# Scope

This document specifies geometry-based point cloud compression.

# Normative references

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

*Recommendation ITU‑T T.35, Procedure for the allocation of ITU‑T defined codes for non-standard facilities*

*ISO/IEC 8825‑1 (Rec. ITU‑T X.690), Information technology — ASN.1 encoding rules — Part 1: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

*ISO/IEC 9834‑1 (Rec. ITU‑T X.660), Information technology — Procedures for the operation of object identifier registration authorities — Part 1: General procedures and top arcs of the international object identifier tree*

*ISO/IEC 9834‑8 (Rec. ITU-T X.667), Information technology — Procedures for the operation of object identifier registration authorities — Part 8: Generation of universally unique identifiers (UUIDs) and their use in object identifiers*

*ISO/IEC 23091‑2, Information technology — Coding-independent code points — Part 2: Video*

# Terms and definitions

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

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

* ISO Online browsing platform: available at <https://www.iso.org/obp>
* IEC Electropedia: available at <http://www.electropedia.org/>

## General terms

point

fundamental element of a *point cloud* (3.1.2) comprising a position specified as *Cartesian coordinates* (3.1.8) and zero or more *attributes* (3.1.20)

point cloud

unordered list of *points* (3.1.1)

point cloud sequence

sequence of one or more *point clouds* (3.1.2)

point cloud frame

*point cloud* (3.1.2) in a *point cloud sequence* (3.1.3)

coded point cloud frame

coded representation of a *point cloud frame* (3.1.4)

reference point cloud frame

a coded point cloud frame (3.1.5) which contains points (3.1.1) that may be used for inter prediction (3.1.28) in the decoding process of subsequent point cloud frames (3.1.4) in decoding order

canonical point order  
canonical decoding order

order of *points* (3.1.1) decoded from a *slice* (3.1.22) according to the decoding and parsing processes specified in this document

bounding box

axis-aligned cuboid defining a spatial region that bounds a set of *points* (3.1.1)

coordinates

<Cartesian> three scalar multiples of respective orthogonal *XYZ* (3.1.12) unit vectors with finite precision and bounds that specify a position relative to a fixed reference

coordinates

<angular> a position specified as the radial distance 𝜌 from the V axis, an azimuth angle 𝜑 in the S-T plane and an indexed elevation

coordinates

<attribute> either *STV* (3.1.13) or scaled *RPI* (3.1.14) point coordinates used to code an attribute

XYZ (axes)

X, Y and Z axes, in that order, used to represent *Cartesian coordinates* (3.1.9)

STV (axes)

S, T and V axes, in that order, that are a sequence-dependent permutation of the *XYZ axes* (3.1.12); used to represent the coded *geometry* (3.1.19)

RPI (axes)

R, P and I axes, in that order, used to represent *angular coordinates* (3.1.10)

sequence coordinate system

scaled and translated application-specific coordinate system that applies to an entire coded *point cloud sequence* (3.1.3), and in which all *points* (3.1.1) have non-negative, fixed-point coordinates

coding coordinate system

scaled *sequence coordinate system* (3.1.15) that applies for an entire coded *point cloud sequence* (3.1.3), and in which all *points* (3.1.1) have non-negative integer coordinates

slice coordinate system

translated *coding coordinate system* (3.1.16) that applies for a single *slice* (3.1.22), and in which all *points* (3.1.1) in the *slice* have non-negative integer coordinates

beam

sampler of point positions using angular coordinates by rays cast with a fixed elevation and from a point on and rotating around the V axis at the angular origin

geometry

*point positions* (3.4.1) associated with a set of *points* (3.1.1)

attribute

scalar or vector property associated with each *point* (3.1.1) in a *point cloud* (3.1.2)

EXAMPLE Colour, reflectance, frame index, etc.

position

<bit> bit in a binary string or value, representing the factor

EXAMPLE The LSB has bit position 0.

slice

geometry and attributes for part of, or an entire, *coded point cloud frame* (3.1.5)

Note 1 to entry: the bounding boxes of any two slices can intersect.

tile

set of *slices* (3.1.22) identified by a common slice\_tag *syntax element* value (3.2.15) whose *geometry* (3.1.19) should be contained within a *bounding box* (3.1.8) specified in a tile inventory data unit

prediction

an embodiment of the *prediction process* (3.1.25)

prediction process

the use of a *predictor* (3.1.26) to provide an estimate of the data element currently being decoded

predictor

a combination of specified values or previously decoded data elements used in the decoding process of subsequent data elements

intra prediction

a *prediction* (3.1.24) derived from only data elements (e.g., *point positions* (3.4.1) or *attributes* (3.1.20)) of the same decoded *slice* (3.1.22)

inter prediction

a *prediction* (3.1.24) derived in a manner that is dependent on data elements (e.g., *point positions* (3.4.1) or *attributes*(3.1.20)) of one or more *reference point cloud frames* (3.1.6)

uni-prediction

an inter *prediction* (3.1.28) derived in a manner that is dependent on data elements (e.g., *point positions* (3.4.1) or *attributes*(3.1.20)) of one *reference point cloud frame* (3.1.6)

bi-prediction

an inter *prediction* (3.1.28) derived in a manner that is dependent on data elements (e.g., *point positions* (3.4.1) or *attributes*(3.1.20)) of two *reference point cloud frames* (3.1.6)

intra (I) slice

a *slice* (3.1.22) that is decoded using *intra prediction* (3.1.27) without referring to a *reference point cloud frame* (3.1.6)

predictive (P) slice

a *slice* (3.1.22) that is decoded using *intra prediction* (3.1.27) or *inter prediction* (3.1.28) from at most one *reference point cloud frame* (3.1.6)

bi-predictive (B) slice

a *slice* (3.1.22) that is decoded using *intra prediction* (3.1.27) or *inter prediction* (3.1.28) from at most two *reference point cloud frames* (3.1.6)

reference slice

a coded *slice* (3.1.22) which contains points (3.1.1) that may be used for inter prediction (3.1.28) in the decoding process of subsequent *slices* (3.1.22) in decoding order

intra (I) frame

a *point cloud frame* (3.1.4) that is decoded using *intra prediction* (3.1.27) without referring to a *reference point cloud frame* (3.1.6)

predictive (P) frame

a *point cloud frame* (3.1.4) that is decoded using *intra prediction* (3.1.27) or *inter prediction* (3.1.28) from at most one *reference point cloud frame* (3.1.6)

bi-predictive (B) frame

a *point cloud frame* (3.1.4) that is decoded using *intra prediction* (3.1.27) or *inter prediction* (3.1.28) from at most two *reference point cloud frames* (3.1.6)

Morton code

non-negative integer obtained by interleaving the bits of three integers

Morton order

elements ordered according to their *Morton code* (3.1.38)

sparse array

array with fewer set elements than total addressable elements; unset elements can have an inferred value when accessed

temporal ID

An identifier associated with a point cloud frame which may be used to support temporal scalability.

## High-level syntax and entropy coding terms

ASN.1  
abstract syntax notation one

notation specified by Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1 that is used for the definition of data types, values and constraints on data types

[SOURCE: Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1]

bin

binary symbol (bit) of the *binarized* (3.2.3) representation of a *syntax element* value (3.2.15)

binarization

specification of a *syntax element*'s value (3.2.15) as a sequence of *bins* (3.2.2)

bypass

<symbol> a static, equiprobable probability model

bypass

<stream> *bypass symbols* (3.2.4) that are not encoded in an arithmetic-coded *bitstream* (3.2.6)

bitstream

<data> sequence of bits

bitstream

<coded sequence> sequence of bits, in the form of encapsulated *data units* (3.2.12), that represents a coded *point cloud sequence* (3.1.3)

set bit

bit with the value 1

unset bit

bit with the value 0

byte

sequence of 8 bits, typeset with the most significant bit on the left and the least significant bit on the right.

Note 1 to entry: When represented in a bitstream, the most significant bit of a byte is first.

byte aligned

*bitstream* (3.2.6) position that is an integer multiple of eight bits from the position of the first bit in the bitstream

data unit  
DU

sequence of *bytes* (3.2.10) conveying a single *syntax structure* (3.2.16) of known length

data unit header

parameters, located from the start of a *data unit* (3.2.12)

data unit footer

parameters, located from the end of a *data unit* (3.2.12)

syntax element

element of data represented in the *bitstream* (3.2.6)

syntax structure

zero or more *syntax elements* (3.2.15) present together in the *bitstream* (3.2.6) in a specified order

parameter set

collection of parameters that apply when activated

sequence parameter set  
SPS

parameters for an entire coded *point cloud sequence* (3.1.3), conveyed by an SPS *data unit* (3.2.12) and activated when referenced by a geometry data unit

geometry parameter set  
GPS

parameters for the coding of *slice* (3.1.22) geometry, conveyed by a GPS *data unit* (3.2.12) and activated when referenced by a geometry data unit

attribute parameter set  
APS

parameters for the coding of a *slice* (3.1.22) attribute, conveyed by an APS *data unit* (3.2.12) and activated when referenced by an attribute data unit

object identifier  
OID

<ASN.1> ordered list of primary integer values from the root of the *international object identifier tree* (3.2.22) to a node, which unambiguously identifies that node

[SOURCE: Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1]

international object identifier tree

tree whose root corresponds to Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1 and whose nodes correspond to *registration authorities* (3.2.24) responsible for allocating arcs from a parent node

[SOURCE: Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1]

registration

<object identifier> assignment of an unambiguous name to an object in a way which makes the assignment available to interested parties

[SOURCE: Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1]

registration authority

<international object identifier tree> an entity such as an organization, a standard or an automated facility that performs *registration* (3.2.23) of one or more types of objects

[SOURCE: Rec. ITU‑T X.660﻿ |‌ ISO/IEC 9834‑1]

application specific

defined by an application or an application standard

unspecified

when used in subclauses specifying values of a particular *syntax element*, indicates that the values have no specified meaning in this document and will not have a specified meaning in future versions of this document

## Tree structure terms

tree

recursive structure of *nodes* (3.3.7) without loops, and containing a single *root node* (3.3.5)

top

<tree> *tree level* (3.3.4) with *depth* of 0 (3.3.8), consisting of the *root node* (3.3.5)

bottom

<tree> *tree level* (3.3.4) with the greatest *depth* (3.3.8)

tree level

set of *nodes* (3.3.7) at the same *depth* (3.3.8) in a *tree* (3.3.1)

root node

<tree> *node* (3.3.7) without a *parent node* (3.3.10)

leaf node

terminal *node* (3.3.7) without any *child nodes* (3.3.9)

node

<tree> element of a *tree* (3.3.1)

depth

<node> number of descendent hops from the *root node* (3.3.5) to a *node* (3.3.7)

child node

direct descendent of a *node* (3.3.7)

parent node

direct ancestor of a *node* (3.3.7)

grandparent node

direct ancestor of a *node*'s (3.3.7) *parent node* (3.3.10)

great-grandparent node

direct ancestor of a *node*'s (3.3.7) *grandparent node* (3.3.11)

sibling nodes

*nodes* (3.3.7) that are *child nodes* (3.3.9) of the same *parent node* (3.3.10)

subtree

part of a *tree* (3.3.1) comprising a *subtree root node* (3.3.15) and all its descendents over all subsequent *tree levels* (3.3.4)

root node

<subtree> single *node* (3.3.7) of a *subtree* (3.3.14) from which all other nodes in the same subtree are descendents

TriSoup node

occupied <occupancy tree> *leaf node* (3.3.6) of an *occupancy tree* (3.4.2) representing a sub-volume of the 3D space (or volume) containing at least one point of the *point cloud* (3.1.2)

## Geometry coding terms

position

<point> three-dimensional coordinates of a *point* (3.1.1)

occupancy tree

eight-ary *tree* (3.3.1) of *occupancy tree nodes* (3.4.4) representing the *geometry* (3.1.19) of a *slice* (3.1.22)

predictive tree

*tree* (3.3.1) of *predictive tree nodes* (3.4.5) representing the *geometry* (3.1.19) of a *slice* (3.1.22)

node

<occupancy tree> *node* (3.3.7) of an *occupancy tree* (3.4.2) representing a sub-volume of the 3D space (or volume) containing the *point cloud* (3.1.2)

node

<predictive tree> *node* (3.3.7) of a *predictive tree* (3.4.3) representing a single *position* (3.4.1) for one or more *points* (3.1.1)

direct node

<occupancy tree> terminal *node* (3.4.4) that codes one or more *point positions* (3.4.1)

occupancy bitmap

8-bit bitmap for an occupancy tree *node* (3.4.4) whose bits indicate the existence of *child nodes* (3.3.9) at particular locations in the next *tree level* (3.3.4)

occupied neighbourhood pattern

indicates the existence and arrangement of the six possible occupancy tree *nodes* (3.4.4) that share faces with a central node

TriSoup

representation of the geometry of the *point cloud* (3.1.2) in a *TriSoup node* (3.3.16) by a set of triangles

TriSoup edge

any edge belonging to a cuboid volume associated with a *TriSoup node* (3.3.16)

TriSoup edge vertex

a point located on a *TriSoup edge* (3.4.10)

TriSoup face vertex

a point located on a face shared by two adjacent *TriSoup* nodes (3.3.16)

TriSoup vertex

a point located on a face shared by two adjacent *TriSoup nodes* (3.3.16)

TriSoup triangle

a triangle belonging to a *TriSoup node* (3.3.16) and whose vertex are defined from the *TriSoup vertices* (3.4.13) belonging to the cuboid volume associated with said TriSoup node

voxelization (of a TriSoup triangle)

process of transforming a *TriSoup* triangle (3.4.14) into a set of points (3.1.1)

## Attribute coding terms

primary attribute component

first, or only, attribute component, identified by the index 0

secondary attribute component

attribute component other than the first component, identified by an index greater than 0

detail level

set of *points* (3.1.1) that represent a subsampled version of the slice *geometry* (3.1.19)

refinement list

set of *points* (3.1.1) present in one *detail level* (3.5.3) that are not present in the next coarsest *detail level*

refinement point

*point* (3.1.1) in a *refinement list* (3.5.4)

predictor set

set of neighbouring *points* (3.1.1) from which an *attribute* (3.1.20) value is predicted

## Fine granularity slices terms

fine granularity slices

a subset of a slice which carries a geometry or an attribute of a subgroup (3.6.3) in a layer-group (3.6.2)

layer-group

a group of consecutive tree levels (3.3.4) of an occupancy tree (3.4.2) or the attribute-assigned occupancy tree (3.6.7)

subgroup

a spatial subset of a layer-group (3.6.2) where the bounding box of a subgroup shall not overlap with other subgroups in the same layer-group

root layer-group

a layer-group (3.6.2) which contains a root node (3.3.5) of the occupancy tree (3.4.2) or the attribute-assigned occupancy tree (3.6.7)

parent subgroup

a layer-group (3.6.2) which contains a root node (3.3.5) of the occupancy tree (3.4.2) or the attribute-assigned occupancy tree (3.6.7)

child subgroup

a subgroup (3.6.3) in a layer-group (3.6.2) adjacent to the maximum depth of the current subgroup, where the bounding box of the child subgroup is a subset of the bounding box of the current subgroup

# Abbreviated terms

## Acronyms

APS Attribute parameter set

ADU Attribute data unit

CBS Chunked bytestream

CPM Contextual probability model

DADU Dependent attribute data unit

DGDU Dependent geometry data unit

DU Data unit

FBDU Frame boundary marker data unit

FGS Fine granularity slice

FSAP Frame-specific attribute properties

GDU Geometry data unit

GOF Group of frames

GPS Geometry parameter set

G-PCC Geometry-based point cloud compression

LoD Level(s) of detail

LSB Least significant bit

MSB Most significant bit

NA Not applicable

QP Quantization parameter

RAHT Region adaptive hierarchical transform

SPS Sequence parameter set

## Mnemonics

EGk Exponential Golomb code of order 𝑘

FL Fixed-length code

FL+S Fixed-length code plus conditional sign bit

TU Truncated unary code

attr Attribute

cnt Count

geom Geometry

idx Index

occtree Occupancy tree

occ Occupancy tree node

ptree Predictive tree

ptn Predictive tree node

ti Tile inventory

tlv Type-length-value

seq Sequence

# Conventions

## General

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

## Symbolic names

Variables and expressions use the following case-insensitive naming conventions to indicate their use:

* 𝑖, 𝑗: general loop or index variable
* 𝑘: component of an XYZ/STV/RPI position, coordinate or location
* 𝑐: component of an attribute
* qc: quantization-parameterized component: 0 – primary; 1 – secondary
* dpth: the depth of a node or level in a tree
* lvl: tree level or detail level, counted from the bottom of a tree or hierarchy
* 𝑚: Morton-coded location
* ns, nt, nv: node coordinates
* nsc, ntc, nvc: child node coordinates
* nsp, ntp, nvp: parent node coordinates
* ptIdx: index of a point in canonical decoding order
* rfmtIdx: index of a point in an array of LoD refinement points
* ni: index for an element in a point's predictor set

## Numerical representation

|  |  |
| --- | --- |
| binary | typeset as 'X…XX' where each base 2-digit X is 0 or 1 |
| octal | typeset as X…XX8 where each base 8-digit X is 0 to 7 |
| decimal | typeset as X…XX where each base 10-digit X is 0 to 9 |
| hexadecimal | typeset as 0xX…XX where each base 16-digit X is 0 to 9 or A to F |

## Arithmetic operators

|  |  |
| --- | --- |
| + | Addition |
| − | Subtraction (as a two-argument operator) or negation (as a unary prefix operator) |
| × | Multiplication |
|  | Exponentiation. Specifies 𝑥 to the power of 𝑦. In other contexts, such notation is used for superscripting not intended for interpretation as exponentiation. |
| / | Integer division with truncation of the result toward zero. For example, 7 / 4 and −7 / −4 are truncated to 1 and −7 / 4 and 7 / −4 are truncated to −1. |
| ÷ | Division where no truncation or rounding is intended |
|  | Division in mathematical equations where no truncation or rounding is intended |
|  | Summation of 𝑓( 𝑖 ) with 𝑖 taking all integer values from 𝑥 up to and including 𝑦 |
| 𝑥 % 𝑦 | Modulus, remainder of 𝑥 divided by 𝑦, defined only for integers 𝑥 and 𝑦 with 𝑥 ≥ 0 and 𝑦 > 0 |

## Logical operators

|  |  |
| --- | --- |
| 𝑥 && 𝑦 | Conditional boolean logical "and" of 𝑥 and 𝑦; the operand 𝑦 is only evaluated if 𝑥 is true. |
| 𝑥 || 𝑦 | Conditional boolean logical "or" of 𝑥 and 𝑦; the operand 𝑦 is only evaluated if 𝑥 is false. |
| ¬ | Boolean logical "not" |
| 𝑥 ? 𝑦 : 𝑧 | If 𝑥 is true or not equal to 0, evaluates to 𝑦; otherwise, evaluates to 𝑧 |

## Relational operators

|  |  |
| --- | --- |
| > | Greater than |
| ≥ | Greater than or equal to |
| < | Less than |
| ≤ | Less than or equal to |
| == | Equal to |
| ≠ | Not equal to |

## Bit-wise operators

|  |  |
| --- | --- |
| & | Bit-wise "and". When operating on integer arguments, operates upon a two's complement representation of the integer value. When operating upon a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding MSBs equal to 0. |
| | | Bit-wise "or". When operating on integer arguments, operates upon a two's complement representation of the integer value. When operating upon a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding MSBs equal to 0. |
| ^ | Bit-wise "exclusive or". When operating on integer arguments, operates upon a two's complement representation of the integer value. When operating upon a binary argument that contains fewer bits than another argument, the shorter argument is extended by adding MSBs equal to 0. |
| 𝑥 >> 𝑦 | Arithmetic right shift as specified by DivExp2Floor( 𝑥, 𝑦 ). It is equivalent to shifting a two's complement integer representation of 𝑥 by 𝑦 binary digits. This operator is defined only for non-negative integer values of 𝑦. |
| 𝑥 << 𝑦 | Arithmetic left shift of a two's complement integer representation of 𝑥 by 𝑦 binary digits. This operator is defined only for non-negative integer values of 𝑦. Bits shifted into the LSBs as a result of the left shift have a value equal to 0. |

According to the rules of precedence (5.11), the expressions 𝑎 + 𝑏 << 𝑐 + 𝑑 and 𝑎 | 𝑏 << 𝑐 | 𝑑 are identical to ( 𝑎 + 𝑏 ) << ( 𝑐 + 𝑑 ) and 𝑎 | ( 𝑏 << 𝑐 ) | 𝑑, respectively, and not 𝑎 + ( 𝑏 << 𝑐 ) + 𝑑 or ( 𝑎 | 𝑏 ) << ( 𝑐 | 𝑑 ).

## Assignment operators

|  |  |
| --- | --- |
| = | Assignment operator |
| := | Expression definition (5.12) |
| ++ | Increment, i.e. 𝑥++ is equivalent to 𝑥 = 𝑥 + 1; when used in an array index, evaluates to the value of the variable prior to the increment operation |
| −− | Decrement, i.e. 𝑥−− is equivalent to 𝑥 = 𝑥 − 1; when used in an array index, evaluates to the value of the variable prior to the decrement operation |
| ×= | Multiply by amount specified and update, i.e. 𝑥 ×= 3 is equivalent to 𝑥 = 𝑥 × 3 |
| += | Increment by amount specified, i.e. 𝑥 += 3 is equivalent to 𝑥 = 𝑥 + 3, and 𝑥 += (−3) is equivalent to 𝑥 = 𝑥 + (−3) |
| −= | Decrement by amount specified, i.e. 𝑥 −= 3 is equivalent to 𝑥 = 𝑥 − 3, and 𝑥 −= (−3) is equivalent to 𝑥 = 𝑥 − (−3) |
| >>= | Arithmetic right shift by amount specified, i.e. 𝑥 >>= 1 is equivalent to 𝑥 = 𝑥 >> 1 |
| <<= | Arithmetic left shift by amount specified, i.e. 𝑥 <<= 1 is equivalent to 𝑥 = 𝑥 << 1 |

## Range notation

|  |  |
| --- | --- |
| 𝑥 = 𝑎 .. 𝑏 | 𝑥 takes on monotonically increasing integer values starting from 𝑎 and proceeding to 𝑏, inclusive, with 𝑥, 𝑎 and 𝑏 being integer numbers |

## Mathematical functions

### General

|  |  |
| --- | --- |
| ArcSin( 𝑥 ) | Trigonometric arc sine function |
| Abs( 𝑥 ) |  |
| Bit( 𝑥, 𝑖 ) | ( 𝑥 >> 𝑖 ) & 1 |
| Ceil( 𝑥 ) | Lowest integer greater than or equal to 𝑥 |
| Clip3( min, max, 𝑥 ) |  |
| CrossProduct( *v1*, *v2* ) |  |
| DivExp2Floor( 𝑥, 𝑝 ) |  |
| DivExp2Fz( 𝑥, 𝑝 ) |  |
| DivExp2Tz( 𝑥, 𝑝 ) | 𝑥 / Exp2( 𝑝 ) |
| DivExp2Up( 𝑥, 𝑝 ) |  |
| DivExp2Inf( 𝑥, 𝑝 ) |  |
| Exp2( 𝑝 ) |  |
| Floor( 𝑥 ) | Greatest integer less than or equal to 𝑥 |
| Gcd( 𝑎, 𝑏 ) | Greatest integer that is a factor of both 𝑎 and 𝑏 |
| InnerProduct( *v1*, *v2* ) |  |
| IntLog2( 𝑥 ) | Floor( Log( 𝑥 ) ÷ Log( 2 ) ) |
| Log( 𝑥 ) | Natural logarithm of the argument 𝑥 |
| Min( 𝑥, 𝑦 ) |  |
| MinVec( *v*) | Min( *v*[0], Min( *v*[1], *v*[2] ) ) where *v* is a three-dimensional vector |
| Max( 𝑥, 𝑦 ) |  |
| MaxVec( *v*) | Max( *v*[0], Max( *v*[1], *v*[2] ) ) where *v* is a three-dimensional vector |
| PopCnt( 𝑥 ) | Number of set bits present in the binary representation of 𝑥 |
| RoundFz( 𝑥 ) | ( 2 × 𝑥 + Sign( 𝑥 ) ) / 2 |
| RoundUp( 𝑥 ) | ( 2 × 𝑥 + 1 ) / 2 |
| Sign( 𝑥 ) |  |
| Sin( 𝑥 ) | Trigonometric sine function |
| Sqrt( 𝑥 ) |  |

[Ed. (YX): DivExp2Up may need to be replaced by DivExp2inf in some decoding process.]

### IntAtan2

The function theta = IntAtan2( 𝑦, 𝑥 ) is a 20-bit fixed-point approximation of the arc tangent of that accounts for the Cartesian quadrant of the parameters. Its:

* parameters 𝑥 and 𝑦 are integer, Cartesian coordinates;
* result shall be equal to the value of the expression intAtan2[ 𝑦 ][ 𝑥 ].

The expression sineThetaI is for a right-angled triangle with catheti adj and opp in the first octant.

sineThetaI := opp × IntRecipSqrt(adj × adj + opp × opp) >> 20  
 where  
 opp := Min(Abs(y), Abs(x))  
 adj := Max(Abs(y), Abs(x))

The angle thetaI is derived by interpolating between values of ArcSinFp for a 9-bit approximation of sine theta.

thetaI := ArcSinFp[idx0] + (alpha × (ArcSinFp[idx1] − ArcSinFp[idx0]) >> 11)  
 where  
 idx0 := sineThetaI >> 11  
 idx1 := Min(362, idx0 + 1)  
 alpha = sineThetaI % Exp2(11)

The expression ArcSinFp[ 𝑥 ] specifies the 20-bit fixed-point approximation for arc sine.

The result is obtained by mapping thetaI to the correct octant according to the signs of the parameters.

intAtan2[y][x] := Sign(y) × thetaIh  
 where thetaIh :=  
 x > 0 && Abs(x) ≥ Abs(y) ? thetaI :  
 x > 0 && Abs(x) < Abs(y) ? 1647099 − thetaI :  
 x < 0 && Abs(x) < Abs(y) ? 1647100 + thetaI :  
 x < 0 && Abs(x) ≥ Abs(y) ? 3294177 − thetaI : 0

### IntCos and IntSin

The functions 𝑥 = IntCos( 𝜃, piBits ) and 𝑦 = IntSin( 𝜃, piBits ) are 24-bit fixed-point approximations of the cosine and sine of 𝜃. Their:

* parameters 𝜃 and piBits specify an angle measured in units of half turns;
* result shall be equal to the value of the expression cathetus.

The fixed-point cathetus for the unit circle is calculated by interpolating between values of SinFp. The values of sgn and theta are determined from 𝜃 for the corresponding function as specified by Table 1. The variable pi, equal to Exp2( piBits ), represents one half turn.

cathetus := sgn × DivExp2Up(iFrac0 × SinFp[idx0] + iFrac1 × SinFp[idx1], fracBits)  
 where  
 fracBits := piBits − 11  
 iFrac1 := theta − ((theta >> fracBits) << fracBits)  
 iFrac0 := Exp2(fracBits) − iFrac1  
 idx0 := Min(1024, theta >> fracBits)  
 idx1 := Min(1024, idx0 + 1)

Table 1 — Values of sgn and theta for functions IntCos and IntSin

| Domain | IntCos | | IntSin | |
| --- | --- | --- | --- | --- |
| sgn | theta | sgn | theta |
| 𝜃 ≤ −pi | −1 | pi/2 | 0 | 0 |
| −pi < 𝜃 < −pi/2 | −1 | −𝜃 − pi/2 | −1 | pi + 𝜃 |
| −pi/2 ≤ 𝜃 < 0 | 1 | 𝜃 + pi/2 | −1 | −𝜃 |
| 0 ≤ 𝜃 < pi/2 | 1 | −𝜃 + pi/2 | 1 | 𝜃 |
| pi/2 ≤ 𝜃 < pi | −1 | 𝜃 − pi/2 | 1 | pi − 𝜃 |
| 𝜃 ≥ pi | −1 | pi/2 | 0 | 0 |

The expression SinFp[ 𝑥 ] specifies the 24-bit fixed-point approximation of sine.

### IntSqrt

The function 𝑟 = IntSqrt( 𝑥 ) is an integer approximation of the principal square root of 𝑥. Its:

* parameter 𝑥 is a non-negative integer;
* result shall be equal to the value of the expression intSqrt[ 𝑥 ].

It is specified in terms of the fixed-point reciprocal square root. If the parameter 𝑥 is greater than or equal to , the calculation uses a quantized value of 𝑥 to ensure computability using 64-bit arithmetic.

IntSqrt( 0 ) is 1.

intSqrt[x] := x ≤ Exp2(46)  
 ? 1 + (x × IntRecipSqrt(x) >> 40)  
 : 1 + (x8 × IntRecipSqrt(x8) >> 32)  
 where  
 x8 := DivExp2(x, 16) + 1

### IntRecipSqrt

The function rRecip = IntRecipSqrt( 𝑥 ) is a 40-bit fixed-point approximation of the reciprocal square root of 𝑥. Its:

* parameter 𝑥 is a non-negative integer;
* result shall be equal to the value of the expression intRecipSqrt[ 𝑥 ].

IntRecipSqrt( 0 ) is 0.

The parameter 𝑥 is scaled to be in the range in the expression xScaled by multiplying or dividing by a power of four. The expression xScaleLog4 is the log4 scale factor.

xScaled := Floor(x × Exp2(2 × xScaleLog4))  
xScaleLog4 := 15 − IntLog2(x) / 2

The reciprocal square root shall be determined by two rounds of the Newton–Raphson method. The initial approximation for the scaled parameter 𝑥 is specified by the expression approxR0. Table 2 specifies the initial approximants over the domain of approxR with 18 fractional bit precision.

approxR0 := approxR[xScaled >> 25]

The second approximation from the first round of the Newton–Raphson method is specified by the expression approxR1.

approxR1 := threeR0[approxR0] − (rCubed0[approxR0] × xScaled >> 32)  
threeR0[r] := 3 × DivExp2Fz(r, 18) << 22  
rCubed0[r] := DivExp2Fz(r × r × r, 54) << 8

The third approximation from the second round of the Newton–Raphson method is specified by the expression approxR2.

approxR2 := threeR1[approxR1] − rCubed1[approxR1] >> 32  
threeR1[r] := r × 3 << 28  
rCubed1[r] := r × (r × (r × pInScaled >> 32) >> 32)

The result is obtained by scaling the third approximation by the square root of the initial scale factor.

intRecipSqrt[x] := x > 0 ? Floor(approxR2 × Exp2(xScaleLog4 − 3)) : 0

Table 2 — Initial approximations approxR[ 𝑖 + 𝑗 ] for IntRecipSqrt( 𝑖 + 𝑗 << 25 ). Values are typeset in hexadecimal form without the 0x prefix.

| 𝑗 | 𝑖 | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **32** | 3F7FFDA | 3E7FFB7 | 3DBFFBD | 3CC0013 | 3BFFFEE | 3AFFFE1 | 3A3FFDE | 397FF96 |
| **40** | 38FFFDE | 383FFC0 | 3780063 | 36FFFCD | 3640014 | 35BFFFA | 34FFF8B | 3480010 |
| **48** | 3400039 | 3380042 | 3300008 | 328002B | 3200046 | 317FFC2 | 3100012 | 307FFDB |
| **56** | 303FFC5 | 2FC004F | 2F3FFE0 | 2EFFF93 | 2E7FF91 | 2E3FF83 | 2DC0037 | 2D7FFB6 |
| **64** | 2D0000B | 2CBFF96 | 2C7FF66 | 2C00017 | 2BC0053 | 2B7FFA4 | 2AFFF43 | 2AC004B |
| **72** | 2A80061 | 2A3FF6A | 2A00032 | 29BFF3B | 297FF74 | 293FFC9 | 28C001E | 287FF6D |
| **80** | 283FF9E | 27FFF93 | 27BFFB1 | 27BFFE8 | 277FF3C | 2700056 | 2700000 | 26C0069 |
| **88** | 26800CE | 26400CD | 25FFF5A | 25BFFA8 | 25BFFD6 | 258008C | 25400B9 | 2500020 |
| **96** | 24BFF93 | 24C00B6 | 24800E6 | 24400B3 | 2400011 | 2400054 | 23C0049 | 237FF98 |
| **104** | 2380104 | 233FFFC | 2300047 | 2300024 | 22C0029 | 2280000 | 2280012 | 223FF79 |
| **112** | 223FF54 | 21FFF56 | 21BFFDE | 21C0078 | 2180111 | 217FF3D | 2140023 | 20FFF0F |
| **120** | 21000F1 | 20C0019 | 20C0137 | 2080015 | 2080091 | 204004F | 20400F7 | 200006B |

### IntRecip

The function ( recip, *fracBits* ) = Recip(divisor ) returns a fixed-point approximation of 1 ÷ divisor and the number of fractional bits in the fixed-point approximation. Its:

* parameter divisor is a non-null integer;
* result is the tuple specified by the value of the expressions recip and *fracBits* .

The number of fractional bits in the fixed-point approximation is provided by fracBits expression.

fracBits := (31 << 1) – fracBitsOffset

Where the fracBitsOffset expression is determined by

fracBitsOffset := 30 – IntLog2(divisor)

fracBitsOffset is determined to maximise the number of bits during the computation of recip while letting the possibly of computing it into 64 bits signed integer registers without any risk of overflow.

The parameter divisor is scaled to be in the range in the expression scaledDivisor by multiplying or dividing by a power of two.

scaledDivisor := Floor(divisor × Exp2(fracBitsOffset))

The approximation of 1 ÷ divisor shall be determined by one iteration of the Newton–Raphson division approximation method. The initial approximation for the method is specified by the expression recip0 and corresponds to a first order approximation of 1 ÷ divisor.

recip0 := (Round(48 ÷ 17 × Exp2(28)) << 31) - Round(32 ÷ 17 × Exp2(28)) × scaledDivisor

The recip expression corresponds to the final approximation and is specified from the first round of the Newton–Raphson method.

recip := recip0 \* ((1 << 31) - (scaleDivisor \* recip0 >> 31)) >> 31

### Div

The function quotient = Div( dividend, divisor, fracBits ) is a fixed-point approximation of dividend ÷ divisor. Its:

* parameters dividend and divisor are integers;
* parameter fracBits is the number of fractional bits in the fixed-point result;
* result is specified by the value of the expression quotient.

quotient := dividend × recipDivisor[idx] >> 16 + excess − fracBits  
 where  
 idx := DivExp2Fz(divisor, excess)  
 excess := Max(0, IntLog2(divisor) − 7)  
 recipDivisor[idx] := RoundFz(Exp2(16) ÷ idx) − 1

### Morton

The function 𝑚 = Morton( 𝑠, 𝑡, 𝑣 ) converts its parameters to a 3D Morton code. Its:

* parameters 𝑠, 𝑡 and 𝑣 are non-negative integers;
* result is specified by the value of the expression morton.

The conversion interleaves the bits of each parameter 𝑣, 𝑡 and 𝑠; in that order, starting from the LSBs. The LSB of 𝑣 is the LSB of 𝑚. Table 3 illustrates the construction of 3D Morton codes from the bit string representation of the parameters 𝑠, 𝑡 and 𝑣.

The expression Morton[ expr ] performs the same conversion for an expression expr that takes an argument 𝑘, 𝑘 ∈ { 0, 1, 2 }.

Morton[expr] := Morton(expr[0], expr[1], expr[2])

Table 3 — Construction of 3D Morton codes 𝑚 from the tuple ( 𝑠, 𝑡, 𝑣 )

| Bit string form | | | | Decimal form |
| --- | --- | --- | --- | --- |
| 𝑠 | 𝑡 | 𝑣 | 𝑚 | 𝑚 |
| '0 0' | '0 0' | '0 0' | '0 0 0  0 0 0' | 0 |
| '0 0' | '0 0' | '0 1' | '0 0 0  0 0 1' | 1 |
| '0 1' | '1 1' | '1 0' | '0 1 1  1 1 0' | 30 |
| '0 1' | '1 1' | '1 1' | '0 1 1  1 1 1' | 31 |
| '1 0' | '0 1' | '1 0' | '1 0 1  0 1 0' | 42 |
| '1 0' | '0 1' | '1 1' | '1 0 1  0 1 1' | 43 |
| '1 1' | '1 0' | '0 0' | '1 1 0  1 0 0' | 52 |
| '1 1' | '1 0' | '0 1' | '1 1 0  1 0 1' | 53 |
| … | … | … | … | … |

### FromMorton

The function ( 𝑠, 𝑡, 𝑣 ) = FromMorton( 𝑚 ) is the inverse of Morton( 𝑠, 𝑡, 𝑣 ). Its:

* parameter 𝑚 is a non-negative, integer, 3D Morton code;
* result is the tuple specified by the value of the expressions 𝑠, 𝑡 and 𝑣.

The conversion deinterleaves the bits of 𝑣, 𝑡 and 𝑠; in that order, starting from the LSB. The LSB of 𝑚 is the LSB of 𝑣.

## Order of operation precedence

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

* Operations of a higher precedence are evaluated before any operation of a lower precedence.
* Operations of the same precedence are evaluated sequentially from left to right.

Table 4 specifies the precedence of operations from highest to lowest; a higher position in the table indicates a higher precedence.

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

Table 4 — Operation precedence from highest (at top of table) to lowest (at bottom of table)

| Operations (with operands 𝑥, 𝑦 and 𝑧) |
| --- |
| 𝑥++, 𝑥−− |
| ¬𝑥, −𝑥 (as a unary prefix operator) |
|  |
| 𝑥 × 𝑦, 𝑥 / 𝑦, 𝑥 ÷ 𝑦, , 𝑥 % 𝑦 |
| 𝑥 + 𝑦, 𝑥 − 𝑦 (as a two-argument operator), |
| 𝑥 << 𝑦, 𝑥 >> 𝑦 |
| 𝑥 < 𝑦, 𝑥 ≤ 𝑦, 𝑥 > 𝑦, 𝑥 ≥ 𝑦 |
| 𝑥 == 𝑦, 𝑥 ≠ 𝑦 |
| 𝑥 & 𝑦 |
| 𝑥 ^ 𝑦 |
| 𝑥 | 𝑦 |
| 𝑥 && 𝑦 |
| 𝑥 || 𝑦 |
| 𝑥 ? 𝑦 : 𝑧 |
| 𝑥 .. 𝑦 |
| 𝑥 = 𝑦, 𝑥 += 𝑦, 𝑥 −= 𝑦, 𝑥 ×= 𝑦, 𝑥 <<= 𝑦, 𝑥 >>= 𝑦 |

## Named expressions

### General

Operations and values in this document are sometimes specified in the form of named expressions. Exemplar named expressions are described in Table 5.

A named expression is a named macro-like statement. Every occurrence of a named expression is substituted by its definition when evaluated. The definition is provided by the ≔ operator.

Substitution is atomic. For example, the substitution for 3 × ExAPlusB is equivalent to 3 × ( 𝑎 + 𝑏 ), not ( 3 × 𝑎 ) + 𝑏.

The definition for a named expression is immutable. For example, ExTwo is equivalent to the value 2; unlike a variable it cannot be modified. All instances of ExTwo could be substituted by the numeric value 2.

The substitution for a named expression can be a variable. The substituted variable in such cases can be modified. For example, ExVar++ increments the variable Var.

Table 5 — Examples of named expressions

| Example | Remarks |
| --- | --- |
| ExTwo := 2 | ExTwo is equivalent to the value 2 |
| ExAPlusB := a + b | ExAPlusB is equivalent to ( 𝑎 + 𝑏 ) |
| ExTwoIndirect := ExTwo | ExTwoIndirect is equivalent to ExTwo |
| Var = 2  ExVar := Var | ExVar is equivalent to (an alias of) the variable Var |
| ExTimesTwo[i] := 2 × i | ExTimesTwo[ 𝑗 + 1 ] is equivalent to 2 × ( 𝑗 + 1 ) |
| ExSquared[i] := i × i | ExSquared[ ExVar++ ] is equivalent to Var × Var, with Var incremented after the evaluation of ExSquared |
| for (Var = 0; Var ≤ 10; Var++)  sum += ExTimesTwo[Var] | sum is incremented, in total, by 110 |
| ExWhere[i] := ExTimesTwo[inner]  where  inner := i + 1 | ExWhere[ 𝑗 ] is equivalent to 2 × ( 𝑗 + 1 ) |
| ExSumA[i] := i > 0  ? i + ExSumA[i − 1]  : 0 | Recursive definition.  ExSumA[ 10 ] evaluates to 55 |
| ExSumB[i] :=  ExSumB = 0  for (; i > 0; i−−)  ExSumB += i | Imperative definition using multiple statements.  ExSumB[ 10 ] evaluates to 55 |
| ExSum10[expr] :=  ExSum10 = 0  for (i = 0; i ≤ 10; i++)  ExSum10 += expr[i] | ExSum10 applies expr to each 𝑖, 𝑖 ∈ 0 .. 10, summing the result.  ExSum10[ ExprTimesTwo ] evaluates to 110  ExSum10[ ExprSquared ] evaluates to 385 |

### Scope

The scope of a named expression is not affected by the relative order of its definition and use; a named expression can be referenced earlier in the document than its definition.

Named expressions identified by a capital initial are "global" definitions that apply to the whole document. They may be directly referenced in other subclauses.

Named expressions identified by a lower-case initial are "local" definitions that apply to the subclause in which they are defined.

If a global definition references a local definition in the same subclause, that local definition is used when the global definition is referenced in another subclause.

### Arguments of named expressions

The definition of an expression can be in terms of one or more parameters. Each parameter is enclosed in square brackets. For example, the definition ExTimesTwo[ 𝑖 ] has a single parameter 𝑖.

A named expression can be applied to one or more arguments. When the definition is substituted for a named expression, every instance of each parameter is replaced by the text of the corresponding argument.

Replacements are atomic. For example, ExTimesTwo[ 𝑗 + 1 ] is equivalent to 2 × ( 𝑗 + 1 ), not ( 2 × 𝑗 ) + 1.

### Sub-expressions

A definition can contain a where-clause that defines further named expressions. They apply only to the definition containing the where-clause. For example, the definition of ExWhere[ 𝑖 ] defines the sub-expression inner.

### Definitions with multiple statements

Some definitions cannot be succinctly expressed by a single statement. In such cases, a definition can consist of multiple statements. The evaluated value for the whole definition is specified by assignments or modifications to a variable with the same name as the named expression. For example, ExSumB.

### Textual definitions

Some definitions are provided by a descriptive equivalence in textual or tabular form. For example:

* "The expression Ex[ 𝑖 ] is specified by Table X (Value for Ex[ 𝑖 ])."
* "The value for the expression Ex is specified by Table X for each axis 𝑘."
* "The expression Ex is equivalent to the following [procedural code]."

## Variables, syntax elements and tables

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

In some cases the syntax tables use the values of variables derived from other syntax elements' values. Such variables appear in the syntax tables, or text, named by a mixture of lower- and upper-case letters and without any underscore characters. Variables with a capital initial are valid for the decoding of the current syntax structure and all dependent syntax structures. They may be used in the decoding process for later syntax structures without mentioning their origin. Variables with a lower-case initial are only used within the clause in which they are derived.

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

Functions that specify properties of the current position in the bitstream are referred to as syntax functions (7.2). These functions assume the existence of a bitstream pointer with an indication of the position of the next bit to be read by the decoding process from the bitstream. Syntax functions are described by their names, which are constructed as syntax element names and end with left and right round brackets including zero or more parameter names (for definition) or arguments (for usage), separated by commas (if more than one).

Functions that are not syntax functions (including mathematical functions specified in 5.10) are described by their names, which start with a capital initial, contain a mixture of lower- and upper-case letters without any underscore characters and end with left and right round brackets surrounding zero or more parameter names (for definition) or arguments (for usage), separated by commas (if more than one).

Arrays are sequences of values identified by a common name. Both syntax elements and variables can be arrays. Subscripts or square brackets are used to index an array.

Boolean true and false values are interchangeable with the integers 1 and 0, respectively; non-zero integers are equivalent to true.

# Point cloud format and relationship to coded and output representations

## General format

A point cloud is an unordered list of points representing geometry, optional attribute information and associated metadata. Geometry information describes the location of points in a three-dimensional Cartesian coordinate system. Attributes are typed properties of each point, such as its colour or reflectance. Metadata is information used to interpret the point cloud, the point geometry and the attribute data.

Each point in a point cloud is a tuple of a three-dimensional position and attribute values for every attribute present in the point cloud. All points shall have the same number of attributes in the same order.

Point cloud metadata may describe, for example, a geometric transformation used to map points to another coordinate system, spatial regions (tiles) within a point cloud, the identification of attribute types and how attribute values are interpreted.

An *N*-point point cloud is symbolically illustrated in Figure 1. Rows of are points. Each point comprises a position with components and values for the components of each attribute of to .

文本

中度可信度描述已自动生成

Figure 1 — A symbolic representation of a point cloud.

Two point clouds are identical if there exists a one-to-one mapping between points in the two point clouds. For example, a permutation of points (rows in ) preserves identity.

## Attributes

### General

An attribute comprises one or more components.

A point cloud, unless otherwise specified, may contain more than one instance of a particular attribute. The significance or interpretation of multiple instances of the same attribute is unspecified.

Metadata can be associated with each attribute instance. Attribute metadata conveys sequence level characteristics such as an attribute label identifier or a frame level interpretation such as an attribute scale and offset information.

### Colour

The colour attribute specifies the colour of a point. The attribute shall comprise one of the following configurations:

* A luma () component (monochrome).
* A luma component and two chroma components ( or ).
* Green, blue and red components (, also known as *RGB*).
* Other unspecified monochrome or tri-stimulus colour systems (e.g., *YZX*, also known as *XYZ*).

The actual colour representation method in use is described using ISO/IEC 23091‑2 coding-independent video code points and is indicated using syntax specified in 7.3.2.7.

The ordering of attribute components is specified by Table 6.

Table 6 — Relationship between colour components and attribute components

| Colour representation | Attribute component index | | |
| --- | --- | --- | --- |
| 0 | 1 | 2 |
| Monochrome |  | – | – |
|  |  |  |  |
|  |  |  |  |
| or RGB |  |  |  |
| YZX or XYZ | *Y* | *Z* | *X* |

### Opacity

Opacity is a single component attribute. When normalized to the interval [ 0, 1 ], the value 0 indicates that a point is completely transparent and the value 1 indicates complete opacity. The opacity attribute may be used to control colour blending when rendering a colour attribute.

Opacity is often called an alpha channel or transparency.

### Reflectance

Reflectance is a single component attribute that represents the ratio of incident light reflected by a point; it is a dimensionless quantity. Values are bounded by a minimum that indicates complete absorption and a maximum that indicates complete reflection or saturation.

### Normal vector

A normal vector is a three-component attribute representing a vector perpendicular to the surface tangent plane at an associated point. The axes identification of the normal vectors is identical to that of the STV axes for the coded point cloud geometry. The length of a normal vector is not required to be one.

Normal vectors may be used when rendering a point cloud. A point's appearance may be modified according to the difference between the incident light direction and its normal vector.

### Material identifier

A material identifier is a single component attribute that associates a point with a material from a range of materials. Points with a common material identifier share a characteristic that may be used to identify an object or type of object. Materials are not specified by this document.

### Frame number/index

The frame number and frame index attributes are single component attributes that indicate how a point cloud frame may be partitioned into one or more ordered sub-frames. Each sub-frame is a partial representation of a point cloud frame, comprising points with the same frame number/index attribute value.

Sub-frame partitioning does not form part of the decoding or output processes specified by this document.

A point cloud sequence shall contain no more than one instance of a frame number/index attribute. A point cloud sequence shall not contain both frame number and frame index attributes.

The frame number attribute may be used to order all sub-frames over the entire point cloud sequence. Points from different point cloud frames shall not have the same value for the frame number attribute.

The frame index attribute may be used to order the sub-frames of a single point cloud frame.

An example of the relationship between frames, sub-frames and their ordering is shown in Table 7. The point cloud frames *a*, *b* and *c* are partitioned into sub-frames. Sub-frame orders are shown for the cases where the attribute is a frame number or a frame index.

Table 7 — Example partitioning of three consecutive frames a, b and c into sub-frames

|  | Frame | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *a* | | | *b* | | *c* | | |
| **Frame number attribute** | | | | | | | | |
| Sub-frame attribute value | 0 | 2 | 1 | 3 | 5 | 4 | 6 | 7 |
| Sub-frame presentation order | *a*0 | *a*1 | *a*2 | *b*3 | *c*4 | *b*5 | *c*6 | *c*7 |
| **Frame index attribute** | | | | | | | | |
| Sub-frame attribute value | 0 | 2 | 1 | 0 | 1 | 0 | 1 | 3 |
| Sub-frame presentation order | *a*0 | *a*1 | *a*2 | *b*0 | *b*1 | *c*0 | *c*1 | *c*3 |

### User defined attributes

The point cloud format supports attributes other than those specified in this document. A user defined attribute shall be identified by an international object identifier. The international object identifier shall either be assigned by a registration authority in accordance with Rec. ITU‑T X.660﻿ |​ ISO/IEC 9834‑1, or generated without registration using a universally unique identifier (UUID) as specified by Rec. ITU‑T X.667﻿ |​ ISO/IEC 9834‑8.

## Codec-derived attributes

### General

Codec-derived attributes represent values that are determined as side-effects of a processes specified in this document.

A decoder may, but is not required to, output one or more codec-derived attributes. Any codec-derived attributes output by a decoder shall conform to the definitions in 6.3.

### Slice identifier

The slice identifier attribute shall be a single component attribute that identifies the slice from which a point is decoded. Identification shall use the slice\_id syntax element value.

### Slice tag

The slice tag attribute shall be a single component attribute that identifies the group of slices from which a point is decoded. Identification shall use the slice\_tag syntax element value.

### Canonical point order

The canonical point order attribute shall be a single component attribute that specifies the order within a slice in which points are decoded by the geometry decoder as specified in this document.

Values of the point decoding order attribute shall be equal to ptIdx of the corresponding point PointPos[ ptIdx ] in a slice.

### Point Morton order

The point Morton order attribute shall be a single component attribute that specifies the order of points within a slice according to ascending values of Morton-coded STV slice position (i.e. prior to 8.3.6).

The Morton order shall be equivalent to the order of points in the finest detail level specified in 10.6.5.2 as if both attr\_canonical\_order\_enabled and attr\_coord\_conv\_enabled are both 0.

For example, if three points 𝑎, 𝑏 and 𝑐 in canonical point order are ordered { 𝑎, 𝑐, 𝑏 } in the finest detail level, then the respective values for the Morton order attribute are 0, 2 and 1.

## Coded point cloud format

### Sequence coordinate system

The sequence coordinate system is specified by the position of its origin in an externally defined application-specific coordinate system and by the length of its unit vectors.

All points in a coded point cloud sequence shall have non-negative coordinates in the sequence coordinate system.

A position in the sequence coordinate system is related to the position in the application coordinate system by the sequence origin SeqOrigin and the unit vector length SeqUnit specified by the active SPS:

The maximum bound on the sequence coordinate system depends upon the level to which the coded point cloud sequence conforms, as specified in Annex A.

An example sequence coordinate system (marked SCS) is illustrated in Figure 2. A point with an 𝑥-coordinate of 75 in the sequence coordinate system has a position in the application-specific coordinate system of 105 SeqUnit. If SeqUnit is 0,8 AppUnit, the 𝑥-coordinate of the point in the application-specific coordinate system (marked ACS) is 4,2.

图示

描述已自动生成

Key

|  |  |
| --- | --- |
| ACS | Application-specific coordinate system |
| CCS | Coding coordinate system |
| SCS | Sequence coordinate system |
| 𝑎:𝑏 | Ratio of SCS to CCS |

Figure 2 — Relationship between application, sequence, and coding coordinate systems.

### Coding coordinate system

The coding coordinate system is a non-negative integer coordinate system used to code point positions. It is either identical to, or a geometric contraction of the sequence coordinate system. Its origin is coincident with the sequence coordinate system origin.

A position in the coding coordinate system is related to the position in the sequence coordinate system by the binary fixed-point scale factor SeqCodedScale:

An example coding coordinate system (marked CCS) is illustrated in Figure 2. A point with an 𝑥-coordinate of 48 in the coding coordinate system has an 𝑥-coordinate, , of 75 when scaled by the scale factor .

Point position components in the coding coordinate system shall satisfy the following level dependent (Annex A) constraint:

Where SeqCodedScaleN and SeqCodedScaleD are the numerator and denominator, respectively, of SeqCodedScale when represented as an irreducible fraction:

This constraint guarantees that conversion from the coding coordinate system to the sequence coordinate system can be performed using 32-bit arithmetic.

### Coded point cloud sequence

The coded representation of a point cloud sequence comprises one or more point cloud frames encoded as a sequence of encapsulated DUs that convey syntax structures as specified in 7.3. An encapsulation format is specified by Annex B. Alternative encapsulation formats may be specified by the application.

The coded point cloud sequence shall include:

* An SPS that enumerates the attributes present in the coded point cloud format and conveys both metadata and decoding parameters that pertain to the whole coded point cloud sequence.
* Any GPSs that convey parameters required for the decoding of geometry data.
* Any APSs that convey parameters required for the decoding of attribute data.
* The slices comprising each coded point cloud frame.

### Coded point cloud frame

A coded point cloud frame comprises a sequence of zero or more slices with the same value of a notional frame counter FrameCtr (8.2.2). An optional frame boundary marker data unit explicitly signals the end of a frame.

It is a requirement of bitstream conformance that:

* Every coded point cloud frame shall have a unique value of FrameCtr within the sequence.
* Coded point cloud frames shall be ordered such that the notional frame counter increases for each successive coded point cloud frame when biprediction\_enabled is equal to 0.
* The value of of gdu\_temporal\_id of each GDU in the frame and adu\_temporal\_id of each ADU in the frame is equal to the same value.
* The decoding of current frame shall not depend on any point cloud frame F in the bitstream where the temporal ID of frame F is greater than the temporal ID of the current frame.

[Ed. (YZ): biprediction\_enabled is recommended to be moved to SPS from GPS.]

An empty frame shall be signalled by a frame boundary marker data unit without any preceding slices with the same value of FrameCtr.

A coded point cloud frame independently codes a single point cloud frame without dependencies upon any previous or subsequent point cloud frame.

A decoded point cloud frame is the concatenation of all points in all constituent slices of the frame.

Unless prohibited by an SPS constraint, coincident points in a point cloud frame may arise from:

* points coded in a single slice with a non-zero duplicate point count;
* distinct points with the same position in a single slice; or
* the concatenation of multiple slices.

### Slice of a coded point cloud frame

Every slice shall include a GDU that codes the slice geometry and ADUs or defaulted attribute DUs that code the slice attributes. A slice is identified by the GDU slice\_id.

The slice geometry is coded in the slice's coordinate system. The bounding boxes of slices may intersect, including within a single frame.

A slice shall start with a GDU. This GDU may be followed by optional redundant GDUs that duplicate the slice geometry. ADUs and defaulted attribute DUs shall occur after all GDUs in the slice. DUs belonging to different slices shall not be interleaved.

Within a slice, other DUs may be present. For example, an APS can occur within a slice to convey parameters for attribute decoding.

It is a requirement of bitstream conformance that:

* All GDUs present in a slice shall reconstruct the same geometry in the same canonical point order.
* Every slice shall have a corresponding ADU or defaulted attribute DU for every attribute enumerated in the SPS.
* All ADUs present in a slice with the same value of adu\_sps\_attr\_idx shall reconstruct the same attribute values.

Only one GDU in a slice shall be decoded; all others shall be ignored when decoding (removed from the bitstream and discarded). A decoder shall choose which GDU is decoded.

ADU parsing depends upon certain GDU header parameters. ADU decoding depends upon the reconstructed slice geometry.

Slices are either independent or dependent. An independent slice does not require any other slice to be decoded first. A dependent slice requires that the immediately preceding slice in bitstream order is decoded first. A slice shall be directly depended upon by no more than a single dependent slice.

A dependent slice shall not depend upon a slice in a different point cloud frame.

### Repetition of slices

Slices may be repeated within a coded point cloud frame. Repetition shall not change the value of slice\_id.

A slice set is the set of slices with the same value of slice\_id within a coded point cloud frame.

It is a requirement of bitstream conformance that all slices in each slice set shall reconstruct the same points in the same canonical order.

From each slice set, only one slice shall be decoded; all others shall be ignored for decoding (removed from the bitstream and discarded). A decoder shall choose which slice is decoded.

### Relationship between tiles and slices

A group of slices can be identified by a common slice tag identifier (slice\_tag).

The tile inventory DU provides a means to associate a bounding box with a group of slices. Each tile comprises a single bounding box and an identifier (tileId). Tile bounding boxes may overlap. Implementations can use a tile inventory to aid spatial random access.

When a tile inventory is present in the bitstream, slice\_tag shall identify a tile by its tileId. Otherwise, the use of slice tags is application specific.

When a slice tag identifies a tile, a dependent slice should not depend upon a slice in a different tile. To do otherwise can prevent decoding of individual tiles (for example, in spatial random access decoding).

Tile information is not used by the decoding processes specified in this document.

图示, 工程绘图

描述已自动生成

Key

|  |  |
| --- | --- |
|  | Slice 𝑛, associated with tile 𝑡 |
|  | Bounding box of tile 𝑡 |

Figure 3 — Example arrangement of tiles and slices.

An example arrangement of tiles and slices within a coded point cloud frame is shown in Figure 3. Slices and are associated with tile and slices , and are associated with tile ; the bounding box of does not include . A decoder that performs spatial random access to decode a region (not shown) can use the tile inventory to determine tile IDs for the set of tiles that intersect . Only slices with matching slice tags would need to be decoded. Since the slice is not included in the bounding box of tile , if intersects but not , the slice is not discoverable using the tile inventory. However, in the case that and intersect, would have a matching slice tag.

### Parameter sets

#### Activation of parameter sets

The parameters contained in an SPS, GPS or APS shall not have any effect until the activation of the respective parameter set.

At most one SPS, GPS and APS are active at any given moment during the decoding process. The activation of a parameter-set shall deactivate any previously active parameter set of the same type.

At the start of a coded point cloud sequence, no parameter sets are active.

An SPS shall be activated by the parsing of a GDU. Once activated, it shall remain active for the whole of the coded point cloud sequence.

A GPS shall be activated by the parsing of a GDU.

An APS shall be activated by the parsing of an ADU.

Other DUs that contain references to SPS, GPS or APS DUs do not cause the referenced parameter-set to be activated.

#### Order of parameter sets

DUs shall be conveyed to a decoder in an order such that any parameter-set to be activated is available prior to the point of activation.

#### Duplication of parameter sets

Parameter-set DUs may be repeated at any point in the coded point cloud sequence.

All parameter-set DUs with the same parameter-set identifier shall be identical for the duration of the coded point cloud sequence.

Parameter-set identifiers are distinct for each type of parameter set.

## Output point cloud format

### General

Point cloud frames decoded from a G-PCC bitstream shall be output in the output point cloud format (6.5).

### Coordinate system

A decoder shall output points in the sequence coordinate system.

The output point cloud format shall indicate the sequence origin SeqOrigin and the sequence unit SeqUnit as point cloud metadata.

### Fixed-point conformance output

A decoder that is configured to output 𝑛-fractional-bit fixed-point positions shall round half-values of away from zero prior to output as :

### Attributes

Attribute values shall be interpreted according to the semantics of the attribute type and any per-sequence or frame-specific attribute properties. For example, if a frame-specific scale and offset property is present for an attribute, the output attribute values for that frame would be interpreted according to 7.4.2.2.5.

### Output point cloud sequence

Decoding a conforming G-PCC bitstream generates a sequence of output point cloud frames. Output point clouds frames are output in the order of FrameCtr of each point cloud frame within the sequence.

### Output point cloud frame

Each output point cloud frame is specified in terms of the following state variables:

* The variable RecCloudPointCnt, the cumulative number of points in the output point cloud frame.
* The array RecCloudPos of decoded point positions; RecCloudPos[ ptIdx ][ 𝑘 ] is the 𝑘-th coordinate of the ptIdx-th output point in the coding coordinate system.
* The array RecCloudAttr of decoded point attributes; RecCloudAttr[ ptIdx ][ attrIdx ][ 𝑐 ] is the 𝑐-th component of the identified attribute for the ptIdx-th point. Attributes are identified by the index attrIdx into the active SPS attribute list.

Decoder implementations may output points in a different order to the canonical order specified by this document.

Immediately prior to outputting the decoded point cloud frame, point positions shall be converted to the sequence coordinate system.

Each decoded point cloud frame shall be stored in buffer. One decoded point cloud frame stored in buffer shall be output when it is the first decoded point cloud frame or the FrameCtr of it is equal to the FrameCtr of the previously one output point cloud frame plus 1. The buffer for one decoded point cloud frame shall be released when the point cloud frame does not serve as the reference frame for any other point cloud frames to be decoded.