INTERNATIONAL STANDARD

ISO/IEC 23090-9

First edition
2023-03

Information technology — Coded representation of immersive media —

Part 9: Geometry-based point cloud compression

Technologies de l'information — Représentation codée de média immersifs —

Partie 9: Compression des nuages de points basée sur la géométrie

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

Reference number
ISO/IEC 23090-9:2023(E)

© ISO/IEC 2023

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2023

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

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

© ISO/IEC 2023 – All rights reserved

Contents

Page

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Foreword

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

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

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

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

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

A list of all parts in the ISO/IEC 23090 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Introduction

Advancements in 3D capturing and rendering technologies are enabling new applications and services in the fields of assisted and autonomous driving, cartography, cultural heritage, industrial processes, immersive real-time communication, and virtual/augmented/mixed reality (VR/AR/MR) content creation, transmission and communication. Point clouds have arisen as one of the main representations for such applications. A point cloud frame consists of a set of 3D points. Every point, in addition to having a 3D position, may also be associated with numerous other attributes such as colour, transparency, reflectance, timestamp, surface normal and classification. Such representations require a large amount of data, which can be costly in terms of storage and transmission. This document provides the method for efficiently compressing point cloud representations.

The International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he/she is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with ISO and IEC. Information may be obtained from the patent database available at www.iso.org/patents or https://patents.iec.ch.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those in the patent database. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Information technology — Coded representation of immersive media —

Part 9: Geometry-based point cloud compression

1 Scope

This document specifies geometry-based point cloud compression.

2 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.

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

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

Rec. ITU-T X.667 | ISO/IEC 9834-8, *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*

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

3 Terms and definitions

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

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

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1 General terms

3.1.1
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* (3.1.19)

3.1.2
point cloud
unordered list of *points* (3.1.1)

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

3.1.3
point cloud sequence
sequence of one or more *point clouds* (3.1.2)

3.1.4
point cloud frame
point cloud (3.1.2) in a *point cloud sequence* (3.1.3)

3.1.5
coded point cloud frame
coded representation of a *point cloud frame* (3.1.4)

3.1.6
canonical point order
order of *points* (3.1.1) decoded from a *slice* (3.1.21) according to the decoding processes

Note 1 to entry: The decoding processes are specified in Clause 9.

3.1.7
bounding box
axis-aligned cuboid defining a spatial region that bounds a set of *points* (3.1.1)

3.1.8
Cartesian coordinates
three scalar multiples of respective orthogonal *XYZ* (3.1.11) unit vectors with finite precision and bounds that specify a position relative to a fixed reference

3.1.9
angular coordinates
position specified as the radial distance ρ from the V axis, an azimuth angle φ in the S-T plane and an indexed elevation

3.1.10
attribute coordinates
STV (3.1.12) or scaled *RPI* (3.1.13) point coordinates used to code an *attribute* (3.1.19)

3.1.11
XYZ
XYZ axes
X, Y and Z axes, in that order, used to represent *Cartesian coordinates* (3.1.8)

3.1.12
STV
STV axes
S, T and V axes, in that order, that are a sequence-dependent permutation of the *XYZ axes* (3.1.11); used to represent the coded *geometry* (3.1.18)

3.1.13
RPI
RPI axes
R, P and I axes, in that order, used to represent *angular coordinates* (3.1.9)

3.1.14
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

3.1.15
coding coordinate system
scaled *sequence coordinate system* (3.1.14) that applies to an entire coded *point cloud sequence* (3.1.3) and in which all *points* (3.1.1) have non-negative integer coordinates

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

3.1.16
slice coordinate system
translated *coding coordinate system* (3.1.15) that applies to a single *slice* (3.1.21) and in which all *points* (3.1.1) in the slice have non-negative integer coordinates

3.1.17
beam
sampler of point positions using *angular coordinates* (3.1.9) by rays cast with a fixed elevation and from a point on and rotating around the V axis at the angular origin

3.1.18
geometry
point positions (3.4.1) associated with a set of *points* (3.1.1)

3.1.19
attribute
scalar or vector property associated with each *point* (3.1.1) in a *point cloud* (3.1.2)

EXAMPLE Colour, reflectance, frame index.

3.1.20
position
<bit> bit in a binary string or value, representing the factor $2^{position}$

EXAMPLE The LSB has bit position 0.

3.1.21
slice
geometry (3.1.18) and *attributes* (3.1.19) for part of, or an entire, *coded point cloud frame* (3.1.5)

Note 1 to entry: the *bounding boxes* (3.1.7) of any two slices can intersect.

3.1.22
tile
set of *slices* (3.1.21) identified by a common slice_tag *syntax element* (3.2.16) value whose *geometry* (3.1.18) should be contained within a *bounding box* (3.1.7) specified in a tile inventory *data unit* (3.2.13)

3.1.23
Morton code
non-negative integer obtained by interleaving the bits of three integers

3.1.24
Morton order
order of elements according to their *Morton code* (3.1.23)

3.1.25
sparse array
array with fewer set elements than total addressable elements

Note 1 to entry: Unset elements can have an inferred value when accessed.

3.2 Terms related to high-level syntax and entropy coding

3.2.1
ASN.1
abstract syntax notation one
notation specified by Rec. ITU-T X.680 | ISO/IEC 8824-1 that is used for the definition of data types, values and constraints on data types

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

3.2.2
bin
binary symbol (bit) of the *binarized* (3.2.3) representation of a *syntax element*'s (3.2.16) value

3.2.3
binarization
specification of a *syntax element*'s (3.2.16) value as a sequence of *bins* (3.2.2)

3.2.4
bypass
<contextual probability model> static, equiprobable probability model

3.2.5
bypass symbol
bypass-contextualized (3.2.4) *bin* (3.2.2)

3.2.6
bypass stream
sequence of *bypass symbols* (3.2.5) that are not encoded in an arithmetic-coded *bitstream* (3.2.7)

3.2.7
bitstream
<data> sequence of bits

3.2.8
bitstream
<coded sequence> sequence of bits, in the form of encapsulated *data units* (3.2.13), that represents a coded *point cloud sequence* (3.1.3)

3.2.9
set bit
bit with the value 1

3.2.10
unset bit
bit with the value 0

3.2.11
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.

3.2.12
byte aligned
bitstream (3.2.7) position that is an integer multiple of eight bits from the position of the first bit in the bitstream

3.2.13
data unit
DU
sequence of *bytes* (3.2.11) conveying a single *syntax structure* (3.2.17) of known length

3.2.14
data unit header
parameters, located from the start of a *data unit* (3.2.13)

3.2.15
data unit footer
parameters, located from the end of a *data unit* (3.2.13)

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

3.2.16
syntax element
element of data represented in the *bitstream* (3.2.7)

3.2.17
syntax structure
zero or more *syntax elements* (3.2.16) present together in the *bitstream* (3.2.7) in a specified order

3.2.18
parameter set
collection of parameters that apply when activated

3.2.19
sequence parameter set
SPS
parameters for an entire coded *point cloud sequence* (3.1.3), conveyed by an SPS *data unit* (3.2.13) and activated when referenced by a *geometry* (3.1.18) data unit

3.2.20
geometry parameter set
GPS
parameters for the coding of *slice* (3.1.21) *geometry* (3.1.18), conveyed by a GPS *data unit* (3.2.13) and activated when referenced by a geometry data unit

3.2.21
attribute parameter set
APS
parameters for the coding of a *slice* (3.1.21) *attribute* (3.1.19), conveyed by an APS *data unit* (3.2.13) and activated when referenced by an attribute data unit

3.2.22
object identifier
OID
<ASN.1> ordered list of primary integer values from the root of the *international object identifier tree* (3.2.23) to a node, which unambiguously identifies that node

[SOURCE: Rec. ITU-T X.660 | ISO/IEC 9834-1:2012, 3.5.11, modified — The abbreviated term "OID" and the domain "<ASN.1>" have been added.]

3.2.23
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* (3.2.25) responsible for allocating arcs from a *parent node* (3.3.10)

[SOURCE: Rec. ITU-T X.660 | ISO/IEC 9834-1:2012, 3.5.5]

3.2.24
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 9834-1:2012, 3.5.16, modified — The domain "<object identifier>" has been added.]

3.2.25
registration authority
<international object identifier tree> entity such as an organization, a standard or an automated facility that performs *registration* (3.2.24) of one or more types of objects

[SOURCE: Rec. ITU-T X.660 | ISO/IEC 9834-1, 3.5.17, modified — The domain "<international object identifier tree>" has been added.]

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

3.2.26
application specific
defined by an application or an application standard

3.2.27
unspecified
having no specified meaning in this document and will not have a specified meaning in future editions of this document

Note 1 to entry: It is used to specify values of a particular *syntax element* (3.2.16).

3.3 Terms related to tree structure

3.3.1
tree
recursive structure of *nodes* (3.3.7) without loops and containing a single *root node* (3.3.5)

3.3.2
top
<tree> *tree level* (3.3.4) with a *depth* (3.3.8) of 0, consisting of the *root node* (3.3.5)

3.3.3
bottom
<tree> *tree level* (3.3.4) with the greatest *depth* (3.3.8)

3.3.4
tree level
set of *nodes* (3.3.7) at the same *depth* (3.3.8) in a *tree* (3.3.1)

3.3.5
root node
node (3.3.7) without a *parent node* (3.3.10)

3.3.6
leaf node
terminal *node* (3.3.7) without any *child nodes* (3.3.9)

3.3.7
node
element of a *tree* (3.3.1)

3.3.8
depth
<node> number of descendent hops from the *root node* (3.3.5) to a *node* (3.3.7)

3.3.9
child node
direct descendent of a *node* (3.3.7)

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

3.3.11
grandparent node
direct ancestor of a *node*'s (3.3.7) *parent node* (3.3.10)

3.3.12
great-grandparent node
direct ancestor of a *node*'s (3.3.7) *grandparent node* (3.3.11)

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

3.3.13
sibling nodes
nodes (3.3.7) that are *child nodes* (3.3.9) of the same *parent node* (3.3.10)

3.3.14
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)

3.3.15
subtree root node
single *node* (3.3.7) of a *subtree* (3.3.14) from which all other nodes in the same subtree are descendents

3.4 Terms related to geometry coding

3.4.1
position
<point> three-dimensional coordinates of a *point* (3.1.1)

3.4.2
occupancy tree
eight-ary *tree* (3.3.1) of *occupancy tree nodes* (3.4.4) representing the *geometry* (3.1.18) of a *slice* (3.1.21)

3.4.3
predictive tree
tree (3.3.1) of *predictive tree nodes* (3.4.5) representing the *geometry* (3.1.18) of a *slice* (3.1.21)

3.4.4
occupancy tree node
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)

3.4.5
predictive tree node
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)

3.4.6
direct node
terminal *occupancy tree node* (3.4.4) that codes one or more *point positions* (3.4.1)

3.4.7
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)

3.4.8
occupied neighbourhood pattern
pattern that indicates the existence and arrangement of the six possible *occupancy tree nodes* (3.4.4) that share faces with a central node

3.5 Terms related to attribute coding

3.5.1
primary attribute component
first, or only, *attribute* (3.1.19) component, identified by the index 0

3.5.2
secondary attribute component
attribute (3.1.19) component other than the first component, identified by an index greater than 0

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

3.5.3
detail level
set of *points* (3.1.1) that represent a subsampled version of the *slice* (3.1.21) *geometry* (3.1.18)

3.5.4
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

3.5.5
refinement point
point (3.1.1) in a *refinement list* (3.5.4)

3.5.6
predictor set
set of neighbouring *points* (3.1.1) from which an *attribute* (3.1.19) value is predicted

4 Abbreviated terms

ADU	Attribute data unit
APS	Attribute parameter set
attr	Attribute
CBS	Chunked bytestream
cnt	Count
CPM	Contextual probability model
DU	Data unit
EGk	Exponential Golomb code of order *k*
fbdu	Frame boundary marker data unit
FL	Fixed-length code
FL+S	Fixed-length code plus conditional sign bit
FSAP	Frame-specific attribute properties
GDU	Geometry data unit
geom	Geometry
GPS	Geometry parameter set
G-PCC	Geometry-based point cloud compression
idx	Index
LoD	Level(s) of detail
LSB	Least significant bit
MSB	Most significant bit
NA	Not applicable

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

occtree	Occupancy tree
occ	Occupancy tree node
ptree	Predictive tree
ptn	Predictive tree node
QP	Quantization parameter
RAHT	Region-adaptive hierarchical transform
seq	Sequence
SPS	Sequence parameter set
ti	Tile inventory
tlv	Type-length-value
TU	Truncated unary code

5 Conventions

5.1 General

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

5.2 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 point order

— *rfmtIdx*: index of a point in an array of LoD refinement points

— *ni*: index for an element in a point's predictor set

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

5.3 Numerical representation

binary	typeset as 'X...XX' where each base 2-digit X is 0 or 1
octal	typeset as $X...XX_8$ 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

5.4 Arithmetic operators

+	Addition
−	Subtraction (as a two-argument operator) or negation (as a unary prefix operator)
×	Multiplication
x^y	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
$\frac{x}{y}$	Division in mathematical equations where no truncation or rounding is intended
$\sum_{i=x}^{y} f(i)$	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 $x \geq 0$ and $y > 0$

5.5 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

5.6 Relational operators

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

5.7 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.

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

5.8 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 $x = x << 1$

5.9 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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

5.10 Mathematical functions

5.10.1 General

ArcSin(x)	Trigonometric arc sine function
Abs(x)	$\begin{cases} x & ; \quad x \geq 0 \\ -x & ; \quad x < 0 \end{cases}$
Bit(x, i)	($x >> i$) & 1
Ceil(x)	Lowest integer greater than or equal to x
Clip3(min, max, x)	$\begin{cases} \text{min} & ; \quad x < \text{min} \\ \text{max} & ; \quad x > \text{max} \\ x & ; \quad \text{otherwise} \end{cases}$
DivExp2Floor(x, p)	Floor($x \div 2^p$)
DivExp2Fz(x, p)	RoundFz($x \div 2^p$)
DivExp2Up(x, p)	RoundUp($x \div 2^p$)
Exp2(p)	2^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
IntLog2(x)	Floor(Log(x) ÷ Log(2))
Log(x)	Natural logarithm of the argument x
Min(x, y)	$\begin{cases} x & ; \quad x \leq y \\ y & ; \quad x > y \end{cases}$
MinVec(x)	$\begin{cases} x_0 & ; \quad x_0 \leq \text{Min}(x_1, x_2) \\ x_1 & ; \quad x_1 < \text{Min}(x_0, x_2) \\ x_2 & ; \quad x_2 < \text{Min}(x_0, x_1) \end{cases}$
Max(x, y)	$\begin{cases} x & ; \quad x \geq y \\ y & ; \quad x < y \end{cases}$
MaxVec(x)	$\begin{cases} x_0 & ; \quad x_0 \geq \text{Max}(x_1, x_2) \\ x_1 & ; \quad x_1 > \text{Max}(x_0, x_2) \\ x_2 & ; \quad x_2 > \text{Max}(x_0, x_1) \end{cases}$
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)	$\begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x == 0 \\ -1 & ; \quad x < 0 \end{cases}$
Sin(x)	Trigonometric sine function
Sqrt(x)	$\sqrt{x}$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

5.10.2 IntAtan2

The function *theta* = IntAtan2(*y*, *x*) is a 20-bit fixed-point approximation of the arc tangent of *y* ÷ *x* 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.

$$ArcSinFp[x] := \left\lfloor 2^{20}\ \mathrm{ArcSin}\left(\frac{x}{2^{9}}\right)+\frac{1}{2}\right\rfloor$$

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
```

5.10.3 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 $2^{-piBits}$ 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)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

Domain	IntCos		IntSin	
	sgn	*theta*	*sgn*	*theta*
$\theta \le -pi$	−1	$pi/2$	0	0
$-pi < \theta < -pi/2$	−1	$-\theta - pi/2$	−1	$pi + \theta$
$-pi/2 \le \theta < 0$	1	$\theta + pi/2$	−1	$-\theta$
$0 \le \theta < pi/2$	1	$-\theta + pi/2$	1	θ
$pi/2 \le \theta < pi$	−1	$\theta - pi/2$	1	$pi - \theta$
$\theta \ge pi$	−1	$pi/2$	0	0

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

$$SinFp[x] := \left\lfloor 2^{24}\ \mathrm{Sin}\left(\frac{\pi\, x}{2^{11}}\right) + \frac{1}{2} \right\rfloor$$

5.10.4 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 2^{46}, the calculation uses a quantized value of *x* to ensure computability using 64-bit arithmetic.

NOTE 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
```

5.10.5 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*].

NOTE IntRecipSqrt(0) is 0.

The parameter *x* is scaled to be in the range $[2^{30}, 2^{32} - 1]$ 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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
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)

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
NOTE	Values are typeset in hexadecimal form without the 0x prefix.							

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

5.10.6 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
```

5.10.7 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*.

© ISO/IEC 2023 – All rights reserved

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*.

$$morton := \sum_{i=0} 2^{3i+2}\,\mathrm{Bit}(s,i) + 2^{3i+1}\,\mathrm{Bit}(t,i) + 2^{3i}\,\mathrm{Bit}(v,i)$$

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

```
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
s_n … s_1 s_0	t_n … t_1 t_0	v_n … v_1 v_0	s_n t_n v_n … s_1 t_1 v_1 s_0 t_0 v_0	…

5.10.8 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*.

$$s := \sum_{i=0} 2^{i}\,\mathrm{Bit}(m,3i+2)$$

$$t := \sum_{i=0} 2^{i}\,\mathrm{Bit}(m,3i+1)$$

$$v := \sum_{i=0} 2^{i}\,\mathrm{Bit}(m,3i)$$

5.11 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

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

Table 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
$x \times y$, x / y, $x \div y$, $\frac{x}{y}$, $x \,\%\, y$
$x + y$, $x - y$ (as a two-argument operator), $\sum_{i=x}^{y} f(i)$
$x << y$, $x >> y$
$x < y$, $x \leq y$, $x > y$, $x \geq y$
$x == y$, $x \neq y$
$x \,\&\, y$
$x \,\hat{}\, y$
$x \mid y$
$x \,\&\&\, y$
$x \mid\mid y$
$x \,?\, y : z$
$x \,..\, y$
$x = y$, $x \mathrel{+}= y$, $x \mathrel{-}= y$, $x \mathrel{\times}= y$, $x \mathrel{<<}= y$, $x \mathrel{>>}= y$

5.12 Named expressions

5.12.1 General

Operations and values in this document are sometimes specified in the form of named expressions. Exemplar named expressions are described in Table 5. An index of all named expressions is provided in Annex E.

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*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

5.12.2 Scope of a named expression

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.

5.12.3 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

5.12.4 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.

5.12.5 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*.

5.12.6 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]."

5.13 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.

NOTE 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

6 Point cloud format and relationship to coded and output representations

6.1 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 $\mathcal{P}$ is symbolically illustrated in Figure 1. Rows of $\mathcal{P}$ are points. Each point $\mathcal{P}_n$ comprises a position with components $p_{n,k}$ and values for the components $a^m_{n,c}$ of each attribute A_m of A_1 to A_M.

$$\mathcal{P} = \begin{pmatrix} \overbrace{p_{1,k}}^{P} & \overbrace{a^1_{1,c}}^{A_1} \cdots \overbrace{a^M_{1,c}}^{A_M} \\ p_{2,k} & a^1_{2,c} \quad a^M_{2,c} \\ \vdots & \vdots \quad \vdots \\ p_{N,k} & a^1_{N,c} \quad a^M_{N,c} \end{pmatrix}$$

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 $\mathcal{P}$) preserves identity.

6.2 Attributes

6.2.1 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.

6.2.2 Colour

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

— a luma (Y') component (monochrome);

— a luma component and two chroma components ($Y'C_BC_R$ or $Y'C_GC_O$);

— green, blue and red components ($G'B'R'$, also known as RGB);

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— other unspecified monochrome or tri-stimulus colour systems (e.g. *YZX*, also known as *XYZ*).

The colour representation should be described using ISO/IEC 23091-2 coding-independent video code points and the 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	Y'	–	–
$Y'C_BC_R$	Y'	C_B	C_R
$Y'C_GC_O$	Y'	C_G	C_O
$G'B'R'$ or RGB	G'	B'	R'
YZX or XYZ	Y	Z	X

6.2.3 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.

NOTE Opacity is often called an alpha channel or transparency.

6.2.4 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.

6.2.5 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.

6.2.6 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.

6.2.7 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.

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

6.2.8 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.

6.3 Codec-derived attributes

6.3.1 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.

6.3.2 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.

6.3.3 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.

6.3.4 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 specified in Clause 9.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

6.3.5 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.

6.4 Coded point cloud format

6.4.1 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 P_{SCS} in the sequence coordinate system is related to the position P_{ACS} in the application coordinate system by the sequence origin *SeqOrigin* and the unit vector length *SeqUnit* specified by the active SPS:

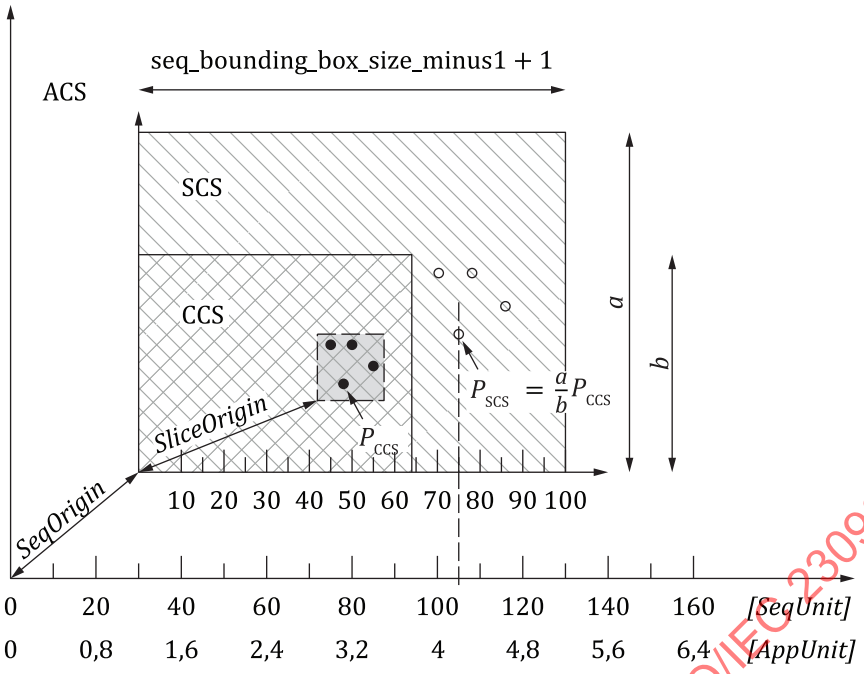
$$P_{ACS} = (P_{SCS} + SeqOrigin) \times SeqUnit$$

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 P_{SCS} 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved



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

6.4.2 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 P_{CCS} in the coding coordinate system is related to the position P_{SCS} in the sequence coordinate system by the binary fixed-point scale factor *SeqCodedScale*:

$$P_{SCS} = P_{CCS} \times SeqCodedScale$$

An example coding coordinate system (marked CCS) is illustrated in Figure 2. A point P_{CCS} with an *x*-coordinate of 48 in the coding coordinate system has an *x*-coordinate, P_{SCS}, of 75 when scaled by the scale factor $SeqCodedScale = a \div b = 25 \div 16$.

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

$$P_{SCS} \times SeqCodedScaleN < \mathrm{Min}\left(2^{MaxSeqBboxDimLog2}, 2^{32} - \frac{1}{2} SeqCodedScaleD\right)$$

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

$$\frac{SeqCodedScaleN}{SeqCodedScaleD} = SeqCodedScale, \qquad Gcd(SeqCodedScaleN, SeqCodedScaleD) = 1$$

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

6.4.3 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. Unless specified by external means, the encapsulation format shall be as specified by Annex B.

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.

6.4.4 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.

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.

6.4.5 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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.

6.4.6 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 point 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.

6.4.7 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.

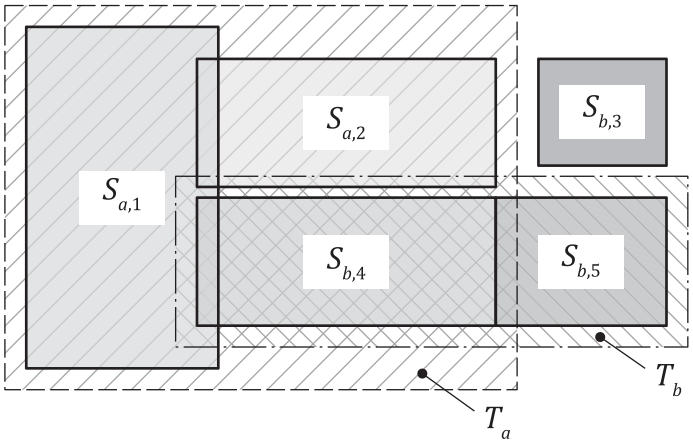
When a tile inventory is present in the bitstream, slice_tag shall identify a tile by its *tileId*. Otherwise, the use of slice tags is application specific.

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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved



Key

$S_{t,n}$	slice n, associated with tile t
T_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 $S_{a,1}$ and $S_{a,2}$ are associated with tile T_a and slices $S_{b,3}$, $S_{b,4}$ and $S_{b,5}$ are associated with tile T_b; the bounding box of T_b does not include $S_{b,3}$. A decoder that performs spatial random access to decode a region R (not shown) can use the tile inventory to determine tile IDs for the set of tiles that intersect R. Only slices with matching slice tags would need to be decoded. Since the slice $S_{b,3}$ is not included in the bounding box of tile T_b, if R intersects $S_{b,3}$ but not T_b, the slice is not discoverable using the tile inventory. However, in the case that R and T_b intersect, $S_{b,3}$ would have a matching slice tag.

6.4.8 Parameter sets

6.4.8.1 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.

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

6.4.8.2 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.

6.4.8.3 Duplication of parameter sets

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

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

6.5 Output point cloud format

6.5.1 General

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

6.5.2 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.

6.5.3 Fixed-point conformance output

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

$$P_{OUT} = \text{RoundFz}\left(P_{CCS} \times SeqCodedScale \times 2^{n}\right) \div 2^{n}$$

6.5.4 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.

6.5.5 Output point cloud sequence

Decoding a conforming G-PCC bitstream generates a sequence of output point cloud frames.

6.5.6 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 point order.

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

7 Syntax and semantics

7.1 Method of specifying syntax in tabular form

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

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

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

/* A "for" structure specifies evaluation of an initial statement, followed by a test of a condition, and if the condition is true, specifies repeated evaluation of a primary statement followed by a subsequent statement until the condition is no longer true. */	
for(initial statement; condition; subsequent statement)	
primary statement	

7.2 Specification of syntax functions and descriptors

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

byte_aligned() is specified as:

— If the next bit in the bitstream is the first bit in a byte (*DataUnitReadIdx* % 8 == 0), the value of byte_aligned() is true.

— Otherwise, the value of byte_aligned() is false.

more_data_in_data_unit() is specified as:

— If parsing of the DU is incomplete (*DataUnitReadIdx* / 8 < *DataUnitLength*), the value of more_data_in_data_unit() is true.

— Otherwise, the value of more_data_in_data_unit() is false.

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

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

— ae(v): adaptive arithmetic entropy-coded syntax element.

— de(v): dictionary coded syntax element.

— oid(v): an ASN.1 object identifier.

— s(*n*): signed integer using an *n*-bit magnitude and a sign bit.

— se(v): signed integer 0-th order Exp-Golomb-coded syntax element.

— u(*n*): unsigned integer using *n* bits. When *n* is "v" in the syntax table, the number of bits varies in a manner dependent upon the value of other syntax elements.

— ue(v): unsigned integer 0-th order Exp-Golomb-coded syntax element.

7.3 Syntax in tabular form

7.3.1 General

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

7.3.2 Parameter sets, ancillary data and byte alignment

7.3.2.1 Sequence parameter set data unit syntax

seq_parameter_set() {	**Descriptor**
simple_profile_compliant	u(1)
dense_profile_compliant	u(1)
predictive_profile_compliant	u(1)
main_profile_compliant	u(1)
reserved_profile_18bits	u(18)
slice_reordering_constraint	u(1)
unique_point_positions_constraint	u(1)
level_idc	u(8)
sps_seq_parameter_set_id	u(4)
frame_ctr_lsb_bits	u(5)
slice_tag_bits	u(5)
seq_origin_bits	ue(v)
if(seq_origin_bits) {	
for(*k* = 0; *k* < 3; *k*++)	
seq_origin_xyz[*k*]	s(v)
seq_origin_log2_scale	ue(v)
}	
seq_bbox_size_bits	ue(v)
if(seq_bbox_size_bits)	
for(*k* = 0; *k* < 3; *k*++)	
seq_bbox_size_minus1_xyz[*k*]	u(v)
seq_unit_numerator_minus1	ue(v)
seq_unit_denominator_minus1	ue(v)
seq_unit_is_metres	u(1)
seq_coded_scale_exponent	ue(v)
seq_coded_scale_mantissa_bits	ue(v)
seq_coded_scale_mantissa	u(v)
num_attributes	ue(v)
for(*attrIdx* = 0; *attrIdx* < num_attributes; *attrIdx*++) {	
attr_components_minus1[*attrIdx*]	ue(v)
attr_instance_id[*attrIdx*]	ue(v)
attr_bitdepth_minus1[*attrIdx*]	ue(v)
attr_label_known[*attrIdx*]	u(1)
if(attr_label_known[*attrIdx*])	
attr_label[*attrIdx*]	ue(v)
else	
attr_label_oid[*attrIdx*]	oid(v)
attr_property_cnt	ue(v)
byte_alignment()	
for(*j* = 0; *j* < attr_property_cnt; *j*++)	

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

attribute_property(*attrIdx*)	
}	
geom_axis_order	u(3)
bypass_stream_enabled	u(1)
entropy_continuation_enabled	u(1)
sps_extension_present	u(1)
if(sps_extension_present)	
while(more_data_in_data_unit())	
sps_extension_data	u(1)
byte_alignment()	
}	

7.3.2.2 Attribute property syntax

attribute_property(*attrIdx*) {	**Descriptor**
attr_prop_type	u(8)
attr_prop_len	u(8)
AttrPropDataLen = attr_prop_len	
if(attr_prop_type == 0) {	
attr_prop_itu_t_t35_country_code	u(8)
AttrPropDataLen−−	
if(attr_prop_itu_t_t35_country_code == 255) {	
attr_prop_itu_t_t35_country_code_extension_byte	u(8)
AttrPropDataLen−−	
}	
attribute_property_data(*attrIdx*, *AttrPropDataLen*)	
} else if(attr_prop_type == 1) {	
attr_prop_oid	oid(v)
AttrPropDataLen = Length(attr_prop_oid) / 8	
attribute_property_data(*attrIdx*, *AttrPropDataLen*)	
} else if(attr_prop_type == 2) {	
attr_cicp_colour_primaries[*attrIdx*]	ue(v)
attr_cicp_transfer_characteristics[*attrIdx*]	ue(v)
attr_cicp_matrix_coeffs[*attrIdx*]	ue(v)
attr_cicp_video_full_range[*attrIdx*]	u(1)
} else if(attr_prop_type == 3) {	
attr_offset_bits	ue(v)
attr_offset[*attrIdx*]	s(v)
attr_scale_bits	ue(v)
attr_scale_minus1[*attrIdx*]	u(v)
attr_frac_bits[*attrIdx*]	ue(v)
} else if(attr_prop_type == 4) {	
for(*c* = 0; *c* ≤ attr_components_minus1[*attrIdx*]; *c*++)	
attr_default_value[*attrIdx*][*c*]	u(v)
} else	

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

attribute_property_data(*attrIdx*, attr_prop_len)	
byte_alignment()	
}	

7.3.2.3 Attribute property data syntax

attribute_property_data(*attrIdx*, *numBytes*) {	**Descriptor**
for(*i* = 0; *i* < *numBytes*; *i*++)	
attr_prop_byte[*i*]	u(8)
}	

7.3.2.4 Tile inventory data unit syntax

tile_inventory() {	**Descriptor**
ti_seq_parameter_set_id	u(4)
ti_frame_ctr_lsb_bits	u(5)
ti_frame_ctr_lsb	u(v)
tile_cnt	u(16)
if(tile_cnt > 0) {	
tile_id_bits	u(5)
tile_origin_bits_minus1	u(8)
tile_size_bits_minus1	u(8)
for(*tileIdx* = 0; *tileIdx* < tile_cnt; *tileIdx*++) {	
tile_id[*tileIdx*]	u(v)
tileId = tile_id_bits ? tile_id[*tileIdx*] : *tileIdx*	
for(*k* = 0; *k* < 3; *k*++)	
tile_origin_xyz[*tileId*][*k*]	s(v)
for(*k* = 0; *k* < 3; *k*++)	
tile_size_minus1_xyz[*tileId*][*k*]	u(v)
}	
ti_origin_bits_minus1	ue(v)
for(*k* = 0; *k* < 3; *k*++)	
ti_origin_xyz[*k*]	s(v)
ti_origin_log2_scale	ue(v)
}	
byte_alignment()	
}	

7.3.2.5 Geometry parameter set data unit syntax

geometry_parameter_set() {	**Descriptor**
gps_geom_parameter_set_id	u(4)
gps_seq_parameter_set_id	u(4)
slice_geom_origin_scale_present	u(1)
if(¬slice_geom_origin_scale_present)	
gps_geom_origin_log2_scale	ue(v)

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

geom_dup_point_counts_enabled	u(1)
geom_tree_type	u(1)
if(geom_tree_type == 0) {	
occtree_point_cnt_list_present	u(1)
occtree_direct_coding_mode	u(2)
if(occtree_direct_coding_mode)	
occtree_direct_joint_coding_enabled	u(1)
occtree_coded_axis_list_present	u(1)
occtree_neigh_window_log2_minus1	u(3)
if(occtree_neigh_window_log2_minus1 > 0) {	
occtree_adjacent_child_enabled	u(1)
occtree_intra_pred_max_nodesize_log2	ue(v)
}	
occtree_bitwise_coding	u(1)
occtree_planar_enabled	u(1)
if(occtree_planar_enabled) {	
for(*i* = 0; *i* < 3; *i*++)	
occtree_planar_threshold[*i*]	ue(v)
if(occtree_direct_coding_mode == 1)	
occtree_direct_node_rate_minus1	u(5)
}	
}	
geom_angular_enabled	u(1)
if(geom_angular_enabled){	
slice_angular_origin_present	u(1)
if(¬slice_angular_origin_present) {	
gps_angular_origin_bits_minus1	ue(v)
for(*k* = 0; *k* < 3; *k*++)	
gps_angular_origin_xyz[*k*]	s(v)
}	
if(geom_tree_type == 1) {	
ptree_ang_azimuth_pi_bits_minus11	ue(v)
ptree_ang_azimuth_step_minus1	ue(v)
ptree_ang_radius_scale_log2	ue(v)
}	
num_beams_minus1	ue(v)
beam_elevation_init	se(v)
beam_voffset_init	se(v)
if(geom_tree_type == 0)	
beam_steps_per_rotation_init_minus1	ue(v)
for(*i* = 1; *i* ≤ num_beams_minus1; *i*++) {	
beam_elevation_diff[*i*]	se(v)
beam_voffset_diff[*i*]	se(v)
if(geom_tree_type == 0)	

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

beam_steps_per_rotation_diff[*i*]	se(v)
}	
if(octree_planar_enabled)	
occtree_planar_buffer_disabled	u(1)
}	
geom_scaling_enabled	u(1)
if(geom_scaling_enabled) {	
geom_qp	ue(v)
geom_qp_mul_log2	u(2)
if(geom_tree_type == 1)	
ptree_qp_period_log2	ue(v)
else if(occtree_direct_coding_mode)	
occtree_direct_node_qp_offset	se(v)
}	
gps_extension_present	u(1)
if(gps_extension_present)	
while(more_data_in_data_unit())	
gps_extension_data	u(1)
byte_alignment()	
}	

7.3.2.6 Attribute parameter set data unit syntax

attribute_parameter_set() {	**Descriptor**
aps_attr_parameter_set_id	u(4)
aps_seq_parameter_set_id	u(4)
attr_coding_type	ue(v)
attr_primary_qp_minus4	ue(v)
attr_secondary_qp_offset	se(v)
attr_qp_offsets_present	u(1)
if(attr_coding_type == 0) {	
raht_prediction_enabled	u(1)
if(raht_prediction_enabled) {	
raht_prediction_subtree_min	ue(v)
raht_prediction_samples_min	ue(v)
}	
} else if(attr_coding_type ≤ 2) {	
pred_set_size_minus1	ue(v)
pred_inter_lod_search_range	ue(v)
for(*k* = 0; *k* < 3; *k*++)	
pred_dist_bias_minus1_xyz[*k*]	ue(v)
if (attr_coding_type == 2)	
last_comp_pred_enabled	u(1)
lod_scalability_enabled	u(1)
if(lod_scalability_enabled)	

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

pred_max_range_minus1	ue(v)
else {	
lod_max_levels_minus1	ue(v)
if(¬lod_max_levels_minus1)	
attr_canonical_order_enabled	u(1)
else {	
lod_decimation_mode	ue(v)
if(lod_decimation_mode > 0)	
for(*lvl* = 0; *lvl* < lod_max_levels_minus1; *lvl*++)	
lod_sampling_period_minus2[*lvl*]	ue(v)
lod_initial_dist_log2	ue(v)
lod_dist_log2_offset_present	u(1)
}	
}	
if(attr_coding_type == 1) {	
pred_direct_max_idx_plus1	ue(v)
if(pred_direct_max_idx_plus1) {	
pred_direct_threshold	u(8)
pred_direct_avg_disabled	u(1)
}	
pred_intra_lod_search_range	ue(v)
if(pred_intra_lod_search_range)	
pred_intra_min_lod	ue(v)
inter_comp_pred_enabled	u(1)
pred_blending_enabled	u(1)
}	
} else if(attr_coding_type == 3)	
raw_attr_width_present	u(1)
if(¬lod_scalability_enabled)	
attr_coord_conv_enabled	u(1)
if(attr_coord_conv_enabled)	
for(*k* = 0; *k* < 3; *k*++) {	
attr_coord_conv_scale_bits_minus1[*k*]	u(5)
attr_coord_conv_scale[*k*]	u(v)
}	
aps_extension_present	u(1)
if(aps_extension_present)	
while(more_data_in_data_unit())	
aps_extension_data	u(1)
byte_alignment()	
}	

7.3.2.7 Frame-specific attribute properties data unit syntax

frame_specific_attribute_properties() {	**Descriptor**

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

fsap_seq_parameter_set_id	u(4)
fsap_frame_ctr_lsb_bits	u(5)
fsap_frame_ctr_lsb	u(v)
fsap_sps_attr_idx	ue(v)
fsap_num_props	ue(v)
byte_alignment()	
for(*i* = 0; *i* < fsap_num_props; *i*++)	
attribute_property(fsap_sps_attr_idx)	
}	

7.3.2.8 Frame boundary marker data unit syntax

frame_boundary_marker() {	**Descriptor**
fbdu_frame_ctr_lsb_bits	u(5)
fbdu_frame_ctr_lsb	u(v)
byte_alignment()	
}	

7.3.2.9 User data data unit syntax

userdata_data_unit() {	**Descriptor**
user_data_oid	oid(v)
while(more_data_in_data_unit())	
user_data_byte	u(8)
}	

7.3.2.10 Byte alignment syntax

byte_alignment() {	**Descriptor**
while(¬byte_aligned())	
alignment_bit_equal_to_zero /* equal to 0 */	u(1)
}	

7.3.3 Geometry data unit

7.3.3.1 Geometry data unit syntax

geometry_data_unit() {	**Descriptor**
geometry_data_unit_header()	
if(geom_tree_type == 0)	
occupancy_tree()	
else if(geom_tree_type == 1)	
predictive_tree()	
geometry_data_unit_footer()	
}	

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

7.3.3.2 Geometry data unit header syntax

geometry_data_unit_header() {	Descriptor	Semantics
gdu_geometry_parameter_set_id	u(4)	7.4.3.2
gdu_reserved_zero_3bits	u(3)	7.4.3.2
slice_id	ue(v)	7.4.3.2
slice_tag	u(v)	7.4.3.2
frame_ctr_lsb	u(v)	7.4.3.2
if(entropy_continuation_enabled) {		
slice_entropy_continuation	u(1)	7.4.3.2
if(slice_entropy_continuation)		
prev_slice_id	ue(v)	7.4.3.2
}		
if(slice_geom_origin_scale_present)		
slice_geom_origin_log2_scale	ue(v)	7.4.3.2
slice_geom_origin_bits_minus1	ue(v)	7.4.3.2
for(*k* = 0; *k* < 3; *k*++)		
slice_geom_origin_xyz[*k*]	u(v)	7.4.3.2
if(slice_angular_origin_present) {		
slice_angular_origin_bits_minus1	ue(v)	7.4.3.2
for(*k* = 0; *k* < 3; *k*++)		
slice_angular_origin_xyz[*k*]	s(v)	7.4.3.2
}		
if(geom_tree_type == 0) {		
occtree_depth_minus1	ue(v)	9.2.3
if(occtree_coded_axis_list_present)		
for(*dpth* = 0; *dpth* ≤ occtree_depth_minus1; *dpth*++)		
for(*k* = 0; *k* < 3; *k*++)		
occtree_coded_axis[*dpth*][*k*]	u(1)	9.2.3
occtree_stream_cnt_minus1	ue(v)	9.2.3
}		
if(geom_scaling_enabled) {		
slice_geom_qp_offset	se(v)	7.4.3.2
if(geom_tree_type == 1)		
slice_ptree_qp_period_log2_offset	se(v)	9.3.2.1
}		
if(geom_tree_type == 1) {		
for(*k* = 0; *k* < 3; *k*++)		
ptn_resid_abs_log2_bits[*k*]	u(3)	9.3.2.1
if(geom_angular_enabled)		
ptn_radius_min	ue(v)	9.3.2.1
}		
byte_alignment()		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

7.3.3.3 Geometry data unit footer syntax

geometry_data_unit_footer() {	Descriptor	Semantics
byte_alignment()		
if (occtree_point_cnt_list_present)		
for(*dpth* = 1; *dpth* < occtree_depth_minus1; *dpth*++)		
occtree_lvl_point_cnt_minus1[*dpth*]	u(24)	9.2.3
slice_num_points_minus1	u(24)	7.4.3.3
}		

7.3.3.4 Occupancy tree syntax

occupancy_tree() {	Descriptor	Semantics
OccQpSubtreeDepth = occtree_depth_minus1 + 1		9.2.14.4
for(*Dpth* = 0; *Dpth* ≤ occtree_depth_minus1; *Dpth*++) {		
occupancy_tree_level(*Dpth*)		
if(*Dpth* + 1 > *OcctreeEntropyStreamDepth*)		9.2.3
occtree_end_of_entropy_stream	ae(v)	9.2.3
}		
}		

7.3.3.5 Occupancy tree level syntax

occupancy_tree_level(*dpth*) {	Descriptor	Semantics
if(geom_scaling_enabled && *dpth* < *OccQpSubtreeDepth*)		
occ_subtree_qp_offset_present	ae(v)	9.2.14.3
if(occ_subtree_qp_offset_present)		
OccQpSubtreeDepth = *dpth*		
for(*NodeIdx* = 0; *NodeIdx* < *OccNodeCnt*[*dpth*]; *NodeIdx*++)		
occupancy_tree_node(*dpth*, *NodeIdx*)		
}		

7.3.3.6 Occupancy tree node syntax

occupancy_tree_node(*dpth*, *nodeIdx*) {	Descriptor	Semantics
if(occ_subtree_qp_offset_present) {		
occ_subtree_qp_offset_abs[*Ns*][*Nt*][*Nv*]	ae(v)	9.2.14.3
if(occ_subtree_qp_offset_abs[*Ns*][*Nt*][*Nv*])		
occ_subtree_qp_offset_sign[*Ns*][*Nt*][*Nv*]	ae(v)	9.2.14.3
}		
if(occtree_planar_enabled)		
for(*k* = 0; *k* < 3; *k*++)		
if(*PlanarEligible*[*k*]) {		9.2.11.5
occ_single_plane[*k*]	ae(v)	9.2.11.2
if(occ_single_plane[*k*])		
occ_plane_pos[*k*]	ae(v)	9.2.11.2

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

}		
if(occtree_direct_coding_mode && *DirectNodePresent*)		9.2.12.3.2
occ_direct_node	ae(v)	9.2.12.2
if(occ_direct_node)		
occupancy_tree_direct_node()		
else {		
if(*OccMaybeSingleChild*)		9.2.6.8
occ_single_child	ae(v)	9.2.6.2
if(occ_single_child)		
for(*k* = 0; *k* < 3; *k*++)		
if(*OccFreeAxis*[*k*])		9.2.6.6
occupancy_idx[*k*]	ae(v)	9.2.6.2
if(*OccMapPresent*)		9.2.6.9
if(occtree_bitwise_coding) {		
for(*i* = 0; *i* < 8; *i*++)		
if(*OccBitPresent*[*i*])		9.2.10.3
occupancy_bit[*i*]	ae(v)	9.2.6.2
} else		
occupancy_byte	de(v)	9.2.6.2
if(*TerminalNode* && geom_dup_point_counts_enabled)		9.2.6.5
for(*child* = 0; *child* < *OccChildCnt*; *child*++)		
occ_dup_point_cnt[*child*]	ae(v)	9.2.6.2
}		
}		

7.3.3.7 Direct node syntax

occupancy_tree_direct_node() {	Descriptor	Semantics
direct_point_cnt_eq2	ae(v)	9.2.12.2
if(geom_dup_point_counts_enabled && ¬direct_point_cnt_eq2)		
direct_dup_point_cnt	ae(v)	9.2.12.2
if(occtree_direct_joint_coding_enabled && direct_point_cnt_eq2)		
for(*k* = 0; *k* < 3; *k*++)		
if(¬geom_angular_enabled || *k* == (1 ^ *AzimuthAxis*)) {		9.2.13.3
direct_joint_prefix[*k*]	ae(v)	9.2.12.2
if(*DnJointDiffBitPresent*[*k*])		9.2.12.5.4
direct_joint_diff_bit[*k*]	ae(v)	9.2.12.2
}		
for(*dnPt* = 0; *dnPt* ≤ direct_point_cnt_eq2; *dnPt*++)		
if(geom_angular_enabled) {		
direct_rem[*dnPt*][1 ^ *AzimuthAxis*]	ae(v)	9.2.12.2
beam_idx_resid_abs[*dnPt*]	ae(v)	9.2.12.2
if(beam_idx_resid_abs[*dnPt*])		
beam_idx_resid_sign[*dnPt*]	ae(v)	9.2.12.2
direct_rem_st_ang[*dnPt*]	ae(v)	9.2.12.2

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

direct_rem_v_ang[*dnPt*]	ae(v)	9.2.12.2
} else		
for(*k* = 0; *k* < 3; *k*++)		
direct_rem[*dnPt*][*k*]	ae(v)	9.2.12.2
}		

7.3.3.8 Predictive tree syntax

predictive_tree() {	Descriptor	Semantics
PtnCnt = 0		
do {		
predictive_tree_node(0, *PtnCnt*)		
ptree_end_of_slice	ae(v)	9.3.2.1
} while(¬ptree_end_of_slice)		
}		

7.3.3.9 Predictive tree node syntax

predictive_tree_node(*dpth*, *nodeIdx*) {	Descriptor	Semantics
PtnCnt++		
if(geom_scaling_enabled && ¬(*nodeIdx* % *PtnQpInterval*)) {		
ptn_qp_offset_abs[*nodeIdx*]	ae(v)	9.3.2.2
if(ptn_qp_offset_abs[*nodeIdx*])		
ptn_qp_offset_sign[*nodeIdx*]	ae(v)	9.3.2.2
}		
if(geom_dup_point_counts_enabled)		
ptn_dup_point_cnt[*nodeIdx*]	ae(v)	9.3.2.2
ptn_child_cnt_xor1[*nodeIdx*]	ae(v)	9.3.2.2
ptn_pred_mode[*nodeIdx*]	ae(v)	9.3.2.2
if(geom_angular_enabled) {		
ptn_phi_mul_abs_prefix[*nodeIdx*]	ae(v)	9.3.2.2
if(ptn_phi_mul_abs_prefix[*nodeIdx*] == 2)		
ptn_phi_mul_abs_minus2[*nodeIdx*]	ae(v)	9.3.2.2
if(ptn_phi_mul_abs_minus2[*nodeIdx*] == 7)		
ptn_phi_mul_abs_minus9[*nodeIdx*]	ae(v)	9.3.2.2
if(ptn_phi_mul_abs_prefix[*nodeIdx*])		
ptn_phi_mul_sign[*nodeIdx*]	ae(v)	9.3.2.2
}		
for(*k* = 0; *k* < 3; *k*++) {		
if(*k* < 2 || ¬geom_angular_enabled || num_beams_minus1) {		
ptn_resid_abs_gt0[*nodeIdx*][*k*]	ae(v)	9.3.2.2
if(ptn_resid_abs_gt0[*nodeIdx*][*k*]) {		
ptn_resid_abs_log2[*nodeIdx*][*k*]	ae(v)	9.3.2.2
ptn_resid_abs_rem[*nodeIdx*][*k*]	ae(v)	9.3.2.2
if(*k* || ptn_pred_mode[*nodeIdx*])		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

ptn_resid_sign[*nodeIdx*][*k*]	ae(v)	9.3.2.2
}		
}		
}		
if(geom_angular_enabled)		
for(*k* = 0; *k* < 3; *k*++) {		
ptn_sec_resid_abs[*nodeIdx*][*k*]	ae(v)	9.3.2.2
if(ptn_sec_resid_abs[*nodeIdx*][*k*])		
ptn_sec_resid_sign[*nodeIdx*][*k*]	ae(v)	9.3.2.2
}		
for(*i* = 0; *i* < (ptn_child_cnt_xor1[*nodeIdx*] ^ 1); *i*++)		
predictive_tree_node(*dpth* + 1, *PtnCnt*)		
}		

7.3.4 Attribute data unit

7.3.4.1 Attribute data unit syntax

attribute_data_unit() {	**Descriptor**
attribute_data_unit_header()	
if(attr_coding_type ≠ 3)	
attribute_coeffs()	
else	
attribute_raw()	
byte_alignment()	
}	

7.3.4.2 Attribute data unit header syntax

attribute_data_unit_header() {	**Descriptor**	**Semantics**
adu_attr_parameter_set_id	u(4)	7.4.4.2
adu_reserved_zero_3bits	u(3)	7.4.4.2
adu_sps_attr_idx	ue(v)	7.4.4.2
adu_slice_id	ue(v)	7.4.4.2
if(lod_dist_log2_offset_present)		
lod_dist_log2_offset	se(v)	10.6.2
if(last_comp_pred_enabled && *AttrDim* == 3)		
for(*dpth* = 0; *dpth* ≤ lod_max_levels_minus1; *dpth*++)		
last_comp_pred_coeff_diff[*dpth*]	se(v)	10.6.10.1
if(inter_comp_pred_enabled)		
for(*dpth* = 0; *dpth* ≤ lod_max_levels_minus1; *dpth*++)		
for(*c* = 1; *c* < *AttrDim*; *c*++)		
inter_comp_pred_coeff_diff[*dpth*][*c*]	se(v)	10.6.10.1
if(attr_qp_offsets_present)		
for(*qc* = 0; *qc* < Min(2, *AttrDim*); *qc*++)		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

attr_qp_offset[*qc*]	se(v)	10.7.1
attr_qp_layers_present	u(1)	10.7.1
if(attr_qp_layers_present) {		
attr_qp_layer_cnt_minus1	ue(v)	10.7.1
for(*dpth* = 0; *dpth* ≤ attr_qp_layer_cnt_minus1; *dpth*++)		
for(*qc* = 0; *qc* < Min(2, *AttrDim*); *qc*++)		
attr_qp_layer_offset[*dpth*][*qc*]	se(v)	10.7.1
}		
attr_qp_region_cnt	ue(v)	10.7.1
if(attr_qp_region_cnt)		
attr_qp_region_bits_minus1	ue(v)	10.7.1
for(*i* = 0; *i* < attr_qp_region_cnt; *i*++) {		
if(¬attr_coord_conv_enabled) {		
for(*k* = 0; *k* < 3; *k*++)		
attr_qp_region_origin_xyz[*i*][*k*]	u(v)	10.7.1
for(*k* = 0; *k* < 3; *k*++)		
attr_qp_region_size_minus1_xyz[*i*][*k*]	u(v)	10.7.1
} else {		
for(*k* = 0; *k* < 3; *k*++)		
attr_qp_region_origin_rpi[*i*][*k*]	u(v)	10.7.1
for(*k* = 0; *k* < 3; *k*++)		
attr_qp_region_size_minus1_rpi[*i*][*k*]	u(v)	10.7.1
}		
for(*ps* = 0; *ps* < Min(2, *AttrDim*); *ps*++)		
attr_qp_region_offset[*i*][*ps*]	se(v)	10.7.1
}		
byte_alignment()		
}		

7.3.4.3 Attribute data unit coefficients syntax

attribute_coeffs() {	Descriptor	Semantics
for(*i* = 0; *i* < *PointCnt*; *i*++) {		
zero_run_length_prefix	ae(v)	10.3.1
if(zero_run_length_prefix == 3) {		
zero_run_length_minus3_div2	ae(v)	10.3.1
if(zero_run_length_minus3_div2 < 4)		
zero_run_length_minus3_mod2	ae(v)	10.3.1
else		
zero_run_length_minus11	ae(v)	10.3.1
}		
i += *ZeroRunLength*		
if(*i* < *PointCnt*)		
attribute_coeff_tuple(*i*)		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

}		

7.3.4.4 Attribute coefficient tuple syntax

attribute_coeff_tuple(*coeffIdx*) {	**Descriptor**	**Semantics**
for(*c* = 0, *inferLastComp* = 1; *c* < *AttrDim*; *c*++) {		
coeff_abs[*c*]	ae(v)	10.3.2
if(coeff_abs[*c*] || (*c* == *AttrDim* − 1 && *inferLastComp*))		
coeff_sign[*c*]	ae(v)	10.3.2
inferLastComp &= coeff_abs[*c*] == 0		
}		
}		

7.3.4.5 Raw attribute value syntax

attribute_raw() {	**Descriptor**	**Semantics**
for(*ptIdx* = 0; *ptIdx* < *PointCnt*; *ptIdx*++)		
for(*c* = 0; *c* < *AttrDim*; *c*++) {		
if(raw_attr_width_present)		
raw_attr_component_length	u(8)	10.3.3
raw_attr_value[*ptIdx*][*c*]	u(v)	10.3.3
}		
}		

7.3.5 Defaulted attribute data unit syntax

defaulted_attribute_data_unit() {	**Descriptor**	**Semantics**
defattr_seq_parameter_set_id	u(4)	7.4.5
defattr_reserved_zero_3bits	u(3)	7.4.5
defattr_sps_attr_idx	ue(v)	7.4.5
defattr_slice_id	ue(v)	7.4.5
for(*c* = 0; *c* < *AttrDim*; *c*++)		
defattr_value[*c*]	u(v)	7.4.5
byte_alignment()		
}		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

7.4 Semantics

7.4.1 General

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

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

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

© ISO/IEC 2023 – All rights reserved

7.4.2 Parameter sets, ancillary data and byte alignment

7.4.2.1 Sequence parameter set data unit semantics

7.4.2.1.1 General

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

simple_profile_compliant specifies whether (when 1) or not (when 0) the bitstream conforms to the Simple profile.

dense_profile_compliant specifies whether (when 1) or not (when 0) the bitstream conforms to the Dense profile.

predictive_profile_compliant specifies whether (when 1) or not (when 0) the bitstream conforms to the Predictive profile.

main_profile_compliant specifies whether (when 1) or not (when 0) the bitstream conforms to the Main profile.

reserved_profile_18bits shall be equal to 0 in bitstreams conforming to this version of this document. Other values for reserved_profile_18bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of reserved_profile_18bits.

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

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

NOTE 1 Even if the points in each slice have unique positions, points from different slices in the same frame can be coincident. In this case, unique_point_positions_constraint would be set to 0.

NOTE 2 Points with identical positions in the same frame are prohibited when unique_point_positions_constraint is 1 even if they have different values of the frame index/number attribute.

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

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

frame_ctr_lsb_bits specifies the length in bits of the syntax element frame_ctr_lsb.

slice_tag_bits specifies the length in bits of the syntax element slice_tag.

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

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

sps_extension_present specifies whether (when 1) or not (when 0) sps_extension_data syntax elements are present in the SPS syntax structure. sps_extension_present shall be 0 in bitstreams

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

conforming to this version of this document. The value of 1 for sps_extension_present is reserved for future use by ISO/IEC.

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

7.4.2.1.2 Coordinate systems

seq_origin_bits specifies the length in bits of each seq_origin_xyz syntax element exclusive of any sign bit.

seq_origin_xyz[*k*] and **seq_origin_log2_scale** together specify the XYZ origin of the sequence and coding coordinate systems in units of the sequence coordinate system from the application-specific coordinate system origin. When seq_origin_bits is 0, seq_origin_xyz[*k*] and seq_origin_log2_scale shall be inferred to be 0. The *k*-th XYZ component of the origin is specified by the expression *SeqOrigin*[*k*].

```
SeqOrigin[k] := seq_origin_xyz[k] << seq_origin_log2_scale
```

seq_bbox_size_bits specifies the length in bits of each seq_bbox_size_minus1_xyz syntax element.

seq_bbox_size_minus1_xyz[*k*] plus 1 specifies the *k*-th XYZ component of the coded volume dimensions in the sequence coordinate system. When seq_bbox_size_bits is 0, the coded volume dimensions are unspecified.

seq_unit_numerator_minus1, **seq_unit_denominator_minus1** and **seq_unit_is_metres** together specify the length represented by the unit vectors of the sequence coordinate system.

seq_unit_is_metres equal to 1 specifies that the sequence unit vectors have a length in metres equal to:

$$SeqUnit := \frac{\text{seq\_unit\_numerator\_minus1} + 1}{\text{seq\_unit\_denominator\_minus1} + 1} \text{ metre}$$

seq_unit_is_metres equal to 0 specifies that the sequence unit vectors have a length relative to the application-specific coordinate system unit vector length, *AppUnit*, equal to:

$$SeqUnit := \frac{\text{seq\_unit\_numerator\_minus1} + 1}{\text{seq\_unit\_denominator\_minus1} + 1} AppUnit$$

seq_coded_scale_exponent, **seq_coded_scale_mantissa_bits** and **seq_coded_scale_mantissa** together specify the scale factor that converts the coding coordinate system to the sequence coordinate system. The scale factor is represented by the syntax elements as a normalized binary floating-point value that is greater than or equal to 1. seq_coded_scale_mantisssa_bits specifies the length in bits of the syntax element seq_coded_scale_mantissa. The scale factor is specified by the expression *SeqCodedScale*.

$$SeqCodedScale := \left(1 + \frac{\text{seq\_coded\_scale\_mantissa}}{2^{\text{seq\_coded\_scale\_mantissa\_bits}}}\right) \times 2^{\text{seq\_coded\_scale\_exponent}}$$

geom_axis_order specifies the correspondence between the XYZ axes and the STV axes of the coded point cloud in accordance with Table 8.

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Table 8 — Definition of *StvToXyz*[*k*] according to the value of geom_axis_order

geom_axis_order	Axis (*k*) label			*StvToXyz*[*k*]		
	0 (S)	1 (T)	2 (V)	0 (S)	1 (T)	2 (V)
0 or 4	Z	Y	X	2	1	0
1 or 7	X	Y	Z	0	1	2
2	X	Z	Y	0	2	1
3	Y	Z	X	1	2	0
5	Z	X	Y	2	0	1
6	Y	X	Z	1	0	2

7.4.2.1.3 Attributes

Attributes are identified by their index into the SPS.

num_attributes specifies the number of attributes enumerated by the SPS attribute list.

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

```
AttrDim := attr_components_minus1[AttrIdx] + 1
AttrBitDepth := attr_bitdepth_minus1[AttrIdx] + 1
AttrMaxVal := Exp2(AttrBitDepth) − 1
```

attr_components_minus1[*attrIdx*] plus 1 specifies the number of components of the identified attribute.

NOTE 1 Attributes with more than three components can only be coded as raw attribute data (attr_coding_type = 3).

attr_instance_id[*attrIdx*] specifies the instance identifier for the identified attribute.

NOTE 2 The value of attr_instance_id can be used to differentiate between attributes with identical attribute labels. For example, a point cloud might have multiple colour attributes sampled from different view points. In this case, attr_instance_id can be used by an application to discriminate between the view points.

attr_bitdepth_minus1[*attrIdx*] plus 1 specifies the bit depth of every component of the identified attribute.

attr_label_known[*attrIdx*], **attr_label**[*attrIdx*] and **attr_label_oid**[*attrIdx*] together identify the type of data conveyed by the identified attribute. attr_label_known[*attrIdx*] specifies whether (when 1) the attribute is an attribute specified in this document by the value of attr_label[*attrIdx*], or (when 0) an externally specified attribute identified by the object identifier attr_label_oid[*attrIdx*].

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

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Table 9 — Identification of attribute type by attr_label

attr_label	Attribute type	Valid component counts
0	Colour	1 or 3
1	Reflectance	1
2	Opacity	1
3	Frame index	1
4	Frame number	1
5	Material identifier	1
6	Normal vector	3

attr_property_cnt specifies the number of attribute_property syntax structures present in the SPS for the attribute.

7.4.2.2 Attribute property semantics

7.4.2.2.1 Identification of an attribute property

An attribute_property(*attrIdx*) syntax structure specifies a property of the attribute identified by *attrIdx*.

attr_prop_type specifies the attribute property type according to Table 10. The interpretation of attribute properties identified as attribute specific are specified in accordance with the registration of attr_label_oid.

Table 10 — Identification of attribute parameter type by attr_prop_type

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

attr_prop_len shall be the length in bytes of the attribute_property syntax structure excluding the syntax elements attr_prop_type and attr_prop_len.

7.4.2.2.2 ITU-T T.35 user defined attribute properties

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

attr_prop_itu_t_t35_country_code is a byte having a value specified as a country code by Rec. ITU-T T.35, Annex A.

attr_prop_itu_t_t35_country_code_extension_byte is a byte having a value specified as a country code by Rec. ITU-T T.35, Annex B.

The ITU-T T.35 terminal provider code and terminal provider oriented code shall be contained in the initial bytes of attr_prop_byte[], in the format specified by the administration that issued the terminal provider code. Any remaining attr_prop_byte data shall be data having syntax and semantics as specified by the entity identified by the ITU-T T.35 country code and terminal provider code.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

7.4.2.2.3 G-PCC user defined attribute properties

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

attr_prop_oid specifies an ASN.1 object identifier value in the international object identifier tree in accordance with Rec. ITU-T X.660 | ISO/IEC 9834-1.

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

7.4.2.2.4 ISO/IEC 23091-2 video code points

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

attr_cicp_colour_primaries[*attrIdx*] specifies the chromaticity coordinates of the attribute's colour primaries. Values shall be interpreted according to the ColourPrimaries code point in ISO/IEC 23091-2.

attr_cicp_transfer_characteristics[*attrIdx*] specifies either the:

— reference opto-electronic transfer characteristic function of the attribute as a function of a source input, linear, optical intensity L_C with a nominal real-valued range of 0 to 1; or

— inverse of the reference electro-optical transfer characteristic function as a function of an output, linear, optical intensity L_O with a nominal real-valued range of 0 to 1.

Values shall be interpreted according to the TransferCharacteristics code point in ISO/IEC 23091-2.

attr_cicp_matrix_coeffs[*attrIdx*] describes the matrix coefficients used to derive the attribute's luma and chroma signals from the green, blue and red, or *Y*, *Z* and *X* primaries. Values shall be interpreted according to the MatrixCoefficients code point in ISO/IEC 23091-2.

attr_cicp_video_full_range[*attrIdx*] specifies the black level and range of the attribute's luma and chroma signals as derived from E'_Y, E'_{PB} and E'_{PR}, or E'_R, E'_G and E'_B real-valued component signals. Values shall be interpreted according to the VideoFullRangeFlag code point in ISO/IEC 23091-2.

7.4.2.2.5 Scale and offset properties

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

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

attr_offset_bits is the length in bits of the subsequent attr_offset[*attrIdx*] syntax element exclusive of any sign bit.

attr_scale_bits is the length in bits of the subsequent attr_scale_minus1[*attrIdx*] syntax element.

attr_offset[*attrIdx*], **attr_scale_minus1**[*attrIdx*] and **attr_frac_bits**[*attrIdx*] together specify how coded attribute values shall be interpreted. When present, the external interpretation A_{EXT} of each coded attribute value A shall be:

$$A_{EXT} = A\frac{\text{attr\_scale\_minus1} + 1}{2^{\text{attr\_frac\_bits}}} + \text{attr\_offset}$$

7.4.2.2.6 Default attribute value

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

attr_default_value[*attrIdx*][*c*] specifies the default value of the *c*-th component of the identified attribute. The length in bits of each syntax element shall be attr_bitdepth_minus1[*attrIdx*] + 1.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

7.4.2.3 Attribute property data semantics

attr_prop_byte[*i*] is a byte containing data having syntax and semantics not specified in this document.

7.4.2.4 Tile inventory data unit semantics

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

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

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

ti_seq_parameter_set_id identifies the active SPS by its sps_seq_parameter_set_id.

ti_frame_ctr_lsb_bits specifies the length in bits of the syntax element ti_frame_ctr_lsb. It is a requirement of bitstream conformance that ti_frame_ctr_lsb_bits shall be equal to frame_ctr_lsb_bits of the active SPS.

ti_frame_ctr_lsb should be the ti_frame_ctr_lsb_bits LSBs of *FrameCtr* for the next coded point cloud frame.

tile_cnt specifies the number of tiles enumerated by the tile inventory.

tile_id_bits specifies the length in bits of each tile_id syntax element. tile_id_bits equal to 0 specifies that tiles shall be identified by the index *tileIdx*.

tile_origin_bits_minus1 plus 1 specifies the length in bits of each tile_origin_xyz syntax element exclusive of any sign bit.

tile_size_bits_minus1 plus 1 specifies the length in bits of each tile_size_minus1_xyz syntax element.

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

tile_origin_xyz[*tileId*][*k*] and **tile_size_minus1_xyz**[*tileId*][*k*] indicate a bounding box in the sequence coordinate system encompassing slices identified by slice_tag equal to *tileId*.

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

tile_size_minus1_xyz[*tileId*][*k*] plus 1 specifies the *k*-th XYZ dimension of the tile bounding box.

ti_origin_bits_minus1 plus 1 specified the length in bits of each ti_origin_xyz syntax element exclusive of any sign bit.

ti_origin_xyz[*k*] and **ti_origin_log2_scale** together indicate the XYZ origin of the sequence coordinate system specified by seq_origin_xyz[*k*] and seq_origin_log2_scale. The values of ti_origin_xyz[*k*] and ti_origin_log2_scale should be equal to seq_origin_xyz[*k*] and seq_origin_log2_scale, respectively.

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

```
TileInventoryOrigin[k] := ti_origin_xyz[k] << ti_origin_log2_scale
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

7.4.2.5 Geometry parameter set data unit semantics

7.4.2.5.1 General parameters

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

gps_geom_parameter_set_id identifies the GPS for reference by other DUs.

gps_seq_parameter_set_id identifies the active SPS by its sps_seq_parameter_set_id.

slice_geom_origin_scale_present specifies whether (when 1) or not (when 0) slice_geom_origin_log2_scale is present in the GDU header. slice_geom_origin_scale_present equal to 0 specifies that the slice origin scale is specified by gps_geom_origin_log2_scale.

gps_geom_origin_log2_scale specifies the scale factor used to derive the slice origin from slice_geom_origin_xyz when slice_geom_origin_scale_present is 0.

geom_dup_point_counts_enabled specifies whether (when 1) or not (when 0) duplicate points can be signalled in a GDU by a per-point duplication count.

NOTE geom_dup_point_counts_enabled equal to 0 does not prohibit the coding of the same point position multiple times within a single slice by means other than the direct_dup_point_cnt, occ_dup_point_cnt or ptn_dup_point_cnt syntax elements.

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

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

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

7.4.2.5.2 Angular coding parameters

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

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

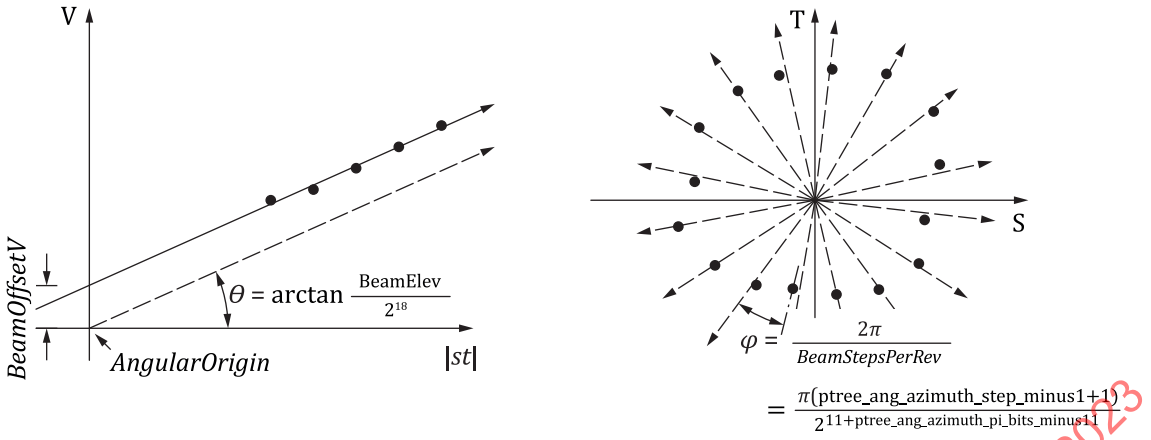


Figure 4 — Origin, elevation angle and azimuth step for a beam

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

gps_angular_origin_bits_minus1 plus 1 specifies the length in bits of each gps_angular_origin_xyz syntax element.

gps_angular_origin_xyz[*k*] specifies the *k*-th XYZ coordinate of the angular origin in the coding coordinate system.

num_beams_minus1 plus 1 specifies the number of beams enumerated by the GPS.

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

```
BeamElev[i] :=
  i == 0 ? beam_elevation_init :
  i == 1 ? beam_elevation_init + beam_elevation_diff[1]
         : 2 × BeamElev[i − 1] − BeamElev[i − 2] + beam_elevation_diff[i]
```

It is a requirement of bitstream conformance that values of *BeamElev*[*i*], *i* ∈ 1 .. num_beams_minus1, shall be greater than *BeamElev*[*i* − 1].

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

```
BeamOffsetV[i] :=
  i == 0 ? beam_voffset_init
         : BeamOffsetV[i − 1] + beam_voffset_diff[i]
```

beam_steps_per_rotation_init_minus1 and **beam_steps_per_rotation_diff**[*i*] specify the number of steps made per revolution by the rotating beams. The value for the *i*-th beam is specified by the expression *BeamStepsPerRev*[*i*].

```
BeamStepsPerRev[i] :=
  i == 0 ? beam_steps_per_rotation_init_minus1 + 1
         : BeamStepsPerRev[i − 1] + beam_steps_per_rotation_diff[i]
```

It is a requirement of bitstream conformance that values of *BeamStepsPerRev*[*i*], *i* ∈ 0 .. num_beams_minus1, shall not be 0.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

ptree_ang_azimuth_pi_bits_minus11 plus 11 specifies the number of bits that represent half a turn of a beam around the V axis. One half-turn is π radians.

ptree_ang_radius_scale_log2 specifies a factor used to scale a point's radial angular coordinate during conversion to Cartesian coordinates.

ptree_ang_azimuth_step_minus1 plus 1 specifies the expected change in azimuth angle of the rotating beams between coded points. Azimuth prediction residuals used in angular predictive tree coding can be coded as a multiple of ptree_ang_azimuth_step_minus1 + 1 and a remainder.

7.4.2.5.3 Occupancy tree parameters

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

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

NOTE Larger values for occtree_direct_coding_mode generally increase the rate of direct node eligibility.

occtree_direct_joint_coding_enabled specifies whether (when 1) or not (when 0) direct nodes that code two points shall jointly code their positions according to a specific ordering of the points.

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

occtree_neigh_window_log2_minus1 plus 1 specifies the number of occupancy tree node locations that form each availability window within a tree level. Nodes outside a window are unavailable to any process related to nodes within the window. occtree_neigh_window_log2_minus1 equal to 0 specifies that only sibling nodes shall be considered available to the current node.

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

occtree_intra_pred_max_nodesize_log2 minus 1 specifies the maximum size of an occupancy tree node that is eligible for intra-slice occupancy prediction. When occtree_intra_pred_max_nodesize_log2 is not present, it shall be inferred to be 0.

occtree_bitwise_coding specifies whether the node occupancy bitmap is coded using (when 1) occupancy_bit syntax elements or (when 0) the dictionary coded syntax element occupancy_byte.

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

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

occtree_direct_node_rate_minus1 specifies, when present, that of every 32 eligible nodes, only occtree_direct_node_rate_minus1 + 1 are permitted to be coded as direct nodes.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

7.4.2.5.4 Scaling parameters

geom_scaling_enabled specifies whether (when 1) or not (when 0) the coded geometry shall be scaled during the geometry decoding process.

geom_qp specifies the geometry QP prior to the addition of per slice and per-node offsets.

geom_qp_mul_log2 specifies the scale factor to be applied to the geometry QP. There are Exp2(3 – geom_qp_mul_log2) QP values for every doubling of the scaling step size.

ptree_qp_period_log2 specifies the period in nodes at which the predictive tree node QP offset is signalled. The period is one in every Exp2(ptree_qp_period_log2) nodes.

occtree_direct_node_qp_offset specifies an offset relative to the slice geometry QP for scaling direct node coded point positions.

7.4.2.6 Attribute parameter set data unit semantics

7.4.2.6.1 General parameters

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

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

aps_attr_parameter_set_id identifies the APS for reference by other DUs.

aps_seq_parameter_set_id identifies the active SPS by its sps_seq_parameter_set_id.

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

Table 11 — Interpretation of attr_coding_type

attr_coding_type	Description	Decoding process
0	Region-adaptive hierarchical transform (RAHT)	10.5
1	LoD with predicting transform	10.6
2	LoD with lifting transform	10.6
3	Raw attribute data	10.3

attr_primary_qp_minus4 plus 4 specifies the QP for the primary attribute component before the addition of per slice, per region and per-transform-level offsets.

attr_secondary_qp_offset specifies an offset to be applied to the primary attribute QP to derive the QP for any secondary attribute components.

attr_qp_offsets_present specifies whether (when 1) or not (when 0) per-slice attribute QP offsets, attr_qp_offset[c], are present in the ADU header.

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

attr_coord_conv_scale_bits_minus1[*k*] plus 1 specifies the length in bits of the syntax element attr_coord_conv_scale[*k*].

attr_coord_conv_scale[*k*] specifies the scale factor used to scale points' *k*-th angular coordinate for attribute coding. The scale factor shall be in units of 2^{-8}.

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

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

7.4.2.6.2 Region-adaptive hierarchical transform parameters

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

raht_prediction_subtree_min and **raht_prediction_samples_min** specify thresholds that control the use of RAHT coefficient prediction.

raht_prediction_samples_min specifies the minimum number of spatially adjacent samples from which RAHT coefficient prediction can be performed.

raht_prediction_subtree_min specifies the minimum number of spatially adjacent samples that need to be present to prevent the disabling of RAHT coefficient prediction for every descendant of a RAHT node.

7.4.2.6.3 Level of detail generation and transform parameters

pred_set_size_minus1 plus 1 specifies the maximum size of the per-point predictor set.

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

pred_dist_bias_minus1_xyz[*k*] plus 1 specifies the factor used to weight the *k*-th XYZ component of the distance vector between two point positions used to calculate inter-point distances in the predictor search for a single refinement point. The expression *PredBias*[*k*] specifies the factor for the *k*-th STV component.

```
PredBias[k] := pred_dist_bias_minus1_xyz[StvToXyz[k]] + 1
```

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

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

It is a requirement of bitstream conformance that lod_scalability_enabled shall be 0 when any of the following four conditions are true:

— geom_tree_type is 1;

— occtree_coded_axis_list_present is 1;

— geom_scaling_enabled is 1 and geom_qp_mul_log2 is not 3;

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— pred_blending_enabled is 1.

pred_max_range_minus1 plus 1 specifies, when present, the distance beyond which point predictor candidates shall be discarded during predictor set pruning for scalable attribute coding. The distance is specified in units of the per-detail-level block size.

lod_max_levels_minus1 plus 1 specifies the maximum number of detail levels that can be generated by the LoD generation process. When lod_max_levels_minus1 is not present, it shall be inferred to be *MaxSliceDimLog2* − 1.

attr_canonical_order_enabled specifies whether (when 1) or not (when 0) point attributes are coded in canonical point order. When attr_canonical_order_enabled is not present, it shall be inferred to be 0.

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

Table 12 — Interpretation of lod_decimation_mode

lod_decimation_mode	Description	Decoding process
0	No decimation	10.6.5.6
1	Periodic subsampling	10.6.5.5
2	Block-based subsampling	10.6.5.8

lod_sampling_period_minus2[*lvl*] plus 2 specifies the sampling period used by LoD generation to sample points in detail level *lvl* to generate the next coarser detail level *lvl* + 1.

lod_initial_dist_log2 specifies the block size at the finest detail level for use by LoD generation and predictor searches. When lod_initial_dist_log2 is not present, it shall be inferred to be 0.

lod_dist_log2_offset_present specifies whether (when 1) or not (when 0) the per-slice block-size offset specified by lod_dist_log2_offset shall be present in the ADU header. When lod_dist_log2_offset_present is not present, it shall be inferred to be 0.

pred_direct_max_idx_plus1 specifies the maximum number of single point predictors that can be used for direct prediction.

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

pred_direct_avg_disabled specifies whether (when 0) or not (when 1) the point predictor set average is a direct prediction mode.

pred_intra_lod_search_range specifies the range of indexes in a detail level's refinement list for which searched for nearest neighbours to include in a point's predictor set.

pred_intra_min_lod specifies the finest detail level in which intra-detail-level prediction is enabled. When pred_intra_min_lod is not present, it shall be inferred to be lod_max_levels_minus1 + 1. It is a requirement of bitstream conformance that pred_intra_min_lod shall be 0 when lod_max_levels_minus1 is 0.

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

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

7.4.2.6.4 Raw attribute parameters

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

7.4.2.7 Frame-specific attribute properties data unit semantics

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

— override any corresponding properties signalled in the active SPS for the specified frame only;

— apply to all ADUs in the frame with *AttrIdx* equal to fsap_sps_attr_idx.

All attribute properties with the same value of attr_prop_type shall be identical within a frame for any single attribute.

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

NOTE The requirements of fsap_frame_ctr_lsb prevent an FSAP DU from preceding the first GDU in the frame to which it applies.

fsap_seq_parameter_set_id identifies the active SPS by its sps_seq_parameter_set_id.

fsap_frame_ctr_lsb_bits specifies the length in bits of the syntax element fsap_frame_ctr_lsb. It is a requirement of bitstream conformance that fsap_frame_ctr_lsb_bits shall be equal to frame_ctr_lsb_bits of the active SPS.

fsap_frame_ctr_lsb identifies the frame to which the frame-specific attribute properties apply. Identification shall use fsap_frame_ctr_lsb_bits LSBs of the notional frame counter, *FrameCtr*. fsap_frame_ctr_lsb shall be equal to frame_ctr_lsb of the preceding GDU.

fsap_sps_attr_idx identifies the coded attribute to which the frame-specific attribute properties shall apply. Identification shall be by the index into the active SPS attribute list.

fsap_num_props specifies the number of attribute properties present in the syntax structure.

7.4.2.8 Frame boundary marker data unit semantics

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

fbdu_frame_ctr_lsb_bits specifies the length in bits of the syntax element fbdu_frame_ctr_lsb. It is a requirement of bitstream conformance that fbdu_frame_ctr_lsb_bits shall be equal to frame_ctr_lsb_bits of the active SPS.

fbdu_frame_ctr_lsb identifies the frame to which the frame boundary marker applies. Identification shall use fbdu_frame_ctr_lsb_bits LSBs of the notional frame counter *FrameCtr*.

7.4.2.9 User data data unit semantics

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

user_data_oid specifies an ASN.1 object identifier value in the international object identifier tree, as specified in Rec. ITU-T X.660 | ISO/IEC 9834-1.

user_data_byte is a byte containing data having syntax and semantics as specified by the registration of the object identifier.

7.4.2.10 Byte alignment semantics

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

alignment_bit_equal_to_zero shall be 0.

7.4.3 Geometry data unit

7.4.3.1 Geometry data unit semantics

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

7.4.3.2 Geometry data unit header semantics

gdu_geometry_parameter_set_id specifies the active GPS by its gps_geom_parameter_set_id.

gdu_reserved_zero_3bits shall be equal to 0 in bitstreams conforming to this version of this document. Other values of gdu_reserved_zero_3bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of gdu_reserved_zero_3bits.

slice_id identifies the slice for reference by other DUs.

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

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

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

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

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

slice_entropy_continuation should be 0 if slice_tag is not equal to the slice_tag of the GDU identified by prev_slice_id. For example, if slice_tag is used to select a subset of slices, then decoding can be prevented if there are dependencies upon slices that were not selected.

slice_geom_origin_bits_minus1 plus 1 specifies the length in bits of each slice_geom_origin_xyz syntax element.

slice_geom_origin_xyz[*k*] and **slice_geom_origin_log2_scale** specify the *k*-th XYZ coordinate of the slice origin in the coding coordinate system. The slice origin in STV coordinates is specified by the expression *SliceOrigin*[*k*]. When slice_geom_origin_log2_scale is not present, it shall be inferred to be gps_geom_origin_log2_scale.

```
SliceOrigin[k] := slice_geom_origin_xyz[StvToXyz[k]] << slice_geom_origin_log2_scale
```

slice_angular_origin_bits_minus1 plus 1 specifies the length in bits of each slice_angular_origin_xyz syntax element.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

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

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

```
AngularOrigin[k] := slice_angular_origin_present
  ? slice_angular_origin_xyz[StvToXyz[k]]
  : gps_angular_origin_xyz[StvToXyz[k]] - SliceOrigin[k]
```

slice_geom_qp_offset specifies the slice geometry QP as an offset to the GPS geom_qp. When slice_geom_qp_offset is not present, it shall be inferred to be 0.

7.4.3.3 Geometry data unit footer semantics

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

slice_num_points_minus1 plus 1 specifies the number of points coded in the DU. It is a requirement of bitstream conformance that slice_num_points_minus1 plus 1 shall be equal to the number of decodable points in the DU. Decoders shall not rely upon bitstream conformance to prevent overflow of implementation buffers.

7.4.4 Attribute data unit

7.4.4.1 Attribute data unit semantics

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

7.4.4.2 Attribute data unit header semantics

adu_attr_parameter_set_id specifies the active APS by its aps_attr_parameter_set_id.

adu_reserved_zero_3bits shall be equal to 0 in bitstreams conforming to this version of this document. Other values of adu_reserved_zero_3bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of adu_reserved_zero_3bits.

adu_sps_attr_idx identifies the coded attribute by its index into the active SPS attribute list.

At the start of every ADU, the variable *AttrIdx* is set to adu_sps_attr_idx:

```
AttrIdx = adu_sps_attr_idx
```

The attribute coded by the ADU shall have at most three components when attr_coding_type is not 3.

adu_slice_id specifies the value of the preceding GDU slice_id.

7.4.5 Defaulted attribute data unit semantics

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

defattr_seq_parameter_set_id specifies the active SPS by its sps_seq_parameter_set_id.

defattr_reserved_zero_3bits shall be 0 in bitstreams conforming to this version of this document. Other values of defattr_reserved_zero_3bits are reserved for future use by ISO/IEC. Decoders shall ignore the value of defattr_reserved_zero_3bits.

defattr_sps_attr_idx identifies the coded attribute by its index into the active SPS attribute list.

At the start of every defaulted attribute data unit, the variable *AttrIdx* is set to defattr_sps_attr_idx:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
AttrIdx = defattr_sps_attr_idx
```

defattr_slice_id specifies the value of the preceding GDU slice_id.

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

8 Decoding process

8.1 General decoding process

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

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

8.2 Frame decoding processes

8.2.1 General

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

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

```
RecCloudPointCnt = 0
```

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

8.2.2 Frame counter

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

```
window = Exp2(frame_ctr_lsb_bits) >> 1
curLsb = FrameCtr % Exp2(frame_ctr_lsb_bits)
curMsb = FrameCtr >> frame_ctr_lsb_bits
if ((frame_ctr_lsb < curLsb) && (curLsb - frame_ctr_lsb) ≥ window)
  curMsb++
else if ((frame_ctr_lsb > curLsb) && (frame_ctr_lsb - curLsb) > window)
  curMsb--
FrameCtr = (curMsb << frame_ctr_lsb_bits) + frame_ctr_lsb
```

8.3 Slice decoding processes

8.3.1 General

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

a) Point positions are decoded from one GDU in the slice as specified by 8.3.3.

b) Default attribute values are set for each attribute as specified by 8.3.4.

c) Point attributes are decoded from each ADU in the slice as specified by 8.3.5.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

d) The decoded point positions are offset and the output point count incremented as specified by 8.3.6.

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

8.3.2 State variables

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

— the variable *PointCnt*, a cumulative count of decoded points;

— the array *PointAng* of angular coordinates for decoded points; *PointAng*[*ptIdx*][*k*] is the *k*-th angular coordinate of the point position *PointPos*[*ptIdx*].

8.3.3 Geometry decoding process

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

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

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

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

When geom_angular_enabled is 1, the geometry decoding process populates the array *PointAng* with points' angular coordinates.

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

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

8.3.4 Default attribute values

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

— All components of the attribute values shall be set to Exp2(attr_bitdepth_minus1[*attrIdx*]).

— If the attribute property attr_default_value[*attrIdx*] is present, the attribute values shall be set to attr_default_value[*attrIdx*][*c*], for each component *c*.

— If the slice contains a defaulted attribute data unit with defattr_sps_attr_idx equal to *attrIdx*, the attribute values shall be set to defattr_value[*c*] of that DU, for each component *c*.

8.3.5 Attribute decoding process

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

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

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

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

8.3.6 At the end of a slice

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
RecCloudPointCnt += PointCnt
```

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

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

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

9 Slice geometry

9.1 General

This clause specifies the coding of slice geometry and the reconstruction of point positions, storing the reconstructed geometry in the arrays *PointPos* and *PointAng*.

9.2 Occupancy tree

9.2.1 General

This subclause specifies the parsing and reconstruction of point positions from a coded occupancy tree. It applies when geom_tree_type is 0.

An occupancy tree represents the slice geometry as a tree of occupancy tree nodes. Parsing or traversing a coded occupancy tree implicitly generates a representation of the slice geometry.

9.2.2 Coded occupancy tree

9.2.2.1 General tree structure

Individual point positions are represented in the occupancy tree either by the position of leaf nodes, or by direct nodes that encode node-relative positions.

An occupancy tree node shall identify the presence of at least one point contained within the volume of an axis-aligned cuboid. The volume is defined in the slice's coordinate system by an inclusive lower corner $p_{\min}$ and an exclusive upper corner $p_{\max}$. The volume edge lengths are non-negative integer powers of two. A node's size, *nodeSize*, is synonymous with the volume dimensions $p_{\max} - p_{\min}$.

The occupancy tree shall be formed of one or more tree levels. Every tree level consists of tree nodes with non-overlapping volumes. All tree nodes within a tree level shall have identical volume dimensions.

The occupancy tree shall contain a single root node. The root node shall be the only node in the top level of the tree. The volume identified by the root node shall have a lower corner at position (0, 0, 0), coincident with the slice origin. The upper corner shall be at an integer position $(2^s, 2^t, 2^v)$ equal to the root node size.

With each subsequent tree level, starting from the top tree level, the node volume dimensions are halved along one or more coded axes. The coded axes in each tree level are enumerated in the GDU header, as specified by occtree_coded_axis.

The location of a node, *nodeLoc*, within a tree level is related to the spatial position of the node volume's lower corner in the slice coordinate system by:

$$p_{min,k} = nodeLoc_k \times nodeSize_k$$

Two tree nodes are spatially adjacent if their volumes share a face.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Unless an early termination condition applies (9.2.6.5), tree nodes with a volume greater than the unit cube shall have one or more child nodes. Depending upon the number of coded axes, these nodes shall have at most eight, four, two, or one child nodes, as illustrated for cubic nodes in Figure 5.

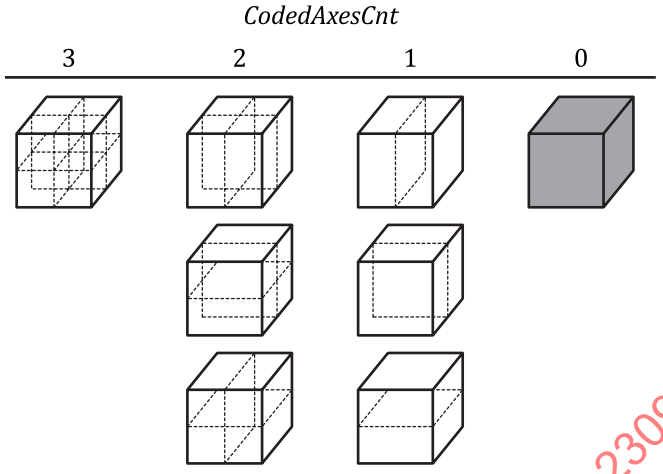


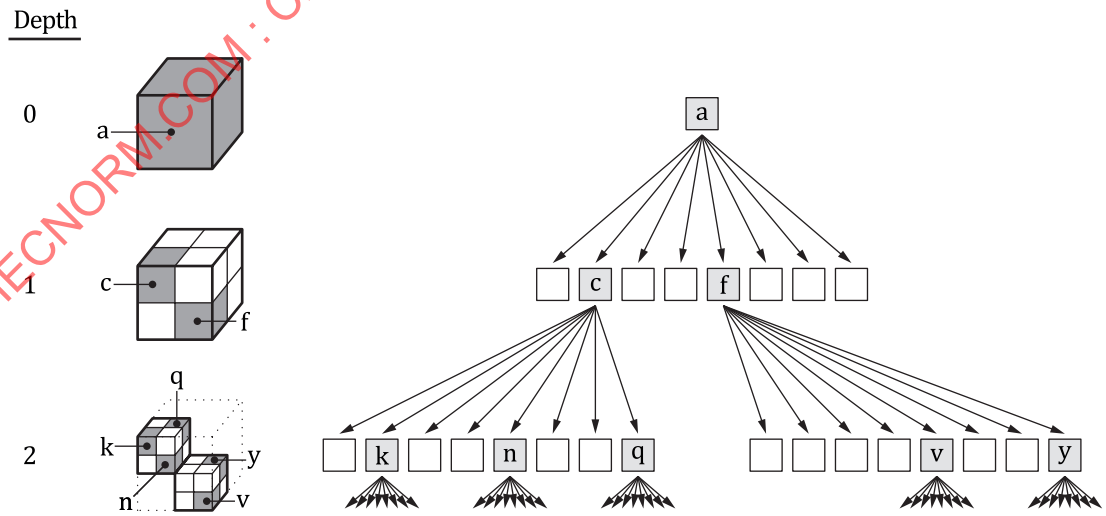
Figure 5 — Arrangement of child nodes depending upon coded axes

Leaf nodes, in the absence of geometry scaling (9.2.14), represent indivisible volumes with dimensions equal to the unit cube. When duplicate point coding is enabled, a leaf node may represent more than one point. In such cases, all points represented by the leaf node shall have identical positions.

9.2.2.2 Tree traversal 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.

The traversal order for an example tree is illustrated in Figure 6. Each tree level progressively refines the slice geometry. Starting from the root node, a, the node traversal order is from a to y. Occupied nodes are shaded.



NOTE This figure illustrates three tree levels.

Figure 6 — Occupancy tree traversal order

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.2.3 Node occupancy bitmap

The structure of the occupancy tree is coded as a sequence of node occupancy bitmaps.

Each node occupancy bitmap shall enumerate the child nodes for a single node. Each set bit position identifies the relative location of a child node in the next level of the occupancy tree as specified by the expression *OccLocC*[*bitIdx*][*k*] and Table 13. The count of set bits in the bitmap shall be the number of child nodes.

When a node has fewer than three coded axes, bits in the occupancy bitmap that do not enumerate a valid child node location shall be unset.

```
OccLocC[bitIdx][k] := Bit(bitIdx, 2 - k)
```

The tree level location of a child node is related to its parent's by:

$$nodeLoc_{\text{child},k} = nodeLoc_{\text{parent},k} \times \frac{nodeSize_{\text{parent},k}}{nodeSize_{\text{child},k}} + relLoc_k$$

Table 13 — Identification of valid relative child node location *relLoc* from set bits in an occupancy bitmap

Coded Axes			Bit position (*bitIdx*) in occupancy bitmap							
S	T	V	7 (MSB)	6	5	4	3	2	1	0 (LSB)
1	1	1	(1, 1, 1)	(1, 1, 0)	(1, 0, 1)	(1, 0, 0)	(0, 1, 1)	(0, 1, 0)	(0, 0, 1)	(0, 0, 0)
1	1	0		(1, 1, 0)		(1, 0, 0)		(0, 1, 0)		(0, 0, 0)
1	0	1			(1, 0, 1)	(1, 0, 0)			(0, 0, 1)	(0, 0, 0)
0	1	1					(0, 1, 1)	(0, 1, 0)	(0, 0, 1)	(0, 0, 0)
1	0	0				(1, 0, 0)				(0, 0, 0)
0	1	0						(0, 1, 0)		(0, 0, 0)
0	0	1							(0, 0, 1)	(0, 0, 0)
0	0	0								(0, 0, 0)

9.2.2.4 Terminal nodes

In the coded occupancy tree, leaf nodes are immediately encoded by their parent (terminal) node.

When geom_dup_point_counts_enabled is 1, terminal nodes shall encode the duplicate point counts for the leaf nodes they contain.

9.2.3 Occupancy tree syntax element semantics

occtree_depth_minus1 plus 1 specifies the maximum number of tree levels present in the coded occupancy tree. When occtree_coded_axis_list_present is 0, the root node size is a cubic volume with edge lengths equal to Exp2(occtree_depth_minus1 + 1).

NOTE Early termination of subtrees can result in fewer coded tree levels than the maximum.

occtree_coded_axis[*dpth*][*k*] specifies whether (when 1) or not (when 0) a subdivision along the *k*-th STV axis is coded by tree nodes at depth *dpth*. occtree_coded_axis shall be used to determine the node volume dimensions in each level of the occupancy tree. When occtree_coded_axis[*dpth*][*k*] is not present, it shall be inferred to be 1.

It is a requirement of bitstream conformance that:

— There shall be at least one coded axis in every tree level specified by occtree_coded_axis; i.e. MaxVec(occtree_coded_axis[*dpth*]) == 1.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— The log2 dimensions of the root node shall be less than or equal to *MaxSliceDimLog2*.

— The largest log2 dimension of the root node shall be greater than occtree_depth_minus1 – 4.

occtree_stream_cnt_minus1 plus 1 specifies the maximum number of entropy streams used to code the occupancy tree. When occtree_stream_cnt_minus1 is greater than zero, each of the bottom occtree_stream_cnt_minus1 tree levels shall be conveyed in a separate entropy stream; the parsing state shall be memorized and restored according to 11.6.

The expression *OcctreeEntropyStreamDepth* is the depth of the last tree level that is coded in the first entropy stream.

```
OcctreeEntropyStreamDepth := occtree_depth_minus1 - occtree_stream_cnt_minus1
```

occtree_end_of_entropy_stream is a non-coded syntax element used to specify the termination point for the arithmetic decoder at the end of an entropy stream. The syntax element has no value.

occtree_lvl_point_cnt_minus1[*dpth*] plus 1 indicates, when present, the number of points that can be partially decoded (See Annex D) from the root node to the end of the tree level at depth *dpth*. occtree_lvl_point_cnt_minus1[0] shall be inferred to be 0. occtree_lvl_point_cnt_minus1[occtree_depth_minus1] shall be inferred to be slice_num_points_minus1.

9.2.4 Node dimensions per tree level

The log2 node dimensions at depth *dpth* are specified by the expression *OccLvlNodeSizeLog2*[*dpth*][*k*]. They are derived from the list of coded axes:

— In the bottom tree level, at depth occtree_depth_minus1 + 1, the node dimensions shall be equal to the unit cube.

— The log2 node dimensions in any tree level shallower than the bottom tree level shall be the count of the respective coded axes, proceeding from the bottom tree level.

```
OccLvlNodeSizeLog2[dpth][k] := lvl > occtree_depth_minus1
  ? OccLvlNodeSizeLog2[dpth + 1][k] + occtree_coded_axis[dpth][k]
  : 0
```

9.2.5 State representation

9.2.5.1 State variables

The occupancy tree is specified in terms of the following state variable:

— The sparse array *OccNodePresent* that identifies nodes present in the occupancy tree. Each element *OccNodePresent*[*dpth*][*ns*][*nt*][*nv*] equal to either 1 or –1 indicates the presence of a node at location (*ns*, *nt*, *nv*) and depth *dpth*. Elements equal to –1 identify leaf nodes that are not coded at that depth (9.2.2.4). Unset elements of *OccNodePresent* are inferred to be 0.

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*.

— 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*.

9.2.5.2 The root node

At the start of the occupancy tree syntax structure, the arrays *OccNodePresent*, *OccNodeLoc* and *OccNodeCnt* are initialized to represent the root node at location (0, 0, 0); all other elements are cleared.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
OccNodePresent[0][0][0][0] = 1
OccNodeLoc[0][0][0] = OccNodeLoc[0][0][1] = OccNodeLoc[0][0][2] = 0
OccNodeCnt[0] = 1
```

9.2.6 Occupancy tree node coding

9.2.6.1 General

This subclause specifies the semantics of the *NodeIdx*-th coded node at tree depth *Dpth*.

9.2.6.2 Syntax element semantics

occ_single_child equal to 1 specifies that the coded node has a single child. occ_single_child equal to 0 specifies that the coded node may generate multiple child nodes. When occ_single_child is not present, it shall be inferred to be 0.

occupancy_idx[*k*] specifies the *k*-th component of the relative child node location for the only child of the coded node.

occupancy_bit and **occupancy_byte** specify the child nodes of the coded node as neighbourhood-permuted node occupancy bitmaps. The syntax elements shall be coded as specified by 9.2.10 and 9.2.9, respectively.

occ_dup_point_cnt[*i*] plus 1 specifies the number of points represented by the *i*-th coded child leaf node. All points represented by a child have identical positions. When occ_dup_point_cnt[*i*] is not present in a terminal node, it is inferred to be 0.

When unique_point_positions_constraint is 1, it is a requirement of bitstream conformance that occ_dup_point_cnt[*i*] shall be 0.

9.2.6.3 Node, parent, grandparent and child tree-level locations

The tree-level location (*Ns*, *Nt*, *Nv*) of the coded node is specified by the expression *Nloc*[*k*].

```
Nloc[k] := OccNodeLoc[Dpth][NodeIdx][k]

Ns := Nloc[0]
Nt := Nloc[1]
Nv := Nloc[2]
```

The parent node has a location (*NsP*, *NtP*, *NvP*) in the tree level at depth *Dpth* − 1. It is specified by the expression *NlocP*[*k*].

```
NlocP[k] := Dpth ? Nloc[k] >> occtree_coded_axis[Dpth - 1][k] : 0

NsP := NlocP[0]
NtP := NlocP[1]
NvP := NlocP[2]
```

The grandparent node has a location (*NsG*, *NtG*, *NvG*) in the tree level at depth *Dpth* − 2. It is specified by the expression *NlocG*[*k*].

```
NlocG[k] := Dpth > 1 ? NlocP[k] >> occtree_coded_axis[Dpth - 2][k] : 0

NsG := NlocG[0]
NtG := NlocG[1]
NvG := NlocG[2]
```

The corresponding location (*NsC*, *NtC*, *NvC*) in the tree level at depth *Dpth* + 1 for the coded node is specified by the expression *NlocC*[*k*].

```
NlocC[k] := Nloc[k] << AxisCoded[k]

NsC := NlocC[0]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
NtC := NlocC[1]
NvC := NlocC[2]
```

9.2.6.4 Node size

The expressions *NodeSizeLog2*[*k*] and *ChildNodeSizeLog2*[*k*] specify the log2 dimensions of the coded node and its children, respectively.

```
NodeSizeLog2[k] := OccLvlNodeSizeLog2[Dpth][k]

ChildNodeSizeLog2[k] := OccLvlNodeSizeLog2[Dpth + 1][k]
```

NOTE When geom_scaling_enabled is 0, *QuantizedNodeSizeLog2*[*k*] is equal to *NodeSizeLog2*[*k*].

9.2.6.5 Whether the node is a terminal node

A node is a terminal node, as specified by the expression *TerminalNode*, if its children are leaf nodes, or it is a direct node.

```
TerminalNode := MaxVec(ChildNodeSizeLog2) == 0
  || geom_scaling_enabled && MaxVec(QuantizedChildNodeSizeLog2) == 0
  || occtree_direct_coding_mode && occ_direct_node
```

9.2.6.6 Coded axes

A node shall only code an axis for child locations when specified by the expression *AxisCoded*[*k*]. An axis is coded when:

— it is specified to be coded in the tree level by the coded axis list, and

— if geometry subtree scaling is enabled, the corresponding dimension of the quantized node size is greater than 1.

```
AxisCoded[k] := occtree_coded_axis[Dpth][k]
  && (¬geom_scaling_enabled || QuantizedNodeSizeLog2[k] > 0)
```

The expression *CodedAxisCnt* is the number of axes coded by the node.

```
CodedAxisCnt := AxisCoded[0] + AxisCoded[1] + AxisCoded[2]
```

The location of child nodes along each coded axis may be constrained by planar occupancy coding (9.2.11.4). A free axis is a coded axis that is not constrained so, as specified by the expression *OccFreeAxis*[*k*]. The number of free axes is specified by the expression *OccFreeAxisCnt*.

NOTE When planar occupancy coding is disabled, *OccFreeAxisCnt* is equal to *CodedAxisCnt*.

```
OccFreeAxis[k] := AxisCoded[k] && (¬occtree_planar_enabled || PlanarFreeAxis[k])

OccFreeAxisCnt := OccFreeAxis[0] + OccFreeAxis[1] + OccFreeAxis[2]
```

9.2.6.7 Limits to the number of child nodes

The number of child nodes a node can contain is constrained by the tree and node syntax.

The maximum number of child nodes is specified by the expression *OccMaxChildren*. Unless occ_single_child is 1, the limit shall be the number of child node locations that can be identified by the free axes. Otherwise, the limit shall be 1 when occ_single_child is 1.

```
OccMaxChildren := occ_single_child ? 1 : Exp2(OccFreeAxisCnt)
```

The minimum number of child nodes is specified by the expression *OccMinChildren*. A node shall contain at least one child node unless:

— planar occupancy coding specifies that there shall be at least two child nodes (9.2.11.3), or

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— occ_single_child is both present and equal to 0, in which case there shall be at least two child nodes.

```
OccMinChildren :=
  OccMaybeSingleChild && ¬occ_single_child ? 2 :
  occtree_planar_enabled                   ? PlanarMinChildren : 1
```

9.2.6.8 Presence of occ_single_child

The presence of occ_single_child is specified by the expression *OccMaybeSingleChild*. It shall be present in the occupancy node syntax when all the following conditions are true:

— No nodes are present in the occupied neighbourhood pattern (9.2.7.4).

— There is at least one free axis to code child node locations.

— Planar occupancy coding does not specify that there shall be at least two child nodes (9.2.11.3).

```
OccMaybeSingleChild :=
  ¬OccNeighPat && OccFreeAxesCnt > 0 && PlanarMinChildren == 1
```

9.2.6.9 Presence of occupancy_bit and occupancy_byte

The node occupancy bitmap shall be coded using either occupancy_bit or occupancy_byte when specified by the expression *OccMapPresent*. One of the two syntax elements shall be present when both of the following conditions are true:

— The maximum number of child nodes is greater than 1.

— The locations of child nodes are not completely prescribed by constraints on occupancy. i.e. the maximum number of child nodes is greater than the minimum number of child nodes.

```
OccMapPresent := OccMaxChildren > 1 && OccMinChildren != OccMaxChildren
```

9.2.6.10 Node occupancy bitmap

This subclause specifies the node occupancy bitmap by the expression *OccupancyMap*.

When occupancy_bit syntax elements are present (*OccMapPresent* is 1), the node occupancy bitmap is specified by bitwise occupancy coding (9.2.10.2).

```
if (OccMapPresent && occtree_bitwise_coding)
  OccupancyMap = OccBitMap
```

When occupancy_byte is present (*OccMapPresent* is 1), the node occupancy bitmap shall be rearranged from the neighbourhood-permuted bitmap (9.2.8) coded by occupancy_byte.

```
if (OccMapPresent && ¬occtree_bitwise_coding)
  OccupancyMap = OccFromNpOcc(occupancy_byte)
```

When constraints on occupancy require there to be a single child node, each component *k* of the child location shall be specified by whichever of occ_plane_pos[*k*] or occupancy_idx[*k*] are present, or shall be 0 if neither is present.

```
if (OccMaxChildren == 1 && OccMinChildren == 1) {
  occupancyIdx[k] :=
    OccFreeAxis[k] && occupancy_idx[k] || ¬PlanarFreeAxis[k] && occ_plane_pos[k]

  OccupancyMap = 1 << Morton[occupancyIdx]
}
```

NOTE In this case, an axis cannot be both a free axis and eligible for planar occupancy coding.

When constraints on occupancy require there to be two child nodes, there shall be one child node at both permitted locations along the free axis.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
if (OccMaxChildren == 2 && OccMinChildren == 2) {
  baseIdx[k] := ¬PlanarFreeAxis[k] && occ_plane_pos[k]

  if (OccFreeAxis[0]) OccupancyMap = 0x11 << Morton[baseIdx]
  if (OccFreeAxis[1]) OccupancyMap = 0x05 << Morton[baseIdx]
  if (OccFreeAxis[2]) OccupancyMap = 0x03 << Morton[baseIdx]
}
```

9.2.6.11 Child node count

The number of child nodes is equal to the number of set bits in the node occupancy bitmap, as specified by the expression *OccChildCnt*.

```
OccChildCnt := PopCnt(OccupancyMap)
```

9.2.6.12 Insertion of non-terminal child nodes

Unless the coded node is a terminal node, its child nodes shall be inserted into the state representation of the occupancy tree and included in the traversal list of the next tree level. The node occupancy bitmap shall be scanned to enumerate the child nodes.

```
if (¬TerminalNode)
  for (occBitIdx = 0; occBitIdx < 8; occBitIdx++) {
    if (¬Bit(OccupancyMap, occBitIdx))
      continue

    cs = NsC + OccLocC[occBitIdx][0]
    ct = NtC + OccLocC[occBitIdx][1]
    cv = NvC + OccLocC[occBitIdx][2]
    OccNodePresent[Dpth + 1][cs][ct][cv] = 1

    childNodeIdx = OccNodeCnt[Dpth + 1]
    OccNodeCnt[Dpth + 1]++

    OccNode[Dpth + 1][childNodeIdx][0] = cs
    OccNode[Dpth + 1][childNodeIdx][1] = ct
    OccNode[Dpth + 1][childNodeIdx][2] = cv
  }
```

9.2.6.13 Points represented by child leaf nodes

When the node is a non-direct terminal node, points represented by child leaf nodes shall be scaled (9.2.14.6) and appended to the output point list. The node occupancy bitmap shall be scanned to enumerate the child nodes.

NOTE When geometry scaling is disabled, this condition is equivalent to the child node size being equal to the unit cube.

When adjacent child occupancy contextualization is enabled, the child leaf nodes shall be inserted into the state representation of the occupancy tree for use by other nodes in the same tree level; but they shall be excluded from traversal in the next tree level. The child leaf nodes shall not be included in the occupied neighbourhood pattern for any node in the next tree level.

```
if (TerminalNode && ¬occ_direct_node)
  for (child = 0, occBitIdx = 0; occBitIdx < 8; occBitIdx++) {
    if (¬Bit(OccupancyMap, occBitIdx))
      continue

    cs = NsC + OccLocC[occBitIdx][0]
    ct = NtC + OccLocC[occBitIdx][1]
    cv = NvC + OccLocC[occBitIdx][2]
    if (occtree_adjacent_child_enabled)
      OccNodePresent[Dpth + 1][cs][ct][cv] = -1

    for (i = 0; i < occ_dup_point_cnt[child] + 1; i++, PointCnt++) {
      PointPos[PointCnt][0] = OccPosScaleK(0, cs)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
      PointPos[PointCnt][1] = OccPosScaleK(1, ct)
      PointPos[PointCnt][2] = OccPosScaleK(2, cv)
    }

    child++
  }
```

9.2.7 Occupied neighbourhood patterns

9.2.7.1 General

Coding of the node occupancy bitmap depends upon the existence and arrangement of up to six spatially adjacent tree nodes within an availability window. The occupied neighbourhood pattern for an occupancy tree node shall identify the spatial arrangement of these adjacent nodes from the 64 possible combinations. Examples of occupied neighbourhood patterns are illustrated in Figure 7.

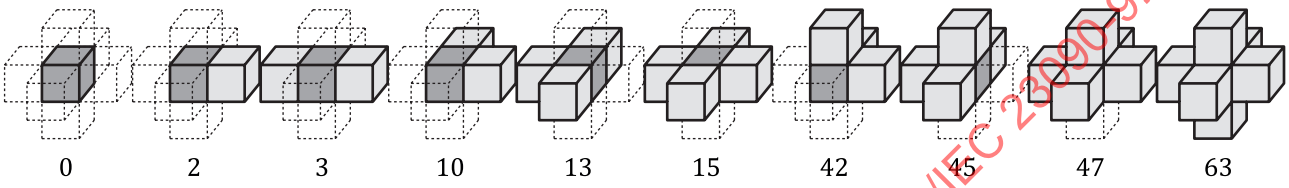


Figure 7 — Characteristic occupied neighbourhood patterns

9.2.7.2 Neighbour availability

Nodes are grouped into availability windows by their spatial location within their tree level. Nodes can form part of the occupied neighbourhood pattern of adjacent nodes within the same window. Nodes shall not form part of the occupied neighbourhood pattern of any node in a different window.

The size of the availability window is specified by the expression *OccAvailWinLog2*[*k*]:

— Unless occtree_neigh_window_log2_minus1 is 0, each availability window shall span *n*×*n*×*n* node locations, *n* = Exp2(occtree_neigh_window_log2_minus1 + 1), within the tree level. The availability windows form a contiguous grid starting from the location (0, 0, 0).

— Otherwise, the availability window for any node shall be restricted to its sibling nodes.

```
OccAvailWinLog2[k] := occtree_neigh_window_log2_minus1
  + (occtree_neigh_window_log2_minus1 > 0 || Dpth > 0 && occtree_coded_axis[Dpth - 1][k])
```

NOTE Only the Main profile permits an element of occtree_coded_axis to be 0 when occtree_neigh_window_log2_minus1 is 0.

The expression *OccNeighAvail*[*ns*][*nt*][*nv*] specifies whether the node at location (*ns*, *nt*, *nv*) is within the same availability window as the coded node (*Ns*, *Nt*, *Nv*).

```
OccNeighAvail[ns][nt][nv] :=
     (ns ^ Ns) >> OccAvailWinLog2[0] == 0
  && (nt ^ Nt) >> OccAvailWinLog2[1] == 0
  && (nv ^ Nv) >> OccAvailWinLog2[2] == 0
```

9.2.7.3 Presence of another coded node within the availability window

The expression *OccNeigh*[*ns*][*nt*][*nv*] identifies whether there exists a node with tree location (*ns*, *nt*, *nv*) and depth *Dpth* that is not a leaf node and is within the availability window of the coded node (*Ns*, *Nt*, *Nv*).

NOTE *OccNodePresent*[*Dpth*][*ns*][*nt*][*nv*] equal to –1 identifies a leaf node.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
OccNeigh[ns][nt][nv] := OccNeighAvail[ns][nt][nv] && OccNodePresent[Dpth][ns][nt][nv] == 1
```

9.2.7.4 Occupied neighbourhood pattern

The occupied neighbourhood pattern for the coded node located at (*Ns*, *Nt*, *Nv*) in the tree level at depth *Dpth* is specified by the expression *OccNeighPat*. It is a linear combination of spatially adjacent nodes coded in the same tree level that are available and adjoin the coded node by a face. Leaf nodes shall not be included in the occupied neighbourhood pattern.

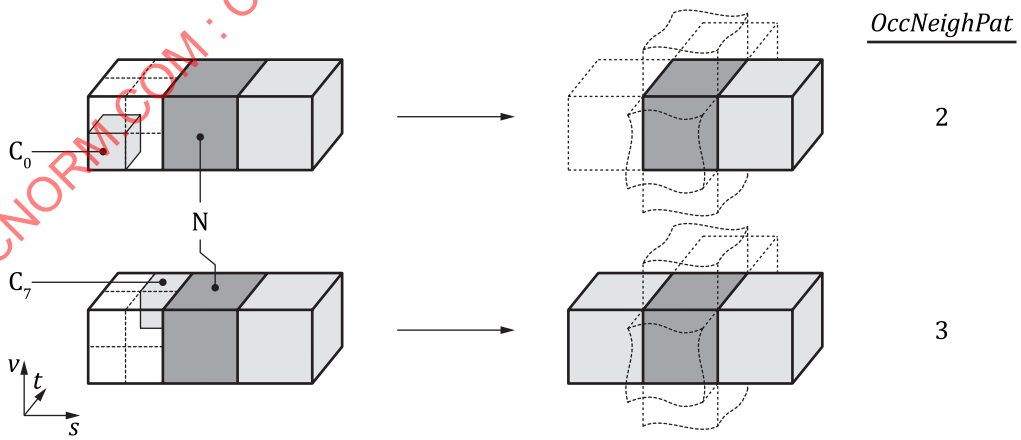
An occupancy tree node with no spatially adjacent nodes has an occupied neighbourhood pattern equal to 0.

```
OccNeighPat := (uN << 5) | (dN << 4) | (bN << 3) | (fN << 2) | (rN << 1) | lN
  where
    rN := OccNeigh[Ns + 1][Nt][Nv]
    lN := OccNeigh[Ns - 1][Nt][Nv] && (¬occtree_adjacent_child_enabled || lNadj)
    bN := OccNeigh[Ns][Nt + 1][Nv]
    fN := OccNeigh[Ns][Nt - 1][Nv] && (¬occtree_adjacent_child_enabled || fNadj)
    uN := OccNeigh[Ns][Nt][Nv + 1]
    dN := OccNeigh[Ns][Nt][Nv - 1] && (¬occtree_adjacent_child_enabled || dNadj)
```

When adjacent child contextualization is enabled (occtree_adjacent_child_enabled is 1), a tree node that adjoins the left (*Ns* − 1), front (*Nt* − 1) or bottom (*Nv* − 1) face shall not be included in the occupied neighbourhood pattern unless it contains at least one child node that also adjoins the same face. Their inclusion is specified by the expressions *lNadj*, *fNadj* and *dNadj*, equivalent to the following:

```
lNadj = fNadj = dNadj = 0
for (s = 0; s ≤ occtree_coded_axis[Dpth][0]; s++)
  for (t = 0; t ≤ occtree_coded_axis[Dpth][1]; t++)
    for (v = 0; v ≤ occtree_coded_axis[Dpth][2]; v++) {
      lNadj |= OccNodePresent[Dpth + 1][NsC - 1][NtC + t][NvC + v] ≠ 0
      fNadj |= OccNodePresent[Dpth + 1][NsC + s][NtC - 1][NvC + v] ≠ 0
      dNadj |= OccNodePresent[Dpth + 1][NsC + s][NtC + t][NvC - 1] ≠ 0
    }
```

The exclusion of a node from the occupied neighbourhood pattern due to adjacent child contextualization is illustrated in Figure 8. The child node C_0 of the left neighbour does not adjoin the left face of the coded node N resulting in its parent node being excluded from the occupied neighbourhood pattern; *OccNeighPat* is 2. The child node C_7 does adjoin N and its parent node is not excluded; *OccNeighPat* is 3.



Key

N coded node

C_m child node at location *m* in left neighbour

NOTE Exclusion (top); inclusion (bottom).

Figure 8 — Effect of adjacent child contextualization on an occupied neighbourhood pattern

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.7.5 Reduced occupied neighbourhood pattern

The occupied neighbourhood pattern shall be reduced to one of a smaller set of patterns as specified by the expression *OccNeighPatR*.

When occtree_neigh_window_log2_minus1 is greater than 0, the smaller set of patterns is specified by Table 14 as a mapping of spatial rotations and reflections in the arrangement of the six spatially adjacent neighbours to produce nine unique arrangements.

Otherwise, when occtree_neigh_window_log2_minus1 is 0, the smaller set of patterns is specified by Table 15 as a mapping of adjacent siblings that produces six arrangements.

```
OccNeighPatR := occtree_neigh_window_log2_minus1 > 0
  ? NeighPat64to9[OccNeighPat]
  : NeighPat64to6[OccNeighPat]
```

Table 14 — Reduction of occupied neighbourhood pattern *j* + *i* to nine patterns, *NeighPat64to9*[*j* + *i*]

j	*i*															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	1	1	1	1	2	2	3	1	2	2	3	1	3	3	4
16	1	2	2	3	2	5	5	6	2	5	5	6	3	6	6	7
32	1	2	2	3	2	5	5	6	2	5	5	6	3	6	6	7
48	1	3	3	4	3	6	6	7	3	6	6	7	4	7	7	8

Table 15 — Reduction of occupied neighbourhood pattern *j* + *i* to six patterns, *NeighPat64to6*[*j* + *i*]

j	*i*															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	5	5	na	5	1	1	na	5	1	1	na	na	na	na	na
16	2	3	3	na	3	7	7	na	3	7	7	na	na	na	na	na
32	2	3	3	na	3	7	7	na	3	7	7	na	na	na	na	na
48	na	na	na	na	na	na	na	na	na	na	na	na	na	na	na	na

NOTE The specification of values 5 and 7 aligns with further reductions performed in bitwise occupancy coding.

9.2.8 Neighbourhood-permuted node occupancy bitmap

The neighbourhood-permuted node occupancy bitmap is a rearrangement of the bits forming the node occupancy bitmap. It is used in the coding of occupancy_byte and occupancy_bit. The permutation shall be selected according to the occupied neighbourhood pattern.

The permutations for every occupied neighbourhood pattern are specified by Table 16. Each entry is a base eight value with digits numbered from right to left. The *i*-th digit is the bit position in the node occupancy bitmap of the *i*-th bit in the neighbourhood-permuted bitmap as specified by the expression *OccBitIdxFromNpBit*[*i*].

```
OccBitIdxFromNpBit[i] := (OccArrangement[OccNeighPat] >> i × 3) & 7
```

The expression *OccFromNpOcc*[*npocc*] is the node occupancy bitmap derived from the neighbourhood-permuted node occupancy bitmap *npocc*.

```
OccFromNpOcc[npocc] :=
  OccFromNpOcc = 0
  for (i = 0; i < 8; i++)
    OccFromNpOcc |= Bit(npocc, i) << OccBitIdxFromNpBit[i]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Two example derivations of *OccupancyMap* from a single occupancy_byte by *OccFromNpOcc* are illustrated in Figure 9. Each derivation uses a different occupied neighbourhood pattern, *OccNeighPat*. Bit b_4 of occupancy_byte is permuted to bit b_1 of *OccupancyMap* when *OccNeighPat* is 17; in this case, *OccBitIdxFromNpBit*[4] would be 1.

NOTE occupancy_bit codes the bits of the neighbourhood-permuted node occupancy bitmap in a different order. i.e. occupancy_bit[i] does not correspond to bit b_i of occupancy_byte.

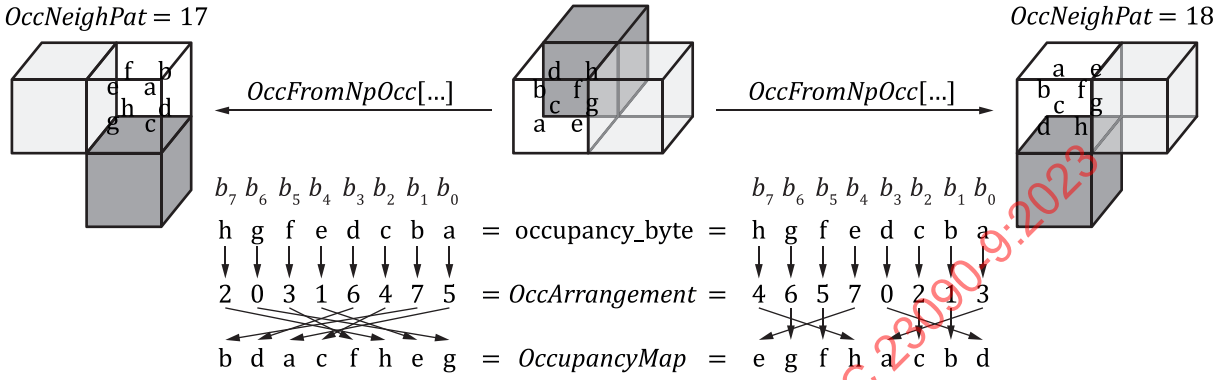


Figure 9 — Example relationships between the node occupancy bitmap *OccupancyMap* and neighbourhood-permuted node occupancy bitmap as coded by occupancy_byte

Table 16 — Arrangements for neighbourhood-permuted node occupancy bitmaps by occupied neighbourhood pattern as *OccArrangement*[*i* + *j*]

i	*j*					
	0	**1**	**2**	**3**	**4**	**5**
0	76543210_8	10325476_8	76543210_8	76543210_8	54107632_8	10325476_8
6	54107632_8	32761054_8	32761054_8	32761054_8	76543210_8	54107632_8
12	32761054_8	76543210_8	10325476_8	76543210_8	26043715_8	20316475_8
18	46570213_8	57134602_8	04152637_8	01234567_8	45016723_8	23670145_8
24	62734051_8	23670145_8	67452301_8	45016723_8	73516240_8	67452301_8
30	01234567_8	67452301_8	37152604_8	31207564_8	57461302_8	46025713_8
36	15043726_8	10325476_8	54107632_8	32761054_8	73625140_8	32761054_8
42	76543210_8	54107632_8	62407351_8	76543210_8	10325476_8	76543210_8
48	37152604_8	64752031_8	02134657_8	57461302_8	26370415_8	57461302_8
54	73625140_8	13570246_8	40516273_8	15043726_8	31207564_8	75316420_8
60	73625140_8	37152604_8	51734062_8	76543210_8		

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

9.2.9 Dictionary coding of occupancy_byte

9.2.9.1 General

The occupancy_byte syntax element shall be coded as symbols by one of nine instances of this dictionary codec. Coding shall proceed according to the syntax and semantics of the occupancy_byte_symbol syntax structure.

Each dictionary instance comprises a list of thirty-two most probable symbols (occupancy_byte values), a list of sixteen recently coded symbols, a histogram of symbol counts and state variables used to control updates to the dictionary state.

© ISO/IEC 2023 – All rights reserved

9.2.9.2 Syntax of a dictionary coded symbol

occupancy_byte_symbol() {	**Descriptor**
occ_histogram_hit	ae(v)
if(occ_histogram_hit)	
occ_histogram_index	ae(v)
else {	
occ_recent_hit	ae(v)
if(occ_recent_hit)	
occ_recent_index	ae(v)
else	
occ_symbol_escape	ae(v)
}	
}	

9.2.9.3 Syntax element semantics of a dictionary coded symbol

occ_histogram_hit specifies whether (when 1) or not (when 0) the coded symbol is present in the list of most probable symbols.

occ_histogram_index specifies the index of the coded symbol in the list of most probable symbols.

occ_recent_hit specifies whether (when 1) or not (when 0) the coded symbol is present in the most recently coded symbol list. When occ_recent_hit is not present, it shall be inferred to be 0.

occ_recent_index specifies the index of the coded symbol in the most recently coded symbol list.

occ_symbol_escape specifies the value of the decoded symbol when occ_histogram_hit and occ_recent_hit are both 0.

9.2.9.4 State variables

The dictionary codec is specified in terms of the following state variables; the index *dictIdx* identifies an instance of the dictionary codec:

— A 9×256 element array *DictsHistogram* of symbol occurrence histograms per dictionary instance; *DictsHistogram*[*dictIdx*][*sym*] is the cumulative count for the symbol *sym*.

— A 9×32 element array *DictsMostProb* of thirty-two most probable symbols per dictionary instance; *DictsMostProb*[*dictIdx*][*i*] is the *i*-th most probable symbol.

— A 9×16 element array *DictsRecent* of sixteen recently coded symbols per dictionary instance; *DictsRecent*[*dictIdx*][*i*] is a recently coded symbol that was not coded using the most probable symbols list.

— A 9 element array *DictsMostProbAge*; *DictsMostProbAge*[*dictIdx*] is the count of symbols since the last generation of the dictionary's most probable symbol list.

— A 9 element array *DictsMostProbMaxAge*; *DictsMostProbMaxAge*[*dictIdx*] is the maximum allowed age in symbols of the dictionary's most probable symbol list.

— A 9 element array *DictsNextEvictIdx*; *DictsNextEvictIdx*[*dictIdx*] is the index of the next element to be evicted from the dictionary's *DictsRecent* array.

9.2.9.5 Initial state

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

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

When slice_entropy_continuation is 1, initialization shall be performed by the parsing state restoration process (11.5.3.2).

Otherwise, (slice_entropy_continuation is 0), the dictionary state variables shall be initialized:

— Elements of *DictsMostProb* shall be initialized according to Table 17.

— Elements of *DictsHistogram* shall be set to 1 if they identify a symbol present in the corresponding most probable symbol sub-array. i.e. *DictsHistogram*[*dictIdx*][*i*] = 1 if *i* ∈ *DictsMostProb*[*dictIdx*]. All other elements shall be set to 0.

— Elements of *DictsRecent*, *DictsRecent*[*dictIdx*][*i*], shall be set to *i*.

— Elements of *DictsNextEvictIdx* and *DictsMostProbAge* shall be set to 0.

— Elements of *DictsMostProbMaxAge* shall be set to 16.

```
if (¬slice_entropy_continuation) {
  … /* Initialize DictsMostProb using Table 17 */

  for (dictIdx = 0; dictIdx < 9; dictIdx++) {
    for (i = 0; i < 16; i++)
      DictsRecent[dictIdx][i] = i

    for (i = 0; i < 32; i++) {
      symbol = DictsMostProb[dictIdx][i]
      DictsHistogram[dictIdx][symbol] = 1
    }

    DictsNextEvictIdx[dictIdx] = 0
    DictsMostProbAge[dictIdx] = 0
    DictsMostProbMaxAge[dictIdx] = 16
  }
}
```

Table 17 — Initial values of *DictsMostProb*[*dictIdx*][*i*]

i	*dictIdx*								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	5	85	128	64	85	16	170	170	255
1	17	255	32	128	170	64	10	255	223
2	34	170	64	192	255	17	42	128	239
3	68	64	16	204	119	80	8	160	251
4	160	80	192	136	127	128	138	136	127
5	136	252	80	68	254	68	15	168	247
6	12	223	160	170	87	32	255	204	119
7	80	84	48	200	223	85	2	240	253
8	192	117	68	85	95	81	14	250	63
9	21	68	8	196	117	84	136	192	191
10	10	247	136	255	245	192	11	238	221
11	48	221	176	4	213	48	175	162	254
12	3	4	2	8	247	51	32	234	238
13	170	192	240	80	93	4	238	223	95
14	168	128	144	160	234	34	47	138	175
15	162	174	17	240	69	240	191	254	240
16	204	253	208	208	238	1	34	10	85
17	85	204	224	76	21	136	239	186	187

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Table 17 *(continued)*

i	*dictIdx*								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
18	14	240	112	221	221	170	245	8	244
19	81	69	19	140	191	255	174	251	250
20	35	127	255	244	253	204	3	2	170
21	69	213	85	72	187	196	63	127	245
22	84	5	51	93	16	160	95	125	117
23	176	119	170	168	251	12	223	247	34
24	51	238	84	250	171	208	253	85	126
25	65	175	162	32	17	69	168	171	51
26	138	160	238	252	5	191	142	32	93
27	200	87	204	187	174	119	246	197	243
28	212	136	1	223	125	21	206	221	207
29	11	16	76	238	239	95	13	87	234
30	50	244	138	243	12	2	162	42	59
31	15	23	187	84	241	206	250	239	236

9.2.9.6 Selection of a dictionary instance

A dictionary instance shall be selected for each coded occupancy_byte syntax element according to the reduced occupied neighbourhood pattern, *OccNeighPatR*.

The following expressions are aliases used in the specification of operations on the selected dictionary instance:

```
OccDictHistogram[i]    := DictsHistogram[OccNeighPatR][i]
OccDictMostProb[i]     := DictsMostProb[OccNeighPatR][i]
OccDictRecent[i]       := DictsRecent[OccNeighPatR][i]
OccDictMostProbAge     := DictsMostProbAge[OccNeighPatR]
OccDictMostProbMaxAge  := DictsMostProbMaxAge[OccNeighPatR]
OccDictNextEvictIdx    := DictsNextEvictIdx[OccNeighPatR]
```

9.2.9.7 The value for occupancy_byte

The decoded value of the syntax element shall be:

— when occ_histogram_hit is 1: *OccDictMostProb*[occ_histogram_index];

— when occ_recent_hit is 1: *OccDictRecent*[occ_recent_index];

— otherwise (neither occ_histogram_hit nor occ_recent_hit is 1): occ_symbol_escape.

9.2.9.8 Update of dictionary state after each coded symbol

9.2.9.8.1 List of most recently coded symbols

After each coded occupancy_byte syntax element when occ_histogram_hit is 0, the syntax element value shall be used to update the list of most recently coded symbols.

If the syntax element value is already present in the list, its position in the list shall be exchanged with the symbol scheduled to be evicted (at index *OccDictNextEvictIdx*).

```
for (i = 0; i < 16; i++)
  if (OccDictRecent[i] == occupancy_byte) {
    OccDictRecent[i] = OccDictRecent[OccDictNextEvictIdx]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
    break
  }
```

The syntax element value shall be inserted into the list, replacing the symbol at index *OccDictNextEvictIdx*.

```
OccDictRecent[OccDictNextEvictIdx] = occupancy_byte
```

After updating the list of most recently coded symbols, the eviction index shall be incremented modulo 16.

```
OccDictNextEvictIdx = (OccDictNextEvictIdx + 1) % 16
```

9.2.9.8.2 Histogram of symbol counts

After each coded occupancy_byte syntax element, the histogram of symbol occurrences shall be updated and the age of the most probable symbol list shall be incremented.

```
OccDictHistogram[occupancy_byte]++
OccDictMostProbAge++
```

When the histogram count of symbols equal to occupancy_byte reaches 1024, all counts in the histogram shall be halved and any fractional parts discarded.

```
if (OccDictHistogram[occupancy_byte] == 1024)
  for (i = 0; i < 256; i++)
    OccDictHistogram[i] >>= 1
```

9.2.9.9 Generation of the most probable symbol list

When *OccDictMostProbAge* is equal to *OccDictMostProbMaxAge*, the most probable symbol list shall be recalculated from the histogram of symbol counts.

The most probable symbol list shall be a stable descending ordering of the *OccDictHistogram* array. Each element *OccDictMostProb*[*i*] shall be the index of the *i*-th largest element in the *OccDictHistogram* array. The ordering shall be such that the following conditions are true:

— *OccDictHistogram*[*OccDictMostProb*[*i*]] ≥ *OccDictHistogram*[*OccDictMostProb*[*i* + 1]], and

— *OccDictMostProb*[*i*] < *OccDictMostProb*[*i* + 1] when *OccDictHistogram*[*OccDictMostProb*[*i*]] is equal to *OccDictHistogram*[*OccDictMostProb*[*i* + 1]].

The age of the generated most probable symbol list shall be 0.

```
OccDictMostProbAge = 0
```

The maximum age of the generated most probable symbol list shall be the next value in the bounded geometric progression specified by the expression *OccDictMostProbMaxAgeNext*:

```
OccDictMostProbMaxAgeNext := Min(5 × OccDictMostProbMaxAge >> 2, 1024)
OccDictMostProbMaxAge = OccDictMostProbMaxAge
```

9.2.9.10 Resetting the histogram of symbol counts

The histogram of symbol counts shall be reset immediately after the first recalculation of the most probable symbol list in each level of the occupancy tree. Counts for symbols that are present in the most probable symbol list shall be set to 1; all other counts shall be set to 0.

```
if (OccDictMostProbAge == 0)
  if (… /* First occurrence in tree level */) {
    for (i = 0; i < 256; i++)
      OccDictHistogram[i] = 0

    for (i = 0; i < 32; i++) {
      symbol = OccDictMostProb[i]
      OccDictHistogram[symbol] = 1
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
    }
  }
```

9.2.9.11 Determination of *CtxIdxDictHg* for a bin of occ_histogram_index

Contextualization depends upon the reduced neighbourhood pattern, the bin index and the MSBs of the occ_histogram_index syntax element value.

Table 18 specifies the value *ctxIdxInc* for the bin. If *ctxIdxInc* is 'bypass', the value of *CtxIdxDictHg* shall be 'bypass'. Otherwise, the value of *CtxIdxDictHg* shall be 5 × *OccNeighPatR* + *ctxIdxInc*.

Table 18 — Values of *ctxIdxInc* for bins of the syntax element occ_histogram_index

MSBs of binarized occ_histogram_index	*BinIdx*				
	0	**1**	**2**	**3**	**4**
'000'	0	1	2	3	4
'001'	0	1	2	bypass	bypass
'01'	0	1	bypass	bypass	bypass
'1'	0	bypass	bypass	bypass	bypass

9.2.10 Bitwise occupancy coding

9.2.10.1 General

This subclause applies when occtree_bitwise_coding is 1.

The neighbourhood-permuted node occupancy bitmap shall be coded as a sequence of individual occupancy_bit syntax elements. Coding uses constraints on occupancy to infer the value of certain occupancy_bit syntax elements.

Entropy coding of each coded bit is contextualized by a combination of the coded bit index, previously coded occupancy_bit syntax elements, the reduced occupied neighbourhood pattern, the number of spatially adjacent child nodes in neighbouring nodes and a ternary prediction based upon the presence of neighbouring nodes.

9.2.10.2 Correspondence between the node occupancy bitmap and occupancy_bit

Bits of the neighbourhood-permuted node occupancy bitmap shall be coded in the order specified by Table 19. Each occupancy_bit[*cbIdx*] syntax element codes the bit *OccBitCodingOrder*[*cbIdx*].

The expression *OccBitIdx*[*cbIdx*] is the bit position in the node occupancy bitmap of the bit coded by occupancy_bit[*cbIdx*]. For example, when *OccNeighPat* is 17, occupancy_bit[6] corresponds to the second bit (b_1) of the node occupancy bitmap.

```
OccBitIdx[cbIdx] := OccBitIdxFromNpBit[OccBitCodingOrder[cbIdx]]
```

The expression *OccBitLocC*[*cbIdx*][*k*] is the node-relative child location represented by occupancy_bit[*cbIdx*].

```
OccBitLocC[cbIdx][k] := OccLocC[OccBitIdx[cbIdx]][k]
```

The expression *OccBitMap* is the node occupancy bitmap.

```
OccBitMap :=
  OccBitMap = 0
  for (cbIdx = 0; cbIdx < 8; cbIdx++)
    OccBitMap = OccBitMap | (occupancy_bit[cbIdx] << OccBitIdx[cbIdx])
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Table 19 — Order for coding bits of the neighbourhood-permuted node occupancy bitmap as occupancy_bit[*cbIdx*]

cbIdx	0	1	2	3	4	5	6	7
***OccBitCodingOrder*[*cbIdx*]**	1	7	5	3	2	6	4	0

9.2.10.3 Presence of occupancy_bit

An occupancy bitmap bit shall not be coded if its value can be inferred to be set or unset. The expression *OccBitPresent*[*cbIdx*] specifies whether occupancy_bit[*cbIdx*] is present.

```
OccBitPresent[cbIdx] := ¬(OccBitInferUnset[cbIdx] || OccBitInferSet[cbIdx])
```

9.2.10.4 Inference of an unset bit

An occupancy bitmap bit shall be inferred to be 0 when either:

— the bit represents an invalid child according to the node's coded axes (9.2.2.3), or

— the bit represents a child within an unoccupied plane signalled by planar occupancy coding.

When the expression *OccBitInferUnset*[*cbIdx*] is equal to 1, occupancy_bit[*cbIdx*] shall be inferred to be 0.

```
OccBitInferUnset[cbIdx] :=
     ¬AxisCoded[0] && OccBitLocC[cbIdx][0]
  || ¬AxisCoded[1] && OccBitLocC[cbIdx][1]
  || ¬AxisCoded[2] && OccBitLocC[cbIdx][2]
  || ¬PlanarFreeAxis[0] && OccBitLocC[cbIdx][0] ^ occ_plane_pos[0]
  || ¬PlanarFreeAxis[1] && OccBitLocC[cbIdx][1] ^ occ_plane_pos[1]
  || ¬PlanarFreeAxis[2] && OccBitLocC[cbIdx][2] ^ occ_plane_pos[2]
```

9.2.10.5 Inference of a set bit

An occupancy bitmap bit occupancy_bit[*cbIdx*] shall be inferred to be 1, as specified by the expression *OccBitInferSet*[*cbIdx*], when:

— the bit is the last present bit and all previous coded bits are 0, or

— the bit is the penultimate present bit, all previous coded bits are 0 and the node is required to have two child nodes, or

— the bit is in a plane identified as occupied by planar occupancy coding, the bit is the last bit in the plane and all previous bits in the plane are 0.

```
OccBitInferSet[cbIdx] :=
     PlanarEligible[0] && PopCnt(OccKnownZero & (0x0F << 4 × OccBitLocC[cbIdx][0])) == 3
  || PlanarEligible[1] && PopCnt(OccKnownZero & (0x33 << 2 × OccBitLocC[cbIdx][1])) == 3
  || PlanarEligible[2] && PopCnt(OccKnownZero & (0x55 << 1 × OccBitLocC[cbIdx][2])) == 3
  || cbIdx == 6 && PopCnt(OccKnown) == 0 && OccMinChildren == 2
  || cbIdx == 7 && PopCnt(OccKnown) == 0
```

The expression *OccKnownMask* is a bit mask that identifies bits of the node occupancy bitmap that have a known value prior to coding occupancy_bit[*cbIdx*].

```
OccKnownMask :=
  OccKnownMask = 0
  for (i = 0; i < cbIdx; i++)
    OccKnownMask |= 1 << OccBitIdx[i]
  for (i = 0; i < 8; i++)
    OccKnownMask |= OccBitInferUnset[i] << OccBitIdx[i]
```

The expression *OccKnown* is the partially coded node occupancy bitmap comprising the bits coded prior to occupancy_bit[*cbIdx*].

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
OccKnown :=
  OccKnown = 0
  for (i = 0; i < cbIdx; i++)
    OccKnown |= occupancy_bit[i] << OccBitIdx[i]
```

The expression *OccKnownZero* is a bitmap of occupancy bits that are known to be 0.

```
OccKnownZero := (0xFF ^ OccKnown) & OccKnownMask
```

9.2.10.6 Contextualization

9.2.10.6.1 General

Contextualization of occupancy_bit syntax elements is a two-stage process. First, context discriminators are used to select a demi-CPM. Then, the demi-CPM is used to select the CPM that codes the syntax element.

A demi-CPM is an 8-bit unsigned integer that models the probability of a coded zero-valued occupancy_bit syntax element.

NOTE The values 0, 128 and 256 represent the probability of a zero bin as impossible, equiprobable and certain, respectively. The values 0 and 256 can never be attained due to the operation of the probability models' update process.

9.2.10.6.2 State variables

Context selection is specified in terms of the following state variable:

— The array *OccCtxSel*; *OccCtxSel*[*selNeigh*][*cbIdx*][*selSib*][*selAdj*][*selPred*] is a demi-CPM, contextualized by *selNeigh*, *cbIdx*, *selSib*, *selAdj* and *selPred*.

9.2.10.6.3 Initial state

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

When slice_entropy_continuation is 1, initialization shall be performed according to the parsing state restoration process (11.6.2.2).

Otherwise (slice_entropy_continuation is 0), all elements of *OccCtxSel* shall be set to 127.

9.2.10.6.4 Determination of *CtxIdxOccBit* for the syntax element occupancy_bit

The expression *OccCtxSelVar* specifies the demi-CPM for the syntax element occupancy_bit[*CbIdx*] using:

— *SelNeigh*, the reduced occupied neighbourhood context discriminator (9.2.10.6.6);

— *SelSib*, the sibling occupancy context discriminator (9.2.10.6.7);

— *SelAdj*, the adjacent child neighbour context discriminator (9.2.10.6.8);

— *SelPred*, the neighbour-predicted occupancy context discriminator (9.2.10.6.9).

```
OccCtxSelVar := OccCtxSel[SelNeigh][CbIdx][SelSib][SelAdj][SelPred]
```

The CPM index, *CtxIdxOccBit*, shall be the value of the demi-CPM exclusive of the bottom three bits:

```
CtxIdxOccBit := OccCtxSelVar >> 3
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.10.6.5 Update after each coded occupancy_bit syntax element

After each coded occupancy_bit syntax element, its demi-CPM shall be updated. The update specified by Table 20 supplies a value for incrementing or decrementing the probability of a zero bin based upon the upper four bits of the demi-CPM's value:

```
if (OccBitPresent[CbIdx])
  if (occupancy_bit[CbIdx])
    OccCtxSelVar += OccCtxSelUpdate[255 - OccCtxSelVar >> 4]
  else
    OccCtxSelVar -= OccCtxSelUpdate[OccCtxSelVar >> 4]
```

Table 20 — Values of *OccCtxSelUpdate*[*i*]

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
***OccCtxSelUpdate*[*i*]**	0	1	1	2	4	7	9	11	14	16	19	23	22	18	13	6

9.2.10.6.6 Reduced occupied neighbourhood context discriminator

The reduced occupied neighbourhood context discriminator shall distinguish between different reduced occupied neighbourhood patterns (*OccNeighPatR*) depending upon the coded bit index (*CbIdx*) as specified by Table 21 as the expression *SelNeigh*.

Table 21 — Discriminated values *SelNeigh* for occupancy_bit[*CbIdx*] and *OccNeighPatR*

CbIdx	***OccNeighPatR***								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0 .. 3	0	1	2	3	4	5	6	7	8
4 .. 5	0	1	2	3	1	2	3	4	4
6	0	1	1	2	2	1	2	2	2
7	0	1	1	1	1	1	1	1	1

9.2.10.6.7 Sibling occupancy context discriminator

The sibling occupancy context discriminator shall distinguish between arrangements of previously coded/inferred siblings for the node coded by occupancy_bit[*CbIdx*] as specified by the expression *SelSib*:

— If there are no nodes present in the occupied neighbourhood pattern, discrimination shall be by the number of present child nodes identified by the syntax elements occupancy_bit[*i*] with *i* < *CbIdx*.

— If there is at least one node present in the occupied neighbourhood pattern, discrimination shall be by the combination of present child nodes identified by the syntax elements occupancy_bit[*i*] with *i* < *CbIdx*.

```
SelSib := OccNeighPat ? occPrevBits : PopCnt(occPrevBits)
```

The expression *occPrevBits* is the concatenation of occupancy_bit[*i*] for *i* < *CbIdx*.

```
occPrevBits :=
  occPrevBits = 0
  for (i = 0; i < CbIdx; i++)
    occPrevBits |= occupancy_bit[i] << i
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.10.6.8 Adjacent child neighbour context discriminator

The adjacent child neighbour context discriminator for child the node coded by occupancy_bit[*CbIdx*] is specified by the expression *SelAdj*. When adjacent child neighbour contextualization is enabled (occtree_adjacent_child_enabled is 1), it distinguishes between contexts by:

— the number of child nodes from available, previously coded nodes in the same tree level (9.2.7.2) that adjoin the coded child by a face; and

— whether any of the available, previously coded nodes in the same tree level that adjoin the coded child node do not have a child node that also adjoins the coded child.

An example is illustrated in Figure 10. The child node C_0 of the coded node N is adjoined by a single child node. There are two available previously coded nodes that adjoin C_0, one of which does not contain a child node that also adjoins C_0.

```
SelAdj := occtree_adjacent_child_enabled
  ? 2 × Min(2, adjCntC) + ((cbIdx ≤ 4 || adjCntC == 1) && adjUnocc)
  : 0
```

The expression *adjOccN*[*k*] identifies whether there is a spatially adjacent node along the *k*-th axis within the occupied neighbourhood availability window. Values for the expressions *ds*, *dt* and *dv* are specified in Table 22 for each axis *k*.

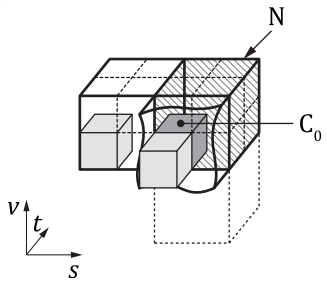
```
adjOccN[k] := ¬OccBitLocC[CbIdx][k] && OccNeigh[Ns + ds][Nt + dt][Nv + dv]
```

The expression *adjOccC*[*k*] identifies whether there is a spatially adjacent child node along the *k*-th axis within the occupied neighbourhood availability window. Values for the expressions *ds*, *dt* and *dv* are specified in Table 22 for each axis, *k*.

```
adjOccC[k] := adjOccN[k] && OccNodePresent[Dpth + 1][cs + ds][ct + dt][cv + dv] ≠ 0
  where
    cs := NsC + OccBitLocC[CbIdx][0]
    ct := NtC + OccBitLocC[CbIdx][1]
    cv := NvC + OccBitLocC[CbIdx][2]
```

Table 22 — Relative neighbour locations (*ds*, *dt*, *dv*) used in the computation of *adjOccN*[*k*] and *adjOccC*[*k*]

k	*ds*	*dt*	*dv*
0	−1	0	0
1	0	−1	0
2	0	0	−1



	S	T	V
adjOccC[k]	0	1	0
adjOccN[k]	1	1	0

Key

N coded node

C_0 contextualized child with *OccBitIdx*[*cbIdx*] = 0

Figure 10 — Example of adjacent child neighbour context discrimination

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

The expressions *adjCntN* and *adjCntC* are the number of spatially adjacent nodes and child nodes, respectively, that are within the occupied neighbourhood availability window.

```
adjCntN := adjOccN[0] + adjOccN[1] + adjOccN[2]
adjCntC := adjOccC[0] + adjOccC[1] + adjOccC[2]
```

The expression *adjUnocc* identifies whether there exists a spatially adjacent node within the occupied neighbourhood availability window that does not have a child node spatially adjacent to the coded child.

```
adjUnocc := adjCntN ≠ adjCntC
```

9.2.10.6.9 Neighbour-predicted-occupancy context discriminator

9.2.10.6.9.1 General

The neighbour-predicted-occupancy context discriminator shall, for eligible nodes (9.2.10.6.9.2), distinguish between three predictions for the presence of the child node coded by occupancy_bit[*CbIdx*]. The discriminator is specified by the expression *SelPred*. The three predictions are that the node is present, not present, or that it is unpredictable.

```
SelPred := SelPredEligible ? OccIntraPred : 0
```

9.2.10.6.9.2 Eligibility

The discriminator shall only form a prediction for eligible nodes as specified by the expression *SelPredEligible*. Eligible nodes shall have both:

— three free axes; and

— a maximum log2 node dimension less than occtree_intra_pred_max_nodesize_log2.

```
SelPredEligible :=
  OccFreeAxisCnt == 3 && MaxVec(NodeSizeLog2) < occtree_intra_pred_max_nodesize_log2
```

9.2.10.6.9.3 Occupancy prediction

Occupancy prediction generates a ternary prediction for the presence of a child node identified by occupancy_bit[*CbIdx*] of a coded node. The prediction is specified by the expression *OccIntraPred*. It is based upon how many of the nodes that neighbour the coded node also adjoin the volume of the identified child node by a face, edge or corner (as illustrated by Figure 11):

— A child node shall be predicted to be not present if there are two or fewer adjoining nodes.

— A child node shall be predicted to be present if there is at least a threshold number of adjoining nodes. The threshold is specified by the expression *OccIntraThreshold*. The threshold is four nodes unless there are more than 13 neighbouring nodes; in which case the threshold is 5 nodes.

— Otherwise, the presence is unpredictable.

NOTE The size of the child node volume is half the size of the neighbour nodes' in each dimension.

```
OccIntraPred := (OccAdjCnt ≤ 2) + 2 × (OccAdjCnt ≥ OccIntraThreshold)
OccIntraThreshold := 4 + (OccNeighCnt ≥ 14)
```

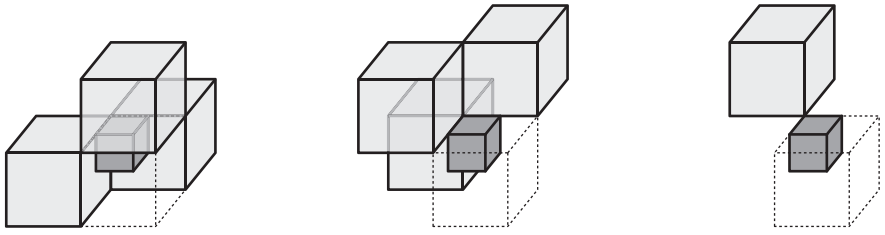


Figure 11 — Nodes that adjoin a child node by a face, edge or corner

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

The expression *OccAdj*[*ds*][*dt*][*dv*] identifies whether a neighbouring node with a relative tree location (*ds*, *dt*, *dv*) to the coded node would adjoin the identified child volume.

```
OccAdj[ds][dt][dv] := (OccBitIdx[CbIdx] & adjMask) == adjLoc
  where
    adjMask := Morton(ds ≠ 0, dt ≠ 0, dv ≠ 0)
    adjLoc  := Morton(ds > 0, dt > 0, dv > 0)
```

The expression *OccAdjCnt* is the number of neighbours that adjoin the identified child volume.

```
OccAdjCnt := SumN26[neighAdj]
  where
    neighAdj[ds][dt][dv] := OccNeigh[Ns + ds][Nt + dt][Nv + dv] && OccAdj[ds][dt][dv]
```

The expression *OccNeighCnt* is the number of nodes that neighbour the coded node.

```
OccNeighCnt := SumN26[neighRel]
  where
    neighRel[ds][dt][dv] := OccNeigh[Ns + ds][Nt + dt][Nv + dv]
```

The expression *SumN26*[*expr*] sums the result of applying *expr* to the relative tree location of each of the 26 possible neighbouring nodes.

```
SumN26[expr] :=
  SumN26 = 0
  for (ds = −1; ds ≤ 1; ds++)
    for (dt = −1; dt ≤ 1; dt++)
      for (dv = −1; dv ≤ 1; dv++)
        if (ds ≠ 0 && dt ≠ 0 && dv ≠ 0)
          SumN26 += expr[ds][dt][dv]
```

9.2.11 Planar occupancy coding

9.2.11.1 General

This subclause applies when occtree_planar_enabled is 1.

Planar occupancy coding decomposes the node occupancy bitmap into axis-aligned planes. Each coded axis has two perpendicular planes that child nodes can occupy as illustrated by Table 23. For each planar-eligible coded axis (9.2.11.5), planar occupancy coding specifies whether one of the two planes is unoccupied. Plane occupancy is then used by bitwise occupancy coding to constrain and infer the coding of bits in the node occupancy bitmap.

There shall be at least one child node in each occupied plane.

NOTE The definition of an occupancy tree node requires that at least one plane is occupied along each coded axis.

For example, if a node has three planar-eligible coded axes, there is a total of six axis-aligned planes. Along the S axis (k = 0), information about the occupied state of the two T-V planes is coded.

Table 23 — Plane, perpendicular to each planar axis, *k*

k	Planar axis	Plan axes
0	S	T-V
1	T	S-V
2	V	S-T

9.2.11.2 Syntax element semantics

occ_single_plane[k] specifies, when present, whether (when 1) the locations of child nodes in the node occupancy bitmap shall occupy a single plane or (when 0) both planes perpendicular to the k-th axis. When equal to 1, the location of the single plane is specified by occ_plane_pos[k]. When not present,

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

the child nodes can be located in either or both planes perpendicular to the *k*-th axis. The number of occupied planes is illustrated in Table 24.

Table 24 — Interpretation of *PlanarEligible*[*k*] and occ_single_plane[*k*]

PlanarEligible[*k*]	occ_single_plane[*k*]	№ occupied planes	*PlanarMinPlanes*[*k*]
0	not present	1 or 2	1
1	0	2	2
1	1	1	1

occ_plane_pos[*k*] specifies the node-relative location along the *k*-th axis for the occupied plane specified by occ_single_plane[*k*] equal to 1.

Examples of planar occupancy are illustrated in Figure 12. Each entry shows the child node indices for a node with three coded axes. If an axis *k* is eligible for planar coding and occ_single_plane[*k*] is 0, the two occupied planes are marked with a dotted line. If occ_single_plane[*k*] is 1, the unoccupied plane is marked by hatching and a dashed (red) line, with its child indices in grey; the occupied plane is not marked for clarity. In the case where three axes are eligible and each has occ_single_plane equal to 1, there is only a single child node present; its location is fully constrained by the planes.

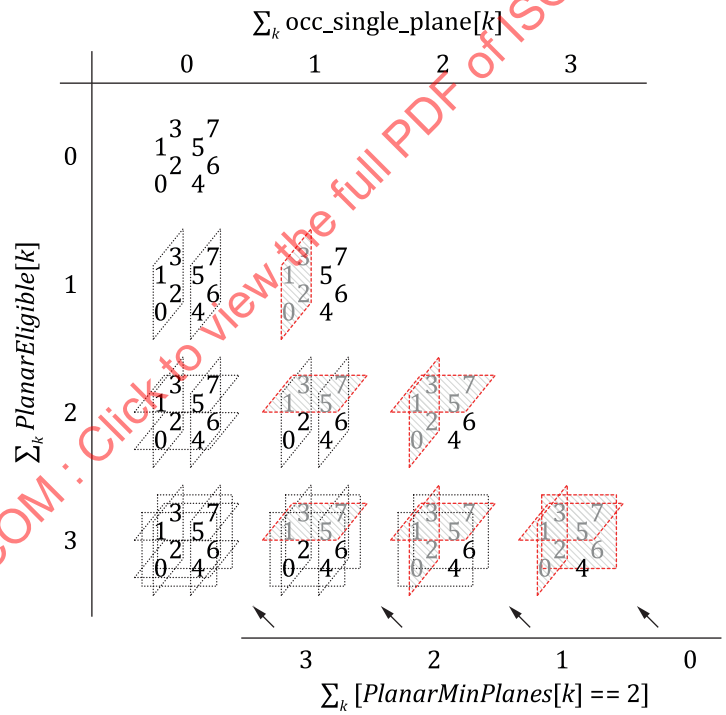


Figure 12 — Example planar constraints on node occupancy bitmap

9.2.11.3 Minimum number of child nodes

Planar occupancy coding can require that the coded node has a minimum of either one or two child nodes, as specified by the expression *PlanarMinChildren*.

A node shall have at least two child nodes if, for any planar-eligible axis *k*, occ_single_plane[*k*] specifies that there shall be a minimum of two occupied planes; otherwise, the node shall have at least one child node.

```
PlanarMinChildren := MaxVec(PlanarMinPlanes)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.11.4 Free axes

A free axis is a coded axis whose occupancy is not constrained to a single plane by occ_single_plane, as specified by the expression *PlanarFreeAxis*[*k*].

```
PlanarFreeAxis[k] := ¬PlanarEligible[k] || PlanarMinPlanes[k] == 2
```

In Figure 12, a free axis is an axis with either two marked planes or no marked planes. In the case where three axes are eligible, of which two have occ_single_plane equal to 1, *PlanarMinChildren* is 2.

9.2.11.5 Per-axis eligibility

9.2.11.5.1 Condition

Only certain axes are eligible for planar occupancy coding. Eligibility for the *k*-th axis is specified by the expression *PlanarEligible*[*k*]. Eligibility shall be determined after any applicable update to the eligibility state (9.2.11.5.5).

An axis is not eligible for planar coding when either planar occupancy coding is disabled, or the axis is not coded. Otherwise, the determination of eligibility depends upon the use of the angular coding and whether the node is eligible for angular contextualization (9.2.13.7.2) as specified in Table 25.

```
PlanarEligible[k] :=
  occtree_planar_enabled && AxisCoded[k]
  && (geom_angular_enabled ? PlanarEligibleByAng[k] : PlanarEligibleByRate[k])
PlanarEligibleByAng[k] :=
  AngularEligible ? k == 2 || k == AzimuthAxis
                  : k == 2 && PlanarEligibleByRate[2]
```

Table 25 — Method to determine eligibility for an axis

Axis	*k*	Angular coding disabled	Angular coding enabled	
			***AngularEligible* == 0**	***AngularEligible* == 1**
S	0	*PlanarEligibleByRate*[0]	Not eligible	*AzimuthAxisIsS*
T	1	*PlanarEligibleByRate*[1]	Not eligible	*AzimuthAxisIsQ*
V	2	*PlanarEligibleByRate*[2]	*PlanarEligibleByRate*[2]	Eligible

Axes whose eligibility is determined by the expression *PlanarEligibeByRate*[*k*] are eligible if:

— the coded node is estimated to have fewer than, approximately, two other sibling nodes, and

— the estimated probability that the axis has only a single occupied plane is greater than the threshold specified by *PlanarRateThreshold*[*k*].

Both estimates are based upon exponential moving averages over previously coded nodes (9.2.11.5.3).

```
PlanarEligibleByRate[k] :=
  OccupancyDensity < 3 × 1024 && PlanarRate[k] > PlanarRateThreshold[k]
```

The threshold *PlanarRateThreshold*[*k*] is chosen according to the relative order of likelihood that each axis would have a single occupied plane. The ordering *PlaneThresholdIdx*[*k*] is specified by Table 26.

```
PlanarRateThreshold[k] := 16 × occtree_planar_threshold[PlaneThresholdIdx[k]]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Table 26 — Value for *PlaneThresholdIdx*[*k*] according to order of *PlanarRate*

Condition	*k*		
	0	**1**	**2**
PlanarRate[0] ≥ *PlanarRate*[1] ≥ *PlanarRate*[2]	0	1	2
PlanarRate[0] ≥ *PlanarRate*[2] > *PlanarRate*[1]	0	2	1
PlanarRate[1] > *PlanarRate*[0] ≥ *PlanarRate*[2]	1	0	2
PlanarRate[1] ≥ *PlanarRate*[2] > *PlanarRate*[0]	2	0	1
PlanarRate[2] > *PlanarRate*[0] ≥ *PlanarRate*[1]	1	2	0
PlanarRate[2] > *PlanarRate*[1] > *PlanarRate*[0]	2	1	0

9.2.11.5.2 State variables

Eligibility is specified in terms of the following state variables:

— The sparse array *NodeOccMap* of node occupancy bitmaps; *NodeOccMap*[*dpth*][*ns*][*nt*][*nv*] is the coded node occupancy bitmap of the node located at (*ns*, *nt*, *nv*) in the tree level at depth *dpth*.

— The array *PlanarRate*; *PlanarRate*[*k*] is the estimated probability that only a single plane along the *k*-th axis is occupied.

— The variable *OccupancyDensity*, an estimate for the number of child nodes.

9.2.11.5.3 State exponential moving average model

The estimates of the per-axis single-plane probability and the mean number of child nodes per node are exponential moving averages. The calculation of the updated average value from the current average *cur* and the value *next* is specified by the expression *ExpMovAvg*[*cur*][*next*].

```
ExpMovAvg[cur][next] := DivExp2Fz(255 * cur + next, 8)
```

The estimated single-plane probability models the probability interval [0, 1] by the range 128 .. 1 921.

The estimated number of child nodes models the interval [1, 8] by the range 1 152 .. 8 065.

9.2.11.5.4 Initial state

At the start of every occupancy tree syntax structure, *PlanarRate* and *OccupancyDensity* shall be initialized:

```
for (k = 0; k < 3; k++)
  PlanarRate[k] = 1024
OccupancyDensity = 4096
```

NOTE The parsing state restoration process (11.6.2.2) reinitializes these state variables at the start of certain occupancy tree levels. It does not initialize them at the start of the occupancy tree.

9.2.11.5.5 State update at the start of every group of sibling nodes

After the root node, the eligibility state shall be updated at the start of every subsequent group of sibling nodes, before any evaluation of *PlanarEligible*[*k*].

The sibling nodes are identified by their parent node's occupancy, as expressed by *OccupancyMapP*.

```
OccupancyMapP := NodeOccMap[Dpth - 1][NsP][NtP][NvP]
```

The number of occupied sibling nodes shall be used to update the *OccupancyDensity* estimate.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
numSiblings = PopCnt(OccupancyMapP)
OccupancyDensity = ExpMovAvg[OccupancyDensity][1024 × numSiblings]
```

The parent node's occupancy information shall be used to determine, along each axis, the presence of a single occupied plane and to update the corresponding planar probability estimate *PlanarRate*[*k*]. The bit masks *OccPlaneMask*[*planeLoc*][*k*] that identify planes in an occupancy bitmap are specified by Table 27.

```
for (k = 0; k < 3; k++) {
  plane0 = (OccupancyMapP & OccPlaneMask[0][k]) ≠ 0
  plane1 = (OccupancyMapP & OccPlaneMask[1][k]) ≠ 0
  hasSinglePlane = plane0 ^ plane1
  PlanarRate[k] = ExpMovAvg[PlanarRate[k]][2048 × hasSinglePlane]
}
```

Table 27 — Bit masks *OccPlaneMask*[*planeLoc*][*k*] that identify planes of a node occupancy bitmap

k	0	1	2
***OccPlaneMask*[0][*k*]**	0x0F	0x33	0x55
***OccPlaneMask*[1][*k*]**	0xF0	0xCC	0xAA

9.2.11.5.6 State update at the end of each node

This subclause applies at the end of each occupancy tree node syntax structure.

The per-axis single-plane probability shall be updated according to the signalled number of planes for each eligible axis:

```
for (k = 0; k < 3; k++)
  if (PlanarEligible[k])
    PlanarRate[k] = ExpMovAvg[PlanarRate[k]][2048 × occ_single_plane[k]]
```

The node occupancy is recorded for use by state updates in the next tree level:

```
NodeOccMap[Dpth][Ns][Nt][Nv] = OccupancyMap
```

9.2.11.6 Previous coded node for contextualization of occ_plane_pos

9.2.11.6.1 General

This subclause does not apply when occtree_planar_buffer_disabled is 1.

Planar contextualization of occ_plane_pos[*k*] can use the following information about the previous planar-eligible coded node that is located in the same plane (9.2.11.6.2) as the coded node:

— The zone within the plane that the node resides (9.2.11.6.3).

— The values for occ_single_plane[*k*] and occ_plane_pos[*k*].

9.2.11.6.2 Identification of the plane

The plane normal to the *k*-th axis of a coded node is identified by its location along the axis modulo 2^{14}.

```
PlanarNodeAxisLoc[k] := Nloc[k] & 0x3FFF
```

9.2.11.6.3 Zone within a plane

The plane normal to the *k*-th axis of a coded node is partitioned into zones according to the ℓ^{∞} norm of the node location within the plane. The expression *PlanarNodeZone*[*k*] identifies the zone for the coded node.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
PlanarNodeZone[k] :=
  k == 0 ? Max(Nt & 0xF8, Nv & 0xF8) >> 3 :
  k == 1 ? Max(Ns & 0xF8, Nv & 0xF8) >> 3 :
  k == 2 ? Max(Ns & 0xF8, Nt & 0xF8) >> 3 : na
```

Figure 13 illustrates the partitioning of an S-T plane (k = 0) according to node location.

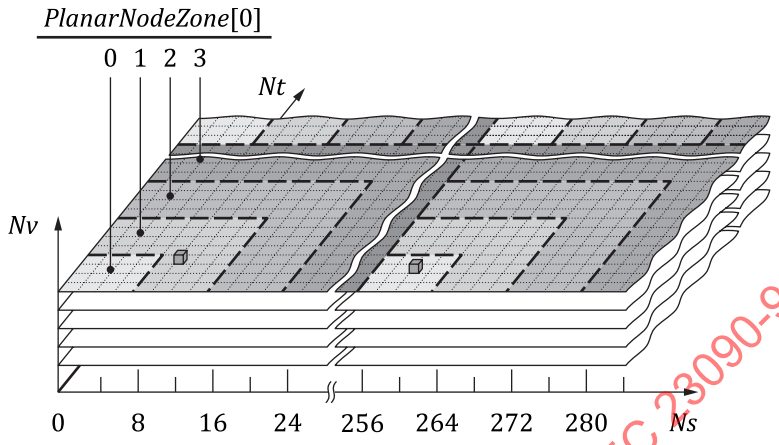


Figure 13 — S-T plane divided into zones

9.2.11.6.4 State variables

Information about previous planar-eligible coded nodes is specified in terms of the following state variables; the indexes k and *axisLoc* identify the location of a plane along the k-th axis:

— The array *PrevPlanarNodeZone*; *PrevPlanarNodeZone*[k][*axisLoc*] is the plane zone of the previous planar-eligible node in the identified plane.

— The array *PrevOccSinglePlane*; *PrevOccSinglePlane*[k][*axisLoc*] is the value of occ_single_plane[k] for the previous planar-eligible node in the identified plane.

— The array *PrevOccPlanePos*; *PrevOccPlanePos*[k][*axisLoc*] is the value of occ_plane_pos[k] for the previous planar-eligible node in the identified plane.

9.2.11.6.5 Initial state

At the start of every occupancy tree level, every element of *PrevOccSinglePlane* shall be initialized to 0.

9.2.11.6.6 State update at the end of each node

After each occupancy_tree_node syntax structure, the state shall be updated for each planar-eligible axis:

```
for (k = 0; k < 3; k++)
  if (PlanarEligible[k]) {
    PrevPlanarNodeZone[k][PlanarNodeAxisLoc[k]] = PlanarNodeZone[k]
    PrevOccSinglePlane[k][PlanarNodeAxisLoc[k]] = occ_single_plane[k]

    if (occ_single_plane[k])
      PrevOccPlanePos[k][PlanarNodeAxisLoc[k]] = occ_plane_pos[k]
  }
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.11.7 Determination of *CtxIdxPlanePos* for occ_plane_pos[*k*]

9.2.11.7.1 Case for angular-ineligible axes

Contextualization of occ_plane_pos[*k*] for nodes not eligible for angular contextualization (*AngularEligible* is 0) is specified by the expression *CtxIdxPlanePos*.

```
CtxIdxPlanePos :=
  occtree_planar_buffer_disabled || ¬PrevOccSinglePlane[k][PlanarNodeAxisLoc[k]]
    ? adjPlaneCtxInc
    : 12 × k + 4 × adjPlaneCtxInc + 2 × zoneCtxInc + prevPlanePosCtxInc + 3
```

The expression *adjPlaneCtxInc* discriminates by whether nodes have adjoining neighbours on a single side along the *k*-th axis, and if so, on which of the two sides they are present. Adjoining neighbours are:

— those along the *k*-th axis identified by the corresponding bits of the occupied neighbourhood pattern (*adjNeighHL*); and

— when the node is in the lower *k*-th axis plane of its parent, the sibling nodes in the corresponding upper plane (identified by *OccPlaneMask*[1][*k*]).

```
adjPlaneCtxInc := (adjNeighHL | sibPlaneH << 1) % 3
  where
    adjNeighHL := (OccNeighPat >> 2 × k) & 3
    sibPlaneH  := (Nloc[k] & 1) ≠ 1 && (OccupancyMapP & OccPlaneMask[1][k]) ≠ 0
```

NOTE Whenever occtree_coded_axis[*Dpth* − 1][*k*] is 0, *sibPlaneH* is always 0.

If occtree_planar_buffer_disabled is 0, contextualization uses information about the previous planar-eligible node in the plane identified by the coded node location (9.2.11.6.2).

The expression *zoneCtxInc* discriminates by whether the coded node is within ±1 zones of the identified previous node.

```
zoneCtxInc := Abs(a − b) > 1
  where
    a := PrevPlanarNodeZone[k][PlanarNodeAxisLoc[k]]
    b := PlanarNodeZone[k]
```

The expression *prevPlanePosCtxInc* discriminates by the occupied plane position of the identified previous node.

```
prevPlanePosCtxInc := PrevOccPlanePos[k][PlanarNodeAxisLoc[k]]
```

9.2.11.7.2 Case for angular-eligible axes

Contextualization of occ_plane_pos[*k*] for nodes eligible for angular contextualization (*AngularEligible* is 1) is specified by 9.2.13.7.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

9.2.12 Direct nodes

9.2.12.1 General

This subclause applies when occtree_direct_coding_mode is not 0.

Certain occupancy tree nodes may immediately code point positions as a direct node, instead of coding a node occupancy bitmap for subsequent traversal. A direct node can represent either two distinct point positions, or a single position that is identical for every represented point.

A direct node codes a position as a residual relative to the node position.

© ISO/IEC 2023 – All rights reserved

Direct coding is limited to nodes that are both eligible and not prohibited by the planar direct node rate limit. Eligibility shall be determined for each occupancy tree node based upon the degree of spatial isolation as specified in 9.2.12.3.

9.2.12.2 Syntax element semantics

occ_direct_node equal to 1 specifies that the occupancy tree node is a direct node that codes the position of at least one point. When occ_direct_node is not present, it shall be inferred to be 0.

direct_point_cnt_eq2 equal to 1 specifies that the direct node codes two point positions. direct_point_cnt_eq2 equal to 0 specifies that the occupancy tree node codes a single point position for one or more points.

direct_dup_point_cnt plus 1 specifies, when present, the number of points the direct node represents when direct_point_cnt_eq2 is 0. When direct_dup_point_cnt is not present, it shall be inferred to be 0.

direct_joint_prefix[*k*] specifies a sequence of identical MSBs in the *k*-th component of two coded position residuals. The MSB position of the syntax element value indicates the number of position bits coded by the syntax element and does not form part of the reconstructed point position.

direct_joint_diff_bit[*k*] specifies, when direct_joint_prefix[*k*] is present, the value of a bit in the binary representation of the *k*-th component of the two coded position residuals. The bit is the most significant non-identical bit of the two coded residual components. Its value is that for the first point. When direct_joint_diff_bit[*k*] is not present, it shall be inferred to be 0.

direct_rem[*dnPt*][*k*], **direct_rem_st_ang**[*dnPt*] and **direct_rem_v_ang**[*dnPt*] specify the remaining position bits of the *dnPt*-th point. When present, direct_rem codes the *k*-th component, direct_rem_st_ang either the *s*- or *t*-component depending upon the node location and direct_rem_v_ang codes the *v*-component.

beam_idx_resid_abs[*dnPt*] and **beam_idx_resid_sign**[*dnPt*] together specify the index of an enumerated beam relative to a per-node prediction. The beam is used in the contextualization of the syntax elements direct_rem_st_ang[*dnPt*] and direct_rem_v_ang[*dnPt*].

9.2.12.3 Eligibility

9.2.12.3.1 Decision for each occupancy tree node

Only certain occupancy tree nodes are eligible to be direct nodes. They are specified by the expression *DirectModeEligible*. An eligible node:

— is not the root node;

— is not the root node of a fully quantized subtree (9.2.14.2.6); and

— meets one of the following mode-dependent conditions:

  — when occtree_direct_coding_mode is 1: if there are no nodes in the occupied neighbourhood pattern of the parent node, the coded node has no siblings and the parent node has at most one sibling;

  — when occtree_direct_coding_mode is 2: if there are no nodes in the occupied neighbourhood pattern of the parent node;

  — when occtree_direct_coding_mode is 3: if the coded node has at least one sibling.

```
DirectModeEligible := occtree_direct_coding_mode > 0
  && Dpth > 0
  && MaxVec(QuantizedNodeSizeLog2) > 0
  && (occtree_direct_coding_mode ≠ 1 || DirectMode1Eligible)
  && (occtree_direct_coding_mode ≠ 2 || DirectMode2Eligible)
  && (occtree_direct_coding_mode ≠ 3 || DirectMode3Eligible)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
DirectMode1Eligible := OccNeighPatEq0[Dpth - 1][NsP][NtP][NvP]
  &&              OccNodeChildCnt[Dpth - 1][NsP][NtP][NvP] == 1
  && (Dpth < 2 || OccNodeChildCnt[Dpth - 2][NsG][NtG][NvG] ≤ 2)

DirectMode2Eligible := OccNeighPatEq0[Dpth - 1][NsP][NtP][NvP]

DirectMode3Eligible := OccNodeChildCnt[Dpth - 1][NsP][NtP][NvP] > 1
```

9.2.12.3.2 Presence of occ_direct_node

The syntax element occ_direct_node shall only be present in occupancy tree nodes that are both eligible for direct coding and not prohibited by the rate limit for direct nodes that applies when planar occupancy coding is enabled.

The direct node rate limit mask is specified by the expression *DnPresenceMask*[*i*], *i* ∈ 0 .. 31.

```
DnPresenceMask[i] := occtree_planar_enabled && occtree_direct_coding_mode == 1
    ? dnRate × i % 32 + (dnRate ≥ 32)
    : 1
  where
    dnRate := occtree_direct_node_rate_minus1 + 1
```

The expression *DirectNodePresent* specifies the presence of the syntax element.

```
DirectNodePresent := DirectModeEligible && DnPresenceMask[(Dpth + DnEligibleCnt) % 32]
```

9.2.12.3.3 State variables

Eligibility is specified in terms of the following state variables; the indexes *dpth*, *ns*, *nt* and *nv* identify a node with location (*ns*, *nt*, *nv*) in the tree level at depth *dpth*:

— The sparse array *OccNeighPatEq0*; *OccNeighPatEq0*[*dpth*][*ns*][*nt*][*nv*] identifies whether the identified node has no nodes present in its occupied neighbourhood pattern.

— The sparse array *OccNodeChildCnt*; *OccNodeChildCnt*[*dpth*][*ns*][*nt*][*nv*] is the number of child nodes of the identified node.

— The variable *DnEligibleCnt*, a cumulative count of eligible nodes in a tree level.

9.2.12.3.4 Initial state

At the start of every occupancy_tree syntax structure, the *OccNodeChildCnt* array shall be cleared; all elements of *OccNodeChildCnt* are unset.

9.2.12.3.5 State update at the start of every occupancy tree level

At the start of every occupancy_tree_level syntax structure, the count of eligible nodes *DnEligibleCnt* shall be set to zero.

9.2.12.3.6 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.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
if (DirectModeEligible)
  DnEligibleCnt++
```

9.2.12.4 Points represented by direct nodes

9.2.12.4.1 General

The unscaled positions for the points coded by the direct node are specified by the expression *DnPtPos*[*dnPt*][*k*]. They are the concatenation of:

— the (quantized) node position,

— any bit corresponding to an occupied plane as determined by planar occupancy coding,

— any bits from joint direct position coding, and

— any remaining bits.

When geometry subtree scaling is enabled, direct nodes code partially quantized positions relative to the quantized node position.

```
DnPtPos[dnPt][k] := Nloc[k] << QuantizedNodeSizeLog2[k] | DnPtPosRem[dnPt][k]
DnPtPosRem[dnPt][k] := DnPlanarPos[k] | DnJointPos[dnPt][k] | DnRemPos[dnPt][k]

DnPtPosS[dnPt] := DnPtPos[dnPt][0]
DnPtPosT[dnPt] := DnPtPos[dnPt][1]
DnPtPosV[dnPt] := DnPtPos[dnPt][2]
```

9.2.12.4.2 Output

At the end of the direct node, the coded points shall be scaled (9.2.14.6) and appended to the output point list:

```
if (occ_direct_node) {
  for (dnPt = 0; dnPt ≤ direct_point_cnt_eq2; dnPt++, PointCnt++)
    for (k = 0; k < 3; k++)
      PointPos[PointCnt][k] = OccPosScaleK(k, DnPtPos[dnPt][k])

  for (i = 0; i < direct_dup_point_cnt; i++, PointCnt++)
    for (k = 0; k < 3; k++)
      PointPos[PointCnt][k] = PointPos[PointCnt - 1][k]
}
```

9.2.12.4.3 Planar-inferred position bits

When an axis is eligible for planar occupancy coding and it has a single occupied plane, the MSB of the position residual for that axis *k* is specified by *DnPlanarPos*[*k*], equal to the occupied plane location.

```
DnPlanarPos[k] := ¬PlanarFreeAxis[k] ? occ_plane_pos[k] << DnBitsAfterPlanar[k] : 0
```

The number of bits coded by each position residual exclusive of any bit derived from planar occupancy coding is specified for each component by the expression *DnBitsAfterPlanar*[*k*].

```
DnBitsAfterPlanar[k] := QuantizedNodeSizeLog2[k] - ¬PlanarFreeAxis[k]
```

9.2.12.4.4 Joint coded position bits

The position residuals shall be jointly coded for an axis when the direct node codes two positions, joint coding is enabled and it has a residual bit to code. When angular coding is enabled, components coded by direct_rem_st_ang and direct_rem_v_ang shall not be jointly coded.

```
DnJointCoded[k] := occtree_direct_joint_coding_enabled && direct_point_cnt_eq2
  && DnBitsAfterPlanar[k] > 0
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
  && (¬geom_angular_enabled || k == (1 ^ AzimuthAxis))
```

The joint-coded bits for the two positions are specified by the expression *DnJointPos*[*dnPt*][*k*]. For each component *k*, they comprise the common MSB (9.2.12.5.2) and the first divergent bit (9.2.12.5.3), if any. Joint coding is exclusive of any planar-inferred position bit.

```
DnJointPos[dnPt][k] :=
  DnJointCoded[k] ? DnJointPosCommon[k] | DnJointPosDiffBit[dnPt][k] : 0
```

9.2.12.4.5 Remaining bits

The number of bits coded by each position residual exclusive of any bits derived from planar occupancy coding or joint coding is specified for each component by the expression *DnRemBits*[*k*].

```
DnRemBits[k] := DnBitsAfterPlanar[k] − DnJointCommonBits[k] − DnJointDiffBits[k]
```

The expression *DnRemPos*[*dnPt*][*k*] specifies the position bits coded by the syntax elements direct_rem, direct_rem_st_ang and direct_rem_v_ang.

```
DnRemPos[dnPt][k] :=
  geom_angular_enabled && k == AzimuthAxis ? direct_rem_st_ang[dnPt] :
  geom_angular_enabled && k == 2           ? direct_rem_v_ang[dnPt]  :
                                           : direct_rem[dnPt][k]
```

9.2.12.5 Joint coding of point positions

9.2.12.5.1 Point order

The coded order of two jointly coded points shall satisfy the constraint *DnPtPosConstraint* equal to 1:

— The *s*-coordinate of the first point shall be less than or equal to that of the second point.

— If the *s*-coordinates are equal, the *t*-coordinate of the first point shall be less than or equal to that of the second point.

— If both the *s*- and *t*-coordinates are respectively equal, the *v*-coordinate of the first point shall be less than or equal to that of the second point

```
DnPtPosConstraint :=
    DnPtPosS[0] < DnPtPosS[1]
    || DnPtPosT[0] < DnPtPosT[1] && dnPtPosSameS
    || DnPtPosV[0] ≤ DnPtPosV[1] && dnPtPosSameS && dnPtPosSameT
  where
    dnPtPosSameS := DnPtPosS[0] == DnPtPosS[1]
    dnPtPosSameT := DnPtPosT[0] == DnPtPosT[1]
```

9.2.12.5.2 Common position bits

The number of position bits coded by direct_joint_prefix[*k*] is specified by *DnJointPrefixBits*[*k*].

```
DnJointPrefixBits[k] := DnJointCoded[k] ? IntLog2(direct_joint_prefix[k]) : 0
```

The number of bits coded by the *k*-th component of each position residual, exclusive of any bits derived from planar occupancy coding or a joint-coded prefix, is specified by the expression *DnBitsAfterJointPrefix*[*k*].

```
DnBitsAfterJointPrefix[k] := DnBitsAfterPlanar[k] − DnJointCommonBits[k]
```

The expression *DnJointPosPrefix*[*k*] specifies the value of the position bits coded by direct_joint_prefix[*k*].

```
DnJointPosPrefix[k] :=
  (direct_joint_prefix[k] ^ Exp2(DnJointPrefixBits[k])) << DnBitsAfterJointPrefix[k]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.12.5.3 First divergent position bit

The existence of a divergent bit in the k-th component of the two jointly coded positions is specified by the expression *DnJointPosDiffBits*[k]. A jointly coded divergent position bit exists if direct_joint_prefix[k] is present and does not complete the coding of the k-th position component.

```
DnJointDiffBits[k] := DnJointCoded[k] ? DnBitsAfterPlanar[k] > DnJointPrefixBits[k] : 0
```

The value of the jointly coded divergent bit is specified for the k-th component of each point by the expression *DnJointPosDiffBit*[*dnPt*][k].

```
DnJointPosDiffBit[dnPt][k] := bit << DnRemBits[k]
  where bit :=
    DnJointDiffBitPresent[k]  ? direct_joint_diff_bit[k] ^ dnPt :
    DnJointDiffBitInferred[k] ? dnPt
                              : 0 /* DnJointDiffBits[k] is 0 */
```

9.2.12.5.4 Presence of the syntax element direct_joint_diff_bit

The presence of direct_joint_diff_bit[k] is specified by *DnJointDiffBitPresent*[k].

```
DnJointDiffBitPresent[k] := DnJointDiffBits[k] > 0 && ¬DnJointDiffBitInferred[k]
```

It is present when a divergent bit position exists in the k-th component of the two jointly coded positions and the bit value cannot be inferred from the ordering constraint on jointly coded positions. The inference condition is specified by the expression *DnJointDiffBitInferred*[k].

```
DnJointDiffBitInferred[k] :=
     k == 0 && DnJointDiffBits[0] == 1
  || k == 1 && DnJointDiffBits[1] == 1 && DnJointDiffBits[0] == 0
  || k == 2 && DnJointDiffBits[2] == 1 && DnJointDiffBits[0] + DnJointDiffBits[1] == 0
```

9.2.12.6 Parsing of direct_joint_prefix

The binarization of direct_joint_prefix[k] interleaves the prefix and suffix bins of the binarized exp-Golomb code (11.4.3). Every non-zero prefix bin shall be followed by a single suffix bin.

The binarization shall have no more than *DnRemBits*[k] suffix bins.

```
maxVal = Exp2(DnRemBits[k])
```

The syntax element is parsed as follows; the variable *BinIdx* is the count of coded bins used for contextualization.

```
value = 1
for (BinIdx = 0; value < maxVal && AeReadBin() == 1; BinIdx++)
  BinIdx++
  value = (value << 1) + AeReadBin()
```

9.2.13 Angular coding

9.2.13.1 General

This subclause applies when geom_angular_enabled is 1.

Angular coding in the occupancy tree specifies the contextualization for the planar occupied plane location occ_plane_pos and the direct position remainders direct_rem_st_ang and direct_rem_v_ang. The contextualization uses rays cast from the angular origin.

9.2.13.2 Node position relative to the angular origin

The angular-origin-relative position of the occupancy tree node position p_{min} is specified by the expression *NposAng*[k].

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
NposAng[k] := AngPosScaleK(k, Nloc[k] << QuantizedNodeSizeLog2[k]) - AngularOrigin[k]
NposAngS := NposAng[0]
NposAngT := NposAng[1]
NposAngV := NposAng[2]
```

9.2.13.3 Azimuth coded axis

Contextualization using a beam's azimuth applies to either the S or T axis. The index of the contextualized axis is specified by the expression *AzimuthAxis*. It is determined using the angular-origin-relative node position.

```
AzimuthAxis := Abs(NposAngS) > Abs(NposAngT)
AzimuthAxisIsS := AzimuthAxis == 0
AzimuthAxisIsT := AzimuthAxis == 1
```

9.2.13.4 State variables

Contextualization of occ_plane_pos[*AzimuthAxis*] and direct_rem_st_ang is specified in terms of the following state variables; the index *beamIdx* identifies an SPS enumerated beam:

— The array *BeamPrevPhiValid*; *BeamPrevPhiValid*[*beamIdx*] indicates whether a value has been recorded by *BeamPrevPhi*[*beamIdx*].

— The array *BeamPrevPhi*; *BeamPrevPhi*[*beamIdx*] records the azimuth of the identified beam computed from the most recently coded syntax element direct_rem_st_ang.

9.2.13.5 Initial state

At the start of every occupancy tree, all elements of *BeamPrevPhiValid* shall have the value 0.

9.2.13.6 Closest beam to a point

This subclause specifies the selection of the beam that can emit the closest rays to an angular-origin-relative point (*as*, *at*, *av*) by the expression *BeamIdxEst*[*as*][*at*][*av*]. The selection shall use rays cast from the angular origin without application of vertical beam displacements.

```
BeamIdxEst[as][at][av] := num_beams_minus1 ? BeamIdxFromGrad[aGrad] : 0
```

The selection shall be performed by comparing the gradient of a ray specified by the expression *aGrad* to the elevation gradients of the enumerated beams. The ray is cast from the angular origin and passes through the point (*as*, *at*, *av*).

Unless performing beam selection for planar occupancy coding (9.2.13.7.3), the expression *aGrad* shall be defined as:

```
aGrad := av × IntRecipSqrt(rs × rs + rt × rt) >> 14
  where
    rs := as << 8
    rt := at << 8
```

When performing beam selection for planar occupancy coding (9.2.13.7.3), the expression *aGrad* shall be defined as:

```
aGrad := (2 × av − 1) × IntRecipSqrt(rs × rs + rt × rt) >> 15
  where
    rs := (as << 8) − 128
    rt := (at << 8) − 128
```

The beam search is specified by the expression *BeamIdxFromGrad*. If the gradient *rayGrad* is half-way between that of two beams, the beam with the lower index shall be chosen.

```
BeamIdxFromGrad[rayGrad] :=
  for (i = 1; i < num_beams_minus1; i++)
    if (BeamElev[i] > rayGrad)
      break
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
  if (rayGrad - BeamElev[i - 1] ≤ BeamElev[i] - rayGrad)
    i--
  BeamIdxFromGrad = i
```

9.2.13.7 Application to occupied plane location coding

9.2.13.7.1 Node dimensions and midpoint

The expression *ScaledNodeSize*[*k*] specifies the scaled volume dimensions of the coded node.

```
ScaledNodeSize[k] := AngPosScaleK(k, Exp2(QuantizedNodeSizeLog2[k]))

ScaledNodeSizeS := ScaledNodeSize[0]
ScaledNodeSizeT := ScaledNodeSize[1]
ScaledNodeSizeV := ScaledNodeSize[2]
```

The expression *ScaledHalfNodeSize*[*k*] specifies the midpoint within the scaled volume of the coded node.

```
ScaledHalfNodeSize[k] := AngPosScaleK(k, Exp2(QuantizedNodeSizeLog2[k]) >> 1)

ScaledHalfNodeSizeS := ScaledHalfNodeSize[0]
ScaledHalfNodeSizeT := ScaledHalfNodeSize[1]
ScaledHalfNodeSizeV := ScaledHalfNodeSize[2]
```

NOTE When geometry scaling is disabled, *ScaledNodeSize*[*k*] and *ScaledHalfNodeSize*[*k*] are equal to Exp2(*NodeSizeLog2*[*k*]) and Exp2(*NodeSizeLog2*[*k*] >> 1) respectively.

The midpoint coordinates of the coded node relative to the angular origin are specified by the expression *NposAngMid*[*k*].

```
NposAngMid[k] := NposAng[k] + ScaledHalfNodeSize[k]

NposAngMidS := NposAngMid[0]
NposAngMidT := NposAngMid[1]
NposAngMidV := NposAngMid[2]
```

9.2.13.7.2 Eligibility

Only certain nodes are eligible for angular contextualization of the occupied plane location. They are specified by the expression *AngularEligible*. Eligibility prevents the use of angular contextualization when a node is sufficiently large to be intersected by rays from multiple beams.

```
AngularEligible := geom_angular_enabled
  && (¬num_beams_minus1 || CathetusV > (ScaledHalfNodeSizeV << 26))
```

Eligibility is specified in terms of:

— the smallest difference in elevation gradient between any two beams, *BeamMinDeltaGrad*;

— the V-axis distance subtended (*CathetusV*) by a ray with elevation gradient *BeamMinDeltaGrad*, over the ℓ^1 distance in the S-T plane from the angular origin to the node midpoint; and

— the *v*-component of the scaled node size.

An example of an eligible node is illustrated in Figure 14.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

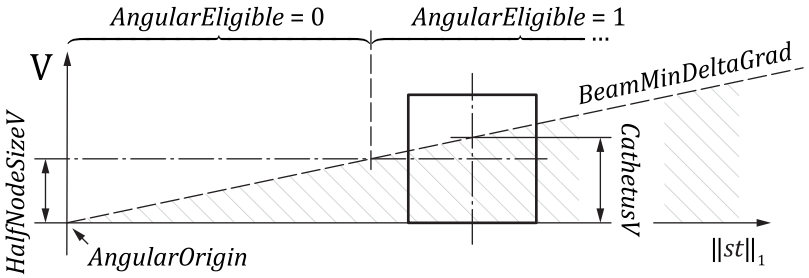


Figure 14 — Illustration of an eligible node

```
CathetusV := BeamMinDeltaGrad × ((rs + rt) >> 1)
  where
    rs := Abs((NposAngMidS << 8) - 128)
    rt := Abs((NposAngMidT << 8) - 128)
```

The expression *BeamMinDeltaGrad* is the smallest difference in elevation gradient between any two beams.

```
BeamMinDeltaGrad :=
  BeamMinDeltaGrad = BeamElev[0]
  for (i = 1; i ≤ num_beams_minus1; i++) {
    delta = BeamElev[i] - BeamElev[i - 1]
    if (delta < BeamMinDeltaGrad)
      BeamMinDeltaGrad = delta
  }
```

9.2.13.7.3 Beam selection

Every node that is eligible for angular contextualization (*AngularEligible* == 1) shall select a beam for that purpose. The selected beam is specified by *PlanarBeamIdx*. It is either the same beam as used for the parent node or the closest beam to the node midpoint (9.2.13.6).

```
PlanarBeamIdx :=
  inheritBeam ? NodeBeamIdx[Dpth - 1][NsP][NtP][NvP]
              : BeamIdxEst[NposAngMidS][NposAngMidT][NposAngMidV]
```

The beam selection is inherited from the parent node when both the parent node has a valid beam and the node is not too small, as specified by the expression *inheritBeam*.

```
inheritBeam :=
  NodeBeamValid[Dpth - 1][NsP][NtP][NvP] && CathetusV > (ScaledHalfNodeSizeV << 28)
```

9.2.13.7.4 State variables

Beam selection is specified in terms of the following state variables; the indexes *dpth*, *ns*, *nt* and *nv* identify a node with location (*ns*, *nt*, *nv*) in the tree level at depth *dpth*:

— The sparse array *NodeBeamValid*; *NodeBeamValid*[*dpth*][*ns*][*nt*][*nv*] indicates whether (when 1) the identified node has a valid beam selection. Unset elements are inferred to be 0.

— The sparse array *NodeBeamIdx*; *NodeBeamIdx*[*dpth*][*ns*][*nt*][*nv*] is the selected beam, if valid, for the identified node.

9.2.13.7.5 Initial state

At the start of every occupancy tree, elements of *NodeBeamValid* shall be unset.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.13.7.6 State update

After selecting a beam (9.2.13.7.3), the beam index and its validity shall be recorded for inheritance by child nodes.

```
NodeBeamIdx[Dpth][Ns][Nt][Nv] = PlanarBeamIdx
NodeBeamValid[Dpth][Ns][Nt][Nv] = 1
```

9.2.13.7.7 Contextualization of occ_plane_pos[*AzimuthAxis*]

The syntax element occ_plane_pos[*AzimuthAzis*] shall be contextualized according to 9.2.13.9 with the argument *phiRangeMulLog2* equal to 2 and the arguments *beamIdx*, *phiPosS*, *phiPosT*, *phiIntvlMidS* and *phiIntvlMidT* as specified in this subclause.

The argument *beamIdx* is the index of the selected beam.

```
beamIdx = PlanarBeamIdx
```

The arguments *phiPosS* and *phiPosT* are the angular-origin-relative coordinates of the lower corner of the occupancy tree node in the S-T plane.

```
phiPosS = NposAngS
phiPosT = NposAngT
```

The arguments *phiIntvlMidS* and *phiIntvlMidT* are the angular-origin-relative coordinates of the scaled node midpoint in the S-T plane.

```
phiIntvlMidS = NposAngMidS
phiIntvlMidT = NposAngMidT
```

9.2.13.7.8 Contextualization of occ_plane_pos[2]

The syntax element occ_plane_pos[2] shall be contextualized according to 9.2.13.10 with the argument *elvIntvlGradDivLog2* equal to 2 and the arguments *beamIdx*, *elvIntvlPosS*, *elvIntvlPosT*, *elvIntvlMidV* and *elvIntvlLenV* as specified in this subclause.

The argument *beamIdx* is the index of the selected beam.

```
beamIdx = PlanarBeamIdx
```

The arguments *elvIntvlPosS*, *elvIntvlPosT* and *elvIntvlMidV* are the angular-origin-relative coordinates of the scaled node midpoint. The argument *elvIntvlLenV* is the length of the V-axis interval represented by the syntax element.

```
elvIntvlPosS = NposAngMidS
elvIntvlPosT = NposAngMidT
elvIntvlMidV = NposAngMidV
elvIntvlLenV = ScaledNodeSizeV
```

9.2.13.8 Application to direct node position coding

9.2.13.8.1 Beam selection

Context selection for direct_rem_st_ang[*dnPt*] and direct_rem_v_ang[*dnPt*] shall be performed using the beam specified by the expression *DnBeamIdx*[*dnPt*].

It is a requirement of bitstream conformance that *DnBeamIdx*[*dnPt*] shall be in the range 0 .. num_beams_minus1.

```
DnBeamIdx[dnPt] := DnBeamIdxEst + beamIdxResid
  where
    beamIdxResid := (1 − 2 × beam_idx_resid_sign[dnPt]) × beam_idx_resid_abs[dnPt]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Every direct node shall determine an estimated beam index as specified by the expression *DnBeamIdxEst*. The beam is estimated from the angular-origin-relative, scaled partial point position specified by *dnBeamPosAng*[*k*].

```
DnBeamIdxEst := BeamIdxEst[beamPosAngS][beamPosAngT][beamPosAngV]
  where
    beamPosAngS := dnBeamPosAng[0]
    beamPosAngT := dnBeamPosAng[1]
    beamPosAngV := dnBeamPosAng[2]
dnBeamPosAng[k] := NposAng[k] + AngPosScaleK(k, DnPlanarPos[k] + dnPlanarPosHalf)
  where
    dnPlanarPosHalf := Exp2(DnBitsAfterPlanar[k]) >> 1
```

9.2.13.8.2 Scaled angular-origin-relative point positions and partial point positions

Contextualization of direct_rem_st_ang and direct_rem_v_ang depends upon approximately scaled partially coded positions. A partially coded position is specified in terms of the MSBs of the complete coded position.

Approximate scaling accumulates applications of geometry scaling to each bit in the partially coded position.

The expression *DnPtPosAng*[*dnPt*][*k*] is the *k*-th component of the scaled, coded point position relative to the angular origin.

```
DnPtPosAng[dnPt][k] := AngPosScaleK(k, DnPtPos[dnPt][k]) - AngularOrigin[k]

DnPtPosAngS[dnPt] := DnPtPosAng[dnPt][0]
DnPtPosAngT[dnPt] := DnPtPosAng[dnPt][1]
```

The expression *DnPartialPosAng*[*dnPt*][*remBits*][*k*] is the *k*-th component of the approximately scaled partially coded position for the *dnPt*-th coded point of the direct node. It excludes the *remBits* LSBs of *DnPtPos*[*dnPt*][*k*].

```
DnPartialPosAng[dnPt][remBits][k] :=
  DnPartialPosAng = NposAng[k]
  for (i = remBits; i < QuantizedNodeSizeLog2[k]; i++)
    DnPartialPosAng += AngPosScaleK(k, DnPtPosRem[dnPt][k] & Exp2(i))

DnPartialPosAngS[dnPt][remBits] := DnPartialPosAng[dnPt][remBits][0]
DnPartialPosAngT[dnPt][remBits] := DnPartialPosAng[dnPt][remBits][1]
DnPartialPosAngV[dnPt][remBits] := DnPartialPosAng[dnPt][remBits][2]
```

9.2.13.8.3 Binarization of direct_rem_st_ang and direct_rem_v_ang

The direct_rem_st_ang and direct_rem_v_ang syntax elements shall be entropy coded using fixed-length binarization. The length in bins shall be equal to the log2 quantized node size less any bit inferred by the presence of a single occupied plane.

The expression *DnRemAngBitsST* is the length in bins of the syntax element direct_rem_st_ang.

```
DnRemAngBitsST := DnBitsAfterPlanar[AzimuthAxis]
```

The expression *DnRemAngBitsV* is the length in bins of the syntax element direct_rem_v_ang.

```
DnRemAngBitsV := DnBitsAfterPlanar[2]
```

9.2.13.8.4 Contextualization of direct_rem_st_ang

Each bin, identified by *BinIdx*, of direct_rem_st_ang[*dnPt*] shall be contextualized according to 9.2.13.9 with the argument *phiRangeMulLog2* equal to 1 and the arguments *beamIdx*, *phiPosS*, *phiPosT*, *phiIntvlMidS* and *phiIntvlMidT* as specified in this subclause.

The argument *beamIdx* is the index of the selected beam.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
beamIdx = DnBeamIdx[dnPt]
```

The arguments *phiPosS* and *phiPosT* are the angular-origin-relative coordinates of the lower end of the interval coded by the bin in the S-T plane.

```
phiPartialSTBits := DnRemAngBitsST - BinIdx

phiPosS = AzimuthAxisIsS ? DnPartialPosAngS[dnPt][phiPartialSTBits] : DnPtPosAngS[dnPt]
phiPosT = AzimuthAxisIsT ? DnPartialPosAngT[dnPt][phiPartialSTBits] : DnPtPosAngT[dnPt]
```

The arguments *phiIntvlMidS* and *phiIntvlMidT* are the angular-origin-relative coordinates of the interval midpoint in the S-T plane. The expression *phiHalfIntvlLen* is the half-interval range.

```
phiHalfIntvlLen := AngPosScaleK(AzimuthAxis, Exp2(DnRemAngBitsST) >> 1 + BinIdx)

phiIntvlMidS = phiPosS + (AzimuthAxisIsS ? phiHalfIntvlLen : 0)
phiIntvlMidT = phiPosT + (AzimuthAxisIsT ? phiHalfIntvlLen : 0)
```

9.2.13.8.5 State update after each direct_rem_st_ang syntax element

After each coded direct_rem_st_ang[*dnPt*] syntax element, the azimuth of the coded point shall be recorded as the azimuth of the selected beam.

```
elvIntvlPosS = AzimuthAxisIsS ? DnPartialPosAngS[dnPt][0] : DnPtPosAngS[dnPt]
elvIntvlPosT = AzimuthAxisIsT ? DnPartialPosAngT[dnPt][0] : DnPtPosAngT[dnPt]

BeamPrevPhi[DnBeamIdx[dnPt]] = IntAtan2(elvIntvlPosT, elvIntvlPosS)
BeamPrevPhiValid[DnBeamIdx[dnPt]] = 1
```

9.2.13.8.6 Contextualization of direct_rem_v_ang

Each bin, identified by *BinIdx*, of direct_rem_v_ang[*dnPt*] shall be contextualized according to 9.2.13.10 with the arguments *beamIdx*, *elvIntvlPosS*, *elvIntvlPosT*, *elvIntvlLenV*, *elvIntvlMidV* and *elvIntvlGradDivLog2* as specified in this subclause.

The argument *beamIdx* is the index of the selected beam.

```
beamIdx = DnBeamIdx[dnPt]
```

The arguments *elvIntvlPosS* and *elvIntvlPosT* are the angular-origin-relative coordinates in the S-T plane of the approximately scaled position of the coded point. The argument *elvIntvlLenV* is the length of the interval represented by the syntax element before division by Exp2(*elvIntvlGradDivLog2*).

```
elvIntvlPosS = AzimuthAxisIsS ? DnPartialPosAngS[dnPt][0] : DnPtPosAngS[dnPt]
elvIntvlPosT = AzimuthAxisIsT ? DnPartialPosAngT[dnPt][0] : DnPtPosAngT[dnPt]
elvIntvlLenV = AngPosScaleV(Exp2(DnRemAngBitsV))
elvIntvlGradDivLog2 = BinIdx
```

NOTE When geometry scaling is disabled or the scaling step size is a power of two, *DnPartialPosAng*[*dnPt*][0][*k*] is equal to *DnPtPosAng*[*dnPt*][*k*].

The argument *elvIntvlMidV* specifies the midpoint of the V-axis interval relative to the angular origin. The expression *elvIntvlHalfLen* is the half-interval range. The start of the V-axis interval, *elvPartialPosV*, is the partially coded and approximately scaled point *v*-coordinate.

```
elvPartialPosV := DnPartialPosAngV[dnPt][DnRemAngBitsV - BinIdx]
elvIntvlHalfLen := AngPosScaleV(Exp2(DnRemAngBitsV) >> 1 + BinIdx)
elvIntvlMidV = elvPartialPosV + elvIntvlHalfLen
```

9.2.13.9 Determination of *CtxIdxAngPhi* for a bin according to beam azimuth

Each contextualized bin represents a half-closed interval along the azimuth coding axis *AzimuthAxis* located along the other S-T plane axis at a distance from the angular origin. The bin values 1 and 0 identify the upper and lower halves of the interval respectively.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Contextualization discriminates by the intersection of a predicted ray with one of eight parameterized azimuth ranges.

The process is parameterized by:

— *beamIdx*, the index of the beam that casts the predicted ray;

— *phiPosS* and *phiPosT*, the angular-origin-relative S-T plane coordinates of a point that defines a nominal azimuth;

— *phiIntvlMidS* and *phiIntvlMidT*, the midpoint of the interval;

— *phiRangeMulLog2*, a syntax element dependent constant used to define the azimuth discrimination ranges.

The azimuth ranges are defined by a nominal azimuth *phiNominal* and the azimuth of the interval midpoint *phiIntvlMid*.

```
phiNominal := IntAtan2(phiPosT, phiPosS)
phiIntvlMid := IntAtan2(phiIntvlMidT, phiIntvlMidS)
```

The predicted ray azimuth is derived from the azimuth of a reference point. The prediction quantizes the nominal azimuth to be an integer number of azimuth steps from the reference point.

The reference azimuth for the predicted ray is that of the previous direct-node-coded point recorded by the selected beam (9.2.13.8.5); if no previous point has been recorded by the selected beam, the reference azimuth is the nominal azimuth.

```
phiRef := BeamPrevPhiValid[beamIdx] ? BeamPrevPhi[beamIdx] : phiNominal
```

The expression *phiPred* specifies the predicted ray azimuth.

```
phiPred := phiRef – beamPhiStep[beamIdx] × phiSteps
  where
    phiSteps := DivExp2Up((phiRef – phiNominal) × BeamPhiStepRecip[beamIdx], 30)
    beamPhiStep := 6588397 / BeamStepsPerRev[beamIdx]
    beamPhiStepRecip := (BeamStepsPerRev[beamIdx] << 30) / 6588397
```

The context selection for the coded bin is specified by the expression *CtxIdxAngPhi*.

```
CtxIdxAngPhi := a + 2 × b + 4 × c
  where
    phiL := phiNominal – phiPred
    phiR := phiIntvlMid – phiPred
    a := (phiL ≥ 0) == (phiR ≥ 0)
    b := Abs(phiL) > Abs(phiR)
    c := Max(Abs(phiL), Abs(phiR)) > (Min(Abs(phiL), Abs(PhiR)) << phiRangeMulLog2)
```

9.2.13.10 Determination of *CtxIdxAngTheta* for a bin according to beam elevation

Each contextualized bin represents a half-closed V-axis interval located at a radial distance in the S-T plane from the angular origin. The bin values 1 and 0 identify the upper and lower halves of the interval respectively.

Contextualization discriminates by the intersection of a predicted ray cast by a beam with one of four parameterized V-axis intervals.

The process is parameterized by:

— *beamIdx*, the index of the beam that casts the predicted ray;

— *elvIntvlPosS* and *elvIntvlPosT*, the coordinates of the V-axis interval in the S-T plane, relative to the angular origin;

— *elvIntvlLenV*, the length of the V-axis interval;

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— *elvIntvlMidV*, the midpoint of the V-axis interval relative to the angular origin;

— *elvIntvlGradDivLog2*, the factor used to derive the sub-interval size.

The expression *rRecip* specifies the reciprocal radial distance of the V-axis interval in the S-T plane from the angular origin.

```
rRecip := IntRecipSqrt(rs × rs + rt × rt)
  where
    rs := (elvIntvlPosS << 8) − 128
    rt := (elvIntvlPosT << 8) − 128
```

The expression *elvIntvlMidGrad* specifies the gradient of a ray cast from the angular origin that intersects the midpoint of the V-axis interval.

```
elvIntvlMidGrad := (2 × elvIntvlMidV − 1) × rRecip / Exp2(15)
```

The expressions *esoTopGrad* and *esoBotGrad* specify the gradients of rays cast from the angular origin that intersect the end points of a V-axis sub-interval.

```
esoIntvlGrad := elvIntvlLenV × rRecip >> 18 + elvIntvlGradDivLog2
esoTopGrad := elvIntvlMidGrad + esoIntvlGrad
esoBotGrad := elvIntvlMidGrad − esoIntvlGrad
```

The expression *elvBeamGrad* specifies the gradient of a ray cast from the angular origin that would intersect the predicted ray at the given radial distance.

```
elvBeamGrad := BeamElev[beamIdx] − BeamOffsetV[beamIdx] × rRecip / Exp2(17)
```

The context selection for the coded bin is specified by the expression *CtxIdxAngTheta*, comparing the gradients of rays cast from the angular origin to those of the threshold points.

```
CtxIdxAngTheta := a + 2 × b
  where
    a := elvBeamGrad ≥ elvIntvlMidGrad
    b := elvBeamGrad ≥ esoTopGrad || elvBeamGrad < esoBotGrad
```

9.2.13.11 Conversion from Cartesian to angular coordinates

This subclause specifies the conversion of decoded points' positions to angular coordinates.

A decoder is not required to perform the conversion unless angular coordinates are required for attribute decoding.

The angular-origin-relative position of every point is specified by the expression *posAng*[*ptIdx*][*k*].

```
posAng[ptIdx][k] := PointPos[ptIdx][k] − AngularOrigin[k]
```

Every angular-origin-relative point position is converted to angular coordinates. The radial distance is the Euclidean distance to the point in the S-T plane; the azimuth angle is the anti-clockwise angle in an S-T plane between the positive S axis and the point; the indexed elevation is the index of the beam that casts the closest rays to the point (9.2.13.6). The conversion is:

```
for (ptIdx = 0; ptIdx < PointCnt; ptIdx++){
  as = posAng[ptIdx][0]
  at = posAng[ptIdx][1]
  av = posAng[ptIdx][2]
  PointAng[ptIdx][0] = IntSqrt(as × as + at × at << 16) >> 8
  PointAng[ptIdx][1] = IntAtan2(at << 8, as << 8) + 3294199 >> 8
  PointAng[ptIdx][2] = BeamIdxEst[as][at][av]
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.14 Subtree scaling

9.2.14.1 Partially quantized coordinates

This subclause applies when geom_scaling_enabled is 1.

Certain subtrees (9.2.14.2.1) can represent point positions using partially quantized coordinates that are parameterized by a scaling volume (9.2.14.2.2) and an applicable QP (9.2.14.5).

The partially quantized representation comprises two concatenated parts:

— an upper part that, when scaled, is the position of the scaling volume's lower corner, $p_{\min}$; scaling is by the scaling volume size rounded down to the next power of two;

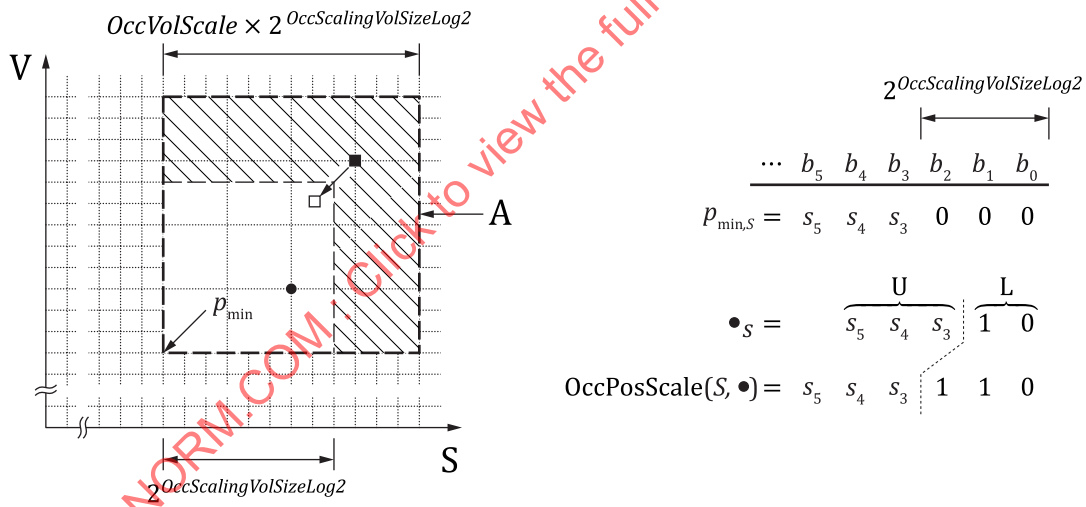
— a lower part that, when scaled, is a position relative to and within the scaling volume; scaling is by the QP derived scale factor.

An example of the representation is illustrated by Figure 15. The coded point • is represented within the scaling volume A. The scaling volume size is 12; rounded down to the next power of two as $8 = 2^{OccScalingVolSizeLog2}$. The QP is 12; the derived scale factor (*OccQpScale*) is 3. The two parts of the point's partially quantized *s*-coordinate are marked U and L:

— The upper part, U, scaled by $2^{OccScalingVolSizeLog2} = 2^3$, is the $p_{\min}$ position of A.

— The lower part, L, scaled by *OccQpScale* = 3 is the $p_{\min}$ relative point position; for the S axis, 2 × 3 = 6. Scaling expands L by 1 bit (*OccQpScaleLog2*) to be 3 bits (*OccScalingVolSizeLog2*) in total.

— If U were 5, then $p_{\min}$ would be $5 \times 2^3 = 40$, and the point's *s*-coordinate would be 46.



Key

A	scaling volume
U, L	upper and lower parts of partially quantized representation
b_i	*i*-th bit

NOTE This figure illustrates the S-V plane in 2D.

Figure 15 — Example of partially quantized coordinates

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.2.14.2 Quantized subtrees

9.2.14.2.1 Identification

Subtrees that code partially quantized coordinates are started by:

— every node at depth *OccQpSubtreeDepth*;

— every direct node at a shallower depth than *OccQpSubtreeDepth*; in this case, the subtree consists only of the subtree root node as a direct node.

The lower parts of partially quantized point coordinates are represented by the location of a leaf node in the subtree or by a direct node that codes a subtree-relative position.

NOTE Identification of a subtree from a direct node at a shallower depth than *OccQpSubtreeDepth* happens after any planar occupancy coding for that node (9.2.14.2.5).

9.2.14.2.2 Subtree scaling volume

A scaling volume is identified by every quantized subtree. Its size depends upon the tree level in which the quantized subtree starts:

— If the quantized subtree starts at a shallower depth than *OccQpSubtreeDepth* and planar occupancy coding is enabled, the scaling volume dimensions are equal to those of the scaled child-node volume of the subtree root node.

— Otherwise, the scaling volume dimensions are identical to those of the scaled subtree root node volume.

The expression *OccScalingVolSizeLog2*[*k*] specifies the integer log2 size of the scaling volume. The scaled dimensions (9.2.14.2.3) are RoundUp(*OccVolScale* × Exp2(*OccScalingVolSizeLog2*[*k*])).

```
OccScalingVolSizeLog2[k] :=
  Dpth ≥ OccQpSubtreeDepth  ? OccLvlNodeSizeLog2[OccQpSubtreeDepth[k] :
  occtree_planar_enabled ? ChildNodeSizeLog2[k] : NodeSizeLog2[k]
```

For example, a quantized subtree root node identifies a volume with dimensions 32×16×32 and an applicable QP of 10 (*OccQpScale* = 2.5). The quantized node size would be 16×8×16 and the scaled node size 40×20×40.

9.2.14.2.3 Scaled occupancy tree node volume dimensions

A scaled node volume is the geometric expansion of the volume associated with a node in the general occupancy tree (9.2.2.1). The expansion is by a QP derived scale factor for the subtree, *OccVolScale*, and is relative to the lower corner, $p_{\min}$, of the subtree scaling volume. Scaling by *OccVolScale* rounds half-values up.

$$OccVolScale := \frac{OccQpScale}{2^{\mathrm{Log2}(OccQpScale)}} = \frac{OccQpScale}{2^{OccQpScaleLog2}} = 1 + \frac{OccQp \bmod 8}{8}$$

The quantized node size *QuantizedNodeSize* is a power-of-two contraction by *OccQpScaleLog2* of the node size for ordinary nodes in the tree level.

```
OccQpScaleLog2 := OccQp / 8
QuantizedNodeSizeLog2[k] := Max(0, NodeSizeLog2[k] - OccQpScaleLog2)
QuantizedChildNodeSizeLog2[k] := Max(0, ChildNodeSizeLog2[k] - OccQpScaleLog2)
```

NOTE The scaled node volume dimensions are equivalent to the rounded expansion of the quantized node size by the subtree scale factor *OccQpScale*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

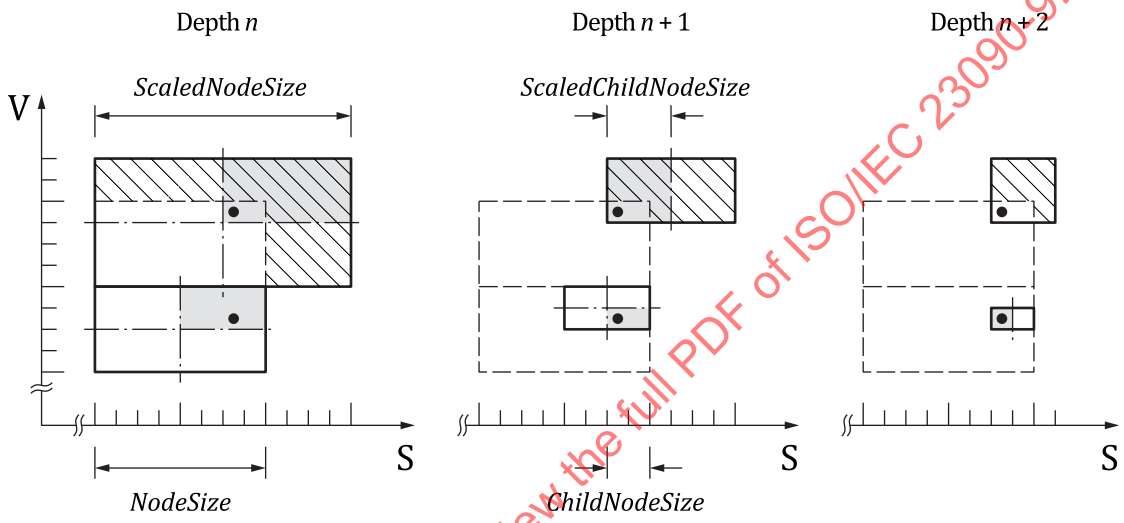
© ISO/IEC 2023 – All rights reserved

9.2.14.2.4 Quantized subtree nodes

A node in a quantized subtree shall identify the presence of at least one point contained within the scaled node volume.

Leaf nodes in the quantized subtree represent indivisible volumes with dimensions equal to the unit cube scaled by the subtree scale factor. Scaled point positions outside the unscaled subtree volume shall be clipped to be within it (hatched area in Figure 15). For example, point ▪ in the figure is clipped to ▫.

Two example subtrees are illustrated over three tree levels by Figure 16. The subtree root node volumes have dashed outlines; the coded nodes in each level have a thick, solid outline. The top subtree uses QP 12, the bottom QP 0. The position of the point • is coded, starting from depth *n*, by the locations of the grey child nodes. The top subtree identifies its leaf node at depth *n* + 1, the bottom subtree at depth *n* + 2.



NOTE QP 12 (top) and QP 0 (bottom).

Figure 16 — Example decomposition of two quantized subtrees

9.2.14.2.5 Direct nodes before *OccQpSubtreeDepth*

This subclause applies when planar occupancy coding is enabled.

Until a node is identified as a direct node through the coding of occ_direct_node, it shall not be considered to start a quantized subtree.

Coding operations that are specified to occur:

— before occ_direct_node shall use the ordinary node volume, equivalent to QP 0;

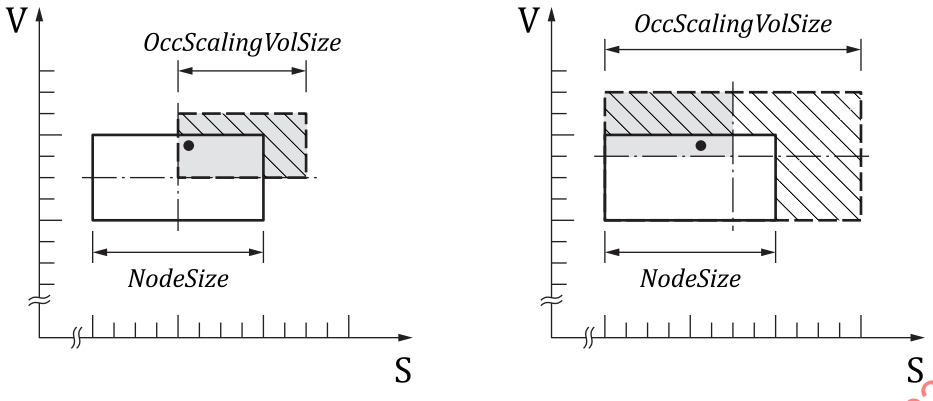
— after occ_direct_node shall use the scaled node volume derived from the direct node QP (*DnQp*),

i.e. the QP and scaled node volume dimensions change during the coding of the node.

This case is illustrated by Figure 17. When planar occupancy coding is enabled (left), the scaled node dimensions are initially equal to *NodeSize*, but changes to *OccScalingVolSize* after occ_direct_node.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved



NOTE Planar occupancy coding enabled (left); planar occupancy coding disabled (right).

Figure 17 — Example subtree scaling volume for subtrees shallower than *OccQpSubtreeDepth*

9.2.14.2.6 Fully quantized subtrees

Sufficiently large subtree QPs can cause a subtree root node to have a log2 quantized node size of 0×0×0. In this case, the subtree root node is a terminal node that:

— is ineligible to be a direct node (9.2.12.3.1)

— has its points represented by a single implicit child leaf node (9.2.6.3); the child node has the same dimensions as the root node – i.e. no axes are coded

9.2.14.3 Syntax element semantics

occ_subtree_qp_offset_present specifies whether (when 1) or not (when 0) per-subtree QP offsets are present in the tree level. QP offsets can only be present in a single tree level. When occ_subtree_qp_offset_present is not present, it shall be inferred to be 0.

occ_subtree_qp_offset_abs[*ns*][*nt*][*nv*] and **occ_subtree_qp_offset_sign**[*ns*][*nt*][*nv*] together specify an offset to the slice geometry QP used to scale point positions. The offset is specified by the expression *OccSubtreeQpOffset*[*ns*][*nt*][*nv*]. The offset applies to all points coded in the subtree of the node located at (*ns*, *nt*, *nv*) in the tree level at depth *OccQpSubtreeDepth*. When occ_subtree_qp_offset_sign[*ns*][*nt*][*nv*] is not present, it shall be inferred to be 0.

```
OccSubtreeQpOffset[ns][nt][nv] :=
  (1 − 2 × occ_subtree_qp_offset_sign[ns][nt][nv]) × occ_subtree_qp_offset_abs[ns][nt][nv]
```

9.2.14.4 State variables

Occupancy tree scaling is specified in terms of the following state variable: The variable *OccQpSubtreeDepth* that identifies the tree level where occ_subtree_qp_offset_present is 1.

9.2.14.5 The subtree QP

9.2.14.5.1 General

The subtree QP is specified by the expression *OccQp*. It is determined for a coded node as:

— when geometry scaling is disabled: 0;

— when the node is a direct node at a depth shallower than *OccQpSubtreeDepth*: specified by 9.2.14.5.2;

— when the node is at a depth equal to or deeper than *OccQpSubtreeDepth*: specified by 9.2.14.5.3.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

NOTE The provisions of 9.2.14.2.5 affect evaluations of *OccQp* during certain nodes.

```
OccQp :=
  ¬geom_scaling_enabled    ? 0 :
  Dpth ≥ OccQpSubtreeDepth ? OccSubtreeQp
  occ_direct_node          ? DnQp : 0
```

9.2.14.5.2 Determination for a direct node before *OccQpSubtreeDepth*

The subtree QP is specified by the expression *DnQp*. It is the slice's direct-node QP limited by the smallest log2 scaling volume dimension.

```
DnQp := Min(dnSliceQp, OccSubtreeQpMax)
  where
    dnSliceQp := geom_qp + occtree_direct_node_qp_offset << geom_qp_mul_log2
```

9.2.14.5.3 Determination for nodes at or after *OccQpSubtreeDepth*

The subtree QP is specified by the expression *OccSubtreeQp*.

```
OccSubtreeQp := sliceQp + (OccSubtreeQpOffset[ss][st][sv] << geom_qp_mul_log2)
  where
    sliceQp := geom_qp + slice_geom_qp_offset << geom_qp_mul_log2
    ss := OccSubtreeQpLoc[0]
    st := OccSubtreeQpLoc[1]
    sv := OccSubtreeQpLoc[2]
```

The expression *OccSubtreeQpLoc* is the subtree root node location in the tree level at depth *OccQpSubtreeDepth* for the node at (*Ns*, *Nt*, *Nv*).

```
OccSubtreeQpLoc[k] := Nloc[k] >> NodeSizeLog2[k] - OccScalingVolSizeLog2[k]
```

9.2.14.5.4 Maximum QP

The maximum QP for a subtree is specified by the expression *OccSubtreeQpMax*.

```
OccSubtreeQpMax := MinVec(OccScalingVolSizeLog2) × 8
```

It is a requirement of bitstream conformance that when occ_subtree_qp_offset_present is 1, *OccSubtreeQp* shall be in the range 0 .. *OccSubtreeQpMax*.

9.2.14.6 Scaling of a position component

This subclause specifies the scaling of the *k*-th partially quantized coordinate by the expressions:

— OccPosScaleK(*k*, *posk*) that clips the scaled position to be within the subtree volume, and

— AngPosScaleK(*k*, *posk*) that does not clip the scaled position.

NOTE When either geom_scaling_enabled or *OccNodeQp* is 0, the expressions OccPosScaleK(*k*, *posk*) and AngPosScaleK(*k*, *posk*) are both equal to *posk*.

The expressions *upperPartQ* and *lowerPartQ* represent the upper and lower parts of the partially quantized coordinate.

```
upperPartQ := posk >> OccScalingVolSizeLog2[k] - OccQpScaleLog2
lowerPartQ := posk & Exp2(OccScalingVolSizeLog2[k] - OccQpScaleLog2) - 1
```

The 3-fractional-bit, fixed-point scale factor used to scale the low part is specified by the expression *sF*.

```
sF := 8 + (OccNodeQp & 7) << OccNodeQp / 8
```

The upper and lower parts are scaled as specified by the expressions *upperPart* and *lowerPart*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
upperPart := upperPartQ << OccScalingVolSizeLog2[k]
lowerPart := DivExp2Up(lowerPartQ × sF, 3)
```

The scaled upper and lower parts are combined to produce the scaled position component:

```
OccPosScaleK(k, posk) := upperPart | Min(lowerPart, Exp2(OccScalingVolSizeLog2[k]) − 1)
AngPosScaleK(k, posk) := upperPart + lowerPart
```

9.3 Predictive tree

9.3.1 General

This subclause specifies the reconstruction of point positions from parsed predictive trees. It applies when geom_tree_type is 1.

The slice geometry can be represented by a sequence of one or more predictive trees of predictive tree nodes. Every tree node specifies a single position for one or more points. Traversal of a predictive tree is in depth-first order.

9.3.2 Syntax element semantics

9.3.2.1 Predictive tree

slice_ptree_qp_period_log2_offset specifies an offset to the GPS-signalled period at which predictive tree QP offsets are signalled. When slice_ptree_qp_period_log2_offset is not present, it shall be inferred to be 0.

A QP offset is signalled once every Exp2(*PtnQpInterval*) nodes.

```
PtnQpInterval := Exp2(ptree_qp_period_log2 + slice_ptree_qp_period_log2_offset)
```

ptn_resid_abs_log2_bits[*k*] specifies the number of bins used for the fixed-length binarization of ptn_resid_abs_log2[*k*].

ptn_radius_min specifies the minimum angular radius coordinate for nodes where ptn_pred_mode is 0.

ptree_end_of_slice specifies whether (when 0) or not (when 1) there is a subsequent predictive tree in the DU.

9.3.2.2 Predictive tree node

Nodes in the predictive trees are numbered and coded according to their position in the depth-first traversal order. The semantics specified in this subclause are for the *nodeIdx*-th coded node.

The syntax structure parameter *depth* is the tree depth of the node.

ptn_qp_offset_abs[*nodeIdx*] and **ptn_qp_offset_sign**[*nodeIdx*] together specify, when present and in accordance with *PtnQpOffset*[*nodeIdx*], an offset to the slice geometry QP used to scale point positions. ptn_qp_offset_sign specifies whether (when 0) the offset's sign is positive or (when 1) negative. When ptn_qp_offset_sign is not present, it shall be inferred to be 0.

```
PtnQpOffset[nodeIdx] :=
  (1 − 2 × ptn_qp_offset_sign[nodeIdx]) × ptn_qp_offset_abs[nodeIdx]
```

The QP for a node is specified by the expression *PtnQp*[*nodeIdx*].

```
PtnQp[nodeIdx] :=
    ¬geom_scaling_enabled   ? 0 :
    nodeIdx % PtnQpInterval ? PtnQp[nodeIdx − 1]
                            : sliceQp + PtnQpOffset[nodeIdx] << geom_qp_mul_log2
  where
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
    sliceQp := geom_qp + slice_geom_qp_offset
```

It is a requirement of bitstream conformance that *PtnQp*[*i*] shall be in the range 0 .. 167 for each node index *i*.

ptn_dup_point_cnt[*nodeIdx*] plus 1 specifies the number of points represented by the node. When ptn_dup_point_cnt[*nodeIdx*] is not present, it shall be inferred to be 0.

ptn_child_cnt_xor1[*nodeIdx*] xor 1 specifies the number of child nodes.

ptn_pred_mode[*nodeIdx*] specifies the method used to predict the node's coded point position.

ptn_phi_mul_abs_prefix[*nodeIdx*], **ptn_phi_mul_abs_minus2**[*nodeIdx*], **ptn_phi_mul_abs_minus9**[*nodeIdx*] and **ptn_phi_mul_sign**[*nodeIdx*] together specify, in accordance with *PtnPhiMul*[*nodeIdx*], an integer number of ptree_ang_azimuth_step_minus1 + 1 steps that form an azimuth prediction residual. ptn_phi_mul_sign specifies whether (when 0) the step change is in the anticlockwise or (when 1) clockwise direction. Any of ptn_phi_mul_sign, ptn_phi_mul_abs_minus2 or ptn_phi_mul_abs_minus9 that are not present shall be inferred to be 0.

```
PtnPhiMul[nodeIdx] := (1 − 2 × ptn_phi_mul_sign[nodeIdx]) × absVal
  where
    absVal := ptn_phi_mul_abs_prefix[nodeIdx]
            + ptn_phi_mul_abs_minus2[nodeIdx] + ptn_phi_mul_abs_minus9[nodeIdx]
```

ptn_resid_abs_gt0[*nodeIdx*][*k*], **ptn_resid_abs_log2**[*nodeIdx*][*k*], **ptn_resid_abs_rem**[*nodeIdx*][*k*] and **ptn_resid_sign**[*nodeIdx*][*k*] together specify, in accordance with *PtnResidual*[*nodeIdx*][*k*], the *k*-th component of the first coordinate-prediction residual. ptn_resid_sign[*nodeIdx*][*k*] specifies whether (when 0) the residual's sign is positive or (when 1) negative. Any of ptn_resid_abs_gt0[*nodeIdx*][*k*], ptn_resid_sign[*nodeIdx*][*k*], ptn_resid_abs_log2[*nodeIdx*][*k*] or ptn_resid_abs_rem[*nodeIdx*][*k*] that are not present shall be inferred to be 0.

```
PtnResidual[nodeIdx][k] := (1 − 2 × ptn_resid_sign[nodeIdx][k]) × absVal[k]
  where
    absVal[k] := ptn_resid_abs_gt0[nodeIdx][k]
               + (Exp2(ptn_resid_abs_log2[nodeIdx][k]) >> 1)
               + ptn_resid_abs_rem[nodeIdx][k]
```

ptn_sec_resid_abs[*nodeIdx*][*k*] and **ptn_sec_resid_sign**[*nodeIdx*][*k*] together specify, in accordance with *PtnSecResidual*[*nodeIdx*][*k*], the *k*-th component of the second coordinate-prediction residual that applies after conversion from angular to Cartesian coordinates. ptn_sec_resid_sign[*nodeIdx*][*k*] specifies whether (when 0) the residual's sign is positive or (when 1) negative. If ptn_sec_resid_sign[*nodeIdx*][*k*] is not present, it shall be inferred to be 0.

```
PtnSecResidual[nodeIdx][k] :=
  (1 − 2 × ptn_sec_resid_sign[nodeIdx][k]) × ptn_sec_resid_abs[nodeIdx][k]
```

9.3.3 Tree traversal for reconstruction of point positions

9.3.3.1 State variables

The reconstruction of point positions from predictive tree nodes is specified in terms of the following state variables:

— The array *PtnStack*, a stack of ancestor node indexes; *PtnStack*[*dpth*] is the node index of the ancestor at depth *dpth* in the predictive tree for the current node.

— The variable *PtnDepth*, indicating the size of the stack of ancestor node indexes and equivalent to the depth of the current node in its predictive tree.

— The variable *PtnIdx*, the node index of the current node in the node traversal order.

— The variable *PtnCnt*, a count of nodes parsed from the bitstream.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.3.3.2 Decoding a sequence of predictive trees

Each predictive tree is decoded according to the recursive application of 9.3.3.3, starting from its root node:

— At the start of each tree, the stack of ancestor node indexes is empty, *PtnDepth* = 0.

— The root node of the first tree has node index *PtnIdx* = 0.

— The index of each subsequent tree's root node is the successor to the index of the last node in the preceding tree.

Predictive trees are decoded while *PtnIdx* is less than *PtnCnt*.

```
for (PtnIdx = 0; PtnIdx < PtnCnt; PtnIdx++) {
  PtnDepth = 0
  … /* decode predictive tree with root node index PtnIdx; see 9.3.3.3 */
}
```

9.3.3.3 Recursive decoding of a subtree

This subclause specifies the reconstruction of a node with index *PtnIdx* and the remainder of its subtree:

— The point positions for the node are reconstructed and appended to the output point lists as specified by 9.3.4.

— The node's index is appended to the stack of ancestor node indexes.

— The subtrees for every child node are reconstructed in turn by the recursive application of this subclause: the index of the first child node is *PtnIdx* + 1; each subsequent child node index is the successor to the last node index of the preceding subtree.

```
PtnStack[PtnDepth++] = PtnIdx /* push */
ptnChildCnt = ptn_child_cnt_xor1[PtnIdx] ^ 1
for (i = 0; i < ptnChildCnt; i++) {
  PtnIdx++
  … /* Recursively reconstruct the i-th child subtree (9.3.3.3) */
}
```

After the node's subtree has been reconstructed, the node is removed from the stack of ancestor node indexes.

```
PtnDepth-- /* pop */
```

9.3.4 Reconstruction of point coordinates

9.3.4.1 General

This subclause specifies the reconstruction of the point positions for a node with index *PtnIdx* at depth *PtnDepth* of a predictive tree.

9.3.4.2 Reconstructed STV coordinates

The node's reconstructed STV coordinates are specified by the expression *PtnPos*[*k*]. They are the sum of a prediction (*predStv*) and a scaled residual (*residStv*), with negative coordinates clipped to 0. The predicted position is:

— when angular geometry coding is disabled: derived from the reconstructed STV coordinates of ancestor nodes (9.3.4.6);

— when angular geometry coding is enabled: a conversion from the node's reconstructed angular coordinates (9.3.4.5).

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
PtnPos[k] := Max(0, predStv[k] + residStv[k])
  where
    predStv[k] := geom_angular_enabled ? PtnAngStv[k] : PtnPred[k]
```

The scaled STV coordinate residuals, specified by *residStv*[*k*], are:

— coded by the node's first residual when angular geometry coding is disabled, and by its second residual when enabled; and

— scaled by the 3-fractional-bit, fixed-point, geometry scale factor specified by the expression *sf* for the node's QP; scaling shall round to the nearest integer with half-values rounded up.

```
residStv[k] := DivExp2Up(resid[k] × sf, 3)
  where
    sf := 8 + (PtnQp[PtnIdx] & 7) << PtnQp[PtnIdx] / 8
    resid[k] := geom_angular_enabled ? PtnSecResidual[PtnIdx][k]
                                     : PtnResidual[PtnIdx][k]
```

9.3.4.3 Reconstructed RPI node coordinates

When angular geometry coding is enabled, the node's reconstructed RPI coordinates are specified by the expression *PtnAng*[*k*]. They are the sum of an angular coordinate prediction (9.3.4.6) and a residual (*residAng*).

It is a requirement of bitstream conformance that *PtnAng*[2] shall be in the range 0 .. num_beams_minus1.

```
PtnAng[k] := PtnPred[k] + residAng[k]
```

The residual RPI coordinates specified by *residAng*[*k*] are the sum of the coded primary residual and a *φ*-component offset, *phiOffset*.

```
residAng[k] := PtnResidual[PtnIdx][k] + (k == 1 ? phiOffset : 0)
  where
    phiOffset := PtnPhiMul[PtnIdx] × (ptree_ang_azimuth_step_minus1 + 1)
```

9.3.4.4 Points represented by a node

The reconstructed STV and, if applicable, RPI point coordinates are appended to the output point lists *PointPos* and *PointAng* for each point represented by the node.

```
for (i = 0; i ≤ ptn_dup_point_cnt[PtnIdx]; i++, PointCnt++)
  for (k = 0; k < 3; k++) {
    PointPos[PointCnt][k] = PtnPos[k]
    if (geom_angular_enabled)
      PointAng[PointCnt][k] = PtnAng[k]
  }
```

9.3.4.5 Predicted STV coordinates for angular coded geometry

When angular geometry coding is enabled, the node's RPI coordinates are converted to Cartesian STV coordinates, as specified by *PtnAngStv*[*k*], for prediction of the coded position.

```
PtnAngStv[k] := AngularOrigin[k] + (
    k == 0 ? DivExp2Fz(ρ × IntCos(φ, ptree_ang_azimuth_pi_bits_minus11 + 11), 24) :
    k == 1 ? DivExp2Fz(ρ × IntSin(φ, ptree_ang_azimuth_pi_bits_minus11 + 11), 24) :
    k == 2 ? DivExp2Fz(DivExp2Fz(BeamElev[i] × ρ, 15) − BeamOffsetV[i], 3)        : na)
  where
    ρ := PtnAng[0] << ptree_ang_radius_scale_log2
    φ := PtnAng[1]
    i := PtnAng[2]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

9.3.4.6 Prediction from ancestor nodes

Node coordinates can be predicted from up to three ancestor nodes. Depending upon whether angular geometry is enabled or disabled, the prediction is for either RPI or STV coordinates. The coordinates of the parent, grandparent and great-grandparent nodes are specified by *ptnP*[*k*], *ptnG*[*k*] and *ptnU*[*k*], respectively.

```
ptnP[k] := PtnRef[1][k]
ptnG[k] := PtnRef[2][k]
ptnU[k] := PtnRef[3][k]

PtnRef[ancestor][k] := geom_angular_enabled ? PointAng[ptIdx][k] : PointPos[ptIdx][k]
  where
    ptIdx := PtnStack[PtnDepth - ancestor]
```

The predicted coordinates, specified by *PtnPred*[*k*], are:

— when the prediction mode is 0 and if angular geometry is:

  — disabled, the origin (0, 0, 0);

  — enabled, a coordinate specified by *predAngMode0*[*k*] whose *ρ*-component is ptn_radius_min, and whose *φ*- and *i*-components are the same as the parent node, or zero if the predicted node is the root node of a predictive tree.

— when the prediction mode is 1, the coordinates of the parent node;

— when the prediction mode is 2, the coordinates of the parent node, translated by the vector from the second to first ancestor;

— when the prediction mode is 3, the coordinates of the parent node, translated by the vector from the third to the second ancestor.

```
PtnPred[k] :=
    predMode == 0 && geom_angular_enabled ? predAngMode0[k] :
    predMode == 0 ? 0 :
    predMode == 1 ? ptnP[k] :
    predMode == 2 ? ptnP[k] + ptnP[k] - ptnG[k] :
    predMode == 3 ? ptnP[k] + ptnG[k] - ptnU[k] : na
  where
    predMode := ptn_pred_mode[PtnIdx]

predAngMode0[k] :=
  k == 0        ? ptn_radius_min :
  PtnDepth > 0 ? ptnP[k]         : 0
```

10 Slice attributes

10.1 General

This clause specifies the reconstruction of a single slice attribute for the coded slice geometry. The reconstructed attribute values are stored in the array *PointAttr*.

10.2 Point coordinates

10.2.1 General

Attribute coding can use either the slice's reconstructed STV point positions or the points' scaled angular coordinates.

The expression *AttrPos*[*ptIdx*][*k*] specifies the coordinates of each point for attribute coding:

— When attr_coord_conv_enabled is 0, *AttrPos* is equivalent to *PointPos*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— Otherwise, *AttrPos*[*ptIdx*][*k*] are angular point coordinates as specified by 10.2.2.

```
AttrPos[ptIdx][k] := attr_coord_conv_enabled
  ? AttrPosAng[ptIdx][k]
  : PointPos[ptIdx][k]
```

10.2.2 Conversion to scaled angular coordinates

The conversion is specified by the expression *AttrPosAng*[*ptIdx*][*k*]. The point's angular coordinates shall be offset by the minimum angular coordinates and scaled. Any negative coordinate after conversion shall be clipped to 0.

```
AttrPosAng[ptIdx][k] := DivExp2Up(relPos × attr_coord_conv_scale[k], 8)
  where
    relPos := Max(0, PointAng[ptIdx][k] - minAng[k])
    minAng[k] := geom_tree_type == 1 && k == 1
      ? -Exp2(ptree_ang_azimuth_pi_bits_minus11 + 10)
      : 0
```

It is a requirement of bitstream conformance that attr_coord_conv_scale shall not cause any converted coordinate, *AttrPosAng*[*ptIdx*][*k*], to be greater than Exp2(*MaxSliceDimLog2*) – 1.

10.3 Syntax element semantics

10.3.1 Attribute data unit coefficients

The array *AttrCoeff*, with elements *AttrCoeff*[*coeffIdx*][*c*], contains transform coefficient values. Elements of the array shall be initialized to zero.

zero_run_length_prefix, **zero_run_length_minus3_div2**, **zero_run_length_minus3_mod2** and **zero_run_length_minus11** together specify, in accordance with the expression *ZeroRunLength*, the number of consecutive transform coefficient tuples with all components equal to zero. Any of zero_run_length_minus3_div2, zero_run_length_minus3_mod2 and zero_run_length_minus11 that are not present shall be inferred to be 0.

```
ZeroRunLength := zero_run_length_prefix
  + 2 × zero_run_length_minus3_div2 + zero_run_length_minus3_mod2
  + zero_run_length_minus11
```

10.3.2 Attribute coefficient tuples

Attribute coefficient values are signalled for a *coeffIdx*-th coefficient tuple when at least one component is not equal to zero.

coeff_abs[*c*] and **coeff_sign**[*c*] together specify the *c*-th transform coefficient component *AttrCoeff*[*coeffIdx*][*c*]. coeff_sign[*c*] specifies whether (when 0) the coefficient's sign is positive or (when 1) negative. If coeff_sign[*c*] is not present, it shall be inferred to be 0.

The coefficients of the *coeffIdx*-th tuple are specified by the derivation of *AttrCoeff*:

```
for (c = 0; c < AttrDim; c++){
  absVal = coeff_abs[c]

  if (c == AttrDim - 1)
    if (AttrDim == 1
     || AttrDim == 2 && coeff_abs[0] == 0
     || AttrDim == 3 && coeff_abs[0] == 0 && coeff_abs[1] == 0)
      absVal++
  AttrCoeff[coeffIdx][(c + 1) % AttrDim] = (1 - 2 × coeff_sign[c]) × absVal
}
```

NOTE When a point is eligible for direct prediction, the LSBs of coeff_abs encode the direct predictor mode.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

10.3.3 Raw attribute values

raw_attr_component_length, when present, specifies the length in bytes of each syntax element raw_attr_value.

raw_attr_value[*ptIdx*][*c*] specifies the attribute value for the *c*-th component of the *ptIdx*-th point in canonical point order. The length in bits of each syntax element is specified by the expression *RawAttrValueBits*.

```
RawAttrValueBits := raw_attr_width_present
  ? 8 × raw_attr_component_length
  : AttrBitDepth
```

10.4 Raw attribute decoding

This subclause applies when attr_coding_type is 3.

Attribute values shall be set equal to the corresponding raw_attr_value syntax elements.

```
for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)
  for (c = 0; c < AttrDim; c++)
    PointAttr[ptIdx][c] = raw_attr_value[ptIdx][c]
```

10.5 Attribute decoding using the region-adaptive hierarchical transform

10.5.1 General

The region-adaptive hierarchical transform specified by 10.5 is a recursive two-point transform. It applies when attr_coding_type is 0.

The transform constructs a spatial tree of 3D transform blocks using the slice geometry (10.5.2). Basis vectors are calculated for each application of the transform, weighted in proportion to the significance of each coefficient. A transform domain prediction process predicts AC coefficients from the DC coefficients of certain adjoining blocks.

The reconstruction process is specified for a single attribute component *Cidx* ∈ 0 .. *AttrDim* − 1. It starts by:

— mapping coded coefficients to the transform tree (10.5.3); and

— scaling the coded coefficients (10.5.4).

Then in turn for each level, starting from the root of the transform tree (*Lvl* = *RahtRootLvl*) and proceeding down the tree until completing level 0:

— performing transform domain prediction of coded coefficients (10.5.5); and

— applying the inverse transform (10.5.6)

The reconstructed attribute values are specified by 10.5.7.

10.5.2 Transform tree

10.5.2.1 General

The tree of transform bocks is defined recursively:

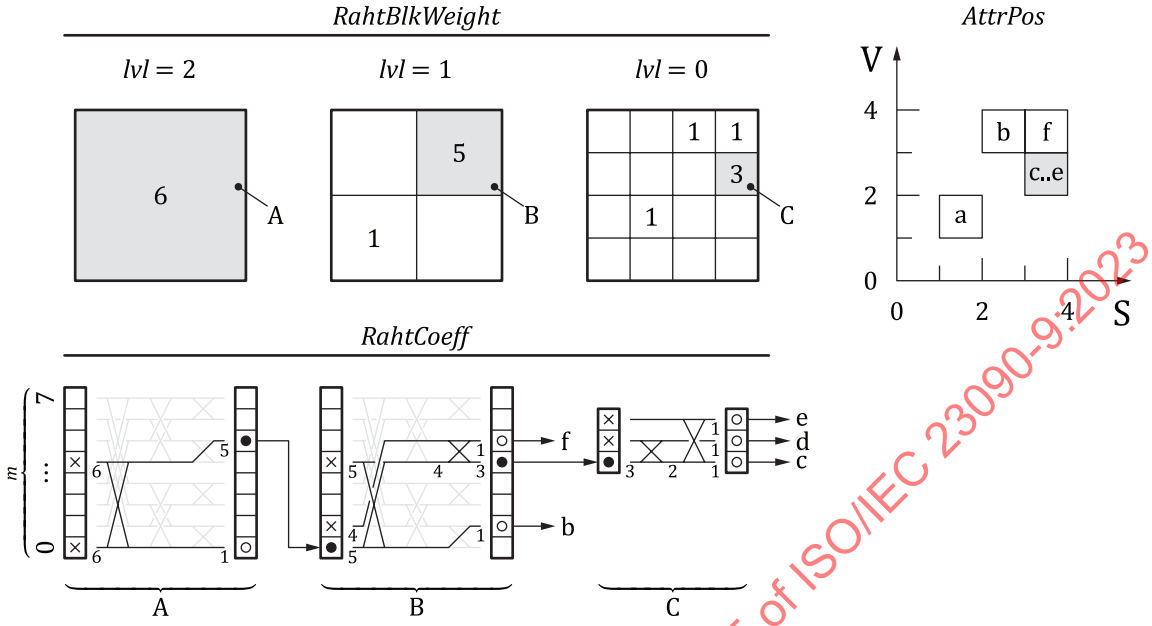
— In tree level 0, each block groups together points with identical attribute coordinates.

— Each subsequent tree level *l* shrinks the preceding level *l* − 1 by a factor of two in each dimension; each 2×2×2 block groups together up to eight blocks from the preceding level.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

An example tree is illustrated in Figure 18. The points a to f are grouped into blocks according to their attribute coordinates: C groups together c, d and e. The weight of each block is the number of points spanned by the block (10.5.2.3): C has a weight of 3; B with a weight of 5 groups together C, b and f; A with a weight of 6 groups together B and a.



Key

a to f	points
A to C	transform blocks
×	transform coefficients at input to inverse transform for labelled block
○	inverse transformed coefficient
•	inherited DC coefficient (See *RahtDcCoeff*)
$_1$ to $_6$	coefficient weights (See *RahtCoeffWeightM*)

Figure 18 — Example RAHT tree, block weights and transform structure

10.5.2.2 State variables

The RAHT tree is specified in terms of the following state variables:

— The sparse array *RahtCoeff* of transform block coefficients; *RahtCoeff*[*lvl*][*bs*][*bt*][*bv*][*i*] is the *i*-th coefficient for the block located at (*bs*, *bt*, *bv*) in transform level *lvl*. Unset elements shall be inferred to be 0.

— The array *RahtBlkLoc* of transform block locations; *RahtBlkLoc*[*lvl*][*nIdx*][*k*] is the location of the *nIdx*-th coded block in transform level *lvl*.

— The array *RahtBlkCnt* of node counts per tree level; *RahtBlkCnt*[*lvl*] is the number of blocks in transform level *lvl*.

— The variable *RahtLvlCnt*, the number of transform levels.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

10.5.2.3 Transform block weight

The weight of the DC transform coefficient for a block located at (*bs*, *bt*, *bv*) in transform level *lvl* is specified by the expression *RahtBlkWeight*[*lvl*][*bs*][*bt*][*bv*]. It is equal to the number of points that the coefficient applies to.

NOTE 1 The sum of all block weights in any transform level is equal to the number of coded points (*PointCnt*).

NOTE 2 A block's weight is equal to the sum of its child block weights.

```
RahtBlkWeight[lvl][bs][bt][bv] :=
    RahtBlkWeight = 0
    for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)
      RahtBlkWeight += isPointInSubtree[ptIdx]
  where
    isPointInSubtree[ptIdx] :=
         bs == AttrPos[ptIdx][0] >> lvl
      && bt == AttrPos[ptIdx][1] >> lvl
      && bv == AttrPos[ptIdx][2] >> lvl
```

10.5.2.4 Number of transform levels and per-level block order

The root node of the transform tree is the lowest block in the tree with a DC coefficient that spans the entire geometry; i.e. it has a weight equal to the number of coded points, *PointCnt*.

The tree level containing the root node is *RahtRootLvl*:

```
RahtRootLvl := RahtLvlCnt - 1
```

Within a transform level, blocks are ordered for coefficient coding by ascending Morton-coded block location, as specified by the derivation of *RahtBlkLoc*. Empty blocks are ignored.

```
for (RahtLvlCnt = 0; !done; RahtLvlCnt++)
  for (mIdx = 0, nIdx = 0, wSum = 0; wSum < PointCnt; mIdx++) {
    (bs, bt, bv) = FromMorton(mIdx)

    wSum += RahtBlkWeight[RahtLvlCnt][bs][bt][bv]
    if (RahtBlkWeight[RahtLvlCnt][bs][bt][bv] == 0)
      continue

    RahtBlkCnt[RahtLvlCnt]++
    RahtBlkLoc[RahtLvlCnt][nIdx][0] = bs
    RahtBlkLoc[RahtLvlCnt][nIdx][1] = bt
    RahtBlkLoc[RahtLvlCnt][nIdx][2] = bv
    nIdx++

    done = RahtBlkWeight[RahtLvlCnt][bs][bt][bv] == PointCnt
  }
```

10.5.2.5 2×2×2 transform block coefficient weights

Transform coefficient weights are specified for each directional stage of the two-point transform for 2×2×2 transform blocks by the expression *RahtCoeffWeightM*[*lvl*][*stage*][*bs*][*bt*][*bv*][*m*]; the parameter(s):

— *bs*, *bt* and *bv* specify a transform block location in tree level *lvl*, *lvl* > 0;

— *m* specifies the transform coefficient index in forward transform stage *stage*.

```
RahtCoeffWeightM[lvl][stage][bs][bt][bv][m] := RahtCoeffWeight[lvl][stage][s][t][v]
  where
    s := 2 × bs + FromMorton[m][0]
    t := 2 × bt + FromMorton[m][1]
    v := 2 × bv + FromMorton[m][2]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Within a block, coefficient weights are determined iteratively starting from its child block weights (stage 0) to the transform block coefficient weights of stage 3. At each transform stage and for each pair of inverse-transformed values *a* and *b*, the weight for the DC (*wL*) and AC (*wH*) coefficient is the sum of the weights for a and b. If the weight for either *a* or *b* is 0, the AC coefficient weight is 0.

The expression *RahtCoeffWeight*[*lvl*][*stage*][*s*][*t*][*v*] specifies the derivation of a weight in transform stage *stage* for the coefficient corresponding to the block located at (*s*, *t*, *v*) in tree level *lvl* − 1.

```
RahtCoeffWeight[lvl][stage][s][t][v] :=
    stage == 0 ? RahtBlkWeight[lvl - 1][s][t][v] :
    stage == 1 ? v % 2 == 0 ? wL[0][0][1] : wH[0][0][-1] :
    stage == 2 ? t % 2 == 0 ? wL[0][1][0] : wH[0][-1][0] :
    stage == 3 ? s % 2 == 0 ? wL[1][0][0] : wH[-1][0][0] : na
  where
    wL[ds][dt][dv] := wSum[ds][dt][dv]
    wH[ds][dt][dv] := wSum[ds][dt][dv] × wHnz[ds][dt][dv]
    wSum[ds][dt][dv] := wP[s][t][v] + wP[s + ds][t + dt][v + dv]
    wHnz[ds][dt][dv] := wP[s][t][v] × wP[s + ds][t + dt][v + dv] > 0
    wP[s][t][v] := RahtCoeffWeight[lvl][stage - 1][s][t][v]
```

NOTE *RahtBlkWeight*[*lvl*][*s*][*t*][*v*] ≡ *RahtCoeffWeight*[*lvl*][3][2 × *s*][2 × *t*][2 × *v*]; *lvl* > 0.

In the example of Figure 18, block B has stage 0 coefficient weights of 1, 3 and 1; stage 1 and 2 weights of 1, 4 and 1; and stage 3 weights of 5, 4 and 5. *RahtCoeffWeight*[1][1][3][0][2] would be 4.

10.5.3 Coefficient order

10.5.3.1 General

This subclause specifies the correspondence between coded transform coefficients and the transform tree.

Starting from the root of the transform tree and proceeding in breadth-first order, coefficients are coded for each transform block; all transform blocks within one tree level are coded before those of the next level. Within a tree level, blocks shall be traversed in ascending Morton order of block location.

The order of coefficients within a transform block is specified by 10.5.3.2 for 2×2×2 blocks (tree levels greater than 0) and 10.5.3.3 for blocks of co-located points (tree level 0).

The mapping from the coded order to the transform tree is specified in terms of the following variables:

— *Lvl*, the index of the mapped transform level;

— *CoeffIdx*, the index into the decoded coefficient array *AttrCoeff* for the next mapped coefficient.

```
CoeffIdx = 0
for (Lvl = RahtLvlCnt; Lvl ≥ 0; Lvl--) {
  if (Lvl > 0) {
    … /* See 10.5.3.2 */
  } else {
    … /* See 10.5.3.3 */
  }
}
```

10.5.3.2 Mapping for a tree level of 2×2×2 transform blocks

This subclause applies to tree levels greater than 0.

For each 2×2×2 transform block, up to 7 AC coefficients are mapped from the bitstream to coefficient indexes within the block. In the case of the root transform block, the DC coefficient is additionally mapped.

Table 28 specifies the order in which transform block coefficients are coded; *RahtCoeffOrder*[*i*] is the block index of the *i*-th coded coefficient.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Only coefficients with a non-zero transform coefficient weight are coded.

Table 28 — 2×2×2 RAHT coefficient coding order

i	0	1	2	3	4	5	6	7
***RahtCoeffOrder*[*i*]**	0	4	2	1	6	5	3	7

```
for (nIdx = 0; nIdx < RahtBlkCnt[Lvl]; nIdx++) {
  bs = RahtBlkLoc[Lvl][nIdx][0]
  bt = RahtBlkLoc[Lvl][nIdx][1]
  bv = RahtBlkLoc[Lvl][nIdx][2]

  for (i = 0; i < 8; i++) {
    /* skip the DC coefficient that will be inherited */
    if (i == 0 && Lvl < RahtRootLvl)
      continue

    if (RahtCoeffWeightM[Lvl][3][bs][bt][bv][RahtCoeffOrder[i]] > 0)
      RahtCoeff[Lvl][bs][bt][bv][RahtCoeffOrder[i]] = AttrCoeff[CoeffIdx++][Cidx]
  }
}
```

10.5.3.3 Mapping for co-located points

This subclause applies to the final tree level (*Lvl* == 0), after all other tree levels have been mapped.

Each transform block with a node weight *w* greater than 1 codes *w* − 1 AC coefficients with the same attribute coordinates.

```
for (nIdx = 0; nIdx < RahtBlkCnt[0]; nIdx++) {
  ns = RahtBlkLoc[0][nIdx][0]
  nt = RahtBlkLoc[0][nIdx][1]
  nv = RahtBlkLoc[0][nIdx][2]

  for (i = 1; i < RahtBlkWeight[0][ns][nt][nv]; i++)
    RahtCoeff[0][ns][nt][nv][i] = AttrCoeff[CoeffIdx++][Cidx]
}
```

10.5.4 Coefficient scaling

10.5.4.1 General

This subclause specifies the scaling of coded coefficients for a block located at (*Bs*, *Bt*, *Bv*) in tree level *Lvl*. It shall be applied to every block in every tree level in any order.

If a regional QP offset is present (i.e. attr_region_cnt > 0), a tree (10.5.4.4) is specified that blends QP offsets along region boundaries according to the structure of the RAHT tree.

10.5.4.2 For a transform block

Within a transform block, coded coefficients shall be scaled according to a per-coefficient QP. The DC coefficient is not scaled except when coded in the root node of the transform tree.

```
mCnt := Lvl > 0 ? 8 : RahtBlkWeight[0][Bs][Bt][Bv]
for (m = 0; m < mCnt; m++) {
  /* skip the DC coefficient that will be inherited */
  if (m == 0 && Lvl < RahtRootLvl)
    continue

  RahtCoeff[Lvl][Bs][Bt][Bv][m] = RahtCoeffScaled[Lvl][Bs][Bt][Bv][m]
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

The scaling of the *m*-th coded coefficient of a transform block located at (*bs*, *bt*, *bv*) in tree level *lvl* is specified by the expression *RahtCoeffScaled*[*lvl*][*bs*][*bt*][*bv*][*m*]: it is scaled by the fixed-point step size *AttrQstep*[*qp*] (10.7.4) and represented as a 15 fractional-bit, fixed-point coefficient value.

```
RahtCoeffScaled[lvl][bs][bt][bv][m] := coeff × AttrQstep[qp] << 7
  where
    coeff := RahtCoeff[lvl][bs][bt][bv][m]
    qp :=  RahtCoeffQp[lvl][bs][bt][bv][m]
```

10.5.4.3 Per coefficient QP

The expression *RahtCoeffQp*[*lvl*][*bs*][*bt*][*bv*][*m*] specifies the QP for the *m*-th coefficient of the transform block located at (*bs*, *bt*, *bv*) in tree level *lvl* for the *Cidx*-th attribute component:

— *rgnOffset*[*qc*] is the per-coefficient offset from the region-dependent QP offset tree.

— *dpth* is the depth of the transform block in the RAHT tree.

```
RahtCoeffQp[lvl][bs][bt][bv][m] := AttrQp[dpth][rgnOffset][Cidx > 0]
  where
    dpth := RahtRootLvl – lvl
    rgnOffset[qc] := RahtTreeQpOffsetM[lvl][bs][bt][bv][m][qc]
```

10.5.4.4 Region-dependent QP offset tree

The integer, averaged region-dependent QP offset for each coefficient of a 2×2×2 transform block is specified by the expression *RahtTreeQpOffsetM*[*lvl*][*bs*][*bt*][*bv*][*m*]. The parameter(s):

— *bs*, *bt* and *bv* specify a transform block location in tree level *lvl*;

— *m* specifies the transform coefficient index from the final forward transform stage.

```
RahtTreeQpOffsetM[lvl][bs][bt][bv][m] :=
    lvl == 0 ? RahtTreeQpOffset[  0][3][bs][bt][bv][qc] >> 4
             : RahtTreeQpOffset[lvl][3][ms][mt][mv][qc] >> 4
  where
    ms := 2 × bs + FromMorton[m][0]
    mt := 2 × bt + FromMorton[m][1]
    mv := 2 × bv + FromMorton[m][2]
```

The fixed-point, region-dependent QP offset tree is structurally identical to the transform tree. It is specified recursively for a QP component *qc* by *RahtTreeQpOffset*[*lvl*][*s*][*t*][*v*][*qc*]:

— For tree level 0, the offset is the regional QP offset for a point with attribute coordinates (*s*, *t*, *v*).

— For each subsequent tree level *l*, a 2×2×2 block of QP offsets is averaged for each transform stage in turn. Each QP offset in a block at stage 0 is that of the DC transform block coefficient for a child block in the preceding level *l* – 1.

Within a block, each subsequent transform stage averages, along the transformed axis, adjacent pairs of QP offsets from the preceding stage that have a non-zero transform coefficient weight. For a pair of QPs *a* and *b* with respective weights *wa* and *wb*:

— The weight for the corresponding DC coefficient is *a* + *b* divided by 2 if *wa* and *wb* are non-zero, or 1 otherwise.

— The weight for the corresponding AC coefficient is *b* if *wa* and *wb* are non-zero, or 0 otherwise.

Averages shall be calculated using four fractional bits.

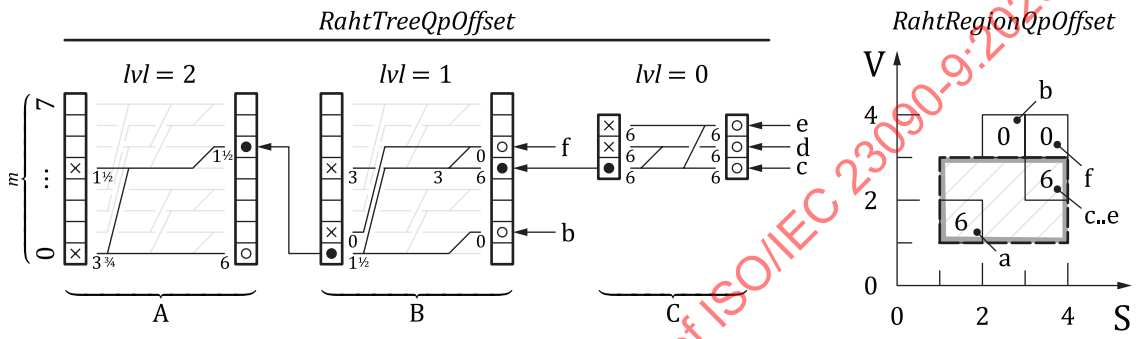
```
RahtTreeQpOffset[lvl][stage][s][t][v][qc] :=
    lvl == 0 ? RahtRegionQpOffset[qs][qt][qv][qc] << 4 :
    stage == 0 ? RahtTreeQpOffset[lvl – 1][3][2 × s][2 × t][2 × v][qc] :
    stage == 1 ? v % 2 == 0 ? qpL[0][0][1] : qpH[0][0][-1] :
    stage == 2 ? t % 2 == 0 ? qpL[0][1][0] : qpH[0][-1][0] :
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
    stage == 3 ? s % 2 == 0 ? qpL[1][0][0] : qpH[-1][0][0] : na
  where
    qpL[ds][dt][dv] := qpSum[ds][dt][dv] >> qpHnz[ds][dt][dv]
    qpH[ds][dt][dv] := qpC[s][t][v] × qpHnz[ds][dt][dv]
    qpSum[ds][dt][dv] := qpC[s][t][v] + qpC[s + ds][t + dt][v + dv]
    qpHnz[ds][dt][dv] :=  wC[s][t][v] ×  wC[s + ds][t + dt][v + dv] > 0
    qpC[s][t][v] := RahtTreeQpOffset[lvl][stage − 1][s][t][v][qc]
    wC[s][t][v]  :=  RahtCoeffWeight[lvl][stage − 1][s][t][v]
```

An example tree is illustrated in Figure 19 for the transform tree of Figure 18. The hatched area has a regional QP offset of +6; co-located points c, d and e have an offset of +6; points b and f, 0. In block A, the stage 3 QPs are used to scale the transform coefficients. For the coefficient at m = 0, 3¾, the QP is the mean of the stage 2 QPs 1½ and 6; for the coefficient at m = 4, it is 1½. Scaling of coefficient uses the integer part of the fractional QP.



Key

b to f	points (See Figure 18)
A to C	transform blocks (See Figure 18)
×	transform coefficients at input to inverse transform for labelled block
○	inverse transformed coefficient
•	inherited DC coefficient (See *RahtDcCoeff*)
$_0$ to $_6$	QP values (See *RahtTreeQpOffset*)

Figure 19 — Example region-dependent QP offset tree

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

10.5.5 Transform domain prediction

10.5.5.1 General

This subclause applies when raht_prediction_enabled is 1. It specifies the transform domain prediction for a block located at (*Bs*, *Bt*, *Bv*) in tree level *Lvl*. It shall be performed for every eligible block (10.5.5.2) in the tree level, in any order, by:

— generating a prediction block (10.5.5.4);

— weighting the values of the prediction block (10.5.5.5) and applying the forward transform (10.5.5.6);

— adding the resulting AC transform coefficients to the coefficient residuals in the coefficient tree.

The prediction block and its transform is specified in terms of the eight-element array *RahtPredBlk*; *RahtPredBlk*[*m*] is the prediction block value for the Morton-coded location *m*.

```
if (RahtPredEligible[Lvl][Bs][Bt][Bv]) {
  for (m = 0; m < 8; m++)
    RahtPredBlk[m] = RahtPredW[m]

  … /* in-place, forward transform of RahtPredBlk (10.5.5.6) */

  for (m = 1; m < 8; m++)
```

© ISO/IEC 2023 – All rights reserved

```
    RahtCoeff[Lvl][Bs][Bt][Bv][m] += RahtPredBlk[m]
}
```

10.5.5.2 Eligibility

When enabled, transform domain prediction shall be performed for 2×2×2 transform blocks unless the block:

— is the root of the transform tree;

— is adjoined (10.5.5.3) by fewer than raht_prediction_samples_min non-empty blocks; or

— has an ancestor, except the root block, that is adjoined by fewer than raht_prediction_subtree_min non-empty blocks.

The expression *RahtPredEligible*[*lvl*][*bs*][*bt*][*bv*] specifies whether the transform block located at (*bs*, *bt*, *bv*) in tree level *lvl* is eligible.

```
RahtPredEligible[lvl][bs][bt][bv] := raht_prediction_enabled
    && lvl > 0
    && lvl < RahtRootLvl
    && RahtNeighCnt[lvl][bs][bt][bv] ≥ raht_prediction_samples_min
    && RahtNeighCntMinAncestor[lvl][bs][bt][bv] ≥ raht_prediction_subtree_min
```

10.5.5.3 Adjoining blocks

A prediction block is generated from up to 19 transform blocks that contain a DC coefficient: the co-located block and those that adjoin the predicted block by a face or an edge.

The expression *RahtNeighCnt*[*lvl*][*bs*][*bt*][*bv*] is the number of non-empty blocks that can be used to predict the block located at (*bs*, *bt*, *bv*) in tree level *lvl*.

```
RahtNeighCnt[lvl][bs][bt][bv] := SumN19[neighWeightGt0]
  where
    neighWeightGt0[ds][dt][dv] := RahtBlkWeight[lvl][bs + ds][bt + dt][bv + dv] > 0
```

The expression *RahtNeighCntMinAncestor*[*lvl*][*bs*][*bt*][*bv*] is lowest value of *RahtNeighCnt* for any ancestor of the block located at (*bs*, *bt*, *bv*) in tree level *lvl*. In determining eligibility, the root node shall be considered to have 19 adjoining blocks.

```
RahtNeighCntMinAncestor[lvl][bs][bt][bv] :=
    lvl >= RahtRootLvl - 1 ? 19 : Min(neighCntP, minAncestorCnt)
  where
    neighCntP := RahtNeighCnt[lvl + 1][bs / 2][bt / 2][bv / 2]
    minAncestorCnt := RahtNeighCntMinAncestor[lvl + 1][bs / 2][bt / 2][bv / 2]
```

The expression SumN19[*expr*] sums the result of applying *expr* to the relative tree location of each of the 19 possible adjacent blocks.

```
SumN19[expr] :=
  SumN19 = 0
  for (ds = −1; ds ≤ 1; ds++)
    for (dt = −1; dt ≤ 1; dt++)
      for (dv = −1; dv ≤ 1; dv++)
        if (Abs(ds) + Abs(dt) + Abs(dv) < 3)
          SumN19 += expr[ds][dt][dv]
```

10.5.5.4 Upsampling

10.5.5.4.1 Normalized DC values

The samples used to generate an upsampled prediction block are transform-block DC coefficients (10.5.6.2) normalized by their weight as specified by *RahtDcNorm*; *RahtDcNorm*[*lvl*][*bs*][*bt*][*bv*] is the sample value for the block located at (*bs*, *bt*, *bv*) in tree level *lvl*.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
RahtDcNorm[lvl][bs][bt][bv] :=
    DivExp2Fz((coeff >> wShift) × (IntRecipSqrt(w) >> 25 - wShift), 30)
  where
    w := RahtBlkWeight[lvl][bs][bt][bv]
    coeff := RahtDcCoeff[lvl][bs][bt][bv]
    wShift := w > 1024 ? IntLog2(w - 1) >> 1 : 0
```

10.5.5.4.2 Exclusion of adjoining blocks

Adjoining blocks shall be excluded from the upsampling process if either their weight is zero or the normalized DC value for their primary attribute component is:

— less than or equal to 0,2 times that of co-located block; or

— greater than or equal to 2,5 times that of the co-located block.

The expression *RahtPredExcluded*[*ds*][*dt*][*dv*] specifies whether the block with relative location (*ds*, *dt*, *dv*) is excluded from contributing to the upsampled prediction of the block (*Bs*, *Bt*, *Bv*).

```
RahtPredExcluded[ds][dt][dv] :=
    Cidx == 0 ? empty || sample ≤ limitMin || sample ≥ limitMax
              : … /* Value of RahtPredExcluded[ds][dt][dv] for Cidx == 0 */
  where
    empty := RahtBlkWeight[Lvl][Bs][Bt][Bv] == 0
    sample := 10 × RahtSample[Lvl][Bs + ds][Bt + dt][Bv + dv]
    limitMin :=  2 × RahtSample[Lvl][Bs][Bt][Bv]
    limitMax := 25 × RahtSample[Lvl][Bs][Bt][Bv]
```

10.5.5.4.3 Upsampled prediction block

The samples of the 2x2x2 prediction block are the weighted averages of the adjoining blocks' normalized DC values. For each sample, the weight for the adjoining block depends upon the relative positions of the block and the sample location.

The expression *RahtPred*[*m*] specifies the value for the Morton-coded sample location m; where:

— *SumN19*[*w*] is sum of the weight of each adjoining block;

— *SumN19*[*wNeigh*] is the sum of the weighted normalized DC values for each adjoining block;

— the 15 fractional-bit, fixed-point reciprocal of the sum of weights, *RahtPredRecipW*[*x*] is specified by Table 29.

NOTE Samples that correspond to child blocks with block weights equal to zero do not contribute to the upsampled prediction block.

```
RahtPred[m] := SumN19[wNeigh] × RahtPredRecipW[SumN19[w]]
  where
    w[ds][dt][dv] := RahtPredExcluded[ds][dt][dv] ? 0 : RahtPredWeight[ds][dt][dv][m]
    wNeigh[ds][dt][dv] := w[ds][dt][dv] × RahtDcNorm[Lvl][Bs + ds][Bs + dt][Bs + dv]
```

The expression *RahtPredWeight*[*ds*][*dt*][*dv*][*m*] is the weight to be applied to the normalized DC value of the block with relative tree location (*ds*, *dt*, *dv*) for the Morton-coded prediction block sample location *m*. The weight shall be 4 for the co-located block, 2 for blocks that adjoin by a face and 1 for blocks that adjoin by only an edge.

```
RahtPredWeight[ds][dt][dv][m] := (m & adjMask) == adjLoc ? weight : 0
  where
    weight := 4 >> (ds ≠ 0) + (dt ≠ 0) + (dv ≠ 0)
    adjMask := Morton(ds ≠ 0, dt ≠ 0, dv ≠ 0)
    adjLoc  := Morton(ds > 0, dt > 0, dv > 0)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Table 29 — Values of *RahtPredRecipW*[*x*]

x	0	1	2	3	4	5	6	7	8	9
***RahtPredRecipW*[*x*]**	8 192	6 554	5 461	4 681	4 096	3 641	3 277	2 979	2 731	2 521

10.5.5.5 Weighted prediction block

The forward transform of an upsampled prediction block shall use weighted sample values. Each sample shall be weighted by its corresponding transform coefficient weight. The weighted sample value is specified by *RahtPredW*[*m*] for the block located at (*Bs*, *Bt*, *Bv*) in tree level *Lvl*.

```
RahtPredW[m] := DivExp2Fz(RahtPred[m] × w, 15)
  where
    w := IntSqrt(RahtCoeffWeightM[Lvl][0][Bs][Bt][Bv][m] << 30)
```

10.5.5.6 Forward transform for a 2×2×2 block prediction block

The forward transform for a 2×2×2 prediction block comprises transforming pairs of coefficients along each axis.

First, along the V axis:

```
rahtFwd1D[2][0][1]
rahtFwd1D[2][2][3]
rahtFwd1D[2][4][5]
rahtFwd1D[2][6][7]
```

Second, along the T axis:

```
rahtFwd1D[1][0][2]
rahtFwd1D[1][1][3]
rahtFwd1D[1][4][6]
rahtFwd1D[1][5][7]
```

Third, along the S axis:

```
rahtFwd1D[0][0][4]
rahtFwd1D[0][1][5]
rahtFwd1D[0][2][6]
rahtFwd1D[0][3][7]
```

The expression *rahtFwd1D*[*k*][*aIdx*][*bIdx*] specifies the invocation of the in-place, forward, two-point transform for the *aIdx*-th and *bIdx*-th coefficients along the *k*-th axis.

```
rahtFwd1D[k][aIdx][bIdx] := RahtFwd(aCoeff, bCoeff, wa, wb)
  where
    aCoeff := RahtPredBlk[aIdx]
    bCoeff := RahtPredBlk[bIdx]
    wa := RahtCoeffWeightM[Lvl][stage][Bs][Bt][Bv][aIdx]
    wb := RahtCoeffWeightM[Lvl][stage][Bs][Bt][Bv][bIdx]
    stage := 2 - k
```

10.5.5.7 Forward two-point transform

This subclause specifies the in-place, forward, two-point transform RahtFwd(*aCoeff*, *bCoeff*, *wa*, *wb*). Its parameters are:

— the expressions *aCoeff* and *bCoeff* that identify the coefficients to be transformed in-place;

— the weights *wa* and *wb* that are the coefficient weights for *aCoeff* and *bCoeff*, respectively.

NOTE The specification of the forward two-point transform applies only to prediction blocks for the reconstruction of point attributes.

The transform basis vectors 15-fractional-bit, fixed-point coefficients *a* and *b*:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
a := IntSqrt(wa << 30) × IntRecipSqrt(wa + wb) >> 40
b := IntSqrt(wb << 30) × IntRecipSqrt(wa + wb) >> 40
```

If both *wa* or *wb* are 0, the transform result is:

```
if (wa == 0 && wb == 0)
  yl = yh = 0
```

If either *wa* or *wb* is 0, the transform result is:

```
if (wa == 0 || wb == 0) {
  yl = wa ≠ 0 ? aCoeff : bCoeff
  yh = 0
}
```

NOTE If either *wa* or *wb* is zero, the respective opposing coefficient *b* or *a* is not necessarily 2^{15}.

Otherwise (both *wa* and *wb* are greater than 0), the transform result is:

```
if (wa ≠ 0 && wb ≠ 0) {
  yl = DivExp2Fz(aCoeff × a, 15) + DivExp2Fz(bCoeff × b, 15)
  yh = DivExp2Fz(bCoeff × a, 15) − DivExp2Fz(aCoeff × b, 15)
}
```

The transform result replaces the input coefficients:

```
aCoeff = yl
bCoeff = yh
```

10.5.6 Inverse transform

10.5.6.1 General

This subclause specifies the inverse transform for a block located at (*Bs*, *Bt*, *Bv*) in tree level *Lvl*. It shall be applied to every block in the tree level in any order.

10.5.6.2 DC transform coefficient inheritance

Each block other than the root node of the transform tree shall inherit its DC coefficient from the corresponding inverse-transformed coefficient in its parent block. The inherited coefficient shall be rounded to retain two fractional bits, with half values rounded away from zero.

```
if (Lvl < RahtRootLvl)
  RahtCoeff[Lvl][Bs][Bt][Bv][0] = DivExp2Fz(RahtDcCoeff[Lvl][Bs][Bt][Bv], 13) << 13
```

For a block located at (*bs*, *bt*, *bv*) in tree level *lvl*, the corresponding coefficient in the parent block is specified by *RahtDcCoeff*[*lvl*][*bs*][*bt*][*bv*].

```
RahtDcCoeff[lvl][bs][bt][bv] := RahtCoeff[lvl + 1][bs / 2][bt / 2][bv / 2][mP]
  where
    mP := Morton[bs & 1][bt & 1][bv & 1]
```

10.5.6.3 For a 2×2×2 transform block

The inverse transform for a 2×2×2 block located at (*Bs*, *Bt*, *Bv*) in tree level *Lvl* comprises transforming pairs of coefficients along each axis.

First, along the S axis:

```
rahtInv1D[0][0][4]
rahtInv1D[0][1][5]
rahtInv1D[0][2][6]
rahtInv1D[0][3][7]
```

Second, along the T axis:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
rahtInv1D[1][0][2]
rahtInv1D[1][1][3]
rahtInv1D[1][4][6]
rahtInv1D[1][5][7]
```

Third, along the V axis:

```
rahtInv1D[2][0][1]
rahtInv1D[2][2][3]
rahtInv1D[2][4][5]
rahtInv1D[2][6][7]
```

The expression *rahtInv1D*[*k*][*aIdx*][*bIdx*] specifies the invocation of the in-place inverse transform for the *aIdx*-th and *bIdx*-th coefficients along the *k*-th axis of the block.

```
rahtInv1D[k][aIdx][bIdx] := RahtInv(aCoeff, bCoeff, wa, wb)
  where
    aCoeff := RahtCoeff[Lvl][Bs][Bt][Bv][aIdx]
    bCoeff := RahtCoeff[Lvl][Bs][Bt][Bv][bIdx]
    wa := RahtCoeffWeightM[Lvl][stage][Bs][Bt][Bv][aIdx]
    wb := RahtCoeffWeightM[Lvl][stage][Bs][Bt][Bv][bIdx]
    stage := 2 - k
```

10.5.6.4 For co-located points

The inverse transform for a block of co-located points located at (*Bs*, *Bt*, *Bv*) in tree level 0 comprises iteratively transforming the block's DC coefficient paired with each successive block coefficient.

```
for (i = 1; i < RahtBlkWeight[0][Bs][Bt][Bv]; i++)
  rahtInvDup[i]
```

The expression *rahtInvDup*[*i*] specifies the invocation of the in-place inverse transform for the *i*-th coefficient.

```
rahtInvDup[i] := RahtInv(aCoeff, bCoeff, wa, 1)
  where
    aCoeff := RahtCoeff[0][Bs][Bt][Bv][0]
    bCoeff := RahtCoeff[0][Bs][Bt][Bv][i]
    wa := RahtBlkWeight[0][Bs][Bt][Bv] - i
```

10.5.6.5 Inverse two-point transform

This subclause specifies the in-place, inverse, two-point transform RahtInv(*aCoeff*, *bCoeff*, *wa*, *wb*). Its parameters are:

— the expressions *aCoeff* and *bCoeff* that respectively identify low- and high-frequency transform coefficients;

— the weights *wa* and *wb* that are the respective coefficient weights for *aCoeff* and *bCoeff*.

The transform basis vectors use 15-fractional-bit, fixed-point coefficients *a* and *b*:

```
a := IntSqrt(wa << 30) × IntRecipSqrt(wa + wb) >> 40
b := IntSqrt(wb << 30) × IntRecipSqrt(wa + wb) >> 40
```

If either *wa* or *wb* is 0, the transform result is:

```
if (wa == 0 || wb == 0) {
  ya = wa ≠ 0 ? aCoeff : 0
  yb = wb ≠ 0 ? aCoeff : 0
}
```

NOTE If either *wa* or *wb* is zero, the respective opposing coefficient *b* or *a* is not necessarily 2^{15}.

Otherwise (both *wa* and *wb* are greater than 0), the transform result is:

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
if (wa ≠ 0 && wb ≠ 0) {
  ya = DivExp2Fz(aCoeff × a, 15) − DivExp2Fz(bCoeff × b, 15)
  yb = DivExp2Fz(bCoeff × a, 15) + DivExp2Fz(aCoeff × b, 15)
}
```

10.5.7 Reconstructed attribute values

Reconstructed attribute values are specified by the expression *RahtRecon*[*s*][*t*][*v*][*i*] for an *i*-th co-located point with attribute coordinates (*s*, *t*, *v*). They are:

— extracted from inverse transformed blocks in the bottom two tree levels; unique points from tree level 1; duplicate points from tree level 0; then

— rounded to discard the 15 fractional bits of the fixed-point representation, with halve values rounded away from zero; then

— clipped to be within the attribute value range [0, *AttrMaxVal*].

```
RahtRecon[s][t][v][i] := Clip3(0, AttrMaxVal, DivExp2Fz(value, 15))
  where
    value := RahtBlkWeight[0][s][t][v] == 1
      ? RahtDcCoeff[0][s][t][v]
      : RahtCoeff[0][s][t][v][i]
```

The mapping of the slice geometry to reconstructed attribute values shall map points with identical attribute coordinates to successive elements *i* of *RahtRecon*[*s*][*t*][*v*][*i*] in canonical point order. i.e. the *i*-th element shall be the *i*-th instance of the attribute coordinates (*s*, *t*, *v*) from the start of *AttrPos*.

The following is specified in terms of the sparse array *dupPtIdx*; *dupPtIdx*[*s*][*t*][*v*] is the cumulative count of points with attribute coordinates (*s*, *t*, *v*). Unset elements of *dupPtIdx* shall be inferred to be 0.

```
for (ptIdx = 0; ptIdx < PointCnt; ptIdx++) {
  s = AttrPos[ptIdx][0]
  t = AttrPos[ptIdx][1]
  v = AttrPos[ptIdx][2]
  i = dupPtIdx[s][t][v]
  dupPtIdx[s][t][v]++

  PointAttr[ptIdx][Cidx] = RahtRecon[s][t][v][i]
}
```

10.6 Attribute decoding using levels of detail

10.6.1 General

The attribute decoding processes specified by 10.6 are distance-based prediction schemes that use a hierarchical level-of-detail representation of the slice geometry. They apply when attr_coding_type is either 1 or 2.

Detail levels are defined by an iterative subsampling process (10.6.5). The finest detail level comprises all points in the slice 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- and intra-detail-level searches. The neighbouring points form a predictor set that is used to predict attribute/transform coefficient values.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Attribute reconstruction (10.6.7) proceeds from the coarsest to the finest detail level. Transform coefficients are coded in the same order.

A coded transform coefficient is associated with each refinement point. The transform (10.6.12) comprises two operations: an update step that modifies the predicting points and a prediction step that adds the transform coefficient to a predicted attribute/coefficient value.

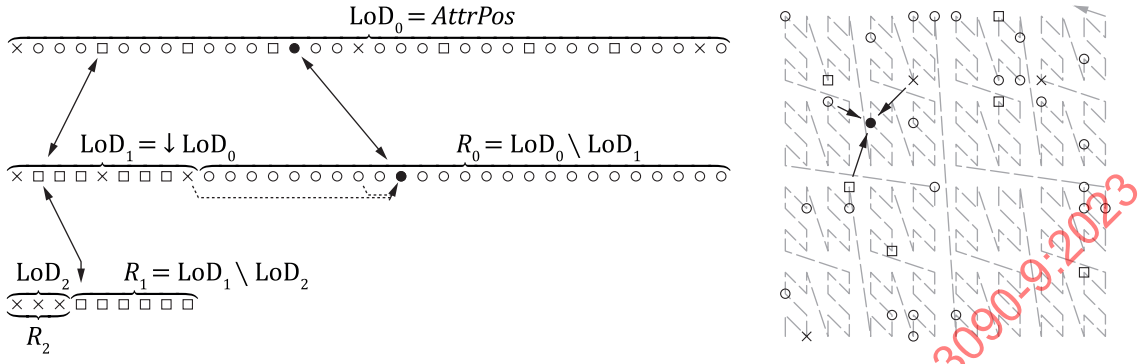


Figure 20 — Example of points in three detail levels and their spatial arrangement

An example level-of-detail hierarchy is illustrated in Figure 20. LoD_0 is the finest detail level and corresponds to all points in the slice. The generation of subsequent detail levels is performed using periodic subsampling with *samplingPeriod* equal to 4. The number of detail levels is limited to 3. The points in LoD_0 that are not assigned to LoD_1 are in refinement list R_0. The attribute value for the marked point • in R_0 for LoD_0 is predicted from a set of spatially neighbouring points found using an inter-detail-level search of LoD_1 and an intra-detail-level search of points earlier in R_0. Transform coefficient values are associated with each refinement point and coded from R_2 to R_0. The refinement list R_2 comprises all points in LoD_2.

10.6.2 Syntax element semantics

lod_dist_log2_offset specifies an offset to the APS-specified finest detail level block size lod_initial_dist_log2 for LoD generation and predictor searches. When lod_dist_log2_offset is not present, it shall be inferred to be 0.

10.6.3 Reconstruction process

The reconstruction of point attribute values comprises:

— deriving a set of detail levels from the slice geometry (10.6.5);

— searching for point predictors (10.6.6);

— determining transform coefficient weights (10.6.11); and

— reconstructing attribute values from coded coefficients (10.6.7).

The reconstructed values are stored in the array *PointAttr*.

10.6.4 State variables

Levels of detail are specified in terms of the following state variables; the index *lvl* identifies a detail level:

— The variable *LodCnt*, a count of detail levels generated from the slice geometry.

— The array *LodPtCnt*, the size of each detail level; *LodPtCnt*[*lvl*] is the number of points in the identified detail level.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— The array *LodPtIdx*, identifying points in each detail level by their canonical point order index; *LodPtIdx*[*lvl*][*i*] is the *AttrPos* index of the *i*-th point in the identified detail level.

— The array *LodRfmtPtCnt*, the size of each detail level's refinement list; *LodRfmtPtCnt*[*lvl*] is the number of points in the refinement list for the identified detail level.

— The array *LodRfmtPtIdx*, identifying points in each refinement list by their canonical point order index; *LodRfmtPtIdx*[*lvl*][*i*] is the *AttrPos* index of the *i*-th refinement point for the identified detail level.

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

— The array *PredCnt*; *PredCnt*[*ptIdx*] is the size of the predictor set for the identified point.

— The array *PredPtIdx*, identifies point predictors by their canonical point order index; *PredPtIdx*[*ptIdx*][*ni*] is the *AttrPos* index of the *ni*-th point in the predictor set for the identified point.

— The array *PredWeight* of point predictor weights; *PredWeight*[*ptIdx*][*ni*] is the prediction weight for the predictor identified by *PredPtIdx*[*ptIdx*][*ni*].

— The array *CoeffWeight* of transform coefficient weights; *CoeffWeight*[*ptIdx*] is the normalization weight for the transform coefficients associated with the identified point.

10.6.5 Levels of detail

10.6.5.1 General generation process

The effect of this process is to represent the LoD structure in the state variables *LodCnt*, *LodPtCnt*, *LodPtIdx*, *LodRfmtPtCnt* and *LodRfmtPtIdx*.

The finest detail level shall contain the entire slice geometry (10.6.5.2). It is identified by the detail level index 0.

Detail levels shall be iteratively subsampled (10.6.5.4), starting from the finest detail level, until either a single point remains or lod_max_levels_minus1 subsampled detail levels have been produced. The variable *Lvl* identifies the detail level to be subsampled.

```
Lvl = 0
for (; Lvl < lod_max_levels_minus1; 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).

10.6.5.2 The finest detail level

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

```
for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)
  LodPtIdx[0][ptIdx] = ptIdx
LodPtCnt[0] = PointCnt
```

Unless attr_canonical_order_enabled is 1, the point indexes of the finest detail level shall be sorted in ascending order of their respective Morton-coded attribute coordinates. The sorted order shall be

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

identical for the decoding of all attributes in a single slice with identical attribute coordinate arrays (*AttrPos*).

NOTE 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 (i = 0; i < LodPtCnt[0]; i++)
  for (j = i + 1; j < LodPtCnt[0]; j++) {
    iPtIdx = LodPtIdx[0][i]
    jPtIdx = LodPtIdx[0][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[0][i], LodPtIdx[0][j])
  }
```

10.6.5.3 The coarsest detail level

After generation of the LoD hierarchy, all points in the coarsest detail level shall be assigned to the its refinement list.

```
for (i = 0; i < LodPtCnt[LodCnt - 1]; i++)
  LodRfmtPtIdx[LodCnt - 1][i] = LodPtIdx[LodCnt - 1][i]
```

10.6.5.4 Generation of a single detail level

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

The following definitions are used in the specification of the subsampling processes:

— The expression *InLodPtCnt* is an alias for *LodPtCnt*[*Lvl*], the number of points in the input detail level.

— The expression *InLodPtIdx*[*i*] is an alias for *LodPtIdx*[*Lvl*][*i*], the point indexes of the input detail level.

— The expression *OutLodPtCnt* is an alias for *LodPtCnt*[*Lvl* + 1], the number of points in the output detail level.

— The expression *OutLodPtIdx*[*i*] is an alias for *LodPtIdx*[*Lvl* + 1][*i*], the point indexes of the output detail level.

— The expression *OutRfmtPtIdx*[*i*] is an alias for *LodRfmtPtIdx*[*Lvl*][*i*], the point indexes of the refinement list for detail level *Lvl*.

Subsampling partitions points in the input detail level into an output detail level and the refinement list for the input detail level. The partitioning process shall preserve the relative ordering of points in the input detail level.

Subsampling shall proceed according to:

— block-based subsampling (10.6.5.8) if lod_scalability_enabled is 1, or lod_decimation_mode is 2;

— periodic subsampling (10.6.5.5) if lod_decimation_mode is 1; or

— distance-based subsampling (10.6.5.6) otherwise.

10.6.5.5 Periodic subsampling

Periodic subsampling generates a subsampled output detail level by sampling every one-in-sampling-period points in the input detail level.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

The sampling period for the current detail level is *subsamplingPeriod*.

```
samplingPeriod := 2 + lod_sampling_period_minus2[Lvl]
```

Input points shall be assigned to either the output detail level or the refinement list according to their index in the input detail level modulo the sampling period:

```
OutLodPtCnt = outRfmtPtCnt = 0
for (i = 0; i < InLodPtCnt; i++) {
  if (i % samplingPeriod)
    OutRfmtPtIdx[outRfmtPtCnt++] = InLodPtIdx[i]
  else
    OutLodPtIdx[OutLodPtCnt++] = InLodPtIdx[i]
}
```

10.6.5.6 Distance-based subsampling

Distance-based subsampling generates a subsampled output detail level by:

— spatially partitioning the input detail level into a lattice of $2^{BlkSizeLog2}$ sized cubic blocks; and

— assigning at most one point from each block to the subsampled detail level.

```
BlkSizeLog2 := lod_initial_dist_log2 + lod_dist_log2_offset + Lvl + 1
```

The subsampling process is specified in terms of the following state variables; the indexes *bs*, *bt* and *bv* identify the block location (*bs*, *bt*, *bv*):

— The sparse array *MapSub*; *MapSub*[*bs*][*bt*][*bv*] equal to 1 indicates that the identified block contains a single point previously assigned to the subsampled detail level. Unset elements of *MapSub* are inferred to be 0.

— The sparse array *MapPtIdx*, identifies points assigned to the subsampled detail level. When *MapSub*[*bs*][*bt*][*bv*] is 1, *MapPtIdx*[*bs*][*bt*][*bv*] is the *AttrPos* index of the point assigned to the subsampled detail level.

The points in the input detail level shall be processed sequentially. For each input point:

— The variable *PtIdx* is the *AttrPos* index of the point.

— The block location (*Bs*, *Bt*, *Bv*) is determined from the point's attribute coordinates.

— Depending upon the result of a per-point test (10.6.5.7), the point shall be assigned to either the output detail level or the refinement list. The result of the test is the variable *IsSubsampledPoint*.

```
OutLodPtCnt = outRfmtPtCnt = 0
for (i = 0; i < InLodPtCnt; i++) {
  PtIdx = InLodPtIdx[i]
  Bs = AttrPos[PtIdx][0] >> BlkSizeLog2
  Bt = AttrPos[PtIdx][1] >> BlkSizeLog2
  Bv = AttrPos[PtIdx][2] >> BlkSizeLog2

  … /* IsSubsampledPoint = result of per-point test (10.6.5.7) */

  if (MapSub[Bs][Bt][Bv] || ¬IsSubsampledPoint)
    OutRfmtPtIdx[outRfmtPtCnt++] = PtIdx
  else {
    OutLodPtIdx[OutPtCnt++] = PtIdx
    MapSub[Bs][Bt][Bv] = 1
    MapPtIdx[Bs][Bt][Bv] = PtIdx
  }
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

10.6.5.7 Per-point decision for distance-based subsampling

The derivation of *IsSubsampledPoint* specifies whether the point shall be assigned to the output detail level.

A point shall be assigned to the output detail level unless the squared distance between it and any previously assigned point from a set of adjacent blocks within an availability window is less than or equal to *sqRadius*. Each availability window shall be a 128×128×128 block volume identified by (*Bs* >> 7, *Bt* >> 7, *Bv* >> 7).

```
sqRadius = 3 << 2 × (BlkSizeLog2 − 1)
```

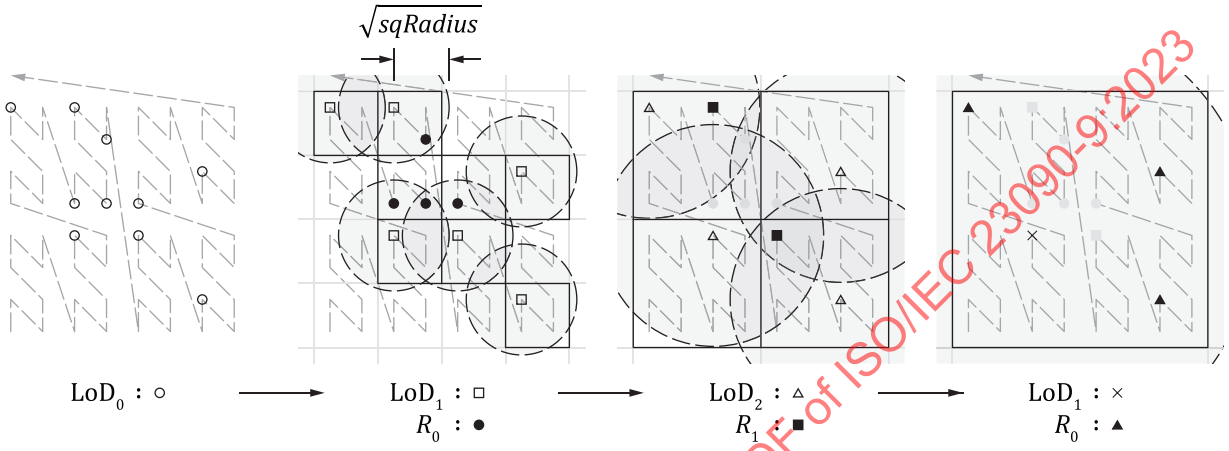


Figure 21 — Example decisions using distance-based subsampling

Example per-point decisions are illustrated in Figure 21. Subsampling generates three detail levels. The points assigned to LoD_1 by subsampling LoD_0 (when *Lvl* = 0) are not within the shaded radius $\sqrt{sqRadius}$ of any other point in LoD_1. The block size used to subsample LoD_0 is *BlkSizeLog2* = 1. All points are within a single availability window.

The array *neighPtIdx* is a *neighCnt*-element list of *AttrPos* indexes of points present in the adjacent blocks of the output detail level that are within the availability window. Table 30 specifies the relative locations of the adjacent blocks.

```
neighCnt = 0
for (i = 0; i < 19; i++) {
  ns = Bs + adjBlkOffset[i][0]
  nt = Bt + adjBlkOffset[i][1]
  nv = Bv + adjBlkOffset[i][2]
  unavailable = (ns ^ Bs) >> 7 || (nt ^ Bt) >> 7 || (nv ^ Bv) >> 7
  if (unavailable)
    continue

  if (MapSub[ns][nt][nv])
    neighPtIdx[neighCnt++] = MapPtIdx[ns][nt][nv]
}
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

Table 30 — Adjacent block coordinates, *adjBlkOffset*[*i*][*k*], relative to (*Bs*, *Bt*, *Bv*)

i	*k*			*i*	*k*			*i*	*k*			*i*	*k*		
	0	1	2		0	1	2		0	1	2		0	1	2
0	−1	0	0	**5**	−1	1	0	**10**	−1	0	−1	**15**	1	−1	−1
1	0	−1	0	**6**	0	1	−1	**11**	−1	−1	0	**16**	−1	1	−1
2	0	0	−1	**7**	1	0	−1	**12**	−1	1	1	**17**	−1	−1	1
3	0	−1	1	**8**	1	−1	0	**13**	1	−1	1	**18**	−1	−1	−1
4	−1	0	1	**9**	0	−1	−1	**14**	1	1	−1				

The point's attribute coordinates shall be compared to those of each point identified by the *neighPtIdx* array to determine the value of *IsSubsampledPoint*.

```
IsSubsampledPoint = 1
for (i = 0; i < neighCnt; i++) {
  sqDist = 0
  for (k = 0; k < 3; k++) {
    d = AttrPos[neighPtIdx[i]][k] - AttrPos[PtIdx][k]
    sqDist += d × d
  }

  if (sqDist ≤ sqRadius)
    IsSubsampledPoint = 0
}
```

10.6.5.8 Block-based subsampling

Block-based subsampling generates a subsampled output detail level by:

— spatially partitioning the input detail level into a lattice of $2^{BlkSizeLog2}$ sized cubic blocks;

— grouping together blocks, in Morton order, according to the number of points they contain; and

— assigning one point from each block group to the subsampled detail level.

```
BlkSizeLog2 := lod_initial_dist_log2 + lod_dist_log2_offset + Lvl + 1
```

NOTE Under certain conditions, blocks correspond to nodes of the occupancy tree. For instance, when lod_scalability_enabled is 1.

A list of block groups shall be generated by traversing the input detail level in canonical point order. Consecutive blocks shall be grouped together until the group spans at least *minGrpPts* points.

```
minGrpPts := lod_scalability_enabled ? 0 : 2 + lod_sampling_period_minus2[Lod]
```

The array *grpBdry*, with elements *grpBdry*[*grpIdx*], identifies block group boundaries as indexes into the input detail level array *InLodPtIdx*.

```
for (i = 1, grpStart = 0; i < InLodPtCnt; i++) {
  ptIdx = InLodPtIdx[i]
  ptIdxPrev = InLodPtIdx[i - 1]
  bdryS = (AttrPos[ptIdx][0] ^ AttrPos[ptIdxPrev][0]) >> BlkSizeLog2
  bdryT = (AttrPos[ptIdx][1] ^ AttrPos[ptIdxPrev][1]) >> BlkSizeLog2
  bdryV = (AttrPos[ptIdx][2] ^ AttrPos[ptIdxPrev][2]) >> BlkSizeLog2
  if (bdryS | bdryT | bdryV)
    if (i - grpStart ≥ minGrpPts)
      grpBdry[grpCnt++] = grpStart = i
}
grpBdry[grpCnt++] = InLodPtCnt
```

For each group of blocks, a test (10.6.5.9) shall be performed to determine the index of the point to be assigned to the output detail level. All other points shall be assigned to the refinement list. The

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

variables *GrpStart* and *GrpEnd* identify the start and end of a block group. The result of the test is the variable *IdxOfSubsampledPoint*.

```
OutLodPtCntSize = outRfmtPtCnt = 0
for (GrpStart = grpIdx = 0; grpIdx < grpCnt; GrpStart = grpBdry[grpIdx++]) {
  GrpEnd = grpBdry[grpIdx]

  … /* IdxOfSubsampledPoint = result of per-point test (10.6.5.9) */

  for (i = GrpStart; i < GrpEnd; i++) {
    if (IdxOfSubsampledPoint == i)
      OutLodPtIdx[OutLodPtCnt++] = InLodPtIdx[i]
    else
      OutRfmtPtIdx[outRfmtPtCnt++] = InLodPtIdx[i]
  }
}
```

10.6.5.9 Per block-group decision for block-based subsampling

The derivation of *IdxOfSubsampledPoint* specifies the input detail level index of the point in the block group that shall be assigned to the output detail level.

The distance to the block group centroid shall be used to select the point assigned to the output detail level. The block group centroid and point distances shall be calculated using attribute coordinates quantized by Exp2(*BlkSizeLog2* − 1). The distance metric shall be the Manhattan distance.

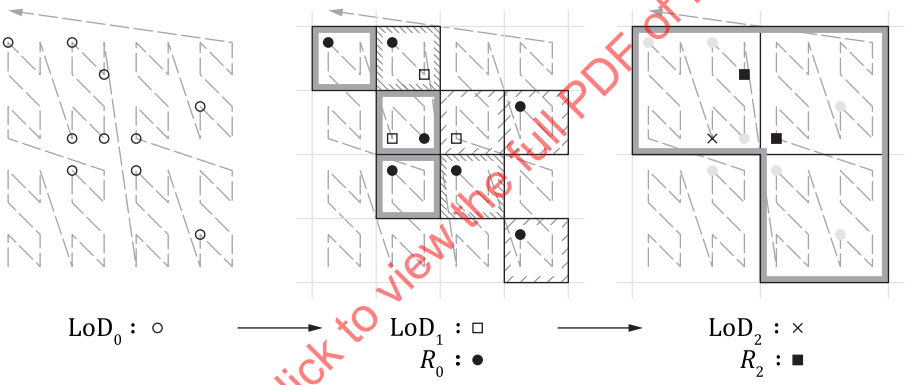


Figure 22 — Example decisions using block group subsampling with *minGrpPts* = 3

Example per-point decisions are illustrated in Figure 22. Subsampling generates two detail levels. To subsample LoD_0, the points are grouped into block groups containing a minimum of three points. In this example, the block size for LoD_0 (*Lvl* = 0) is *BlkSizeLog2* = 1. The first block group (solid shading) comprises four points from three blocks each with one, one and two points, respectively. The first point that is the closest to the centroid of the points in the block group is assigned to LoD_1.

The block group centroid shall be the sum of all quantized attribute coordinates, *centroidSum*, divided by the number of points in the block group, *numPtsInGrp*.

```
numPtsInGrp := GrpEnd − GrpStart

for (k = 0; k < 3; k++)
  centroidSum[k] = 0

for (i = 0; i < numPtsInGrp; i++) {
  ptIdx = InLodPtIdx[GrpStart + i]
  for (k = 0; k < 3; k++)
    centroidSum[k] += AttrPos[ptIdx][k] >> BlkSizeLog2 − 1
}
```

The array *ptDist* maps the index of each point in the block group to the distance between it and the centroid.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
for (i = 0; i < numPtsInGrp; i++) {
  ptIdx = InLodPtIdx[GrpStart + i]
  ptDist[i] = 0
  for (k = 0; k < 3; k++) {
    posk = AttrPos[ptIdx][k] >> BlkSizeLog2 - 1
    ptDist[i] += Abs(posk × numPtsInGrp - centroidSum[k])
  }
}
```

The point closest to the block group centroid shall be assigned to the output detail level. In the case that the block group contains multiple closest points, the selected point is the closest point with:

— when lod_scalability_enabled is 1 and *Lvl* is odd: the greatest *InLodPtIdx* index;

— when lod_scalability_enabled is 0 or *Lvl* is even: the lowest *InLodPtIdx* index.

```
last := lod_scalability_enabled ? Lvl & 1 : 1
minIdx = 0
for (i = 1; i < numPtsInGrp; i++)
  if (last ? dist[i] ≤ dist[minIdx] : dist[i] < dist[minIdx])
    minIdx = i
IdxOfSubsampledPoint = GrpStart + minIdx
```

10.6.6 Predictor search

10.6.6.1 General process

The points used to predict the refinement points of each detail level shall be determined by a search (10.6.6.3).

The effect of this process is to represent the point predictors in the state variables *PredCnt*, *PredPtIdx* and *PredWeight*.

When attr_coding_type is 2, no searches shall be performed for the refinement points of the coarsest detail level.

```
maxLvl = LodCnt - (attr_coding_type == 2)
for (Lvl = 0; Lvl < maxLvl; Lvl++)
  for (RfmtIdx = 0; RfmtIdx < LodRfmtPtCnt[Lvl]; RfmtIdx++) {
    … /* find predictors (10.6.6.3) of the current point */
  }
```

10.6.6.2 Minimum reference detail level for inter-level predictor searches

The variable *MinInterRefLvl* identifies the finest detail level that shall be used as a reference for inter-detail level prediction. When lod_scalability_enabled is 1, it shall be the finest detail level with fewer refinement points than the total number of refinement points associated with all finer detail levels.

```
MinInterRefLvl = 1
if (lod_scalability_enabled) {
  for (lvl = 1; lvl < LodCnt - 1; lvl++) {
    if (LodRfmtPtCnt[lvl] < slice_num_points_minus1 - LodPtCnt[lvl])
      break
    MinInterRefLvl++
  }
}
```

10.6.6.3 Predictor search for a single refinement point

For a refinement point with index *RfmtIdx* in detail level *Lvl*, a search shall be performed to find the closest neighbouring points from a set of candidate neighbours.

The search process is specified in terms of the following variables:

— The variable *PtIdx*, the *AttrPos* index of the refinement point.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— The variable *RefLvl*, the reference detail level used for inter-level predictor searches.

```
PtIdx = LodRfmtPtIdx[RfmtIdx]
RefLvl = Max(Lvl + 1, MinInterRefLvl)
```

An inter-detail-level search shall be performed prior to any intra-level search. Except for the coarsest detail level, the following inter-level searches shall be performed:

— an initial search (10.6.6.6);

— if fewer than three predictors are found (*PredCnt*[*PtIdx*] < 3), an extended search (10.6.6.7).

When *Lvl* is greater than or equal to pred_intra_min_lod, an intra-detail-level search (10.6.6.8) shall be performed.

After completing the searches, weights shall be calculated for each predictor (10.6.6.9), during which the predictor set is pruned and re-sorted. When pred_blending_enabled is 1, predictor weights shall be blended (10.6.6.10).

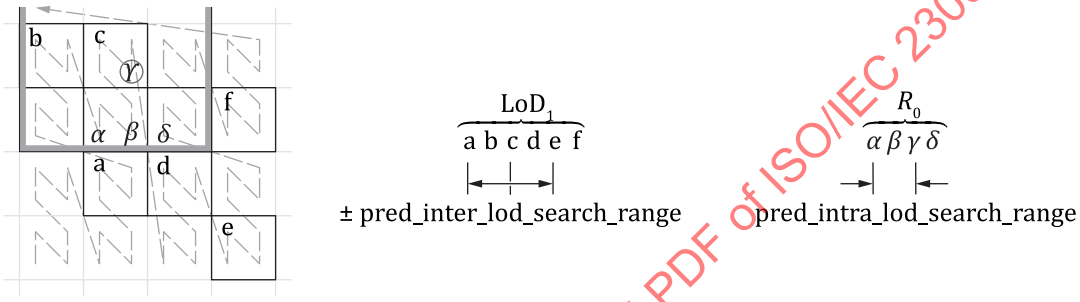


Figure 23 — Example of searches performed for a single refinement point

An example predictor search is illustrated in Figure 23. Searches are performed for the refinement point γ in R_0 of LoD_0; other points in R_0 are denoted α, β and δ. Points in the next coarsest detail level, LoD_1, are marked a to f. After the initial inter-level search, the predictor set is { c, b }. Then, since fewer than three predictors were found, an extended inter-level search is performed over ±pred_inter_lod_search_range points in the LoD_1 point list. This search adds predictor d to the predictor set { c, b, d }. Finally, an intra-level search is performed over pred_intra_lod_search_range previous points in R_0. The final predictor set for γ is { c, β, α }.

10.6.6.4 Inclusion of a candidate point in the predictor set (InsertPredictor)

This subclause defines the function InsertPredictor(*candPtIdx*) that conditionally inserts a candidate point into the predictor set of the current refinement point. Each candidate shall be tested against the refinement point's predictor set to determine if and where it is to be inserted.

The parameter *candPtIdx* is the *AttrPos* index of the candidate point.

A candidate shall only be inserted into the a point's neighbour set once. If *candPresent* is 1, the candidate is not inserted into the predictor set.

```
candPresent = 0
for (i = 0; i < PredCnt[PtIdx]; i++)
  candPresent |= PredPtIdx[PtIdx][i] == candPtIdx
```

Otherwise (the candidate is not already present), the spatial distance between the candidate and the refinement point shall be used to decide the inclusion in the predictor set. The distance shall be calculated as the biased ℓ^1 norm weighted by *PredBias*.

```
dist = BiasedNorm1(PtIdx, candPtIdx)
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

The point shall be inserted into the predictor set, with elements ordered according to the biased ℓ^1 distance to the refinement point. Points at the same distance shall be ordered by insertion order, with earlier members being ordered before later members.

```
for (i = 0; i < PredCnt[PtIdx]; i++)
  if (dist < BiasedNorm1(PtIdx, PredPtIdx[PtIdx][i]))
    break
for (j = PredCnt[PtIdx]; j > i; j--)
  PredPtIdx[PtIdx][j] = PredPtIdx[PtIdx][j - 1]
PredPtIdx[PtIdx][i] = candPtIdx
```

The size of the predictor set shall be limited to three elements by discarding the furthest predictor if necessary.

```
PredCnt[PtIdx] = Min(3, PredCnt[PtIdx] + 1)
```

10.6.6.5 Distance computation using the biased L1 norm (BiasedNorm1)

This subclause defines the function BiasedNorm1(*ptIdxA*, *ptIdxB*) that is the weighted Manhattan distance between two points.

The parameters *ptIdxA* and *ptIdxB* are two *AttrPos* indexes.

The result of this function is specified by the expression BiasedNorm1. The expression *pos*[*ptIdx*][*k*] specifies the attribute coordinates used to calculate the distance: when lod_scalability_enabled is 1, coordinates shall be quantized according to the detail level.

```
BiasedNorm1(ptIdxA, ptIdxB) := dist[0] + dist[1] + dist[2]
  where
    dist[k] := Abs(pos[ptIdxA][k] - pos[ptIdxB][k]) × PredBias[k]
    pos[ptIdx][k] := lod_scalability_enabled
      ? (AttrPos[ptIdx][k] >> Lvl) << Lvl
      : AttrPos[ptIdx][k]
```

10.6.6.6 Initial inter-level predictor search

The initial inter-level search shall be performed by spatially partitioning the reference detail level into a lattice of $2^{BlkSizeLog2}$ sized cubic blocks. Only blocks adjacent to the block containing the refinement point that are within an availability window shall be searched.

```
BlkSizeLog2 := lod_initial_dist_log2 + lod_dist_log2_offset + Lvl + 1
```

The block location (*bs*, *bt*, *bv*) identifies the block containing the refinement point.

```
bs := AttrPos[PtIdx][0] >> BlkSizeLog2
bt := AttrPos[PtIdx][1] >> BlkSizeLog2
bv := AttrPos[PtIdx][2] >> BlkSizeLog2
```

The availability window shall be a 128×128×128 block volume identified by (*bs* >> 7, *bt* >> 7, *bv* >> 7).

The search shall proceed over the search blocks in the order specified by Table 31. Within each search block, points shall be searched in ascending order of index within the reference detail level.

```
for (si = 0; si < 27; si++) {
  ss = bs + searchBlkOffsets[si][0]
  st = bt + searchBlkOffsets[si][1]
  sv = bv + searchBlkOffsets[si][2]
  unavailable = (ss ^ bs) >> 7 || (st ^ bt) >> 7 || (sv ^ bv) >> 7
  if (unavailable)
    continue

  for (i = 0; i < LodPtCnt[RefLvl]; i++) {
    candPtIdx = LodPtIdx[RefLvl][i]
    cs = AttrPos[candPtIdx][0]
    ct = AttrPos[candPtIdx][1]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
    cv = AttrPos[candPtIdx][2]

    inSblk  = cs ≥ (ss << BlkSizeLog2) && cs < (ss + 1 << BlkSizeLog2)
    inSblk &= ct ≥ (st << BlkSizeLog2) && ct < (st + 1 << BlkSizeLog2)
    inSblk &= cv ≥ (sv << BlkSizeLog2) && cv < (sv + 1 << BlkSizeLog2)

    if (inSblk)
      InsertPredictor(candPtIdx)
  }
}
```

NOTE For each search block, the indices *i* for which *inSblk* is true are consecutive.

Table 31 — Search block coordinates, *searchBlkOffsets*[*i*][*k*], relative to (*bs*, *bt*, *bv*)

i	*k*			*i*	*k*			*i*	*k*			*i*	*k*		
	0	1	2		0	1	2		0	1	2		0	1	2
0	0	0	0	**7**	0	1	1	**14**	1	0	–1	**21**	1	–1	1
1	–1	0	0	**8**	1	0	1	**15**	1	–1	0	**22**	1	1	–1
2	0	–1	0	**9**	1	1	0	**16**	0	–1	–1	**23**	1	–1	–1
3	0	0	–1	**10**	0	–1	1	**17**	–1	0	–1	**24**	–1	1	–1
4	1	0	0	**11**	–1	0	1	**18**	–1	–1	0	**25**	–1	–1	1
5	0	1	0	**12**	–1	1	0	**19**	1	1	1	**26**	–1	–1	–1
6	0	0	1	**13**	0	1	–1	**20**	–1	1	1				

10.6.6.7 Extended inter-level search

The extended inter-level search evaluates predictor candidates over a span of indexes in the reference detail level.

The span shall be centred around the index *centre* in the reference detail level. It shall be the index of:

— if at least one predictor has been found for the current point: the first predictor; or

```
if (PredCnt[PtIdx])
  for (centre = 0; centre < LodPtCnt[RefLvl] - 1; centre++)
    if (LodPtIdx[RefLvl][centre] == PredPtIdx[PtIdx][0])
      break
```

— otherwise (no predictors have been found): the first point with Morton-coded attribute coordinates greater than those of the current point.

```
if (PredCnt[PtIdx] == 0) {
  mortonCurPt = Morton[AttrPos[PtIdx]]
  for (centre = 0; centre < LodPtCnt[RefLvl] - 1; centre++) {
    mortonCentre = Morton[AttrPos[LodPtIdx[RefLvl][centre]]]
    if (mortonCurPt < mortonCentre)
      break
  }
}
```

The extended search shall proceed over each index offset *i* of the following, in order: 0, +1, –1, +2, –2, +3 .. pred_inter_lod_search_range, and –(3 .. pred_inter_lod_search_range).

A predictor candidate shall be evaluated for each valid search index *centre* + *i* that is within the range specified by pred_inter_lod_search_range and does not exceed the bounds of the reference detail level.

```
if (Abs(i) ≤ pred_inter_lod_search_range)
  if (centre + i ≥ 0 && centre + i < LodPtCnt[RefLvl])
    InsertPredictor(LodPtIdx[RefLvl][centre + i])
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

10.6.6.8 Intra-level search

The intra-level search evaluates predictor candidates over a span of indexes in the refinement list of the current detail level. Intra-level predictor candidates shall precede the refinement point in the refinement list.

A predictor candidate shall be evaluated for each valid search index offset $-i$ from the refinement point, for i = 1 .. pred_inter_lod_search_range, that does not exceed the bounds of the refinement list.

```
for (i = 1; i ≤ Min(RfmtIdx, pred_intra_lod_search_range); i++)
  InsertPredictor(LodRfmtPtIdx[Lvl][RfmtIdx − i])
```

10.6.6.9 Predictor set pruning and generation of prediction weights

After the predictor search for a refinement point is complete, its predictor set shall be pruned, weights computed for each qualifying predictor and the predictors ordered according to weight.

The size of the predictor set shall be limited to pred_set_size_minus1 + 1 elements by discarding the furthest predictors if necessary.

```
PredCnt[PtIdx] = Min(pred_set_size_minus1 + 1, PredCnt[PtIdx])
```

Predictor weights shall be calculated using the biased squared distance between each predictor and the current point.

```
for (ni = 0; ni < PredCnt[PtIdx]; ni++)
  dist[ni] = BiasedNorm2(PtIdx, PredPtIdx[PtIdx][ni])
```

If the first predictor is spatially coincident with the current point, all other predictors shall be discarded.

```
if (dist[0] == 0)
  PredCnt[PtIdx] = 1
```

When lod_scalability_enabled is 1, predictors with an unbiased squared distance greater than a threshold shall be discarded:

```
if (lod_scalability_enabled) {
  threshold = 3 × (pred_max_range_minus1 + 1) << 2 × Lvl
  for (ni = 1; ni < PredCnt[PtIdx]; ni++)
    if (Norm2(ptIdx, PredPtIdx[PtIdx][ni]) > threshold) {
      PredCnt[PtIdx] = ni
      break
    }
}
```

The predictors shall be reordered according to their biased squared distance to the current point:

— An array *order* shall have elements such that *dist*[*order*[i]], for i = 0 .. *PredCnt*[*PtIdx*] − 1, is an ascending stable sorting of the array *dist*.

— The members of the predictor set and the *dist* array shall be permuted according to the elements of the array *order*.

The predictor distances shall be normalized by the smallest distance to produce initial weights.

```
n = Max(0, IntLog2(dist[0] − 8))
for (ni = 0; ni < PredCnt[PtIdx]; ni++)
  weight[ni] = DivExp2Up(dist[ni], n)
```

Any predictors with a weight 256 times greater than or equal to the smallest weight shall be discarded.

```
if (PredCnt[PtIdx] == 3 && weight[2] ≥ 256 × weight[0])
  PredCnt[PtIdx] = 2

if (PredCnt[PtIdx] == 2 && weight[1] ≥ 256 × weight[0])
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
  PredCnt[PtIdx] = 1
```

The final weights shall be derived as:

```
if (PredCnt[PtIdx] == 1)
  PredWeight[PtIdx][0] = 256

if (PredCnt[PtIdx] == 2) {
  PredWeight[PtIdx][1] = Div(weight[0], weight[0] + weight[1], 8)
  PredWeight[PtIdx][0] = 256 - PredWeight[PtIdx][1]
}
if (PredCnt[PtIdx] == 3) {
  d1d2 = weight[1] × weight[2]
  d0d2 = weight[0] × weight[2]
  d0d1 = weight[0] × weight[1]
  sum = d1d2 + d0d2 + d0d1
  PredWeight[PtIdx][2] = Div(d0d1, sum, 8)
  PredWeight[PtIdx][1] = Div(d0d2, sum, 8)
  PredWeight[PtIdx][0] = 256 - PredWeight[PtIdx][1] - PredWeight[PtIdx][2]
}
```

10.6.6.10 Blending of predictor weights

When a point has three predictors in its predictor set and pred_blending_enabled is 1, the predictor weights shall be blended according to the distance between the predicting points.

The squared distance between each of the three predictors shall be determined:

```
distA := Norm2(PredPtIdx[PtIdx][0], PredPtIdx[PtIdx][1])
distB := Norm2(PredPtIdx[PtIdx][0], PredPtIdx[PtIdx][2])
distC := Norm2(PredPtIdx[PtIdx][1], PredPtIdx[PtIdx][2])
```

Blending weights shall be selected according to distance:

```
b1 := distA ≤ distB ? 1 : 5
b2 := distA ≤ distC ? 5 : 1
b3 := distB ≤ distC ? 1 : 5
```

The predictor weights shall be blended and updated:

```
if (PredCnt[PtIdx] == 3 && pred_blending_enabled) {
  w0 = PredWeight[PtIdx][0]
  w1 = PredWeight[PtIdx][1]
  w2 = PredWeight[PtIdx][2]

  w0p = w0 × 10 + w1 × (6 - b2) + w2 × b3 >> 4
  w1p = w0 × b1 + w2 × (6 - b3) + w1 × 10 >> 4

  PredWeight[PtIdx][0] = w0p
  PredWeight[PtIdx][1] = w1p
  PredWeight[PtIdx][2] = 256 - w0p - w1p
}
```

10.6.6.11 Distance computation using the biased L2 norm (BiasedNorm2)

This subclause defines the function BiasedNorm2(*ptIdxA*, *ptIdxB*) that is the weighted squared distance between two points.

The parameters *ptIdxA* and *ptIdxB* are two *AttrPos* indexes.

The result of this function is specified by the expression BiasedNorm2. The expression *pos*[*ptIdx*][*k*] represents the attribute coordinates used to calculate the distance: when lod_scalability_enabled is 1, coordinates shall be quantized according to the detail level.

```
BiasedNorm2(ptIdxA, ptIdxB) := dist2[0] + dist2[1] + dist2[2]
  where
    dist[k] := Abs(pos[ptIdxA][k] - pos[ptIdxB][k]) × PredBias[k]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
    dist2[k] := dist[k] × dist[k]
    pos[ptIdx][k] := lod_scalability_enabled
      ? (AttrPos[ptIdx][k] >> Lvl) << Lvl
      : AttrPos[ptIdx][k]
```

10.6.6.12 Distance computation using the unbiased L2 norm (Norm2)

This subclause defines the function Norm2(*ptIdxA*, *ptIdxB*) that is the squared distance between two points.

The arguments *ptIdxA* and *ptIdxB* are two *AttrPos* indexes.

The result of this function is specified by the expression Norm2. The expression *pos*[*ptIdx*][*k*] represents the attribute coordinates used to calculate the distance. when lod_scalability_enabled is 1, coordinates shall be quantized according to the detail level.

```
Norm2(ptIdxA, ptIdxB) := dist2[0] + dist2[1] + dist2[2]
  where
    dist[k] := Abs(pos[ptIdxA][k] − pos[ptIdxB][k])
    dist2[k] := dist[k] × dist[k]
    pos[ptIdx][k] := lod_scalability_enabled
      ? (AttrPos[ptIdx][k] >> Lvl) << Lvl
      : AttrPos[ptIdx][k]
```

10.6.7 Reconstruction of attribute values

10.6.7.1 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 ≥ 0; Lvl−−)
  … /* process a detail level */
```

10.6.7.2 Coefficient processing order within a detail level

Within a detail level, processing proceeds in coded coefficient order. The variable *PtIdx* is the *AttrPos* index of the current coefficient. The variable *CoeffIdx* is the *AttrCoeff* array index of the current coefficient.

```
for (rfmtIdx = 0; rfmtIdx < LodRfmtPtCnt[Lvl]; rfmtIdx++) {
  PtIdx = LodRfmtPtIdx[Lvl][rfmtIdx]
  CoeffIdx = LodPtCnt[Lvl] − rfmtIdx
  … /* process current coefficient */
}
```

10.6.7.3 Processing a detail level (attr_coding_type = 1)

When attr_coding_type is 1, the following operations shall be performed in turn for each coefficient in the coefficient processing order of the current detail level:

— Prediction mode information is decoded from an encoded coefficient tuple (10.6.8.1). The result is the variable *PredMode*.

— The unencoded coefficient tuple is scaled (10.6.9.1) to produce transform coefficients.

— Transform coefficient components are divided by 256 with half-values rounded up.

```
for (c = 0; c < AttrDim; c++)
  PointAttr[PtIdx][c] = DivExp2Up(PointAttr[PtIdx][c], 8)
```

— Transform coefficient components are predicted using inter-component prediction (10.6.10.2) to form prediction residuals.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

— The attribute value is predicted and combined with the prediction residual (10.6.12).

— The reconstructed attribute value is clipped.

```
for (c = 0; c < AttrDim; c++)
  PointAttr[PtIdx][c] = Clip3(0, AttrMaxVal, PointAttr[PtIdx][c])
```

10.6.7.4 Processing a detail level (attr_coding_type = 2)

When attr_coding_type is 2, the following operations shall be performed in turn, each over all the coefficients in the detail level:

— Coefficient tuples are scaled (10.6.9.1) to produce transform coefficients.

— Transform coefficient components are predicted using last-component prediction (10.6.10.3).

— Transform coefficients are weighted by transform coefficient weights (10.6.11.4).

If *Lvl* is less than *LodCnt* − 1, the transform shall be applied (10.6.12):

— Attribute values predicted from the coarser detail level, *Lvl* + 1, are modified by the transform update operator (10.6.12.1).

— Attribute values corresponding to coefficients in the current detail level are predicted and combined with the scaled transform coefficient to produce the detail level output (10.6.12.1).

When *Lvl* is 0, 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 == 0)
  for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)
    for (c = 0; c < AttrDim; c++)
      PointAttr[ptIdx][c] = Clip3(0, AttrMaxVal, DivExp2Fz(PointAttr[ptIdx][c], 8))
```

10.6.8 Prediction mode coding

10.6.8.1 General

This subclause specifies the conditional coding of the prediction mode *PredMode* in coefficient tuples. It applies when pred_direct_max_idx_plus1 is greater than zero; when pred_direct_max_idx_plus1 is 0, *PredMode* shall be 0.

A per-transform-coefficient test (10.6.8.2) shall be performed to determine whether the coefficient tuple encodes a prediction mode. The result of the test is the variable *PredModePresent*.

If *PredModePresent* is:

— false, *PredMode* shall be 0;

— true, the coded prediction mode (*PredModeCoded*) shall be decoded according to the number of attribute components (10.6.8.3, 10.6.8.4) and the maximum codable prediction mode *PredModeMax*. As a side-effect of decoding the prediction mode, the coefficient tuple (in *AttrCoeff*) is updated.

```
PredModeMax := pred_direct_max_idx_plus1 + ¬pred_direct_avg_disabled
```

The prediction mode shall be derived from the coded prediction mode:

```
PredMode = PredModePresent ? PredModeCoded + pred_direct_avg_disabled : 0
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

10.6.8.2 Presence of an encoded direct prediction mode

The derivation of *PredModePresent* specifies the presence of an encoded direct prediction mode:

— A direct prediction mode shall not be coded when disabled, or for refinement points with fewer than two predictors.

```
if (PredCnt[PtIdx] < 2 || pred_direct_max_idx_plus1 == 0)
  PredModePresent = 0
```

— Otherwise, a prediction mode shall be coded for a refinement point if, for any component, the absolute difference in attribute value between any of its predictors exceeds a bit-depth adjusted threshold.

```
for (ni = 0; ni < PredCnt[PtIdx]; ni++) {
  ptIdx  = PredPtIdx[PtIdx][ni]
  for (c = 0; c < AttrDim; c++) {
    minVal[c] = ni ? Min(minVal[c], PointAttr[ptIdx][c]) : PointAttr[ptIdx][c]
    maxVal[c] = ni ? Max(maxVal[c], PointAttr[ptIdx][c]) : PointAttr[ptIdx][c]
  }
}
maxDiff = 0
for (c = 0; c < AttrDim; c++)
  maxDiff = Max(maxDiff, maxVal[c] - minVal[c])

threshold = pred_direct_threshold << Max(0, AttrBitDepth - 8)
PredModePresent = maxDiff ≥ threshold
```

10.6.8.3 Decoding process for single-component attributes

For single-component attributes (*AttrDim* == 1), the prediction mode *PredModeCoded* is encoded by the LSBs of the coefficient magnitude:

```
PredModeCoded = 0
absCoeff = Abs(AttrCoeff[CoeffIdx][0])
if (PredModeMax == 4){
  PredModeCoded = absCoeff & 3
  absCoeff >>= 2
}

if (PredModeMax == 3) {
  PredModeCoded = absCoeff & 1
  absCoeff >>= 1
  if (PredModeCoded){
    PredModeCoded += absCoeff & 1
    absCoeff >>= 1
  }
}
if (PredModeMax == 2){
  PredModeCoded = absCoeff & 1
  absCoeff >>= 1
}
```

After decoding the prediction mode, the coefficients shall be updated.

```
AttrCoeff[CoeffIdx][0] = Sign(AttrCoeff[CoeffIdx][0]) × absCoeff
```

10.6.8.4 Decoding process for multi-component attributes

For multi-component attributes (*AttrDim* > 1), the prediction mode *PredModeCoded* is encoded by the LSB of the last two component's coefficient magnitude:

```
PredModeCoded = 0
absCoeffA = Abs(AttrCoeff[CoeffIdx][AttrDim - 2])
absCoeffB = Abs(AttrCoeff[CoeffIdx][AttrDim - 1])

if (PredModeMax == 4) {
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
  PredModeCoded = ((absCoeffA & 1) << 1) + (absCoeffB & 1)
  absCoeffA >>= 1
  absCoeffB >>= 1
}
if (PredModeMax == 3) {
  PredModeCoded = absCoeffA & 1
  absCoeffA >>= 1
  if (PredModeCoded) {
    PredModeCoded += absCoeffB & 1
    absCoeffB >>= 1
  }
if (PredModeMax == 2) {
  PredModeCoded = absCoeffA & 1
  absCoeffA >>= 1
}
```

After decoding the prediction mode, the coefficients shall be updated.

```
sgnCoeffA = Sign(AttrCoeff[CoeffIdx][AttrDim − 2])
sgnCoeffB = Sign(AttrCoeff[CoeffIdx][AttrDim − 1])

AttrCoeff[CoeffIdx][AttrDim − 2] = sgnCoeffA × absCoeffA
AttrCoeff[CoeffIdx][AttrDim − 1] = sgnCoeffB × absCoeffB
```

10.6.9 Scaling

10.6.9.1 Derivation of per-point QP

The QP for a point depends upon the detail level and its attribute coordinates as specified by the expression *LodCoeffQp*[*qc*] for a QP component *qc*:

— *rgnOffset*[*qc*] is the per-coefficient offset from the region-dependent QP offset tree.

— *dpth* is the depth of detail level in the LoD hierarchy.

```
LodCoeffQp[qc] := AttrQp[dpth][rgnOffset][Cidx > 0]
  where
    s := AttrPos[PtIdx][0]
    t := AttrPos[PtIdx][1]
    v := AttrPos[PtIdx][2]
    dpth := LodCnt − 1 − Lvl
    rgnOffset[qc] := AttrRegionQpOffset[s][t][v][qc]
```

10.6.9.2 Scaling by quantization step size

The coefficient tuple shall be scaled by the quantization step size (10.7.4) for the primary and secondary attribute components.

```
for (c = 0; c < AttrDim; c++)
  PointAttr[PtIdx][c] = AttrCoeff[CoeffIdx][c] × AttrQstep[LodCoeffQp[c > 0]]
```

10.6.10 Coefficient prediction

10.6.10.1 Syntax element semantics

last_comp_pred_coeff_diff[*dpth*] specifies in accordance with *LastCompPredCoeff*[*dpth*] the two-fractional-bit, fixed-point scale factor applied at depth *dpth* of the LoD hierarchy to second coefficient components to predict third coefficient components. The syntax element codes the scale factor relative to *LastCompPredCoeffPrev*[*dpth*].

```
LastCompPredCoeff[dpth] := last_comp_pred_enabled
  ? LastCompPredCoeffPrev[dpth] + last_comp_pred_coeff_diff[dpth]
  : 0
LastCompPredCoeffPrev[dpth] := dpth == 0 ? 4 : LastCompPredCoeff[dpth − 1]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

It is a requirement of bitstream conformance that *LastCompPredCoeff*[*dpth*] shall be in the range –128 .. 127 for *dpth* ∈ 0 .. lod_max_levels_minus1.

inter_comp_pred_coeff_diff[*dpth*][*c*] specifies in accordance with *InterCompPredCoeff*[*dpth*][*c*] the two-fractional-bit, fixed-point scale factor applied at depth *dpth* of the LoD hierarchy to first coefficient components to predict *c*-th coefficient components. The syntax element codes the scale factor relative to *InterCompPredCoeffPrev*[*dpth*][*c*].

```
InterCompPredCoeff[dpth][c] := inter_comp_pred_enabled
  ? predCoeff + inter_comp_pred_coeff_diff[dpth][c]
  : 0

InterCompPredCoeffPrev[dpth][c] := dpth == 0 ? 4 : InterCompPredCoeff[dpth – 1][c]
```

It is a requirement of bitstream conformance that *InterCompPredCoeff*[*dpth*][*c*] shall be in the range –128 .. 127 for *dpth* ∈ 0 .. lod_max_levels_minus1.

10.6.10.2 Inter-component prediction

When attr_coding_type is 1 and inter_comp_pred_enabled is 1, secondary attribute coefficient components are residuals to a prediction by the first scaled coefficient component. The predicted value shall round the two fractional bits from the scale factor, with half-values rounded up.

```
for (c = 1; c < AttrDim; c++) {
  icpCoeff = InterCompPredCoeff[LodCnt – 1 – Lvl][c]
  PointAttr[PtIdx][c] += DivExp2Up(icpCoeff × PointAttr[PtIdx][0], 2)
}
```

10.6.10.3 Last-component prediction

When attr_coding_type is 2 and last_comp_pred_enabled is 1, the third attribute coefficient component is, if present, a residual to a prediction by the second scaled coefficient component. The predicted value shall round the two fractional bits from the scale factor towards negative infinity.

```
if (AttrDim == 3) {
  lcpCoeff = LastCompPredCoeff[LodCnt – 1 – Lvl]
  PointAttr[PtIdx][2] += DivExp2Floor(lcpCoeff × PointAttr[PtIdx][1], 2)
}
```

10.6.11 Transform coefficient weights

10.6.11.1 General

Coefficient weights represent the relative significance of a coefficient. Coefficients with larger weights have a greater influence on the decoded attribute values.

The array *CoeffWeight* is initialized by setting all elements to 256.

```
for (i = 0; i < PointCnt; i++)
  CoeffWeight[i] = 256
```

The derivation of coefficient weights depends upon whether LoD scalability is enabled.

10.6.11.2 Non-scalable case

When lod_scalability_enabled is 0, coefficient weights are calculated accumulatively, proceeding from the finest to the coarsest detail level.

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 = 0; lvl < LodCnt – 1; lvl++)
  for (rfmtIdx = 0; rfmtIdx < LodRfmtPtCnt[lvl]; rfmtIdx++) {
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

```
    ptIdx = LodRfmtPtIdx[lvl][rfmtIdx]
    coeffW = CoeffWeight[ptIdx]
    for (ni = 0; ni < PredCnt[ptIdx]; ni++) {
      predW = PredWeight[ptIdx][ni]
      CoeffWeight[PredPtIdx[ptIdx][ni]] += DivExp2Up(coeffW × predW, 8)
    }
  }
```

10.6.11.3 Scalable case

When lod_scalability_enabled is 1, a single weight shall be assigned to all refinement points within a detail level:

```
for (lvl = 1; lvl < LodCnt - 1; lvl++) {
  weight = (slice_num_points_minus1 + 1) / LodPtCnt[lvl]
  for (rfmtIdx = 0; rfmtIdx < LodRfmtPtCnt[lvl]; rfmtIdx++)
    CoeffWeight[LodRfmtPtIdx[lvl][rfmtIdx]] = weight × 256
}
```

10.6.11.4 Application to coefficient scaling

Transform coefficients shall be scaled by the integer reciprocal square root of their coefficient weight and divided by 2^{36} with half-values rounded away from zero.

```
weight = IntRecipSqrt(CoeffWeight[PtIdx])
for (c = 0; c < AttrDim; c++)
  PointAttr[PtIdx][c] = DivExp2Fz(PointAttr[PtIdx][c] × weight, 36)
```

10.6.12 Transform

10.6.12.1 Update operation

When attr_coding_type is 2, the transform update operator shall redistribute coefficient values to predicting points in the coarser detail level.

```
for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)
  updateN[ptIdx] = updateD[ptIdx] = 0
for (rfmtIdx = 0; rfmtIdx < LodRfmtPtCnt[Lvl]; rfmtIdx++) {
  rfmtPtIdx = LodRfmtPtIdx[Lvl][rfmtIdx]
  coeffW = CoeffWeight[rfmtPtIdx]
  for (ni = 0; ni < PredCnt[rfmtPtIdx]; ni++) {
    nPtIdx = PredPtIdx[rfmtPtIdx][ni]
    nWeight = DivExp2Up(PredWeight[rfmtPtIdx][ni] × coeffW, 8)
    updateD[nPtIdx] += nWeight
    for (c = 0; c < AttrDim; c++)
      updateN[nPtIdx][c] += nWeight × PointAttr[rfmtPtIdx][c]
  }
}
for (ptIdx = 0; ptIdx < PointCnt; ptIdx++)
  if (updateD[ptIdx])
    PointAttr[ptIdx] -= Div(updateN[ptIdx], updateD[ptIdx], 0)
```

10.6.12.2 Direct prediction

When attr_coding_type is 1 and *PredMode* for a refinement point is greater than zero, its value shall be predicted to be the same as the point with predictor set index *PredMode* − 1. If the indicated predictor is invalid, prediction shall not be performed.

It is a requirement of bitstream conformance that *PredMode* shall be less than or equal to *PredCnt*[*CoeffIdx*].

```
if (PredMode && PredMode ≤ PredCnt[PtIdx])
  for (c = 0; c < AttrDim; c++)
    PointAttr[PtIdx][c] += PointAttr[PredPtIdx[PtIdx][PredMode - 1]][c]
```

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved

10.6.12.3 Average prediction

When attr_coding_type is 2 or *PredMode* for a refinement point is 0, the weighted average of the predictor set shall predict the value of the refinement point:

```
if (PredMode == 0)
  for (c = 0; c < AttrDim; c++) {
    sum = 0
    for (ni = 0; ni < PredCnt[PtIdx]; ni++)
      sum += PredWeight[PtIdx][ni] × PointAttr[PredPtIdx[PtIdx][ni]][c]
    PointAttr[PtIdx][c] += DivExp2Fz(sum, 8)
  }
```

10.7 Attribute quantization parameters

10.7.1 Syntax element semantics

attr_qp_offset[*qc*] specifies per-slice offsets used to derive QPs for the primary (*qc* = 0) and any secondary (*qc* = 1) attribute components. When attr_qp_offset[*qc*] is not present, it shall be inferred to be 0.

attr_qp_layers_present specifies whether (when 1) or not (when 0) per-transform-layer QP offsets are present in the ADU.

attr_qp_layer_cnt_minus1 plus 1 specifies, when present, the number of levels in the LoD hierarchy or RAHT tree for which QP offsets are signalled.

attr_qp_layer_offset[*dpth*][*qc*] specifies QP offsets used for the primary (*qc* = 0) and any secondary (*qc* = 1) attribute components. Each offset applies to transform coefficients at depth *dpth* of the LoD hierarchy or RAHT tree. If the LoD hierarchy or RAHT tree has a greater number of levels than attr_qp_layer_cnt_minus1 + 1, attr_qp_layer_offset[attr_qp_layer_cnt_minus1][*qc*] also specifies the QP offsets for transform coefficients at a depth greater than attr_qp_layer_cnt_minus1.

The expression *AttrQpLayerOffset*[*dpth*][*qc*] specifies the per layer QP offsets at depth *dpth* of the LoD hierarchy or RAHT tree.

```
AttrQpLayerOffset[dpth][qc] =
  ? attr_qp_layers_present > 0
  ? attr_qp_layer_offset[Min(attr_qp_layer_cnt_minus1, dpth)][qc]
  : 0
```

attr_qp_region_cnt specifies the number of spatial regions within the slice that have a region QP offset signalled.

NOTE In profiles specified in this version of this document, all but the first region are ignored.

attr_qp_region_bits_minus1 plus 1 specifies the length in bits of each syntax element attr_qp_region_origin_xyz, attr_qp_region_size_minus1_xyz, attr_qp_region_origin_rpi and attr_qp_region_size_minus1_rpi.

attr_qp_region_origin_xyz[*i*][*k*] and **attr_qp_region_size_minus1_xyz**[*i*][*k*] specify, when present, the *i*-th spatial region in the slice where attr_qp_region_offset[*i*][*qc*] applies. The region is a bounding box in the slice coordinate system with lower corner XYZ coordinates attr_qp_region_origin_xyz[*i*][*k*] and dimensions attr_qp_region_size_minus1_xyz[*i*][*k*] + 1.

attr_qp_region_origin_rpi[*i*][*k*] and **attr_qp_region_size_minus1_rpi**[*i*][*k*] specify, when present, the *i*-th spatial region in the slice where attr_qp_region_offset[*i*][*qc*] applies. The region is a bounding box in the scaled angular coordinate system (10.2.2) used for attribute coding with lower corner RPI coordinates attr_qp_region_origin_rpi[*i*][*k*] and dimensions attr_qp_region_size_minus1_rpi[*i*][*k*] + 1.

The expressions *AttrRegionQpOrigin*[*i*][*k*] and *AttrRegionQpSize*[*i*][*k*] specify the *k*-th component of the bounding box origin and size for the *i*-th QP region in attribute coordinates.

IECNORM.COM : Click to view the full PDF of ISO/IEC 23090-9:2023

© ISO/IEC 2023 – All rights reserved