 ISO/IEC JTC 1/SC 29/WG 07 N0126

**ISO/IEC JTC 1/SC 29/WG 07  
MPEG 3DG   
Convenorship: FR (France)**

**Document type: Output Document**

**Title:** Corrections for V3C and V-PCC 1st Edition

**Status:** Approved

**Date of document:** 2021-06-01

**Source:** ISO/IEC JTC 1/SC 29/WG 07

**No. of pages:** 5 (with cover page)

**Email of Convenor:** marius.preda @ it-sudparis.eu

**Committee URL:** <https://isotc.iso.org/livelink/livelink/open/jtc1sc29wg7>

**INTERNATIONAL ORGANIZATION FOR STANDARDIZATION**

**ORGANISATION INTERNATIONALE DE NORMALISATION**

**ISO/IEC JTC 1/SC 29/WG 03 MPEG SYSTEMS**

**ISO/IEC JTC 1/SC 29/WG 03 N0126**

**April 2021, Virtual**

|  |  |
| --- | --- |
| **Title** | **Corrections for V3C and V-PCC 1st Edition** |
| **Source** | **WG 07, MPEG 3DG** |
| **Status** | **Approved** |
| **Serial Number** | **20440** |

**ISO/IEC 23090-5:2021/COR**

ISO/IEC TC JTC 1/SC 29/WG7

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) — Corrigendum**

**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.

*To help you, this guide on writing standards was produced by the ISO/TMB and is available at* [*https://www.iso.org/iso/how-to-write-standards.pdf*](https://www.iso.org/iso/how-to-write-standards.pdf)

*A model manuscript of a draft International Standard (known as “The Rice Model”) is available at* [*https://www.iso.org/iso/model\_document-rice\_model.pdf*](https://www.iso.org/iso/model_document-rice_model.pdf)

© ISO 2021

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

ISO copyright office

CP 401 • Ch. de Blandonnet 8

CH-1214 Vernier, Geneva

Phone: +41 22 749 01 11

Email: copyright@iso.org

Website: www.iso.org

Published in Switzerland

*In subclause 3.36, replace*

3.36

coded atlas sequence

CAS

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

*with*

3.36

coded atlas sequence

CAS

sequence of *coded atlas access units* (3.35)*,* in decoding order, of an *IRAP coded atlas access unit* (3.74) with NoOutputBeforeRecoveryFlag equal to 1, followed by zero or more *coded atlas access units* (3.35)that are not *IRAP coded atlas access units* (3.74) with NoOutputBeforeRecoveryFlag equal to 1, including all subsequent *coded atlas access units* (3.35)up to but not including any subsequent *coded atlas access unit* (3.35) that is an *IRAP coded atlas access unit* (3.74) with NoOutputBeforeRecoveryFlag equal to 1.

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

*In subclause 8.3.7.3, replace*

|  |  |
| --- | --- |
| 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 ) |  |
| } |  |

*with*

|  |  |
| --- | --- |
| 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 ] ) { |  |
| **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 ) |  |
| } |  |

*In subclause 8.4.5.2, replace*

|  |  |  |  |
| --- | --- | --- | --- |
| 40 | NAL\_EOS | End of sequence end\_of\_seq\_rbsp( ) | non-ACL |

*with*

|  |  |  |  |
| --- | --- | --- | --- |
| 40 | NAL\_EOS | End of sequence end\_of\_sequence\_rbsp( ) | non-ACL |

*In subclause 8.4.6.2.2, replace*

**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.

*with*

**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.

The variable AftiSignalledTileIDBitCount is computed as follows:

When afti\_signalled\_tile\_id\_length\_minus1 is not present

AftiSignalledTileIDBitCount = Ceil( Log2( afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1 ) )

Otherwise

AftiSignalledTileIDBitCount = afti\_signalled\_tile\_id\_length\_minus1 + 1

**afti\_tile\_id[ i ]** specifies the tile ID of the i-th tile. 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 AftiSignalledTileIDBitCount bits.

*In subclause 8.4.6.11, replace*

**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.

*with*

**ath\_type** specifies the coding type of the current atlas tile 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.

*In subclause 8.4.6.11, replace*

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.

*with*

The following applies:

— The length of ath\_id is AftiSignalledTileIDBitCount 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.

*In subclause 8.4.7.3, replace*

**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.

*with*

**pdu\_lod\_scale\_x\_minus1**[ tileID ][ p ] plus 1 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 1.

*In subclause 9.2.5.1, replace*

sizeY = TilePatch2dSize[ tileID ][ p ]

*with*

sizeY = TilePatch2dSizeY[ tileID ][ p ]

*In subclause 9.2.5.1, replace*

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 |

*with*

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 |

*In subclause 9.2.5.2, replace*

TilePatchLoDScaleX[ tileID ][ p ] =   
 pdu\_lod\_enabled\_flag[ p ] ? pdu\_lod\_scale\_x\_minus1[ tileID ][ p ] + 1 : 1 (56)

*with*

TilePatchLoDScaleX[ tileID ][ p ] =   
 pdu\_lod\_enabled\_flag[ tileID ][ p ] ? pdu\_lod\_scale\_x\_minus1[ tileID ][ p ] + 1 : 1 (56)

*In subclause F.2.12.4, replace*

|  |  |
| --- | --- |
| 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) |
| } |  |
| } |  |
| } |  |

*with*

|  |  |
| --- | --- |
| 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 ] | ue(v) |
| **vri\_bounding\_box\_left**[ p ] | ue(v) |
| **vri\_bounding\_box\_width**[ p ] | ue(v) |
| **vri\_bounding\_box\_height**[ p ] | ue(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) |
| } |  |
| } |  |
| } |  |

*In subclause A.3, replace*

**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) |

*with*

**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 | 'vvi1' |
| Reserved | 4..126 | – |
| MP4RA | 127 | provided by component codec mapping SEI message (F.2.11) |

*In subclause A.3, replace*

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.

*with*

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 'vvi1'. The codec ids associated with all video sub-bitstreams in the active VPS are inferred to be mapped to 'vvi1'. The component codec mapping SEI message (F.2.11), when present, shall also indicate the value of ‘vvi1' for all instances j of ccm\_codec\_4cc[ j ] present in this SEI message.

*In subclause F.3.16.2.2.3, replace*

Let the function AFPSCommonByteString( stringByte, posByte ) be defined as follows:

AFPSCommonByteString( stringByte, posByte ) {  
 stringByte[ posByte++ ] = afti\_num\_tiles\_in\_atlas\_frame\_minus1 && 0xFF  
 for ( i = 1; i < afti\_num\_tiles\_in\_atlas\_frame\_minus1 + 1; i++ ) {  
 stringByte[ posByte++ ] = TileOffsetX[ i ] && 0xFF  
 stringByte[ posByte++ ] = (TileOffsetX[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = TileOffsetY[ i ] && 0xFF  
 stringByte[ posByte++ ] = (TileOffsetY[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = TileWidth[ i ] && 0xFF  
 stringByte[ posByte++ ] = (TileWidth[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = TileHeight[ i ] && 0xFF  
 stringByte[ posByte++ ] = (TileHeight[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = AuxTileOffset[ i ] && 0xFF  
 stringByte[ posByte++ ] = (AuxTileOffset[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = AuxTileHeight[ i ] && 0xFF  
 stringByte[ posByte++ ] = (AuxTileHeight[ i ] >> 8 )&& 0xFF  
 stringByte[ posByte++ ] = afti\_tile\_id[ i ] && 0xFF  
 stringByte[ posByte++ ] = (afti\_tile\_id[ i ] >> 8 )&& 0xFF  
 }  
 return posByte  
 }

*with*

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  
 }

*In subclause H.7.3, replace*

— 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 ]

*with*

— 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 ]

*In subclause H.7.3, replace*

yBlock = nbBlock / ( ( AtlasPatch2dSizeY[ pIdx ] + offset ) / PatchPackingBlockSize )

*with*

yBlock = nbBlock / ( ( AtlasPatch2dSizeX[ pIdx ] + offset ) / PatchPackingBlockSize )