# 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.

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

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

## Specification of syntax functions and descriptors

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

byte\_aligned( ) is specified as:

* If the next bit in the bitstream is the first bit in a byte (DataUnitReadIdx % 8 == 0), the value of byte\_aligned( ) is true.
* Otherwise, the value of byte\_aligned( ) is false.

more\_data\_in\_data\_unit( ) is specified as:

* If parsing of the DU is incomplete (DataUnitReadIdx / 8 < DataUnitLength), the value of more\_data\_in\_data\_unit( ) is true.
* Otherwise, the value of more\_data\_in\_data\_unit( ) is false.

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

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

* ae(v): adaptive arithmetic entropy-coded syntax element.
* de(v): dictionary coded syntax element.
* oid(v): an ASN.1 object identifier.
* s(𝑛): signed integer using an 𝑛-bit magnitude and a sign bit.
* se(v): signed integer 0-th order Exp-Golomb-coded syntax element.
* u(𝑛): unsigned integer using 𝑛 bits. When 𝑛 is "v" in the syntax table, the number of bits varies in a manner dependent upon the value of other syntax elements.
* ue(v): unsigned integer 0-th order Exp-Golomb-coded syntax element.

## Syntax in tabular form

### General

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

### Parameter sets, ancillary data and byte alignment

#### Sequence parameter set data unit syntax

|  |  |
| --- | --- |
| seq\_parameter\_set( ) { | Descriptor |
| simple\_profile\_compliant | u(1) |
| dense\_profile\_compliant | u(1) |
| predictive\_profile\_compliant | u(1) |
| main\_profile\_compliant | u(1) |
| reserved\_profile\_18bits | u(18) |
| slice\_reordering\_constraint | u(1) |
| unique\_point\_positions\_constraint | u(1) |
| level\_idc | u(8) |
| sps\_seq\_parameter\_set\_id | u(4) |
| frame\_ctr\_lsb\_bits | u(5) |
| slice\_tag\_bits | u(5) |
| seq\_origin\_bits | ue(v) |
| if( seq\_origin\_bits) { |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |
| seq\_origin\_xyz[ 𝑘 ] | s(v) |
| seq\_origin\_log2\_scale | ue(v) |
| } |  |
| seq\_bbox\_size\_bits | ue(v) |
| if( seq\_bbox\_size\_bits ) |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |
| seq\_bbox\_size\_minus1\_xyz[ 𝑘 ] | u(v) |
| seq\_unit\_numerator\_minus1 | ue(v) |
| seq\_unit\_denominator\_minus1 | ue(v) |
| seq\_unit\_is\_metres | u(1) |
| seq\_coded\_scale\_exponent | ue(v) |
| seq\_coded\_scale\_mantissa\_bits | ue(v) |
| seq\_coded\_scale\_mantissa | u(v) |
| num\_attributes | ue(v) |
| for( attrIdx = 0; attrIdx < num\_attributes; attrIdx++ ) { |  |
| attr\_components\_minus1[ attrIdx ] | ue(v) |
| attr\_instance\_id[ attrIdx ] | ue(v) |
| attr\_bitdepth\_minus1[ attrIdx ] | ue(v) |
| attr\_label\_known[ attrIdx ] | u(1) |
| if( attr\_label\_known[ attrIdx ] ) |  |
| attr\_label[ attrIdx ] | ue(v) |
| else |  |
| attr\_label\_oid[ attrIdx ] | oid(v) |
| attr\_property\_cnt | ue(v) |
| byte\_alignment( ) |  |
| for( 𝑗 = 0; 𝑗 < attr\_property\_cnt; 𝑗++ ) |  |
| attribute\_property(attrIdx ) |  |
| } |  |
| geom\_axis\_order | u(3) |
| bypass\_stream\_enabled | u(1) |
| entropy\_continuation\_enabled | u(1) |
| sps\_extension\_present | u(1) |
| if( sps\_extension\_present ) { |  |
| if(num\_attributes > 1) |  |
| cross\_attr\_prediction\_enabled | u(1) |
| bypass\_bin\_coding\_prob\_update\_disabled | u(1) |
| inter\_frame\_prediction\_enabled | u(1) |
| if(inter\_frame\_prediction\_enabled) |  |
| inter\_entropy\_continuation\_enabled | u(1) |
| fgs\_layer\_group\_enabled | u(1) |
| if(fgs\_layer\_group\_enabled) |  |
| fgs\_parameter ( ) |  |
| while( more\_data\_in\_data\_unit( ) ) |  |
| sps\_extension\_data | u(1) |
| } |  |
| byte\_alignment( ) |  |
| } |  |

#### Attribute property syntax

|  |  |
| --- | --- |
| attribute\_property( attrIdx ) { | Descriptor |
| attr\_prop\_type | u(8) |
| attr\_prop\_len | u(8) |
| AttrPropDataLen = attr\_prop\_len |  |
| if( attr\_prop\_type == 0 ) { |  |
| attr\_prop\_itu\_t\_t35\_country\_code | u(8) |
| AttrPropDataLen−− |  |
| if( attr\_prop\_itu\_t\_t35\_country\_code == 255 ) { |  |
| attr\_prop\_itu\_t\_t35\_country\_code\_extension\_byte | u(8) |
| AttrPropDataLen−− |  |
| } |  |
| attribute\_property\_data( attrIdx, AttrPropDataLen ) |  |
| } else if( attr\_prop\_type == 1 ) { |  |
| attr\_prop\_oid | oid(v) |
| AttrPropDataLen −= Length( attr\_prop\_oid ) / 8 |  |
| attribute\_property\_data( attrIdx, AttrPropDataLen ) |  |
| } else if( attr\_prop\_type == 2 ) { |  |
| attr\_cicp\_colour\_primaries[ attrIdx ] | ue(v) |
| attr\_cicp\_transfer\_characteristics[ attrIdx ] | ue(v) |
| attr\_cicp\_matrix\_coeffs[ attrIdx ] | ue(v) |
| attr\_cicp\_video\_full\_range[ attrIdx ] | u(1) |
| } else if( attr\_prop\_type == 3 ) { |  |
| attr\_offset\_bits | ue(v) |
| attr\_offset[ attrIdx ] | s(v) |
| attr\_scale\_bits | ue(v) |
| attr\_scale\_minus1[ attrIdx ] | u(v) |
| attr\_frac\_bits[ attrIdx ] | ue(v) |
| } else if( attr\_prop\_type == 4 ) { |  |
| for( 𝑐 = 0; 𝑐 ≤ attr\_components\_minus1[ attrIdx ]; 𝑐++ ) |  |
| attr\_default\_value[ attrIdx ][ 𝑐 ] | u(v) |
| } else |  |
| attribute\_property\_data( attrIdx, attr\_prop\_len ) |  |
| byte\_alignment( ) |  |
| } |  |

#### Attribute property data syntax

|  |  |
| --- | --- |
| attribute\_property\_data( attrIdx, numBytes ) { | Descriptor |
| for( 𝑖 = 0; 𝑖 < numBytes; 𝑖++) |  |
| attr\_prop\_byte[ 𝑖 ] | u(8) |
| } |  |

#### Tile inventory data unit syntax

|  |  |
| --- | --- |
| tile\_inventory( ) { | Descriptor |
| ti\_seq\_parameter\_set\_id | u(4) |
| ti\_frame\_ctr\_lsb\_bits | u(5) |
| ti\_frame\_ctr\_lsb | u(v) |
| tile\_cnt | u(16) |
| if( tile\_cnt > 0 ) { |  |
| tile\_id\_bits | u(5) |
| tile\_origin\_bits\_minus1 | u(8) |
| tile\_size\_bits\_minus1 | u(8) |
| for( tileIdx = 0; tileIdx < tile\_cnt; tileIdx++ ) { |  |
| tile\_id[ tileIdx ] | u(v) |
| tileId = tile\_id\_bits ? tile\_id[ tileIdx ] : tileIdx |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |
| tile\_origin\_xyz[ tileId ][ 𝑘 ] | s(v) |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |
| tile\_size\_minus1\_xyz[ tileId ][ 𝑘 ] | u(v) |
| } |  |
| ti\_origin\_bits\_minus1 | ue(v) |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |
| ti\_origin\_xyz[ 𝑘 ] | s(v) |
| ti\_origin\_log2\_scale | ue(v) |
| } |  |
| byte\_alignment( ) |  |
| } |  |

#### Geometry parameter set data unit syntax

|  |  |
| --- | --- |
| geometry\_parameter\_set( ) { | Descriptor |
| gps\_geom\_parameter\_set\_id | u(4) | |
| gps\_seq\_parameter\_set\_id | u(4) | |
| slice\_geom\_origin\_scale\_present | u(1) | |
| if( ¬slice\_geom\_origin\_scale\_present ) |  | |
| gps\_geom\_origin\_log2\_scale | ue(v) | |
| geom\_dup\_point\_counts\_enabled | u(1) | |
| geom\_tree\_type | u(1) | |
| if( geom\_tree\_type == 0 ) { |  | |
| occtree\_point\_cnt\_list\_present | u(1) | |
| occtree\_direct\_coding\_mode | u(2) | |
| if( occtree\_direct\_coding\_mode ) |  | |
| occtree\_direct\_joint\_coding\_enabled | u(1) | |
| occtree\_coded\_axis\_list\_present | u(1) | |
| occtree\_neigh\_window\_log2\_minus1 | u(3) | |
| if( occtree\_neigh\_window\_log2\_minus1 > 0 ) { |  | |
| occtree\_adjacent\_child\_enabled | u(1) | |
| occtree\_intra\_pred\_max\_nodesize\_log2 | ue(v) | |
| } |  | |
| occtree\_bitwise\_coding | u(1) | |
| occtree\_planar\_enabled | u(1) | |
| if( occtree\_planar\_enabled ) { |  | |
| for( 𝑖 = 0; 𝑖 < 3; 𝑖++) |  | |
| occtree\_planar\_threshold[ 𝑖 ] | ue(v) | |
| if( occtree\_direct\_coding\_mode == 1 ) |  | |
| occtree\_direct\_node\_rate\_minus1 | u(5) | |
| } |  | |
| } |  | |
| geom\_angular\_enabled | u(1) | |
| if( geom\_angular\_enabled ) { |  | |
| slice\_angular\_origin\_present | u(1) | |
| if( ¬slice\_angular\_origin\_present ) { |  | |
| gps\_angular\_origin\_bits\_minus1 | ue(v) | |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  | |
| gps\_angular\_origin\_xyz[ 𝑘 ] | s(v) | |
| } |  | |
| if( geom\_tree\_type == 1 ) { |  | |
| ptree\_ang\_azimuth\_pi\_bits\_minus11 | ue(v) | |
| ptree\_ang\_azimuth\_step\_minus1 | ue(v) | |
| ptree\_ang\_radius\_scale\_log2 | ue(v) | |
| } |  | |
| num\_beams\_minus1 | ue(v) | |
| beam\_elevation\_init | se(v) | |
| beam\_voffset\_init | se(v) | |
| if( geom\_tree\_type == 0 ) |  | |
| beam\_steps\_per\_rotation\_init\_minus1 | ue(v) | |
| for( 𝑖 = 1; 𝑖 ≤ num\_beams\_minus1; 𝑖++ ) { |  | |
| beam\_elevation\_diff[ 𝑖 ] | se(v) | |
| beam\_voffset\_diff[ 𝑖 ] | se(v) | |
| if( geom\_tree\_type == 0 ) |  | |
| beam\_steps\_per\_rotation\_diff[ 𝑖 ] | se(v) | |
| } |  | |
| if( occtree\_planar\_enabled ) |  | |
| occtree\_planar\_buffer\_disabled | u(1) | |
| } |  | |
| geom\_scaling\_enabled | u(1) | |
| if( geom\_scaling\_enabled ) { |  | |
| geom\_qp | ue(v) | |
| geom\_qp\_mul\_log2 | u(2) | |
| if( geom\_tree\_type == 1 ) |  | |
| ptree\_qp\_period\_log2 | ue(v) | |
| else if( occtree\_direct\_coding\_mode ) |  | |
| occtree\_direct\_node\_qp\_offset | se(v) | |
| } |  | |
| gps\_extension\_present | u(1) | |
| if( gps\_extension\_present ) { |  | |
| if(geom\_tree\_type==0) { |  | |
| trisoup\_enable\_flag | u(1) | |
| if(trisoup\_enable\_flag) { |  | |
| trisoup\_non\_cubic\_node\_start\_edge\_presence\_flag | u(1) | |
| trisoup\_non\_cubic\_node\_end\_edge\_presence\_flag | u(1) | |
| } |  | |
| } |  | |
| if(geom\_tree\_type == 0 && geom\_angular\_enabled) |  | |
| occtree\_angular\_extension\_enabled | u(1) | |
| if(geom\_tree\_type == 0 || geom\_angular\_enabled) |  | |
| inter\_prediction\_enabled | u(1) | |
| if(inter\_prediction\_enabled) { |  | |
| global\_motion\_enabled | u(1) | |
| if(geom\_tree\_type == 1) { |  | |
| inter\_azim\_scale\_log2 | ue(v) | |
| resampling\_enabled | u(1) | |
| max\_points\_per\_entry\_minus1 | ue(v) | |
| if(max\_points\_per\_entry\_minus1 > 0) |  | |
| down\_sampling\_range | ue(v) | |
| } |  | |
| biprediction\_enabled | ue(v) | |
| if(biprediction\_enabled > 0) |  | |
| frame\_merge\_enabled | u(1) | |
| } |  | |
| if( geom\_tree\_type == 0 && geom\_angular\_enabled && occtree\_direct\_coding\_mode && inter\_prediction\_enabled) |  | |
| occtree\_inter\_angular\_direct\_coding\_enabled | u(1) | |
| if(occtree\_planar\_enabled && geom\_angular\_enabled && occtree\_direct\_coding\_mode) |  | |
| geo\_disable\_planar\_idcm\_angular | u(1) | |
| if( geom\_tree\_type == 1 && geom\_angular\_enabled) { |  | |
| ptree\_sec\_resid\_disabled | u(1) | |
| ptree\_ang\_azimuth\_scaling\_enabled | u(1) | |
| if(ptree\_ang\_azimuth\_scaling\_enabled) { |  | |
| ptree\_ang\_max\_pred\_index | ue(v) | |
| ptree\_ang\_pred\_list\_radius\_resid\_threshold | ue(v) | |
| ptree\_ang\_radius\_resid\_context\_qphi\_threshold\_present | u(1) | |
| if(ptree\_ang\_radius\_resid\_context\_qphi\_threshold\_present) { |  | |
| ptree\_ang\_radius\_resid\_context\_qphi\_threshold | ue(v) | |
| } |  | |
| } |  | |
| } |  | |
| if(occtree\_planar\_enabled && geom\_angular\_enabled == 0 ) |  | |
| **octree\_planar\_neigh\_prediction\_enabled** | u(1) | |
| while( more\_data\_in\_data\_unit( ) ) |  | |
| gps\_extension\_data | u(1) | |
| } |  | |
| byte\_alignment( ) |  | |
| } |  | |

#### Attribute parameter set data unit syntax

|  |  |
| --- | --- |
| attribute\_parameter\_set( ) { | Descriptor |
| aps\_attr\_parameter\_set\_id | u(4) |
| aps\_seq\_parameter\_set\_id | u(4) |
| attr\_coding\_type | ue(v) |
| attr\_primary\_qp\_minus4 | ue(v) |
| attr\_secondary\_qp\_offset | se(v) |
| attr\_qp\_offsets\_present | u(1) |
| if( attr\_coding\_type == 0 ) { |  |
| raht\_prediction\_enabled | u(1) |
| if( raht\_prediction\_enabled ) { |  |
| raht\_prediction\_subtree\_min | ue(v) |
| raht\_prediction\_samples\_min | ue(v) |
| } |  |
| } else if( attr\_coding\_type ≤ 2 ) { |  |
| pred\_set\_size\_minus1 | ue(v) |
| pred\_inter\_lod\_search\_range | ue(v) |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |
| pred\_dist\_bias\_minus1\_xyz[ 𝑘 ] | ue(v) |
| if ( attr\_coding\_type == 2 ) |  |
| last\_comp\_pred\_enabled | u(1) |
| lod\_scalability\_enabled | u(1) |
| if( lod\_scalability\_enabled ) |  |
| pred\_max\_range\_minus1 | ue(v) |
| else { |  |
| lod\_max\_levels\_minus1 | ue(v) |
| if( ¬lod\_max\_levels\_minus1 ) |  |
| attr\_canonical\_order\_enabled | u(1) |
| else { |  |
| lod\_decimation\_mode | ue(v) |
| if( lod\_decimation\_mode > 0 ) |  |
| for( lvl = 0; lvl < lod\_max\_levels\_minus1; lvl++ ) |  |
| lod\_sampling\_period\_minus2[ lvl ] | ue(v) |
| lod\_initial\_dist\_log2 | ue(v) |
| lod\_dist\_log2\_offset\_present | u(1) |
| } |  |
| } |  |
| if( attr\_coding\_type == 1 ) { |  |
| pred\_direct\_max\_idx\_plus1 | ue(v) |
| if( pred\_direct\_max\_idx\_plus1 ) { |  |
| pred\_direct\_threshold | u(8) |
| pred\_direct\_avg\_disabled | u(1) |
| } |  |
| pred\_intra\_lod\_search\_range | ue(v) |
| if( pred\_intra\_lod\_search\_range ) |  |
| pred\_intra\_min\_lod | ue(v) |
| inter\_comp\_pred\_enabled | u(1) |
| pred\_blending\_enabled | u(1) |
| } |  |
| } else if( attr\_coding\_type == 3 ) |  |
| raw\_attr\_width\_present | u(1) |
| if( ¬lod\_scalability\_enabled ) |  |
| attr\_coord\_conv\_enabled | u(1) |
| if( attr\_coord\_conv\_enabled ) |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) { |  |
| attr\_coord\_conv\_scale\_bits\_minus1[ 𝑘 ] | u(5) |
| attr\_coord\_conv\_scale[ 𝑘 ] | u(v) |
| } |  |
| aps\_extension\_present | u(1) |
| if( aps\_extension\_present ) { |  |
| if(cross\_attr\_prediction\_enabled) { |  |
| cross\_attr\_prediction\_enabled\_this\_type | u(1) |
| if( cross\_attr\_prediction\_enabled\_this\_type) |  |
| refAttrIdx | ue(v) |
| } |  |
| if( attr\_coding\_type == 0 ) { |  |
| lossless\_coding\_enabled | u(1) |
| raht\_last\_comp\_pred\_enabled | u(1) |
| } |  |
| if( attr\_coding\_type == 1) |  |
| for( *i* = 0;  *i* <= pred\_set\_size\_minus1;  *i*++ ) |  |
| quant\_neigh\_weight[*i*] | ue(v) |
| attr\_inter\_prediction\_enabled | u(1) |
| if( attr\_inter\_prediction\_enabled) |  |
| if( attr\_coding\_type == 0 ) { |  |
| raht\_inter\_layer\_depth\_minus1 | ue(v) |
| if(lossless\_coding\_enabled == 0 ) { |  |
| raht\_send\_inter\_filters | u(1) |
| raht\_inter\_skip\_layers | ue(v) |
| } |  |
| raht\_inter\_layer\_code\_enabled | u(1) |
| } |  |
| else |  |
| attr\_inter\_prediction\_search\_range | ue(v) |
| if( (attr\_coding\_type == 1 || attr\_coding\_type == 2) && ¬lod\_scalability\_enabled && ¬lod\_max\_levels\_minus1) |  |
| max\_points\_per\_sort\_log2\_plus1 | ue(v) |
| if( (attr\_coding\_type == 1 || attr\_coding\_type == 2) && pred\_set\_size\_minus1 >= 2) |  |
| prediction\_with\_distribution\_enabled | U(1) |
| if( attr\_coding\_type == 0 ) |  |
| raht\_buffer\_extension\_flag | u(1) |
| if( attr\_coding\_type == 0 && raht\_prediction\_enabled) { |  |
| raht\_subnode\_prediction\_enabled | u(1) |
| raht\_intra\_layer\_code\_enabled | u(1) |
| if( raht\_subnode\_prediction\_enabled) |  |
| for( *i* = 0;  *i* < 5;  *i* ++ ) |  |
| raht\_prediction\_weights[ *i* ] | ue(v) |
| raht\_prediction\_search\_range | ue(v) |
| } |  |
| if( (attr\_coding\_type == 1 || attr\_coding\_type == 2)  && fgs\_layer\_group\_enabled) |  |
| fgs\_attr\_parameter( ) |  |
| while( more\_data\_in\_data\_unit( ) ) |  |
| aps\_extension\_data | u(1) |
| } |  |
| byte\_alignment( ) |  |
| } |  |

#### Frame-specific attribute properties data unit syntax

|  |  |
| --- | --- |
| frame\_specific\_attribute\_properties( ) { | Descriptor |
| fsap\_seq\_parameter\_set\_id | u(4) |
| fsap\_frame\_ctr\_lsb\_bits | u(5) |
| fsap\_frame\_ctr\_lsb | u(v) |
| fsap\_sps\_attr\_idx | ue(v) |
| fsap\_num\_props | ue(v) |
| byte\_alignment( ) |  |
| for( 𝑖 = 0; 𝑖 < fsap\_num\_props; 𝑖++ ) |  |
| attribute\_property( fsap\_sps\_attr\_idx ) |  |
| } |  |

#### Frame boundary marker data unit syntax

|  |  |
| --- | --- |
| frame\_boundary\_marker( ) { | Descriptor |
| fbdu\_frame\_ctr\_lsb\_bits | u(5) |
| fbdu\_frame\_ctr\_lsb | u(v) |
| byte\_alignment( ) |  |
| } |  |

#### User data data unit syntax

|  |  |
| --- | --- |
| userdata\_data\_unit( ) { | Descriptor |
| user\_data\_oid | oid(v) |
| while( more\_data\_in\_data\_unit( ) ) |  |
| user\_data\_byte | u(8) |
| } |  |

#### Byte alignment syntax

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

### Geometry data unit

#### Geometry data unit syntax

|  |  |
| --- | --- |
| geometry\_data\_unit( ) { | Descriptor |
| geometry\_data\_unit\_header( ) |  |
| if( geom\_tree\_type == 0 ) { |  |
| if(fgs\_layergroup\_enabled) |  |
| fgs\_occupancy\_tree(startDepth, endDepth   ) |  |
| else |  |
| occupancy\_tree(occtreeMaxDepthMinus1   ) |  |
| if(trisoup\_enable\_flag == 1) |  |
| trisoup( ) |  |
| } else if( geom\_tree\_type == 1 ) |  |
| predictive\_tree( ) |  |
| geometry\_data\_unit\_footer(occtreeMaxDepthMinus1 ) |  |
| } |  |

#### Geometry data unit header syntax

|  |  |  |
| --- | --- | --- |
| geometry\_data\_unit\_header( ) { | Descriptor | Semantics |
| gdu\_geometry\_parameter\_set\_id | u(4) | 7.4.3.2 |
| gdu\_temporal\_id | u(3) | 7.4.3.2 |
| slice\_id | ue(v) | 7.4.3.2 |
| slice\_tag | u(v) | 7.4.3.2 |
| frame\_ctr\_lsb | u(v) | 7.4.3.2 |
| if( entropy\_continuation\_enabled ) { |  |  |
| slice\_entropy\_continuation | u(1) | 7.4.3.2 |
| if( slice\_entropy\_continuation ) |  |  |
| prev\_slice\_id | ue(v) | 7.4.3.2 |
| } |  |  |
| if( slice\_geom\_origin\_scale\_present ) |  |  |
| slice\_geom\_origin\_log2\_scale | ue(v) | 7.4.3.2 |
| slice\_geom\_origin\_bits\_minus1 | ue(v) | 7.4.3.2 |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| slice\_geom\_origin\_xyz[ 𝑘 ] | u(v) | 7.4.3.2 |
| if( slice\_angular\_origin\_present ) { |  |  |
| slice\_angular\_origin\_bits\_minus1 | ue(v) | 7.4.3.2 |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| slice\_angular\_origin\_xyz[ 𝑘 ] | s(v) | 7.4.3.2 |
| } |  |  |
| if( geom\_tree\_type == 0 ) { |  |  |
| occtree\_depth\_minus1 | ue(v) | 9.2.3 |
| if( occtree\_coded\_axis\_list\_present ) |  |  |
| for( dpth = 0; dpth ≤ occtreeMaxDepthMinus1; dpth++ ) |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| occtree\_coded\_axis[ dpth ][ 𝑘 ] | u(1) | 9.2.3 |
| occtree\_stream\_cnt\_minus1 | ue(v) | 9.2.3 |
| } |  |  |
| if( geom\_scaling\_enabled ) { |  |  |
| slice\_geom\_qp\_offset | se(v) | 7.4.3.2 |
| if( geom\_tree\_type == 1 ) |  |  |
| slice\_ptree\_qp\_period\_log2\_offset | se(v) | 9.3.2.1 |
| } |  |  |
| if(trisoup\_enable\_flag == 1 ) { |  |  |
| trisoup\_node\_size\_log2\_minus2 | u(3) | 9.4.2.1 |
| trisoup\_sampling\_value\_minus1 | u(8) | 9.4.2.1 |
| trisoup\_num\_unique\_segments\_bits\_minus1 | ue(v) | 9.4.2.1 |
| trisoup\_num\_unique\_segments\_minus1 | u(v) | 9.4.2.1 |
| trisoup\_vertex\_number\_bits | u(3) | 9.4.2.1 |
| trisoup\_centroid\_vertex\_residual\_flag | u(1) | 9.4.2.1 |
| if( trisoup\_centroid\_vertex\_residual\_flag) |  |  |
| trisoup\_face\_vertex\_flag | u(1) | 9.4.2.1 |
| trisoup\_halo\_flag | u(1) | 9.4.2.1 |
| if( trisoup\_halo\_flag) { |  |  |
| trisoup\_adaptive\_halo\_flag | u(1) | 9.4.2.1 |
| } |  |  |
| trisoup\_vertex\_merge | u(1) | 9.4.2.1 |
| if( trisoup\_non\_cubic\_node\_start\_edge\_presence\_flag) { |  |  |
| trisoup\_slice\_bb\_pos\_bits | ue(v) | 9.4.2.1 |
| if( trisoup\_slice\_bb\_pos\_bits) { |  |  |
| trisoup\_slice\_bb\_pos\_log2\_scale | ue(v) | 9.4.2.1 |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| trisoup\_slice\_bb\_pos\_xyz[*k*] | u(v) | 9.4.2.1 |
| } |  |  |
| } |  |  |
| if( trisoup\_non\_cubic\_node\_end\_edge\_presence\_flag) { |  |  |
| trisoup\_slice\_bb\_width\_bits | ue(v) | 9.4.2.1 |
| if( trisoup\_slice\_bb\_width\_bits) { |  |  |
| trisoup\_slice\_bb\_width\_log2\_scale | ue(v) | 9.4.2.1 |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| trisoup\_slice\_bb\_width\_xyz[*k*] | u(v) | 9.4.2.1 |
| } |  |  |
| } |  |  |
| } |  |  |
| if( geom\_tree\_type == 1 ) { |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| ptn\_resid\_abs\_log2\_bits[ 𝑘 ] | u(3) | 9.3.2.1 |
| if( geom\_angular\_enabled ) |  |  |
| ptn\_radius\_min | ue(v) | 9.3.2.1 |
| } |  |  |
| if( inter\_prediction\_enabled) { |  |  |
| slice\_inter\_prediction | u(1) | 7.4.3.2 |
| if( slice\_inter\_prediction) { |  |  |
| if( global\_motion\_enabled) { |  |  |
| global\_motion\_params(0) |  |  |
| if(biprediction\_enabled) { |  |  |
| slice\_biprediction | u(1) | 7.4.3.2 |
| if( slice\_biprediction && global\_motion\_enabled) { |  |  |
| global\_motion\_params(1) |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| if( slice\_inter\_prediction &&  inter\_entropy\_continuation\_enabled) { |  |  |
| slice\_inter\_entropy\_continuation | u(1) | 7.4.3.2 |
| if( slice\_inter\_entropy\_continuation ) { |  |  |
| prev\_inter\_entropy\_frame\_ctr\_lsb | u(v) | 7.4.3.2 |
| prev\_inter\_entropy\_slice\_id | ue(v) | 7.4.3.2 |
| } |  |  |
| } |  |  |
| if(fgs\_layer\_group\_enabled) |  |  |
| if( geom\_tree\_type == 0 ) |  |  |
| fgs\_geometry\_data\_unit\_parameter() |  | E.3.1.3.1 |
| byte\_alignment( ) |  |  |
| } |  |  |

#### Geometry data unit footer syntax

|  |  |  |
| --- | --- | --- |
| geometry\_data\_unit\_footer(occtreeMaxDepthMinus1 ) { | Descriptor | Semantics |
| byte\_alignment( ) |  |  |
| if( occtree\_point\_cnt\_list\_present ) |  |  |
| for( dpth = 1; dpth < occtreeMaxDepthMinus1; dpth++ ) |  |  |
| occtree\_lvl\_point\_cnt\_minus1[ dpth ] | u(24) | 9.2.3 |
| slice\_num\_points\_minus1 | u(24) | 7.4.3.3 |
| } |  |  |

#### Occupancy tree syntax

|  |  |  |
| --- | --- | --- |
| occupancy\_tree(occtreeMaxDepthMinus1) { | Descriptor | Semantics |
| if( slice\_inter\_prediction && global\_motion\_enabled && MotionPartitionType == 1) |  |  |
| for(idx = 0; idx < NumMotionBlocks; idx++) |  |  |
| gm\_comp\_partition\_block[idx] | ae(v) | 9.2.15.2.2 |
| OccQpSubtreeDepth = occtreeMaxDepthMinus1 + 1 |  | 9.2.14.4 |
| for( Dpth = 0; Dpth ≤ occtreeMaxDepthMinus1; Dpth++ ) { |  |  |
| occupancy\_tree\_level( Dpth ) |  |  |
| if( Dpth + 1 > OcctreeEntropyStreamDepth ) |  | 9.2.3 |
| occtree\_end\_of\_entropy\_stream | ae(v) | 9.2.3 |
| } |  |  |
| } |  |  |

#### Occupancy tree level syntax

|  |  |  |
| --- | --- | --- |
| occupancy\_tree\_level( dpth ) { | Descriptor | Semantics |
| if( geom\_scaling\_enabled && dpth < OccQpSubtreeDepth ) |  |  |
| occ\_subtree\_qp\_offset\_present | ae(v) | 9.2.14.3 |
| if( occ\_subtree\_qp\_offset\_present ) |  |  |
| OccQpSubtreeDepth = dpth |  |  |
| for( NodeIdx = 0; NodeIdx < OccNodeCnt[ dpth ]; NodeIdx++ ) |  |  |
| occupancy\_tree\_node( dpth, NodeIdx ) |  |  |
| } |  |  |

#### Occupancy tree node syntax

|  |  |  |
| --- | --- | --- |
| occupancy\_tree\_node( dpth, nodeIdx ) { | Descriptor | Semantics |
| if( occ\_subtree\_qp\_offset\_present ) { |  |  |
| occ\_subtree\_qp\_offset\_abs[ Ns ][ Nt ][ Nv ] | ae(v) | 9.2.14.3 |
| if( occ\_subtree\_qp\_offset\_abs[ Ns ][ Nt ][ Nv ] ) |  |  |
| occ\_subtree\_qp\_offset\_sign[ Ns ][ Nt ][ Nv ] | ae(v) | 9.2.14.3 |
| } |  |  |
| if(occtree\_direct\_coding\_mode && DirectNodePresent && geo\_disable\_planar\_idcm\_angular) |  |  |
| occ\_direct\_node | ae(v) | 9.2.12.2 |
| if( occtree\_planar\_enabled ) { |  |  |
| if(gps\_extension\_present ) { |  |  |
| if(AllowPlanarCopyMode ) |  | 9.2.11.10 |
| planar\_copy\_mode | ae(v) | 9.2.11.2 |
| if((¬planar\_copy\_mode && MultiPlanarEligible ) |  | 9.2.11.8 |
| multi\_planar\_flag | ae(v) | 9.2.11.2 |
| } |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| if( PlanarEligible[ 𝑘 ] ) { |  | 9.2.11.5 |
| if( ¬gps\_extension\_present || ¬PlanarInferred[ 𝑘 ]) |  | 9.2.11.9 |
| occ\_single\_plane[ 𝑘 ] | ae(v) | 9.2.11.2 |
| if( occ\_single\_plane[ 𝑘 ]  || ¬PlanarPosInferred[ 𝑘 ]) |  | 9.2.11.11 |
| occ\_plane\_pos[ 𝑘 ] | ae(v) | 9.2.11.2 |
| } |  |  |
| } |  |  |
| if( occtree\_direct\_coding\_mode && DirectNodePresent && geo\_disable\_planar\_idcm\_angular  ) |  | 9.2.12.3.2 |
| occ\_direct\_node | ae(v) | 9.2.12.2 |
| if( occ\_direct\_node ) |  |  |
| occupancy\_tree\_direct\_node( ) |  |  |
| else { |  |  |
| if( OccMaybeSingleChild ) |  | 9.2.6.8 |
| occ\_single\_child | ae(v) | 9.2.6.2 |
| if( occ\_single\_child ) |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| if( OccFreeAxis[ 𝑘 ] ) |  | 9.2.6.6 |
| occupancy\_idx[ 𝑘 ] | ae(v) | 9.2.6.2 |
| if( OccMapPresent ) |  | 9.2.6.9 |
| if( occtree\_bitwise\_coding ) { |  |  |
| for( 𝑖 = 0; 𝑖 < 8; 𝑖++ ) |  |  |
| if( OccBitPresent[ 𝑖 ] ) |  | 9.2.10.3 |
| occupancy\_bit[ 𝑖 ] | ae(v) | 9.2.6.2 |
| } else |  |  |
| occupancy\_byte | de(v) | 9.2.6.2 |
| if( TerminalNode && geom\_dup\_point\_counts\_enabled ) |  | 9.2.6.5 |
| for( child = 0; child < OccChildCnt; child++ ) |  |  |
| occ\_dup\_point\_cnt[ child ] | ae(v) | 9.2.6.2 |
| } |  |  |
| } |  |  |

#### Direct node syntax

|  |  |  |
| --- | --- | --- |
| occupancy\_tree\_direct\_node( ) { | Descriptor | Semantics |
| direct\_point\_cnt\_eq2 | ae(v) | 9.2.12.2 |
| if( geom\_dup\_point\_counts\_enabled && ¬direct\_point\_cnt\_eq2 ) |  |  |
| direct\_dup\_point\_cnt | ae(v) | 9.2.12.2 |
| if( occtree\_direct\_joint\_coding\_enabled && direct\_point\_cnt\_eq2 ) |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| if( ¬geom\_angular\_enabled || 𝑘 == ( 1 ^ AzimuthAxis ) ) { |  | 9.2.13.3 |
| direct\_joint\_prefix[ 𝑘 ] | ae(v) | 9.2.12.2 |
| if( DnJointDiffBitPresent[ 𝑘 ] ) |  | 9.2.12.5.4 |
| direct\_joint\_diff\_bit[ 𝑘 ] | ae(v) | 9.2.12.2 |
| } |  |  |
| for( dnPt = 0; dnPt ≤ direct\_point\_cnt\_eq2; dnPt++ ) |  |  |
| if( geom\_angular\_enabled ) { |  |  |
| direct\_rem[ dnPt ][ 1 ^ AzimuthAxis ] | ae(v) | 9.2.12.2 |
| beam\_idx\_resid\_abs[ dnPt ] | ae(v) | 9.2.12.2 |
| if( beam\_idx\_resid\_abs[ dnPt ] ) |  |  |
| beam\_idx\_resid\_sign[ dnPt ] | ae(v) | 9.2.12.2 |
| direct\_rem\_st\_ang[ dnPt ] | ae(v) | 9.2.12.2 |
| if( occtree\_angular\_extension\_enabled) { |  |  |
| if(DnBitsAfterPlanar[2] > 0) |  |  |
| direct\_v\_ang\_resid\_abs[ dnPt ] | ae(v) | 9.2.12.2 |
| if( direct\_v\_ang\_resid\_abs[ dnPt ] > 0) |  |  |
| direct\_v\_ang\_resid\_sign[ dnPt ] | ae(v) | 9.2.12.2 |
| } else |  |  |
| direct\_rem\_v\_ang[ dnPt ] | ae(v) | 9.2.12.2 |
| } else |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| direct\_rem[ dnPt ][ 𝑘 ] | ae(v) | 9.2.12.2 |
| } |  |  |

#### Predictive tree syntax

|  |  |  |
| --- | --- | --- |
| predictive\_tree( ) { | Descriptor | Semantics |
| PtnCnt = 0 |  |  |
| do { |  |  |
| predictive\_tree\_node( 0, PtnCnt ) |  |  |
| ptree\_end\_of\_slice | ae(v) | 9.3.2.1 |
| } while( ¬ptree\_end\_of\_slice ) |  |  |
| } |  |  |

#### Predictive tree node syntax

|  |  |  |
| --- | --- | --- |
| predictive\_tree\_node( dpth, nodeIdx ) { | Descriptor | Semantics |
| PtnCnt++ |  |  |
| if( geom\_scaling\_enabled && ¬( nodeIdx % PtnQpInterval ) ) { |  |  |
| ptn\_qp\_offset\_abs[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if( ptn\_qp\_offset\_abs[ nodeIdx ] ) |  |  |
| ptn\_qp\_offset\_sign[ nodeIdx ] | ae(v) | 9.3.2.2 |
| } |  |  |
| if( geom\_dup\_point\_counts\_enabled ) |  |  |
| ptn\_dup\_point\_cnt[ nodeIdx ] | ae(v) | 9.3.2.2 |
| ptn\_child\_cnt\_xor1[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if(slice\_inter\_prediction && dpth) |  |  |
| ptn\_inter\_flag[nodeIdx] | ae(v) | 9.3.2.2 |
| if(slice\_inter\_prediction && slice\_biprediction && dpth) |  |  |
| ptn\_pred\_direction[nodeIdx] | ae(v) | 9.3.2.2 |
| if(ptn\_inter\_flag[nodeIdx]) |  |  |
| ptn\_inter\_pred\_mode[nodeIdx] | ae(v) | 9.3.2.2 |
| else |  |  |
| if( ¬ptree\_ang\_azimuth\_scaling\_enabled) |  |  |
| ptn\_pred\_mode[ nodeIdx ] | ae(v) | 9.3.2.2 |
| else |  |  |
| ptn\_pred\_idx[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if( geom\_angular\_enabled ) { |  |  |
| ptn\_phi\_mul\_abs\_prefix[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if( ptn\_phi\_mul\_abs\_prefix[ nodeIdx ] == 2 ) |  |  |
| ptn\_phi\_mul\_abs\_minus2[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if( ptn\_phi\_mul\_abs\_minus2[ nodeIdx ] == 7 ) |  |  |
| ptn\_phi\_mul\_abs\_minus9[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if( ptn\_phi\_mul\_abs\_prefix[ nodeIdx ] ) |  |  |
| ptn\_phi\_mul\_sign[ nodeIdx ] | ae(v) | 9.3.2.2 |
| } |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) { |  |  |
| if( 𝑘 == 0 && ptree\_ang\_azimuth\_scaling\_enabled) { |  |  |
| **ptn\_radius\_resid\_abs**[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if(ptn\_radius\_resid\_abs[ nodeIdx ] > 0) |  |  |
| **ptn\_radius\_resid\_sign**[ nodeIdx ] | ae(v) | 9.3.2.2 |
| } |  |  |
| else if( 𝑘 == 1 && ptree\_ang\_azimuth\_scaling\_enabled) { |  |  |
| if( BoundPhiResid[ nodeIdx ] > 0) |  |  |
| **ptn\_phi\_resid\_abs\_gt0**[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if(ptn\_phi\_resid\_abs\_gt0[ nodeIdx ]) { |  |  |
| **ptn\_phi\_resid\_sign**[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if(BoundPhiResid[ nodeIdx ] > 1) |  |  |
| **ptn\_phi\_resid\_abs\_gt1**[ nodeIdx ] | ae(v) | 9.3.2.2 |
| if(ptn\_phi\_resid\_abs\_gt1[ nodeIdx ]) |  |  |
| **ptn\_phi\_resid\_abs\_rem**[ nodeIdx ] | ae(v) | 9.3.2.2 |
| } |  |  |
| } |  |  |
| else if( 𝑘 < 2 || ¬geom\_angular\_enabled || num\_beams\_minus1 ) { |  |  |
| ptn\_resid\_abs\_gt0[ nodeIdx ][ 𝑘 ] | ae(v) | 9.3.2.2 |
| if( ptn\_resid\_abs\_gt0[ nodeIdx ][ 𝑘 ] ) { |  |  |
| ptn\_resid\_abs\_log2[ nodeIdx ][ 𝑘 ] | ae(v) | 9.3.2.2 |
| ptn\_resid\_abs\_rem[ nodeIdx ][ 𝑘 ] | ae(v) | 9.3.2.2 |
| if( 𝑘 || ptn\_pred\_mode[ nodeIdx ] ) |  |  |
| ptn\_resid\_sign[ nodeIdx ][ 𝑘 ] | ae(v) | 9.3.2.2 |
| } |  |  |
| } |  |  |
| } |  |  |
| if( geom\_angular\_enabled  && ¬ptree\_sec\_resid\_disabled) |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) { |  |  |
| ptn\_sec\_resid\_abs[ nodeIdx ][ 𝑘 ] | ae(v) | 9.3.2.2 |
| if( ptn\_sec\_resid\_abs[ nodeIdx ][ 𝑘 ] ) |  |  |
| ptn\_sec\_resid\_sign[ nodeIdx ][ 𝑘 ] | ae(v) | 9.3.2.2 |
| } |  |  |
| for( 𝑖 = 0; 𝑖 < ( ptn\_child\_cnt\_xor1[ nodeIdx ] ^ 1 ); 𝑖++ ) |  |  |
| predictive\_tree\_node( dpth + 1, PtnCnt ) |  |  |
| } |  |  |

#### TriSoup syntax

|  |  |  |
| --- | --- | --- |
| trisoup(  ) { | Descriptor | Semantics |
| for( edgeIdx = 0; edgeIdx <  numberTriSoupEdges; edgeIdx++ ) { |  |  |
| vertex\_flag[edgeIdx] | ae(v) | 9.4.2.2 |
| if( vertex\_flag[edgeIdx]){ |  |  |
| for( bit = 0; bit <  trisoup\_vertex\_number\_bits; bit++ ) |  |  |
| vertex\_position[edgeIdx][bit] | ae(v) | 9.4.2.2 |
| } |  |  |
| } |  |  |
| for( nodeIdx = 0; nodeIdx <  numberTriSoupNodes; nodeIdx++ ) { |  |  |
| if( trisoup\_centroid\_vertex\_residual\_flag &&  trisoup\_sampling\_value\_minus1 ≤ 3 && numVertex > 3){ |  |  |
| centroid\_residual\_is\_zero[nodeIdx] | ae(v) | 9.4.2.3 |
| if(¬centroid\_residual\_is\_zero[nodeIdx]) |  |  |
| if( highBound && lowBound ) |  |  |
| centroid\_residual\_sign | ae(v) | 9.4.2.3 |
| magBound = (centroid\_residual\_sign ? highBound : lowBound) - 1 |  |  |
| m = 0 |  |  |
| while( magBound > 0){ |  |  |
| centroid\_residual\_magnitude[*m*] | ae(v) | 9.4.2.3 |
| if( centroid\_residual\_magnitude[*m*] ) |  |  |
| Break |  |  |
| magBound-- |  |  |
| *m*++ |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| if(trisoup\_face\_vertex\_flag) |  |  |
| for( nodeIdx = 0; nodeIdx <  numberTriSoupNodes; nodeIdx++ ) |  |  |
| for ( fvIdx = 0; fvIdx < 3; fvIdx++) |  |  |
| if (FaceEligible[nodeIdx][fvIdx]) |  | 9.4.2.4 |
| has\_face\_vertex[nodeIdx][fvIdx] | ae(v) | 9.4.2.3 |
| } |  |  |

#### Global motion parameters syntax

|  |  |  |
| --- | --- | --- |
| global\_motion\_params(idx ) { | Descriptor | Semantics |
| if( geom\_tree\_type == 1 ) { |  |  |
| slice\_inter\_frame\_ref\_gmc[ *idx*] | u(1) | 7.4.3.2 |
| if( geom\_tree\_type == 0 ||  slice\_inter\_frame\_ref\_gmc[ *idx*] ) { |  |  |
| for(*i* = 0; *i* < 3;*i*++ ) |  |  |
| for( *j* = 0; *j* < 3; *j*++ ) |  |  |
| gm\_matrix[ *idx*][*i* ][*j* ] | se(v) | 7.4.3.2 |
| for(*j* = 0; *j* < 3; *j*++ ) |  |  |
| gm\_trans[ *idx*][*j* ] | se(v) | 7.4.3.2 |
| } |  |  |
| if( geom\_tree\_type == 0 && idx == 0) { |  |  |
| motion\_partition\_type | u(1) | 7.4.3.2 |
| motion\_zero\_origin\_flag | u(1) | 7.4.3.2 |
| if(MotionPartitionType == 1) |  |  |
| for(*k* = 0; *k* < 3; *k*++ ) |  |  |
| motion\_block\_size[ *k*] | ue(v) | 7.4.3.2 |
| } |  |  |
| if( MotionPartitionType == 0 || slice\_inter\_frame\_ref\_gmc[ *idx*]) { |  |  |
| gm\_thres\_top[ *idx*] | se(v) | 7.4.3.2 |
| gm\_thres\_bot[ *idx*] | se(v) | 7.4.3.2 |
| } |  |  |
| } |  |  |

### Attribute data unit

#### Attribute data unit syntax

|  |  |
| --- | --- |
| attribute\_data\_unit( ) { | Descriptor |
| attribute\_data\_unit\_header( ) |  |
| if( attr\_coding\_type ≠ 3 ) { |  |
| if( raht\_inter\_layer\_code\_enabled || raht\_intra\_layer\_code\_enabled) |  |
| raht\_layer\_pred\_modes( ) |  |
| attribute\_coeffs( ) |  |
| } |  |
| else |  |
| attribute\_raw( ) |  |
| byte\_alignment( ) |  |
| } |  |

#### Attribute data unit header syntax

|  |  |  |
| --- | --- | --- |
| attribute\_data\_unit\_header( ) { | Descriptor | Semantics |
| adu\_attr\_parameter\_set\_id | u(4) | 7.4.4.2 |
| adu\_temporal\_id | u(3) | 7.4.4.2 |
| adu\_sps\_attr\_idx | ue(v) | 7.4.4.2 |
| adu\_slice\_id | ue(v) | 7.4.4.2 |
| if( lod\_dist\_log2\_offset\_present  || attr\_inter\_prediction\_enabled) |  |  |
| lod\_dist\_log2\_offset | se(v) | 10.6.2 |
| if( last\_comp\_pred\_enabled && AttrDim == 3) |  |  |
| for( dpth = 0; dpth ≤ lod\_max\_levels\_minus1; dpth++ ) |  |  |
| last\_comp\_pred\_coeff\_diff[ dpth ] | se(v) | 10.6.10.1 |
| if( inter\_comp\_pred\_enabled ) |  |  |
| for( dpth = 0; dpth ≤ lod\_max\_levels\_minus1; dpth++ ) |  |  |
| for( 𝑐 = 1; 𝑐 < AttrDim; 𝑐++) |  |  |
| inter\_comp\_pred\_coeff\_diff[ dpth ][ 𝑐 ] | se(v) | 10.6.10.1 |
| if( attr\_qp\_offsets\_present ) |  |  |
| for( qc = 0; qc < Min( 2, AttrDim ); qc++) |  |  |
| attr\_qp\_offset[ qc ] | se(v) | 10.7.1 |
| attr\_qp\_layers\_present | u(1) | 10.7.1 |
| if( attr\_qp\_layers\_present ) { |  |  |
| attr\_qp\_layer\_cnt\_minus1 | ue(v) | 10.7.1 |
| for( dpth = 0; dpth ≤ attr\_qp\_layer\_cnt\_minus1; dpth++ ) |  |  |
| for( qc = 0; qc < Min( 2, AttrDim ); qc++ ) |  |  |
| attr\_qp\_layer\_offset[ dpth ][ qc ] | se(v) | 10.7.1 |
| } |  |  |
| attr\_qp\_region\_cnt | ue(v) | 10.7.1 |
| if( attr\_qp\_region\_cnt ) |  |  |
| attr\_qp\_region\_bits\_minus1 | ue(v) | 10.7.1 |
| for( 𝑖 = 0; 𝑖 < attr\_qp\_region\_cnt; 𝑖++ ) { |  |  |
| if( ¬attr\_coord\_conv\_enabled ) { |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| attr\_qp\_region\_origin\_xyz[ 𝑖 ][ 𝑘 ] | u(v) | 10.7.1 |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| attr\_qp\_region\_size\_minus1\_xyz[ 𝑖 ][ 𝑘 ] | u(v) | 10.7.1 |
| } else { |  |  |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| attr\_qp\_region\_origin\_rpi[ 𝑖 ][ 𝑘 ] | u(v) | 10.7.1 |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| attr\_qp\_region\_size\_minus1\_rpi[ 𝑖 ][ 𝑘 ] | u(v) | 10.7.1 |
| } |  |  |
| for( ps = 0; ps < Min( 2, AttrDim ); ps++) |  |  |
| attr\_qp\_region\_offset[ 𝑖 ][ ps ] | se(v) | 10.7.1 |
| } |  |  |
| if(attr\_coding\_type == 0 && lossless\_coding\_enabled == 0){ |  |  |
| attr\_AC\_qp\_offset\_present | u(1) | 10.7.1 |
| if(attr\_AC\_qp\_offset\_present){ |  |  |
| attr\_AC\_qp\_layer\_cnt\_minus1 | ue(v) | 10.7.1 |
| for( dpth = 0; dpth < attr\_AC\_qp\_layer\_cnt\_minus1; dpth++) |  |  |
| for( ACcompidx = 0; ACcompidx < 7; ACcompidx++) |  |  |
| for( qc = 0; qc < Min( 2, AttrDim ); qc++) |  |  |
| attr\_AC\_qp\_offset[dpth][qc ][ACcompidx] | se(v) | 10.7.1 |
| } |  |  |
| } |  |  |
| if(attr\_inter\_prediction\_enabled){ |  |  |
| slice\_attr\_inter\_prediction | u(1) | 7.4.4.2 |
| if(slice\_biprediction) |  |  |
| slice\_attr\_inter\_prediction2 | u(1) | 7.4.4.2 |
| if( slice\_attr\_inter\_prediction && raht\_send\_inter\_filters &&  ¬raht\_inter\_layer\_code\_enabled){ |  |  |
| num\_inter\_filters | ue(v) | 7.4.4.2 |
| for( filteridx = 0; filteridx < num\_inter\_filters; filteridx++) |  |  |
| raht\_inter\_filter\_qidx[filteridx] | se(v) | 7.4.4.2 |
| } |  |  |
| if( raht\_inter\_layer\_code\_enabled && slice\_attr\_inter\_prediction){ |  |  |
| **layer\_code\_depth** | ue(v) | 10.7.1 |
| if( raht\_send\_inter\_filters) |  |  |
| for( filteridx = 0; filteridx < layer\_code\_depth + 1 – raht\_inter\_skip\_layer; filteridx++) |  |  |
| if( filteridx == 0 && raht\_inter\_skip\_layer == 0) |  |  |
| raht\_inter\_filter\_qidx[filteridx] | se(v) | 7.4.4.2 |
| else if( slice\_raht\_inter\_layer\_code\_mode[filteridx + raht\_inter\_skip\_layer - 1]) |  |  |
| raht\_inter\_filter\_qidx[filteridx] | se(v) | 7.4.4.2 |
| } |  |  |
| } |  |  |
| if(fgs\_layer\_group\_enabled) |  |  |
| if(attr\_coding\_type == 1 || attr\_coding\_type == 2) |  |  |
| fgs\_attribute\_data\_unit\_parameter() |  | E.3.1.4.1 |
| byte\_alignment( ) |  |  |
| } |  |  |

#### Attribute data unit coefficients syntax

|  |  |  |
| --- | --- | --- |
| attribute\_coeffs( ) { | Descriptor | Semantics |
| for( 𝑖 = 0; 𝑖 < PointCnt; 𝑖++ ) { |  |  |
| zero\_run\_length\_prefix | ae(v) | 10.3.1.1 |
| if( zero\_run\_length\_prefix == 3 ) { |  |  |
| zero\_run\_length\_minus3\_div2 | ae(v) | 10.3.1.1 |
| if( zero\_run\_length\_minus3\_div2 < 4) |  |  |
| zero\_run\_length\_minus3\_mod2 | ae(v) | 10.3.1.1 |
| else |  |  |
| zero\_run\_length\_minus11 | ae(v) | 10.3.1.1 |
| } |  |  |
| 𝑖 += ZeroRunLength |  |  |
| if( 𝑖 < PointCnt ) |  |  |
| attribute\_coeff\_tuple( 𝑖 ) |  |  |
| } |  |  |
| } |  |  |

#### Attribute coefficient tuple syntax

|  |  |  |
| --- | --- | --- |
| attribute\_coeff\_tuple( coeffIdx ) { | Descriptor | Semantics |
| for( 𝑐 = 0, inferLastComp = 1; 𝑐 < AttrDim; 𝑐++ ) { |  |  |
| coeff\_abs[ 𝑐 ] | ae(v) | 10.3.1.2 |
| if( coeff\_abs[ 𝑐 ] || ( 𝑐 == AttrDim – 1 && inferLastComp ) ) |  |  |
| coeff\_sign[ 𝑐 ] | ae(v) | 10.3.1.2 |
| inferLastComp &= coeff\_abs[ 𝑐 ] == 0 |  |  |
| } |  |  |
| } |  |  |

#### Raw attribute value syntax

|  |  |  |
| --- | --- | --- |
| attribute\_raw( ) { | Descriptor | Semantics |
| for( ptIdx = 0; ptIdx < PointCnt; ptIdx++ ) |  |  |
| for( 𝑐 = 0; 𝑐 < AttrDim; 𝑐++ ) { |  |  |
| if( raw\_attr\_width\_present ) |  |  |
| raw\_attr\_component\_length | u(8) | 10.3.1.3 |
| raw\_attr\_value[ ptIdx ][ 𝑐 ] | u(v) | 10.3.1.3 |
| } |  |  |
| } |  |  |

#### Region-adaptive hierarchical layer prediction mode syntax

|  |  |  |
| --- | --- | --- |
| raht\_layer\_pred\_modes( ) { | Descriptor | Semantics |
| start = RahtRootLvl |  |  |
| for( lvl = start; lvl  ≥ 0; lvl --) { |  |  |
| if(  raht\_inter\_layer\_code\_enabled) |  |  |
| slice\_raht\_inter\_layer\_code\_mode[ lvl ] | ae(v) | 7.4.4.2 |
| if( raht\_intra\_layer\_code\_enabled ) |  |  |
| slice\_raht\_intra\_layer\_code\_mode[ lvl ] | ae(v) | 7.4.4.2 |
| } |  |  |
| } |  |  |

### Defaulted attribute data unit syntax

|  |  |  |
| --- | --- | --- |
| defaulted\_attribute\_data\_unit( ) { | Descriptor | Semantics |
| defattr\_seq\_parameter\_set\_id | u(4) | 7.4.5 |
| defattr\_reserved\_zero\_3bits | u(3) | 7.4.5 |
| defattr\_sps\_attr\_idx | ue(v) | 7.4.5 |
| defattr\_slice\_id | ue(v) | 7.4.5 |
| for( 𝑐 = 0; 𝑐 < AttrDim; 𝑐++ ) |  |  |
| defattr\_value[ 𝑐 ] | u(v) | 7.4.5 |
| byte\_alignment( ) |  |  |
| } |  |  |

## Semantics

### General

The semantics associated with the syntax structures and with the syntax elements within these structures are specified either in 7.4 or in the subclause identified by the semantics column of the syntax table.

When the semantics of a syntax element are specified in tabular form, any values that are not specified in the table(s) shall not be present in the bitstream unless otherwise specified in this document.

General constraints on syntax element values are specified in Annex A.

### Parameter sets, ancillary data and byte alignment

#### Sequence parameter set data unit semantics

##### General

The parameters specified by an SPS shall apply to any DU where that SPS is activated.

simple\_profile\_compliant specifies whether (when 1) or not (when 0) the bitstream conforms to the Simple profile.

dense\_profile\_compliant specifies whether (when 1) or not (when 0) the bitstream conforms to the Dense profile.

predictive\_profile\_compliant specifies whether (when 1) or not (when 0) the bitstream conforms to the Predictive profile.

main\_profile\_compliant specifies whether (when 1) or not (when 0) the bitstream conforms to the Main profile.

reserved\_profile\_18bits shall be equal to 0 in bitstreams conforming to this version of this document. Other values for reserved\_profile\_18bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of reserved\_profile\_18bits.

slice\_reordering\_constraint specifies whether (when 1) or not (when 0) the bitstream is sensitive to the reordering or removal of slices within a coded point cloud frame. If slices are reordered or removed when slice\_reordering\_constraint is 1, the resulting bitstream might not be fully decodable.

unique\_point\_positions\_constraint equal to 1 specifies that in each coded point cloud frame, all points shall have unique positions. unique\_point\_positions\_constraint equal to 0 specifies that in any coded point cloud frame, two or more points may have the same position.

* 1. Even if the points in each slice have unique positions, points from different slices in the same frame can be coincident. In this case, unique\_point\_positions\_constraint would be set to 0.
  2. Points with identical positions in the same frame are prohibited when unique\_point\_positions\_constraint is 1 even if they have different values of the frame index/number attribute.

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

sps\_seq\_parameter\_set\_id identifies the SPS for reference by other DUs. sps\_seq\_parameter\_set\_id shall be 0 in bitstreams conforming to this version of this document. Other values of sps\_seq\_parameter\_set\_id are reserved for future use by ISO/IEC.

frame\_ctr\_lsb\_bits specifies the length in bits of the syntax element frame\_ctr\_lsb.

slice\_tag\_bits specifies the length in bits of the syntax element slice\_tag.

bypass\_stream\_enabled specifies whether bypass bins for arithmetic-coded syntax elements are conveyed in a separate data stream. When equal to 1, the two data streams are multiplexed using a sequence of fixed-length chunks (11.3). When equal to 0, bypass bins form part of the arithmetic-coded bitstream.

cross\_attr\_prediction\_enabled specifies whether (when 1) or not (when 0) attribute values shall be coded using correlations across different types of attributes when num\_attributes is greater than 1. When cross\_attr\_prediction\_enabled is not present, it shall be inferred to be 0.

bypass\_bin\_coding\_prob\_update\_disabled specifies whether (when 1) or not (when 0) probability update for coding bypass bins shall be disabled. When bypass\_stream\_enabled is 0, bypass\_bin\_coding\_prob\_update\_disabled shall be applied. When bypass\_bin\_coding\_prob\_update\_disabled is not present, it shall be inferred to be 0.

entropy\_continuation\_enabled specifies whether (when 1) or not (when 0) the entropy parsing of a DU may depend upon the final entropy parsing state of a DU in the preceding slice. It is a requirement of bitstream conformance that entropy\_continuation\_enabled shall be 0 when slice\_reordering\_constraint is 0.

inter\_frame\_prediction\_enabled equal to 1 specifies that inter prediction may be used to derive the positions and attributes in the DU. inter\_frame\_prediction\_enabled equal to 0 specifies that inter prediction is not used to derive the positions and attributes in the DU.

inter\_entropy\_continuation\_enabled specifies whether (when 1) or not (when 0) the entropy parsing of a DU may depend upon the final entropy parsing state of a DU in the preceding frame in bitstream order. When inter\_entropy\_continuation\_enabled is not present, it shall be inferred to be 0.

It is a requirement of bitstream conformance that inter\_frame\_prediction\_enabled shall be 0 and inter\_entropy\_continuation\_enabled shall be 0 when slice\_reordering\_constraint is 0.

sps\_extension\_present specifies whether (when 1) or not (when 0) sps\_extension\_data syntax elements are present in the SPS syntax structure. sps\_extension\_present shall be 0 in bitstreams conforming to this version of this document. The value of 1 for sps\_extension\_present is reserved for future use by ISO/IEC.

sps\_extension\_data may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this document. Decoders shall ignore all sps\_extension\_data syntax elements.

fgs\_layer\_group\_enabled equals to 1 specifies that a slice comprises multiple fine granularity slices of partial slice geometry or partial slice attribute. fgs\_layer\_group\_enabled equals to 0 specifies that a slice is not comprised by fine granularity slices. When fgs\_layer\_group\_enabled equals to 1, partially decoded occupancy tree can be reconstructed as specified in Annex E. When fgs\_layer\_group\_enabled is not present, fgs\_layer\_group\_enabled is inferred to 0.

It is a requirement for bitstream conformance that fgs\_layer\_group\_enabled shall be 0 under any of the following conditions:

* geom\_tree\_type is 1, or
* occtree\_coded\_axis\_list\_present is 1, or
* geom\_scaling\_enabled is 1 and geom\_qp\_mul\_log2 is not 3, or
* geom\_angular\_enabled is 1, or
* inter\_prediction\_enabled is 1.

##### Coordinate systems

seq\_origin\_bits specifies the length in bits of each seq\_origin\_xyz syntax element exclusive of any sign bit.

seq\_origin\_xyz[ 𝑘 ] and seq\_origin\_log2\_scale together specify the XYZ origin of the sequence and coding coordinate systems in units of the sequence coordinate system from the application-specific coordinate system origin. When seq\_origin\_bits is 0, seq\_origin\_xyz[ 𝑘 ] and seq\_origin\_log2\_scale shall be inferred to be 0. The 𝑘-th XYZ component of the origin is specified by the expression SeqOrigin[ 𝑘 ].

SeqOrigin[k] := seq\_origin\_xyz[k] << seq\_origin\_log2\_scale

seq\_bbox\_size\_bits specifies the length in bits of each seq\_bbox\_size\_minus1\_xyz syntax element.

seq\_bbox\_size\_minus1\_xyz[ 𝑘 ] plus 1 specifies the 𝑘-th XYZ component of the coded volume dimensions in the sequence coordinate system. When seq\_bbox\_size\_bits is 0, the coded volume dimensions are unspecified.

seq\_unit\_numerator\_minus1, seq\_unit\_denominator\_minus1 and seq\_unit\_is\_metres together specify the length represented by the unit vectors of the sequence coordinate system.

seq\_unit\_is\_metres equal to 1 specifies that the sequence unit vectors have a length in metres equal to:

seq\_unit\_is\_metres equal to 0 specifies that the sequence unit vectors have a length relative to the application-specific coordinate system unit vector length, AppUnit, equal to:

seq\_coded\_scale\_exponent, seq\_coded\_scale\_mantissa\_bits andseq\_coded\_scale\_mantissa together specify the scale factor that converts the coding coordinate system to the sequence coordinate system. The scale factor is represented by the syntax elements as a normalized binary floating-point value that is greater than or equal to 1. seq\_coded\_scale\_mantisssa\_bits specifies the length in bits of the syntax element seq\_coded\_scale\_mantissa. The scale factor is specified by the expression SeqCodedScale.

geom\_axis\_order specifies the correspondence between the XYZ axes and the STV axes of the coded point cloud in accordance with Table 8.

Syntax elements ending in "\_xyz" are specified using the XYZ axes. The expression StvToXyz[ 𝑘 ] is the component index of the XYZ axis that corresponds to 𝑘-th STV component. Values for StvToXyz[ 𝑘 ] are specified for every geom\_axis\_order in Table 8.

Table 8 — Definition of StvToXyz[ 𝑘 ] according to the value of geom\_axis\_order

| geom\_axis\_order | Axis (𝑘) label | | | StvToXyz[ 𝑘 ] | | |
| --- | --- | --- | --- | --- | --- | --- |
| 0 (S) | 1 (T) | 2 (V) | 0 (S) | 1 (T) | 2 (V) |
| 0 or 4 | Z | Y | X | 2 | 1 | 0 |
| 1 or 7 | X | Y | Z | 0 | 1 | 2 |
| 2 | X | Z | Y | 0 | 2 | 1 |
| 3 | Y | Z | X | 1 | 2 | 0 |
| 5 | Z | X | Y | 2 | 0 | 1 |
| 6 | Y | X | Z | 1 | 0 | 2 |

##### Attributes

Attributes are identified by their index into the SPS.

num\_attributes specifies the number of attributes enumerated by the SPS attribute list.

The expressions AttrDim, AttrBitDepth and AttrMaxVal specify the number of components, the bit depth and the maximum value respectively of the attribute identified by the variable AttrIdx. The decoding of an attribute data unit sets AttrIdx.

AttrDim := attr\_components\_minus1[AttrIdx] + 1  
  
AttrBitDepth := attr\_bitdepth\_minus1[AttrIdx] + 1  
  
AttrMaxVal := Exp2(AttrBitDepth) − 1

attr\_components\_minus1[ attrIdx ] plus 1 specifies the number of components of the identified attribute.

* 1. Attributes with more than three components can only be coded as raw attribute data (attr\_coding\_type = 3).

attr\_instance\_id[ attrIdx ] specifies the instance identifier for the identified attribute.

* 1. The value of attr\_instance\_id can be used to differentiate between attributes with identical attribute labels. For example, a point cloud might have multiple colour attributes sampled from different view points. In this case, attr\_instance\_id can be used by an application to discriminate between the view points.

attr\_bitdepth\_minus1[ attrIdx ] plus 1 specifies the bit depth of every component of the identified attribute.

attr\_label\_known[ attrIdx ], attr\_label[ attrIdx ] and attr\_label\_oid[ attrIdx ] together identify the type of data conveyed by the identified attribute. attr\_label\_known[ attrIdx ] specifies whether (when 1) the attribute is an attribute specified in this document by the value of attr\_label[ attrIdx ], or (when 0) an externally specified attribute identified by the object identifier attr\_label\_oid[ attrIdx ].

Attribute types identified by attr\_label are specified in Table 9. It is a requirement of bitstream conformance that an attribute identified by attr\_label shall have only as many components as specified as valid. Values of attr\_label not specified are reserved for future use by ISO/IEC. A decoder should decode attributes with reserved values of attr\_label.

Attribute types identified by attr\_label\_oid are not specified in this document. attr\_label\_oid specifies an ASN.1 object identifier value in the international object identifier tree. The international object identifier shall either be assigned by a registration authority in accordance with Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1 or generated without registration using a universally unique identifier (UUID) as specified by Rec. ITU‑T X.667﻿ |‌ ISO/IEC 9834‑8.

Table 9 — Identification of attribute type by attr\_label

| attr\_label | Attribute type | Valid component counts |
| --- | --- | --- |
| 0 | Colour | 1 or 3 |
| 1 | Reflectance | 1 |
| 2 | Opacity | 1 |
| 3 | Frame index | 1 |
| 4 | Frame number | 1 |
| 5 | Material identifier | 1 |
| 6 | Normal vector | 3 |

attr\_property\_cnt specifies the number of attribute\_property syntax structures present in the SPS for the attribute.

#### Attribute property semantics

##### Identification of an attribute property

An attribute\_property( attrIdx ) syntax structure specifies a property of the attribute identified by attrIdx.

attr\_prop\_type specifies the attribute property type according to Table 10. The interpretation of attribute properties identified as attribute specific are specified in accordance with the registration of attr\_label\_oid.

Table 10 — Identification of attribute parameter type by attr\_prop\_type

| attr\_prop\_type | Description |
| --- | --- |
| 0 | ITU‑T T.35 user defined |
| 1 | G-PCC user defined |
| 2 | ISO/IEC 23091‑2 video code points |
| 3 | Attribute scale and offset |
| 4 | Default attribute value |
| 5 .. 127 | Reserved for future use by ISO/IEC |
| 128 .. 255 | Attribute specific |

attr\_prop\_len shall be the length in bytes of the attribute\_property syntax structure excluding the syntax elements attr\_prop\_type and attr\_prop\_len.

##### ITU‑T T.35 user defined attribute properties

ITU‑T T.35 user defined properties contain user data registered in accordance with Rec. ITU‑T T.35. The user data are not specified by this document.

attr\_prop\_itu\_t\_t35\_country\_code is a byte having a value specified as a country code by Annex A of Rec. ITU‑T T.35.

attr\_prop\_itu\_t\_t35\_country\_code\_extension\_byte is a byte having a value specified as a country code by Annex B of Rec. ITU‑T T.35.

The ITU‑T T.35 terminal provider code and terminal provider oriented code shall be contained in the initial bytes of attr\_prop\_byte[ ], in the format specified by the administration that issued the terminal provider code. Any remaining attr\_prop\_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.

##### G-PCC user defined attribute properties

G-PCC user defined properties contain user data identified by an ASN.1 object identifier. The user data are not specified by this document.

attr\_prop\_oid specifies an ASN.1 object identifier value in the international object identifier tree in accordance with Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1.

Any attr\_prop\_byte data present shall be data having syntax and semantics as specified in accordance with the registration of the object identifier.

##### ISO/IEC 23091‑2 video code points

ISO/IEC 23091‑2 video code points establish properties of a video representation.

attr\_cicp\_colour\_primaries[ attrIdx ] specifies the chromaticity coordinates of the attribute's colour primaries in accordance with the ColourPrimaries code point in ISO/IEC 23091‑2.

attr\_cicp\_transfer\_characteristics[ attrIdx ] specifies, in accordance with the TransferCharacteristics code point in ISO/IEC 23091‑2, either the:

* reference opto-electronic transfer characteristic function of the attribute as a function of a source input, linear, optical intensity with a nominal real-valued range of 0 to 1; or
* inverse of the reference electro-optical transfer characteristic function as a function of an output, linear, optical intensity with a nominal real-valued range of 0 to 1.

attr\_cicp\_matrix\_coeffs[ attrIdx ] describes the matrix coefficients used to derive the attribute's luma and chroma signals from the green, blue and red, or *Y*, *Z* and *X* primaries in accordance with the MatrixCoefficients code point in ISO/IEC 23091‑2.

attr\_cicp\_video\_full\_range[ attrIdx ] specifies the black level and range of the attribute's luma and chroma signals as derived from , and , or , and real-valued component signals in accordance with the VideoFullRangeFlag code point in ISO/IEC 23091‑2.

##### Scale and offset properties

Attribute scale and offset parameters specify how to interpret the range of output attribute values.

The decoding process in this document does not scale attribute values prior to output.

attr\_offset\_bits is the length in bits of the subsequent attr\_offset[ attrIdx ] syntax element exclusive of any sign bit.

attr\_scale\_bits is the length in bits of the subsequent attr\_scale\_minus1[ attrIdx ] syntax element.

attr\_offset[ attrIdx ], attr\_scale\_minus1[ attrIdx ] and attr\_frac\_bits[ attrIdx ] together specify how coded attribute values shall be interpreted. When present, the external interpretation of each coded attribute value shall be:

##### Default attribute value

A default attribute value property specifies the value for an attribute that is not otherwise determined by an ADU.

attr\_default\_value[ attrIdx ][ 𝑐 ] specifies the default value of the 𝑐-th component of the identified attribute. The length in bits of each syntax element shall be attr\_bitdepth\_minus1[ attrIdx ] + 1.

#### Attribute property data semantics

attr\_prop\_byte[ 𝑖 ] is a byte containing data having syntax and semantics not specified in this document.

#### Tile inventory data unit semantics

A tile inventory, when present, contains metadata that defines the spatial region of each enumerated tile. Each tile is identified by either an implicit or explicit tile id.

A tile inventory shall apply from the next coded point cloud frame that follows the tile inventory data unit. It shall remain valid until it is replaced by another tile inventory.

A tile inventory DU shall occur before the first GDU of the coded point cloud frame from which it applies. It shall not occur before the last DU of any coded point cloud frame that precedes that from which it applies in data unit order.

ti\_seq\_parameter\_set\_id identifies the active SPS by its sps\_seq\_parameter\_set\_id.

ti\_frame\_ctr\_lsb\_bits specifies the length in bits of the syntax element ti\_frame\_ctr\_lsb. It is a requirement of bitstream conformance that ti\_frame\_ctr\_lsb\_bits shall be equal to frame\_ctr\_lsb\_bits of the active SPS.

ti\_frame\_ctr\_lsb should be the ti\_frame\_ctr\_lsb\_bits LSBs of FrameCtr for the next coded point cloud frame.

tile\_cnt specifies the number of tiles enumerated by the tile inventory.

tile\_id\_bits specifies the length in bits of each tile\_id syntax element. tile\_id\_bits equal to 0 specifies that tiles shall be identified by the index tileIdx.

tile\_origin\_bits\_minus1 plus 1 specifies the length in bits of each tile\_origin\_xyz syntax element exclusive of any sign bit.

tile\_size\_bits\_minus1 plus 1 specifies the length in bits of each tile\_size\_minus1\_xyz syntax element.

tile\_id[ tileIdx ] specifies the identifier of the tileIdx-th tile in the tile inventory. When tile\_id\_bits is 0, the value of tile\_id[ tileIdx ] shall be inferred to be tileIdx. It is a requirement of bitstream conformance that all values of tile\_id shall be unique within a tile inventory.

tile\_origin\_xyz[ tileId ][ 𝑘 ] and tile\_size\_minus1\_xyz[ tileId ][ 𝑘 ] indicate a bounding box in the sequence coordinate system encompassing slices identified by slice\_tag equal to tileId.

tile\_origin\_xyz[ tileId ][ 𝑘 ] specifies the 𝑘-th XYZ coordinate of the tile bounding box's lower corner relative to the tile inventory origin.

tile\_size\_minus1\_xyz[ tileId ][ 𝑘 ] plus 1 specifies the 𝑘-th XYZ dimension of the tile bounding box.

ti\_origin\_bits\_minus1 plus 1 specified the length in bits of each ti\_origin\_xyz syntax element exclusive of any sign bit.

ti\_origin\_xyz[ 𝑘 ] and ti\_origin\_log2\_scale together indicate the XYZ origin of the sequence coordinate system specified by seq\_origin\_xyz[ 𝑘 ] and seq\_origin\_log2\_scale. The values of ti\_origin\_xyz[ 𝑘 ] and ti\_origin\_log2\_scale should be equal to seq\_origin\_xyz[ 𝑘 ] and seq\_origin\_log2\_scale, respectively.

The tile inventory's 𝑘-th XYZ origin coordinate is specified by the expression TileInventoryOrigin[ 𝑘 ].

TileInventoryOrigin[k] := ti\_origin\_xyz[k] << ti\_origin\_log2\_scale

#### Geometry parameter set data unit semantics

##### General parameters

The parameters specified by a GPS shall apply to any DU where that GPS is activated.

gps\_geom\_parameter\_set\_id identifies the GPS for reference by other DUs.

gps\_seq\_parameter\_set\_id identifies the active SPS by its sps\_seq\_parameter\_set\_id.

slice\_geom\_origin\_scale\_present specifies whether (when 1) or not (when 0) slice\_geom\_origin\_log2\_scale is present in the GDU header. slice\_geom\_origin\_scale\_present equal to 0 specifies that the slice origin scale is specified by gps\_geom\_origin\_log2\_scale.

gps\_geom\_origin\_log2\_scale specifies the scale factor used to derive the slice origin from slice\_geom\_origin\_xyz when slice\_geom\_origin\_scale\_present is 0.

geom\_dup\_point\_counts\_enabled specifies whether (when 1) or not (when 0) duplicate points can be signalled in a GDU by a per-point duplication count.

geom\_dup\_point\_counts\_enabled equal to 0 does not prohibit the coding of the same point position multiple times within a single slice by means other than the direct\_dup\_point\_cnt, occ\_dup\_point\_cnt or ptn\_dup\_point\_cnt syntax elements.

geom\_tree\_type equal to 0 specifies that slice geometry is coded using an occupancy tree (7.3.3.4). geom\_tree\_type equal to 1 specifies that slice geometry is coded using a predictive tree (7.3.3.8).

gps\_extension\_present specifies whether (when 1) or not (when 0) gps\_extension\_data syntax elements are present in the GPS syntax structure. gps\_extension\_present shall be 0 in bitstreams conforming to this version of this document. The value of 1 for gps\_extension\_present is reserved for future use by ISO/IEC.

gps\_extension\_data may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this document. Decoders shall ignore all gps\_extension\_data syntax elements.

##### Angular coding parameters

geom\_angular\_enabled specifies whether (when 1) or not (when 0) slice geometry is coded using information about a set of beams located along and rotating around the V axis of the angular origin. When enabled, point positions are assumed to have been sampled along a ray cast by a beam.

The angular origin AngularOrigin, the apparent V-axis offset BeamOffsetV, the elevation angle 𝜃 of emitted rays and the rotation step angle 𝜑 advanced between ray emissions are illustrated for a single beam in Figure 4.

**图示

描述已自动生成**

Figure 4 — Origin, elevation angle and azimuth step for a beam.

slice\_angular\_origin\_present specifies whether (when 1) or not (when 0) a slice-relative angular origin is signalled in the GDU header. slice\_angular\_origin\_present equal to 0 specifies that the angular origin is gps\_angular\_origin\_xyz. When slice\_angular\_origin\_present is not present, it shall be inferred to be 0.

gps\_angular\_origin\_bits\_minus1 plus 1 specifies the length in bits of each gps\_angular\_origin\_xyz syntax element.

gps\_angular\_origin\_xyz[ 𝑘 ] specifies the 𝑘-th XYZ coordinate of the angular origin in the coding coordinate system.

num\_beams\_minus1 plus 1 specifies the number of beams enumerated by the GPS.

beam\_elevation\_init and beam\_elevation\_diff[ 𝑖 ] together specify beam elevations as gradients above the S-T plane. The elevation gradient for the 𝑖-th beam is specified by the expression BeamElev[ 𝑖 ]. It is a binary fixed-point value with 18 fractional bits.

BeamElev[i] :=  
 i == 0 ? beam\_elevation\_init :  
 i == 1 ? beam\_elevation\_init + beam\_elevation\_diff[1]  
 : 2 × BeamElev[i – 1] – BeamElev[i − 2] + beam\_elevation\_diff[i]

It is a requirement of bitstream conformance that values of BeamElev[ 𝑖 ], 𝑖 ∈ 1 .. num\_beams\_minus1, shall be greater than BeamElev[ 𝑖 – 1 ].

beam\_voffset\_init and beam\_voffset\_diff[ 𝑖 ] together specify the V-axis offsets of the enumerated beams from the angular origin. The offset is specified in units of the coding coordinate system. The offset for the 𝑖-th beam is specified by the expression BeamOffsetV[ 𝑖 ].

BeamOffsetV[i] :=  
 i == 0 ? beam\_voffset\_init  
 : BeamOffsetV[i − 1] + beam\_voffset\_diff[i]

beam\_steps\_per\_rotation\_init\_minus1 and beam\_steps\_per\_rotation\_diff[ 𝑖 ] specify the number of steps made per revolution by the rotating beams. The value for the 𝑖-th beam is specified by the expression BeamStepsPerRev[ 𝑖 ].

BeamStepsPerRev[i] :=  
 i == 0 ? beam\_steps\_per\_rotation\_init\_minus1 + 1  
 : BeamStepsPerRev[i – 1] + beam\_steps\_per\_rotation\_diff[i]

It is a requirement of bitstream conformance that values of BeamStepsPerRev[ 𝑖 ], 𝑖 ∈ 0 .. num\_beams\_minus1, shall not be 0.

ptree\_ang\_azimuth\_pi\_bits\_minus11 plus 11 specifies the number of bits that represent half a turn of a beam around the V axis. One half-turn is 𝜋 radians.

ptree\_ang\_radius\_scale\_log2 specifies a factor used to scale a point's radial angular coordinate during conversion to Cartesian coordinates.

ptree\_ang\_azimuth\_step\_minus1 plus 1 specifies the expected change in azimuth angle of the rotating beams between coded points. Azimuth prediction residuals used in angular predictive tree coding can be coded as a multiple of ptree\_ang\_azimuth\_step\_minus1 + 1 and a remainder.

occtree\_angular\_extension\_enabled specifies whether (when 1) or not (when 0) angular coding extension shall be used with occupancy tree coding. The effect is to use extended coding tools for improving coding performances. When occtree\_angular\_extension\_enabled is not present, it shall be inferred to be 0.

octree\_planar\_neigh\_prediction\_enabled specifies whether (when 1) or not (when 0) neighbour prediction coding shall be used with planar occupancy coding. When octree\_planar\_neigh\_prediction\_enabled is not present, it shall be inferred to 0.

geo\_disable\_planar\_idcm\_angular equal to 1 specifies that if the geom\_angular\_enabled is equal to 1, the planar mode is disabled for the IDCM coded nodes. If geom\_angular\_enabled is equal to 0, geo\_disable\_planar\_idcm\_angular is set equal to 0. When geo\_disable\_planar\_idcm\_angular is equal to 1, occ\_direct\_node should be derived before deciding per axis planar eligibility.

ptree\_sec\_resid\_disabled specifies whether (when 1) or not (when 0) second coordinate-prediction residual coding shall be disabled. When ptree\_sec\_resid\_disabled is not present, it shall be inferred to be 0.

ptree\_ang\_azimuth\_scaling\_enabled specifies whether (when 1) or not (when 0) predictive geometry azimuth angle residuals shall be scaled according to the radius. The effect is to provide an adaptive quantization step size of the predictive geometry azimuth angle residuals resulting in a uniform quantization of circular arcs for any radius. When ptree\_ang\_azimuth\_scaling\_enabled is not present, it shall be inferred to be 0.

ptree\_ang\_max\_pred\_index specifies the maximum predictor index being usable in the prediction list for angular coordinates.

ptree\_ang\_pred\_list\_radius\_resid\_threshold specifies a threshold value on the absolute value of predictive geometry radius residual. This threshold is used during the dynamic update process of the prediction list for angular coordinates (9.3.4.7).

ptree\_ang\_radius\_resid\_context\_qphi\_threshold\_presentspecifies whether (when 1) or not (when 0) the threshold of the number of azimuthal angle steps used in selecting the context of the decoding radius residual is present in ptree\_ang\_redius\_resid\_context\_qphi\_threshold. When ptree\_ang\_radius\_resid\_context\_qphi\_threshold\_present is not present, it shall be inferred to be 0.

ptree\_ang\_redius\_resid\_context\_qphi\_threshold specifies the threshold of the number of azimuthal angle steps used in selecting the context of the decoding radius residual. When ptree\_ang\_redius\_resid\_context\_qphi\_threshold is not present, it shall be inferred to be 0.

The value thQphi specify the threshold used in the table of values of CtxTbl and CtxIdx for binarized ae(v) coded GDU syntax elements (11.5.3.4).

thQphi := ptree\_ang\_redius\_resid\_context\_qphi\_threshold

##### Occupancy tree parameters

occtree\_point\_cnt\_list\_present specifies whether (when 1) or not (when 0) the GDU footer enumerates the number of points in each occupancy tree level. When occtree\_point\_cnt\_list\_present is not present, it shall be inferred to be 0.

occtree\_direct\_coding\_mode greater than 0 specifies that point positions may be coded by eligible direct nodes of the occupancy tree. occtree\_direct\_coding\_mode equal to 0 specifies that direct nodes shall not be present in the occupancy tree.

Larger values for occtree\_direct\_coding\_mode generally increase the rate of direct node eligibility.

occtree\_direct\_joint\_coding\_enabled specifies whether (when 1) or not (when 0) direct nodes that code two points shall jointly code their positions according to a specific ordering of the points.

occtree\_coded\_axis\_list\_present equal to 1 specifies that the GDU header contains occtree\_coded\_axis syntax elements that are used to derive the node size for each occupancy tree level. occtree\_coded\_axis\_list\_present equal to 0 specifies that occtree\_coded\_axis syntax elements are not present in the GDU syntax and that the occupancy tree represents a cubic volume specified by the tree depth.

occtree\_neigh\_window\_log2\_minus1 plus 1 specifies the number of occupancy tree node locations that form each availability window within a tree level. Nodes outside a window are unavailable to any process related to nodes within the window. occtree\_neigh\_window\_log2\_minus1 equal to 0 specifies that only sibling nodes shall be considered available to the current node.

occtree\_adjacent\_child\_enabled specifies whether (when 1) or not (when 0) the adjacent children of neighbouring occupancy tree nodes are used in bitwise occupancy contextualization. When occtree\_adjacent\_child\_enabled is not present, it shall be inferred to be 0.

occtree\_intra\_pred\_max\_nodesize\_log2 minus 1 specifies the maximum size of an occupancy tree node that is eligible for intra-slice occupancy prediction. When occtree\_intra\_pred\_max\_nodesize\_log2 is not present, it shall be inferred to be 0.

occtree\_bitwise\_coding specifies whether the node occupancy bitmap is coded using (when 1) occupancy\_bit syntax elements or (when 0) the dictionary coded syntax element occupancy\_byte.

occtree\_planar\_enabled specifies whether (when 1) or not (when 0) the coding of node occupancy bitmaps is performed, in part, by the signalling of occupied and unoccupied planes. When occtree\_planar\_enabled is not present, it shall be inferred to be 0.

occtree\_planar\_threshold[ 𝑖 ] specify thresholds used in part to determine the per-axis eligibility for planar occupancy coding. The thresholds are specified from the most (𝑖 = 0) to the least (𝑖 = 2) probable planar axis. Each threshold specifies the minimum likelihood for an eligible axis that occ\_single\_plane is expected to be 1. The range [ 8, 120 ] for occtree\_planar\_threshold corresponds to the likelihood interval [ 0, 1 ).

occtree\_direct\_node\_rate\_minus1 specifies, when present, that of every 32 eligible nodes, only occtree\_direct\_node\_rate\_minus1 + 1 are permitted to be coded as direct nodes.

occtree\_planar\_buffer\_disabled specifies whether (when 1) or not (when 0) the contextualization of per-node occupied plane locations using the plane locations of previously coded nodes shall be disabled. When occtree\_planar\_buffer\_disabled is not present, it shall be inferred to be 0.

##### Scaling parameters

geom\_scaling\_enabled specifies whether (when 1) or not (when 0) the coded geometry shall be scaled during the geometry decoding process.

geom\_qp specifies the geometry QP prior to the addition of per slice and per-node offsets.

geom\_qp\_mul\_log2 specifies the scale factor to be applied to the geometry QP. There are Exp2( 3 − geom\_qp\_mul\_log2 ) QP values for every doubling of the scaling step size.

ptree\_qp\_period\_log2 specifies the period in nodes at which the predictive tree node QP offset is signalled. The period is one in every Exp2( ptree\_qp\_period\_log2 ) nodes.

occtree\_direct\_node\_qp\_offset specifies an offset relative to the slice geometry QP for scaling direct node coded point positions.

##### Inter prediction parameters

inter\_prediction\_enabled specifies whether (when 1) or not (when 0) inter prediction may be used to code the points of the point cloud. When inter\_prediction\_enabled is not present, it shall be inferred to be 0.

It is a requirement of bitstream conformance that when inter\_frame\_enabled\_flag is 0, inter\_prediction\_enabled shall be 0.

biprediction\_enabled specifies whether (when 1 or 2) or not (when 0) bi-prediction may be used to code the points of the point cloud. When biprediction\_enabled is not present, it shall be inferred to be 0.

frame\_merge\_enabled specifies whether (when 1) or not (when 0) the two reference frames of bi-prediction may be merged into a merged reference frame. When frame\_merge\_enabled is not present, it shall be inferred to be 0.

global\_motion\_enabled specifies whether (when 1) or not (when 0) global motion compensation is applied to the reference frame used for inter prediction. When global\_motion\_enabled is not present, it shall be inferred to be 0.

inter\_azim\_scale\_log2 specifies a scale factor to be applied to azimuth coordinates used to obtain azimuth look-up values during inter prediction. The values MaxQAzim and MinQAzim specify the maximum and minimum azimuth look up values.

MaxQAzim [i] := 1 << ptree\_ang\_azimuth\_pi\_bits\_minus11 + 11 – inter\_azim\_scale\_log2 - 1

MinQAzim [i] := -(1 << ptree\_ang\_azimuth\_pi\_bits\_minus11 + 11 – inter\_azim\_scale\_log2)

resampling\_enabled specifies a whether (when 1) or not (when 0) the radii of one or more points in a reference frame is updated using motion parameters.

max\_points\_per\_entry\_minus1 plus one specifies the maximum number of points of a reference frame that may be stored for a given laser ID and azimuth look up value derived using inter\_azim\_scale\_log2.

down\_sampling\_range specifies the range of downsampling a reference frame. The syntax down\_sampling\_range may be present only when max\_points\_per\_entry\_minus1 is greater than to 0. When down\_sampling\_range is not present, it shall be inferred to be -1.

occtree\_inter\_angular\_direct\_coding\_enabled specifies whether (when 1) or not (when 0) use angular information to determine the eligibility for direct coding of the occupancy tree nodes. When occtree\_inter\_angular\_direct\_coding\_enabled is not present, it shall be inferred to be 0.

##### TriSoup parameters

trisoup\_enable\_flag specifies whether (when 1) or not (when 0) an occupancy tree (7.3.3.4) used for coding slice geometry is followed by TriSoup (7.3.3.10) The syntax trisoup\_enable\_flag may be present only when geom\_tree\_type is equal to 0. When not present, trisoup\_enable\_flag is inferred to 0.

trisoup\_non\_cubic\_node\_start\_edge\_presence\_flag and trisoup\_non\_cubic\_node\_end\_edge\_presence\_flag together specifies whether (when one or both are 1) or not (when both are 0) the location TriSoupNodeLoc[ nodeIdx ][ k ] and the edge length TriSoupNodeSize[ nodeIdx ][ k ] of the nodeIdx-th TriSoup node are modified as specified in 9.4.1.1.

#### Attribute parameter set data unit semantics

##### General parameters

The parameters specified by an APS shall apply to any DU where that APS is activated.

A single APS can be used by multiple coded attributes. The attributes are not required to be of the same type or to have the same number of components.

aps\_attr\_parameter\_set\_id identifies the APS for reference by other DUs.

aps\_seq\_parameter\_set\_id identifies the active SPS by its sps\_seq\_parameter\_set\_id.

attr\_coding\_type specifies the attribute coding method. Valid values are specified by Table 11. Other values are reserved for future use by ISO/IEC. Decoders conforming to this version of this document shall ignore (remove from the bitstream and discard) attribute data units coded with reserved values of attr\_coding\_type.

Table 11 — Interpretation of attr\_coding\_type

| attr\_coding\_type | Description | Decoding process |
| --- | --- | --- |
| 0 | Region Adaptive Hierarchical Transform (RAHT) | 10.5 |
| 1 | LoD with Predicting Transform | 10.6 |
| 2 | LoD with Lifting Transform | 10.6 |
| 3 | Raw attribute data | 10.3 |

attr\_primary\_qp\_minus4 plus 4 specifies the QP for the primary attribute component before the addition of per slice, per region and per-transform-level offsets.

attr\_secondary\_qp\_offset specifies an offset to be applied to the primary attribute QP to derive the QP for any secondary attribute components.

cross\_attr\_prediction\_enabled\_this\_type specifies whether (when 1) or not (when 0) the cross-attribute prediction is enabled for coding the current attribute if cross\_attr\_prediction\_enabled is 1. When cross\_attr\_prediction\_enabled\_this\_type is not present, it shall be inferred to be 0.

refAttrIdx specifies the index of attribute identified by its attrIdx that is used for decoding the current attribute. It shall range from 0 to num\_attributes - 1 when cross\_attr\_\_prediction\_enabled\_this\_type is 1. When refAttrIdx is not present, it shall be inferred to be - 1.

attr\_qp\_offsets\_present specifies whether (when 1) or not (when 0) per-slice attribute QP offsets, attr\_qp\_offset[ 𝑐 ], are present in the ADU header.

attr\_coord\_conv\_enabled specifies whether (when 1) attribute coding shall use scaled angular coordinates or (when 0) slice-relative STV point positions. It is a requirement of bitstream conformance that attr\_coord\_conv\_enabled shall be 0 when geom\_angular\_enabled is 0. When attr\_coord\_conv\_enabled is not present, it shall be inferred to be 0.

attr\_coord\_conv\_scale\_bits\_minus1[ 𝑘 ] plus 1 specifies the length in bits of the syntax element attr\_coord\_conv\_scale[ 𝑘 ].

attr\_coord\_conv\_scale[ 𝑘] specifies the scale factor used to scale points' 𝑘-th angular coordinate for attribute coding. The scale factor shall be in units of .

aps\_extension\_present specifies whether aps\_extension\_data syntax elements are present in the APS syntax structure. aps\_extension\_present shall be 0 in bitstreams conforming to this version of this document. The value of 1 for aps\_extension\_present is reserved for future use by ISO/IEC.

aps\_extension\_data may have any value. Its presence and value do not affect decoder conformance to profiles specified in this version of this document. Decoders shall ignore all aps\_extension\_data syntax elements.

##### Region adaptive hierarchical transform parameters

raht\_prediction\_enabled specifies whether (when 1) or not (when 0) RAHT coefficients are predicted by upsampling and transforming the preceding coarser transform level.

raht\_prediction\_subtree\_min and raht\_prediction\_samples\_min specify thresholds that control the use of RAHT coefficient prediction.

**raht\_prediction\_samples\_min** specifies the minimum number of spatially adjacent samples from which RAHT coefficient prediction can be performed.

**raht\_prediction\_subtree\_min** specifies the minimum number of spatially adjacent samples that need to be present to prevent the disabling of RAHT coefficient prediction for every descendant of a RAHT node.

lossless\_coding\_enabled specifies whether (when 1) or not (when 0) lossless coding is applied by using Haar Transform. When lossless\_coding\_enabled is not present, it shall be inferred to be 0.

raht\_subnode\_prediction\_enabledspecifies whether (when 1) or not (when 0) the adjoining blocks are replaced by their children blocks when generate the upsampled prediction block. When raht\_subnode\_prediction\_enabled is not present, it shall be inferred to be 0.

raht\_intra\_layer\_code\_enabled specifies whether (when 1) or not (when 0) per-RAHT-layer AC coding mode is present in the ADU. It is a requirement of bitstream conformance that raht\_intra\_layer\_code\_enabled shall be 0 when raht\_prediction\_enabled is 0. When raht\_intra\_layer\_code\_enabled is not present, it shall be inferred to be 0.

raht\_prediction\_weights[  ] specifies the prediction weights applied to the normalized DC value of the blocks in transform domain prediction. raht\_prediction\_weights[ ] specify the prediction weights for the co-located block, blocks that adjoin by a face, blocks that adjoin by only an edge, child blocks that adjoin by a face and child blocks that adjoin by only an edge, respectively.

raht\_buffer\_extension\_flag specifies whether (when 1) or not (when 0) to use the extended reconstruction buffer for upsampling the attributes. When equal to 1, the attributes of previous layers are stored without rounding. When equal to 0, the attributes of previous layers are rounded prior to be stored to the buffer. When raht\_buffer\_extension\_flag is not present, it shall be inferred to be 0.

raht\_prediction\_search\_range specifies the range of Morton code in the preceding coarser transform level for which searched for spatially adjacent samples. When raht\_prediction\_search\_range is not present, it shall be inferred to be MaxSlicePoints – 1.

raht\_last\_comp\_pred\_enabled specifies whether (when 1) or not (when 0) the second coefficient component of a three-component attribute shall be used to predict the value of the third coefficient component when RAHT is used for attribute coding. When raht\_last\_comp\_pred\_enabled is not present, it shall be inferred to be 0.

##### Level of detail generation and transform parameters

pred\_set\_size\_minus1 plus 1 specifies the maximum size of the per-point predictor set.

pred\_inter\_lod\_search\_range specifies the range of indexes around a search centre which can be searched in an extended inter-detail-level search for nearest neighbours to include in a point's predictor set.

pred\_dist\_bias\_minus1\_xyz[ 𝑘 ] plus 1 specifies the factor used to weight the 𝑘-th XYZ component of the distance vector between two point positions used to calculate inter-point distances in the predictor search for a single refinement point. The expression PredBias[ 𝑘 ] specifies the factor for the 𝑘-th STV component.

PredBias[k] := pred\_dist\_bias\_minus1\_xyz[StvToXyz[k]] + 1

last\_comp\_pred\_enabled specifies whether (when 1) or not (when 0) the second coefficient component of a three-component attribute shall be used to predict the value of the third coefficient component. When last\_comp\_pred\_enabled is not present, it shall be inferred to be 0.

lod\_scalability\_enabled specifies whether (when 1) or not (when 0) attribute values shall be coded using constrained LoD generation and predictor searches. When equal to 1, attribute values can be reconstructed for a partially decoded occupancy tree as specified in Annex D. When lod\_scalability\_enabled is not present, it shall be inferred to be 0.

It is a requirement of bitstream conformance that lod\_scalability\_enabled shall be 0 when any of the following conditions are true:

* geom\_tree\_type is 1, or
* occtree\_coded\_axis\_list\_present is 1, or
* geom\_scaling\_enabled is 1 and geom\_qp\_mul\_log2 is not 3, or
* pred\_blending\_enabled is 1.

pred\_max\_range\_minus1 plus 1 specifies, when present, the distance beyond which point predictor candidates shall be discarded during predictor set pruning for scalable attribute coding. The distance is specified in units of the per-detail-level block size.

lod\_max\_levels\_minus1 plus 1 specifies the maximum number of detail levels that can be generated by the LoD generation process. When lod\_max\_levels\_minus1 is not present, it shall be inferred to be MaxSliceDimLog2 − 1.

attr\_canonical\_order\_enabled specifies whether (when 1) or not (when 0) the order in which point attributes are coded is the canonical order that points are output by the geometry decoding processes specified in this document. When attr\_canonical\_order\_enabled is not present, it shall be inferred to be 0.

lod\_decimation\_mode specifies the decimation method used to generate detail levels. Valid values are specified by Table 12. Other values are reserved for future use by ISO/IEC. Decoders conforming to this version of this document shall ignore (remove from the bitstream and discard) attribute data units coded with reserved values of lod\_decimation\_mode.

Table 12 — Interpretation of lod\_decimation\_mode

| lod\_decimation\_mode | Description | Decoding process |
| --- | --- | --- |
| 0 | No decimation | 10.6.5.6 |
| 1 | Periodic subsampling | 10.6.5.5 |
| 2 | Block-based subsampling | 10.6.5.8 |

lod\_sampling\_period\_minus2[ lvl ] plus 2 specifies the sampling period used by LoD generation to sample points in detail level lvl to generate the next coarser detail level lvl + 1.

lod\_initial\_dist\_log2 specifies the block size at the finest detail level for use by LoD generation and predictor searches. When lod\_initial\_dist\_log2 is not present, it shall be inferred to be 0.

lod\_dist\_log2\_offset\_present specifies whether (when 1) or not (when 0) the per-slice block-size offset specified by lod\_dist\_log2\_offset shall be present in the ADU header. When lod\_dist\_log2\_offset\_present is not present, it shall be inferred to be 0.

pred\_direct\_max\_idx\_plus1 specifies the maximum number of single point predictors that can be used for direct prediction.

pred\_direct\_threshold specifies when a point shall be eligible for direct prediction. The threshold is for the maximum difference between predictor values in a point's predictor set. When the maximum difference is greater than or equal to the threshold, direct prediction is eligible. When the attribute bit depth is greater than eight bits, the threshold shall be scaled by Exp2( AttrBitDepth − 8 ).

pred\_direct\_avg\_disabled specifies whether (when 0) or not (when 1) the point predictor set average is a direct prediction mode.

pred\_intra\_lod\_search\_range specifies the range of indexes in a detail level's refinement list for which searched for nearest neighbours to include in a point's predictor set.

pred\_intra\_min\_lod specifies the finest detail level in which intra-detail-level prediction is enabled. When pred\_intra\_min\_lod is not present, it shall be inferred to be lod\_max\_levels\_minus1 + 1. It is a requirement of bitstream conformance that pred\_intra\_min\_lod shall be 0 when lod\_max\_levels\_minus1 is 0.

inter\_comp\_pred\_enabled specifies whether (when 1) or not (when 0) the first component of a multi-component attribute coefficient shall be used to predict the coefficients of any subsequent components. When inter\_comp\_pred\_enabled is not present, it shall be inferred to be 0.

pred\_blending\_enabled specifies whether (when 1) or not (when 0) the neighbour weights used for neighbourhood average prediction shall be blended according to the relative spatial positions of the associated points. When pred\_blending\_enabled is not present, it shall be inferred to be 0.

quant\_neigh\_weight[k] specifies the weights of k-nearest neighbor points used by point quantization weights derivation. When quant\_neigh\_weight[k] is not present, it shall be inferred to be 0.

max\_points\_per\_sort\_log2\_plus1 minus 1 specifies the max group size that points are ordered by group before point attributes are coded. When max\_points\_per\_sort\_log2\_plus1 is equal to 0, the max group size is equal to the points number in slice. When max\_points\_per\_sort\_log2\_plus1 is not present, it shall be inferred to be 0.

prediction\_with\_distribution\_enabled specifies whether (when 1) or not (when 0) prediction coefficients are derived based on the spatial distribution of the predictors. When prediction\_with\_distribution\_enabled is not present, it shall be inferred to be 0.

##### Raw attribute parameters

raw\_attr\_width\_present specifies whether (when 0) raw attribute values shall use the same fixed length encoding for every syntax element or (when 1) a per-syntax-element length.

##### Attribute inter prediction parameters

attr\_inter\_prediction\_enabled specifies whether (when 1) or not (when 0) inter prediction may be used to code the attributes of the point cloud. When attr\_inter\_prediction\_enabled is not present, it shall be inferred to be 0.

It is a requirement of bitstream conformance that when inter\_frame\_enabled\_flag is 0 or inter\_prediction\_enabled is 0, attr\_inter\_prediction\_enabled shall be 0.

raht\_inter\_layer\_depth\_minus1 +1 specifies the number of layers of the transform tree where the inter prediction block (10.5.5.9) is used to modify the prediction block. When raht\_inter\_layer\_depth\_minus1 is not present, it shall be inferred to be 0.

raht\_send\_inter\_filters specifies whether (when 1) or not (when 0) RAHT inter-prediction (temporal) filters will be sent to the decoder. When raht\_send\_inter\_filters is not present, it shall be inferred to be 0.

raht\_inter\_skip\_layers specifies the number of layers of the transform tree for which the inter prediction block is skipped from the temporal filtering process (10.5.5.11). When raht\_inter\_skip\_layers is not present, it shall be inferred to be 0.

raht\_inter\_layer\_code\_enabled specifies whether (when 1) or not (when 0) per-RAHT-layer AC coding mode is present in the ADU. It is a requirement of bitstream conformance that raht\_inter\_layer\_code\_enabled shall be 0 when attr\_inter\_prediction\_enabled is 0. When raht\_inter\_layer\_code\_enabled is not present, it shall be inferred to be 0.

**attr\_inter\_prediction\_search\_range** specifies the range of indexes in a detail level's refinement list in the reference slice for which searched for nearest neighbours to include in a point's predictor set. When attr\_inter\_prediction\_search\_range is not present, it shall be inferred to be 0.

#### Frame-specific attribute properties data unit semantics

Frame-specific attribute properties apply to an attribute of a specific frame. The properties shall:

* override any corresponding properties signalled in the active SPS for the specified frame only;
* apply to all ADUs in the frame with AttrIdx equal to fsap\_sps\_attr\_idx.

All attribute properties with the same value of attr\_prop\_type shall be identical within a frame for any single attribute.

Each FSAP DU shall occur, at least, before the first ADU within the frame to which it applies.

The requirements of fsap\_frame\_ctr\_lsb prevent an FSAP DU from preceding the first GDU in the frame to which it applies.

fsap\_seq\_parameter\_set\_id identifies the active SPS by its sps\_seq\_parameter\_set\_id.

fsap\_frame\_ctr\_lsb\_bits specifies the length in bits of the syntax element fsap\_frame\_ctr\_lsb. It is a requirement of bitstream conformance that fsap\_frame\_ctr\_lsb\_bits shall be equal to frame\_ctr\_lsb\_bits of the active SPS.

fsap\_frame\_ctr\_lsb identifies the frame to which the frame-specific attribute properties apply. Identification shall use fsap\_frame\_ctr\_lsb\_bits LSBs of the notional frame counter, FrameCtr. fsap\_frame\_ctr\_lsb shall be equal to frame\_ctr\_lsb of the preceding GDU.

fsap\_sps\_attr\_idx identifies the coded attribute to which the frame-specific attribute properties shall apply. Identification shall be by the index into the active SPS attribute list.

fsap\_num\_props specifies the number of attribute properties present in the syntax structure.

#### Frame boundary marker data unit semantics

The frame boundary marker DU explicitly marks the end of a frame.

fbdu\_frame\_ctr\_lsb\_bits specifies the length in bits of the syntax element fbdu\_frame\_ctr\_lsb. It is a requirement of bitstream conformance that fbdu\_frame\_ctr\_lsb\_bits shall be equal to frame\_ctr\_lsb\_bits of the active SPS.

fbdu\_frame\_ctr\_lsb identifies the frame to which the frame boundary marker applies. Identification shall use fbdu\_frame\_ctr\_lsb\_bits LSBs of the notional frame counter FrameCtr.

#### User data data unit semantics

The user data DU contains user data identified by an ASN.1 object identifier. The user data are not specified by this document.

user\_data\_oid specifies an ASN.1 object identifier value in the international object identifier tree, as specified in Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1.

user\_data\_byte is a byte containing data having syntax and semantics as specified by the registration of the object identifier.

#### Byte alignment semantics

The byte\_alignment syntax structure causes the bitstream to become byte-aligned.

alignment\_bit\_equal\_to\_zero shall be 0.

### Geometry data unit

#### Geometry data unit semantics

A GDU conveys the geometry of a slice and associated slice information such as a frame counter or a slice origin. A GDU comprises a GDU header, geometry coded using either an occupancy tree (when geom\_tree\_type is 0) or a predictive tree (when geom\_tree\_type is 1), and a GDU footer.

#### Geometry data unit header semantics

gdu\_geometry\_parameter\_set\_id specifies the active GPS by its gps\_geom\_parameter\_set\_id.

gdu\_temporal\_id specifies the temporal ID of the frame associated with the geometry data unit.

slice\_id identifies the slice for reference by other DUs.

slice\_tag identifies the slice as a member of a slice group with the same values for slice\_tag. When a tile inventory DU is present, the slice group shall be a tile identified by a tile id. Otherwise, when tile inventory DUs are not present, the interpretation of slice\_tag is application specific.

frame\_ctr\_lsb specifies the frame\_ctr\_lsb\_bits LSBs of the notional frame counter FrameCtr. Consecutive slices with different values of frame\_ctr\_lsb form parts of separate output point cloud frames. Consecutive slices with identical values of frame\_ctr\_lsb without an intervening frame boundary marker data unit form parts of the same coded point cloud frame.

slice\_entropy\_continuation equal to 1 specifies that the entropy parsing state restoration process (11.6.2.2 and 11.6.3.2) shall be applied at the start of the GDU and any ADUs in the slice. slice\_entropy\_continuation equal to 0 specifies that the parsing of the GDU and any ADUs in the slice is independent of any other slice in the frame when slice\_inter\_entropy\_continuation is 0. When slice\_entropy\_continuation is not present, it shall be inferred to be 0.

It is a requirement of bitstream conformance that slice\_entropy\_continuation shall be 0 when the GDU is the first GDU in a coded point cloud frame. A decoder shall ignore (remove from the bitstream and discard) all slices in a coded point cloud frame with slice\_entropy\_continuation equal to 1 that are not preceded by a slice in the same frame with slice\_entropy\_continuation equal to 0.

prev\_slice\_id shall be equal to the GDU slice\_id of the preceding slice in bitstream order. A decoder shall ignore (remove from the bitstream and discard) slices where prev\_slice\_id is both present and not equal to slice\_id of the preceding slice in the same frame.

It is recommended that slice\_entropy\_continuation is 0 if slice\_tag is not equal to the slice\_tag of the GDU identified by prev\_slice\_id. For example, if slice\_tag is used to select a subset of slices, then decoding might be prevented if there are dependencies upon slices that were not selected.

slice\_geom\_origin\_bits\_minus1 plus 1 specifies the length in bits of each slice\_geom\_origin\_xyz syntax element.

slice\_geom\_origin\_xyz[ 𝑘 ] and slice\_geom\_origin\_log2\_scale specify the 𝑘-th XYZ coordinate of the slice origin in the coding coordinate system. The slice origin in STV coordinates is specified by the expression SliceOrigin[ 𝑘 ]. When slice\_geom\_origin\_log2\_scale is not present, it shall be inferred to be gps\_geom\_origin\_log2\_scale.

SliceOrigin[k] := slice\_geom\_origin\_xyz[StvToXyz[k]] << Max(slice\_geom\_origin\_log2\_scale, *triSoupNodeSizeLog2*)

slice\_angular\_origin\_bits\_minus1 plus 1 specifies the length in bits of each slice\_angular\_origin\_xyz syntax element.

slice\_angular\_origin\_xyz[ 𝑘 ] specifies the 𝑘-th XYZ coordinate of the angular origin relative in the slice's coordinate system. When slice\_angular\_origin\_xyz[ 𝑘 ] is not present, it shall be inferred to be 0.

The slice-relative angular origin in STV coordinates is specified by the expression AngularOrigin[ 𝑘 ].

AngularOrigin[k] := slice\_angular\_origin\_present  
 ? slice\_angular\_origin\_xyz[StvToXyz[k]]  
 : gps\_angular\_origin\_xyz[StvToXyz[k]] − SliceOrigin[k]

slice\_geom\_qp\_offset specifies the slice geometry QP as an offset to the GPS geom\_qp. When slice\_geom\_qp\_offset is not present, it shall be inferred to be 0.

slice\_inter\_prediction equal to 1 specifies that inter prediction may be used to derive the positions in the GDU. slice\_inter\_prediction equal to 0 specifies that inter prediction is not used to derive the positions in the GDU. When slice\_inter\_prediction is not present, it shall be inferred to be 0.

slice\_biprediction equal to 1 specifies that bi-prediction may be used to derive the positions in the GDU. slice\_biprediction equal to 0 specifies that bi-prediction is not used to derive the positions in the GDU. When slice\_biprediction is not present, it shall be inferred to be 0.

gm\_matrix[ i ][ j ] and gm\_trans[ i ] specify the motion compensation parameters in the form of a matrix and an offset that is to be applied when global motion is enabled. When gm\_matrix[ i ][ j ] is not present, it is inferred to be 0. When gm\_trans[ i ] is not present, it is inferred to be 0. The global motion matrix is specified by the expression GMMatrix[ i ][ j ].

GMMatrix[i][j] := (i == j ? 65536 : 0) + gm\_matrix[i][j]

gm\_thres\_top and gm\_thres\_bot specify the two thresholds used to determine the points to which motion compensation is applied.

**slice\_inter\_frame\_ref\_gmc** specifies the inter prediction GMC reference frame to be applied to the GDU unit associated with the GDU header as specified by 9.3.3.11. When slice\_inter\_frame\_ref\_gmc is not present, it shall be inferred to be 0.

gm\_matrix2[ i ][ j ] and gm\_trans2[ i ] specify the motion compensation parameters in the form of a matrix and an offset that is to be applied to the second reference frame when global motion is enabled. When gm\_matrix2[ i ][ j ] is not present, it is inferred to be 0. When gm\_trans2[ i ] is not present, it is inferred to be 0. The global motion matrix is specified by the expression GMMatrix2[ i ][ j ].

GMMatrix2[i][j] := (i == j ? 65536 : 0) + gm\_matrix2[i][j]

gm\_thres\_top2 and gm\_thres\_bot2 specify the two thresholds used to determine the points of the second reference frame to which motion compensation is applied.

**slice\_inter\_frame\_ref\_gmc2** specifies the second inter prediction GMC reference frame to be applied to the GDU unit associated with the GDU header as specified by 9.3.3.11. When slice\_inter\_frame\_ref\_gmc2 is not present, it shall be inferred to be 0.

motion\_partition\_type equal to 0 specifies that road and object partitioning shall be used to determine the points for which global motion compensation is to be applied. motion\_partition\_type equal to 1 specifies that cuboid partitioning shall be used to determine the points for which global motion compensation is to be applied.

The variable MotionPartitionType is set equal to -1 when motion\_partition\_type is not present and set equal to motion\_partition\_type when present.

motion\_zero\_origin\_flag equal to 1 specifies that the origin position {0, 0, 0} shall be used in global motion compensation using cuboid partitioning. motion\_zero\_origin\_flag equal to 0 specifies that the origin position sps.seq\_origin\_xyz shall be used in global motion compensation using cuboid partitioning.

motion\_block\_size[k] specifies the partition block size that is to be used in global motion compensation using cuboid partitioning. When motion\_block\_size[k] is equal to 0, the block size in the k-th dimension is equal to the size the k-th dimension of the slice bounding box.

The number of motion partition blocks in the k-th axis is specified by the expression NumMotionBlocksPerAxis[k].

NumMotionBlocksPerAxis[k] := motion\_block\_size[k]   
 ? (Bbox.max[0] – Bbox.min[k] + motion\_block\_size[k] – 1)/ motion\_block\_size[k] : 1

[Ed. (YX): Definition missing for Bbox.max() and BBox.min().]

The number of motion partition blocks is specified by the expression NumMotionBlocks.

NumMotionBlocks := NumMotionBlocksPerAxis[0] \* NumMotionBlocksPerAxis[1] \* NumMotionBlocksPerAxis[2]

slice\_inter\_entropy\_continuation equal to 1 specifies that the entropy parsing state restoration process (11.6.2.2 and 11.6.3.2) shall be applied at the start of the GDU and any ADUs in the slice. slice\_inter\_entropy\_continuation equal to 0 specifies that the parsing of the GDU and any ADUs in the slice is independent of any other slice when slice\_entropy\_continuation is 0. When slice\_inter\_entropy\_continuation is not present, it shall be inferred to be 0.

It is a requirement of bitstream conformance that slice\_inter\_entropy\_continuation shall be 0 when the GDU is the not the first GDU in a coded point cloud frame.

prev\_inter\_entropy\_frame\_ctr\_lsb shall be equal to the frame\_ctr\_lsb of the preceding frame in bitstream order. A decoder shall ignore (remove from the bitstream and discard) slices where prev\_inter\_entropy\_frame\_ctr\_lsb is both present and not equal to frame\_ctr\_lsb of the preceding frame in bitstream order. The number of bits used to code prev\_inter\_entropy\_frame\_ctr\_lsb is frame\_ctr\_lsb\_bits

prev\_inter\_entropy\_slice\_id shall be equal to the GDU slice\_id of the preceding slice of the preceding frame in bitstream order. A decoder shall ignore (remove from the bitstream and discard) slices where prev\_inter\_entropy\_slice\_id is both present and not equal to slice\_id of the preceding slice in the preceding frame in bitstream order.

It is recommended that slice\_inter\_entropy\_continuation is 0 if slice\_tag is not equal to the slice\_tag of the GDU identified by prev\_inter\_entropy\_slice\_id of the preceding frame. For example, if slice\_tag is used to select a subset of slices, then decoding might be prevented if there are dependencies upon slices that were not selected.

num\_subsequent\_subgroups specifies the number of the subsequent dependent data units which reference the context state of the current data unit.

subgroup\_planar\_eligibility\_by\_density[ i ] equals to 1 indicates that planar eligibility is enabled for the (i + startDepth)-th depth of the current subgroup. subgroup\_planar\_eligibility\_by\_density[ i ] equals to 0 indicates that the planar eligibility is disabled for the (i + startDepth)-th depth of the current subgroup. When not present, subgroup\_planar\_eligibility\_by\_density[ i ] is inferred to 0.

#### Geometry data unit footer semantics

The start of the GDU footer shall be determined from the end of the GDU as specified by 11.2.4.

slice\_num\_points\_minus1 plus 1 specifies the number of points coded in the DU. It is a requirement of bitstream conformance that slice\_num\_points\_minus1 plus 1 shall be equal to the number of decodable points in the DU. Decoders shall not rely upon bitstream conformance to prevent overflow of implementation buffers.

### Attribute data unit

#### Attribute data unit semantics

An ADU codes attribute values for a single attribute in a slice. It comprises an ADU header and either attribute coefficients (attribute\_coeffs) when transform coding is enabled or directly coded attribute values (attribute\_raw).

#### Attribute data unit header semantics

adu\_attr\_parameter\_set\_id specifies the active APS by its aps\_attr\_parameter\_set\_id.

adu\_temporal\_id specifies the temporal ID of the frame associated with the attribute data unit.

adu\_sps\_attr\_idx identifies the coded attribute by its index into the active SPS attribute list.

At the start of every ADU, the variable AttrIdx is set to adu\_sps\_attr\_idx:

AttrIdx = adu\_sps\_attr\_idx

The attribute coded by the ADU shall have at most three components when attr\_coding\_type is not 3.

adu\_slice\_id specifies the value of the preceding GDU slice\_id.

**slice\_attr\_inter\_prediction** equal to 1 specifies that inter prediction may be used to derive the attributes in the ADU. slice\_attr\_inter\_prediction equal to 0 specifies that inter prediction is not used to derive the attributes in the ADU. When slice\_attr\_inter\_prediction is not present, it shall be inferred to be 0.

**slice\_attr\_inter\_prediction2** equal to 1 specifies that the inter prediction from the second reference frame may be used to derive the attributes in the ADU. slice\_attr\_inter\_prediction equal to 0 specifies that the inter prediction from the second reference frame is not used to derive the attributes in the ADU. When slice\_attr\_inter\_prediction2 is not present, it shall be inferred to be 0.

num\_inter\_filters specifies the number of RAHT temporal filters. When num\_inter\_filters is present, it is a requirement of bitstream conformance that shall be equal to Min(raht\_inter\_layer\_depth\_minus1+1, RahtRootLvl) – raht\_send\_inter\_filters. When num\_inter\_filters is not presented, it shall be infered to be 0.

raht\_inter\_filter\_qidx[] specifies the quantized residual of the RAHT temporal filters.

layer\_code\_depthspecifies the number of layers in the transform tree for which AC coefficients coding mode are signalled. When layer\_code\_depth is present, it is a requirement of bitstream conformance that layer\_code\_depth shall be equal to Min(raht\_inter\_layer\_depth\_minus1, RahtRootLvl - 1). When layer\_code\_depth is not present, it shall be inferred to be 0.

slice\_raht\_inter\_layer\_code\_mode[d]equal to 1 specifies that inter prediction is enabled for AC coefficients at the prediction depth *d* of the transform tree. slice\_raht\_inter\_layer\_code\_mode[*d*] equal to 0 specifies that inter prediction is disabled for AC coefficients at the prediction depth *d* of the transform tree.

slice\_raht\_intra\_layer\_code\_mode[d]equal to 1 specifies that intra prediction is enabled for AC coefficients at the prediction depth d of the transform tree. slice\_raht\_intra\_layer\_code\_mode[d] equal to 0 specifies that intra prediction is disabled for AC coefficients at the prediction depth d of the transform tree.

### Defaulted attribute data unit semantics

A defaulted attribute data unit specifies a single attribute value for all points in the slice.

defattr\_seq\_parameter\_set\_id specifies the active SPS by its sps\_seq\_parameter\_set\_id.

defattr\_reserved\_zero\_3bits shall be 0 in bitstreams conforming to this version of this document. Other values of defattr\_reserved\_zero\_3bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of defattr\_reserved\_zero\_3bits.

defattr\_sps\_attr\_idx identifies the coded attribute by its index into the active SPS attribute list.

At the start of every defaulted attribute data unit, the variable AttrIdx is set to defattr\_sps\_attr\_idx:

AttrIdx = defattr\_sps\_attr\_idx

defattr\_slice\_id specifies the value of the preceding GDU slice\_id.

defattr\_value[ 𝑐 ] specifies the value of the 𝑐-th attribute component for every point in the slice. The length in bits of defattr\_value[ 𝑐 ] is AttrBitDepth.

# Decoding process

## General decoding process

The reconstruction of a point cloud is specified such that all decoders that conform to a specified profile and level will produce numerically identical output point cloud frames for a bitstream conforming to that profile and level. Any decoding process that produces an identical output point cloud sequence to that produced by the process described herein conforms to the decoding process requirements of this document.

The frame decoding process (8.2) shall be repeatedly performed for each coded point cloud frame in the coded point cloud sequence.

## Frame decoding processes

### General

The result of this process is a reconstructed point cloud frame.

At the start of every coded point cloud frame, the output point cloud frame shall be initialized to the empty point cloud.

RecCloudPointCnt = 0

When inter prediction is enabled, PtnCurrFramePos is initialized to -1 for all entries. When the current frame is not an I-frame and the slices of the frame are coded using octree coding, the reference frame generation process in subclause 9.2.15 is invoked

The slice decoding process (8.3) shall be repeatedly performed for each slice in the coded point cloud frame.

### Frame counter

The variable FrameCtr represents the notional frame counter. For the first decoded frame, FrameCtr shall be set equal to frame\_ctr\_lsb. Otherwise, the variable FrameCtr shall be updated for each frame:

window = Exp2(frame\_ctr\_lsb\_bits) >> 1  
curLsb = FrameCtr % Exp2(frame\_ctr\_lsb\_bits)  
curMsb = FrameCtr >> frame\_ctr\_lsb\_bits  
if ((frame\_ctr\_lsb < curLsb) && (curLsb − frame\_ctr\_lsb) ≥ window)  
 curMsb++  
else if ((frame\_ctr\_lsb > curLsb) && (frame\_ctr\_lsb – curLsb) > window)  
 curMsb−−  
FrameCtr = (curMsb << frame\_ctr\_lsb\_bits) + frame\_ctr\_lsb

## Slice decoding processes

### General

A slice in a coded point cloud frame shall be decoded as follows:

1. Point positions are decoded from one GDU in the slice as specified by 8.3.3.
2. Default attribute values are set for each attribute as specified by 8.3.4.
3. Point attributes are decoded from each ADU in the slice as specified by 8.3.5.
4. The decoded point positions are offset and the output point count incremented as specified by 8.3.6.

Only one slice shall be decoded for every set of slices in a coded point cloud frame with the same value of slice\_id as specified in 6.4.6.

### State variables

Slice decoding is specified in terms of the following state variables:

* The variable PointCnt, a cumulative count of decoded points.
* The array PointAng of angular coordinates for decoded points; PointAng[ ptIdx ][ 𝑘 ] is the 𝑘-th angular coordinate of the point position PointPos[ ptIdx ].

### Geometry decoding process

The GDU shall be decoded and the reconstructed positions stored in the output point cloud.

The expression PointPos[ ptIdx ][ 𝑘 ] is an alias into the output point cloud for points in the slice.

PointPos[ptIdx][k] := RecCloudPos[RecCloudPointCnt + ptIdx][k]

The definition of PointPos implicitly concatenates the points of consecutive slices.

When geom\_angular\_enabled is 1, the geometry decoding process populates the array PointAng with points' angular coordinates.

At the start of every slice, PointCnt is initialized to 0. It is incremented for each point decoded by the geometry decoding process.

Point positions shall be decoded and reconstructed as specified by Clause 9.

### Default attribute values

Attribute values for every point in the slice shall be set to their respective default values. This process shall be equivalent to the following steps for each attribute, attrIdx = 0 .. num\_attributes − 1:

* All components of the attribute values shall be set to Exp2( attr\_bitdepth\_minus1[ attrIdx ] ).
* If the attribute property attr\_default\_value[ attrIdx ] is present, the attribute values shall be set to attr\_default\_value[ attrIdx ][ 𝑐 ], for each component 𝑐.
* If the slice contains a defaulted attribute data unit with defattr\_sps\_attr\_idx equal to attrIdx, the attribute values shall be set to defattr\_value[ 𝑐 ] of that DU, for each component 𝑐.

### Attribute decoding process

The ADU shall be decoded and the reconstructed attribute values stored in the corresponding output point cloud attribute.

The expression PointAttr[ ptIdx ][ 𝑐 ] is an alias into the output point cloud attribute array for the points in the slice.

PointAttr[ptIdx][c] := RecCloudAttr[RecCloudPointCnt + ptIdx][AttrIdx][c]

Point attributes shall be decoded and reconstructed as specified by Clause 10.

### At the end of a slice

The variable RecCloudPointCnt is incremented by the number of points decoded.

RecCloudPointCnt += PointCnt

The slice geometry shall be translated from the slice's coordinate system to the coding coordinate system by the addition of the slice origin, SliceOrigin.

The attribute decoding processes specified in Clause 10 are performed prior to the coordinate system conversion.

for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)  
 for (k = 0; k < 3; k++)  
 PointPos[ptIdx][k] += SliceOrigin[k]

### Update of inter prediction buffer

This subclause is invoked when inter prediction is enabled under predictive geometry coding and biprediction\_enabled is 0. The points in the reference frame are downsampled using the azimuth scale and added such that for each beamId and azimuth look value, at most max\_points\_per\_entry\_minus1 points are stored.

if (inter\_prediction\_enabled)  
 for(ptIdx = 0; ptIdx < PointCnt; ptIdx++)  
 for(j = 0; j <= max\_points\_per\_entry\_minus1; j++) {  
 if (PtnCurrFramePos[beamId][qAzim][j][0] == -1) {  
 if (max\_points\_per\_entry\_minus1 > 0 && j > 0) {  
 lAttr = PtnCurrFrameAttr[beamId][qAzim][j-1]  
 lPtn = PtnCurrFramePos[beamId][qAzim][j-1]  
 if (lAttr[0] == PtnCurrFrameAttr[beamId][qAzim][j][0])  
 if ((Abs(lPtn[0] - PtnCurrFramePnt[beamId][qAzim][j][0]) > dnRadiusRange) ||  
 (Abs(lPtn[1] - PtnCurrFramePnt[beamId][qAzim][j][1]) > dnAzimuthRange)) {  
 for(k = 0; k < 3; k ++)  
 PtnCurrFramePos[beamId][qAzim][j][k] = PointAng[ptIdx][k]  
 PtnCurrFrameAttr[beamId][qAzim][j][0] = PointAttr[ptIdx][0]  
 break  
 }  
 }  
 for(k = 0; k < 3; k ++)  
 PtnCurrFramePos[beamId][qAzim][j][k] = PointAng[ptIdx][k]  
 PtnCurrFrameAttr[beamId][qAzim][j][0] = PointAttr[ptIdx][0]  
 break  
 }  
 }  
where  
 beamId := PointAng[ptIdx][2]  
 qAzim := DivExp2Fz(PointAng[ptIdx][1], inter\_azim\_scale\_log2)  
 dnRadiusRange := down\_sampling\_range × (attr\_coord\_conv\_scale[0] >> 8)  
 dnAzimuthRange := down\_sampling\_range