# Parsing process

## General

Syntax elements are parsed according to the processes corresponding to the syntax element’s descriptor and name as specified in Tables 33 to 35.

Table 46 — Descriptor parsing processes

| Descriptor | Parsing process | Arguments | Channel read method (readBit) |
| --- | --- | --- | --- |
| u(𝑛) | 11.4.1 | maxBins = 𝑛 | DuNextBit (11.2.5) |
| u(v) | 11.4.1 | See Table 33 | DuNextBit (11.2.5) |
| ue(v) | 11.4.3 | 𝑘 = 0 | DuNextBit (11.2.5) |
| s(𝑛) | 11.4.2 | maxBins = 𝑛 | DuNextBit (11.2.5) |
| s(v) | 11.4.2 | See Table 33 | DuNextBit (11.2.5) |
| se(v) | 11.4.3, 11.4.6 | 𝑘 = 0 | DuNextBit (11.2.5) |
| oid(v) | 11.4.7 |  | DuNextBit (11.2.5) |
| ae(v) | See Table 34 | See Table 34 | AeReadBin (11.5.2) |
| de(v) | 9.2.9 |  | na |

Table 47 — Syntax element specific parsing processes (non-ae(v))

| Syntax element | Parsing process | Arguments |
| --- | --- | --- |
| attr\_coord\_conv\_scale[ ] | 11.4.1 (FL) | numBins = attr\_coord\_conv\_scale\_bits\_minus1[ 𝑘 ] + 1 |
| attr\_default\_value[ attrIdx ][ ] | 11.4.1 (FL) | numBins = attr\_bitdepth\_minus1[ attrIdx ] + 1 |
| attr\_offset[ ] | 11.4.2 (FL+S) | numBins = attr\_offset\_bits |
| attr\_qp\_region\_origin\_rpi[ ] | 11.4.1 (FL) | numBins = attr\_qp\_region\_bits\_minus1 + 1 |
| attr\_qp\_region\_origin\_xyz[ ] | 11.4.1 (FL) | numBins = attr\_qp\_region\_bits\_minus1 + 1 |
| attr\_qp\_region\_size\_minus1\_rpi[ ] | 11.4.1 (FL) | numBins = attr\_qp\_region\_bits\_minus1 + 1 |
| attr\_qp\_region\_size\_minus1\_xyz[ ] | 11.4.1 (FL) | nmBins = attr\_qp\_region\_bits\_minus1 + 1 |
| attr\_scale\_minus1[ ] | 11.4.1 (FL) | numBins = attr\_scale\_bits |
| defattr\_value[ ] | 11.4.1 (FL) | numBins = AttrBitDepth |
| fbdu\_frame\_ctr\_lsb | 11.4.1 (FL) | numBins = fbdu\_frame\_ctr\_lsb\_bits |
| frame\_ctr\_lsb | 11.4.1 (FL) | numBins = frame\_ctr\_lsb\_bits |
| fsap\_frame\_ctr\_lsb | 11.4.1 (FL) | numBins = fsap\_frame\_ctr\_lsb\_bits |
| gps\_angular\_origin\_xyz[ ] | 11.4.1 (FL) | numBins = gps\_angular\_origin\_bits\_minus1 + 1 |
| raw\_attr\_value[ ][ ] | 11.4.1 (FL) | numBins = RawAttrValueBits |
| seq\_bbox\_size\_minus1\_xyz[ ] | 11.4.1 (FL) | numBins = seq\_bbox\_size\_bits |
| seq\_coded\_scale\_mantissa | 11.4.1 (FL) | numBins = seq\_coded\_scale\_mantissa\_bits |
| seq\_origin\_xyz[ ] | 11.4.2 (FL+S) | numBins = seq\_origin\_bits |
| slice\_angular\_origin\_xyz[ ] | 11.4.2 (FL+S) | numBins = slice\_angular\_origin\_bits\_minus1 + 1 |
| slice\_geom\_origin\_xyz[ ] | 11.4.1 (FL) | numBins = slice\_geom\_origin\_bits\_minus1 + 1 |
| slice\_tag | 11.4.1 (FL) | numBins = slice\_tag\_bits |
| ti\_frame\_ctr\_lsb | 11.4.1 (FL) | numBins = ti\_frame\_ctr\_lsb\_bits |
| ti\_origin\_xyz[ ] | 11.4.2 (FL+S) | numBins = ti\_origin\_bits\_minus1 + 1 |
| tile\_id | 11.4.1 (FL+S) | numBins = tile\_id\_bits |
| tile\_origin\_xyz[ ][ ] | 11.4.2 (FL) | numBins = tile\_origin\_bits\_minus1 + 1 |
| tile\_size\_minus1\_xyz[ ][ ] | 11.4.1 (FL) | numBins = tile\_size\_bits\_minus1 + 1 |
| subgroup\_bbox\_origin\_xyz [ ] |  | numBins = subgroup\_bbox\_origin\_\_bits\_minus1 + 1 |
| subgroup\_bbox\_size\_xyz [ ] |  | numBins = subgroup\_bbox\_size\_\_bits\_minus1 + 1 |

Table 48 — Syntax element specific parsing processes (ae(v))

| Syntax element | Parsing process | Arguments |
| --- | --- | --- |
| beam\_idx\_resid\_abs[ ] | 11.4.4 (TU+EGk) | maxOffset = 3, 𝑘 = 1 |
| beam\_idx\_resid\_sign[ ] | 11.4.1 (FL) | numBins = 1 |
| coeff\_abs | 11.4.4 (TU+EGk) | maxOffset = 2, 𝑘 = 1 |
| coeff\_sign | 11.4.1 (FL) | numBins = 1 |
| direct\_dup\_point\_cnt | 11.4.4 (TU+EGk) | maxOffset = 2, 𝑘 = 0 |
| direct\_joint\_diff\_bit[ ] | 11.4.1 (FL) | numBins = 1 |
| direct\_joint\_prefix[ ] | 9.2.12.6 |  |
| direct\_point\_cnt\_eq2 | 11.4.1 (FL) | numBins = 1 |
| direct\_rem\_st\_ang[ ] | 11.4.1 (FL) | numBins = DnRemAngBitsST (9.2.13.8.3) |
| direct\_rem\_v\_ang[ ] | 11.4.1 (FL) | numBins = DnRemAngBitsV (9.2.13.8.3) |
| direct\_rem[ 𝑘 ] | 11.4.1 (FL) | numBins = DnRemBits[ 𝑘 ] (9.2.12.4.5) |
| direct\_v\_ang\_resid\_abs[ ] | 11.4.4 (TU+EGk) | maxOffset = 3, 𝑘 = 1 |
| direct\_v\_ang\_resid\_sign[ ] | 11.4.1 (FL) | numBins = 1 |
| occ\_direct\_node | 11.4.1 (FL) | numBins = 1 |
| occ\_dup\_point\_cnt[ ] | 11.4.4 (TU+EGk) | maxOffset = 1, 𝑘 = 0 |
| occ\_histogram\_hit | 11.4.1 (FL) | numBins = 1 |
| occ\_histogram\_index | 11.4.1 (FL) | numBins = 5 |
| planar\_copy\_mode | 11.4.1 (FL) | numBins = 1 |
| occ\_plane\_pos[ ] | 11.4.1 (FL) | numBins = 1 |
| occ\_recent\_hit | 11.4.1 (FL) | numBins = 1 |
| occ\_recent\_index | 11.4.1 (FL) | numBins = 4 |
| occ\_single\_child | 11.4.1 (FL) | numBins = 1 |
| occ\_single\_plane[ ] | 11.4.1 (FL) | numBins = 1 |
| occ\_subtree\_qp\_offset\_abs | 11.4.4 (TU+EGk) | maxOffset = 1, 𝑘 = 0 |
| occ\_subtree\_qp\_offset\_present | 11.4.1 (FL) | numBins = 1 |
| occ\_subtree\_qp\_offset\_sign | 11.4.1 (FL) | numBins = 1 |
| occ\_symbol\_escape | 11.4.1 (FL) | numBins = 8 |
| occtree\_end\_of\_entropy\_stream | 11.4.1 (FL) | numBins = 1 |
| occupancy\_bit[ ] | 11.4.1 (FL) | numBins = 1 |
| occupancy\_byte | 9.2.9 |  |
| occupancy\_idx[ ] | 11.4.1 (FL) | numBins = 1 |
| gm\_comp\_partition\_block[] | 11.4.1 (FL) | numBins = 1 |
| ptn\_child\_cnt\_xor1[ ] | 11.4.5 (TU) | maxVal = 3 |
| ptn\_dup\_point\_cnt | 11.4.4 (TU+EGk) | maxOffset = 1, 𝑘 = 0 |
| ptn\_phi\_mul\_abs\_minus2 | 11.4.1 (FL) | numBins = 3 |
| ptn\_phi\_mul\_abs\_minus9 | 11.4.2 (EGk) | 𝑘 = 0 |
| ptn\_phi\_mul\_abs\_prefix | 11.4.5 (TU) | maxVal = 2 |
| ptn\_phi\_mul\_sign | 11.4.1 (FL) | numBins = 1 |
| ptn\_radius\_resid\_abs | 11.4.1 (TU+EGk) | maxOffset = 3, 𝑘 = 2 |
| ptn\_radius\_resid\_sign | 11.4.1 (FL) | numBins = 1 |
| ptn\_phi\_resid\_abs\_gt0 | 11.4.1 (FL) | numBins = 1 |
| ptn\_phi\_resid\_sign | 11.4.1 (FL) | numBins = 1 |
| ptn\_phi\_resid\_abs\_gt1 | 11.4.1 (FL) | numBins = 1 |
| ptn\_phi\_resid\_abs\_rem | 11.4.2 (EGk) | 𝑘 = 1 |
| ptn\_inter\_flag | 11.4.1 (FL) | numBins = 1 |
| ptn\_pred\_direction | 11.4.1 (FL) | numBins = 1 |
| ptn\_inter\_pred\_mode | 11.4.1 (FL) | numBins = global\_motion\_enabled ? 2: 1 |
| ptn\_pred\_mode[ ] | 11.4.1 (FL) | numBins = 2 |
| ptn\_pred\_idx[ ] | 11.4.1 (TU) | maxVal = ptree\_ang\_max\_pred\_index |
| ptn\_qp\_offset\_abs | 11.4.4 (TU+EGk) | maxOffset = 1, 𝑘 = 0 |
| ptn\_qp\_offset\_sign | 11.4.1 (FL) | numBins = 1 |
| ptn\_resid\_abs\_gt0[ ] | 11.4.1 (FL) | numBins = 1 |
| ptn\_resid\_abs\_log2[ 𝑘 ] | 11.4.1 (FL) | numBins = ptn\_resid\_abs\_log2\_bits[ 𝑘 ] |
| ptn\_resid\_abs\_rem[ 𝑘 ] | 11.4.1 (FL) | numBins = ptn\_resid\_abs\_log2[ 𝑘 ] − 1 |
| ptn\_resid\_sign[ ] | 11.4.1 (FL) | numBins = 1 |
| ptn\_sec\_resid\_abs[ ] | 11.4.4 (TU+EGk) | maxOffset = 2, 𝑘 = 0 |
| ptn\_sec\_resid\_sign[ ] | 11.4.1 (FL) | numBins = 1 |
| ptree\_end\_of\_slice | 11.4.1 (FL) | numBins = 1 |
| zero\_run\_length\_prefix | 11.4.5 (TU) | maxVal = 3 |
| zero\_run\_length\_minus3\_div2 | 11.4.5 (TU) | maxVal = 4 |
| zero\_run\_length\_minus3\_mod2 | 11.4.1 (FL) | numBins = 1 |
| zero\_run\_length\_minus11 | 11.4.2 (EGk) | 𝑘 = 2 |

## Data unit buffer

### General

The parsing of syntax elements is specified as operations on a DU buffer. The DU buffer represents the coded DU as a sequence of unencapsulated bytes as provided by an encapsulation format such as that specified by Annex B or by another application-specific means.

### State

The DU buffer is specified in terms of the following state variables:

* The array DataUnitBytes, representing the DU buffer; DataUnitBytes[ 𝑖 ] is the 𝑖-th byte of the data unit.
* The variable DataUnitLength, equal to the length of the DU in bytes.
* The variable DataUnitReadIdx, equal to the byte index and bit position of the next bit to be read from the DU buffer.

### Initialization at the start of parsing a data unit

At the start of every DU, parsing shall commence at the first bit of the DU buffer.

DataUnitReadIdx = 0

### Initialization at the start of parsing a geometry data unit footer

The parsing of a geometry\_data\_unit\_footer syntax structure shall commence at an offset from the end of the DU buffer. The length of the GDU footer is specified by the expression DuFooterLen. The expression DuIsGdu is equal to 1 when the DU is a GDU.

GduFooterLen := 3 × (1 + occtree\_point\_cnt\_list\_present × occtreeMaxDepthMinus1)  
  
DuFooterLen := DuIsGdu ? GduFooterLen : 0  
DataUnitReadIdx = 8 × (DataUnitLength − DuFooterLen)

### Definition of DuNextBit

This subclause specifies the reading of a single bit from the DU buffer by the expression DuNextBit. Each evaluation of DuNextBit returns the next unread bit from the buffer.

duStreamByte[bitIdx] := DataUnitBytes[bitIdx >> 8]  
duStreamBit[bitIdx] := Bit(duStreamByte[bitIdx], 7 − (bitIdx & 7))  
  
DuNextBit := duStreamBit[DataUnitReadIdx++]

## Chunked bytestream parsing

### General

This subclause applies to GDUs and ADUs that contain syntax elements with ae(v) descriptors when bypass\_stream\_enabled is 1.

An ADU with attr\_coding\_type equal to 3 does not contain any ae(v) syntax elements.

The CBS representation conveys two multiplexed data streams as a sequence of chunks: a stream of arithmetic-coded bytes (AeBits) and a stream of bits that bypass the arithmetic decoding engine (BpBits). Every chunk is a block of 256 bytes, with the exception of the final chunk which may be shorter.

An example CBS is illustrated in Figure 24. It starts with two ChunkLen length chunks. From the CBS, two subtreams, AeBits and BpBits are extracted.

手机屏幕截图

描述已自动生成

Figure 24 — Multiplexed data streams in a chunked bytestream.

When occtree\_stream\_cnt\_minus1 is greater than 0, each of an occupancy tree's entropy streams shall be conveyed in a separate CBS (11.3.12). Consecutive CBSs shall be spliced together (11.3.11) such that the last chunk of each CBS is merged with the first chunk of the next. Splicing pads the last chunk of a CBS to 256 bytes.

### Chunk syntax

|  |  |
| --- | --- |
| ae\_chunk( ) { | **Descriptor** |
| chunk\_ae\_len | u(8) |
| for( 𝑖 = 0; 𝑖 < chunk\_ae\_len; 𝑖++ ) |  |
| chunk\_ae\_byte[ 𝑖 ] | u(8) |
| for( 𝑖 = 0; 𝑖 < ChunkPadLen; 𝑖++ ) |  |
| chunk\_splice\_byte[ 𝑖 ] |  |
| if( chunk\_ae\_len < ChunkLen − 1 − ChunkPadLen ) { |  |
| chunk\_bypass\_5bits | u(5) |
| chunk\_bypass\_flushed\_bits | u(3) |
| } |  |
| for( 𝑖 = 0; 𝑖 < ChunkLen − 2 − chunk\_ae\_len − ChunkPadLen; 𝑖++) |  |
| chunk\_bypass\_byte[ 𝑖 ] | u(8) |
| } |  |

### Chunk semantics

chunk\_ae\_len specifies the number of chunk\_ae\_byte syntax elements present in the chunk. It is a requirement of bitstream conformance that chunk\_ae\_len shall be less than ChunkLen.

chunk\_ae\_byte[ 𝑖 ] specifies the 𝑖-th arithmetic-coded byte conveyed by the chunk.

chunk\_splice\_byte[ 𝑖 ] specifies a padding byte used to pad the last chunk of a CBS. The padding bytes shall consist of bytes moved from the start of the next CBS.

chunk\_bypass\_byte[ 𝑖 ], chunk\_bypass\_5bits and chunk\_bypass\_flushed\_bits together specify the bypass-coded bits conveyed by the chunk. Within a chunk, the bits are in reverse order, as specified by the unpacking process (11.3.8).

### State

The CBSs are specified in terms of the following state variables:

* The 256 byte array ChunkBuf, a buffer used to merge and unpack chunks.
* The array AeBits of unpacked arithmetic-coded bits; each element is a single bit. The variable AeBitsLen is the length of the array.
* The array BpBits of unpacked bypass-coded bits; each element is a single bit. The variable BpBitsLen is the length of the array.
* The variables AeBitsReadIdx and BpBitsReadIdx, indexes of the next element to be read from the AeBits and BpBits arrays, respectively.

### Span of chunked bytestream data within a data unit

Each applicable DU comprises a header, the CBS data and a footer (if present).

The CBS data starts at the byte aligned position at or prior to the first ae(v) coded syntax element.

When a DU footer is present, the CBS data ends immediately prior to the first non-ae(v) coded syntax element of the fixed-length footer. Otherwise, the end shall coincide with the end of the DU buffer.

The number of bytes remaining in the CBS data is specified by the expression ChunkDuRem.

ChunkDuRem := DataUnitLength − (DataUnitReadIdx >> 3) − DuFooterLen

### The chunk buffer

Immediately prior to parsing a chunk, the chunk buffer is populated with the bytes of the next chunk from the CBS data span.

Unless 11.3.11 applies, the chunk buffer is populated by the next ChunkLen unparsed bytes from the DU buffer. Every chunk shall be either 256 bytes in length, or as long as the remaining bytes in the CBS data span, whichever is shorter.

ChunkLen = Min(256, ChunkDuRem)  
for (i = 0; i < ChunkLen; i++) {  
 ChunkBuf[i] = DataUnitBytes[DataUnitReadIdx >> 3]  
 DataUnitReadIdx += 8  
}

### State update at the start of every CBS

No unpacked data shall be preserved across CBSs. Immediately prior to unpacking the first chunk of a CBS, the unpacked arithmetic- and bypass-coded bit buffers and their respective read positions shall be cleared.

AeBitsLen = BpBitsLen = 0  
AeBitsReadIdx = BpBitsReadIdx = 0

### Unpacking a single chunk

Unpacking a single chunk comprises parsing the contents of the chunk buffer and appending the per-stream data to the unpacked streams. Parsing shall be performed according to the syntax and semantics of the ae\_chunk syntax structure with ChunkPadLen assumed to be 0.

Any arithmetic-coded data are appended to the unpacked array AeBits.

for (i = 0; i < chunk\_ae\_len; i++)  
 for (b = 7; b ≥ 0; b−−)  
 AeBits[AeBitsLen++] = Bit(chunk\_ae\_byte[i], b)

Any bypass data are appended to the unpacked array BpBits. Bypass data are appended in reverse order of the chunk data. The last chunk\_bypass\_flushed\_bits are excluded.

numChunkBypassBytes := Max(0, ChunkLen − 2 − chunk\_ae\_len)

for (j = numChunkBypassBytes − 1; j ≥ 0; j−−)  
 for (b = 7; b ≥ 0; b−−)  
 BpBits[BpBitsLen++] = Bit(chunk\_bypass\_byte[j], b)

for (b = 4; b ≥ 0; b−−)  
 BpBits[BpBitsLen++] = Bit(chunk\_bypass\_5bits, b)  
BpBitsLen −= chunk\_bypass\_flushed\_bits

### Definition of ChunkNextAeBit

This subclause specifies the reading of a single bit from the arithmetic-coded bitstream by the expression ChunkNextAeBit. Each evaluation of the expression returns the next unread bit from the stream.

Prior to reading a bit from the stream, if there are no unread bits left in the stream buffer, subsequent chunks shall be unpacked as specified by 11.3.8 until an unread bit is available.

ChunkNextAeBit :=  
 while (AeBitsReadIdx ≥ AeBitsLen) {  
 … /\* unpack chunk as specified by 11.3.6 \*/  
 }  
 ChunkNextAeBit = AeBits[AeBitsReadIdx++]

### Definition of ChunkNextBpBit

This subclause specifies the reading of a single bit from the bypass-coded bitstream by the expression ChunkNextBpBit. Each evaluation of the expression returns the next unread bit from the stream.

Prior to reading a bit from the stream, if there are no unread bits left in the stream buffer, subsequent chunks shall be unpacked as specified by 11.3.8 until an unread bit is available.

ChunkNextBpBit :=  
 while (BpBitsReadIdx ≥ BpBitsLen) {  
 … /\* unpack chunk as specified by 11.3.6 \*/  
 }  
 ChunkNextBpBit = BpBits[BpBitsReadIdx++]

### Boundary between spliced chunked bytestreams

This subclause applies when occtree\_stream\_cnt\_minus1 is greater than 0.

Multiple CBSs shall be spliced to form a contiguous span of CBS data. Splicing pads the last chunk of each CBS to maintain a fixed-length chunk size within that CBS. To pad a the padding data shall consist of bytes moved from the start of the next CBS, . If the length of is less than 256 bytes, then it shall first be spliced with , if existent, prior to the splicing of with .

* 1. The definition of splicing is recursive. For example, if the splicing of two CBSs A and B is denoted by A :: B, then splicing four CBSs, A to D, is performed as A :: ( B :: ( C :: D ) ).
  2. The padding process permits the start of the bypass data in the last chunk of any CBS to be located after the parsing of that CBS has commenced.

图示

描述已自动生成

Figure 25 — Extraction of the first chunk from a spliced CBS.

At the boundary between two CBSs, the padding data from the preceding CBS, shall form the initial part of the first chunk of the next CBS, , as illustrated by chunk\_splice\_byte[  ] in Figure 25. The rest of the first chunk shall follow the last chunk of the preceding CBS.

The length of the padding data shall be derived from the unconsumed length of the bypass-coded bitstream specified by ChunkPadLen. The length of padding data includes the bits used to code chunk\_bypass\_flushed\_bits and the number of bits discarded.

ChunkPadLen = (BpBitsLen − BpBitsReadIdx + chunk\_bypass\_flushed\_bits + 3) / 8

* 1. The initial parsing of the chunk (11.3.8) assumes that there are no padding bytes present. For the last chunk in a CBS, this assumption might be wrong and the parsed values of chunk\_bypass\_5bits and chunk\_bypass\_flushed\_bits are meaningless.

To recover the first chunk of the next CBS, the last chunk is re-parsed according to the syntax and semantics of the ae\_chunk syntax structure with the determined value of ChunkPadLen. Any padding data is moved to the start of the chunk buffer, and the remainder of the chunk buffer populated by the next unparsed bytes from the CBS data span.

for (i = 0; i < ChunkPadLen; i++)  
 ChunkBuf[i] = chunk\_splice\_byte[i]

ChunkPartBLen := Min(256 − ChunkPadLen, ChunkDuRem)

ChunkLen = ChunkPadLen + ChunkPartBLen  
for (i = ChunkPadLen; i < ChunkLen; i++) {  
 ChunkBuf[i] = DataUnitBytes[DataUnitReadIdx >> 3]  
 DataUnitReadidx += 8  
}

After populating the chunk buffer, the parsing state shall be updated for the start of the next CBS (11.3.7) and the first chunk shall be unpacked (11.3.8).

### Location of chunked bytestream boundaries

This subclause applies when occtree\_stream\_cnt\_minus1 is greater than 0.

An additional CBS shall commence at the start of each occupancy\_tree\_level( dpth ) syntax structure where dpth is greater than OcctreeEntropyStreamDepth.

## General inverse binarization processes

### Parsing unsigned fixed-length codes (FL)

Parsing is parameterized by:

* numBits, the number of bits that represent the syntax element;
* readBit, the channel read method expression.

The result is the unsigned syntax element value PartValue, parsed and constructed as:

PartVal = 0  
for (BinIdx = 0; BinIdx < numBits; BinIdx++)  
 PartVal = (PartVal << 1) + readBit

### Parsing signed fixed-length codes (FL+S)

Parsing is parameterized by:

* numBits, the number of bits that represent the absolute syntax element value;
* readBit, the channel read method expression.

The unsigned syntax element magnitude is parsed:

PartVal = 0  
for (BinIdx = 0; BinIdx < numBits; BinIdx++)  
 PartVal = (PartVal << 1) + readBit

The result is the signed syntax element value val, parsed and constructed as:

sign = readBit  
val = sign ? −PartVal : PartVal

### Parsing 𝑘-th order exp-Golomb codes (EGk)

Parsing is parameterized by:

* 𝑘, the order of the exp-Golomb code;
* readBit, the channel read method expression.

First, a unary encoded prefix is parsed as:

prefix = 0  
for (BinIdxPfx = 0; readBit ≠ 0; BinIdxPfx++)  
 prefix++

Then, a suffix comprising 𝑘 + prefix bins is parsed:

suffix = 0  
for (BinIdxSfx = 0; BinIdxSfx < k + prefix; BinIdxSfx++)  
 suffix = (suffix << 1) + readBit

The result is the unsigned syntax element value val, constructed as:

val = Exp2(prefix + k) + suffix − Exp2(k)

### Parsing concatenated truncated unary and 𝑘-th order exp-Golomb codes (TU+EGk)

Parsing is parameterized by:

* maxOffset, the limit for the truncated unary offset encoding;
* 𝑘, the order of the exp-Golomb code;
* readBit, the channel read method expression.

First, a truncated unary encoded offset is parsed:

offset = 0  
for (BinIdxTu = 0; offset < maxOffset && readBit == 1; BinIdxTu++)  
 offset++

Second, if the value of offset is equal to maxOffset, a unary encoded prefix is parsed:

prefix = 0  
if (offset == maxOffset)  
 for (BinIdxPfx = 0; readBit ≠ 0; BinIdxPfx++)  
 prefix++

Then, if the value of offset is equal to maxOffset, a suffix comprising 𝑘 + prefix bins is parsed:

suffix = 0  
if (offset == maxOffset)  
 for (BinIdxSfx = 0; BinIdxSfx < k + prefix; BinIdxSfx++)  
 suffix = (suffix << 1) + readBit

The result is the unsigned syntax element value val, constructed as:

val = offset + Exp2(prefix + k) + suffix − Exp2(k)

### Parsing truncated unary codes (TU)

Parsing is parameterized by:

* maxVal, the limit for the encoding;
* readBit, the channel read method expression.

The result is the unsigned syntax element value PartVal, parsed and constructed as:

PartVal = 0  
for (BinIdxTu = 0; PartVal < maxVal && readBit == 1; BinIdxTu++)  
 PartVal++

### Mapping process for signed codes

The signed value of a syntax element parsed according to the descriptor se(v) shall be converted from its unsigned, parsed value. If the parsed value val is:

* even, the signed syntax element value is − ( val >> 1 );
* odd, the signed syntax element value is val + 1 >> 1.

Examples of the conversion are shown in Table 35.

Table 49 — Conversion of unsigned values for signed syntax elements

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Unsigned value | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Signed value | 0 | 1 | −1 | 2 | −2 | 3 | −3 |

### Parsing ASN.1 object identifiers

#### Object identifier syntax

|  |  |
| --- | --- |
| oid( ) { | Descriptor |
| oid\_forbidden\_zero\_bit | u(1) |
| oid\_length | u(7) |
| for( 𝑖 = 0; 𝑖 < oid\_length; 𝑖++ ) |  |
| oid\_contents\_octets[ 𝑖 ] | u(8) |
| } |  |

#### Object identifier semantics

The coded representation of an ASN.1 object identifier shall follow the ASN.1 distinguished encoding rules specified in Rec. ITU‑T X.690﻿ |‌ ISO/IEC 8825‑1.

oid\_forbidden\_zero\_bit shall be 0.

oid\_length specifies the number of octets present in oid\_contents\_octets.

oid\_contents\_octets[ 𝑖 ] is the 𝑖-th contents octet of an object identifier value encoding as specified in Rec. ITU‑T X.690﻿ |‌ ISO/IEC 8825‑1.

## CABAC parsing processes

### Initialization

The arithmetic decoding engine and CPMs shall be initialized according to 11.5.4.3 and 11.5.3.2 at the start of the following syntax structures:

* occupancy\_tree (7.3.3.4);
* occupancy\_tree\_level( dpth ) (7.3.3.5) when dpth is greater than OcctreeEntropyStreamDepth;
* predictive\_tree (7.3.3.8);
* attribute\_coeffs (7.3.4.3).

### Definition of AeReadBin

This subclause specifies the reading of a single arithmetic-coded bin as the expression AeReadBin. Each evaluation reads a single bin, parameterized by the name of the coded syntax element.

A CPM identified by the expression Ctx shall be selected according to 11.5.3.4.

If the value of Ctx is neither equal to 'bypass' nor 'terminate':

* The value of the decoded bin shall be determined in accordance with 11.5.4.5 for a single arithmetic-coded bin with Ctx as the argument prob0.
* The selected CPM shall then be updated in accordance with 11.5.3.3 using the decoded bin value as the argument binVal.

If the value of Ctx is 'bypass', the value of the decoded bin shall be determined:

* When bypass\_stream\_enabled is 0, in accordance with 11.5.4.6 for an arithmetic-coded bypass bin.
* When bypass\_stream\_enabled is 1, by evaluating the expression ChunkNextBpBit (11.3.10).

If the value of Ctx is 'terminate':

* The arithmetic decoder shall be flushed in accordance with 11.5.4.8.

### Contextual probability models

#### General

A CPM is a 16-bit unsigned integer value that models the probability of a zero bin.

The values 0, and represent the probability of a zero bin as impossible, equiprobable and certain respectively. The values 0 and can never be attained due to the operation of the context update process.

The array Contexts, with elements Contexts[ ctxTbl ][ ctxIdx ], represents individual adaptive CPMs used by the CABAC parsing process.

#### Initialization

When slice\_entropy\_continuation is 1 or slice\_inter\_entropy\_continuation is 1, initialization shall be performed by the parsing state restoration process (11.6).

Otherwise (slice\_entropy\_continuation is 0 and slice\_inter\_entropy\_continuation is 0), all CPMs shall be initialized to .

#### Update after each coded bin

After each bin coded using an adaptive CPM, the modelled probability shall be updated.

The parameter binVal is the value of the coded bin and the expression Ctx identifies the CPM used to arithmetically code it (11.5.3.4).

The update shall increase or decrease the modelled probability of a zero-valued bin according to the known value of the coded bin, the upper eight bits of the modelled probability and the channel model specified by Table 36:

if (binVal)  
 Ctx −= CtxUpdateDelta[Ctx >> 8]  
else  
 Ctx += CtxUpdateDelta[255 − (Ctx >> 8)]

Table 50 — Values of CtxUpdateDelta[ 𝑖 + 𝑗 ]

| 𝑗 | 𝑖 | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **0** | 0 | 2 | 5 | 8 | 11 | 15 | 20 | 24 | 29 | 35 | 41 | 47 |
| **12** | 53 | 60 | 67 | 74 | 82 | 89 | 97 | 106 | 114 | 123 | 132 | 141 |
| **24** | 150 | 160 | 170 | 180 | 190 | 201 | 211 | 222 | 233 | 244 | 256 | 267 |
| **36** | 279 | 291 | 303 | 315 | 327 | 340 | 353 | 366 | 379 | 392 | 405 | 419 |
| **48** | 433 | 447 | 461 | 475 | 489 | 504 | 518 | 533 | 548 | 563 | 578 | 593 |
| **60** | 609 | 624 | 640 | 656 | 672 | 688 | 705 | 721 | 738 | 754 | 771 | 788 |
| **72** | 805 | 822 | 840 | 857 | 875 | 892 | 910 | 928 | 946 | 964 | 983 | 1 001 |
| **84** | 1 020 | 1 038 | 1 057 | 1 076 | 1 095 | 1 114 | 1 133 | 1 153 | 1 172 | 1 192 | 1 211 | 1 231 |
| **96** | 1 251 | 1 271 | 1 291 | 1 311 | 1 332 | 1 352 | 1 373 | 1 393 | 1 414 | 1 435 | 1 456 | 1 477 |
| **108** | 1 498 | 1 520 | 1 541 | 1 562 | 1 584 | 1 606 | 1 628 | 1 649 | 1 671 | 1 694 | 1 716 | 1 738 |
| **120** | 1 760 | 1 783 | 1 806 | 1 828 | 1 851 | 1 874 | 1 897 | 1 920 | 1 935 | 1 942 | 1 949 | 1 955 |
| **132** | 1 961 | 1 968 | 1 974 | 1 980 | 1 985 | 1 991 | 1 996 | 2 001 | 2 006 | 2 011 | 2 016 | 2 021 |
| **144** | 2 025 | 2 029 | 2 033 | 2 037 | 2 040 | 2 044 | 2 047 | 2 050 | 2 053 | 2 056 | 2 058 | 2 061 |
| **156** | 2 063 | 2 065 | 2 066 | 2 068 | 2 069 | 2 070 | 2 071 | 2 072 | 2 072 | 2 072 | 2 072 | 2 072 |
| **168** | 2 072 | 2 071 | 2 070 | 2 069 | 2 068 | 2 066 | 2 065 | 2 063 | 2 060 | 2 058 | 2 055 | 2 052 |
| **180** | 2 049 | 2 045 | 2 042 | 2 038 | 2 033 | 2 029 | 2 024 | 2 019 | 2 013 | 2 008 | 2 002 | 1 996 |
| **192** | 1 989 | 1 982 | 1 975 | 1 968 | 1 960 | 1 952 | 1 943 | 1 934 | 1 925 | 1 916 | 1 906 | 1 896 |
| **204** | 1 885 | 1 874 | 1 863 | 1 851 | 1 839 | 1 827 | 1 814 | 1 800 | 1 786 | 1 772 | 1 757 | 1 742 |
| **216** | 1 727 | 1 710 | 1 694 | 1 676 | 1 659 | 1 640 | 1 622 | 1 602 | 1 582 | 1 561 | 1 540 | 1 518 |
| **228** | 1 495 | 1 471 | 1 447 | 1 422 | 1 396 | 1 369 | 1 341 | 1 312 | 1 282 | 1 251 | 1 219 | 1 186 |
| **240** | 1 151 | 1 114 | 1 077 | 1 037 | 995 | 952 | 906 | 857 | 805 | 750 | 690 | 625 |
| **252** | 553 | 471 | 376 | 255 |  | | | | | | | |

#### Selection

[Ed. (JT): Variable nodeIdx is unavailable in the context derivation below; find a way to refer it cleanly.]

A CPM shall be selected for each bin of the coded syntax element as specified by the expression Ctx. The values CtxTbl and CtxIdx shall be determined according to the entries for the syntax element in Table 37 (GDU) and Table 38 (ADU). Entries qualified by Offset, Prefix or Suffix individually apply when selecting a CPM for a bin of that part of the binarized syntax element.

Ctx := CtxIdx ≠ 'bypass' && CtxIdx ≠ 'terminate' ? Contexts[CtxTbl][CtxIdx] : CtxIdx

Table 51 — Values of CtxTbl and CtxIdx for binarized ae(v) coded GDU syntax elements

| Syntax element | CtxTbl | CtxIdx | | Count |
| --- | --- | --- | --- | --- |
| beam\_idx\_resid\_abs[ ] | 1 | **Offset** | 3 × (BeamPrevIdxResid[   DnBeamIdxEst] != 0)  + BinIdxTu | 6 |
| **Prefix** | 6 | 1 |
| **Suffix** | bypass | 0 |
| beam\_idx\_resid\_sign[ ] | 2 | 2 × (BeamPrevIdxResid[DnBeamIdxEst] < 0)  + (BeamPrevIdxResid[DnBeamIdxEst] > 0) | | 3 |
| direct\_dup\_point\_cnt | 3 | **Offset** | BinIdxTu | 2 |
| **Prefix** | 2 | 1 |
| **Suffix** | bypass | 0 |
| direct\_joint\_diff\_bit[ ] | na | bypass | | 0 |
| direct\_joint\_prefix[ 𝑘 ] | 4 | BinIdx & 1  ? 'bypass'  : 5 × 𝑘 + Min( 4, BinIdx / 2 ) | | 15 |
| direct\_point\_cnt\_eq2 | 5 | 0 | | 1 |
| direct\_rem\_st\_ang[ ] | 6 | CtxIdxAngPhi (9.2.13.8.4) | | 24 |
| direct\_rem\_v\_ang[ ] | 7 | CtxIdxAngTheta (9.2.13.8.6) | | 4 |
| direct\_rem[ ] | na | bypass | | 0 |
| direct\_v\_ang\_resid\_abs[ ] | 41 | Offset | BinIdxTu | 3 |
| **Prefix** | 3 | 1 |
| **Suffix** | bypass | 0 |
| direct\_v\_ang\_resid\_sign[ ] | 42 | 0 | | 1 |
| occ\_direct\_node | 8 | 0 | | 1 |
| occ\_dup\_point\_cnt | 3 | **Offset** | 0 | 1 |
| **Prefix** | 2 | 1 |
| **Suffix** | bypass | 0 |
| occ\_histogram\_hit | 9 | NeighPatR | | 9 |
| occ\_histogram\_index | 10 | CtxIdxDictHg (9.2.9.11) | | 45 |
| planar\_copy\_mode | TBA | CtxIdxPlanarCopyMode (9.2.11.12) | | 128 |
| occ\_plane\_pos[ ],   when ¬AngularEligible | 11 | CtxIdxPlanePos (9.2.11.7) | | 51 |
| occ\_plane\_pos[ 𝑘 ],  when AngularEligble && 𝑘 < 2 | 12 | CtxIdxAngPhi (9.2.13.7.7) | | 8 |
| occ\_plane\_pos[ 𝑘 ],  when AngularEligble && 𝑘 == 2 | 13 | CtxIdxAngTheta (9.2.13.7.8) | | 4 |
| occ\_recent\_hit | 14 | NeighPatR | | 9 |
| occ\_recent\_index | na | bypass | | 0 |
| occ\_single\_child | 15 | 0 | | 1 |
| occ\_single\_plane[ 𝑘 ] | 16 | 𝑘 + PlaneRef[𝑘] ? (3 \* PlanePosRef[𝑘] + 1 : 0) | | 9 |
| occ\_subtree\_qp\_offset\_abs | 17 | **Offset** | 0 | 1 |
| **Prefix** | 1 | 1 |
| **Suffix** | bypass | 0 |
| occ\_subtree\_qp\_offset\_present | na | bypass | | 0 |
| occ\_subtree\_qp\_offset\_sign | 18 | 0 | | 1 |
| occ\_symbol\_escape | 19 | NeighPatR | | 9 |
| occtree\_end\_of\_entropy\_stream | na | terminate | | 0 |
| occupancy\_bit[ ] | 20 | CtxIdxOccBit (9.2.10.6) | | 32 |
| occupancy\_idx[ ] | na | bypass | | 0 |
| gm\_comp\_partition\_block[] | 48 | 0 | | 1 |
| ptn\_child\_cnt\_xor1[ ] | 21 | BinIdxTu | | 3 |
| ptn\_dup\_point\_cnt | 22 | **Offset** | 0 | 1 |
| **Prefix** | 1 | 1 |
| **Suffix** | bypass | 0 |
| ptn\_phi\_mul\_abs\_minus2 | 23 | 16 × ptn\_inter\_flag[ nodeIdx ]   + 8 x (ptn\_inter\_flag[ nodeIdx ]        ? ptn\_inter\_pred\_mode[ nodeIdx ] > 1       : ptn\_pred\_idx[ nodeIdx ] == 0)   + Exp2( BinIdx ) + PartVal − 1 | | 32 |
| ptn\_phi\_mul\_abs\_minus9 | 24 | **Prefix** | 2 ×ptn\_inter\_flag[ nodeIdx ]   + (ptn\_inter\_flag[ nodeIdx ]   ? ptn\_inter\_pred\_mode[ nodeIdx ] > 1  : ptn\_pred\_idx[ nodeIdx ] == 0) | 4 |
| **Suffix** | bypass | 0 |
| ptn\_phi\_mul\_abs\_prefix | 25 | 4 × ptn\_inter\_flag[ nodeIdx ]  + 2 x (ptn\_inter\_flag[ nodeIdx ]        ? ptn\_inter\_pred\_mode[ nodeIdx ] > 1       : ptn\_pred\_idx[ nodeIdx ] == 0)  + BinIdxTu | | 8 |
| ptn\_phi\_mul\_sign | 26 | 2 ×ptn\_inter\_flag[ nodeIdx ]  + (ptn\_inter\_flag[ nodeIdx ]     ? ptn\_inter\_pred\_mode[ nodeIdx ] > 1    : ptn\_pred\_idx[ nodeIdx ] == 0) | | 4 |
| ptn\_pred\_mode[ ] | 27 | Exp2( BinIdx ) + PartVal − 1 | | 3 |
| ptn\_qp\_offset\_abs | 28 | **Offset** | 0 | 1 |
| **Prefix** | 1 | 1 |
| **Suffix** | bypass | 0 |
| ptn\_qp\_offset\_sign | 29 | 0 | | 1 |
| ptn\_resid\_abs\_gt0[ 𝑘 ] | 30 | 𝑘 +3 × ptn\_inter\_flag[ nodeIdx ] | | 3 |
| ptn\_resid\_abs\_log2[ 𝑘 ],  when 𝑘 == 0 || geom\_angular\_enabled | 31 + 𝑘 | Exp2( BinIdx ) + PartVal – 1  +186 × ptn\_inter\_flag[ nodeIdx ] | | 62 |
| ptn\_resid\_abs\_log2[ 𝑘 ],  when 𝑘 > 0 && ¬geom\_angular\_enabled | 31 + 𝑘 | Min( 4, ptn\_resid\_abs\_log2[ 0 ] + 1 >> 1 )  × 31  + Exp2( BinIdx ) + PartVal − 1 | | 155 |
| ptn\_resid\_abs\_rem[ ] | na | bypass | | 0 |
| ptn\_resid\_sign[ 𝑘 ] | 34 | 𝑘 +3 × ptn\_inter\_flag[ nodeIdx ] | | 6 |
| ptn\_sec\_resid\_abs[ 𝑘 ] | 35 | **Offset** | 𝑘 × BinIdxTu | 6 |
| **Prefix** | 6 + 9 × 𝑘 + Min( 4, BinIdxPfx ) | 15 |
| **Suffix** | 11 + 9 × 𝑘 + Min( 3, BinIdx Sfx) | 12 |
| ptn\_sec\_resid\_sign[ 𝑘 ] | 36 | 𝑘 | | 3 |
| ptree\_end\_of\_slice | 37 | 0 | | 1 |
| ptn\_radius\_resid\_abs | 38 | **Offset** | ( ptn\_inter\_flag[ nodeIdx ] ?   ptn\_inter\_pred\_mode[ nodeIdx ] > 1 :   ptn\_pred\_idx[ nodeIdx ] == 0)  + 2 × (ptn\_inter\_flag[ nodeIdx ] ? Abs(PtnPhiMul[ nodeIdx ]) > 2 : Abs(PtnPhiMul[ nodeIdx ]) > thQphi)  + 4 × BinIdxTu  +12 × ptn\_inter\_flag[ nodeIdx ] | 24 |
| **Prefix** | 10 × (   ( ptn\_inter\_flag[ nodeIdx ] ?   ptn\_inter\_pred\_mode[ nodeIdx ] > 1 : ptn\_pred\_idx[ nodeIdx ] ≠ 0) +  2 × (ptn\_inter\_flag[ nodeIdx ] ? Abs(PtnPhiMul[ nodeIdx ]) > 2 : Abs(PtnPhiMul[ nodeIdx ] ) > thQphi) ) + Min( 9, BinIdxPfx ) + 40 × ptn\_inter\_flag[ nodeIdx ] | 80 |
| **Suffix** | 10 × (   ( ptn\_inter\_flag[ nodeIdx ] ?   ptn\_inter\_pred\_mode[ nodeIdx ] > 1 : ptn\_pred\_idx[ nodeIdx ] ≠ 0) +  2 × (ptn\_inter\_flag[ nodeIdx ] ? Abs(PtnPhiMul[ nodeIdx ]) > 2 : Abs(PtnPhiMul[ nodeIdx ]) > thQphi) ) + Min( 9, BinIdxSfx ) +40 × ptn\_inter\_flag[ nodeIdx ] | 80 |
| ptn\_radius\_resid\_sign | 39 | PrevRadiusResidSign    + 2 × (PtnPhiMul[ nodeIdx ] ≠ 0)    + 4 × (PrevPhiMul ≠ 0)    + 8 × (ptn\_inter\_flag[ nodeIdx ] ?           ptn\_inter\_pred\_mode[ nodeIdx ] > 1 :           ptn\_pred\_idx[ nodeIdx ]  == 0)    +16 × (ptn\_inter\_flag[ nodeIdx ] ? 2 : InterFlagHist & 0x1) | | 48 |
| ptn\_phi\_resid\_abs\_gt0 | 41 | (ptn\_inter\_flag[ nodeIdx ] ?   ptn\_inter\_pred\_mode[ nodeIdx ] > 1 :   ptn\_pred\_idx[ nodeIdx ] == 0) + 2 × ptn\_inter\_flag[ nodeIdx ] | | 4 |
| ptn\_phi\_resid\_sign | 42 | (ptn\_inter\_flag[ nodeIdx ] ?   ptn\_inter\_pred\_mode[ nodeIdx ] > 1 :  ( ptn\_pred\_idx[ nodeIdx ] == 0) ) × 5  + (ptn\_inter\_flag[ nodeIdx ] ? 4 :     (InterFlagHist & 0x1 ?      PrevInterFrameRefIdx + 2 : PrevPhiResidSign) ) | | 10 |
| ptn\_phi\_resid\_abs\_gt1 | 43 | ( ptn\_inter\_flag[ nodeIdx ] ?   ptn\_inter\_pred\_mode[ nodeIdx ] > 1 : ptn\_pred\_idx[ nodeIdx ] == 0) + 2 × ptn\_inter\_flag[ nodeIdx ] | | 4 |
| ptn\_phi\_resid\_abs\_rem | 44 | **Prefix** | Min( 3, BinIdxPfx ) + 4 × (ptn\_inter\_flag[ nodeIdx ] ? (ptn\_inter\_pred\_mode[ nodeIdx ] > 1 ) + 1 : 0) | 24 |
| **Suffix** | Min( 3, BinIdx Sfx)  + 4 × (ptn\_inter\_flag[ nodeIdx ] ? (ptn\_inter\_pred\_mode[ nodeIdx ] > 1 ) + 1 : 0) | 24 |
| ptn\_pred\_idx | 45 | BinIdxTu | | 7 |
| ptn\_inter\_flag | 46 | InterFlagHist & 1F | | 32 |
| ptn\_pred\_direction | 49 | 0 | | 1 |
| ptn\_pred\_inter\_mode | 47 | Exp2( BinIdx ) + PartVal − 1 | | 3 |
| 1. The syntax elements occ\_dup\_point\_cnt and direct\_dup\_point\_cnt use the same context table despite using different binarizations. | | | | |

Table 52 — Values of CtxTbl and CtxIdx for binarized ae(v) coded ADU syntax elements

| Syntax element | CtxTbl | CtxIdx | | Count |
| --- | --- | --- | --- | --- |
| coeff\_abs[ 0 ]  when aps\_extension\_present == 0 | 38 | **Offset** | BinIdxTu | 2 |
| **Prefix** | 4 + Min( 4, BinIdx Pfx) | 3 |
| **Suffix** | 9 + Min( 2, BinIdx Sfx) | 3 |
| coeff\_abs[ 1 ]  when aps\_extension\_present == 0 | 38 | **Offset** | For BinIdxTu == 0:    2 + ( coeff\_abs[ 0 ] ≠ 0 )  For BinIdxTu == 1:    4 + ( coeff\_abs[ 0 ] ≤ 1 ) | 4 |
| **Prefix** | 4 + Min( 4, BinIdx Pfx ) | 3 |
| **Suffix** | 9 + Min( 2, BinIdxSfx ) | 3 |
| coeff\_abs[ 2 ]  when aps\_extension\_present == 0 | 39 | **Offset** | For BinIdxTu == 0:    ( coeff\_abs[ 0 ] ≠ 0 )    + 2 × ( coeff\_abs[ 1 ] ≠ 0 )  For BinIdxTu == 1:    4 + ( coeff\_abs[ 0 ] ≤ 1 )    + 2 × ( coeff\_abs[ 1 ] ≤ 1 ) | 8 |
| **Prefix** | 6 + Min( 4, BinIdx Pfx  ) | 3 |
| **Suffix** | 11 + Min( 2, BinIdx Sfx) | 3 |
| coeff\_abs[ 0 ],  when aps\_extension\_present == 1 | 38 | **Offset** | BinIdxTu | 2 |
| **Prefix** | 4 + Min( 13, BinIdx Pfx) | 12 |
| **Suffix** | 18 + Min( 11, BinIdx Sfx) | 12 |
| coeff\_abs[ 1 ],  when aps\_extension\_present == 1 | 38 | **Offset** | For BinIdxTu == 0:    2 + ( coeff\_abs[ 0 ] ≠ 0 )  For BinIdxTu == 1:    4 + ( coeff\_abs[ 0 ] ≤ 1 ) | 4 |
| **Prefix** | 4 + Min( 13, BinIdx Pfx ) | 12 |
| **Suffix** | 18 + Min( 11, BinIdxSfx ) | 12 |
| coeff\_abs[ 2 ],  when aps\_extension\_present == 1 | 39 | **Offset** | For BinIdxTu == 0:    ( coeff\_abs[ 0 ] ≠ 0 )    + 2 × ( coeff\_abs[ 1 ] ≠ 0 )  For BinIdxTu == 1:    4 + ( coeff\_abs[ 0 ] ≤ 1 )    + 2 × ( coeff\_abs[ 1 ] ≤ 1 ) | 8 |
| **Prefix** | 6 + Min( 13, BinIdx Pfx  ) | 12 |
| **Suffix** | 19 + Min( 11, BinIdx Sfx) | 12 |
| coeff\_sign[ ] | na | bypass | | 0 |
| zero\_run\_length\_prefix | 40 | BinIdxTu | | 3 |
| zero\_run\_length\_minus3\_div2 | 40 | 3 | | 1 |
| zero\_run\_length\_minus3\_mod2 | na | bypass | | 0 |
| zero\_run\_length\_minus11 | 40 | **Prefix** | 4 | 1 |
| **Suffix** | bypass | 0 |
| slice\_raht\_inter\_layer\_code\_mode[ lvl ] | na | bypass | | 0 |
| slice\_raht\_intra\_layer\_code\_mode[ lvl] | na | bypass | | 0 |
| 1. The prefix and suffix bins of the syntax elements coeff\_abs[ 0 ] and coeff\_abs[ 1 ] use the same values of CtxIdx and CtxTbl. | | | | |

### Arithmetic decoding engine

#### General

The arithmetic decoding engine is a context-adaptive, binary arithmetic decoder, performing binary renormalization and producing binary outputs.

* 1. An arithmetic encoding engine that complements this decoding engine is described in Annex C.
  2. The arithmetic decoding engine is related to that of SMPTE VC-2.

#### State variables

The arithmetic decoder is specified in terms of the following state variables:

* IvlLow, representing the beginning of the 16-bit coding interval.
* IvlRange, representing the size of the 16-bit coding interval.
* IvlCode, a codeword within the interval [ IvlLow, IvlLow + IvlRange − 1 ], updated from the arithmetic-coded bitstream.

#### Initial state

The arithmetic decoding state variables shall be initialized as follows; and 16 bits shall be read from the arithmetic-coded bitstream:

IvlLow = 0  
IvlRange = 0xFFFF  
IvlCode = 0  
for (i = 0; i < 16; i++) {  
 IvlCode <<= 1  
 IvlCode += NextAeStreamBit  
}

#### Arithmetic-coded bitstream

The next bit to be consumed as input to the arithmetic decoder is specified by the expression NextAeStreamBit.

NextAeStreamBit := bypass\_stream\_enabled ? ChunkNextAeBit : DuNextBit

#### Decoding a single binary symbol

Decoding is parameterized by the probability prob0 that the decoded binary symbol is zero-valued.

The decoded binary value binVal is determined and the state variables IvlRange and IvlCode are updated:

rangeTimesProb = IvlRange × prob0 >> 16  
binVal = rangeTimeProb ≤ IvlCode − IvlLow  
if (¬binVal)  
 IvlRange = rangeTimesProb  
else {  
 IvlLow += rangeTimesProb  
 IvlRange −= rangeTimesProb  
}

#### Decoding a single binary bypass symbol

If bypass\_bin\_coding\_prob\_update\_disabled is 0, the decoded binary value binVal is determined and the state variables IvlRange and IvlCode are updated:

rangeTimesProb = IvlRange >> 1  
binVal = rangeTimeProb ≤ IvlCode − IvlLow  
if (¬binVal)  
 IvlRange = rangeTimesProb  
else {  
 IvlLow += rangeTimesProb  
 IvlRange −= rangeTimesProb  
}

if bypass\_bin\_coding\_prob\_update\_disabled is 1, The decoded binary value binVal is determined and the state variables IvlLow and IvlCode are updated:

IvlCode <<= 1  
IvlLow <<= 1  
binVal = IvlRange ≤ IvlCode − IvlLow  
if (binVal)  
 IvlLow -= IvlRange  
}

#### Arithmetic decoder state renormalization

Renormalization stops the arithmetic decoding engine from losing accuracy. Renormalization shall be applied while the size of the coding interval is less than or equal to a quarter of the total available 16-bit range. Each renormalization doubles the interval and reads a bit into the codeword.

If IvlRange is less than or equal to , the state variables IvlRange, IvlLow and IvlCode are updated:

if ((IvlLow + IvlRange − 1) ^ IvlLow ≥ 0x8000) {  
 IvlCode ^= 0x4000  
 IvlLow ^= 0x4000  
}  
IvlRange <<= 1  
IvlLow = (IvlLow << 1) & 0xFFFF  
IvlCode = ((IvlCode << 1) | NextAeStreamBit) & 0xFFFF

If IvlRange remains less than or equal to , the process shall be repeated until it is not.

#### Arithmetic decoder flushing process

The arithmetic decoder shall be flushed at the end of each occupancy tree entropy stream.

Flushing shall repeatedly perform state renormalization until IvlRange is greater than , and then discard bits from the arithmetic-coded bitstream until it is byte aligned.

while (IvlRange ≤ 0x4000) {  
 NextAeStreamBit  
 IvlRange <<= 1  
}

/\* byte−align \*/  
while (ReadAeStreamIdx % 8)  
 NextAeStreamBit

## Parsing state memorization and restoration

### General

Subclause 11.6 applies when either entropy\_continuation\_enabled is 1, slice\_inter\_entropy\_continuation is 1 or occtree\_stream\_cnt\_minus1 is greater than 0.

At certain moments, the entropy parsing state is recorded and later, used as the initial state for parsing other DUs or occupancy tree entropy streams.

The entropy parsing state shall comprise:

* for a GDU, the CABAC CPMs (11.5.3), the demi-CPMs for bitwise occupancy coding (9.2.10.6), the dictionary codec state for bytewise occupancy coding (9.2.9.4), the planar occupancy coding state (9.2.11.5.2) and the state of the variables PrevInterFrameRefIdx, PrevPhiResidSign, PrevPhiMul and PrevRadiusResidSign (9.3.3.1);
* for an ADU, the CABAC CPMs only (11.5.3).

The entropy parsing state shall be recorded and restored independently according to DU type (ADU versus GDU) and for each different value of ADU AttrIdx. For example, a coded point cloud sequence with num\_attributes equal to 2 would require storage for three sets of entropy parsing state.

At the start of any GDU with slice\_entropy\_continuation equal to 0 and slice\_inter\_entropy\_continuation equal to 0, all previously recorded GDU and ADU entropy parsing state shall be discarded.

### Geometry data units

#### Memorization

The GDU entropy parsing state shall be recorded at:

* the end of every geometry\_data\_unit syntax structure (7.3.3.1); and
* the end of every occupancy\_tree\_level( dpth ) syntax structure (7.3.3.5) where dpth is equal to OcctreeEntropyStreamDepth − 1.

Memorization shall record the elements and values of the GDU entropy parsing state for restoration by the restoration process (11.6.2.2).

#### Restoration

The GDU entropy parsing state shall be restored at:

* the start of a geometry\_data\_unit syntax structure (7.3.3.1) when slice\_entropy\_continuation is 1; and
* the start of a geometry\_data\_unit syntax structure (7.3.3.1) when slice\_entropy\_continuation is 0, slice\_inter\_entropy\_continuation is 1; and
* the start of every occupancy\_tree\_level( dpth ) syntax structure (7.3.3.5) where dpth is greater than OcctreeEntropyStreamDepth.

Restoration shall restore the elements and values of the GDU entropy parsing state to those previously recorded by the memorization process (11.6.2.1). At the start of a geometry\_data\_unit syntax structure, restoration shall exclude the planar occupancy coding state.

### Attribute data units

#### Memorization

The ADU entropy parsing state shall be recorded at the end of every attribute\_data\_unit syntax structure (7.3.4.1).

Memorization shall record the elements and values of the ADU entropy parsing state for restoration by the restoration process (11.6.3.2). The state shall be recorded separately for each value of AttrIdx.

#### Restoration

The ADU entropy parsing state shall be restored at the start of each attribute\_data\_unit syntax structure (7.3.4.1) when either slice\_entropy\_continuation is 1 or slice\_inter\_entropy\_continuation is 1. The restoration shall be from the state recorded by the memorization process (11.6.3.1) with the same value of AttrIdx.

### Defaulted attribute data units

The recorded ADU entropy parsing state for the attribute identified by AttrIdx shall be initialized at the start of each defaulted attribute data unit when slice\_entropy\_continuation is 0 or slice\_inter\_entropy\_continuation is 0. The initialization shall be according to 11.5.3.2 as if the data unit contained a syntax element with the descriptor ae(v).

While a defaulted attribute data unit does not use arithmetic coding, it is necessary to record the initialized ADU entropy parsing state when slice\_entropy\_continuation is 0 or slice\_inter\_entropy\_continuation is 0 so that ADUs in any following slices where slice\_entropy\_continuation is 1 or slice\_inter\_ entropy\_continuation is 1 do not have an indeterminate ADU parsing state.

# OBUF parsing process

## General

The acronym OBUF stands for “Optimal Binary coder with Update on the Fly”. An OBUF instance decodes information from the bitstream to obtain a bit bin.

An OBUF instance is called with two contextual information info1 and info2 as input and follows the steps of

* obtaining, based on internal statistics, an index ctxIdx pointing to an element of a table of OBUF adaptive context probability models (ACPMs) as defined in 12.2.2,
* decoding an arithmetic-coded bin bin by using the CABAC decoder with the pointed OBUF ACPM (12.5),
* and updating internal statistics.

Internal statistics are modeled by a memory channel that evolves (is updated) after the decoding of each bin. The memory channel is stored into three elements:

* an OBUF tree constituted of three arrays *kShift*[ ][ ], *nVisit*[ ][ ] and *ctxIdxMap*[ ][ ],
* a buffer *obufBuffer*[ ][ ] of OBUF tree leaves,
* and an array *obufCtxArray*[ ] of OBUF ACPMs.

## Creation of an OBUF instance

An OBUF instance is a set made of the three above elements, namely an OBUF tree, a buffer of OBUF tree leaves and an array of OBUF ACPMs. The OBUF tree and the array of OBUF ACPMs are uniquely attached to the OBUF instance. However, one buffer of OBUF tree leaves can be shared among several OBUF instances.

Creating an OBUF instance follows the steps of

* associating a buffer of OBUF tree leaves, that has been previously created according to clause 12.3, with the instance,
* creating and initializing OBUF trees according to clause 12.2.1,
* and creating and initializing an array of OBUF ACPMs according to clause 12.2.2.

A buffer of OBUF tree leaves is created before creating an OBUF instance because a buffer can be shared by multiple OBUF instances.

The OBUF instance is created based on the given of

* an OBUF buffer *obufBuffer*[ ][ ] of OBUF tree leaves of depth obufLeafDepth,
* a first size nBit1 in bits corresponding to the size of first contextual information info1 used as input when calling the OBUF instance,
* a second size nBit2 in bits corresponding to the size of second contextual information info2 used as input when calling the OBUF instance,
* and optionally an initialization array initObufArray.

Creation and initialization of the OBUF trees is performed according to clause 12.2.1 based on nBit1, nBit2, obufLeafDepth and initObufArray. Creation and initialization of the array of OBUF ACPMs is performed according to clause 12.2.2.

### Creation and initialization of OBUF trees

The sizes s1 and s2 of the two contextual information as well as the size obufTreeSize2 of the s1 OBUF trees are determined by

s1 = 1 << nBit1  
s2 = 1 << nBit2  
obufTreeSize2 = 1 << nBit2 – obufLeafDepth

Three double-entry 8-bit arrays *kShift*[ ][ ], *nVisit*[ ][ ] and *ctxIdxMap*[ ][ ] are created with size s1 along the first entry and size obufTreeSize2 along the second entry for a total size of s1 × obufTreeSize2 each.

The array *kShift*[ ][ ] is initialized with all s1 × obufTreeSize2 values set to s2. The array *nVisit* [ ][ 0] is initialized with all of the s1 values set to 0. The array *ctxIdxMap* [ ][ 0] is initialized with all of the s1 values set to 127.

The array *kShift*[ ][ ] corresponds to the number of erased bits of a secondary information info2 when the instance is called. The initialising to s2 indicates that all bits of any second information info2 are erased at initial state of the OBUF instance.

The array *nVisit*[ ][ ] corresponds to the number of visits of nodes of the OBUF trees. The initialising to 0 indicates that root nodes of the OBUF trees have not been visited yet at initial state of the OBUF instance.

The array *ctxIdxMap*[ ][ ] corresponds to 8-bit context indices pointing (after right shift by 3) to OBUF ACPMs of the array *obufCtxArray*[ ]. The initialising to 127 indicates pointing to the OBUF ACPM with associated probability 0.5 at initial state of the OBUF instance.

When the optional initialization array initObufArray of size s1 is provided, the array *ctxIdxMap*[ ][ ] is further initialized by

for (j = 0; j < s1; j++)  
 ctxIdxMap[j][0] = initObufArray[j]

This further initialising provides an initial statistical model for the root nodes of the s1 OBUF trees.

### Creation and initialization of an array of OBUF ACPMs

An array of OBUF ACPMS is an extension of an array of ACPMs. It is made of

* a 16-bit array *obufCtxArray*[ ] of size 32 of ACPMs as defined in 11.5.3,
* a 16-bit array obufCtxProbaBounds[ ] of size 33 of probability bounds associated with the 32 models of the array of ACPMs such that the probability *obufCtxArray*[k ] remains between the two bounds *obufCtxProbaBounds*[k ] and *obufCtxProbaBounds*[k+1 ] before decoding a bin.

The array *obufCtxArray*[ ] is initialized by the table ObufInitProba [ ] defined by Table 33, and the array *obufCtxProbaBounds*[ ] is initialized by the table ObufInitProbaBounds [ ] defined by Table 34.

Table 53 — Values of ObufInitProba[ k ]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| value | 65461 | 65160 | 64551 | 63637 | 62426 | 60929 | 59163 | 57141 |
| k | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| value | 54884 | 52413 | 49753 | 46929 | 43969 | 40899 | 37750 | 34553 |
| k | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |
| value | 31338 | 28135 | 24977 | 21893 | 18914 | 16067 | 13382 | 10883 |
| k | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** |
| value | 8596 | 6542 | 4740 | 3210 | 1967 | 1023 | 388 | 75 |

Table 54 — Values of ObufInitProbaBound[ k ]

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |  |
| value | 65535 | 65388 | 64933 | 64169 | 63105 | 61747 | 60112 | 58214 |  |
| k | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |  |
| value | 56069 | 53699 | 51128 | 48379 | 45480 | 42458 | 39340 | 36160 |  |
| k | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** |  |
| value | 32946 | 29730 | 26541 | 23413 | 20374 | 17454 | 14681 | 12083 |  |
| k | **24** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** |
| value | 9684 | 7509 | 5575 | 3905 | 2515 | 1419 | 627 | 150 | 0 |

## Creation of a buffer of OBUF tree leaves

A buffer of OBUF tree leaves is a rolling buffer whose elements are fully deployed trees that can be attached to a leaf node of an OBUF tree to prolong the latter.

A buffer of OBUF tree leaves is created based on the given of

* a buffer size obufBufferSize,
* and a depth obufLeafDepth of fully deployed trees constituting each element of the buffer.

The number deployedTreeSize of leaf nodes of fully deployed trees is

deployedTreeSize = 1 << obufLeafDepth

The buffer is composed of

* one 8-bit array *obufBuffer*[ ][ ] of size obufBufferSize along its first component and of size deployedTreeSize along its second component. The k-th OBUF tree leaf in the buffer is the array *obufBuffer*[k ][ ] of size deployedTreeSize,
* a buffer element index nextUsableIdx indicating the position of the next OBUF tree leaf *obufBuffer*[usableIdx] [ ] that is usable for attaching to a leaf node of an OBUF tree,
* and a rolling flag obufBufferRolled that indicates if the buffer element index has rolled back to the start of the buffer at least once.

[Ed. (SL): not sure the TM source code is as bullet proof as the introduction of the flag. Must be checked.]

An array *obufBuffer*[k ][ ] is made of 8-bit context indices pointing (after right shift by 3) to OBUF ACPMs of the array *obufCtxArray*[ ] associated to an OBUF instance. An array *obufBuffer*[k][] is used to prolong the array *ctxIdxMap*[ ][ ] of the OBUF instance once attached to a leaf node of the OBUF tree of the OBUF instance.

The buffer element index nextUsableIdx and the rolling flag obufBufferRolled are initialized to 0. However, the array *obufBuffer*[ ][ ] does not require initialization as each OBUF tree leaf *obufBuffer*[k][ ] will be initialized when attached to a leaf node of an OBUF tree.

## Call and update of an OBUF instance

An OBUF instance is called with input

* a first contextual information info1 of size nBit1 bits,
* and a second contextual information info2 of size nBit2 bits.

The output is a decoded bin bin.

A reduced second contextual information info2Red and the number nErasedBit of bits to be erased from the second contextual information are obtained by

info2Red = info2 >> obufLeafDepth  
nErasedBit = kShift[ info1][ info2Red]

Depending on the value of nErasedBit, the decoding and update process is performed according to either the OBUF trees or the buffer of OBUF tree leaves. In case nErasedBit >= obufLeafDepth , the process continues to clause 12.4.1; otherwise, the process continues to clause 12.4.2.

### Decode and update according to OBUF trees

#### Decode of a bin and context index update

The second contextual information info2Erased with erased bits, according to the number of erased nErasedBit bits, is computed by

nErasedBitTree = nErasedBit – obufLeafDepth  
info2Erased = (info2Red >> nErasedBitTree) << nErasedBitTree

An 8-bit context index ctxIdx is obtained from the array *ctxIdxMap*[ ][ ].

ctxIdx = ctxIdxMap[info1 ][ info2Erased]

The decoded bin bin is obtained by applying clause 12.5.

Depending on the value of the decoded bin bin, the obtained context index is updated in the array *ctxIdxMap*[ ][ ] by using ObufCtxIdxDelta [ ] as defined in Table 35.

if (bin)   
 ctxIdxMap[info1 ][ info2Erased] += ObufCtxIdxDelta[(255 - ctxIdx) >> 4]   
else  
 ctxIdxMap[info1 ][ info2Erased] -= ObufCtxIdxDelta[ctxIdx >> 4]

Table 55 — Values of ObufCtxIdxDelta [ k ]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| k | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| value | 0 | 1 | 1 | 2 | 4 | 7 | 9 | 11 |
| k | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| value | 14 | 16 | 19 | 23 | 22 | 22 | 20 | 15 |

The number of visits nVisit is incremented by one unit

nVisit[info1 ][ info2Erased]++

and a threshold thVisit on the number of visits is obtained by

thVisit = 3 + abs((ctxIdx-127) >> 4)

If the number of visits nVisit is strictly lower than the threshold thVisit then the call to the OBUF instance is finished. Otherwise, an updating process is performed, and the process continues to clause 12.4.1.2 if nErasedBit > 0 or continues to clause 12.4.1.3 otherwise.

#### OBUF tree update

The node of OBUF trees referenced by info1 and info2Erased is split into two new nodes with no visit for each.

nVisit[ info1][info2Erased] = 0 /\* first new node \*/  
nVisit[ info1][info2Erased + (1 << nErasedBit -1)] = 0 /\* second new node \*/

The context index *ctxIdxMap*[ info1][info2Erased] of the first of the two new nodes is automatically inherited from the node. The context index of the second of the two new nodes is obtained by copying the context index of the node.

ctxIdxMap[ info1][info2Erased + (1 << nErasedBit -1)] = ctxIdxMap[ info1][info2Erased]

The number of erased bits is decreased by one unit for all entries of *kShift*[info1 ][i ] where i corresponds to the two new nodes.

for (i = 0; i < (1<< nErasedBit); i++)   
 kShift[info1][info2Erased + i]--

This terminates the call of the OBUF instance.

#### Attaching of a buffer element

A buffer element of the buffer of OBUF leave is attached to the node of the OBUF tree referenced by info1 and info2Erased. Attaching depends on the availability of a non-used buffer element. This availability is obtained from the value of the rolling flag obufBufferRolled.

When the rolling flag obufBufferRolled is equal to 0, the buffer element *obufBuffer*[nextUsableIdx][ ] is initialized by the context index of the node

for (i = 0; i < deployedTreeSize; i++)   
 obufBuffer[nextUsableIdx][ i] = ctxIdx

and the pointer nextUsableIdx pointing to the attached element of the buffer of OBUF tree leaves is stored as pointer hidden information in the two arrays *ctxIdxMap*[ ][] and *nVisit*[ ][] by

nVisit[ info1][info2Erased] = nextUsableIdx & 255  
ctxIdxMap[ info1][info2Erased] = nextUsableIdx >> 8

The pointer nextUsableIdx is incremented.

nextUsableIdx++

Otherwise, when the rolling flag obufBufferRolled is equal to 1, there is no non-used buffer element left. A search for the best element to attach within a search window between nextUsableIdx and nextUsableIdx + 20 is performed.

distMin = 256  
idxMin = nextUsableIdx  
mask = (1 << obufLeafDepth) – 1

for (b = nextUsableIdx; b < nextUsableIdx + 20; b++) {  
 d = abs(ctxIdx - obufBuffer[b][info2 & mask])   
 if (d < dmin) {  
 distMin = d  
 idxMin = b  
 }

}

The pointer idxMin points to the best element of the buffer of OBUF tree leaves to be attached and this pointer is stored as pointer hidden information in the two arrays *ctxIdxMap*[ ][] and *nVisit*[ ][] by

nVisit[ info1][info2Erased] = idxMin & 255  
ctxIdxMap[ info1][info2Erased] = idxMin >> 8

The pointer nextUsableIdx is then modified by pointing to the element next to the attached element.

nextUsableIdx = idxMin + 1

After attaching a buffer element, independently on the value of obufBufferRolled, if the index nextUsableIdx is greater than or equal to the buffer size obufBufferSize, the buffer is rolled: the index is reset to 0 and the rolling flag obufBufferRolled is set to 1.

The number of erased bits of the node is decremented.

kShift[info1][info2Erased]--

This terminates the call of the OBUF instance.

### Decode and update according to an element of the buffer of OBUF tree leaves

A 16-bit pointer leafIdx to an element of the buffer of OBUF tree leaves is obtained from the pointer hidden information (as created in clause 12.4.1.3) in *ctxIdxMap*[ ][] and *nVisit*[ ][].

leafIdx = (ctxIdxMap[ info1][info2Erased] << 8) + nVisit[ info1][info2Erased]

An 8-bit context index ctxIdx is obtained from the pointed OBUF tree leaf *obufBuffer*[leafIdx][ ].

mask = (1 << obufLeafDepth) - 1  
ctxIdx = obufBuffer[leafIdx][info2 & mask ]

The decoded bin bin is obtained by applying clause 12.5.

Depending on the value of the decoded bin bin, the obtained context index is updated in the array *obufBuffer*[ ][ ] by using ObufCtxIdxDelta[ ] as defined in Table 35.

if (bin)   
 obufBuffer[leafIdx][info2 & mask ] += ObufCtxIdxDelta[(255 - ctxIdx) >> 4]   
else  
 obufBuffer[leafIdx][info2 & mask ] -= ObufCtxIdxDelta[ctxIdx >> 4]

This terminates the call of the OBUF instance.

## Decode of a bin based on an OBUF ACPM

A selected context SelCtx is obtained from the 8-bit context index ctxIdx and the array *obufCtxArray*[ ] of OBUF ACPMs by

idx = ctxIdx >> 3  
SelCtx = obufCtxArray[idx ]

A sanity check and correction of the 16-bit probability prob0 associated with the selected context SelCtx are performed (12.5.1) before decoding a bin (12.5.2) by using CABAC with the corrected selected context SelCtx as input.

### enProbability correction based on probability bounds

Lower and upper bounds of probability for the selected context are obtained by

lowProba = obufCtxProbaBounds[idx+1]   
upProba = obufCtxProbaBounds[idx]

In case the probability prob0 of the selected context SelCtx is not within the bounds, this probability is corrected toward the bounds and the bounds are adjusted by using the table CtxUpdateDelta[  ] defined by Table 36.

if (prob0 > upProba) {  
 prob0 = upProba  
 upProba += CtxUpdateDelta[255 - (upProba >> 8)] >> 2  
 if (idx > 0 && upProba > obufCtxProbaBounds [idx - 1]) {  
 upProba = obufCtxProbaBounds [idx - 1]   
 }  
}

if (prob0 < lowProba) {  
 prob0 = lowProba  
 lowProba -= CtxUpdateDelta[lowProba >> 8] >> 2  
 if (idx < 31 && lowProba > obufCtxProbaBounds [idx + 2]) {  
 lowProba = obufCtxProbaBounds [idx + 2]   
 }  
}

### Decoding of a bin using CABAC

A bin bin of information is decoded according to the clause 11.5.4.5 by using the corrected probability prob0 of the selected context SelCtx. Then, the probability evolves according to clause 11.5.3.3.

1. (normative)  
   Profiles and levels
   1. Overview of profiles and levels

Profiles and levels specify restrictions on bitstreams and hence limits on the capabilities needed to decode the bitstreams. Profiles and levels may also be used to indicate interoperability points between individual implementations.

* 1. This document does not include individually selectable options at the decoder, as this would increase interoperability difficulties.

Each profile specifies a subset of algorithmic features and limits that shall be supported by all decoders conforming to that profile.

* 1. Encoders are not required to use any particular subset of features supported by a profile.

Each level specifies a set of limits on the values that may be coded by G-PCC syntax elements. Level definitions apply to all profiles. For any given profile, a level generally corresponds to a particular decoder processing load and memory capability.

* 1. Requirements on decoder capability

The capabilities of decoders conforming to this document are specified in terms of the ability to decode bitstreams conforming to the constraints of profiles and levels specified in this annex.

When expressing the capabilities of a decoder for a specific profile, the level supported for that profile should also be expressed.

* 1. Profiles
     1. General

All constraints for SPSs, GPSs and APSs that are specified are constraints for the parameter sets that are active during bitstream decoding.

* + 1. Simple, Predictive, Dense and Main profiles

Bitstreams conforming to the Simple, Predictive, Dense or Main profiles shall satisfy:

* the constraints specified in Table A.1;
* the level constraints specified in A.4.

Conformance of a bitstream to a particular profile shall be indicated by:

* simple\_profile\_compliant equal to 1 for bitstreams conforming to the Simple profile;
* predictive\_profile\_compliant equal to 1 for bitstreams conforming to the Predictive profile;
* dense\_profile\_compliant equal to 1 for bitstreams conforming to the Dense profile;
* main\_profile\_compliant equal to 1 for bitstreams conforming to the Main profile.

Decoders conforming to a particular profile at a specific level shall be capable of decoding all bitstreams that:

* indicate conformance with the profile, and
* indicate conformance with a level that is lower than or equal to the decoder's level.

Table A.1 — Allowed values of syntax elements according to profile

| Parameter set | Syntax element | Profile | | | |
| --- | --- | --- | --- | --- | --- |
| Simple | Predictive | Dense | Main |
| SPS | simple\_profile\_compliant | 1 | 0 | 0 | 0 |
| dense\_profile\_compliant | 0 | 0 | 1 | 0 or 1 |
| predictive\_profile\_compliant | 0 | 1 | 0 | 0 or 1 |
| main\_profile\_compliant | 0 | 0 or 1 | 0 or 1 | 1 |
| bypass\_stream\_enabled | 1 | 1 | 0 or 1 | 1 |
| GPS | geom\_tree\_type | 0 | 1 | 0 | 0 or 1 |
| occtree\_bitwise\_coding | 0 | na | 1 | 1 |
| occtree\_coded\_axis\_list\_present | 0 | na | 0 or 1 | 0 or 1 |
| occtree\_neigh\_window\_log2\_minus1 | 0 | na | ≥ 1 | ≥ 0 |
| occtree\_intra\_pred\_max\_nodesize\_log2 | 0 | na | ≥ 0 | ≥ 0 |
| occtree\_adjacent\_child\_enabled | 0 | na | 0 or 1 | 0 or 1 |
| occtree\_planar\_enabled | 0 | na | 0 or 1 | 0 or 1 |
| geom\_angular\_enabled | 0 | 0 or 1 | 0 | 0 or 1 |
| geo\_disable\_planar\_idcm\_angular | 0 | na | 0 or 1 | 0 or 1 |
| APS | attr\_coding\_type | 0 .. 3 | 1 | 0 .. 3 | 0 .. 3 |
| lod\_max\_levels\_minus1 | ≤ 20 | 0 | ≤ 20 | ≤ 20 |
| lod\_decimation\_mode | 1 or 2 | na | 0 .. 2 | 1 or 2 |
| lod\_scalability\_enabled | 0 | 0 | 0 or 1 | 0 or 1 |
| raht\_prediction\_enabled | 0 | na | 0 or 1 | 0 |
| attr\_coord\_conv\_enabled | 0 | 0 or 1 | 0 | 0 or 1 |

* 1. Levels
     1. Level limits

The level to which a bitstream conforms is indicated by a value of level\_idc 20 times the level number specified in Table A.2. All other values of level\_idc are reserved for future use by ISO/IEC.

When comparing level capabilities, a particular level 𝑎 shall be considered to be a lower level than another level 𝑏 when the value of level\_idc for 𝑎 is less than that for 𝑏.

Table A.2 specifies limits for each level:

* MaxSlicePoints specifies the maximum number of points that can be coded by a slice.
* MaxSliceDimLog2 specifies the maximum number of bits that represent a point coordinate in a slice.
* MaxSeqBboxDimLog2 specifies the maximum number of bits that represent a point coordinate in the sequence coordinate system.

Table A.2 — Level limits

| Level | MaxSlicePoints | MaxSliceDimLog2 | MaxSeqBboxDimLog2 |
| --- | --- | --- | --- |
| 4 |  | 21 | 21 |
| 6 |  | 21 | 32 |

* 1. Permitted ranges for syntax elements

Tables A.3 to A.11 specify constraints on coded syntax element values in bitstreams conforming to this version of this document. Other constraints specified in this document may further constrain their permitted ranges.

Unless otherwise specified in this document, a decoder conforming to this version of this document may reject bitstreams containing syntax elements outside the permitted ranges.

[Ed. (YX): Permitted ranges for RAHT inter syntaxes to be updated.]

Table A.3 — Permitted ranges for sequence parameter set syntax elements

| Syntax element | Range |
| --- | --- |
| reserved\_profile\_18bits | 0 |
| sps\_seq\_parameter\_set\_id | 0 |
| seq\_origin\_bits | 0 .. 31 |
| seq\_origin\_log2\_scale | 0 .. 31 |
| seq\_bbox\_size\_bits | 0 .. MaxSeqBboxDimLog2 |
| seq\_unit\_numerator\_minus1 | 0 .. 31 |
| seq\_unit\_denominator\_minus1 | 0 .. 31 |
| seq\_coded\_scale\_exponent | 0 .. 31 |
| seq\_coded\_scale\_mantissa\_bits | 0 .. 31 |
| num\_attributes | 0 .. 63 |
| attr\_components\_minus1 | 0 .. 3 |
| attr\_instance\_id | 0 .. 63 |
| attr\_bitdepth\_minus1 | 0 .. 63 |
| attr\_label | 0 .. 6 |
| attr\_property\_cnt | 0 .. 16 |
| sps\_extension\_present | 0 |

Table A.4 — Permitted ranges for attribute parameter syntax elements

| Syntax element | Range |
| --- | --- |
| attr\_cicp\_colour\_primaries | 0 .. 255 |
| attr\_cicp\_transfer\_characteristics | 0 .. 255 |
| attr\_cicp\_matrix\_coeffs | 0 .. 255 |
| attr\_offset\_bits | 0 .. 64 |
| attr\_scale\_bits | 0 .. 16 |
| attr\_frac\_bits | 0 .. 31 |

Table A.5 — Permitted ranges for tile inventory syntax elements

| Syntax element | Range |
| --- | --- |
| ti\_seq\_parameter\_set\_id | 0 |
| tile\_origin\_bits\_minus1 | 0 .. 30 |
| tile\_size\_bits\_minus1 | 0 .. 30 |
| ti\_origin\_bits\_minus1 | 0 .. 30 |
| ti\_origin\_log2\_scale | 0 .. 31 |

Table A.6 — Permitted ranges for geometry parameter set syntax elements

| Syntax element | Range |
| --- | --- |
| gps\_seq\_parameter\_set\_id | 0 |
| gps\_geom\_origin\_log2\_scale | 0 .. 31 |
| occtree\_intra\_pred\_max\_nodesize\_log2 | 0 .. MaxSliceDimLog2 |
| occtree\_planar\_threshold | 0 .. 127 |
| gps\_angular\_origin\_bits\_minus1 | 0 .. 31 |
| ptree\_ang\_azimuth\_pi\_bits\_minus11 | 0 .. 9 |
| ptree\_ang\_azimuth\_step\_minus1 | 0 .. |
| ptree\_ang\_radius\_scale\_log2 | 0 .. 31 |
| num\_beams\_minus1 | 0 .. 254 |
| beam\_elevation\_init | ± |
| beam\_voffset\_init | ± |
| beam\_steps\_per\_rotation\_init\_minus1 | 0 .. 6 588 396 |
| beam\_elevation\_diff[ 𝑖 ] | ± |
| beam\_voffset\_diff[ 𝑖 ] | ± |
| beam\_steps\_per\_rotation\_diff[ 𝑖 ] | ±6 588 396 |
| geom\_qp | 0 .. 167 |
| ptree\_qp\_period\_log2 | 0 .. 21 |
| occtree\_direct\_node\_qp\_offset | ±167 |
| gps\_extension\_present | 0 .. 1 |
| biprediction\_enabled | 0 .. 2 |
| ptree\_ang\_azimuth\_scaling\_enabled | 0 .. 1 |
| ptree\_ang\_max\_pred\_index | 0 .. 7 |
| ptree\_ang\_pred\_list\_radius\_resid\_threshold | xxx .. xxx[Ed. (JT): To be defined.] |
| ptree\_ang\_radius\_resid\_context\_qphi\_threshold | 0 .. (ptree\_ang\_azimuth\_pi\_bits\_minus11+12) / (ptree\_ang\_azimuth\_step\_minus1+1) |

Table A.7 — Permitted ranges for attribute parameter set syntax elements

| Syntax element | Range |
| --- | --- |
| aps\_seq\_parameter\_set\_id | 0 |
| attr\_coding\_type | 0 .. 3 |
| attr\_primary\_qp\_minus4 | 0 .. 95 |
| attr\_secondary\_qp\_offset | ±95 |
| raht\_prediction\_subtree\_min | 0 .. 19 |
| raht\_prediction\_samples\_min | 0 .. 19 |
| pred\_set\_size\_minus1 | 0 .. 2 |
| pred\_inter\_lod\_search\_range | 0 .. MaxSlicePoints − 1 |
| pred\_dist\_bias\_minus1\_xyz | 0 .. |
| pred\_max\_range\_minus1 | 0 .. |
| lod\_max\_levels\_minus1 | 0 .. MaxSliceDimLog2 – 1 |
| lod\_decimation\_mode | 0 .. 2 |
| lod\_sampling\_period\_minus2 | 0 .. MaxSlicePoints − 2 |
| lod\_initial\_dist\_log2 | 0 .. MaxSliceDimLog2 |
| pred\_direct\_max\_idx\_plus1 | 0 .. pred\_set\_size\_minus1 + 1 |
| pred\_intra\_min\_lod | 0 .. lod\_max\_levels\_minus1 + 1 |
| pred\_intra\_lod\_search\_range | 0 .. MaxSlicePoints − 1 |
| aps\_extension\_present | 0 |
| attr\_inter\_prediction\_search\_range | 0 .. MaxSlicePoints − 1 |
| refAttrIdx | 0 .. num\_attributes − 1 |
| raht\_prediction\_weights[ ] | 0 .. 63 a |
| max\_points\_per\_sort\_log2\_plus1 | 0 .. IntLog2(MaxSlicePoints) + 2 |
| raht\_prediction\_search\_range | 0 .. MaxSlicePoints − 1 |
| a   The following condition should be satisfied: raht\_prediction\_weights[0 ] + 3 × Max(raht\_prediction\_weights[1 ], raht\_prediction\_weights[3 ]) + 3 × Max(raht\_prediction\_weights[2 ], raht\_prediction\_weights[4 ]) ≤ 63 | |

Table A.8 — Permitted ranges for frame-specific attribute properties syntax elements

| Syntax element | Range |
| --- | --- |
| fsap\_seq\_parameter\_set\_id | 0 |
| fsap\_sps\_attr\_idx | 0 .. num\_attributes – 1 |
| fsap\_num\_props | 0 .. 15 |

Table A.9 — Permitted ranges for geometry data unit syntax elements

| Syntax element | Range |
| --- | --- |
| gdu\_temporal\_id | 0 .. 7 |
| slice\_id | 0 ..  − 1 |
| prev\_slice\_id | 0 ..  − 1 |
| slice\_geom\_origin\_log2\_scale | 0 .. 31 |
| slice\_geom\_origin\_bits\_minus1 | 0 .. MaxSeqBboxDimLog2 − 1 |
| slice\_angular\_origin\_bits\_minus1 | 0 .. MaxSeqBboxDimLog2 − 1 |
| occtree\_depth\_minus1 | 0 .. MaxSliceDimLog2 + 3 |
| occtree\_stream\_cnt\_minus1 | 0 .. occtree\_depth\_minus1 |
| slice\_geom\_qp\_offset | ±167 |
| slice\_ptree\_qp\_period\_log2\_offset | ±21 |
| ptn\_radius\_min | 0 .. |
| occ\_subtree\_qp\_offset\_abs[ ][ ][ ] | 0 .. 167 |
| occ\_dup\_point\_cnt[ ] | 0 .. MaxSlicePoints − 1 |
| direct\_dup\_point\_cnt | 0 .. MaxSlicePoints − 1 |
| beam\_idx\_resid\_abs[ ] | 0 .. 254 |
| ptn\_qp\_offset\_abs[ ] | 0 .. 167 |
| ptn\_dup\_point\_cnt[ ] | 0 .. MaxSlicePoints − 1 |
| ptn\_child\_cnt\_xor1[ ] | 0 .. 3 |
| ptn\_inter\_pred\_mode[ ] | 0 .. global\_motion\_enabled ? 3 : 1 |
| ptn\_pred\_mode[ ] | 0 .. Min( 3, dpth ) ab |
| ptn\_phi\_mul\_abs\_minus9[ ][ ] | 0 .. |
| ptn\_sec\_resid\_abs[ ][ ] | 0 .. |
| a   where dpth is per predictive\_tree\_node( dpth, nodeIdx )  b   ptn\_pred\_mode[ ] may be 1 when dpth is 0 if ptree\_ang\_azimuth\_scaling\_enabled is 1 | |

Table A.10 — Permitted ranges for attribute data unit syntax elements

| Syntax element | Range |
| --- | --- |
| adu\_temporal\_id | 0 .. 7 |
| adu\_sps\_attr\_idx | 0 .. num\_attributes – 1 |
| adu\_slice\_id | 0 ..  − 1 |
| last\_comp\_pred\_coeff\_diff | ±255 |
| inter\_comp\_pred\_coeff\_diff | ±255 |
| lod\_dist\_log2\_offset | ±21 |
| attr\_qp\_offset | ±95 |
| attr\_qp\_layer\_cnt\_minus1 | 0 .. MaxSliceDimLog2 − 1 |
| attr\_qp\_layer\_offset[ 𝑖 ][ 𝑐 ] | ±95 |
| attr\_qp\_region\_cnt | 0 .. 1 |
| attr\_qp\_region\_bits\_minus1 | 0 .. MaxSliceDimLog2 − 1 |
| attr\_qp\_region\_offset[ 𝑖 ][ 𝑐 ] | ±95 |
| zero\_run\_length\_minus11 | 0 .. MaxSlicePoints − 11 |
| coeff\_abs | 0 ..  − 1 |
| attr\_AC\_qp\_layer\_cnt\_minus1 | **0 .. MaxSliceDimLog2 − 1** |
| attr\_AC\_qp\_offset[ 𝑖 ][ 𝑐 ][*a*] | ±95 |

Table A.11 — Permitted ranges for defaulted attribute data unit syntax elements

| Syntax element | Range |
| --- | --- |
| defattr\_seq\_parameter\_set\_id | 0 |
| defattr\_reserved\_zero\_3bits | 0 |
| defattr\_sps\_attr\_idx | 0 .. num\_attributes − 1 |
| defattr\_slice\_id | 0 ..  − 1 |
| defattr\_value | 0 .. AttrMaxVal |

1. (normative)  
   Type-length-value encapsulated bytestream format
   1. General

This annex specifies the syntax and semantics of a bytestream format for use by applications that deliver DUs as an ordered stream of bytes without any requirement for further encapsulation in a file format.

The bytestream format comprises a sequence of type-length-value encapsulation structures that each represent a single coded DU syntax structure.

* 1. Syntax and semantics
     1. Syntax

|  |  |
| --- | --- |
| tlv\_encapsulation( ) { | Descriptor |
| tlv\_type | u(8) |
| tlv\_num\_payload\_bytes | u(32) |
| for( 𝑖 = 0; 𝑖 < tlv\_num\_payload\_bytes; 𝑖++ ) |  |
| tlv\_payload\_byte[ 𝑖 ] | u(8) |
| } |  |

* + 1. Semantics

The order of tlv\_encapsulation structures shall follow the decoding order for the encapsulated syntax structures.

tlv\_type identifies the syntax structure represented by tlv\_payload\_byte[ ] as specified by Table B.1.

Table B.1 — Mapping of tlv\_type and associated data unit to syntax tables

| tlv\_type | Syntax table | Description |
| --- | --- | --- |
| 0 | 7.3.2.1 | Sequence parameter set data unit |
| 1 | 7.3.2.5 | Geometry parameter set data unit |
| 2 | 7.3.3.1 | Geometry data unit |
| 3 | 7.3.2.6 | Attribute parameter set data unit |
| 4 | 7.3.4.1 | Attribute data unit |
| 5 | 7.3.2.4 | Tile inventory data unit |
| 6 | 7.3.2.8 | Frame boundary marker data unit |
| 7 | 7.3.5 | Defaulted attribute data unit |
| 8 | 7.3.2.7 | Frame-specific attribute properties data unit |
| 9 | 7.3.2.9 | User data data unit |
| 10 | 7.3.3.1 | Geometry data unit unused for reference |
| 11 | 7.3.4.1 | Attribute data unit unused for reference |

tlv\_num\_payload\_bytes specifies the length in bytes of the syntax element array tlv\_payload\_byte[ ].

tlv\_payload\_byte[ 𝑖 ] is the 𝑖-th byte of payload data.

It is a requirement of bitstream conformance that when a TLV structure containing coded slice geometry has tlv\_type equal to 10, the TLV structure containing the coded slice attributes of the coded slice geometry shall have a tlv\_type equal to 11.

It is a requirement of bitstream conformance that all the geometry TLV data units for all slices within a frame must have the same TLV type, all the attribute TLV data units for all slices within a frame must have the same TLV type.

* 1. Parsing process

The decoder repeatedly parses tlv\_encapsulation structures until the end of the bytestream is encountered (as determined by unspecified means) and the last tlv\_encapsulation structure in the bytestream has been decoded.

After parsing each tlv\_encapsulation structure:

* The array DataUnitBytes is set equal to tlv\_payload\_byte[ ].
* The variable DataUnitLength is set to tlv\_num\_payload\_bytes.
* The parsing process for the syntax structure corresponding to tlv\_type as specified in Table B.1 is performed.

1. (informative)  
   Arithmetic encoding engine

This annex does not form an integral part of this document.

* 1. General

This annex describes an arithmetic encoding engine that complements the arithmetic decoding engine specified by 11.5.4. The encoding engine is essentially symmetric with the decoding engine, i.e. its complementary processes are performed in the same order. Table C.1 illustrates the correspondence between decoding and encoding processes.

Table C.1 — Correspondence between decoder and encoder arithmetic coding processes

| Process | Decoder | Encoder |
| --- | --- | --- |
| Initialization | 11.5.4.3 | C.3 |
| Symbol coding | 11.5.4.4 | C.4 |
| Renormalization | 11.5.4.7 | C.5 |
| Termination | 11.5.4.8 | C.6 |

* 1. State variables

The arithmetic encoding engine is described in terms of the following state variables:

* IvlLow, indicating the bottom of the 16-bit encoding interval.
* IvlRange, indicating the size of the 16-bit encoding interval.
* IvlCarry, a count of unresolved straddle conditions during renormalization.
  1. Initial state

The arithmetic encoding state is initialized before encoding the first binary symbol for an entropy stream:

IvlLow = 0  
IvlRange = 0xFFFF  
IvlCarry = 0

With 16-bit accuracy, 0xFFFF corresponds to an interval width value of (almost) 1.

* 1. Encoding process for a single binary symbol

Encoding is parameterized by the binary symbol binVal and its associated contextual probability prob0 of it being zero-valued.

The binary symbol is encoded by updating the encoding interval bounds [ IvlLow, IvlLow + IvlRange ] according to the symbol value and the contextual probability:

rangeTimesProb = (IvlRange × prob0) >> 16  
if (¬binVal)  
 IvlRange = rangeTimesProb  
else {  
 IvlLow += rangeTimesProb  
 IvlRange −= rangeTimesProb  
}

After encoding the symbol, the interval is renormalized and any available entropy stream bits output according to C.5.

* 1. Arithmetic encoder state renormalization process

Renormalization causes IvlLow and IvlRange to be modified exactly as for the decoder. It is performed when IvlRange is less than or equal to .

If, during renormalization, IvlLow and IvlLow + IvlRange straddle , a carry is recorded.

Bits are output to the entropy stream when IvlLow and IvlLow + IvlRange do not straddle . The output bits include any accumulated carries.

if (IvlRange ≤ 0x4000) {  
 if ((IvlLow + IvlRange − 1) ^ IvlLow ≥ 0x8000) {  
 IvlLow ^= 0x4000  
 IvlCarry++  
 } else {  
 writeBit(Bit(IvlLow, 15))  
 for (; IvlCarry > 0; IvlCarry−−)  
 writeBit(¬Bit(IvlLow, 15))  
 }  
 IvlRange <<= 1  
 IvlLow <<= 1  
 IvlLow &= 0xFFFF  
}

If IvlRange remains less than or equal to , the process is repeated until it is not.

* 1. Arithmetic encoding engine termination process

After encoding all binary symbols, there might be insufficient bits written to the entropy stream for a decoder to determine the final encoded symbols; partly because further renormalization is required – for example, MSBs might agree but the range is still larger than – and partly because there may be unresolved carries.

The following four-stage process adequately flushes the encoder by outputting remaining resolved MSBs, resolving remaining straddle conditions, flushing carry bits and finally byte aligning the output with padding bits.

while ((IvlLow + IvlRange − 1) ^ IvlLow < 0x8000) {  
 writeBit(Bit(IvlLow, 15))  
 for (; IvlCarry > 0; IvlCarry−−)  
 writeBit(¬Bit(IvlLow, 15))  
 IvlRange <<= 1  
 IvlLow <<= 1  
 IvlLow &= 0xFFFF  
}

while ((IvlLow & 0x4000) && ((IvlLow + IvlRange − 1) & 0x4000)) {  
 carry++  
 IvlLow ^= 0x4000  
 IvlLow &= 0x7FFF  
 IvlLow <<= 1  
 IvlRange <<= 1  
}

writeBit(Bit(IvlLow, 15))  
for (; IvlCarry > 0; IvlCarry−−)  
 writeBit(¬Bit(IvlLow, 15))

byte\_align()

1. (normative)  
   Partial decoding and spatial scalability
   1. General

A decoder may decode and reconstruct slice geometry coded by the occupancy tree at a lower precision than specified by 9.2. When used in conjunction with level of detail attribute scalability (lod\_scalability\_enabled is 1), attributes may be decoded and reconstructed for the lower precision geometry.

This annex specifies the lower precision slice geometry a decoder shall output for a partially decoded occupancy tree that excludes nodes smaller than a minimum node size (D.2). This annex shall only apply when the minimum node size is greater than the unit cube.

If LoD attribute scalability is enabled, attribute values shall be determined by partially decoding the slice attribute data in accordance with D.3. Otherwise, attribute values shall not be output by a decoder.

* 1. Partial slice geometry
     1. General

A decoder shall generate a lower-precision slice geometry that is equivalent to that specified by subclause D.2 at the end of the geometry decoding specified by Clause 9.

While the lower-precision slice geometry is specified as a post-process, it is equivalent to halting the decoding of an occupancy tree at the start of the tree level where NodeSizeLog2, is equal to MinNodeSizeLog2, quantizing the positions of any points coded by direct nodes and eliminating any coincident points.

The lower-precision slice geometry is specified in terms of the following variables:

* The variable MinNodeSizeLog2, an application-specific minimum occupancy tree node size that specifies the precision of output points.
* The array PartialPtIdx that maps points in the lower-precision slice geometry to point indexes in PointPos; PartialPtIdx[ idx ] is a point's index into the array PointPos.
* The variable PartialPtCnt, a cumulative count of points in the lower-precision slice geometry.
  + 1. Selection of partial point positions

To select the points that form the lower-precision slice geometry:

* Spatially partition the full slice geometry into a lattice sized cubic blocks.
* Select one point from each occupied block, recording its PointPos index.

In the following, the sparse array blkPtCnt identifies the blocks of the partitioned geometry; blkPtCnt[ ps ][ pt ][ pv ] greater than 0 indicates that the quantized point position ( ps, pt, pv ) is already present in the output partial slice geometry. Unset elements of blkPtCnt are inferred to be 0.

PartialPtCnt = 0  
for (ptIdx = 0; ptIdx < PointCnt; ptIdx++) {  
 ps = PointPos[ptIdx][0] >> MinNodeSizeLog2  
 pt = PointPos[ptIdx][1] >> MinNodeSizeLog2  
 pv = PointPos[ptIdx][2] >> MinNodeSizeLog2  
  
 blkPtCnt[ps][pt][pv]++  
 if (blkPtCnt[ps][pt][pv] > 1)  
 continue  
  
 PartialPtIdx[PartialPtCnt++] = ptIdx  
}

* + 1. Partial point positions

The points selected for the lower-precision slice geometry shall be quantized according to the minimum node size:

for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)  
 for (k = 0; k < 3; k++)  
 PointPos[ptIdx][k] = (PointPos[ptIdx][k] >> MinNodeSizeLog2) << MinNodeSizeLog2

If MinNodeSizeLog2 is greater than 1, points shall be centred within their corresponding block:

for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)  
 for (k = 0; k < 3; k++)  
 PointPos[ptIdx][k] |= (MinNodeSizeLog2 > 1) << MinNodeSizeLog2 − 1

* + 1. Output points

Partial decoding shall be equivalent to only outputting the selected points; i.e. PartialPtIdx[ 𝑖 ], 𝑖 ∈ 0 .. PartialPtCnt − 1.

* 1. Partial attribute decoding
     1. General

Slice attributes with lod\_scalability\_enabled equal to 1 shall be reconstructed in accordance with Clause 10; but:

* using the lower-precision slice geometry (D.2.3);
* constructing the finest detail level in accordance with D.3.2 instead of 10.6.5.2.

This is equivalent to not reconstructing LoDs with Lvl < MinNodeSizeLog2.

* + 1. The finest detail level

The finest detail level shall comprise the lower-precision slice geometry specified by D.2:

for (i = 0; i < PartialPtCnt; i++)  
 LodPtIdx[0][i] = PartialPtIdx[i]  
LodPtCnt[0] = PartialPointCnt

The point indexes of the finest detail level shall be sorted in ascending order of their respective Morton-coded attribute coordinates.

1. (normative)  
   Fine granularity slices
   1. General

This annex specifies the fine granularity slices decoder. This annex shall only apply when fgs\_layer\_group\_enabled equals to 1.

* 1. Coded point cloud format
     1. Fine granularity slices

A slice can comprise FGSs, where each FGS is mapped one-to-one to geometry or attribute of a subgroup in a layer-group.

The slice of FGSs is identified by a common slice identifier (slice\_id).

A FGS is identified by a pair consisting of a layer-group index (layer\_group\_idx) and a subgroup index (subgroup\_idx). When FGSs have the same pair of indexes, they shall be either the geometry or attribute of the subgroup indicated by the layer-group index and the subgroup index.

Every FGS shall include a GDU or DGDU that codes the partial slice geometry, or ADUs or DADUs that code the partial slice attributes.

The first FGS in a slice shall be FGS of GDU. This FGS may be followed by FGSs of DGDU that depends on the previously decoded GDU and DGDU.

The first FGS in a slice attribute shall be FGS of ADU. This FGS may be followed by FGSs of DADU that depends on the previously decoded ADU and DADU. FGSs of ADUs and DADUs shall be occur after FGSs of GDU and DGDU.

* + 1. Layer-group structure

In the layer-group structure, nodes in the occupancy tree or attribute-assigned occupancy tree are grouped into a layer-group and a subgroup.

A layer-group is a group of consecutive tree levels, where each tree levels shall belong to only one layer-group. The minimum depth of a subgroup shall be the maximum depth of its parent subgroup plus 1, or 0 for the root layer-group. The maximum depth of a subgroup shall be the minimum depth of its child subgroup minus 1, or the greatest depth of the occupancy tree for the last layer-group. A layer-group is identified by a layer-group index (layer\_group\_idx).

A subgroup is a spatial subset of a layer-group, where a node in a tree level shall belong to only one subgroup in a layer-group. The range of the position of the nodes in a subgroup shall be described by a bounding box, which must not overlap with the bounding boxes of other subgroups in the same layer-group. The set of the nodes in all subgroups in a layer-group shall be identical to the set of the nodes in the layer-group. There is only one subgroup for the root layer-group. A subgroup in a layer-group is identified by a subgroup index (subgroup\_idx).

In Figure X (left), a layer-group structure of an occupancy tree with maximum depth of 8 is depicted. In this example, three layer-groups are defined and each layer-group comprises tree levels from depth 0 to 3, 4 to 6, and 7 and 8, respectively. Except for the root layer-group, layer-groups may comprise subgroups. A subgroup is indicated by the pair consisting of the layer-group index and the subgroup index. For example, the root layer-group is indicated by (0, 0).

In Figure X (right), the spatial region of subgroups from Figure X (left) are depicted by a rectangular bounding box in a xy-plane. When the bounding box of a subgroup in a layer-group is a superset of the bounding box of one or more subgroups in the next layer-group, the subgroups in adjacent layer-groups are in a parent and child relationship. In this example, subgroup (0,0) is the parent of subgroups (1,0) and (1,1). Similarly, subgroups (2,0) and (2,1) are children of subgroup (1,0). Each subgroup, indicated by a pair of layer-group index and subgroup index, is in different FGSs.

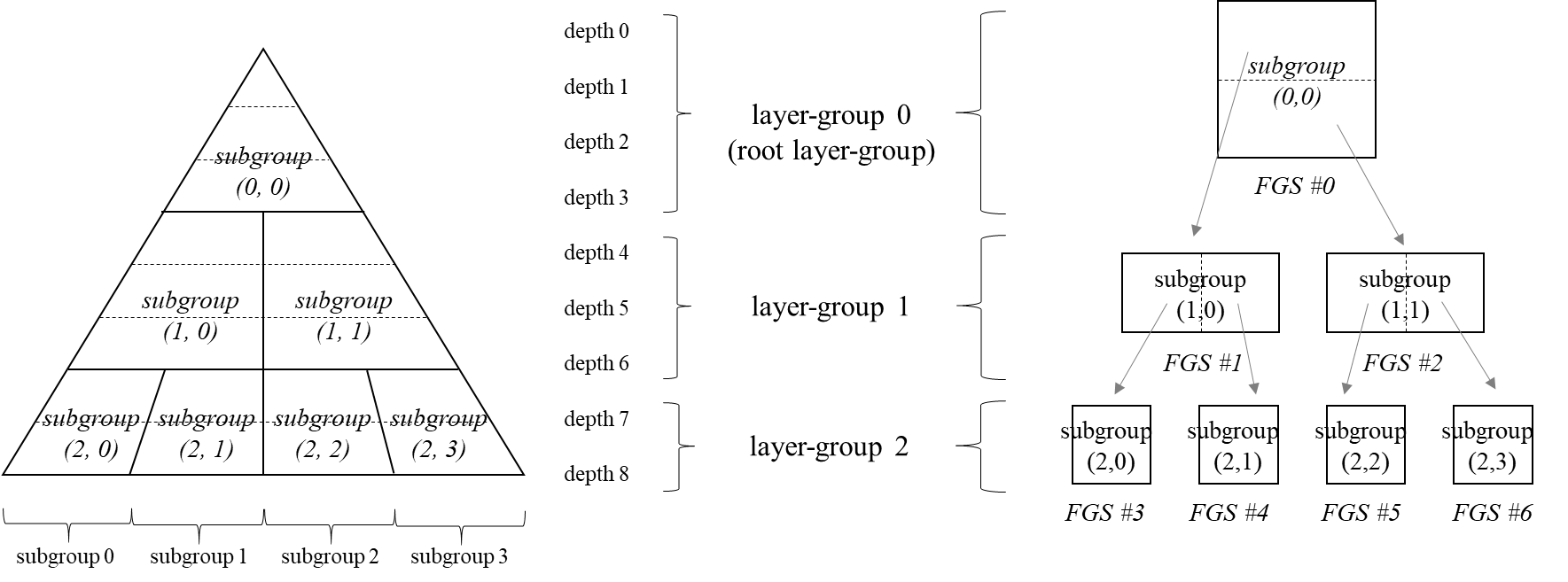


Figure X — (Left) A layer-group structure of an occupancy tree   
(Right) parent and child relationship between subgroups and corresponding FGSs.

* 1. Syntax and semantics for fine granularity slices
     1. Syntax in tabular form
        1. General

The fine granularity slice syntax structures and the syntax elements within these structures are specified in E.3.1. Any values that are not specified in the tables shall not be present in the bitstream unless otherwise specified in this document.

* + - 1. FGS parameter sets
         1. FGS parameter syntax

|  |  |
| --- | --- |
| fgs\_ parameter ( ) { | Descriptor |
| num\_layer\_groups\_minus1 | u(8) |
| for( i = 0; i ≤ num\_layer\_groups\_minus1; i++ ) { |  |
| layer\_group\_id[ i ] | u(8) |
| num\_layers\_minus1[ i ] | u(8) |
| subgroup\_enabled[ i ] | u(1) |
| } |  |
| fgs\_subgroup\_enabled | u(1) |
| if(fgs\_subgroup\_enabled ) { |  |
| subgroup\_bbox\_origin\_bits\_minus1 | ue(v) |
| subgroup\_bbox\_size\_bits\_minus1 | ue(v) |
| } |  |
| for( k = 0; k ≤ 3; k++ ) |  |
| root\_subgroup\_bbox\_size\_log2[ k] | u(8) |
| } |  |

[Ed. (HH): suggested to change fgs\_parameter syntax position. The proponent/editor will prepare a contribution. Comments are the followings: -check whether the position is correct or not. Suggested to move the position to GDU header or GPS (if there is clear reason to move those syntax into GPS) instead of SPS. Those syntax are designed for a slice. -check whether layer\_group\_id is necessary. Suggested to remove layer\_group\_id.]

* + - * 1. FGS attribute parameter syntax

|  |  |
| --- | --- |
| fgs\_ attr\_parameter ( ) { | Descriptor |
| attr\_ref\_id\_present | u(1) |
| } |  |

* + - * 1. FGS layer-group structure inventory syntax

|  |  |  |
| --- | --- | --- |
| fgs\_layer\_group\_structure\_inventory( ) { | Descriptor | Semantics |
| lgsi\_seq\_parameter\_set\_id | u(4) | E.3.2.2.3 |
| lgsi\_frame\_ctr\_lsb\_bits | u(5) | E.3.2.2.3 |
| lgsi\_frame\_ctr\_lsb | u(v) | E.3.2.2.3 |
| lgsi\_num\_slice\_minus1 | u(16) | E.3.2.2.3 |
| if(lgsi\_num\_slices\_minus1 0 ) { |  |  |
| for(sId=0; sId ≤ lgsi\_num\_slices\_minus1; sId++) { |  |  |
| lgsi\_slice\_id[sId] | ue(v) | E.3.2.2.3 |
| lgsi\_num\_layer\_groups\_minus1[sId] | u(8) | E.3.2.2.3 |
| lgsi\_subgroup\_bbox\_origin\_bits\_minus1 | ue(v) | E.3.2.2.3 |
| lgsi\_subgroup\_bbox\_size\_bits\_minus1 | ue(v) | E.3.2.2.3 |
| for(gId=0; gId ≤ lgsi\_num\_layer\_groups\_minus1[sId]; gId++) { |  |  |
| lgsi\_layer\_group\_id[sId][gId] | u(8) | E.3.2.2.3 |
| lgsi\_num\_layers\_minus1[sId][gId] | u(8) | E.3.2.2.3 |
| lgsi\_num\_subgroups\_minus1[sId][gId] | u(16) | E.3.2.2.3 |
| for(sgId=0; sgId ≤ lgsi\_num\_subgroups\_minus1[sId][gId]; sgId++) { |  |  |
| lgsi\_subgroup\_id[sId][gId][sgId] | u(16) | E.3.2.2.3 |
| lgsi\_parent\_subgroup\_id[sId][gId][sgId] | u(16) | E.3.2.2.3 |
| for(k=0; k<3; k++) |  |  |
| lgsi\_subgroup\_bbox\_origin\_xyz[sId][gId][sgId][k] | u(v) | E.3.2.2.3 |
| for(k=0; k<3; k++) |  |  |
| lgsi\_subgroup\_bbox\_size\_xyz[sId][gId][sgId][k] | u(v) | E.3.2.2.3 |
| } |  |  |
| } |  |  |
| } |  |  |
| } |  |  |
| lgsi\_origin\_bits\_minus1 | ue(v) | E.3.2.2.3 |
| for(k=0; k<3; k++) |  |  |
| lgsi\_origin\_xyz[k] | se(v) | E.3.2.2.3 |
| lgsi\_origin\_log2\_scale | ue(v) | E.3.2.2.3 |
| byte\_alignment( ) |  |  |
| } |  |  |

* + - 1. FGS geometry data unit
         1. FGS geometry data unit parameter syntax

|  |  |  |
| --- | --- | --- |
| fgs\_geometry\_data\_unit\_parameter( ) { | Descriptor | Semantics |
| for( i = 1; i ≤ num\_layer\_groups\_minus1; i ++ ) |  |  |
| num\_subsequent\_subgroups[ i ] | u(8) | E.3.2.3.2 |
| if( occtree\_planar\_enabled && ! geom\_angular\_enabled) { |  |  |
| for( i = 0; i ≤ num\_layers\_minus1[ 0 ]; i++ ) |  |  |
| subgroup\_planar\_eligibility\_by\_density[ i ] | u(1) | E.3.2.3.2 |
| } |  |  |
| } |  |  |

[Ed. (HH): The syntax for gps.geom\_octree\_depth\_planar\_eligibiity\_enabled\_flag is not defined yet. The condition for subgroup\_planar\_elibility by density[] needs to be updated to (octree\_planar\_enabled && gps.geom\_octree\_depth\_planar\_eligibiity\_enabled\_flag && geom\_angular\_enabled)]

* + - * 1. FGS occupancy tree syntax

|  |  |  |
| --- | --- | --- |
| fgs\_occupancy\_tree( startDepth, endDepth) { | Descriptor | Semantics |
| OccQpSubtreeDepth = occtree\_depth\_minus1 + 1 |  | 9.2.14.4 |
| for( Dpth = startDepth; Dpth ≤ endDepth; Dpth++ ) { |  |  |
| occupancy\_tree\_level( Dpth ) |  |  |
| if( Dpth + 1 > OcctreeEntropyStreamDepth ) |  | 9.2.3 |
| occtree\_end\_of\_entropy\_stream | ae(v) | 9.2.3 |
| } |  |  |
| } |  |  |

* + - * 1. FGS dependent geometry data unit syntax

|  |  |
| --- | --- |
| dependent\_geometry\_data\_unit( ) { | Descriptor |
| dependent\_geometry\_data\_unit\_header( ) |  |
| fgs\_occupancy\_tree(startDepth, endDepth) |  |
| geometry\_data\_unit\_footer(*occtreeMaxDepthMinus1*  ) |  |
| } |  |

* + - * 1. FGS dependent geometry data unit header syntax

|  |  |  |
| --- | --- | --- |
| dependent\_geometry\_data\_unit\_header( ) { | Descriptor | Semantics |
| dgdu\_geometry\_parameter\_set\_id | u(4) | E.3.2.3.4 |
| dgdu\_slice\_id | ue(v) | E.3.2.3.4 |
| layer\_group\_id | u(8) | E.3.2.3.4 |
| if( subgroup\_enabled[ *layer\_group\_id*] ) { |  |  |
| subgroup\_id | u(8) | E.3.2.3.4 |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| subgroup\_bbox\_origin[ 𝑘 ] | u(v) | E.3.2.3.4 |
| for( 𝑘 = 0; 𝑘 < 3; 𝑘++ ) |  |  |
| subgroup\_bbox\_size[ 𝑘 ] | u(v) | E.3.2.3.4 |
| } |  |  |
| ref\_layer\_group\_id | u(8) | E.3.2.3.4 |
| if( subgroup\_enabled[ *layer\_group\_id*] ) |  |  |
| ref\_subgroup\_id | u(8) | E.3.2.3.4 |
| subgroup\_context\_reference\_indication\_enabled | u(1) | E.3.2.3.4 |
| if( subgroup\_context\_reference\_indication\_enabled ) |  |  |
| for( i = 1; i ≤ num\_layer\_groups\_minus1; i ++ ) |  |  |
| num\_subsequent\_subgroups[ i ] | u(8) | E.3.2.3.2 |
| if( occtree\_planar\_enabled && ! geom\_angular\_enabled) { |  |  |
| for( i = 0; i ≤ num\_layers\_minus1[ *layer\_group\_id*  ]; i++ ) |  |  |
| subgroup\_planar\_eligibility\_by\_density[ i ] | u(1) | E.3.2.3.4 |
| } |  |  |
| byte\_alignment( ) |  |  |
| } |  |  |

[Ed. (HH): The syntax for gps.geom\_octree\_depth\_planar\_eligibiity\_enabled\_flag is not defined yet. The condition for subgroup\_planar\_elibility by density[] needs to be updated to (octree\_planar\_enabled && gps.geom\_octree\_depth\_planar\_eligibiity\_enabled\_flag && geom\_angular\_enabled)]

* + - 1. FGS attribute data unit
         1. FGS attribute data unit parameter syntax

|  |  |  |
| --- | --- | --- |
| fgs\_attribute\_data\_unit\_parameter( ) { | Descriptor | Semantics |
| for( i = 1; i ≤ num\_layer\_groups\_minus1; i ++ ) |  |  |
| num\_subsequent\_subgroups[ i ] | u(8) | E.3.2.3.2 |
| subgroup\_weight\_adjustment\_enabled | u(1) | E.3.2.4.2 |
| if( subgroup\_weight\_adjustment\_enabled) { |  |  |
| subgroup\_weight\_adj\_coeff\_a | se(v) | E.3.2.4.2 |
| subgroup\_weight\_adj\_coeff\_b | se(v) | E.3.2.4.2 |
| } |  |  |
| } |  |  |

* + - * 1. FGS dependent attribute data unit syntax

|  |  |
| --- | --- |
| dependent\_attribute\_data\_unit( ) { | Descriptor |
| dependent\_attribute\_data\_unit\_header( ) |  |
| if(  attr\_coding\_type == 1 || attr\_coding\_type == 2 ) |  |
| attribute\_coeffs( ) |  |
| byte\_alignment( ) |  |
| } |  |

* + - * 1. FGS dependent attribute data unit header syntax

|  |  |  |
| --- | --- | --- |
| dependent attribute\_data\_unit\_header( ) { | Descriptor | Semantics |
| dadu\_attr\_parameter\_set\_id | u(4) | E.3.2.4.4 |
| dadu\_sps\_attr\_idx | ue(v) | E.3.2.4.4 |
| dadu\_slice\_id | ue(v) | E.3.2.4.4 |
| dadu\_layer\_group\_id |  |  |
| if(subgroup\_enabled[dadu\_layer\_group\_id]) |  |  |
| dadu\_subgroup\_id |  |  |
| if( attr\_ref\_id\_present ) |  |  |
| attr\_ref\_layer\_group\_id | u(8) | E.3.2.4.4 |
| if(subgroup\_enabled[dadu\_layer\_group\_id]) |  |  |
| if( attr\_ref\_id\_present) { |  |  |
| attr\_ref\_subgroup\_id | u(16) | E.3.2.4.4 |
| attr\_subgroup\_context\_reference\_indication\_enabled | u(1) | E.3.2.4.4 |
| if(attr\_ subgroup\_context\_reference\_indication\_enabled ) |  |  |
| for( i = 1; i ≤ num\_layer\_groups\_minus1; i ++ ) |  |  |
| num\_subsequent\_subgroups[ i ] | u(8) | E.3.2.3.2 |
| } |  |  |
| if( last\_comp\_pred\_enabled && AttrDim == 3) |  |  |
| for( dpth = 0; dpth ≤ num\_layers\_minus1[dadu\_layer\_group\_id]; dpth++ ) |  |  |
| last\_comp\_pred\_coeff\_diff[ dpth ] | se(v) | 10.6.10.1 |
| if( inter\_comp\_pred\_enabled ) |  |  |
| for( dpth = 0; dpth ≤ num\_layers\_minus1[dadu\_layer\_group\_id]; 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 |
| } |  |  |
| subgroup\_weight\_adjustment\_enabled | u(1) | E.3.2.4.4 |
| if( subgroup\_weight\_adjustment\_enabled) { |  |  |
| subgroup\_weight\_adj\_coeff\_a | se(v) | E.3.2.4.4 |
| subgroup\_weight\_adj\_coeff\_b | se(v) | E.3.2.4.4 |
| } |  |  |
| byte\_alignment( ) |  |  |
| } |  |  |

* + 1. Semantics
       1. General

The semantics associated with the fine granularity slice syntax structures and with the syntax elements within these structures are specified either in E.3.2 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.

* + - 1. FGS parameter sets
         1. FGS parameter semantics

num\_layer\_groups\_minus1 plus 1 specifies the number of layer-groups where the layer-group represents a group of consecutive tree levels within the occupancy tree. num\_layer\_groups\_minus1 shall be in the range of 0 to the number of coding tree layers.

layer\_group\_id[ i ] specifies the indicator of the i-th layer-group of a slice. The range of layer\_group\_id[ i ] shall be in the range 0 .. num\_layer\_groups\_minus1.

num\_layers\_minus1[ i ] plus 1 specifies the number of tree levels in the i-th layer-group. The total number of layer-groups shall be derived by adding all (num\_layers\_minus1[ i ] + 1) for i equal to 0 to num\_layer\_groups\_minus1[ i ].

subgroup\_enabled[ i ] equals to 1 specifies that the i-th layer-group comprises two or more subgroups. subgroup\_enabled[ i ] equals to 0 specifies that the i-th layer-group comprises a subgroup.

When subgroup\_enabled[ i ] equals to 1, the aggregation of the nodes in each subgroup in the i-th layer-group shall be identical to the set of nodes in the layer-group.

When subgroup\_enabled[ i ] is equal to 1, subgroup\_enabled[ j ] shall equal to 1 when j is greater than i.

fgs\_subgroup\_enabled equals to 1 specifies that any layer-group in FGS comprises two or more subgroups. fgs\_subgroup\_enabled equals to 0 specifies that all layer-groups comprise a subgroup.

fgs\_subgroup\_enabled = 0  
for (  
i := 0; i <= num\_layer\_groups\_minus1; i++)  
 fgs\_subgroup\_enabled |= subgroup\_enabled[i]

subgroup\_bbox\_origin\_bits\_minus1 plus 1 specifies the length in bits of the syntax elements subgroup\_ bbox\_origin.

subgroup\_bbox\_size\_bits\_minus1 plus 1 is the length in bits of the syntax elements subgroup\_bbox\_size.

root\_subgroup\_bbox\_size\_log2[ k ] specifies the size of the bounding box of the root subgroup of the coded occupancy tree. MaxVec(root\_subgroup\_bbox\_size\_log2) shall be the number of coded tree layers from root to the leaf layer.

When fgs\_layer\_group\_enabled is equal to 1, the value of occtreeMaxDepthMinus1 is set as

occtreeMaxDepthMinus1 = MaxVec(root\_subgroup\_bbox\_size\_log2)- 1

The number of missing layers of a subgroup shall be the difference between occtreeMaxDepthMinus1 plus 1 and number of levels of decoded occupancy tree. The number of missing layers of a subgroup shall be used to derive the sampling direction in the subgroup LoD generation (E.6.3.3.1) or to compensate the geometry position of a node in the intermediate layers (E.8.2.3).

* + - * 1. FGS attribute parameter semantics

attr\_ref\_id\_present specifies whether (when 1) or not (when 0) the context references of the attribute FGS are present in the DADU header. When attr\_ref\_id\_present equals to 1 indicates the context reference of the attribute FGSs will be indicated by the attr\_ref\_layer\_group\_id and attr\_ref\_subgroup\_id, and the context state usage in the followed attribute slices is indicated by the attr\_subgroup\_context\_reference\_indication\_enabled in the DADU header. When attr\_ref\_id\_present is equal to 0, the context reference and the context state usage shall be inherited from the geometry FGS whose layer\_group\_id and subgroup\_id are identical to dadu\_layer\_group\_id and dadu\_subgroup\_id of the current dependent attribute FGS, respectively.

* + - * 1. FGS layer-group structure inventory semantics

A layer-group structure inventory, when present, contains metadata that defines the spatial region and parent-child relationship of each enumerated layer-groups and subgroups. Each layer-group and subgroup is identified by either an implicit or explicit layer-group id and subgroup id.

A layer-group structure inventory shall apply from the next coded point cloud frame that follows the layer-group structure inventory. It shall remain valid until it is replaced by another layer-group structure inventory.

A layer-group structure inventory 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.

lgsi\_seq\_parameter\_set\_id identifies that active SPS by its sps\_seq\_parameter\_set\_id.

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

lgsi\_frame\_ctr\_lsb specifies the lgsi\_frame\_ctr\_lsb\_bits least significant bits of FrameCtr from which the group structure inventory is valid. A layer-group structure inventory remains valid until it is replaced by another layer-group structure inventory.

lgsi\_num\_slices\_minus1plus 1 specifies the number of slices present in the layer-group structure inventory.

lgsi\_slice\_id[sId] specifies the slice id of the sId-th slice within the layer-group structure inventory.

lgsi\_num\_layer\_groups\_minus1[sId] plus 1 specifies the number of layer-group in the sId-th slice.

lasi\_subgroup\_bbox\_origin\_bits\_minus1[sId] plus 1 is the length in bits of the syntax elements lgsi\_ subgroup\_bbox\_origin\_xyz in the sId-th slice.

lgsi\_subgroup\_bbox\_size\_bits\_minus1[sId] plus 1 is the length in bits of the syntax elements lgsi\_ subgroup\_bbox\_size\_xyz in the sId-th slice.

lgsi\_layer\_group\_id[sId][gId]specifies the indicator of a layer-group. The range of lgsi\_layer\_group\_id shall be in the range of 0 to lgsi\_num\_layer\_groups\_minus1. It is a requirement of bitstream conformance that all values of lgsi\_layer\_group\_id of gId-th layer-group in the sId-th slice are unique within a layer-group structure inventory.

lgsi\_num\_layers\_minus1[sId][gId]plus 1 specifies the number of layers in the layer-group of gId-th layer-group in the sId-th slice.

lgsi\_num\_subgroups\_minus1[sId][gId]plus 1 specifies the number of subgroups in the gId-th layer-group in the sId-th slice.

lgsi\_subgroup\_id[sId][gId][sgId]specifies the indicator of a subgroup. The range of lgsi\_subgroup\_id shall be in 0 to lgsi\_num\_subgroups\_minus1[sId][gId].

lgsi\_parent\_subgroup\_id[sId][gId][sgId] specifies the indicator of a parent subgroup of the sgId-th subgroup in the gId-th layer-group in the sId-th slice. The range of lgsi\_parent\_subgroup\_id shall be in 0 to gi\_num\_subgroups\_minus1[sId][gId-1].

lgsi\_subgroup\_bbox\_origin\_xyz[sId][gId][sgId][ 𝑘 ] specifies the 𝑘-th XYZ coordinate of the bounding box of the sgId-th subgroup in the gId-th layer-group in the sId-th slice.

lgsi\_subgroup\_bbox\_size\_xyz[sId][gId][sgId][ 𝑘 ] specifies the 𝑘-th XYZ dimension of the bounding box of the sgId-th subgroup in the gId-th layer-group in the sId-th slice.

lgsi\_origin\_bits\_minus1 plus 1 specifies the length in bits of the lgsi\_origin\_xyzsyntax elements.

lgsi\_origin\_xyz[ 𝑘 ] indicate the XYZ origin of the sequence. The value of lgsi\_origin\_xyz[k] should be equal to seq\_origin\_xyz[ 𝑘 ].

lgsi\_origin\_log2\_scale indicates a scaling factor to scale components of lgsi\_origin\_xyz. The value of lgsi\_origin\_log2\_scale should be equal to seq\_origin\_log2\_scale.

* + - 1. FGS geometry data unit
         1. FGS geometry data unit semantics

When fgs\_layer\_group\_enabled is equal to 1, a GDU conveys the partial slice geometry of a root layer-group and associated slice information. A GDU of fine granularity slices comprises a GDU header, geometry coded using an occupancy tree, and a GDU footer.

* + - * 1. FGS geometry data unit parameter semantics

num\_subsequent\_subgroups[ i ] specifies the number of the subsequent dependent data units belong to the i-th layer-group which reference the context state of the current data unit. When not present, the value of num\_subsequent\_subgroups[ i ] is inferred to be zero.

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.

* + - * 1. FGS dependent geometry data unit semantics

A DGDU conveys partial slice geometry and associated fine granularity slice information such as a pair of layer-group index and subgroup index, subgroup bounding box, and context state reference index. A DGDU comprises a DGDU header, geometry coded using an occupancy tree, and a GDU footer.

When fgs\_layer\_group\_enabled equals to 1 and layer\_group\_id greater than 0, slice\_num\_points\_minus1 plus 1 shall equal to the number of nodes at the maximum depth of tree levels in the DGDU.

[Ed. (HH): if layer\_group\_id of FGS geometry parameter is removed, an update may be required]

* + - * 1. FGS dependent geometry data unit header semantics

dgdu\_geometry\_parameter\_set\_id specifies the active GPS indicated by gps\_geom\_parameter\_set\_id. The value of dgdu\_geometry\_parameter\_set\_id shall be identical to the value of gdu\_geometry\_parameter\_set\_id in the same slice.

dgdu\_slice\_id specifies the slice to which the current dependent geometry data unit belongs.

layer\_group\_id specifies the indicator of a layer-group in the layer-group structure related to the slice. The range of layer\_group\_id shall be in the range 0 .. num\_layer\_groups\_minus1. When not present, layer\_group\_id shall be inferred to be 0.

subgroup\_id specifies the indicator of the subgroup in the layer-group referred by layer\_group\_id. The range of subgroup\_id shall be in the range 0 .. num\_subgroups\_minus1[layer\_group\_id] where the subgroup\_id represent a partial region in a layer-group specified by layer\_group\_id. When not present, subgroup\_id shall be inferred to be 0.

subgroup\_bbox\_origin\_xyz[ 𝑘 ] specifies the minimum 𝑘-th XYZ component position of the subgroup bounding box of the subgroup indicated by the pair of layer-group index layer\_group\_id and the subgroup index subgroup\_id.

subgroup\_bbox\_size\_xyz[ 𝑘 ] specifies the 𝑘-th XYZ size component of the subgroup bounding box of the subgroup indicated by the pair of layer-group index layer\_group\_id and the subgroup index subgroup\_id.

ref\_layer\_group\_id specifies the indicator of the layer-group identifier of the context reference of the current dependent data unit. The range of the ref\_layer\_group\_id shall be in the range 0 .. num\_layer\_groups\_minus1.

ref\_subgroup\_id specifies the indicator of the subgroup identifier of the context reference of the current dependent data unit. When not present, ref\_subgroup\_id is inferred to be 0.

The reference context state is identified by the pair of layer-group index ref\_layer\_group\_id and subgroup index ref\_subgroup\_id.

subgroup\_context\_reference\_indication\_enabled equals to 1 indicates that the context state of the current data unit will be used to initialize one or more subsequent data units. subgroup\_context\_reference\_indication\_enabled equals to 0 indicates that the context state of the current data unit will not be used to initialize the subsequent data units. When not present, subgroup\_context\_reference\_indication\_enabled is inferred to be 1.

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

* + - 1. FGS attribute data unit
         1. FGS attribute data unit semantics

An ADU codes attribute values for a single attribute in a slice or a FGS. ADU is described in Subclause 7.4.4.1.

* + - * 1. FGS attribute data unit parameter semantics

subgroup\_weight\_adjustment\_enabledequals to 1 indicates that the subgroup weight adjustment coefficients subgroup\_weight\_adj\_coeff\_a and subgroup\_weight\_adj\_coeff\_b are present for the current subgroup corresponding to the current FGS. subgroup\_weight\_adjustment\_enabled equals to 0 indicates that the subgroup weight adjustment coefficients are not present and subgroup\_weight\_adj\_coeff\_a and subgroup\_weight\_adj\_coeff\_b are inferred to be 0.

subgroup\_weight\_adj\_coeff\_a and subgroup\_weight\_adj\_coeff\_b indicate the coefficient of subgroup weight adjustment.

* + - * 1. FGS dependent attribute data unit semantics

A DADU codes attribute values for a single attribute in a FGS. It comprises an DADU header and either attribute coefficients (attribute\_coeffs) when transform coding is equal to 1 or 2.

* + - * 1. FGS dependent attribute data unit header semantics

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

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

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

AttrIdx = dadu\_sps\_attr\_idx

The attribute coded by the DADU shall have at most three components.

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

dadu\_layer\_group\_idspecifies the indicator of a layer-group in the layer-group structure related to the slice. The range of dadu\_layer\_group\_id shall be in the range 0 .. num\_layer\_groups\_minus1. When not present, dadu\_layer\_group\_id shall be inferred to be 0.

dadu\_subgroup\_id specifies the indicator of the subgroup in the layer-group referred by dadu\_layer\_group\_id. The range of dadu\_subgroup\_id shall be in the range 0 .. num\_subgroups\_minus1[dadu\_layer\_group\_id] where the dadu\_subgroup\_id represent a partial region in a layer-group specified by dadu\_layer\_group\_id. When not present, dadu\_subgroup\_id shall be inferred to be 0.

attr\_ref\_layer\_group\_id specifies the indicator of the layer-group identifier of the context reference of the current DADU. The range of the attr\_ref\_layer\_group\_id shall be in the range of 0 to the dadu\_layer\_group\_id of the current DADU. When not present, attr\_ref\_layer\_group\_id is inferred to be ref\_layer\_group\_id of the geometry FGS whose layer\_group\_id and subgroup\_id in DGDU are identical to dadu\_layer\_group\_id and dadu\_subgroup\_id of the current attribute FGS.

attr\_ref\_subgroup\_id specifies the indicator of the reference subgroup of the layer-group indicated by attr\_ref\_layer\_group\_id. The range of the attr\_ref\_subgroup\_id shall be in the range of 0 to num\_subgroup\_id\_minus1 of the layer-group indicated by attr\_ref\_layer\_group\_id. When not present, attr\_ref\_subgroup\_id is inferred to be ref\_subgroup\_id of the geometry FGS whose layer\_group\_id and subgroup\_id in DGDU are identical to dadu\_layer\_group\_id and dadu\_subgroup\_id of the current attribute FGS.

attr\_subgroup\_context\_reference\_indication\_enabled equals to 1 indicates that the context state of the current DADU will be used to initialize one or more subsequent data units. attr\_subgroup\_context\_reference\_indication\_enabled equals to 0 indicates that the context state of the current attribute data unit will not be used to initialize the subsequent data units. When not present, attr\_subgroup\_context\_reference\_indication\_enabled is inferred to be 1.

[Ed. (HH): the semantics for the other syntaxes defined for DADUH is defined at the different subclauses.

The use of different syntax with the same semantics and vice versa is under discussion.]

* 1. Decoding process
     1. General

The decoding process is described in Clause 8. This annex specifies the difference in the decoding process for the fine granularity slice.

* + 1. Fine granularity slice decoding process
       1. General

When fgs\_layer\_group\_enabled equals to 1, a FGS in a coded point cloud frame shall be decoded as follows:

1. Point positions are decoded from one GDU and zero or more DGDUs as specified by E.4.2.3.
2. Default attribute values are set for each attribute as specified by 8.3.4.
3. Point attributes are decoded from one ADU and zero or more DADUs as specified by E.4.2.4. The ADU and DADUs shall be decoded after decoding of the GDU and DGDUs those are indicated by the identical pair of subgroup index layer\_group\_id and subgroup\_id.
4. The decoded point positions are offset and the output point count incremented as specified by 8.3.6.
   * + 1. State variables

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

* The variable startDepth, a start depth of tree level in a data unit
* The variable *endDepth*, a end depth of tree level in a data unit
  + - 1. FGS geometry decoding process

The GDU shall be decoded before all of the DGDUs in a slice. DGDUs of child subgroups shall be decoded after DGDU of the parent subgroup.

When decoding a GDU, startDepth and  endDepth are set to 0 and (num\_layers\_minus1[ 0] + 1), respectively.

When decoding a DGDU, startDepth is set to accumulated values of (num\_layers\_minus1[ 𝑘 ] + 1) where 𝑘 is in the range of 0 .. layer\_group\_id – 1.  endDepth is set to accumulated value of (num\_layers\_minus1[ 𝑘 ] + 1) where 𝑘 is in the range of 0 .. layer\_group\_id.

The expression SubgroupNodePos[ layerGroupIdx ][ subgroupIdx ][ nodeIdx ][ 𝑘 ] is an alias into the nodes at endDepth-th depth of a subgroup identified by layerGroupIdx and subgroupIdx, where layerGroupIdx is equal to layer\_group\_id and subgroupIdx is equal to subgroup\_id.

SubgroupNodePos[layerGroupIdx ][ subgroupIdx ][nodeIdx][k] := OccNodeLoc[ endDepth ][nodeIdx][k]

The expression SubgroupNodeCnt[ layerGroupIdx ][ subgroupIdx ] is an alias into the number of nodes at endDepth-th depth of a subgroup identified by layerGroupIdx and subgroupIdx.

SubgroupNodeCnt[layerGroupIdx ][ subgroupIdx ] := OccNodeCnt[endDepth]

The expression SubgroupBBoxMin[ layerGroupIdx ][ subgroupIdx ] and SubgroupBBoxMax [ layerGroupIdx ][ subgroupIdx ] are alias into the minimum and maximum point position of the bounding box of a subgroup identified by layerGroupIdx and subgroupIdx, respectively.

SubgroupBBoxMin[layerGroupIdx][subgroupIdx][k] := subgroup\_bbox\_origin[k]  
SubgroupBBoxMax[layerGroupIdx][subgroupIdx][k] := subgroup\_bbox\_origin[k] + subgroup\_bbox\_size[k]

The sparse array SubgroupOccNeighPatEq0[ CurrLayerGroupIdx ][ CurrSubgroupIdx ][ ns ][ nt ][ nv ] identifies whether the identified node of the parent subgroup has no nodes present in its occupied neighbourhood pattern.

The sparse array SubgroupOccNodeChildCnt[ CurrLayerGroupIdx ][ CurrSubgroupIdx ][ *k* ][ ns ][ nt ][ nv ] identifies the number of child node of the parent subgroup.

The expression SubgroupDirectNodePointCnt[ layerGroupIdx ][ subgroupIdx ] is an alias into the number of the points of the direct nodes at endDepth-th depth of a subgroup identified by layerGroupIdx and subgroupIdx.

SubgroupDirectNodePointCnt[layerGroupIdx ][ subgroupIdx ] :=

[Ed. (HH): complete the expression for SubgroupDirectNodePointCnt]

When layer\_group\_id is equal to num\_layer\_groups\_minus1, the position of nodes in the subgroup are copied to the output point cloud.

if (layer\_group\_id == num\_layer\_groups\_minus1) {  
 for (i = 0; i < SubgroupNodeCnt[layerGroupIdx ][ subgroupIdx ]; i++, PointCnt++)  
 for (k = 0; k < 3; k++)  
 PointPos[PointCnt][k] = SubgroupNodePos[layerGroupIdx ][ subgroupIdx ][i][k]  
}

Node positions shall be decoded and reconstructed as specified by E.5.

* + - 1. FGS attribute decoding process

The ADU shall be decoded before all of the DADUs in a slice. DADUs of child subgroups shall be decoded after DADU of the parent subgroup.

The ADU and DADU shall be decoded and the reconstructed attribute values stored in the corresponding output point cloud attribute or the leaf nodes of the occupancy tree in the subgroup.

When decoding a ADU, startDepth is set to 0.

When decoding a DADU, startDepth is set to accumulated values of (num\_layers\_minus1[ 𝑘 ] + 1) where 𝑘 is in the range of 0 .. layer\_group\_id – 1.

DirectNodePointCntis set to the number of coded points in direct nodes in each subgroup.

DirectNodePointCnt := SubgroupDirectNodePointCnt[layerGroupIdx][subgroupIdx]

The expression SubgroupNodeAttr[ layerGroupIdx ][ subgroupIdx ][ ptIdx ][ c ] is an alias into the output point cloud attribute array for the points in a subgroup identified by layerGroupIdx and subgroupIdx, where layerGroupIdx is equal to layer\_group\_id and subgroupIdx is equal to subgroup\_id.

SubgroupNodeAttr[layerGroupIdx][subgroupIdx][ptIdx][c] :=

[Ed. (HH): complete the expression for SubgroupNodeAttr]

When layer\_group\_id is equal to num\_layer\_groups\_minus1, the expression PointAttr[ ptIdx ][ 𝑐 ] is an alias into the output point cloud attribute array for the points in the FGS.

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

Otherwise, the expression PointAttr[ ptIdx ][ 𝑐 ] is an alias into the decoded points or the leaf nodes of the occupancy tree in the subgroup.

If ptIdx < DirectNodePointCnt:

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

Otherwise :

PointAttr[ptIdx][c] := SubgroupNodeAttr[layerGroupIdx][subgroupIdx][ptIdx- DirectNodePointCnt][c]

Point attributes shall be decoded and reconstructed as specified by E.6.

* 1. Fine granularity slice geometry
     1. General

Slice geometry shall be parsed and reconstructed point position from a coded occupancy tree in accordance with Clause 9. This annex specifies the difference of the fine granularity slice geometry decoder for parsing and reconstruction of point position.

When fgs\_layer\_group\_enabled equals to 1, the reconstructed geometry of each subgroup is stored in the arrays SubgroupNodePos (E.4.2.3).

* + 1. Partial tree of occupancy tree
       1. General

When geom\_tree\_type is 0, slice geometry shall be parsed and reconstructed point position from a coded occupancy tree in accordance with Subclause 9.2.

When fgs\_layer\_group\_enabled equals to 1, a partial occupancy tree represents the fine granularity slice geometry as a partial tree of occupancy tree nodes in terms of spatial region and depth. Fine granularity slice geometry shall be parsed or reconstructed point position from a coded partial occupancy tree as specified by E.5.2.

* + - 1. Coded occupancy tree
         1. General tree structure

An occupancy tree node shall identify the presence of at least one point contained within the volume of an axis-aligned cuboid.

When fgs\_layer\_group\_enabled equals to 1, an occupancy tree node shall identify the presence of at least one point contained within the volume of a cube. The volume is defined in the slice's coordinate system by an inclusive lower corner and an exclusive upper corner . The volume edge lengths are non-negative integer powers of two. A node's size, nodeSize, is synonymous with the volume dimensions .

* + - * 1. Tree traverse order

The coded occupancy tree shall be traversed in breadth-first order. Traversal shall start from the top tree level. All nodes in a tree level shall be sequentially traversed before proceeding to the next level. Within a tree level, nodes shall be traversed in ascending Morton order of node location.

When fgs\_layer\_group\_enabled equals to 1, the coded partial occupancy tree shall be traversed in breadth-first order in a subgroup. Traversal shall start from the minimum depth of the subgroup. All nodes in a tree level of the subgroup shall be sequentially traversed before proceeding to the next level. Within a tree level of the subgroup, nodes shall be traversed in ascending Morton order of node location.

* + - 1. State representation
         1. State variables

Traversal of the occupancy tree is specified in terms of the following state variables:

* The array OccNodeCnt; OccNodeCnt[ dpth ] is the cumulative count of nodes present at depth dpth. When fgs\_layer\_group\_enabled equals to 1, OccNodeCnt[ dpth ] represents the cumulative count of nodes present at depth dpth of a subgroup.
* The array OccNodeLoc; OccNodeLoc[ dpth ][ nodeIdx ][ 𝑘 ] identifies the 𝑘-th location component of the nodeIdx-th coded node in the traversal order of the tree level at depth dpth. When fgs\_layer\_group\_enabled equals to 1, OccNodeLoc[ dpth ][ nodeIdx ][ 𝑘 ] identifies the 𝑘-th location component of the nodeIdx-th coded node in the traversal order of the tree level at depth dpth of a subgroup.
  + - 1. Initialization

At the start of the occupancy tree syntax structure in GDU and DGDU, all elements of OccNodePresent, OccNodeLoc and OccNodeCnt are cleared.

When layer\_group\_id equals to 0, the arrays OccNodePresent, OccNodeLoc and OccNodeCnt are initialized by 9.2.5.2.

When layer\_group\_id is greater than 0, the arrays OccNodePresent, OccNodeLoc and OccNodeCnt are initialized by E.5.2.4.1 and E.5.2.4.2 to decode the occupancy tree continuous to the parent subgroup.

* + - * 1. Parent subgroup detection

The layer-group index PrtLayerGroupIdx and the subgroup index PrtSubgroupIdx of a parent subgroup are inferred by using the parent-child subgroup relationship of SubgroupBBoxMin and SubgroupBBoxMax as follows.

CurrLayerGroupIdx := layer\_group\_id  
PrtLayerGroupIdx   
:= CurrLayerGroupIdx – 1  
CurrSubgroupIdx := subgroup\_id  
  
for(i=0; i<numSubgroups[PrtLayerGroupIdx]; i++)   
 if((SubgroupBBoxMin[PrtLayerGroupIdx][i][0] ≤  
 SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][0] &&  
 SubgroupBBoxMin[PrtLayerGroupIdx][i][1] ≤  
 SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][1] &&   
 SubgroupBBoxMin[PrtLayerGroupIdx][i][2] ≤  
 SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][2]) &&   
 (SubgroupBBoxMax[PrtLayerGroupIdx][i][0] >  
 SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][0]  
 SubgroupBBoxMax[PrtLayerGroupIdx][i][1] >  
 SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][1]  
 SubgroupBBoxMax[PrtLayerGroupIdx][i][2] >  
 SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][2])) {  
 PrtSubgroupIdx = i  
 break  
}

[Ed. (HH): make shorter variable names]

* + - * 1. Inheritance of parent subgroup output nodes

Given parent subgroup index and the bounding box information of decoded subgroups, OccNodeLoc[ startDepth ] is set as a subset of the parent subgroup output nodes SubgroupNodePos by selecting nodes in a region within the bounding box of the current subgroup depicted by SubgroupBBoxMin[ CurrLayerGroupIdx ][ CurrSubgroupIdx ] and SubgroupBBoxMax[ CurrLayerGroupIdx ][ CurrSubgroupIdx ].

count = 0  
numMissingLayers = occtree\_depth\_minus1 + 1 - endDepth  
for(i = 0; i < SubgroupNodeCnt[PrtLayerGroupIdx][PrtSubgroupIdx]; i++) {  
 for(k = 0; k < 3; k++)  
 nodePos[k] = SubgroupNodePos[ layerGroupIdx ][ subgroupIdx ][i][k] << numMissingLayers  
  
 if ((nodePos[0] ≥ SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][0] &&  
 nodePos[1] ≥ SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][1] &&  
 nodePos[2] ≥ SubgroupBBoxMin[CurrLayerGroupIdx][CurrSubgroupIdx][2]) &&  
 (nodePos[0] < SubgroupBBoxMax[CurrLayerGroupIdx][CurrSubgroupIdx][0] &&  
 nodePos[1] < SubgroupBBoxMax[CurrLayerGroupIdx][CurrSubgroupIdx][1] &&  
 nodePos[2] < SubgroupBBoxMax[CurrLayerGroupIdx][CurrSubgroupIdx][2])) {  
 for(k = 0; k < 3; k++)  
 OccNodeLoc[ startDepth ][count][k] = SubgroupNodePos[ PrtLayerGroupIdx  ][  PrtSubgroupIdx  ][i][k]   
 count++   
 }  
}

[Ed. (HH): make shorter variable names]

OccNodeCnt[ startDepth ] is set as the number of nodes in OccNodeLoc[ startDepth ].

OccNodeCnt[ startDepth ] = count

OccNodePresent[ startDepth ][ ns ][ nt ][ nv ] is set to 1 at a tree location ( ns, nt, nv ) equal to OccNodeLoc[ startDepth ][ nodeIdx ].

for(nodeIdx = 0; nodeIdx < OccNodeCnt[ startDepth ]; nodeIdx ++) {  
 ns = OccNodeLoc[startDepth][nodeIdx][0]  
 nt = OccNodeLoc[startDepth][nodeIdx][1]  
 nv = OccNodeLoc[startDepth][nodeIdx][2]  
 OccNodePresent[startDepth][ns][nt][nv] = 1  
}

* + - 1. Dictionary coding of occupancy\_byte
         1. Initial state

The dictionary state shall be initialized at the start of every GDU.

At the start of every DGDU, the dictionary state shall be initialized by the parsing state restoration process (11.5.3.2).

* + - 1. Bitwise occupancy coding
         1. Contextualization

Initial state

The demi-CPMs shall be initialized at the start of every GDU.

At the start of every DGDU, the demi-CPMs shall be initialized according to the parsing state restoration process (E.7.3.1.2).

* + - 1. Planar occupancy coding
         1. Per-axis eligibility

Condition

The expression *PointDensity*[*dpth* ] is a factor that identifies the density of the points in the tree level at depth *dpth*:

PointDensity[dpth ]:= (slice\_num\_points\_minus1 + 1 – DirectNodePointCnt) × 10 / OccNodeCnt[ dpth ]

When fgs\_layer\_group\_enabled equals to 1, *PointDensity*[*dpth* ] is specified by subgroup\_planar\_eligibility\_by\_density in the GDUH or DGDUH.

PointDensity[dpth] = subgroup\_planar\_eligibility\_by\_density[dpth -  startDepth ]

* + - 1. Direct nodes
         1. General

The number of points coded in direct nodes is counted cumulatively, *DirectNodePointCnt*. The variable is initialized to 0 at the start of fine granularity slice.

* + - * 1. Eligibility

Initial state

At the start of every occupancy\_tree syntax structure, the OccNodeChildCnt array shall be cleared; all elements of OccNodeChildCnt are unset.

When fgs\_layer\_group\_enabled equals to 1, at the start of every occupancy\_tree syntax structure, the initial states of OccNeighPatEq0 and OccNodeChildCntare set as follows.

if(layer\_group\_enabled && dpth == startDepth){  
 OccNeighPatEq0[ Dpth-1 ]  
 = SubgroupOccNeighPatEq0[ PrtLayerGroupIdx ][ PrtSubgroupIdx ]  
 OccNodeChildCnt[ Dpth-1 ]   
 = SubgroupOccNodeChildCnt[ PrtLayerGroupIdx ][ PrtSubgroupIdx ][0]   
 OccNodeChildCnt[ Dpth-2 ]  
 = SubgroupOccNodeChildCnt[ PrtLayerGroupIdx ][ PrtSubgroupIdx ][1]   
}

State update after each coded occupancy tree node

This subclause applies at the end of every occupancy\_tree\_node syntax structure.

The number of child nodes and the presence of any nodes in the occupied neighbourhood pattern are recorded for use in subsequent eligibility decisions.

OccNodeChildCnt[Dpth][Ns][Nt][Nv] = direct\_node ? 0 : OccChildCnt  
OccNeighPatEq0[Dpth][Ns][Nt][Nv] = OccNeighPat == 0

If the node is eligible for direct coding, irrespective of the presence of occ\_direct\_node, the count of eligible nodes shall be incremented.

if (DirectModeEligible)  
 DnEligibleCnt++

When fgs\_layer\_group\_enabled equals to 1 and dpth is equal to endDepth, the SubgroupOccNeighPatEq0and SubgroupOccNodeChildCnt are set as follows.

if(layer\_group\_enabled && dpth == endDepth){  
 SubgroupOccNeighPatEq0[ LayerGroupIdx ][ SubgroupIdx ][ Ns ][ Nt ][ Nv ]  
 = OccNeighPatEq0[ Dpth ][ Ns ][ Nt ][ Nv ]   
 SubgroupOccNodeChildCnt[ LayerGroupIdx ][SubgroupIdx ][0][ Ns ][ Nt ][ Nv ]   
 = OccNodeChildCnt[ Dpth ][ Ns ][ Nt ][ Nv ]  
 SubgroupOccNodeChildCnt[ LayerGroupIdx ][ SubgroupIdx ][1][ NsP ][ NtP ][ NvP ]  
 = OccNodeChildCnt[ Dpth-1 ][ NsP ][ NtP ][ NvP ]  
}

* 1. Fine granularity slice attributes
     1. General

Slice attributes shall be reconstructed attributes from a coded slice geometry in accordance with Clause 10. E.6 specifies the reconstruction of a FGS attribute for the coded FGS geometry. This annex specifies the difference of the fine granularity slice attribute decoder.

When fgs\_layer\_group\_enabled equals to 1, the reconstructed attributes of each subgroup is stored in the arrays SubgroupNodeAttr (E.4.2.4).

* + 1. Point coordinates
       1. General

When fgs\_layer\_group\_enabled is 1, attr\_coord\_conv\_enabled is 0 and attr\_inter\_prediction\_enabled is 0. Attribute coding can use the FGS geometry's reconstructed STV point positions.

The expression AttrPos[ ptIdx ][ 𝑘 ] specifies the coordinates of each point for attribute coding. AttrPos is equivalent to SubgroupNodePos, which is the FGS geometry in the slice’s coordinate system.

AttrPos[ptIdx][k] := SubgroupNodePos[ptIdx][k]

* + 1. Attribute decoding using levels of detail
       1. General

When attr\_coding\_type is either 1 or 2, slice attributes shall be reconstructed the attributes from a coded geometry in accordance with Subclause 10.6. The different or additional processes for the fine granularity slice’s attribute decoder are specified in E.6.3.

The attribute decoding processes specified by Subclause 10.6 and E.6.3 are distance-based prediction schemes that use a hierarchical level-of-detail representation of the slice geometry.

Detail levels are defined by an iterative subsampling process (10.6.5). The finest detail level comprises all points in the slice geometry or the FGS geometry. With each iteration, a coarser detail level is generated from the previous coarsest detail level.

Every detail level comprises a list of points present in the detail level, and is associated with a list of refinement points. A refinement point is a point that is present in a detail level and not present in any coarser detail level; the refinement points for detail level lvl, when combined with the coarser detail level lvl + 1, form detail level lvl.

For each refinement point, a set of neighbouring points is determined (10.6.6) using inter-detail-level, intra-detail-level and inter-frame and inter-layer-group searches. The neighbouring points form a predictor set that is used to predict attribute/transform coefficient values.

The index variable lvl identifies a detail level. The index lvl starts from 0. When fgs\_layer\_group\_enabled is equal to 1, the lvl value is initialized to LodMinLevel, which is set to log2 quantized node size of the leaf nodes of the FGS geometry.

* + - 1. State variables

Point predictors are specified in terms of the following state variables; the index ptIdx identifies a point by its index into AttrPos:

* The array PredPtIdx, identifies point predictors by their index in the canonical decoding order; PredPtIdx[ ptIdx ][ ni ] is the AttrPos index of the ni-th point in the predictor set for the identified point. When fgs\_layer\_group\_enabled equals to 1, PredPtIdx[ ptIdx ][ ni ] is the RefAttrPos index of the ni-th point in the predictor set for the identified point.
* The array PredPtRef of the flags to specify whether the point predictors are searched from the reference slice. When PredPtRef[ ptIdx ][ ni ] is equal to 1, the ni-th point in the predictor set is specified to be searched from the reference slice for the predictor identified by PredPtIdx[ ptIdx ][ ni ]; When PredPtRef[ ptIdx ][ ni ] is equal to 0, the ni-th point in the predictor set is specified to be searched from the current slice for the predictor identified by PredPtIdx[ ptIdx ][ ni ]. When fgs\_layer\_group\_enabled equals to 1 and PredPtRef[ ptIdx ][ ni ] is equal to 1, the ni-th point in the predictor set is specified to be searched from the parent FGS for the predictor identified by PredPtIdx[ ptIdx ][ ni ]; When fgs\_layer\_group\_enabled equals to 1 and PredPtRef[ ptIdx ][ ni ] is equal to 0, the ni-th point in the predictor set is specified to be searched from the current FGS for the predictor identified by PredPtIdx[ ptIdx ][ ni ].

When fgs\_layer\_group\_enabled is equal to 1 and layer\_group\_id is greater than 0, the finest detail level of the parent FGS is specified in the array ParentLodPtIdx, identifying points in the finest detail level by their index in the canonical decoding order; ParentLodPtIdx [ 𝑖 ] is the RefAttrPos index of the 𝑖-th point in the finest detail level.

The variable LodMinLevel identifies a minimum level of detail levels generated from the FGS geometry.

* + - 1. Levels of detail

When fgs\_layer\_group\_enabled is 1, general LoD generation process is specified by E.6.3.3.1.

* + - * 1. General generation process

When fgs\_layer\_group\_enabled is 1, the minimum level of LoD is specified in the variable LodMinLevel. The maximum level of the LoD is set to overlap with the minimum level of the parent FGS's LOD.

The finest detail level is identified by the detail level index LodMinLevel. LodMinLevel is set to log2 quantized node size of the leaf nodes of the FGS geometry.

Detail levels shall be iteratively subsampled (E.6.3.3.4), starting from the finest detail level, until either a single point remains or LodMaxLevel subsampled detail levels have been produced. The variable Lvl identifies the detail level to be subsampled. When layer\_group\_id is 0, LodMaxLevel is set to log2 quantized node size of the root nodes of the FGS geometry. Otherwise, LodMaxLevel is set to plus 1 for log2 quantized node size of the root nodes of the FGS geometry.

LodMinLevel = occtreeMaxDepthMinus1 – (startDepth-1)  
LodMaxLevel = LodMinLevel+num\_layers\_minus1[layer\_group\_id]+1   
if(layer\_group\_id>0)  
 LodMaxLevel++  
  
Lvl = LodMinLevel   
for (; Lvl < LodMaxLevel; Lvl++) {  
 if (LodPtCnt[Lvl] == 1)  
 break  
 … /\* subsample LodPtIdx[Lvl] \*/  
}  
LodCnt = Lvl + 1

The coarsest detail level is identified by the detail level index LodCnt − 1. All points in the coarsest detail level shall be assigned to the coarsest level's refinement list (10.6.5.3).

* + - * 1. The finest detail level

The AttrPos point indexes of the finest detail level shall have an initial one-to-one correspondence with the canonical decoding order of the FGS geometry.

subgroupPointCnt = SubgroupNodeCnt[layerGroupIdx][subgroupIdx] + SubgroupDirectNodePointCnt[layerGroupIdx][subgroupIdx]  
for (ptIdx = 0; ptIdx < subgroupPointCnt; ptIdx++)  
 LodPtIdx[LodMinLevel][ptIdx] = ptIdx  
LodPtCnt[LodMinLevel] = subgroupPointCnt

The point indexes of the finest detail level shall be sorted by group in ascending order of their respective Morton-coded attribute coordinates. The variable maxPtsPerSort identifies the max group size when sorting by group.

maxPtsPerSort = !attr\_canonical\_order\_enabled && !max\_points\_per\_sort\_log2\_plus1   
? LodPtCnt[LodMinLevel] : 1 << (max\_points\_per\_sort\_log2\_plus1 - 1)

The sorted order shall be identical for the decoding of all attributes in a FGS with identical attribute coordinate arrays (AttrPos).

Performing a stable sort for each attribute, or reusing the reordered points would satisfy the requirement for identical orders.

An example (inefficient) sorting process is:

for (benIdx = 0; benIdx < LodPtCnt[0]; benIdx += maxPtsPerSort) {  
 endIdx = Min(benIdx + maxPtsPerSort, LodPtCnt[LodMinLevel]);  
 for (i = benIdx; i < endIdx; i++)  
 for (j = i + 1; j < endIdx; j++) {  
 iPtIdx = LodPtIdx[LodMinLevel][i]  
 jPtIdx = LodPtIdx[LodMinLevel][j]  
 iMorton = Morton(AttrPos[iPtIdx][0], AttrPos[iPtIdx][1], AttrPos[iPtIdx][2])  
 jMorton = Morton(AttrPos[jPtIdx][0], AttrPos[jPtIdx][1], AttrPos[jPtIdx][2])  
 if (iMorton > jMorton)  
 Swap(LodPtIdx[LodMinLevel][i], LodPtIdx[LodMinLevel][j])  
 }  
}

* + - * 1. The finest detail level of the parent FGS

The expression *RefAttrPos* [*ptIdx*][*k*] specifies the coordinates of each point for attribute coding in the parent FGS. The RefAttrPos point indexes of the finest detail level shall have an initial one-to-one correspondence with the canonical decoding order of the parent FGS geometry. The variable refPointCnt, the size of the number of points in the parent FGS.

for (ptIdx = 0; ptIdx < refPointCnt; ptIdx++)  
 LodPtIdx[ptIdx] = ptIdx  
parentLodPtCnt = refPointCnt

The point indexes of the finest detail level shall be sorted by group in ascending order of their respective Morton-coded attribute coordinates. The sorted order shall be identical for the decoding of all attributes in a single FGS with identical attribute coordinate arrays (RefAttrPos).

Performing a stable sort for each attribute, or reusing the reordered points would satisfy the requirement for identical orders.

An example (inefficient) sorting process is:

for (benIdx = 0; benIdx < refPointCnt; benIdx += maxPtsPerSort) {  
 endIdx = Min(benIdx + maxPtsPerSort, refPointCnt);  
 for (i = benIdx; i < endIdx; i++)  
 for (j = i + 1; j < endIdx; j++) {  
 iPtIdx = ParentLodPtIdx[i]  
 jPtIdx = ParentLodPtIdx[j]  
 iMorton = Morton(RefAttrPos[iPtIdx][0], RefAttrPos[iPtIdx][1],   
 RefAttrPos[iPtIdx][2])  
 jMorton = Morton(RefAttrPos[jPtIdx][0], RefAttrPos[jPtIdx][1],   
 RefAttrPos[jPtIdx][2])  
 if (iMorton > jMorton)  
 Swap(ParentLodPtIdx[i], ParentLodPtIdx[j])  
 }  
}

* + - * 1. Generation of a single detail level

The coarser detail level Lvl + 1 shall be produced by subsampling the points of detail level Lvl.

When fgs\_layer\_group\_enabled is equal to 1, block-based subsampling (10.6.5.8) shall proceed.

* + - 1. Predictor search
         1. General process

The index variable lvl value is initialized to LodMinLevel.

maxLvl = LodCnt − (attr\_coding\_type == 2)  
for (Lvl = LodMinLevel; Lvl < maxLvl; Lvl++)  
 for (RfmtIdx = 0; RfmtIdx < LodRfmtPtCnt[Lvl]; RfmtIdx++) {  
 … /\* find predictors (10.6.6.2) of the current point \*/  
 }

* + - * 1. Minimum reference detail level for inter-level predictor searches

MinInterRefLvl = LodMinLevel+1  
if (lod\_scalability\_enabled) {  
 for (lvl = LodMinLevel+1; lvl < LodCnt − 1; lvl++) {  
 if (LodRfmtPtCnt[lvl] < slice\_num\_points\_minus1 − LodPtCnt[lvl])  
 break  
 MinInterRefLvl++  
 }  
}

* + - * 1. Predictor search for a single refinement point

When fgs\_layer\_group\_enabled is equal to 1 and layer\_group\_id is greater than or equal to 1, only an inter-layer-group search (E.6.3.4.4) shall be performed for the coarsest detail level.

* + - * 1. Inter-layer-group predictor search

The inter-layer-group search shall be performed by finding the parent point which a Morton code matches.

candPtIdx = ParentLodPtIdx[RfmtIdx]  
PredCnt[PtIdx] = 1  
PredPtIdx[PtIdx][i] = candPtIdx  
PredPtRef[PtIdx][i] = 1

* + - 1. Reconstruction of attribute values
         1. General process

Each detail level shall be processed in turn, proceeding from the coarsest to the finest level, according to attr\_coding\_type (10.6.7.3, 10.6.7.4). The variable Lvl is the index of the current detail level.

for (Lvl = LodCnt − 1; Lvl ≥ LodMinLevel; Lvl−−)  
 … /\* process a detail level \*/

* + - * 1. Processing of a detail level (attr\_coding\_type = 2)

When Lvl is LodMinLevel, the reconstructed attributes values shall be divided by 256 with half-values rounded away from zero and clipped to the maximum attribute value:

if (Lvl == LodMinLevel) {  
subgroupPointCnt = SubgroupNodeCnt[layerGroupIdx][subgroupIdx] + SubgroupDirectNodePointCnt[layerGroupIdx][subgroupIdx]   
for (ptIdx = 0; ptIdx < subgroupPointCnt; ptIdx++)  
 for (c = 0; c < AttrDim; c++)  
 PointAttr[ptIdx][c] = Clip3(0, AttrMaxVal, DivExp2Fz(PointAttr[ptIdx][c], 8))  
}

* + - 1. Transform coefficient weights
         1. Fine granularity slice case

When fgs\_layer\_group\_enabled is equal to 1, coefficient weights are calculated accumulatively, proceeding from the finest to the coarsest detail level of a FGS.

The accumulated coefficient weight of each refinement point in a detail level shall be distributed to the points in its predictor set. The distribution is proportional to the respective predictor weights:

for (lvl = LodMinLevel; lvl < LodCnt − 1; lvl++)  
 for (rfmtIdx = 0; rfmtIdx < LodRfmtPtCnt[lvl]; rfmtIdx++) {  
 ptIdx = LodRfmtPtIdx[lvl][rfmtIdx]  
 coeffW = CoeffWeight[ptIdx]  
 for (ni = 0; ni < PredCnt[ptIdx]; ni++) {  
 if (!PredPtRef[ptIdx][ni]){  
 predW = PredWeight[ptIdx][ni]  
 CoeffWeight[PredPtIdx[ptIdx][ni]] += DivExp2Up(coeffW × predW, 8)  
 }  
 }  
 }

* + - 1. Transform coefficient weights
         1. Quantization weights derivation

The index variable lvl value is initialized to LodMinLevel.

for (lvl = LodMinLevel; lvl < LodCnt − 1; lvl++)  
 for (rfmtIdx = 0; rfmtIdx < LodRfmtPtCnt[lvl]; rfmtIdx++) {  
 ptIdx = LodRfmtPtIdx[lvl][rfmtIdx]  
 coeffW = QuantWeight[ptIdx]  
 for (ni = 0; ni < PredCnt[ptIdx]; ni++) {  
 if (!PredPtRef[ptIdx][ni]){  
 QuantWeight[PredPtIdx[ptIdx][ni]] +=  
 DivExp2Inf(coeffW × quant\_neigh\_weight[ni], 8)  
 }  
 }  
 }

When fgs\_layer\_group\_enabled equals to 1 and subgroup\_weight\_adjustment\_enabled equals to 1, quantization weights of a subgroup is adjusted to alleviate the missing information out of the layer-group boundary.

maxRefNodes = 0;  
for (lvl = LodMinLevel; lvl < LodCnt − 1; lvl++)  
 for (rfmtIdx = 0; rfmtIdx < LodRfmtPtCnt[lvl]; rfmtIdx++) {  
 ptIdx = LodRfmtPtIdx[lvl][rfmtIdx]   
 for (ni = 0; ni < PredCnt[ptIdx]; ni++) {  
 if (!PredPtRef[ptIdx][ni]){  
 numRefNodes[PredPtIdx[ptIdx][ni]]++;  
 if(maxRefNodes < numRefNodes[PredPtIdx[ptIdx][ni]])  
 maxRefNodes = numRefNodes[PredPtIdx[ptIdx][ni]]  
 }  
 }  
 }  
  
numPointsInSubgroup = SubgroupNodeCnt[dadu\_layer\_group\_id ][ dadu\_subgroup\_id  ] + SubgroupDirectNodePointCnt[dadu\_layer\_group\_id ][ dadu\_subgroup\_id  ]  
for(i = 0; i < numPointsInSubgroup; i++){  
 QuantWeight[i] += (numRefNodes[i]\* subgroup\_weight\_adj\_coeff\_a)/maxRefNodes + subgroup\_weight\_adj\_coeff\_b  
}

[Ed. (HH): The definition of numRefNodes is missing.]

* + 1. Parent FGS generation for attributes
       1. General

When fgs\_layer\_group\_enabled is equal to 1 and layer\_group\_id is greater than 1, E.6.4.2 shall be applied.

After the point coordinates generation of the current FGS as specified by E.6.2, the parent FGS generation is proceeded.

The output of this process are the parent FGS attribute and the parent FGS geometry for attribute coding. The expression *RefPointAttr* [*ptIdx*][*c*] specifies the attribute values of each point in the parent FGS. The expression *RefAttrPos* [*ptIdx*][*k*] specifies the coordinates of each point for attribute coding in the parent FGS.

* + - 1. Parent FGS generation

The attribute values and attribute coordinates of the parent FGS are derived based on the bounding box of the coordinates of the current FGS. The number of points in the parent FGS is specifies by the variable *refPointCnt*.

refPointCnt = 0  
nodeSizeLog2OfParent = occtreeMaxDepthMinus1 - startDepth  
parentNodeCnt := SubgroupNodeCnt[layerGroupIdx-1][prtSubgroupIdx]   
parentNodePos[i][k] := SubgroupNodePos[layerGroupIdx-1][prtSubgroupIdx][i][k]  
parentNodeAttr[i][c] := SubgroupNodeAttr[layerGroupIdx-1][ prtSubgroupIdx][i][AttrIdx][c]  
for (i = 0; i < parentNodeCnt; i++){  
 parentNodePosShift = parentNodePos[i] << nodeSizeLog2OfParent  
if ((parentNodePosShift[0] < SubgroupBBoxMax[layerGroupIdx-1][prtSubgroupIdx][0] &&  
 parentNodePosShift[1] < SubgroupBBoxMax [layerGroupIdx-1][prtSubgroupIdx][1] &&  
 parentNodePosShift[2] < SubgroupBBoxMax [layerGroupIdx-1][prtSubgroupIdx][2]) &&   
 (parentNodePosShift[0] >= SubgroupBBoxMin[layerGroupIdx-1][prtSubgroupIdx][0] &&   
 parentNodePosShift[1] >= SubgroupBBoxMin [layerGroupIdx-1][prtSubgroupIdx][1] &&  
 parentNodePosShift[2] >= SubgroupBBoxMin[layerGroupIdx-1][prtSubgroupIdx][2])) {  
 for (c = 0; c < AttrDim; c++)  
 RefPointAttr [refPointCnt][c] = parentNodeAttr[i][c]  
 for (k = 0; k < 3; k++)  
 RefAttrPos [refPointCnt][k] = parentNodePos[i][k]  
 refPointCnt++  
}

* 1. Parsing process
     1. General

The parsing process of syntax elements is described in Clause 11. This annex specifies additional processes for the fine granularity slice.

* + 1. Data unit buffer
       1. Initialization at the start of parsing a geometry data unit footer

The parsing of a geometry\_data\_unit\_footer syntax structure shall commence at an offset from the end of the DU buffer. The length of the GDU footer is specified by the expression DuFooterLen. The expression DuIsGdu is equal to 1 when the DU is a GDU or DGDU.

GduFooterLen := 3 × (1 + occtree\_point\_cnt\_list\_present × occtreeMaxDepthMinus1)  
  
DuFooterLen := DuIsGdu ? GduFooterLen : 0  
DataUnitReadIdx = 8 × (DataUnitLength − DuFooterLen)

* + 1. CABAC parsing processes
       1. Contextual probability models
          1. General

A CPM is a 16-bit unsigned integer value that models the probability of a zero bin.

The array Contexts, with elements Contexts[ ctxTbl ][ ctxIdx ], represents individual adaptive CPMs used by the CABAC parsing process.

When fgs\_layer\_group\_enabled equals to 1, the array SubgroupContexts and SubgroupContextsForAttributes are used.

The array SubgroupContexts[ layerGroupIdx ][ subgroupIdx ] represents the array of Contexts of a subgroup indicated by the layer-group index layerGroupIdx and the subgroup index subgroupIdx.

The arraySubgroupContextsForAttributes[ AttrIdx ][ layerGroupIdx ] [ subgroupIdx ] represents the array of Contexts of a subgroup attributes indicated by the attribute index AttrIdx, and the layer-group index layerGroupIdx and the subgroup index subgroupIdx.

* + - * 1. Initialization

When fgs\_layer\_group\_enabled equals to 1, initialization shall be performed by the parsing state restoration process (E.7.4).

* + 1. Parsing state memorization and restoration
       1. General

E.7.3 applies when fgs\_layer\_group\_enabled is 1.

At certain moments, the entropy parsing state is recorded and later, used as the initial state for parsing other DUs or occupancy tree entropy streams.

The entropy parsing state shall comprise:

* for a GDU or a DGDU, the CABAC CPMs (E.7.3.1), the demi-CPMs for bitwise occupancy coding (9.2.10.6), the dictionary codec state for bytewise occupancy coding (9.2.9.4), the planar occupancy coding state (9.2.11.5.2) and the state of the variables PrevInterFrameRefIdx, PrevPhiResidSign, PrevPhiMul and PrevRadiusResidSign (9.3.3.1);
* for an ADU or DADU, the CABAC CPMs only (E.7.3.1).

The entropy parsing state shall be recorded and restored independently according to DU type (ADU and DADU versus GDU and DGDU) and for each different value of ADU AttrIdx. For example, a coded point cloud sequence with num\_attributes equal to 2 would require storage for three sets of entropy parsing state.

At the start of any GDU with slice\_entropy\_continuation equal to 0 or inter\_entropy\_continuation\_enabled equal to 0, all previously recorded GDU, ADU, DGDU and DADU entropy parsing state shall be discarded.

* + - 1. Geometry data units
         1. Memorization

The GDU or DGDU entropy parsing state shall be recorded at:

* the end of every dependent\_geometry\_data\_unit syntax structure (E.3.1.3.4) when subgroup\_context\_reference\_indication\_enabled equals to 1 and layer\_group\_id is less than num\_layer\_groups\_minus1.

When fgs\_layer\_group\_enabled equals to 1, memorization shall record the elements and values of the GDU or DGDU entropy parsing state for restoration by the restoration process (E.7.4.2.2). When subgroup\_context\_reference\_indication\_enabled equals to 1 and layer\_group\_id is less than num\_layer\_groups\_minus1, the state shall be recorded to the subgroup array context SubgroupContexts[ layerGroupIdx ][ subgroupIdx ] of layer-group index layerGroupIdx equals to layer\_group\_id and subgroup index subgroupIdx equals to subgroup\_id.

SubgroupContexts[ layer\_group\_id ][ subgroup\_id ] = Contexts

A decoder may release SubgroupContexts[ layerGroupIdx ][ subgroupIdx ] during decoding process of fine granularity slices. For the context buffer release, SubgroupContextsCounter[ layerGroupIdx ][ subgroupIdx ] may defined as an integer counter of SubgroupContexts[ layerGroupIdx ][ subgroupIdx ]. SubgroupContextsCounter is initialized by the number of subsequent subgroups referencing to the context state of the current subgroup. When GDU or DGDU entropy parsing state identified by the subgroup context reference identifiers ref\_layer\_group\_id and ref\_subgroup\_id is restored, SubgroupContextsCounter[ layerGroupIdx ][ subgroupIdx ] is decreased by 1. When the integer counter SubgroupContextsCounter[ layerGroupIdx ][ subgroupIdx ] is equal to 0, the corresponding context state SubgroupContexts[ layerGroupIdx ][ subgroupIdx ] may be released from the context buffer.

* + - * 1. Restoration

The GDU entropy parsing state shall be restored at:

* the start of a geometry\_data\_unit syntax structure (7.3.3.1) when slice\_entropy\_continuation is 1; and
* the start of a geometry\_data\_unit syntax structure (7.3.3.1) when slice\_entropy\_continuation is 0, slice\_inter\_prediction is 1 and inter\_entropy\_continuation\_enabled is 1; and
* the start of every occupancy\_tree\_level( dpth ) syntax structure (7.3.3.5) where dpth is greater than OcctreeEntropyStreamDepth.

The GDU or DGDU entropy parsing state shall be restored at:

* the start of a dependent\_geometry\_data\_unit syntax structure (E.3.1.3.4).

When fgs\_layer\_group\_enabled equals to 1, restoration shall restore the elements and values of the GDU entropy parsing state to those previously recorded by the memorization process (E.7.4.2.1). At the start of a geometry\_data\_unit syntax structure, restoration shall exclude the planar occupancy coding state.

Restoration shall restore the elements and values of the DGDU entropy parsing state to those previously recorded by the memorization process (E.7.4.2.1) of the layer-group index ref\_layer\_group\_id and subgroup index ref\_subgroup\_id.

* + - 1. Attribute data units
         1. Memorization

The ADU entropy parsing state shall be recorded at :

* the end of every attribute\_data\_unit syntax structure (7.3.4.1).

When fgs\_layer\_group\_enabled is 1, the DADU entropy parsing state shall be recorded at :

* the end of every dependent\_attribute\_data\_unit syntax structure (E.3.1.4.2) when attr\_subgroup\_context\_reference\_indication\_enabled equals to 1 and dadu\_layer\_group\_id is less than num\_layer\_groups\_minus1.

Memorization shall record the elements and values of the ADU entropy parsing state for restoration by the restoration process (11.6.3.2). The state shall be recorded separately for each value of AttrIdx. When fgs\_layer\_group\_enabled is 1, memorization shall record the elements and values of the DADU entropy parsing state for restoration by the restoration process (E.7.4.3.2). The state shall be recorded separately for each value of AttrIdx, dadu\_layer\_group\_id and dadu\_subgroup\_id.

When attr\_subgroup\_context\_reference\_indication\_enabled equals to 1 and dadu\_layer\_group\_id is less than num\_layer\_groups\_minus1, the state shall be recorded to the subgroup array context for attribute SubgroupContextsForAttributes[ AttrIdx ][ layerGroupIdx ][ subgroupIdx ] of layer-group index layerGroupIdx equals to dadu\_layer\_group\_id and subgroup index subgroupIdx equals to dadu\_subgroup\_id.

SubgroupContextsForAttributes[AttrIdx][ dadu\_layer\_group\_id ][ dadu\_subgroup\_id ] = Contexts

* + - * 1. Restoration

The ADU entropy parsing state shall be restored at :

* the start of each attribute\_data\_unit syntax structure (7.3.4.1) when either slice\_entropy\_continuation is 1 or slice\_inter\_entropy\_continuation is 1. The restoration shall be from the state recorded by the memorization process (11.6.3.1) with the same value of AttrIdx.

When fgs\_layer\_group\_enabled is 1, the DADU entropy parsing state shall be restored at :

* the start of each dependent\_attribute\_data\_unit syntax structure (E.3.1.4.2). The restoration shall be from the state recorded by the memorization process (E.7.4.3.1) which is indicated by AttrIdx, ref\_layer\_group\_id and ref\_subgroup\_id.

When fgs\_layer\_group\_enabled equals to 1, restoration shall restore the elements and values of the ADU or DADU entropy parsing state to those previously recorded by the memorization process (E.7.4.3.1) of the layer-group index attr\_ref\_layer\_group\_id, subgroup index attr\_ref\_subgroup\_id, and the same value of AttrIdx.

Contexts = SubgroupContextsForAttributes[AttrIdx][ attr\_ref\_layer\_group\_id ][attr\_ref\_ subgroup\_id ]

* 1. Partial decoding
     1. General

A decoder may decode and reconstruct part of fine granularity slices. A slice which is segmented by fine granularity slices supports partial decoding in terms of density (E.8.2) and/or spatial region (E.8.3). When partial decoding is used, the unnecessary data units to produce the partial output shall be filtered before decoding occupancy tree or attribute coefficients.

* + 1. Partial density decoding
       1. General

A decoder shall generate a lower density slice point cloud.

The lower density FGS point cloud is specified in terms of the following variables:

* The variable SkippedLayerGroup, an application-specific number of skipped layer-groups for partial decoding in the direction of the density. The value of SkippedLayerGroup shall be in the range 0 .. num\_layer\_groups\_minus1.
* The variable MinNodeSizeLog2, a minimum occupancy tree node size that is specified by the SkippedLayerGroup.
* The arrays SubgroupNodePos[ layerGroupIdx ][ subgroupIdx ][ ptIdx ][ 𝑘 ], the subgroup output nodes of the layer-group index layerGroupIdx and the subgroup index subgroupIdx.
* The arrays SubgroupNodeCnt[ layerGroupIdx ][ subgroupIdx ], the number of nodes in the subgroup output nodes of the layer-group index layerGroupIdx and the subgroup index subgroupIdx
  + - 1. Selection of FGS

If SkippedLayerGroup is greater than 0, the layer-groups whose index is in the range 0 .. OutLayerGroup is selected to be decoded. The maximum value of the layer-group index of partial decoding OutLayerGroup is specified by the total number of layer-groups minus SkippedLayerGroup.

OutLayerGroup := num\_layer\_groups\_minus1 – SkippedLayerGroup  
  
if (layer\_group\_id == 0)  
 decode GDU or ADU  
else if (layer\_group\_id ≤ OutLayerGroup)  
 decode DGDU or DADU  
else  
 skip DGDU or DADU

Consequently, the depth of the geometry occupancy tree of the partial decoding PartialDepth is inferred by the sum of number of layers in each layer-group whose index is in the range 0 .. OutLayerGroup.

PartialDepth = 0  
for (i=0; I ≤ OutLayerGroup; i++)  
 PartialDepth += num\_layers\_minus1[i] + 1

* + - 1. Geometry position compensation

The maximum depth of the geometry occupancy tree when decoding all layer-groups is inferred by the sum of number of layers in each layer-group whose index is in the range 0 .. num\_layer\_groups\_minus1.

TotalDepth = 0  
for (i=0; i< num\_layer\_groups\_minus1; i++)  
 TotalDepth += num\_layers\_minus1[i] + 1

The MinNodeSizeLog2 is inferred by the difference between occtreeMaxDepthMinus1 and PartialDepth.

MinNodeSizeLog2 = occtreeMaxDepthMinus1 + 1 - PartialDepth

If MinNodeSizeLog2 is greater than 1, points shall be centred within their corresponding block:

for (ptIdx = 0; ptIdx < SubgroupNodeCnt[ layerGroupIdx ][ subgroupIdx ]; ptIdx++)  
 for (k = 0; k < 3; k++)  
 SubgroupNode[ layerGroupIdx ][ subgroupIdx ][ ptIdx ][ 𝑘 ] |= (MinNodeSizeLog2 > 1) << (MinNodeSizeLog2 – 1)

* + 1. Partial region decoding
       1. General

A decoder shall generate the point cloud of a partial region of a slice.

The partial region FGS point cloud is specified in terms of the following variables:

* The arrays RoiBBoxMin and RoiBBoxMax, an application-specific array which specifies the region of interest as the minimum and the maximum position of the bounding box.
* The arrays SubgroupNodePos[ layerGroupIdx ][ subgroupIdx ], the subgroup output nodes of the layer-group index layerGroupIdx and the subgroup index subgroupIdx.
* The arrays SubgroupNodeCnt[ layerGroupIdx ][ subgroupIdx ], the number of nodes in the subgroup output nodes of the layer-group index layerGroupIdx and the subgroup index subgroupIdx
  + - 1. Selection of FGS

If RoiBBoxMin and RoiBBoxMax are present, the subgroups whose subgroup bounding box is overlapped with the bounding box of region of interest is selected to be decoded.

if (layerGroupIdx == 0)  
 decode GDU or ADU  
else if ((RoiBBoxMin[0] < SubgroupBBoxMax[layerGroupIdx][subgroupIdx][0] &&   
 RoiBBoxMin[1] < SubgroupBBoxMax[layerGroupIdx][subgroupIdx][1] &&  
 RoiBBoxMin[2] < SubgroupBBoxMax[layerGroupIdx][subgroupIdx][2]) &&  
 (RoiBBoxMax[0] > SubgroupBBoxMin[layerGroupIdx][subgroupIdx][0] &&  
 RoiBBoxMax[1] > SubgroupBBoxMin[layerGroupIdx][subgroupIdx][1] &&  
 RoiBBoxMax[2] > SubgroupBBoxMin[layerGroupIdx][subgroupIdx][2]))  
 decode DGDU or DADU  
else  
 skip DGDU or DADU

[Ed. (HH): make shorter variable names]

Bibliography

1. Society of Motion Picture and Television Engineers ST 2042-1 (2017), *VC-2 Video Compression*

Index of named expressions and variables

AeBits, 11.3.4

AeBitsReadIdx, 11.3.4

AeReadBin, 11.5.2

AngPosScaleK, 9.2.14.6

AngularEligible, 9.2.13.7.2

AngularOrigin, 7.4.3.2

AttrBitDepth, 7.4.2.1.3

AttrCoeff, 10.3.1.1

AttrDim, 7.4.2.1.3

AttrIdx, 7.4.4.2, 7.4.5

AttrLevelScale, 10.7.4

AttrMaxVal, 7.4.2.1.3

AttrPos, 10.2.1

AttrPosAng, 10.2.2

AttrQp, 10.7.3

AttrQpLayerOffset, 10.7.1

AttrQstep, 10.7.4

AttrRegionQpOffset, 10.7.2

AttrRegionQpOrigin, 10.7.1

AttrRegionQpSize, 10.7.1

AttrRegionSizeConstraint, 10.7.1

AxisCoded, 9.2.6.6

AzimuthAxis, 9.2.13.3

AzimuthAxisIsS/T, 9.2.13.3

BeamElev, 7.4.2.5.2

BeamIdxEst, 9.2.13.6

BeamIdxFromGrad, 9.2.13.6

BeamMinDeltaGrad, 9.2.13.7.2

BeamOffsetV, 7.4.2.5.2

BeamStepsPerRev, 7.4.2.5.2

BiasedNorm1, 10.6.6.5

BiasedNorm2, 10.6.6.12

BinIdxPfx, 11.4.3, 11.4.4

BinIdxSfx, 11.4.3, 11.4.4

BinIdxTu, 11.4.4, 11.4.5

BlkSizeLog2, 10.6.5.6, 10.6.5.8, 10.6.6.6

BpBits, 11.3.4

BpBitsReadIdx, 11.3.4

CanonicalLodSubsampling, 10.6.5.5

CathetusV, 9.2.13.7.2

ChildNodeSizeLog2, 9.2.6.4

ChunkBuf, 11.3.4

ChunkDuRem, 11.3.5

ChunkLen, 11.3.6

ChunkNextAeBit, 11.3.9

ChunkNextBpBit, 11.3.10

ChunkPadLen, 11.3.8, 11.3.11

ChunkPartBLen, 11.3.11

CloudAttrPos, 10.8.2

CodedAxisCnt, 9.2.6.6

CoeffIdx, 10.5.3.1

CoeffWeight, 10.6.4, 10.6.11.1

Contexts, 11.5.3.1

Ctx, 11.5.3.4

CtxIdx, 11.5.3.4

CtxIdxAngPhi, 9.2.13.9

CtxIdxAngTheta, 9.2.13.10

CtxIdxDictHg, 9.2.9.11

CtxIdxOccBit, 9.2.10.6.4

CtxIdxPlanePos, 9.2.11.7.1

CtxTbl, 11.5.3.4

CtxUpdateDelta, 11.5.3.3

DataUnitBytes, 11.2.2

DataUnitLength, 11.2.2

DataUnitReadIdx, 11.2.2

DictsHistogram, 9.2.9.4

DictsMostProb, 9.2.9.4

DictsMostProbAge, 9.2.9.4

DictsMostProbMaxAge, 9.2.9.4

DictsNextEvictIdx, 9.2.9.4

DictsRecent, 9.2.9.4

DirectMode1Eligible, 9.2.12.3.1

DirectMode2Eligible, 9.2.12.3.1

DirectMode3Eligible, 9.2.12.3.1

DirectModeEligible, 9.2.12.3.1

DirectNodePresent, 9.2.12.3.2

DnBeamIdx, 9.2.13.8.1

DnBeamIdxEst, 9.2.13.8.1

DnBitsAfterJointPrefix, 9.2.12.5.2

DnBitsAfterPlanar, 9.2.12.4.3

DnEligibleCnt, 9.2.12.3.3

DnJointCoded, 9.2.12.4.4

DnJointDiffBitInferred, 9.2.12.5.4

DnJointDiffBitPresent, 9.2.12.5.4

DnJointDiffBits, 9.2.12.5.3

DnJointPos, 9.2.12.4.4

DnJointPosDiffBit, 9.2.12.5.3

DnJointPosPrefix, 9.2.12.5.2

DnJointPrefixBits, 9.2.12.5.2

DnPartialPosAng, 9.2.13.8.2

DnPlanarPos, 9.2.12.4.3

DnPresenceMask, 9.2.12.3.2

DnPtPos, 9.2.12.4.1

DnPtPosAng, 9.2.13.8.2

DnPtPosAngS, 9.2.13.8.2

DnPtPosConstraint, 9.2.12.5.1

DnPtPosRem, 9.2.12.4.1

DnPtPosS/T/V, 9.2.12.4.1

DnQp, 9.2.14.5.2

DnRemAngBitsST, 9.2.13.8.3

DnRemAngBitsV, 9.2.13.8.3

DnRemBits, 9.2.12.4.5

DnRemPos, 9.2.12.4.5

Dpth, 9.2.6.1

DuFooterLen, 11.2.4

DuIsGdu, 11.2.4

DuNextBit, 11.2.5

ExpMovAvg, 9.2.11.5.3

FixedRahtFilters, 10.5.5.11

FrameCtr, 8.2.2

GduFooterLen, 11.2.4

IdxOfSubsampledPoint, 10.6.5.9

InLodPtCnt, 10.6.5.4

InLodPtIdx, 10.6.5.4

InterCompPredCoeff, 10.6.10.1

InterCompPredCoeffPrev, 10.6.10.1

IsSubsampledPoint, 10.6.5.7

IsReplaced, 10.5.5.4.3

IvlCarry, C.2

IvlCode, 11.5.4.2

IvlLow, 11.5.4.2, C.2

IvlRange, 11.5.4.2, C.2

LastCompPredCoeff, 10.6.10.1

LastCompPredCoeffPrev, 10.6.10.1

LcpScale, 10.5.5.12

Length, 7.2

LodCnt, 10.6.4

LodCoeffQp, 10.6.9.1

LodPtCnt, 10.6.4

LodPtIdx, 10.6.4

LodRfmtPtCnt, 10.6.4

LodRfmtPtIdx, 10.6.4

Lvl, 10.5.3.1, 10.6.5.1

MapPtIdx, 10.6.5.6

MaxPtsPerSort, 10.6.5.2

MapSub, 10.6.5.6

MaxSeqBboxDimLog2, A.4.1

MaxSliceDimLog2, A.4.1

MaxSlicePoints, A.4.1

MinAng, 10.2.2

MinCurAng, 10.2.2

MinInterRefLvl, 10.6.6.2

MinNodeSizeLog2, D.2.1

MinRefAng, 10.2.2

Morton, 5.10.7

NextAeStreamBit, 11.5.4.4

Nloc, 9.2.6.3

NlocC, 9.2.6.3

NlocG, 9.2.6.3

NlocP, 9.2.6.3

NodeBeamIdx, 9.2.13.7.4

NodeBeamValid, 9.2.13.7.4

NodeIdx, 9.2.6.1

NodeOccMap, 9.2.11.5.2

NodeSizeLog2, 9.2.6.4

Norm2, 10.6.6.13

NposAng, 9.2.13.2

NposAngMid, 9.2.13.7.1

NposAngMidS/T/V, 9.2.13.7.1

NposAngS/T/V, 9.2.13.2

Ns/Nt/Nv, 9.2.6.3

NsC/NtC/NvC, 9.2.6.3

NsG/NtG/NvG, 9.2.6.3

NsP/NtP/NvP, 9.2.6.3

ObufCtxIdxDelta, 12.4.1.1

ObufInitProba, 12.2.2

ObufInitProbaBounds, 12.2.2

OccAdj, 9.2.10.6.9.3

OccAdjCnt, 9.2.10.6.9.3

OccAvailWinLog2, 9.2.7.2

OccBitCodingOrder, 9.2.10.2

OccBitIdx, 9.2.10.2

OccBitIdxFromNpBit, 9.2.8

OccBitInferSet, 9.2.10.5

OccBitInferUnset, 9.2.10.4

OccBitLocC, 9.2.10.2

OccBitMap, 9.2.10.2

OccBitPresent, 9.2.10.3

OccChildCnt, 9.2.6.11

OccCtxSel, 9.2.10.6.2

OccCtxSelUpdate, 9.2.10.6.5

OccCtxSelVar, 9.2.10.6.4

OccDictHistogram, 9.2.9.6

OccDictMostProb, 9.2.9.6

OccDictMostProbAge, 9.2.9.6

OccDictMostProbMaxAge, 9.2.9.6

OccDictMostProbMaxAgeNext, 9.2.9.9

OccDictNextEvictIdx, 9.2.9.6

OccDictRecent, 9.2.9.6

OccFreeAxis, 9.2.6.6

OccFreeAxisCnt, 9.2.6.6

OccFromNpOcc, 9.2.8

OccIntraPred, 9.2.10.6.9.3

OccKnown, 9.2.10.5

OccKnownMask, 9.2.10.5

OccKnownZero, 9.2.10.5

OccLocC, 9.2.2.3

OccLvlNodeSizeLog2, 9.2.4

OccMapPresent, 9.2.6.9

OccMaxChildren, 9.2.6.7

OccMaybeSingleChild, 9.2.6.8

OccMinChildren, 9.2.6.7

OccNeigh, 9.2.7.3

OccNeighAvail, 9.2.7.2

OccNeighCnt, 9.2.10.6.9.3

OccNeighPat, 9.2.7.4

OccNeighPatEq0, 9.2.12.3.3

OccNeighPatR, 9.2.7.5, 9.2.10.6.6

OccNodeChildCnt, 9.2.12.3.3

OccNodeCnt, 9.2.5.1

OccNodeLoc, 9.2.5.1

OccNodePresent, 9.2.5.1

OccPlaneMask, 9.2.11.5.5

OccPosScaleK, 9.2.14.6

OccQp, 9.2.14.5.1

OccQpScaleLog2, 9.2.14.2.3

OccQpSubtreeDepth, 9.2.14.4

OccScalingVolSizeLog2, 9.2.14.2.2

OccSubtreeQp, 9.2.14.5.3

OccSubtreeQpLoc, 9.2.14.5.3

OccSubtreeQpMax, 9.2.14.5.4

OccSubtreeQpOffset, 9.2.14.3

OcctreeEntropyStreamDepth, 9.2.3

OccupancyDensity, 9.2.11.5.2

OccupancyMapP, 9.2.11.5.5

OccVolScale, 9.2.14.2.3

OutLodPtCnt, 10.6.5.4, 10.6.5.6

OutLodPtIdx, 10.6.5.4

OutRfmtPtIdx, 10.6.5.4

PartialPtCnt, D.2.1

PartialPtIdx, D.2.1

PartVal, 11.4.5

PartValue, 11.4.1

PlanarBeamIdx, 9.2.13.7.3

PlanarEligible, 9.2.11.5.1

PlanarEligibleByAng, 9.2.11.5.1

PlanarEligibleByRate, 9.2.11.5.1

PlanarFreeAxis, 9.2.11.4

PlanarMinChildren, 9.2.11.3

PlanarMinPlanes, 9.2.11.2

PlanarNodeAxisLoc, 9.2.11.6.2

PlanarNodeZone, 9.2.11.6.3

PlanarRate, 9.2.11.5.2

PlanarRateThreshold, 9.2.11.5.1

PlaneThresholdIdx, 9.2.11.5.1

PointAng, 8.3.2

PointAttr, 8.3.5

PointCnt, 8.3.2

PointPos, 8.3.3

PredBias, 7.4.2.6.3

PredCnt, 10.6.4

PredMode, 10.6.8.1

PredModeCoded, 10.6.8.3, 10.6.8.4

PredModeMax, 10.6.8.1

PredModePresent, 10.6.8.2

PredPtIdx, 10.6.4

PredPtRef, 10.6.4

PredWeight, 10.6.4

PrevOccPlanePos, 9.2.11.6.4

PrevOccSinglePlane, 9.2.11.6.4

PrevPlanarNodeZone, 9.2.11.6.4

PtnAng, 9.3.4.3

PtnAngStv, 9.3.4.5

PtnCnt, 9.3.3.1

PtnDepth, 9.3.3.1

PtnIdx, 9.3.3.1

PtnPhiMul, 9.3.2.2

PtnPos, 9.3.4.2

PtnPred, 9.3.4.6

PtnQp, 9.3.2.2

PtnQpInterval, 9.3.2.1

PtnQpOffset, 9.3.2.2

PtnRef, 9.3.4.6

PtnResidual, 9.3.2.2

PtnSecResidual, 9.3.2.2

PtnStack, 9.3.3.1

QuantizedChildNodeSizeLog2, 9.2.14.2.3

QuantizedNodeSizeLog2, 9.2.14.2.3

QuantWeight, 10.6.4, 10.6.13.1

RahtBlkCnt, 10.5.2.2

RahtBlkChildCnt, 10.5.5.2

RahtBlkLoc, 10.5.2.2

RahtBlkSumAttrRef, 10.5.2.6

RahtBlkWeight, 10.5.2.3

RahtBlkWeightRef, 10.5.2.6

RahtCoeff, 10.5.2.2

RahtCoeffLcp, 10.5.5.12

RahtCoeffOrder, 10.5.3.2

RahtCoeffQp, 10.5.4.3

RahtCoeffScaled, 10.5.4.2

RahtCoeffWeight, 10.5.2.5

RahtCoeffWeightM, 10.5.2.5

RahtCoeffWeightMRef, 10.5.2.6

RahtDcCoeff, 10.5.6.2

RahtPredDcCoeff, 10.5.7.2

RahtDcNorm, 10.5.5.4.1

RahtFwd, 10.5.5.7

RahtInterFilter, 10.5.5.11

RahtInterPred, 10.5.5.9

RahtInterPredBlk, 10.5.5.1

RahtInterPredEligible, 10.5.5.8

RahtInterPredW, 10.5.5.9

RahtInv, 10.5.6.5

RahtLvlCnt, 10.5.2.2

RahtNeighCnt, 10.5.5.3

RahtNeighCntMinAncestor, 10.5.5.3

RahtPred, 10.5.5.4.4

RahtPredBlk, 10.5.5.1

RahtPredEligible, 10.5.5.2

RahtPredExcluded, 10.5.5.4.2

RahtPredRecipW, 10.5.5.4.4

RahtPredRecipWC, 10.5.5.4.5

RahtPredW, 10.5.5.5

RahtPredWeight, 10.5.5.4.4

RahtPredWeightB, 10.5.5.4.5

RahtPredWeightC, 10.5.5.4.5

RahtRecon, 10.5.7

RahtRootLvl, 10.5.2.4

RahtTreeQpOffset, 10.5.4.4

RahtTreeQpOffsetM, 10.5.4.4

RawAttrValueBits, 10.3.1.3

RecCloudAttr, 6.5.6

RecCloudPointCnt, 6.5.6

RecCloudPos, 6.5.6

RefAttrPos, 10.8.1

RefCloudAttrPos, 10.8.3

RefCloudAttrPosFirst, 10.8.3.2

RefCloudAttrPosSecond, 10.8.3.2

RefCloudAttr, 10.8.3

RefCloudAttrFirst, 10.8.3.2

RefCloudAttrSecond, 10.8.3.2

RefLodPtIdx, 10.6.5.10

RefPointAttr, 10.8.1

ScaledHalfNodeSize, 9.2.13.7.1

ScaledHalfNodeSizeS/T/V, 9.2.13.7.1

ScaledNodeSize, 9.2.13.7.1

ScaledNodeSizeS/T/V, 9.2.13.7.1

SelAdj, 9.2.10.6.8

SeCtx, 12.5

SelNeigh, 9.2.10.6.6

SelPred, 9.2.10.6.9.1

SelPredEligible, 9.2.10.6.9.2

SelSib, 9.2.10.6.7

SeqCodedScale, 7.4.2.1.2

SeqOrigin, 7.4.2.1.2

SeqUnit, 7.4.2.1.2

SliceOrigin, 7.4.3.2

StvToXyz, 7.4.2.1.2

SubgroupBBoxMin, E.4.2.3

SubgroupContexts, E.7.3.1.1

SubgroupContextsCounter, F.2.1

SubgroupContextsForAttributes, E.7.3.1.1

SubgroupDirectNodePointCnt, E.4.2.3

SubgroupNodeAttr, E.4.2.4

SubgroupNodeCnt, E.4.2.3

SubgroupNodePos, E.4.2.3

SubgroupOccNeighPatEq0, E.4.2.3

SubgroupOccNodeChildCnt, E.4.2.3

SumN19, 10.5.5.3

SumN26, 9.2.10.6.9.3

TerminalNode, 9.2.6.5

TileInventoryOrigin, 7.4.2.4

ZeroRunLength, 10.3.1.1