtCnt = 0  
 for( dx = 0; dx < 2; dx++ ) {  
 for( dy = 0; dy < 2; dy++ ) {  
 for( dz = 0; dz < 2; dz++ ) {  
 xIdx = cellAnchor[ 0 ] + dx  
 yIdx = cellAnchor[ 1 ] + dy  
 zIdx = cellAnchor[ 2 ] + dz  
 tmpIdx = cellIdxLut[ xIdx ][ yIdx ][ zIdx ]  
 idxArr[ dx ][ dy ][ dz ] = tmpIdx  
 if( cellDoSmoothing[ tmpIdx ] && cellCnt[ tmpIdx ] )  
 otherClusterPtCnt = 1  
 }  
 }  
 }

* + 1. Trilinear filtering
       1. Trilinear filtering for geometry smoothing

This process carries out trilinear filtering for geometry smoothing.

Inputs to this process are:

* gridSize, the size of the cell in x, y or z direction,
* numBoundaryCells, the number of cells containing a boundary point or neighbouring such a cell,
* recPos3D[ k ], k =0..2, position of a reconstructed point,
* centroid[ x ][ y ][ z ][ k ], for x, y, and z in the range of 0..1, k =0..2, an array specifying the geometry centroids for the 2 × 2 × 2 grid of cells,
* cellAnchor[ k ], k = 0..2, an array specifying the grid indices corresponding to the origin of 2 × 2 × 2 grid of cells in x, y, and z direction, respectively,
* idxArr[ dx ][ dy ][ dz ], for dx, dy, and dz in the range of 0..1, an array specifying the indices into 1D lookup table for the 2 × 2 × 2 grid of cells, and
* cellCnt[ j ], j = 0..numBoundaryCells – 1, an array containing the number of reconstructed points in a cell.

Outputs of this process are:

* recPos3DFilt[ k ], k = 0..2, a filtered version of the reconstructed point recPos3D[ k ] and
* cnt, a weighted average of the number of points from the 2 × 2 ×2 grid of cells.

The following applies:

* The weights for trilinear filtering are calculated as follows:

for( k = 0; k < 3 ; k++ ) {  
 w[ k ][ 1 ] = ( recPos3D[ k ] – ( cellAnchor[ k ] \* gridSize ) – ( gridSize / 2 ) ) \* 2 + 1  
 w[ k ][ 0 ] = ( gridSize \* 2 ) – w[ k ][ 1 ]  
 }

* The trilinear filtering is performed as follows:

cnt = 0  
for( k = 0; k < 3; k++ )  
 filtVal[ k ] = 0  
for( dx = 0; dx < 2; dx++ ) {  
 wx = w[ 0 ][ dx ]  
 for( dy = 0; dy < 2; dy++ ) {  
 wy = w[ 1 ][ dy ]  
 for( dz = 0; dz < 2; dz++ ) {  
 wz = w[ 2 ][ dz ]  
 tmpIdx = idxArr[ dx ][ dy ][ dz ]  
 for( k = 0; k < 3; k++ )  
 filtVal[ k ] += wx \* wy \* wz \* centroid[dx ][ dy ][ dz ][ k ]  
 cnt += wx \* wy \* wz \* cellCnt[ tmpIdx ]  
 }  
 }  
}  
cnt = cnt ÷ ( 8 \* gridSize \* gridSize \* gridSize )  
for( k = 0; k < 3; k++ )  
 recPos3DFilt[ k ] = filtVal[ k ] ÷ ( 8 \* gridSize \* gridSize \* gridSize )

* + - 1. Trilinear filtering for attribute smoothing

This process carries out trilinear filtering for attribute smoothing.

Inputs to this process are:

* attrGridSize, the size of the cell in x, y or z direction,
* numComps, number of components for the attribute
* numBoundaryCells, the number of cells containing a boundary point or neighbouring such a cell,
* recPos3D[ k ], k =0..2, position of a reconstructed point,
* centroid[ x ][ y ][ z ][ j ], for x, y, and z in the range of 0..1, and j = 0..numComps − 1, an array specifying the attribute centroids on the 2 × 2 × 2 grid of cells,
* cellAnchor[ k ], k = 0..1, an array specifying the index of anchor of the 2 × 2 × 2 grid of cells in x, y, and z directions, respectively,
* currentCell1DIdx, 1D index of the cell to which the reconstructed point recPos3D belongs to,
* idxArr[ dx ][ dy ][ dz ], dx, dy, and dz in the range of 0 to 1, inclusive, an array specifying the 1D indices for the 2 × 2 × 2 grid of cells, and
* cellCnt[ i ], i = 0..numBoundaryCells – 1, an array containing number of reconstructed points in a cell.

Output of this process is:

* attr3dFilt[ j ], j = 0..numComps − 1, a filtered version of the reconstructed attribute.

The following steps apply:

* The weights for trilinear filtering are calculated as follows:

for( k = 0; k < 3 ; k++ ) {  
 w[ k ][ 1 ] = ( recPos3D[ k ] – ( cellAnchor[ k ] \* attrGridSize) – ( attrGridSize / 2 ) ) \* 2 + 1  
 w[ k ][ 0 ] = ( attrGridSize \* 2 ) – w[ k ][ 1 ]  
 }

* The trilinear filtering is performed as follows:

for( j = 0; j < numComps; j++ )  
 filtVal[ j ] = 0  
 for( dx = 0; dx < 2; dx++ ) {  
 wx = w[ 0 ][ dx ]  
 for( dy = 0; dy < 2; dy++ ) {  
 wy = w[ 1 ][ dy ]  
 for( dz = 0; dz < 2; dz++ ) {  
 wz = w[ 2 ][ dz ]  
 tmpIdx = idxArr[ dx ][ dy ][ dz ]  
 for( j = 0; j < numComps; j++ )  
 filtVal[ j ] += wx \* wy \* wz \* centroid[ dx ][ dy ][ dz ][ j ]  
 }  
 }  
 }  
 for( j = 0; j < 3; j++ )  
 attr3DFilt[ j ] = filtVal[ j ] ÷ ( 8 \* attrGridSize \* attrGridSize \* attrGridSize )

* + 1. Attribute transfer
       1. General

This process updates the attribute values of the reconstructed points whose geometry position has been modified by another process after reconstruction, such as the geometry smoothing process defined in subclause H.12.2.

The inputs to this process are:

* a variable pointCnt, specifying the number of points in the current reconstructed point cloud frame,
* a 1D array isUpdatedPoint of size pointCnt, indicating whether a point was changed by a geometry modification operation,
* a pointCnt × 3 2D array recPcGeo of size pointCnt × 3, containing the geometry positions of the points in the current reconstructed point cloud frame before geometry modification,
* a pointCnt × 3 2D array recPcGeoMod , containing the geometry positions of the points in the current reconstructed point cloud frame after geometry modification,
* a 3D array recPcAttr, containing the reconstructed attribute values for each point in the current reconstructed point cloud frame associated with the atlas with atlas ID equal to PostRecAtlasID, and
* a 2D array bitDepth[ i ], containing the bit depth for each dimension of the i-th attribute.

Outputs of this process are:

* a 3D array recPcAttrMod[ n ][ i ][ j ], containing the updated attribute values for each point in the current reconstructed point cloud frame associated with the atlas with atlas ID equal to PostRecAtlasID, and
* a 2D array bitDepthMod[ i ], containing the bit depth for each dimension of the i-th attribute after modification due to the attribute transfer process,

where j is in the range of 0 to ai\_attribute\_dimension\_minus1[ PostRecAtlasID ][ i ], inclusive and i is in the range of 0 to ai\_attribute\_count[ PostRecAtlasID ] – 1, inclusive.

If the bit depths of different dimensions of an attribute are not the same, the attribute values should be normalized to the same bit depth, e.g. the highest bit depth.

For i = 0..ai\_attribute\_count[ PostRecAtlasID ] – 1,the following process is performed:

* A 2D recPcAttr0 is derived as follows:

for( n = 0; n < pointCnt; n++ ) {  
 for( j = 0; j <= ai\_attribute\_dimension\_minus1[ PostRecAtlasID ][ i ]; j++ ) {  
 recPcAttr0[ n ][ j ] = recPcAttr[ n ][ i ][ j ]  
 }  
 }

* Subclause H.12.8.2 is invoked with variables pointCnt and i and bitDepth[ i ], and arrays isUpdatedPoint, recPcGeo, recPcGeoMod, recPcAttr0, and as inputs. The outputs of this process are an array recPcAttrMod0, which contains the updated attribute values of the points in the current reconstructed point cloud frame after attribute transfer.
* A 3D array recPcAttrMod is derived as follows:

for( n = 0; n < pointCnt; n++ ) {  
 for( j = 0; j <= ai\_attribute\_dimension\_minus1[ PostRecAtlasID ][ i ]; j++ ) {  
 recPcAttrMod[ n ][ i ][ j ] = recPcAttrMod0[ n ][ j ]  
 }  
 }

* A 2D array bitDepthMod is derived as follows:

for( i = 0; i < ai\_attribute\_count[ PostRecAtlasID ]; i++ ) {  
 bitDepthMod[ i ] = bitDepth[ i ]  
 }

* + - 1. Process for attribute transfer

This process updates the attribute values of the reconstructed points whose geometry position has been modified by another process after reconstruction, such as the geometry smoothing process defined in subclause H.12.2.

The inputs to this process are:

* a variable pointCnt, specifying the number of points in the current reconstructed point cloud frame,
* a variable i, specifying the index of the current attribute to be processed,
* a variable bitDepth, containing the bit depth for the i-th attribute.
* a 1D array isUpdatedPoint of size pointCnt, indicating whether the geometry position of a point was modified by another process after reconstruction,
* a pointCnt × 3 2D array recPcGeo, containing the geometry positions of the points in the current reconstructed point cloud frame prior to any modification,
* a pointCnt × 3 2D array recPcGeoMod, an array containing the geometry positions of the points in the current reconstructed point cloud frame after any modification, and
* a 2D array recPcAttr[ n ][ j ], containing the reconstructed attribute values in the current reconstructed point cloud frame associated with the atlas with atlas ID equal to PostRecAtlasID.

Output of this process is:

* a 2D array recPcAttrMod[ n ][ j ], containing the updated attribute values for each point in the current reconstructed point cloud frame associated with the atlas with atlas ID equal to PostRecAtlasID,

where n = 0..pointCnt – 1 and j = 0..ai\_attribute\_dimension\_minus1[ PostRecAtlasID ][ i ].

Let the process Assign( arraySrc, sizeSrc ) be defined as follows:

Assign( arraySrc, sizeSrc ) {  
 for( c = 0; c < sizeSrc; c++ ) {  
 arrayDst[ c ] = arraySrc[ c ]  
 }  
 return arrayDst  
 }

The following applies:

* A variable attrDimension be initialized to ai\_attribute\_dimension\_minus1[ PostRecAtlasID ][ i ].
* A 1D array forwardSet of size pointCnt, is initialized to 0.
* A pointCnt × 3 2D array forwardCentroid, which indicates a weighted average, i.e. the centroid, of the attribute values of a set of neighbouring points, is calculated as follows:
* 2D arrays forwardDistance and forwardSet, of size pointCnt × maxNeighbour are initialized to 0.

A forward search is performed for n = 0..pointCnt – 1 as follows:

* A variable maxNeighbour is set to 8.
* If isUpdatedPoint[ n ] is equal to 1, the point neighbour search process defined in subclause H.12.8.4 is invoked with variables maxNeighbour and pointCnt, and arrays recPcGeoMod[ n ] and recPcGeo. The outputs are a variable neighbourPoints, and arrays neighbourSet and distanceSet.
* Arrays forwardSet[ n ] and forwardDistance[ n ] are updated as follows:

forwardSet[ n ] = Assign( neighbourSet, neighbourPoints)  
 forwardDistance[ n ] = Assign( distanceSet, neighbourPoints)

* The array isForwardSet is modified as follows:

for( r = 0; r < neighbourPoints; r++)  
 isForwardSet[ forwardSet[ n ][ r ] ] = 1

* The array forwardCentroid[ n ] is calculated as follows:

if( forwardDistance[ n ][ 0 ] == 0 )  
 forwardCentroid[ n ] = Assign( recPcAttr[ forwardSet[ n ][ 0 ] ], attrDimension )  
 else {  
 if( neighbourPoints == 1 ) {  
 forwardCentroid[ n ]  
  = Assign( recPcAttr[ forwardSet[ n ][ 0 ] ], attrDimension )  
 } else {   
 sumWeights = 0  
 for( r = 0; r < neighbourPoints[ n ]; r++ ) {  
 sqrtDistance[ r ] = Sqrt( forwardDistance[ n ][ r ] ) + 4  
 sumWeights += 1 ÷ sqrtDistance[ r ]  
 }  
 for( j = 0; j < attrDimension; j++ ) {  
 centroid = 0  
 for( r = 0; r < neighbourPoints; r++ ) {  
 tmp = recPcAttr[ forwardSet[ n ][ r ] ][ j ]  
 centroid += tmp ÷ ( Sqrt( forwardDistance[ n ][ r ] ) + 4 )  
 }  
 forwardCentroid[ n ][ i ][ j ] = centroid ÷ sumWeights  
 }  
 }  
 }

After the completion of the forward search process, a backward search is performed to determine a 2D array backwardCentroid of size pointCnt × 3, as follows:

2D Arrays backwardSet and backwardDistance and the 1D array backwardNeighbourPoints are initialized to 0.

For m = 0..pointCnt – 1 the following applies:

* If isForwardSet[ m ] is equal to 1, the point neighbour search process defined in subclause H.12.8.4 is invoked with variables maxNeighbourBackward and pointCnt, and arrays recPcGeo[ m ] and recPcGeoMod as inputs and the outputs are the updated variable neighbourPoints and the updated arrays neighbourSet and distanceSet.
* The arrays backwardSet, backwardDistance, and backwardNeighbourPoints are updated as follows:

tmpIdx = neighbourSet[ 0 ]  
 cnt = backwardNeighbourPoints[ tmpIdx ]  
 backwardSet[ tmpIdx ][ cnt ] = m  
 backwardDistance[tmpIdx ][ cnt ] = distanceSet[ 0 ]  
 backwardNeighbourPoints[ tmpIdx ] ++

The pointCnt × 3 2D array backwardCentroid is calculated as follows:

for( n = 0; n < pointCnt; n++ ) {  
 if( backwardNeighbourPoints[ n ] == 0 )  
 continue  
 if( backwardNeighbourPoints[ n ] == 1 ) {  
 backwardCentroid[ n ] =  
  assign( recPcAttr[ backwardSet[ n ][ 0 ] ], attrDimension ) } else {  
 sumWeights = 0  
 for( r = 0; r < backwardNeighbourPoints[ n ]; r++ ) {  
 sqrtDistance[ r ] = Sqrt( backwardDistance[ n ][ r ] ) + 4  
 sumWeights += 1 ÷ sqrtDistance[ r ]  
 }  
 for( j = 0; j < ai\_attribute\_dimension\_minus1[ PostRecAtlasID ][ i ]; j++ ) {  
 centroid = 0  
 for( r = 0; r < backwardNeighbourPoints[ n ]; r++ ) {  
 tmp = recPcAttr[ backwardSet[ n ][ r ] ][ i ][ j ]  
 centroid += tmp ÷ sqrtDistance[ r ]  
 }  
 backwardCentroid[ n ][ i ][ j ] = centroid ÷ sumWeights  
 }  
 }  
 }

The array recPcAttrMod[ n ] is calculated as follows:

* If isUpdatedPoint[ n ] is equal to 1 and backwardNeighbourPoints[ n ] is equal to 0 then:

recPcAttrMod[ n ] = Assign(forwardCentroid[ n ], attrDimension )

* Otherwise if isUpdatedPoint[ n ] is equal to 1

recPcAttrMod[ n ] = Assign( backwardCentroid[ n ], attrDimension )

* Otherwise (isUpdatedPoint[ n ] is equal to 0)

recPcAttrMod[ n ] = Assign( recPcAttr[ n ][ c ], attrDimension )

* + - 1. Process to compute the distance between two arrays

Let the function Distance( arrayA, arrayB, size ) be defined as follows:

Distance( arrayA, arrayB, size ) {  
 dist = 0  
 for( k = 0; k < size; k++ ) {  
 dist += ( arrayA[ k ] − arrayB[ k ] ) \* ( arrayA[ k ] − arrayB[ k ] )  
 }  
 return dist  
 }

* + - 1. Point neighbour search process

Inputs to this process are:

* a variable N, specifying the maximum number of neighbouring points,
* a variable numPoints, specifying the number of points in the array pointSet,
* a 1D array pointA of size 3, specifying a point in the 3D space for which neighbours are computed, and
* a numPoints × 3 2D array pointSet, containing the geometry positions of points to be considered for neighbour search.

Outputs of this process are:

* a variable numNeighbourPoints, specifying the number of nearest points in the set neighbourSet,
* a sorted 1D array neighbourSet, of size numNeighbourPoints, containing the indices of points in pointSet, arranged in increasing order of distance from pointA, and
* a sorted 1D array distanceSet, of size numNeighbourPoints, specifying the distance of each entry in neighbourSet from pointA.

The following applies:

* If numPoints is less than N, then N is set equal to numPoints.
* The distance between each point in the array pointSet and point A is calculated and stored in a 1D array distancePoints, of size numPoints, as follows:

for( n = 0; n < numPoints; n++ )  
 distancePoints[ n ] = Distance( pointSet[ n ], pointA, 3 )

* Finally, the N smallest distance values in the array distancePoints and their corresponding indices in distanceSet are assigned to arrays distanceSet and neighbourSet, respectively. How the sorting is performed is outside the scope of this document. The variable neighbourPoints is set equal to N.

NOTE – This process could be performed much faster using methods that do not require the computation of the distance for all points.

* 1. Adaptation process

The inputs to this process are:

* pointCnt, the number of points in the current reconstructed point cloud frame,
* an array pointCloudGeo[ n ][ k ], n = 0..pointCnt – 1, k = 0..2, containing the positions of points in the current reconstructed point cloud frame equal to:
* recPcGeo[ n ][ k ], specified in H.11, or
* recPcGeoSmoothed[ n ][ k ], specified in H.12, when geometry smoothing is applied
* vui\_anchor\_point[ k ], k = 0..2, and
* OffsetMatrix, RotationMatrix, and ScaleMatrix matrices

Output of this process is:

* RecPcGeo[ n ], n = 0..pointCnt – 1, an array containing the list of points in the reconstructed point cloud geometry frame after an adaptation process

This process specifies the transformation from the 3D model coordinate system to the 3D world coordinate system. The following applies:

for( n = 0; n < PointCnt ; n++)  
 pointCloudGeoShifted[ n ][ 3 ] = 1  
 for( k = 0 ; k < 3 ; k++ )  
 pointCloudGeoShifted[ n ][ k ] = pointCloudGeo[ n ][ k ] – vui\_anchor\_point[ k ]

for( n = 0 ; n < PointCnt ; n++ )  
 RecPcGeo[ n ] = OffsetMatrix \* RotationMatrix \* ScaleMatrix \* pointCloudGeoShifted[ n ]

* 1. Parsing process

The specifications in Clause 14 and its subclauses apply.

* 1. Profiles, tiers, and levels
     1. Overview of profiles, tiers, and levels

The specifications in subclause A.1 and its subclauses apply.

* + 1. Profile, tier, and level structure

The specifications in subclause A.2 and its subclauses apply.

* + 1. CodecGroup profile components

The specifications in subclause A.3 and its subclauses apply.

* + 1. Toolset profile components
       1. V-PCC Basic, V-PCC Basic Still, V-PCC Extended, and V-PCC Extended Still toolset profile components

V3C toolset profile components indicating the V-PCC Basic (ptl\_profile\_toolset\_idc = 0), V-PCC Basic Still (ptl\_profile\_toolset\_idc = 0 and ptc\_one\_v3c\_frame\_only\_flag = 1), V-PCC Extended (ptl\_profile\_toolset\_idc = 1), or V-PCC Extended Still (ptl\_profile\_toolset\_idc = 1 and ptc\_one\_v3c\_frame\_only\_flag = 1) toolset profile component shall conform to the syntax element restrictions specified in Table H-3. If a syntax element is not mentioned in Table H-3 it is not restricted through a toolset profile component.

Table H-3 – Max allowed syntax element values for the V-PCC toolset profile components

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Profile name** | | | |
| **Syntax element** | **V-PCC Basic** | **V-PCC  Basic Still** | **V-PCC Extended** | **V-PCC Extended Still** |
| ptl\_profile\_toolset\_idc | 0 | 0 | 1 | 1 |
| ptc\_one\_v3c\_frame\_only\_flag | ‒ | 1 | ‒ | 1 |
| asps\_eom\_patch\_enabled\_flag | 0 | | ‒ | |
| asps\_map\_count\_minus1 | Min( 1, LevelMapCount – 1 ) | | LevelMapCount – 1 | |
| vps\_multiple\_map\_streams\_present\_flag  (when vps\_map\_count\_minus1 > 0) | when present, 1 | | ‒ | |
| vps\_atlas\_count\_minus1 | 0 | | 0 | |
| asps\_plr\_enabled\_flag | 0 | | ‒ | |
| ai\_attribute\_dimension\_minus1 | 2 | | ‒ | |
| ai\_attribute\_dimension\_partitions\_minus1 | 0 | | ‒ | |
| ai\_attribute\_partition\_channels\_minus1 | ‒ | | 2 | |
| asps\_use\_eight\_orientations\_flag | 0 | | ‒ | |
| asps\_extended\_projection\_enabled\_flag | 0 | | ‒ | |
| vps\_auxiliary\_video\_present\_flag | ‒ | | ‒ | |
| vps\_occupancy\_video\_present\_flag | 1 | | 1 | |
| vps\_geometry\_video\_present\_flag | 1 | | 1 | |
| vps\_attribute\_video\_present\_flag | ‒ | | ‒ | |
| vps\_extension\_present\_flag | 0 | | 0 | |
| vps\_packing\_information\_present\_flag | 0 | | 0 | |
| vps\_miv\_extension\_present\_flag | 0 | | 0 | |
| asps\_miv\_extension\_present\_flag | 0 | | 0 | |
| aaps\_miv\_extension\_present\_flag | 0 | | 0 | |
| caf\_extension\_present\_flag | 0 | | 0 | |
| caf\_miv\_extension\_present\_flag | 0 | | 0 | |
| vuh\_unit\_type | V3C\_VPS, V3C\_AD, V3C\_OVD, V3C\_GVD, or V3C\_AVD | | V3C\_VPS, V3C\_AD, V3C\_OVD, V3C\_GVD, or V3C\_AVD | |

The following restrictions shall apply to a bitstream or a collection of V3C sub-bitstreams conforming to either the V-PCC Basic, V-PCC Basic Still, V-PCC Extended, or the V-PCC Extended Still toolset profile component:

* In bitstreams conforming to the V-PCC Basic Still or the V-PCC Extended Still profile component, the bitstream shall contain only one atlas frame with nal\_layer\_id equal to 0.
* The values of asps\_atlas\_sequence\_parameter\_set\_id and afps\_atlas\_sequence\_parameter\_set\_id shall be in the range of 0 to 15, inclusive.
* The values of afps\_atlas\_frame\_parameter\_set\_id and ath\_atlas\_frame\_parameter\_set\_id shall be in the range of 0 to 63, inclusive.
* The values of aaps\_atlas\_adaptation\_parameter\_set\_id and ath\_atlas\_adaptation\_parameter\_set\_id shall be in the range of 0 to 63, inclusive.
* In bitstreams conforming to the V-PCC Basic Still or the V-PCC Extended Still profile component, active ASPSs shall have asps\_max\_dec\_atlas\_frame\_buffering\_minus1 equal to 0 only.
* In bitstreams conforming to the V-PCC Basic profile component that do not conform to the V-PCC Basic Still profile component, ptl\_level\_idc shall not be equal to 255 (which indicates level 8.5).
* In bitstreams conforming to the V-PCC Extended profile component that do not conform to the V‑PCC Extended Still profile component, ptl\_level\_idc shall not be equal to 255 (which indicates level 8.5).
* The tier and level constraints specified in subclause H.15.6, as applicable, shall be fulfilled.
* TemporalID shall be equal to 0.
* If DecOccFullRange, DecOccColourPrimaries, DecOccTransferCharacteristics, and DecOccMatrixCoeffs are present, their values shall be set to 1, 2, 8, and 2, respectively.
* If DecGeoFullRange, DecGeoColourPrimaries, DecGeoTransferCharacteristics, and DecGeoMatrixCoeffs are present, their values shall be set to 1, 2, 8, and 2, respectively.
* If DecGeoAuxFullRange, DecGeoAuxColourPrimaries, DecGeoAuxTransferCharacteristics, and DecGeoAuxMatrixCoeffs are present, their values shall be set to 1, 2, 8, and 2, respectively.

Furthermore, the following restrictions shall apply to a bitstream or a collection of V3C sub-bitstream components conforming to the V-PCC Basic toolset profile component:

* For each decoded V3C video component output picture, associated with a specific composition time value, picCompTimeVal, there shall exist a decoded output atlas frame with a composition time value atlasCompTimeVal, where atlasCompTimeVal is equal to picCompTimeVal. Determining the composition time values picCompTimeVal and atlasCompTimeVal is outside the scope of this document.

NOTE – The above restriction implies that a decoded output picture for any V3C video component is present only if there is a corresponding atlas frame in the V3C atlas component.

* All video sub-bitstreams shall have an intra random access point (IRAP) at IRAP points of an atlas sub-bitstream.
* If asps\_pixel\_deinterleaving\_enabled\_flag is equal to 1, asps\_map\_count\_minus1 shall be equal to 0.

Decoders conforming to the V-PCC Basic toolset profile component at a specific level (identified by a specific value of ptl\_level\_idc) of a specific tier (identified by a specific value of ptl\_tier\_flag) shall be capable of decoding all bitstreams for which all of the following conditions apply:

* The bitstream is indicated to conform to the V-PCC Basic or the V-PCC Basic Still profile component.
* The bitstream is indicated to conform to a level that is not level 8.5 and is lower than or equal to the specified level.
* The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.

Decoders conforming to the V-PCC Basic Still toolset profile component at a specific level (identified by a specific value of ptl\_level\_idc) of a specific tier (identified by a specific value of ptl\_tier\_flag) shall be capable of decoding all bitstreams for which all of the following conditions apply:

* The bitstream is indicated to conform to the V-PCC Basic Still profile component.
* The bitstream is indicated to conform to a level that is not level 8.5 and is lower than or equal to the specified level.
* The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.

Decoders conforming to the V-PCC Extended toolset profile component at a specific level (identified by a specific value of ptl\_level\_idc) of a specific tier (identified by a specific value of ptl\_tier\_flag) shall be capable of decoding all bitstreams for which all of the following conditions apply:

* The bitstream is indicated to conform to the V-PCC Basic, V-PCC Basic Still, V-PCC Extended, or the V‑PCC Extended Still profile component.
* The bitstream is indicated to conform to a level that is not level 8.5 and is lower than or equal to the specified level.
* The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.

Decoders conforming to the V-PCC Extended Still toolset profile component at a specific level (identified by a specific value of ptl\_level\_idc) of a specific tier (identified by a specific value of ptl\_tier\_flag) shall be capable of decoding all bitstreams for which all of the following conditions apply:

* The bitstream is indicated to conform to the V-PCC Basic Still or V-PCC Extended Still profile component.
* The bitstream is indicated to conform to a level that is not level 8.5 and is lower than or equal to the specified level.
* The bitstream is indicated to conform to a tier that is lower than or equal to the specified tier.
  + 1. Reconstruction profile components
       1. General

The specifications in Annex A.5 and its subclauses apply with following additions.

Table H-4 provides a list of defined reconstruction profile components for V-PCC and of the allowed values for syntax element ptl\_profile\_reconstruction\_idc. Currently 3 additional reconstruction profile components, other than the Rec Unconstrained profile component, namely Rec0, Rec1, and Rec2 are defined. Other values indicated as reserved in this table are reserved for future use by ISO/IEC. The reconstruction profile components may be independent from one another.

Table H-5 provides a list of the supported reconstruction functionalities for each reconstruction profile component. The “ignored” entries in the table imply that a decoder may choose to ignore the corresponding reconstruction operations. For example, a decoder may choose to reconstruct points from RAW patches or discard them.

Table H-4 – Defined reconstruction profile components for V-PCC

|  |  |
| --- | --- |
| **ptl\_profile\_reconstruction\_idc** | **Reconstruction profile component** |
| 0 | Rec0 |
| 1 | Rec1 |
| 2 | Rec2 |
| 3..254 | Reserved |
| 255 | Rec Unconstrained |

Table H-5 – Supported reconstruction operations for each reconstruction profile component

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reconstruction Operation** | **Rec0** | **Rec1** | **Rec2** | **Rec Unconstrained** |
| chroma format conversion | see subclause B.3.5 | see subclause B.3.5 | see subclause B.3.5 |  |
| resolution upsampling | see subclause B.3.3 | see subclause B.3.3 | see subclause B.3.3 |  |
| Frame rate conversion | see subclause B.3.4 | see subclause B.3.4 | see subclause B.3.4 |  |
| Pixel deinterleaving | ignored | reconstructed, according to subclause H.11.2.3 | reconstructed, according to subclause H.11.2.3 | unconstrained |
| PLR | ignored | reconstructed, according to subclause H.11.2.4 | reconstructed, according to subclause H.11.2.4 | unconstrained |
| EOM | ignored | reconstructed, according to subclause H.11.3 | reconstructed, according to subclauseH.11.3 | unconstrained |
| Duplicate point removal | ignored | reconstructed, according to subclause H.11.2.2 | reconstructed, according to subclause H.11.2.2 | unconstrained |
| RAW patches | ignored | reconstructed, according to subclause  H.11.4 | reconstructed, according to subclause H.11.4 | unconstrained |
| Geometry Smoothing | ignored | reconstructed, according to subclause H.12.2 | ignored | unconstrained |
| Attribute Transfer | ignored | reconstructed, according to subclause H.12.8 | ignored | unconstrained |
| Attribute Smoothing | ignored | reconstructed, according to subclause H.12.4 | reconstructed, according to subclause H.12.4 | unconstrained |
| Occupancy Synthesis | ignored | ignored | reconstructed, according to subclause H.10.2 | unconstrained |

For the Rec Unconstrained reconstruction profile, the reconstruction process is outside the scope of this document.

For the Rec0 reconstruction profile, the following processes apply in the specified order:

* Post-decoding conversion of the decoded V3C video components to a nominal video format is performed, if required. The process as specified in Annex B may be used for such conversion. However, other equivalent methods could also be used.
* Point cloud reconstruction is performed using all nominal format decoded V3C components as specified in subclause H.11. Certain reconstruction operations from subclause H.11 may be ignored, as specified in Table H-5.
* Point cloud adaptation is performed on the reconstructed point cloud as specified in subclause H.13.

For the Rec1 reconstruction profile, the following processes apply in the specified order:

* Post-decoding conversion of the decoded V3C video components to a nominal video format is performed, if required. The process as specified in Annex B may be used for such conversion. However, other equivalent methods could also be used.
* Point cloud reconstruction is performed using all nominal format decoded V3C components as specified in subclause H.11.
* Point cloud post-reconstruction such as geometry smoothing, attribute transfer, and attribute smoothing, may be performed on the reconstructed point cloud as specified in subclause H.12.
* Point cloud adaptation is performed on the post-reconstructed point cloud as specified in subclause H.13.

For the Rec2 reconstruction profile, the following processes apply in the specified order:

* Post-decoding conversion of the decoded V3C video components to a nominal video format is performed, if required. The process as specified in Annex B may be used for such conversion. However, other equivalent methods could also be used.
* Point cloud pre-reconstruction, such as occupancy synthesis, is performed using all nominal format decoded V3C components, as specified in subclause H.10.
* Point cloud reconstruction is performed using the point cloud pre-reconstruction V3C components as specified in subclause H.11.
* Point cloud post-reconstruction in the form of attribute smoothing may be performed on the reconstructed point cloud as specified in subclause H.12.3, and
* Point cloud adaptation is performed on the post-reconstructed point cloud as specified in subclause H.13.
  + - 1. V-PCC reconstruction decoder conformance

Decoders conforming to a V-PCC profile with a reconstruction profile component shall be able to perform reconstruction operations required by this reconstruction profile component. Conformance is assessed at conformance point A (decoded attribute, geometry, and occupancy bitstreams together with decoded patch metadata and decoded block to patch map) and conformance point B (reconstructed point cloud).

In order to check the decoder conformance at point B, in addition to the operations specified in Table H-5 the following hypothetical reference reconstruction operations are defined:

* Hypothetical reference resolution upsampling (B.3.3)
* Hypothetical reference frame rate conversion (B.3.4)
* Hypothetical reference chroma format conversion (B.3.5)

In addition, the following applies:

* The values of the reconstructed points at the output of the reconstruction stage (subclause H.11) are clipped to the range 0 to ( 1 << ( asps\_geometry\_3d\_bit\_depth\_minus1 + 1 ) – 1 ), inclusive.
* The input parameter attrBitDepth for subclause H.12.3 and the input parameter bitDepth for subclause H.12.8.2 are set to ( ai\_attribute\_2d\_bit\_depth\_minus1[ atlasID ][ attrIdx ] + 1 ) for attribute with index attrIdx for the atlas with atlas ID atlasID.

The hypothetical reference reconstruction operations are intended for checking decoder conformance at point B. A conformant V-PCC decoder may implement alternative processes that produce similar or better visual quality. The definition of similar visual quality is outside the scope of this document.

* + 1. Tier and Levels

The specifications in A.6 and its subclauses apply.

* 1. Post-decoding conversion to nominal video formats

The specifications in Annex B apply and its subclauses apply.

* 1. V3C Sample Stream Format

The specifications in Annex C and its subclauses apply.

* 1. NAL Sample Stream Format

The specifications in Annex D and its subclauses apply.

* 1. Atlas Hypothetical Reference Decoder

The specifications in Annex E and its subclauses apply.

* 1. Supplemental Enhancement Information
     1. General

The specifications in F.1 apply.

* + 1. SEI payload syntax
       1. General SEI message syntax

The specifications in F.2.1 apply.

* + - 1. Filler payload SEI message syntax

The specifications in F.2.2 apply.

* + - 1. User data registered by Recommendation ITU-T T.35 SEI message syntax

The specifications in F.2.3 apply.

* + - 1. User data unregistered SEI message syntax

The specifications in F.2.4 apply.

* + - 1. Recovery point SEI message syntax

The specifications in F.2.5 apply.

* + - 1. No reconstruction SEI message syntax

The specifications in F.2.6 apply.

* + - 1. Reserved SEI message syntax

The specifications in F.2.7 apply.

* + - 1. SEI manifest SEI message syntax

The specifications in F.2.8 apply.

* + - 1. SEI prefix indication SEI message syntax

The specifications in F.2.9 apply.

* + - 1. Active sub-bitstreams SEI message syntax

The specifications in F.2.10 apply.

* + - 1. Component codec mapping SEI message syntax

The specifications in F.2.11 apply.

* + - 1. Volumetric annotation SEI message family syntax

The specifications in F.2.12 and its subclauses apply.

* + - 1. Buffering period SEI message syntax

The specifications in F.2.13 apply.

* + - 1. Atlas frame timing SEI message syntax

The specifications in F.2.14 apply.

* + - 1. Viewport SEI messages family syntax

The specifications in F.2.15 and its subclauses apply.

* + - 1. Decoded Atlas Information Hash SEI message syntax

The specifications in F.2.16 apply.

* + - 1. Attribute transformation parameters SEI message syntax

|  |  |
| --- | --- |
| attribute\_transformation\_params( payloadSize ) { | **Descriptor** |
| **atp\_cancel\_flag** | u(1) |
| if( !atp\_cancel\_flag ) { |  |
| **atp\_num\_attribute\_updates** | ue(v) |
| for( j = 0; j < atp\_num\_attribute\_updates; j++ ) { |  |
| **atp\_attribute\_idx**[ j ] | u(8) |
| k = atp\_attribute\_idx[ j ] |  |
| **atp\_dimension\_minus1**[k] | u(8) |
| for( i = 0; i <= atp\_dimension\_minus1[ k ]; i++ ) { |  |
| **atp\_scale\_params\_enabled\_flag**[ k ][ i ] | u(1) |
| **atp\_offset\_params\_enabled\_flag**[ k ][ i ] | u(1) |
| if( atp\_scale\_params\_enabled\_flag[ k ][ i ] ) |  |
| **atp\_attribute\_scale\_minus1**[ k ][ i ] | u(32) |
| if( atp\_offset\_params\_enabled\_flag[ k ][ i ] ) |  |
| **atp\_attribute\_offset**[ k ][ i ] | i(32) |
| } |  |
| } |  |
| **atp\_persistence\_flag** | u(1) |
| } |  |
| } |  |

* + - 1. Occupancy synthesis SEI message syntax

|  |  |
| --- | --- |
| occupancy\_synthesis( payloadSize ) { | **Descriptor** |
| **os\_persistence\_flag** | u(1) |
| **os\_reset\_flag** | u(1) |
| **os\_instances\_updated** | u(8) |
| for( i = 0; i < os\_instances\_updated; i++ ) { |  |
| **os\_instance\_index**[ i ] | u(8) |
| k = os\_instance\_index[ i ] |  |
| **os\_instance\_cancel\_flag**[ k ] | u(1) |
| if( os\_instance\_cancel\_flag[ k ] != 1 ) { |  |
| **os\_method\_type**[ k ] | ue(v) |
| if( os\_method\_type[ k ] == 1 ) { |  |
| **os\_log2\_threshold\_minus1**[ k ] | u(2) |
| **os\_passes\_count\_minus1**[ k ] | u(2) |
| **os\_filter\_size\_minus1**[ k ] | u(3) |
| } |  |
| } |  |
| } |  |
| } |  |

* + - 1. Geometry smoothing SEI message syntax

|  |  |
| --- | --- |
| geometry\_smoothing( payloadSize ) { | **Descriptor** |
| **gs\_persistence\_flag** | u(1) |
| **gs\_reset\_flag** | u(1) |
| **gs\_instances\_updated** | u(8) |
| for( i = 0; i < gs\_instances\_updated; i++ ) { |  |
| **gs\_instance\_index**[ i ] | u(8) |
| k = gs\_instance\_index[ i ] |  |
| **gs\_instance\_cancel\_flag**[ k ] | u(1) |
| if( gs\_instance\_cancel\_flag[ k ] != 1 ) { |  |
| **gs\_method\_type**[ k ] | ue(v) |
| if( gs\_method\_type[ k ] == 1 ) { |  |
| **gs\_filter\_eom\_points\_flag**[ k ] | u(1) |
| **gs\_grid\_size\_minus2**[ k ] | u(5) |
| **gs\_threshold**[ k ] | u(8) |
| } |  |
| } |  |
| } |  |
| } |  |

* + - 1. Attribute smoothing SEI message syntax

|  |  |
| --- | --- |
| attribute\_smoothing( payloadSize ) { | **Descriptor** |
| **as\_persistence\_flag** | u(1) |
| **as\_reset\_flag** | u(1) |
| **as\_num\_attributes\_updated** | ue(v) |
| for( j = 0; j < as\_num\_attributes\_updated; j++ ) { |  |
| **as\_attribute\_idx**[ j ] | u(7) |
| k = as\_attribute\_idx[ j ] |  |
| **as\_attribute\_smoothing\_cancel\_flag**[ k ] | u(1) |
| **as\_instances\_updated**[ k ] | u(8) |
| for( i=0; i < as\_instances\_updated[ k ]; i++ ) { |  |
| **as\_instance\_index**[ k ][ i ] | u(8) |
| m = as\_instance\_index[ k ][ i ] |  |
| **as\_instance\_cancel\_flag**[ k ][ m ] | u(1) |
| if( as\_instance\_cancel\_flag[ k ][ m ] != 1 ) { |  |
| **as\_method\_type**[ k ][ m ] | ue(v) |
| if(as\_method\_type[ k ][ m ] == 1) { |  |
| **as\_filter\_eom\_points\_flag**[ k ][ m ] | u(1) |
| **as\_grid\_size\_minus2**[ k ][ m ] | u(5) |
| **as\_threshold**[ k ][ m ] | u(8) |
| **as\_threshold\_variation**[ k ][ m ] | u(8) |
| **as\_threshold\_difference**[ k ][ m ] | u(8) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

* + 1. SEI payload semantics
       1. General SEI payload semantics

The specifications in subclause F.3.1 apply.

* + - 1. Filler payload SEI message semantics

The specifications in F.3.2 apply.

* + - 1. User data registered by Recommendation ITU-T T.35 SEI message semantics

The specifications in F.3.3 apply.

* + - 1. User data unregistered SEI message semantics

The specifications in F.3.4 apply.

* + - 1. Recovery point SEI message semantics

The specifications in F.3.5 apply.

* + - 1. No display SEI message semantics

The specifications in F.3.6 apply.

* + - 1. Reserved SEI message semantics

The specifications in F.3.7 apply.

* + - 1. SEI manifest SEI message semantics

The specifications in F.3.8 apply.

* + - 1. SEI prefix indication SEI message semantics

The specifications in F.3.9 apply.

* + - 1. Active sub-bitstreams SEI message semantics

The specifications in F.3.10 apply.

* + - 1. Component codec mapping SEI message semantics

The specifications in F.3.11 apply.

* + - 1. Volumetric annotation SEI messages family syntax

The specifications in F.3.12 and subclauses apply.

* + - 1. Buffering period SEI message semantics

The specifications in F.3.13 apply.

* + - 1. Atlas frame timing SEI message semantics

The specifications in F.3.14 apply.

* + - 1. Viewport SEI messages family semantics

The specifications in F.3.15 and subclauses apply

* + - 1. Decoded Atlas Information Hash SEI message semantics
         1. General

The specifications in F.3.16.1 apply.

* + - * 1. Strings of bytes arrangement for hash calculation

General

The specifications in F.3.16.2.1 apply.

High-level atlas information string of bytes

Common ASPS level string of bytes

The specifications in F.3.16.2.2.1 apply.

Application-based ASPS level string of bytes

Let the function ASPSApplicationByteString( stringByte, posByte ) be defined as follows:

ASPSApplicationByteString( stringByte, posByte ) {  
 if( asps\_pixel\_deinterleaving\_enabled\_flag ) {  
 for( j = 0; j <= asps\_map\_count\_minus1; j++ )  
 stringByte[ posByte++ ] = asps\_map\_pixel\_deinterleaving\_flag[ j ] && 0xFF  
 }  
  
 stringByte[ posByte++ ] = asps\_raw\_patch\_enabled\_flag && 0xFF  
 stringByte[ posByte++ ] = asps\_eom\_patch\_enabled\_flag && 0xFF  
 if( asps\_eom\_patch\_enabled\_flag && asps\_map\_count\_minus1 == 0 ) {  
 stringByte[ posByte++ ] = asps\_eom\_fix\_bit\_count\_minus1 && 0xFF  
 }  
  
 if( ( asps\_auxiliary\_video\_enabled\_flag ) &&  
 ( asps\_raw\_patch\_enabled\_flag || asps\_eom\_patch\_enabled\_flag ) ) {  
 stringByte[ posByte++ ] = AuxVideoWidthNF && 0xFF  
 stringByte[ posByte++ ] = (AuxVideoWidthNF >> 8 ) && 0xFF  
 stringByte[ posByte++ ] = AuxVideoHeightNF && 0xFF  
 stringByte[ posByte++ ] = ( AuxVideoHeightNF >> 8 ) && 0xFF  
 }  
  
 stringByte[ posByte++ ] = asps\_plr\_enabled\_flag && 0xFF  
 if( asps\_plr\_enabled\_flag ) {   
 for( i = 0; i < asps\_map\_count\_minus1 + 1; i++) {  
 stringByte[ posByte++ ] = plri\_map\_present\_flag[ i ] && 0xFF  
 if( plri\_map\_present\_flag[ i ] ) {  
 stringByte[ posByte++ ] = plri\_number\_of\_modes\_minus1[ i ] && 0xFF  
 for( j = 0; j < plri\_number\_of\_modes\_minus1[ i ] + 1; j++ ) {  
 stringByte[ posByte++ ] = plri\_interpolate\_flag[ i ][ j ] && 0xFF  
 stringByte[ posByte++ ] = plri\_filling\_flag[ i ][ j ] && 0xFF  
 stringByte[ posByte++ ] = plri\_minimum\_depth[ i ][ j ] && 0xFF  
 stringByte[ posByte++ ] = plri\_neighbour\_minus1[ i ][ j ] && 0xFF  
 }  
 stringByte[ posByte++ ] = plri\_block\_threshold\_per\_patch\_minus1[ i ] && 0xFF  
 }  
 }  
 }  
  
 stringByte[ posByte++ ] = asps\_vpcc\_remove\_duplicate\_point\_enabled\_flag&& 0xFF  
 stringByte[ posByte++ ] = asps\_vpcc\_surface\_thickness\_minus1 && 0xFF  
 stringByte[ posByte++ ] = ( asps\_vpcc\_surface\_thickness\_minus1 >> 8 ) && 0xFF  
  
 return posByte  
 }

Common AFPS level string of bytes

The specifications in F.3.16.2.2.3 apply.

Application-based AFPS level string of bytes

The specifications in F.3.16.2.2.4 apply.

Atlas information string of bytes

Common atlas patch string of bytes

The specifications in F.3.16.2.3.1 apply.

Application-based atlas patch string of bytes

Let the function AtlasPatchApplicationByteString( stringByte, posByte, p ) be defined as follows:

AtlasPatchApplicationByteString( stringByte, posByte, p ) {  
 stringByte[ posByte++ ] = AtlasPatchInAuxVideo[ p ] & 0xFF  
 if( AtlasPatchType[ p ] == RAW ) {  
 stringByte[ posByte++ ] = AtlasPatchRawPoints[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatchRawPoints[ p ] >> 8 ) & 0xFF  
 } else if( AtlasPatchType[ p ] == PROJECTED ) {  
 stringByte[ posByte++ ] = AtlasPatch45DegreeMode[ p ] & 0xFF  
 blockCnt = BlockCnt( AtlasPatch2dSizeX[ p ], AtlasPatch2dSizeY[ p ] )  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 if(AtlasPlrdLevel[ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 stringByte[ posByte++ ] =  
 AtlasPlrdBlockModeMinus1[ p ][ m ][ j ] & 0xFF  
 stringByte[ posByte++ ] =  
 (AtlasPlrdBlockModeMinus1[ p ][ m ][ j ] >> 8 ) & 0xFF  
 }  
 } else {  
 stringByte[ posByte++ ] = AtlasPlrdModeMinus1[ p ][ m ] & 0xFF  
 stringByte[ posByte++ ] =  
 (AtlasPlrdBlockModeMinus1[ p ][ m ] >> 8 ) & 0xFF  
 }  
 }  
 }  
 } else if( AtlasPatchType[ p ] == EOM ) {  
 stringByte[ posByte++ ] = AtlasPatchEOMPatchCount[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( AtlasPatchEOMPatchCount[ p ] >> 8 ) & 0xFF  
 for( i = 0; i < AtlasEpduPatchCount[ p ]; i++ ) {  
 stringByte[ posByte++ ] = AtlasEpduPoints[ p ][ i ] & 0xFF  
 stringByte[ posByte++ ] = (AtlasEpduPoints[ p ][ i ] >> 8 ) & 0xFF  
 }  
 }  
 return posByte  
 }

Tile information string of bytes

Common tile patch string of bytes

The specifications in F.3.16.2.4.1 apply.

Application based tile patch string of bytes

Let the function TilePatchApplicationByteString( stringByte, posByte, t, p ) be defined as follows:

TilePatchApplicationByteString( stringByte, posByte, t, p) {  
 stringByte[ posByte++ ] = TilePatchInAuxVideo[ p ] & 0xFF  
 if( TilePatchType[ t ][ p ] == RAW ) {  
 stringByte[ posByte++ ] = TilePatchRawPoints[ p ] & 0xFF  
 stringByte[ posByte++ ] = ( TilePatchRawPoints[ p ] >> 8 ) & 0xFF  
 } else if( TilePatchType[ t ][ p ] == PROJECTED ) {  
 stringByte[ posByte++ ] = TilePatch45DegreeMode[ t ][ p ] & 0xFF  
 blockCnt = BlockCnt( AtlasPatch2dSizeX[ t ][ p ], AtlasPatch2dSizeY[ t ][ p ] )  
 for( m = 0; m < asps\_map\_count\_minus1 + 1; m++ ) {  
 if( plri\_map\_present\_flag[ m ] == 1 ) {  
 if(TilePlrdLevel[ t ][ p ][ m ] == 0 ) {  
 for( j = 0; j < blockCnt; j++ ) {  
 stringByte[ posByte++ ] =  
 TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ][ j ] & 0xFF  
 stringByte[ posByte++ ]=  
 (TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ][ j ] >> 8 ) & 0xFF  
 }  
 } else {  
 stringByte[ posByte++ ] = TilePatchPlrdModeMinus1[ t ][ p ][ m ] & 0xFF  
 stringByte[ posByte++ ] =  
 (TilePatchPlrdBlockModeMinus1[ t ][ p ][ m ] >> 8 ) & 0xFF  
 }  
 }  
 }  
 } else if( TilePatchType[ tileID ][ p ] == EOM ) {  
 stringByte[ posByte++ ] = TilePatchEOMPatchCount[ t ][ p ] & 0xFF  
 stringByte[ posByte++ ] = (TilePatchEOMPatchCount[ t ][ p ] >> 8 ) & 0xFF  
 for( i = 0; i < TilePatchEomPatchCount[ t ][ p ]; i++ ) {  
 stringByte[ posByte++ ] = TileEpduPoints[ t ][ p ][ i ] & 0xFF  
 stringByte[ posByte++ ] = (TileEpduPoints[ t ][ p ][ i ] >> 8 ) & 0xFF  
 }  
 }  
 return posByte  
 }

Atlas block to patch string of bytes

The specifications in F.3.16.2.5 apply.

Tile block to patch string of bytes

The specifications in F.3.16.2.6 apply.

* + - 1. Time code SEI message semantics

The specifications in F.3.17 apply.

* + - 1. Attribute transformation parameters SEI message semantics

This SEI message provides information that is related to attribute processing and interpretation during reconstruction.

**atp\_cancel\_flag** equal to 1 indicates that the attribute transformation parameters SEI message cancels the persistence of any previous attribute transformation parameters SEI message in output order that applies to the current atlas and all parameters associated with this SEI message are set to their default values. atp\_cancel\_flag equal to 0 indicates that attribute transformation parameters information follows.

**atp\_num\_attribute\_updates** indicates the number of attributes that is to be updated by the current SEI message.

**atp\_attribute\_idx**[ j ] indicates the attribute index of the j-th attribute that is to be updated by the current SEI message.

**atp\_dimension\_minus1**[ k ] plus 1 indicates the number of dimensions that are expected to be associated with the attribute with index k in the current bitstream.

**atp\_offset\_params\_enabled\_flag**[ k ][ i ]equal to 1 indicates that attribute offset parameters are present in the current atlas tile attribute parameter set for the i-th dimension of the attribute with index k atp\_offset\_params\_enabled\_flag equal to 0 indicates that attribute offset parameters are not present in the current atlas tile attribute parameter set. The default value of atp\_offset\_params\_enabled\_flag[ k ][ i ] is equal to 0.

**atp\_attribute\_scale\_minus1**[ k ][ i ] plus 1 indicates the value of the scale to be applied to the values of the i‑th dimension of an attribute with index k. The value of atp\_attribute\_scale\_minus1[ k ][ i ] shall be in the range of 0 to 232 – 1, inclusive**.** The default value of atp\_attribute\_scale\_minus1[ k ][ i ] is equal to 0.

**atp\_attribute\_offset**[ k ][ i ] indicates the value of the offset to be added to the values of the i‑th dimension of an attribute with index k. The value of atp\_attribute\_offset[ k ][ i ] shall be in the range of −231 to 231 – 1, inclusive**.** The default value of atp\_attribute\_offset[ k ][ i ] is equal to 0.

In particular, for a point p, if the value of the i-th dimension of its attribute with index k is equal to attrValue, its transformed value is derived as follows:

transValue = ( atp\_attribute\_scale\_minus1[ k ][ i ] + 1 )\* attrValue + atp\_attribute\_offset[ k ][ i ]

**atp\_persistence\_flag** specifies the persistence of the attribute transformation parameters SEI message for the current layer. atp\_persistence\_flag equal to 0 specifies that the attribute transformation parameters SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. atp\_persistence\_flag equal to 1 specifies that the attribute transformation parameters SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing an attribute transformation parameters SEI message, with either an atp\_cancel\_flag equal to 1 or the same value of atp\_persistence\_flag, applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.
  + - 1. Occupancy synthesis SEI message semantics

This SEI message specifies the recommended occupancy synthesis methods and their associated parameters that could be used to process the occupancy video frames associated with the current point cloud frame when the occupancy video frames are of a lower resolution than the nominal resolution of the atlas.

Up to 256 different occupancy synthesis instances could be specified for use with each point cloud frame. These instances are indicated using an array OccupancySynthesisMethod. The occupancy synthesis instance that a decoder may select to operate in, is outside the scope of this document.

At the start of each sequence, let OccupancySynthesisMethod[ i ] = 0, where i corresponds to the occupancy synthesis instance index and is in the range of 0 to 255, inclusive. When OccupancySynthesisMethod[ i ] is equal to 0 it means that no occupancy synthesis method is indicated for the occupancy synthesis instance with index i.

**os\_persistence\_flag** specifies the persistence of the occupancy synthesis SEI message for the current layer. os\_persistence\_flag equal to 0 specifies that the occupancy synthesis SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. os\_persistence\_flag equal to 1 specifies that the occupancy synthesis SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing an occupancy synthesis SEI message with the same value of os\_persistence\_flag and applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**os\_reset\_flag** equal to 1 resets all entries in the array OccupancySynthesisMethod to 0 and all parameters associated with this SEI to their default values.

**os\_instances\_updated** specifies the number of occupancy synthesis instances that will be updated in the current occupancy synthesis SEI message.

**os\_instance\_index**[ i ] indicates the i-th occupancy synthesis instance index in the array OccupancySynthesisMethod that is to be updated by the current SEI message.

**os\_instance\_cancel\_flag**[ k ]equal to 1 indicates that the value of OccupancySynthesisMethod[ k ] and all parameters associated with the occupancy synthesis instance with index k should be set to 0 and to their default values, respectively.

**os\_method\_type**[ k ]indicates the occupancy synthesis method, OccupancySynthesisMethod[ k ], that can be used for processing the current point cloud frame as specified in Table H-6 for occupancy synthesis instance with index k.

**Table H-6 – Definition of os\_method\_type[ k ]**

|  |  |
| --- | --- |
| **Value** | **Interpretation** |
| 0 | No occupancy synthesis / Pixel replication |
| 1 | Patch border filtering |
| 2- | Reserved |

Values of os\_method\_type[ k ] greater than 1 are reserved for future use by ISO/IEC. It is a requirement of bitstream conformance that bitstreams conforming to this version of this document shall not contain such values of os\_method\_type[ k ]. Decoders shall ignore occupancy synthesis SEI messages that contain reserved values of os\_method\_type[ k ]. The default value of os\_method\_type[ k ] is equal to 0.

**os\_log2\_threshold\_minus1**[ k ] specifies the value of the variable PbfThreshold[ k ] used for processing the occupancy of the current point cloud frame for occupancy synthesis instance with index k when the patch border filtering process is used.

The value of PbfThreshold[ k ] is computed as follows:

PbfThreshold[ k ] = 2os\_log2\_threshold\_minus1[ k ] + 1

The value of os\_log2\_threshold\_minus1[ k ] shall be in the range of 0 to 3, inclusive. The default value of os\_log2\_threshold\_minus1[ k ] is equal to 0.

**os\_passes\_count\_minus1**[ k ]plus 1 specifies the value of the variable PbfPasses[ k ] used for processing the occupancy of the current point cloud frame for occupancy synthesis instance with index k when the patch border filtering process is used. The value of os\_passes\_count\_minus1[ k ] shall be in the range of 0 to 3, inclusive. The default value of os\_passes\_count\_minus1[ k ] is equal to 0.

The value of PbfPasses[ k ] is computed as follows:

PbfPasses[ k ] = os\_passes\_count\_minus1[ k ] + 1

**os\_filter\_size\_minus1**[ k ]plus 1 specifies the values of the variables PbfSizeX[ k ] and PbfSizeY[ k ] used for processing the occupancy of the current point cloud frame for occupancy synthesis instance with index k when the patch border filtering process is used. The values of PbfSizeX[ k ] and PbfSizeY[ k ] are computed as follows:

PbfSizeX[ k ] = os\_filter\_size\_minus1[ k ] + 1

PbfSizeY[ k ] = PbfSizeX[ k ] >> 1

The value of os\_filter\_size\_minus1[ k ]shall be in the range of 0 to 7, inclusive. The default value of os\_filter\_size\_minus1[ k ] is equal to 0.

* + - 1. Geometry smoothing SEI message semantics

This SEI message specifies the recommended geometry smoothing methods and their associated parameters that could be used to process the geometry of the current point cloud frame after it is reconstructed, so as to obtain improved reconstructed geometry quality.

Up to 256 different geometry smoothing instances could be specified for use with each point cloud frame. These instances are indicated using an array GeometrySmoothingMethod. The geometry smoothing instance that a decoder may select to operate in, is outside the scope of this document.

At the start of each sequence, let GeometrySmoothingMethod[ i ] be set equal to 0, where i corresponds to the geometry smoothing instance index and is in the range of 0 to 255, inclusive. When GeometrySmoothingMethod[ i ] is equal to 0 it means that no geometry smoothing filter is indicated for the geometry smoothing instance with index i.

**gs\_persistence\_flag** specifies the persistence of the geometry smoothing SEI message for the current layer. gs\_persistence\_flag equal to 0 specifies that the geometry smoothing SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. gs\_persistence\_flag equal to 1 specifies that the geometry smoothing SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing a geometry smoothing SEI message with the same value of gs\_persistence\_flag and applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**gs\_reset\_flag** equal to 1 resets all entries in the array GeometrySmoothingMethod to 0 and all parameters associated with this SEI message are set to their default values.

**gs\_instances\_updated** specifies the number of geometry smoothing instances that will be updated in the current geometry smoothing SEI message.

**gs\_instance\_index**[ i ] indicates the i-th geometry smoothing instance index in the array GeometrySmoothingMethod that is to be updated by the current SEI message.

**gs\_instance\_cancel\_flag**[ k ]equal to 1 indicates that the value of GeometrySmoothingMethod[ k ] and that all parameters associated with the geometry smoothing instance with index k should be set to 0 and to their default values, respectively.

**gs\_method\_type**[ k ]indicates the geometry smoothing method, GeometrySmoothingMethod[ k ], that can be used for processing the current point cloud frame as specified in Table H-7 for geometry smoothing instance with index k.

**Table** H-7 – Definition of gs\_method\_type[ k ]

|  |  |
| --- | --- |
| **Value** | **Interpretation** |
| 0 | No geometry smoothing |
| 1 | Grid based geometry smoothing |
| 2- | Reserved |

Values of gs\_method\_type[ k ] greater than 1 are reserved for future use by ISO/IEC. It is a requirement of bitstream conformance that bitstreams conforming to this version of this document shall not contain such values of gs\_method\_type[ k ]. Decoders shall ignore geometry smoothing SEI messages that contain reserved values of gs\_method\_type[ k ]. The default value of gs\_method\_type[ k ] is equal to 0.

**gs\_filter\_eom\_points\_flag**[ k ]equal to 1specifies that geometry smoothing is applied to EOM points for the geometry smoothing instance with index k. gs\_filter\_eom\_points\_flag[ k ] equal to 0specifies that geometry smoothing is not applied to EOM points for the geometry smoothing instance with index k. The default value of gs\_filter\_eom\_points\_flag[ k ] is equal to 0.

**gs\_grid\_size\_minus2**[ k ] plus 2 specifies the value of the variable GeometrySmoothingGridSize[ k ] used when processing the current point cloud frame for geometry smoothing instance with index k. The value of gs\_grid\_size\_minus2[ k ] shall be in the range of 0 to 31, inclusive. The default value of gs\_grid\_size\_minus2[ k ] is equal to 0.

The value of GeometrySmoothingGridSize[ k ] is computed as follows:

GeometrySmoothingGridSize[ k ] = gs\_grid\_size\_minus2[ k ] + 2

**gs\_threshold**[ k ] indicates the smoothing threshold used for processing the current point cloud frame for geometry smoothing instance with index k. The value of gs\_threshold[ k ] shall be in the range of 0 to 255, inclusive. The default value of gs\_threshold[ k ] is equal to 0.

* + - 1. Attribute smoothing SEI message semantics

This SEI message specifies the recommended attribute smoothing methods and any associated parameters that could be used to process the attributes of the current point cloud frame after it is reconstructed, so as to obtain improved reconstructed attribute quality.

For each attribute, up to 256 different attribute smoothing instances could be specified for use with each point cloud frame. These instances are indicated using a two dimensional array AttributeSmoothingMethod. The attribute smoothing instance that a decoder may select to operate in, is outside the scope of this document.

At the start of each sequence, let AttributeSmoothingMethod[ j ][ i ] be set equal to 0 for all j and i, where j corresponds to the attribute index and is in the range of 0 to 127, inclusive, and i corresponds to the attribute smoothing instance index and is in the range of 0 to 255, inclusive. When AttributeSmoothingMethod[ j ][ i ] is equal to 0 it means that no attribute smoothing filter is indicated for an attribute index j and its corresponding attribute smoothing instance with index i.

**as\_persistence\_flag** specifies the persistence of the attribute smoothing SEI message for the current layer. as\_persistence\_flag equal to 0 specifies that the attribute smoothing SEI message applies to the current decoded atlas frame only.

Let aFrmA be the current atlas frame. as\_persistence\_flag equal to 1 specifies that the attribute smoothing SEI message persists for the current layer in output order until any of the following conditions are true:

* A new CAS begins.
* The bitstream ends.
* An atlas frame aFrmB in the current layer in a coded atlas access unit containing an attribute smoothing SEI message with the same value of as\_persistence\_flag and applicable to the current layer is output for which AtlasFrmOrderCnt( aFrmB ) is greater than AtlasFrmOrderCnt( aFrmA ), where AtlasFrmOrderCnt( aFrmB ) and AtlasFrmOrderCnt( aFrmA ) are the AtlasFrmOrderCntVal values of aFrmB and aFrmA, respectively, immediately after the invocation of the decoding process for atlas frame order count for aFrmB.

**as\_reset\_flag** equal to 1 resets all entries in the array AttributeSmoothingMethod to 0 and all parameters associated with this SEI message are set to their default values.

**as\_num\_attributes\_updated** specifies the number of attributes for which attribute smoothing instances will be updated in the current attribute smoothing SEI message.

**as\_attribute\_idx**[ j ]specifies the j-th attribute index that is to be updated.

**as\_attribute\_smoothing\_cancel\_flag**[ k ]equal to 1 indicates that all attribute smoothing instances associated with the attribute with index k should be set to 0, and that AttributeSmoothingMethod[ k ][ i ] and all parameters associated with attribute with index k and attribute smoothing instance with index i, for i being in the range of 0 to 255, should be set to 0 and to their default values, respectively.

**as\_instances\_updated**[ k ]specifies the number of attribute smoothing instances that will be updated for the attribute with index k.

**as\_instance\_index**[ k ][ i ]specifies the i-th index of the attribute smoothing instance to be updated for attribute with index k.

**as\_instance\_cancel\_flag**[ k ][ m ]equal to 1 indicates that the attribute smoothing instance with index m associated with the attribute with index k should be set to 0, and that AttributeSmoothingMethod[ k ][ m ] and all parameters associated with attribute with index k and attribute smoothing instance with index m should be set to 0 and their default values, respectively.

**as\_method\_type**[ k ][ m ]indicates the attribute smoothing method, AttributeSmoothingMethod[ k ][ m ] that is associated with the m-th attribute smoothing instance of attribute with index k as specified in Table H-8.

**Table H-8 – Definition of as\_method\_type[ k ][ m ]**

|  |  |
| --- | --- |
| **Value** | **Interpretation** |
| 0 | No attribute smoothing |
| 1 | Grid based attribute smoothing |
| 2- | Reserved |

Values of as\_method\_type[ k ][ m ] greater than 1 are reserved for future use by ISO/IEC. It is a requirement of bitstream conformance that bitstreams conforming to this version of this document shall not contain such values of as\_method\_type[ k ][ m ]. Decoders shall ignore attribute smoothing SEI messages that contain reserved values of as\_method\_type[ k ][ m ]. The default value of as\_method\_type[ k ][ m ] is equal to 0.

**as\_filter\_eom\_points\_flag**[ k ][ m ]equal to 1specifies that attribute smoothing is applied to EOM points for the m-th attribute smoothing instance of attribute with index k. as\_filter\_eom\_points\_flag[ k ][ m ]equal to 0specifies that attribute smoothing is not applied to EOM points for the m-th attribute smoothing instance of attribute with index k. The default value of as\_filter\_eom\_points\_flag[ k ][ m ] is equal to 0.

**as\_grid\_size\_minus2**[ k ][ m ] plus 2 specifies the value of the variable AttributeSmoothingGridSize[ k ][ m ] used for attribute with index k that is associated with attribute smoothing instance with index The default value of as\_grid\_size\_minus2[ k ][ m ] is equal to 0.

The value of AttributeSmoothingGridSize[ k ][ m ] is computed as follows:

AttributeSmoothingGridSize[ k ][ m ] = as\_grid\_size\_minus2[ k ][ i ] + 2

**as\_threshold**[ k ][ m ] indicates the attribute smoothing threshold for attribute smoothing instance with index m of attribute with index k, assuming a nominal bit depth of 8 bits. The value of as\_threshold[ k ][ m ] shall be in the range of 0 to 255, inclusive. The default value of as\_threshold[ k ][ m ] is equal to 0.

**as\_threshold\_variation**[ k ][ m ] indicates the threshold of attribute variation for attribute smoothing instance with index m of attribute with index k, assuming a nominal bit depth of 8 bits. The value of as\_threshold\_variation[ k ][ m ] shall be in the range of 0 to 255, inclusive. The default value of as\_threshold\_variation[ k ][ m ] is equal to 0.

**as\_threshold\_difference**[ k ][ m ]indicates the threshold of attribute difference for attribute smoothing instance with index m of attribute with index k, assuming a nominal bit depth of 8 bits. The value of as\_threshold\_difference[ k ][ m ] shall be in the range of 0 to 255, inclusive. The default value of as\_threshold\_difference[ k ][ m ] is equal to 0.

* 1. Volumetric Usability Information

The specifications in Annex G and its subclauses apply.

Bibliography

1. Recommendation ITU-T H.222.0 (in force), *Information technology – Generic coding of moving pictures and associated audio information: Systems.*
2. Recommendation ITU-T H.320 (in force), *Narrow-band visual telephone systems and terminal equipment*.
3. ISO/IEC 13818-1(in force), *Information technology – Generic coding of moving pictures and associated audio information – Part 1: Systems.*
4. ISO/IEC 11578 (in force), *Information technology – Open Systems Interconnection – Remote Procedure Call (RPC).*
5. ISO/IEC DIS 23090-10: *Information technology – Coded representation of immersive media— Part 10: Carriage of Visual Volumetric Video-based Coding Data*.[[5]](#footnote-6)
6. SMPTE ST 12-1 (2014), *Time and Control Code.*
7. ISO/IEC CD 23090-12: *Information technology – Coded representation of immersive media— Part 12: MPEG Immersive Video*.[[6]](#footnote-7)

1. Under preparation. Stage at time of publication: ISO/IEC DIS 23090-3:2020. [↑](#footnote-ref-2)
2. Under preparation. Stage at time of publication: ISO/IEC CD 23090-12:2020 [↑](#footnote-ref-3)
3. Under preparation. Stage at time of publication: ISO/IEC DIS 23090-10:2020 [↑](#footnote-ref-4)
4. Under preparation. Stage at time of publication: ISO/IEC DIS 23090-3:2020 [↑](#footnote-ref-5)
5. Under preparation. Stage at time of publication: ISO/IEC DIS 23090-10:2020 [↑](#footnote-ref-6)
6. Under preparation. Stage at time of publication: ISO/IEC CD 23090-12:2020 [↑](#footnote-ref-7)