ICS 35.040
CCS L 71

团　　体　　标　　准

T/AI 128.2-2024

信息技术 时空图形数据编码 第2部分：点云

Information Technology Space-Time Graphical Data Coding

Part 2: Point Cloud

2024－10－14 发布　　2024－10－14 实施

中关村视听产业技术创新联盟　发布

目　　次

前　言

本文件按照GB/T 1.1—2020 《标准化工作导则 第1部分：标准化文件的结构和起草规则》的规定起草。

本文件为T/AI 128《信息技术 时空图形数据编码》的第2部分。T/AI 128已发布了以下部分：

——第2部分：点云。

本文件由数字音视频编解码技术标准工作组提出。

本文件由中关村视听产业技术创新联盟归口。

本文件起草单位：鹏城实验室、腾讯科技（深圳）有限公司、北京大学深圳研究生院、深圳市大疆创新科技有限公司、西安电子科技大学、OPPO 广东移动通信有限公司、中山大学、浙江大学、北京大学、上海交通大学、电子科技大学、武汉大学、阿里巴巴集团高德软件有限公司、中兴通讯股份有限公司、维沃移动通信有限公司、西北工业大学、南京大学、广东博华超高清创新中心有限公司、中国科学院大学、咪咕文化科技有限公司、中国科学院计算技术研究所、清华大学深圳国际研究生院、深圳佑驾创新科技股份有限公司、中国移动通信集团有限公司。

本文件起草人：高文、黄铁军、李革、刘杉、郑萧桢、王静、高文（腾讯）、张伟（西电）、朱文婕、张琦、王东、梁凡、虞露、马思伟、陈悦汝、魏红莲、李璞、余越、徐异凌、邵薏婷、孙泽星、李鼎权、胡颖、赵文博、马闯、王文义、宋菲、邹春雨、陈建文、陈震中、张悦、秦泰、杨付正、安禹豪、邓江伟、许晓中、高伟、高莹、吕卓逸、万帅、王贵旗，田腾亚，鲁静芸、杨丽慧、刘晓宇、马展、赵海英、张伟民、张伟（鹏城实验室）、何坚、刘祎、黄锐珊、陈嘉枫、金佳民、何盈燊、张新峰、李琳、王苫社、冯亚楠、尹茜、徐逸群、于浩平、谢绍伟、黄成、任青山、魏志文、赵丽丽、章海华、张雨航、郭宇、侯礼志、王剑强、金欣、郑伟。

引　言

海量点云数据的高效存储、传输、发布、共享和标准化，是点云应用的关键。点云空间分布的无规则对点云数据的压缩提出了挑战，并且由于点云压缩标准的缺失导致各类点云设备无法实现有效的互联互通。本标准旨在面向自动驾驶、兼顾数字博物馆及虚拟现实等行业应用中的点云数据，提供高效的压缩表示，支持高精度、海量点数的点云数据高效编码，定义高效的点云解码过程和解码规范。T/AI XXX《信息技术 时空图形数据编码》标准拟由四部分组成：

——第 1 部分：数据传输。规定了点云/网格编码数据的文件封装格式、传输格式与信令格式。

——第 2 部分：点云。规定了高效的点云解码过程和解码规范。

——第 3 部分：质量评价。规定了点云/网格数据质量评价规范。

——第 4 部分：网格。规定了高效的网格解码过程和解码规范。

本文件的发布机构提请注意，声明符合本文件时，可能涉及到：9.3.6 与《一种基于邻居权重的参数选取和传递的点云属性编码和解码的方法及设备》（专利申请号：CN201910942969.3）；9.3.2 与《对三维数据点集进行编码或解码的方法和装置》（专利申请号：CN201980005064.1）；9.2.2 、9.2.3 与《用于三维数据点集处理的方法和设备》（专利申请号：CN201980005156.X）；9.2.2 与《三维数据点的编解码方法和装置》（专利申请号：CN201980005594.6）；9.2.2 与《一种点云编码方法、点云解码方法及相关设备》（专利申请号：CN201980007958.4）；9.2.2 、9.2.3 与《一种用于点云处理、解码的方法、设备及存储介质》（专利申请号：CN201980012174.0）；9.2.2 、9.2.3 与《三维数据点的编解码方法和装置》（专利申请号：CN201980012178.9）；8.3.3 与《点云属性编码方法和装置及点云属性解码方法和装置》（专利申请号：CN201980039255.X）；9.2.3 与《一种用于点云处理、解码的方法、设备及存储介质》（专利申请号：CN201980094938.5）；7.1.2.6 、7.1.2.7 、7.1.2.8 、7.1.2.9 、7.1.2.10 与《一种数据编码、数据解码的方法、设备及存储介质》（专利申请号：CN201980094966.7）；9.2.3 与《一种点云几何编码方法、解码方法、编码设备及解码设备》（专利申请号：CN202010238176.6）；9.3.13 与《一种点云属性编码方法、解码方法、编码设备及解码设备》（专利申请号：CN202010967090.7）；9.2.3 与《一种用于点云编码、解码的方法和设备》（专利申请号：CN202080004086.9）；9.2.3 与《点云的编解码方法和装置》（专利申请号：CN202080004105.8）；9.2.3 与《点云的编解码方法和装置》（专利申请号：CN202080005108.3）；9.3.3 与《点云编码和解码方法、装置及计算机可读存储介质》（专利申请号：CN202080081330.1）；9.3.3 与《点云编码方法、点云解码方法、装置及存储介质》（专利申请号：CN202080081334.X）；9.3.14 与《点云属性编码方法、装置、解码方法、装置及相关设备》（专利申请号：CN202110264409.4）；9.3.10 与《一种点云的属性熵编码和熵解码的方法及设备》（专利申请号：CN202110269745.8）；9.3.5 与《点云邻居确定、点云预测、点云编码、点云解码方法及设备》（专利申请号：CN202110277920.8）；7.1.2.6 、7.1.2.9 、7.2.3 、7.2.6 与《点云编码处理方法、解码处理方法及装置》（专利申请号：CN202110656048.8）；7.2.3 、7.2.6 与《熵编码、解码方法及装置》（专利申请号：CN202110656066.6）；9.3.14 与《点云属性编码方法、装置、解码方法、装置及相关设备》（专利申请号：CN202110656783.9）；9.2.3 与《一种点云孤立点编码方法、解码方法及装置》（专利申请号：CN202110657395.2）；9.3.9 与《一种点云属性预测方法、装置、终端及存储介质》（专利申请号：CN202110657405.2）；7.2.8 、9.2.6 与《点云编码方法、解码方法及装置》（专利申请号：CN202110941633.2）；9.3.13 与《点云属性编码方法、装置、解码方法以及装置》（专利申请号：CN202110969710.5）；9.2.6 与《点云编码方法、解码方法、点云编码设备及解码设备》（专利申请号：CN202110975997.2）；9.3.7 、9.3.13 与《点云预测、点云编码、点云解码方法及设备》（专

利申请号：CN202110976025.5）；7.2.7 、9.3.11 与《属性量化、反量化方法、装置及设备》（专利申请号：CN202111465429.4）；9.3.5 、9.3.6 、9.3.9 、9.3.10 与《点云属性编码方法、点云属性解码方法及终端》（专利申请号：CN202111465642.5）；9.3.5 、9.3.6 与《一种点云属性编码方法、解码方法、编码设备及解码设备》（专利申请号：CN202111482269.4）；9.2.3 与《点云几何解码方法、装置、计算机可读存储介质》（专利申请号：CN202180081867.2）；7.1.2.7 与《点云属性解码方法和点云属性编码方法》（专利申请号：CN202180083707.1）；9.3.14 与《一种点云属性编码方法、点云属性解码方法及存储介质》（专利申请号：CN202210699502.2）；9.2.6 与《基于预测树的点云几何编码、解码方法及设备》（专利申请号：CN202211596014.5）；9.2.3 与《点云几何编码方法、解码方法、编码设备及解码设备》（专利申请号：CN202211604777.X）；7.2.4 、7.1.4 与《点云属性编码解码方法、装置、电子设备及存储介质》（专利申请号：CN202311129407.X）；9.3.2 与《对三维数据点集进行编码或解码的方法和装置》（专利申请号：PCT/CN2019/071238）；9.2.2 、9.2.3 与《用于三维数据点集处理的方法和设备》（专利申请号：PCT/CN2019/071240）；9.2.2 与《三维数据点的编解码方法和装置》（专利申请号：PCT/CN2019/071494）；8.3.3 与《点云属性编码方法和装置及点云属性解码方法和装置》（专利申请号：PCT/CN2019/079150）；9.2.2 与《一种点云编码方法、点云解码方法及相关设备》（专利申请号：PCT/CN2019/091097）；9.2.2 、9.2.3 与《三维数据点的编解码方法和装置》(专利申请号:PCT/CN2019/091351);7.1.2.6 、7.1.2.7 、7.1.2.8 、7.1.2.9 、7.1.2.10 与《一种数据编码、数据解码的方法、设备及存储介质》（专利申请号：PCT/CN2019/122209）；9.2.3 与《一种用于点云处理、解码的方法、设备及存储介质》(专利申请号：PCT/CN2019/123821）；9.2.2 、9.2.3 与《一种用于点云处理、解码的方法、设备及存储介质》（专利申请号：PCT/CN2019/126090）；9.2.3 与《一种用于点云编码、解码的方法和设备》（专利申请号：PCT/CN2020/082593）；9.2.3 与《点云的编解码方法和装置》（专利申请号：PCT/CN2020/082608）；9.2.3 与《点云的编解码方法和装置》（专利申请号：PCT/CN2020/093754）；9.3.3 与《点云编码方法、点云解码方法、装置及存储介质》（专利申请号：PCT/CN2020/131246）；9.3.3 与《点云编码和解码方法、装置及计算机可读存储介质》（专利申请号：PCT/CN2020/134355）；9.2.3 与《点云几何解码方法、装置、计算机可读存储介质》（专利申请号：PCT/CN2021/073422）；7.1.2.7 与《点云属性解码方法和点云属性编码方法》（专利申请号：PCT/CN2021/080539）；7.2.4 、9.3.5 、9.3.9 与《点云属性的预测方法、装置及编解码器》(专利申请号：PCT/CN2021/099872)；7.1.4 、7.2.9 、9.3.6 、9.3.9 、9.3.10 、9.3.11 、9.3.12 与《解码方法、编码方法、解码器以及编码器》(专利申请号：PCT/CN2021/135529)；7.1.4 、7.2.9 、8.1 、8.3.4 、9.3.10 与《解码方法、编码方法、解码器以及编码器》（专利申请号：PCT/CN2022/099635）；9.2.3 与《点云编码解码方法、装置、设备及存储介质》（专利申请号：PCT/CN2022/122117）；9.3.10 与《点云编解码方法、编码器、解码器、码流及存储介质》(专利申请号：PCT/CN2022/132330)；8.3.3 与《编解码方法、解码器、编码器、码流及存储介质》（专利申请号：PCT/CN2022/138185）；9.3.10 与《点云编解码方法、编码器、解码器、码流及存储介质》（专利申请号：PCT/CN2023/071279）；7.1.2.7 、7.2.4 与《解码方法、编码方法、解码器以及编码器》（专利申请号：PCT/CN2023/077410）；7.1.2.9 、7.2.6 与《编解码方法、码流、编码器、解码器以及存储介质》（专利申请号：PCT/CN2023/077451）；7.1.2 与《点云解码和编码方法及设备、系统、存储介质、码流》(专利申请号：PCT/CN2023/077494)；7.1.4 、7.2.9 、8.3.3 与《点云的编解码方法、码流、编码器、解码器以及存储介质》（专利申请号：PCT/CN2023/086334）；7.1.4 、7.2.9 、8.3.3 与《点云的编解码方法、码流、编码器、解码器以及存储介质》（专利申请号：PCT/CN2023/086337）；7.1.2.6 与《编解码方法、码流、编码器、解码器以及存储介质》（专利申请号：PCT/CN2023/113792）；7.2.4 与《编解码方法、编解码器以及存储介质》（专利申请号：PCT/CN2023/113831）；9.2.5 与《点云编解码方法、码流、编码器、解码器以及存储介质》(专利申请号：PCT/CN2023/113835)；9.3.8 与《编解码方法、编解码器以及存储介质》

（专利申请号：PCT/CN2023/113839）；9.3.5 、9.3.6 与《编解码方法、编解码器以及存储介质》（专利申请号：PCT/CN2023/113845）；7.1.2.7 、7.1.4 、7.2.4 与《编解码方法、码流、编码器、解码器以及存储介质》（专利申请号：PCT/CN2023/114533）。

本文件的发布机构对于该专利的真实性、有效性和范围无任何立场。

该专利持有人已向本文件的发布机构承诺，他愿意同任何申请人在合理且无歧视的条款和条件下，就专利授权许可进行谈判。该专利持有人的声明已在本文件的发布机构备案，相关信息可以通过以下联系方式获得：

专利持有人：鹏城实验室，地址：广东省深圳市南山区兴科一街 2 号；腾讯科技（深圳）有限公司，地址：深圳市南山区海天二路腾讯滨海大厦；北京大学深圳研究生院，地址：广东省深圳市南山区西丽深圳大学城北大园区 H 栋 208 室；深圳市大疆创新科技有限公司。地址：深圳市南山区西丽街道西丽社区仙元路 53 号大疆天空之城 T2-17 楼；OPPO 广东移动通信有限公司，地址：广东省深圳市南山区科苑南路 2666 号中国华润大厦 39 楼；维沃移动通信有限公司，地址：广东省东莞市长安镇维沃路 1 号。

联 系 人：黄铁军（数字音视频编解码技术标准工作组秘书长）

通讯地址：北京大学理科 2 号楼 2641 室

邮政编码：100871

电子邮件：tjhuang@pku.edu.cn

电　　话：+8610-62756172

传　　真：+8610-62751638

网　　址：http://www.avs.org.cn

请注意除上述专利外，本文件的某些内容仍可能涉及专利。本文件的发布机构不承担识别这些专利的责任。

信息技术 时空图形数据编码 第2部分：点云

1 范围

本文件规定了点云编码位流的结构、位流的语法和语义、解析过程及解码过程。

本文件适用于自主导航系统、实时巡检系统、地理信息系统、数字文化遗产、自由视点广播、三维沉浸通信和三维沉浸交互等应用。

2 规范性引用文件

下列文件中的内容通过文中的规范性引用而构成本文件必不可少的条款。其中，注日期的引用文件，仅该日期对应的版本适用于本文件；不注日期的引用文件，其最新版本（包括所有的修改单）适用于本文件。

IEEE Std 754™-2019 IEEE Standard for Floating-Point Arithmetic

ISO/IEC 9899:2018 Information technology — Programming languages — C

3 术语和定义

下列术语和定义适用于本文件。

3.1

八叉树 octree

树中的任一节点包含零个或者至多八个子节点，用于表示点云的几何位置。

3.2

保留 reserved

定义了一些特定语法元素值，这些值用于将来对本文件的扩展。

这些值不应出现在符合本文件的位流中。

3.3

包围盒 bounding box

轴对齐方向包含一组点的长方体。

3.4

残差 residual

样本或数据元素的重建值与其预测值之差。

3.5

档次 profile

本文件规定的语法、语义及算法的子集。

3.6

笛卡尔坐标 Cartesian coordinate

有限精度和动态范围的三个标量(x, y, z)，其中动态范围的三个标量 x、y 和 z 为三个非负整数，用于表示一个点在笛卡尔直角坐标系下的位置。

3.7

点 point

点云的基本元素，其位置表示为三维空间笛卡尔坐标(x, y, z)，并包含零个或多个属性，比如颜

色和反射率等。

3.8

点云 point cloud

一组无序的点的集合，表达三维物体或场景的空间结构及表面属性。

3.9

点云帧 point cloud frame

某个时刻的点云。

3.10

二元符号 bin

值的二进制化表示的二进制符号，包括 0 和 1。

3.11

二元符号串 bin string

有限位二元符号组成的有序序列，最左边符号是最高有效位，最右边符号是最低有效位。

3.12

反变换 inverse transform

将变换系数矩阵转换成空域样值矩阵的过程。

3.13

反量化 dequantization

对量化值乘以量化步长缩放后得到乘积值的过程。

3.14

反射率 reflectance

表示点对光的反射强度的一维信号。

3.15

父节点 parent node

树中当前节点靠近根节点方向的上一层节点。一个父节点包含多个节点。

3.16

根节点 root node

树中没有父节点的节点。

3.17

级别 level

在某一档次下对语法元素和语法元素参数值的限定集合。

3.18

几何 geometry

点云中的点的位置，表示为三维空间中的笛卡尔坐标。

3.19

节点 node

树中的元素，表示包含整个点云的空间或者一个子空间。

3.20

解析过程 parse

由位流获得语法元素的过程。

3.21

禁止 forbidden

定义了一些特定语法元素值。禁止某些值的目的通常是为了避免在位流中出现伪起始码。

3.22

亮度 luma

Y 表示亮度信号的样值矩阵或单个样值。

3.23

莫顿码 Morton code

根据莫顿序将三个 d-bit 的非负整数的位交错放置后形成的 3d-bit 的非负整数。

3.24

片 slice

一系列的语法元素，用于表示一部分或者整个点云。

3.25

色度 chroma

用于表示亮度和红色、蓝色的差值的信号。

3.26

属性 attribute

点云中的点对应的标量特征，比如反射率，或颜色，或矢量特征。

3.27

填充位 stuffing bits

编码时插入位流中的位串，在解码时被丢弃。

3.28

位串 bit string

有限个二进制位的有序序列，其最左边位是最高有效位（MSB），最右边位是最低有效位（LSB）。

3.29

位流 bitstream

编码点云所形成的二进制数据流。

3.30

希尔伯特码 Hilbert code

根据希尔伯特序将三个 d-bit 的非负整数的位交错放置后形成的 3d-bit 的非负整数。

3.31

颜色 color

点对应的三维 RGB 或 YUV 信号。

3.32

预测 prediction

利用先前已解码的样值或数据元素的组合得到估计值的过程。

3.33

预测补偿 prediction compensation

计算由语法元素解码得到的样本残差与其对应的预测值之和的过程。

3.34

预测值 prediction value

在样值或数据元素的解码过程中，经过预测得到的估计值。

3.35

语法元素 syntax element

位流中的数据单元解析后的结果。

3.36

占位码 occupancy code

用于表示一个节点的子节点占用情况的一个字节，其中每个位的值对应该节点的一个子节点的占用情况。

3.37

字节 byte

8 位的位串。

3.38

子节点 child node

树中当前节点的远离根节点方向的下一层节点。一个当前节点包含至多 8 个子节点。

3.39

字节对齐 byte alignment

从位流的第一个二进制位开始，某二进制位的位置是 8 的整数倍。

4 缩略语

下列缩略语适用于本文件。

AEC	高级熵编码（advanced entropy code）
APS	属性参数集（attribute parameter set）
ASH	属性片头（attribute slice header）
attr	属性（attribute）
ctx	二元符号模型（context）
geom	几何（geometry）
GPS	几何参数集（geometry parameter set）
GSH	几何片头（geometry slice header）
idx	索引（index）
LCU	最大编码单元（largest coding unit）
LSB	最低有效位（least significant bit）
MSB	最高有效位（most significant bit）
QP	量化参数（quantization parameter）
QTBT	四叉树和二叉树（quadtree and binary tree）
refl	反射率（reflectance）
SPS	序列参数集（sequence parameter set）

5 约定

5.1 概述

本文件中使用的数学运算符和优先级参照 C 语言的约定， 符合 ISO/IEC 9899:2018 中的规定。但对整型除法和算术移位操作进行了特定定义。除特别说明外，约定编号和计数从 0 开始。

5.2 算术运算符

算术运算符定义见表 1。

表1　算术运算符定义

算术运算符	定义
+	加法运算
-	减法运算（作为双参数运算符）或取反（作为单参数前缀运算符）
*	乘法运算
a^b	幂运算，表示 a 的 b 次幂。也可表示上标
a^b	幂运算，表示 a 的 b 次幂
/	整除运算，沿向 0 的取值方向截断。例如，7/4 和-7/-4 截断至 1，-7/4 和 7/-4 截断至-1
$\frac{a}{b}$	除法运算，不做截断或四舍五入
$\sum_{i=a}^{b} f(i)$	自变量 i 取由 a 到 b（含 b）的所有整数值时，函数f(i)的累加和
$\prod_{i=a}^{b} f(i)$	自变量 i 取由 a 到 b（含 b）的所有整数值时，函数f(i)的累乘积
a % b	模运算，a 除以 b 的余数，其中 a 与 b 都是正整数
⌈.⌉	上取整
⌊.⌋	下取整
|.|	取绝对值

5.3　逻辑运算符

逻辑运算符定义见表 2。

表2　逻辑运算符定义

逻辑运算符	定义
a && b	a 和 b 之间的与逻辑运算
a || b	a 和 b 之间的或逻辑运算
!	逻辑非运算
a ? b : c	如果 a 为真或不为 0，则使用 b 进行赋值；否则，使用 c 进行赋值

5.4　关系运算符

关系运算符定义见表 3。

表3　关系运算符定义

关系运算符	定义
>	大于
>=	大于或等于
<	小于
<=	小于或等于
==	等于
!=	不等于

5.5 位运算符

位运算符定义见表 4。

表4 位运算符定义

位运算符	定义
&	与运算
|	或运算
a >> b	将 a 以 2 的补码整数表示的形式向右移 b 位。仅当 b 取正数时定义此运算
a << b	将 a 以 2 的补码整数表示的形式向左移 b 位。仅当 b 取正数时定义此运算

5.6 赋值

赋值运算定义见表 5。

表5 赋值运算定义

赋值运算	定义
=	赋值运算符
++	递增，x++相当于 x = x + 1。当用于数组下标时，在递增运算前先求变量值
--	递减，x--相当于 x = x - 1。当用于数组下标时，在递减运算前先求变量值
+=	自加指定值，例如 x += 3 相当于 x = x + 3，x += (-3)相当于 x = x + (-3)
-=	自减指定值，例如 x -= 3 相当于 x = x - 3,，x -= (-3)相当于 x = x - (-3)
<<=	算术左移指定的量，即 x <<= 1 等于 x = x<< 1
|=	按位或赋值运算符，对左侧变量和右侧表达式执行按位或操作，然后将结果赋值给左侧的变量。例如 a |= b 的操作等价于 a = a | b

5.7 数学函数

5.7.1 通用函数

数学函数定义见式（1）～式（5）。

$$Clip(i,j,x)=\begin{cases} i\,;x<i \\ j\,;x>j \\ x\,;其他 \end{cases} \quad \cdots\cdots (1)$$

式中：

x ——自变量 x；

i ——下界；

j ——上界。

$$Max(x,y)=\begin{cases} x\,;x\geq y \\ y\,;x<y \end{cases} \quad \cdots\cdots (2)$$

式中：

x ——自变量 x；

y ——自变量 y。

$$Min(x,y)=\begin{cases} x\,;x\leq y \\ y\,;x>y \end{cases} \quad \cdots\cdots (3)$$

式中：

x ——自变量 x；

y ——自变量 y。

$$Round(x) = \lfloor x + 0.5 \rfloor \quad \cdots\cdots (4)$$

式中：

x ——自变量x。

$$Sign(x) = \begin{cases} 1 ; x > 0 \\ 0 ; x = 0 \\ -1 ; x < 0 \end{cases} \quad \cdots\cdots (5)$$

式中：

x ——自变量x。

5.7.2 三维笛卡尔坐标到莫顿码的转换

输入：三维笛卡尔坐标(X,Y,Z)，其中 X、Y 和 Z 表示成 n-bit 的二进制数如下:

$X = x_{n-1}x_{n-2}\cdots x_1x_0$

$Y = y_{n-1}y_{n-2}\cdots y_1y_0$

$Z = z_{n-1}z_{n-2}\cdots z_1z_0$

输出：对应的三维莫顿码 M 的二进制表示即为：

$M = x_{n-1}y_{n-1}z_{n-1}x_{n-2}y_{n-2}z_{n-2}\cdots x_1y_1z_1x_0y_0z_0$

下表 6 举例展示了该转换过程：

表6 三维笛卡尔坐标到莫顿码的转换

二进制表示				整数表示
X	Y	Z	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
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
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
$x_{n-1}\cdots x_0$	$y_{n-1}\cdots y_0$	$z_{n-1}\cdots z_0$	$x_{n-1}y_{n-1}z_{n-1}\cdots x_0y_0z_0$	...

5.7.3 三维笛卡尔坐标到三维希尔伯特码的转换

输入：三维笛卡尔坐标(X,Y,Z)，其中 X、Y 和 Z 表示成 n-bit 的二进制数如下：

$X = x_{n-1}x_{n-2}\cdots x_1x_0$

$Y = y_{n-1}y_{n-2}\cdots y_1y_0$

$Z = z_{n-1}z_{n-2}\cdots z_1z_0$

输出：对应的三维希尔伯特码。

三维希尔伯特码 hilbertCode 通过下述方法得到：

```
state = 4
hilbertCode = 0
hilbertCode <<= 6
pos = ((X & 0xC0000) >> 14) | ((Y & 0xC0000) >> 16) | ((Z & 0xC0000) >> 18)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
pos = ((X & 0x30000) >> 12) | ((Y & 0x30000) >> 14) | ((Z & 0x30000) >> 16)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
pos = ((X & 0xC000) >> 10) | ((Y & 0xC000) >> 12) | ((Z & 0xC000) >> 14)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
pos = ((X & 0x3000) >> 8) | ((Y & 0x3000) >> 10) | ((Z & 0x3000) >> 12)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
pos = ((X & 0xC00) >> 6) | ((Y & 0xC00) >> 8) | ((Z & 0xC00) >> 10)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
pos = ((X & 0x300) >> 4) | ((Y & 0x300) >> 6) | ((Z & 0x300) >> 8)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
pos = ((X & 0xC0) >> 2) | ((Y & 0xC0) >> 4) | ((Z & 0xC0) >> 6)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
pos = (X & 0x30) | ((Y & 0x30) >> 2) | ((Z & 0x30) >> 4)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
```

```
pos = ((X & 0xC) << 2) | (Y & 0xC) | ((Z & 0xC) >> 2)
hilbertCode += HilbertTable[state][pos][1]
state = HilbertTable[state][pos][0]

hilbertCode <<= 6
pos = ((X & 0x3) << 4) | ((Y & 0x3) << 2) | (Z & 0x3)
hilbertCode += HilbertTable[state][pos][1]
```

其中三维 HilbertTable 查询如下：

```
HilbertTable[12][64][2] = {
{{4,0},   {10,3},  {6,60},  {2,63},  {9,7},   {10,4},  {6,59},  {8,56},
{0,8},    {5,9},   {1,54},  {0,55},  {8,11},  {5,10},  {1,53},  {9,52},
{0,1},    {0,2},   {0,61},  {0,62},  {7,6},   {7,5},   {7,58},  {7,57},
{11,15},  {6,14},  {10,49}, {11,48}, {8,12},  {6,13},  {10,50}, {9,51},
{9,26},   {8,29},  {9,34},  {8,37},  {0,27},  {0,28},  {0,35},  {0,36},
{0,16},   {5,17},  {1,46},  {0,47},  {8,19},  {5,18},  {1,45},  {9,44},
{9,25},   {8,30},  {9,33},  {8,38},  {6,24},  {10,31}, {6,32},  {10,39},
{11,23},  {6,22},  {10,41}, {11,40}, {8,20},  {6,21},  {10,42}, {9,43}},
{{11,38}, {11,37}, {1,62},  {0,63},  {3,33},  {3,34},  {1,61},  {9,60},
{11,30},  {11,29}, {1,2},   {4,3},   {3,25},  {3,26},  {1,1},   {7,0},
{4,39},   {1,36},  {10,57}, {11,56}, {9,32},  {1,35},  {10,58}, {9,59},
{4,31},   {1,28},  {10,5},  {4,4},   {9,24},  {1,27},  {10,6},  {3,7},
{5,40},   {1,47},  {5,48},  {1,55},  {4,41},  {2,46},  {4,49},  {2,54},
{9,22},   {8,17},  {9,14},  {8,9},   {5,23},  {1,16},  {5,15},  {1,8},
{3,43},   {3,44},  {3,51},  {3,52},  {4,42},  {2,45},  {4,50},  {2,53},
{9,21},   {8,18},  {9,13},  {8,10},  {11,20}, {11,19}, {11,12}, {11,11}},
{{6,52},  {2,55},  {5,56},  {1,63},  {6,51},  {8,48},  {4,57},  {2,62},
{6,44},   {2,47},  {2,36},  {5,37},  {6,43},  {8,40},  {7,39},  {5,38},
{0,53},   {0,54},  {3,59},  {3,60},  {7,50},  {7,49},  {4,58},  {2,61},
{0,45},   {0,46},  {2,35},  {6,34},  {7,42},  {7,41},  {3,32},  {6,33},
{11,10},  {11,9},  {7,4},   {7,3},   {3,13},  {3,14},  {4,5},   {2,2},
{11,18},  {11,17}, {2,28},  {5,29},  {3,21},  {3,22},  {7,31},  {5,30},
{5,11},   {2,8},   {6,7},   {10,0},  {5,12},  {8,15},  {4,6},   {2,1},
{5,19},   {2,16},  {2,27},  {6,26},  {5,20},  {8,23},  {3,24},  {6,25}},
{{2,20},  {5,21},  {1,42},  {4,43},  {7,23},  {5,22},  {1,41},  {7,40},
{5,24},   {1,31},  {5,32},  {1,39},  {4,25},  {2,30},  {4,33},  {2,38},
{2,19},   {6,18},  {10,45}, {4,44},  {3,16},  {6,17},  {10,46}, {3,47},
{3,27},   {3,28},  {3,35},  {3,36},  {4,26},  {2,29},  {4,34},  {2,37},
{2,12},   {5,13},  {1,50},  {4,51},  {7,15},  {5,14},  {1,49},  {7,48},
{11,6},   {11,5},  {11,58}, {11,57}, {3,1},   {3,2},   {3,61},  {3,62},
{2,11},   {6,10},  {10,53}, {4,52},  {3,8},   {6,9},   {10,54}, {3,55},
{4,7},    {1,4},   {5,59},  {2,56},  {9,0},   {1,3},   {5,60},  {8,63}},
{{5,0},   {1,7},   {4,8},   {10,11}, {4,1},   {2,6},   {9,15},  {10,12},
```

{1,26}, {4,27}, {4,16}, {10,19}, {1,25}, {7,24}, {9,23}, {10,20},
{3,3}, {3,4}, {0,9}, {0,10}, {4,2}, {2,5}, {7,14}, {7,13},
{10,29}, {4,28}, {0,17}, {0,18}, {10,30}, {3,31}, {7,22}, {7,21},
{7,60}, {7,59}, {11,54}, {11,53}, {4,61}, {2,58}, {3,49}, {3,50},
{1,34}, {4,35}, {11,46}, {11,45}, {1,33}, {7,32}, {3,41}, {3,42},
{6,63}, {10,56}, {4,55}, {1,52}, {4,62}, {2,57}, {9,48}, {1,51},
{10,37}, {4,36}, {4,47}, {1,44}, {10,38}, {3,39}, {9,40}, {1,43}},
{{0,0}, {5,1}, {11,26}, {11,25}, {8,3}, {5,2}, {3,29}, {3,30},
{2,60}, {5,61}, {11,34}, {11,33}, {7,63}, {5,62}, {3,37}, {3,38},
{11,7}, {6,6}, {5,27}, {2,24}, {8,4}, {6,5}, {5,28}, {8,31},
{2,59}, {6,58}, {5,35}, {2,32}, {3,56}, {6,57}, {5,36}, {8,39},
{5,8}, {1,15}, {5,16}, {1,23}, {4,9}, {2,14}, {4,17}, {2,22},
{9,54}, {8,49}, {9,46}, {8,41}, {5,55}, {1,48}, {5,47}, {1,40},
{3,11}, {3,12}, {3,19}, {3,20}, {4,10}, {2,13}, {4,18}, {2,21},
{9,53}, {8,50}, {9,45}, {8,42}, {11,52}, {11,51}, {11,44}, {11,43}},
{{7,52}, {7,51}, {7,44}, {7,43}, {4,53}, {2,50}, {4,45}, {2,42},
{9,10}, {8,13}, {9,18}, {8,21}, {0,11}, {0,12}, {0,19}, {0,20},
{6,55}, {10,48}, {6,47}, {10,40}, {4,54}, {2,49}, {4,46}, {2,41},
{9,9}, {8,14}, {9,17}, {8,22}, {6,8}, {10,15}, {6,16}, {10,23},
{0,56}, {5,57}, {6,36}, {2,39}, {8,59}, {5,58}, {6,35}, {8,32},
{2,4}, {5,5}, {6,28}, {2,31}, {7,7}, {5,6}, {6,27}, {8,24},
{11,63}, {6,62}, {0,37}, {0,38}, {8,60}, {6,61}, {7,34}, {7,33},
{2,3}, {6,2}, {0,29}, {0,30}, {3,0}, {6,1}, {7,26}, {7,25}},
{{2,52}, {5,53}, {1,10}, {4,11}, {7,55}, {5,54}, {1,9}, {7,8},
{4,56}, {10,59}, {6,4}, {2,7}, {9,63}, {10,60}, {6,3}, {8,0},
{2,51}, {6,50}, {10,13}, {4,12}, {3,48}, {6,49}, {10,14}, {3,15},
{0,57}, {0,58}, {0,5}, {0,6}, {7,62}, {7,61}, {7,2}, {7,1},
{2,44}, {5,45}, {1,18}, {4,19}, {7,47}, {5,46}, {1,17}, {7,16},
{7,36}, {7,35}, {7,28}, {7,27}, {4,37}, {2,34}, {4,29}, {2,26},
{2,43}, {6,42}, {10,21}, {4,20}, {3,40}, {6,41}, {10,22}, {3,23},
{6,39}, {10,32}, {6,31}, {10,24}, {4,38}, {2,33}, {4,30}, {2,25}},
{{6,20}, {2,23}, {0,24}, {5,25}, {6,19}, {8,16}, {8,27}, {5,26},
{6,12}, {2,15}, {9,6}, {8,1}, {6,11}, {8,8}, {5,7}, {1,0},
{0,21}, {0,22}, {11,31}, {6,30}, {7,18}, {7,17}, {8,28}, {6,29},
{0,13}, {0,14}, {9,5}, {8,2}, {7,10}, {7,9}, {11,4}, {11,3},
{11,42}, {11,41}, {0,32}, {5,33}, {3,45}, {3,46}, {8,35}, {5,34},
{11,50}, {11,49}, {9,58}, {8,61}, {3,53}, {3,54}, {0,59}, {0,60},
{5,43}, {2,40}, {11,39}, {6,38}, {5,44}, {8,47}, {8,36}, {6,37},
{5,51}, {2,48}, {9,57}, {8,62}, {5,52}, {8,55}, {6,56}, {10,63}},
{{1,38}, {0,39}, {4,40}, {10,43}, {1,37}, {9,36}, {9,47}, {10,44},
{9,62}, {8,57}, {4,48}, {10,51}, {5,63}, {1,56}, {9,55}, {10,52},
{10,33}, {11,32}, {0,41}, {0,42}, {10,34}, {9,35}, {7,46}, {7,45},
{9,61}, {8,58}, {0,49}, {0,50}, {11,60}, {11,59}, {7,54}, {7,53},

```
{1,30},    {0,31},    {11,22},   {11,21},   {1,29},    {9,28},    {3,17},    {3,18},
{9,2},     {8,5},     {11,14},   {11,13},   {0,3},     {0,4},     {3,9},     {3,10},
{10,25},   {11,24},   {4,23},    {1,20},    {10,26},   {9,27},    {9,16},    {1,19},
{9,1},     {8,6},     {4,15},    {1,12},    {6,0},     {10,7},    {9,8},     {1,11}},
{{7,20},   {7,19},    {7,12},    {7,11},    {4,21},    {2,18},    {4,13},    {2,10},
{9,42},    {8,45},    {9,50},    {8,53},    {0,43},    {0,44},    {0,51},    {0,52},
{6,23},    {10,16},   {6,15},    {10,8},    {4,22},    {2,17},    {4,14},    {2,9},
{9,41},    {8,46},    {9,49},    {8,54},    {6,40},    {10,47},   {6,48},    {10,55},
{4,24},    {10,27},   {1,6},     {0,7},     {9,31},    {10,28},   {1,5},     {9,4},
{4,32},    {10,35},   {1,58},    {4,59},    {9,39},    {10,36},   {1,57},    {7,56},
{0,25},    {0,26},    {10,1},    {11,0},    {7,30},    {7,29},    {10,2},    {9,3},
{0,33},    {0,34},    {10,61},   {4,60},    {7,38},    {7,37},    {10,62},   {3,63}},
{{9,38},   {8,33},    {9,30},    {8,25},    {5,39},    {1,32},    {5,31},    {1,24},
{0,40},    {5,41},    {1,22},    {0,23},    {8,43},    {5,42},    {1,21},    {9,20},
{9,37},    {8,34},    {9,29},    {8,26},    {11,36},   {11,35},   {11,28},   {11,27},
{11,47},   {6,46},    {10,17},   {11,16},   {8,44},    {6,45},    {10,18},   {9,19},
{11,62},   {11,61},   {11,2},    {11,1},    {3,57},    {3,58},    {3,5},     {3,6},
{0,48},    {5,49},    {1,14},    {0,15},    {8,51},    {5,50},    {1,13},    {9,12},
{4,63},    {1,60},    {5,3},     {2,0},     {9,56},    {1,59},    {5,4},     {8,7},
{11,55},   {6,54},    {10,9},    {11,8},    {8,52},    {6,53},    {10,10},   {9,11}}}
```

5.8 结构关系符

结构关系符定义见表 7。

表7 结构关系符

结构关系符	定义
->	例如：a->b 表示 a 是一个结构，b 是 a 的一个成员变量
()	取值范围区间，例如：x 的取值范围是(a, b)，表示 a<x<b
[]	取值范围区间，例如：x 的取值范围是[a, b]，表示 a<=x<=b

5.9 位流语法、解析过程和解码过程的描述方法

5.9.1 位流语法的描述方法

位流语法描述方法类似 C 语言。位流的语法元素使用粗体字表示。每个语法元素通过名字（用下划线分割的英文字母组，所有字母都是小写）、语法和语义来描述。语法表和正文中语法元素的值用常规字体表示。

某些情况下，可在语法表中应用从语法元素导出的其他变量值，这样的变量在语法表或正文中用不带下划线的小写字母和大写字母混合命名。大写字母开头的变量用于解码当前以及相关的语法结构，也可用于解码后续的语法结构。小写字母开头的变量只在它们所在的小节内使用。

语法元素值的助记符和变量值的助记符与它们的值之间的关系在正文中说明。在某些情况下，二者等同使用。助记符由一个或多个使用下划线分隔的字母组表示，每个字母组以大写字母开始，也可包括

多个大写字母。

位串的长度是 4 的整数倍时，可使用十六进制符号表示。十六进制的前缀是‘0x’，例如‘0x1a’表示位串‘0001 1010’。

条件语句中 0 表示 FALSE，非 0 表示 TRUE。

语法表描述了所有符合本文件的位流语法的超集，附加的语法限制在相关条中说明。

表 8 给出了描述语法的伪代码例子。当语法元素出现时，表示从位流中读一个数据单元。

表8　语法描述的伪代码

伪代码	描述符
/*语句是一个语法元素的描述符，或者说明语法元素的存在、类型和数值，下面给出两个例子。*/	
syntax_element	ue(v)
conditioning statement	
/*花括号括起来的语句组是复合语句，在功能上视作单个语句。*/	
{	
statement	
…	
}	
/*“while”语句测试 condition 是否为 TRUE，如果为 TRUE，则重复执行循环体，直到 condition 不为 TRUE。*/	
while (condition)	
statement	
/*“do … while”语句先执行循环体一次，然后测试 condition 是否为 TRUE，如果为 TRUE，则重复执行循环体，直到 condition 不为 TRUE。*/	
do	
statement	
while (condition)	
/*“if … else”语句首先测试 condition，如果为 TRUE，则执行 primary 语句，否则执行 alternative 语句。如果 alternative 语句不需要执行，结构的“else”部分和相关的 alternative 语句可忽略。*/	
if (condition)	
primary statement	
else	
alternative statement	
/*“for”语句首先执行 initial 语句，然后测试 condition，如果 condition 为 TRUE，则重复执行 primary 语句和 subsequent 语句直到 condition 不为 TRUE。*/	

表 8　语法描述的伪代码（续）

伪代码	描述符
for (initial statement; condition; subsequent statement)	
primary statement	
/*“break”语句用于 do-while、while 和 for 循环体中，可使当前循环体立即终止循环。*/	
break	

解析过程和解码过程用文字和类似 C 语言的伪代码描述。

5.9.2　函数

5.9.2.1　概述

以下函数用于语法描述。假定解码器中存在一个位流指针，这个指针指向位流中要读取的下一个二进制位的位置。函数由函数名及左右圆括号内的参数构成。函数也可没有参数。

5.9.2.2　byte_aligned( )

如果位流的当前位置是字节对齐的，返回 TRUE，否则返回 FALSE。

5.9.2.3　next_bits(n)

返回位流的随后 n 个二进制位，MSB 在前，不改变位流指针。如果剩余的二进制位少于 n，则返回 0。

5.9.2.4　byte_aligned_next_bits(n)

如果位流当前位置不是字节对齐的，返回位流当前位置的下一个字节开始的 n 个二进制位，MSB 在前，不改变位流指针；如果位流当前位置是字节对齐的，返回位流随后的 n 个二进制位，MSB 在前，不改变位流指针。如果剩余的二进制位少于 n，则返回 0。

5.9.2.5　next_start_code( )

在位流中寻找下一个起始码，将位流指针指向起始码前缀的第一个二进制位。函数定义见表 9。

表9　next_start_code 函数的定义

函数定义	描述符
next_start_code( ) {	
stuffing_bit	1
while (!byte_aligned( ))	
stuffing_bit	0
while (next_bits(24) != 0000 0000 0000 0000 0000 0001)	
stuffing_byte	00000000
}	

stuffing_byte 应出现图像头之后和第一个片起始码之前。

5.9.2.6 is_stuffing_pattern( )

在位流中检测当前字节中剩下的位或在字节对齐时，判断下一个字节是否是片结尾填充的二进制位，如果是，则返回 TRUE，否则返回 FALSE。此函数不修改位流指针。函数定义见表 10。

表10 is_stuffing_pattern 函数的定义

函数定义	描述符
is_stuffing_pattern( ) {	
if (next_bits(8-n) == (1<<(7-n))) /* n=0～7，为位流指针在当前字节的位置偏移，n 为 0 时位流指针指向当前字节最高位 */	
return TRUE	
else	
return FALSE	
}	

5.9.2.7 read_bits(n)

返回位流的随后 n 个二进制位，MSB 在前，同时位流指针前移 n 个二进制位。如果 n 等于 0，则返回 0，且位流指针不前移。

函数也用于解析过程和解码过程的描述。

5.9.3 描述符

描述符表示不同语法元素的解析过程，见表 11。

表11 描述符

描述符	说明
ae(v)	高级熵编码的语法元素。解析过程在 8.3 中定义。
b(8)	一个任意取值的字节。解析过程由函数 read_bits(8)的返回值规定。
f(n)	取特定值的连续 n 个二进制位。解析过程由函数 read_bits(n)的返回值规定。
i(n)	n 位整数。在语法表中，如果 n 是“v”，其位数由其他语法元素值确定。解析过程由函数 read_bits(n)的返回值规定，该返回值用高位在前的 2 的补码表示。
r(n)	连续 n 个 0。解析过程由函数 read_bits(n)的返回值规定。
se(v)	有符号整数语法元素，用指数哥伦布码编码。解析过程在 8.2 中定义。
u(n)	n 位无符号整数。在语法表中，如果 n 是“v”，其位数由其他语法元素值确定。解析过程由函数 read_bits(n)的返回值规定，该返回值用高位在前的二进制表示。
ue(v)	无符号整数语法元素，用指数哥伦布码编码。解析过程在 8.2 中定义。

5.9.4 保留、禁止和标记位

本文件定义的位流语法中，某些语法元素的值被标注为“保留”(reserved)或“禁止”(forbidden)。

“保留”定义了一些特定语法元素值。这些值用于将来对本文件的扩展，且不应出现在符合本文件的位流中。

“禁止”定义了一些特定语法元素值。这些值不应出现在符合本文件的位流中。

“标记位”（marker_bit）指该位的值应为1。

位流中的“保留位”（reserved_bits）表明保留了一些语法单元用于将来对本文件的扩展，解码处理应忽略这些位。“保留位”不应出现从任意字节对齐位置开始的21个以上连续的0。

6 编码位流的结构

6.1 点云序列

点云序列是位流的最高层语法结构。点云序列由序列头、几何头和属性头（若存在）开始，后面跟着一个或多个点云帧。每个点云帧应包含帧头和一个或多个点云片数据。

6.2 点云片

点云片由几何片头、几何数据、属性片头和属性数据组成，在本标准中片及点云片具有相同的含义。

7 位流的语法和语义

7.1 语法描述

7.1.1 起始码

起始码是一组特定的位串。在符合本文件的位流中，除起始码外的任何情况下都不应出现这些位串。

起始码由起始码前缀和起始码值构成。起始码前缀是位串‘0000 0000 0000 0000 0000 0001’。所有的起始码都应字节对齐。

起始码值是一个8位整数，用来表示起始码的类型，见表12。

表12 起始码值

起始码类型	起始码值（十六进制）
点云序列起始码（sequence_start_code）	00
点云序列结束码（sequence_end_code）	01
几何序列起始码（geometry_start_code）	02
属性序列起始码（attribute_start_code）	03
点云帧起始码（frame_start_code）	04
用户数据起始码（user_data_start_code）	05
点云几何片头起始码（geometry_slice_header_start_code）	06
点云属性片头起始码（attribute_slice_header_start_code）	07，08
几何数据起始码（geometry_slice_payload_start_code）	09
颜色数据起始码（color_slice_payload_start_code）	0A
反射率数据起始码（refl_slice_payload_start_code）	0B
保留	0C～FF

其中，attribute_slice_header_start_code = 07表示颜色对应的点云属性片头，attribute_slice_header_start_code = 08表示反射率对应的点云属性片头。

部分语法元素取特定值时可得到与起始码前缀相同的位串，称为伪起始码。符合本文件的编码器和

解码器应使用附录A 定义的方法处理伪起始码问题。

7.1.2 点云序列及片

7.1.2.1 点云序列

点云序列定义见表 13。

表13 点云序列定义

伪代码	描述符
PCC_sequence( ) {	
do {	
sequence_header( )	
geometry_header( )	
if (attribute_present_flag) {	
attribute_header( )	
}	
if (next_bits(32) == user_data_start_code)	
userdata_bitstream( )	
do {	
frame_header( )	
if (next_bits(32) == user_data_start_code)	
userdata_bitstream( )	
do {	
geometry_slice( )	
if (attribute_present_flag) {	
do {	
attribute_slice( )	
} while (next_bits(32) == attribute_slice_header_start_code)	
}	
} while (next_bits(32) == geometry_slice_header_start_code)	
} while (next_bits(32) == frame_start_code)	
} while (next_bits(32) != sequence_end_code)	
}	

7.1.2.2 几何片

几何片定义见表 14。

表14　几何片定义

伪代码	描述符
geometry_slice( ) {	
geometry_slice_header( )	
general_geometry_data_bitstream( )	
}	

7.1.2.3　属性片

属性片定义见表 15。

表15　属性片定义

伪代码	描述符
attribute_slice( ) {	
attribute_slice_header( )	
general_attribute_data_bitstream( )	
}	

7.1.2.4　序列头

序列头 SPS 定义见表 16。

表16　序列头定义

伪代码	描述符
sequence_header( ) {	
sequence_start_code	f(32)
profile_id	u(4)
level_id	u(8)
frame_rate_code	u(4)
geom_remove_duplicate_flag	u(1)
attribute_present_flag	u(1)
if (attribute_present_flag) {	
max_num_attributes_minus1	u(7)
multi_attributes_set_flag	u(1)
}	
byte_alignment( )	
}	

7.1.2.5　字节对齐

字节对齐定义见表 17。

表17　字节对齐定义

伪代码	描述符
byte_alignment( ) {	
while (!byte_aligned( ))	
alignment_bit_equal_to_one /* equal to 1 */	
}	

7.1.2.6　几何头

几何头 GPS 定义见表 18。

表18　几何头定义

伪代码	描述符
geometry_header( ) {	
geometry_start_code	f(32)
geometry_quant_step_significand	u(21)
marker_bit	f(1)
geometry_quant_step_exponent	u(5)
geom_max_tree_size_log2_minus8	ue(v)
implicit_geom_partition_flag	u(1)
single_mode_flag	u(1)
occupancy_search_range_side_log2	ue(v)
save_state_flag	u(1)
if (!save_state_flag) {	
lcu_dependency_flag	u(1)
}	
byte_alignment( )	
}	

7.1.2.7　属性头

属性头 APS 定义见表 19。

表19　属性头定义

伪代码	描述符
attribute_header( ) {	
attribute_start_code	f(32)
for (attrIdx = 0; attrIdx < (max_num_attributes_minus1 + 1); attrIdx ++) {	
attribute_data_present_flag[attrIdx]	u(1)
if (attribute_data_present_flag[attrIdx]) {	
attribute_data_num_set_minus1[attrIdx]	ue(v)
if ((attrIdx == 1) && (attribute_data_num_set_minus1[1] > 0)) {	
for (i = 0; i < ((attribute_data_num_set_minus1[1] + 1); I ++) {	
multi_attr_group_id[i]	ue(v)
}	
}	
if (multi_attributes_set_flag)	
multi_data_set_flag[attrIdx]	u(1)
if (multi_data_set_flag[attrIdx])	
attribute_info_num_set_minus1[attrIdx]	ue(v)
for (i = 0; i < (attribute_info_num_set_minus1[attrIdx] + 1); i++) {	
output_bit_depth_minus1[attrIdx][i]	ue(v)
attr_quant_param[attrIdx][i]	ue(v)
if (attrIdx == 0) {	
order_switch[i]	u(1)
color_reorder_mode[i]	ue(v)
color_golomb_num[i]	ue(v)
golomb_group_size_log2[i]	ue(v)
}	
if (attrIdx == 1) {	
axis_bias_minus1[i]	ue(v)
refl_reorder_mode[i]	ue(v)
refl_golomb_num[i]	ue(v)
pred_fixed_point_frac_bit[i]	ue(v)
}	
transform[attrIdx][i]	u(2)
if ((transform[attrIdx][i] == 0) || (transform[attrIdx][i] == 2)) {	
max_num_of_neighbours_log2_minus7[attrIdx][i]	u(2)
if (attrIdx == 0) {	
cross_component_pred[i]	u(1)
chroma_qp_offset_cb[i]	se(v)
chroma_qp_offset_cr[i]	se(v)
}	

表 19　属性头定义（续）

伪代码	描述符
if (attrIdx == 1) {	
nearest_pred_param1[i]	ue(v)
nearest_pred_param2[i]	ue(v)
pred_dist_weight_group_size_log2[i]	ue(v)
}	
}	
if (transform[attrIdx][i] == 1) {	
transform_segment_size_upper[attrIdx][i]	u(16)
marker_bit	f(1)
transform_segment_size_lower[attrIdx][i]	u(16)
marker_bit	f(1)
k_frac_bits[attrIdx][i]	ue(v)
attr_transform_qp_delta[attrIdx][i]	ue(v)
trans_res_layer[attrIdx][i]	u(1)
}	
if (transform[attrIdx][i] == 2) {	
max_num_of_coeff_log2_minus8[attrIdx][i]	ue(v)
qp_offset_dc[attrIdx][i]	se(v)
qp_offset_ac[attrIdx][i]	se(v)
if (attrIdx == 0) {	
color_max_trans_num[i]	ue(v)
chroma_qp_offset_dc[i]	se(v)
chroma_qp_offset_ac[i]	se(v)
color_qp_adjust_flag[i]	u(1)
}	
if (attrIdx == 1) {	
refl_max_trans_num[i]	ue(v)
refl_group_pred[i]	u(1)
}	
}	
coeff_length_control_log2_minus8[attrIdx][i]	ue(v)
}	
}	
}	
if ((attribute_data_num_set_minus1[0] == 0) && (attribute_data_num_set_minus1[1] == 0)) {	

表 19　属性头定义（续）

伪代码	描述符
cross_attr_type_pred	u(1)
if (cross_attr_type_pred) {	
attr_coding_order	u(1)
cross_attr_type_pred_param1	u(15)
marker_bit	f(1)
cross_attr_type_pred_param2	u(21)
marker_bit	f(1)
}	
}	
byte_alignment( )	
}	

7.1.2.8　点云帧头

帧头 frame_header 定义见表 20。

表20　帧头定义

伪代码	描述符
frame_header( ) {	
frame_start_code	f(32)
frame_idx	ue(v)
marker_bit	f(1)
frame_num_slice_minus1	ue(v)
lcu_node_size_log2_minus1	ue(v)
geom_num_points_upper	u(16)
marker_bit	f(1)
geom_num_points_lower	u(16)
marker_bit	f(1)
bounding_box_offset_x_upper	u(16)
marker_bit	f(1)
bounding_box_offset_x_lower	u(16)
marker_bit	f(1)
bounding_box_offset_y_upper	u(16)
marker_bit	f(1)
bounding_box_offset_y_lower	u(16)
marker_bit	f(1)
bounding_box_offset_z_upper	u(16)
marker_bit	f(1)
bounding_box_offset_z_lower	u(16)

表 20　帧头定义（续）

伪代码	描述符
marker_bit	f(1)
bounding_box_size_width_upper	u(16)
marker_bit	f(1)
bounding_box_size_width_lower	u(16)
marker_bit	f(1)
bounding_box_size_height_upper	u(16)
marker_bit	f(1)
bounding_box_size_height_lower	u(16)
marker_bit	f(1)
bounding_box_size_depth_upper	u(16)
marker_bit	f(1)
bounding_box_size_depth_lower	u(16)
marker_bit	f(1)
byte_alignment( )	
}	

7.1.2.9　几何片头

几何片头 GSH 定义见表 21。

表21　几何片头定义

伪代码	描述符
geometry_slice_header( ) {	
geometry_slice_header_start_code	f(32)
slice_id	ue(v)
marker_bit	f(1)
context_mode	u(1)
if (implicit_geom_partition_flag) {	
max_num_implicit_qtbt_before_ot	ue(v)
min_size_implicit_qtbt	ue(v)
}	
if (single_mode_flag) {	
gsh_single_mode_flag	u(1)
}	
planar_mode	u(1)
marker_bit	f(1)
slice_bounding_box_offset_x_upper	u(16)
marker_bit	f(1)
slice_bounding_box_offset_x_lower	u(16)

表 21　几何片头定义（续）

伪代码	描述符
marker_bit	f(1)
slice_bounding_box_offset_y_upper	u(16)
marker_bit	f(1)
slice_bounding_box_offset_y_lower	u(16)
marker_bit	f(1)
slice_bounding_box_offset_z_upper	u(16)
marker_bit	f(1)
slice_bounding_box_offset_z_lower	u(16)
marker_bit	f(1)
slice_bounding_box_sizeXLog2	u(6)
slice_bounding_box_sizeYLog2	u(6)
slice_bounding_box_sizeZLog2	u(6)
marker_bit	f(1)
slice_num_points_upper	u(16)
marker_bit	f(1)
slice_num_points_lower	u(16)
marker_bit	f(1)
byte_alignment( )	
}	

7.1.2.10　属性片头

属性片头 ASH 定义见表 22。

表22　属性片头定义

伪代码	描述符
attribute_slice_header( ) {	
attribute_slice_header_start_code	f(32)
slice_id	ue(v)
marker_bit	f(1)
attribute_id	ue(v)
qp_offset	se(v)
color_init_pred_trans_ratio	se(v)
refl_init_pred_trans_ratio	se(v)
if (color_qp_adjust_flag) {	
color_qp_adjust_scalar	ue(v)
}	
byte_alignment( )	
}	

7.1.3 几何数据

7.1.3.1 通用几何语法

通用几何语法见表 23。

表23 通用几何语法

伪代码	描述符
general_geometry_data_bitstream( ) {	
geometry_slice_payload_start_code	f(32)
geometry_data( )	
byte_alignment( )	
}	

7.1.3.2 几何数据语法

几何数据语法见表 24。

表24 几何数据语法

伪代码	描述符
geometry_data( ) {	
depthX = depthY = depthZ = 0	
for (depth = 0; depth < MaxGeometryOctreeDepth; depth++) {	
if (gsh_single_mode_flag && depth != 0)	
single_point_eligible_flag_per_depth[depth]	ae(v)
for (nodeIdx = 0; nodeIdx < NumNodesAtDepth[depth]; nodeIdx++) {	
if (lcu_node_size_log2 > max(NodeSizeXLog2, NodeSizeYLog2, NodeSizeZLog2)) {	
geom_tree_type	ae(v)
if (geom_tree_type == 0) {	
lcu_octree_data( )	
} else {	
lcu_predtree_data( )	
}	
} else {	
xN = NodeX[depth][nodeIdx]	
yN = NodeY[depth][nodeIdx]	
zN = NodeZ[depth][nodeIdx]	
geometry_node(depthX, depthY, depthZ, partitionSkip, nodeIdx, xN, yN, zN , single_point_eligible_flag_per_depth[depth])	
}	
} // end for nodeIdx loop	

表 24　几何数据语法（续）

伪代码	描述符
if (!(partitionSkip & 4))	
depthX = depthX + 1	
if (!(partitionSkip & 2))	
depthY = depthY + 1	
if (!(partitionSkip & 1))	
depthZ = depthZ + 1	
} // end of depth loop	
termination_bit_one /* equals to 1 */	ae(v)
}	

7.1.3.3　八叉树数据语法

八叉树数据语法见表 25。

表25　八叉树数据语法

伪代码	描述符
lcu_octree_data( ) {	
lcuDepthX = lcuDepthY = lcuDepthZ = 0	
for (lcuDepth = 0; lcuDepth < lcuMaxGeometryOctreeDepth; lcuDepth++) {	
if (gsh_single_mode_flag && lcuDepth != 0)	
single_point_eligible_flag_per_depth[depth]	ae(v)
for (nodeIdx = 0; nodeIdx < lcuNumNodesAtDepth[lcuDepth]; nodeIdx++) {	
xN = lcuNodeX[lcuDepth][nodeIdx]	
yN = lcuNodeY[lcuDepth][nodeIdx]	
zN = lcuNodeZ[lcuDepth][nodeIdx]	
geometry_node(lcuDepthX, lcuDepthY, lcuDepthZ, lcuPartitionSkip, nodeIdx, xN, yN, zN, single_point_eligible_flag_per_depth[depth])	
} // end for nodeIdx	
if (!(lcuPartitionSkip & 4))	
lcuDepthX = lcuDepthX + 1	
if (!(lcuPartitionSkip & 2))	
lcuDepthY = lcuDepthY + 1	
If (!(lcuPartitionSkip & 1))	
lcuDepthZ = lcuDepthZ + 1	
}	
termination_bit_one /* equals to 1 */	ae(v)
}	

7.1.3.4 预测树数据语法

预测树数据语法见表 26。

表26 预测树数据语法

伪代码	描述符
lcu_predtree_data( ) {	
num_bits_in_lcu_num_points	ae(v)
for (i = 0; i < num_bits_in_lcu_num_points; i++) {	
lcu_num_points[i]	ae(v)
}	
for (n = 0; n < lcu_num_points; n++) {	
for (k = 0; k < 3; k++) {	
is_geom_residual_zero[n][k]	ae(v)
if (!is_geom_residual_zero[n][k]) {	
num_bits_geom_residual_minus1[n][k]	ae(v)
for (j = 0; j < num_bits_geom_residual_minus1[n][k] – 1; j++) {	
geom_residual_minus1_div2[n][k][j]	ae(v)
}	
geom_residual_minus1_div2_remain[n][k]	ae(v)
}	
} // loop over k	
if (geom_residual_max_rel_sign[n] >= 4) {	
geom_residual_ord_rel_sign[n][0]	ae(v)
}	
if ((geom_residual_ord_rel_sign[n][0] >> 2) + 2 <= geom_residual_max_rel_sign[n]) {	
geom_residual_ord_rel_sign[n][1]	ae(v)
}	
if ((geom_residual_ord_rel_sign[n][0] >> 2) + (geom_residual_ord_rel_sign[n][1] >> 1) + 1 <= geom_residual_max_rel_sign[n]) {	
geom_residual_ord_rel_sign[n][2]	ae(v)
}	
} // loop over points	
termination_bit_one /* equals to 1 */	ae(v)
}	

7.1.3.5 几何节点语法

几何节点语法见表 27。

表27　几何节点语法

伪代码	描述符
geometry_node(depth, depthX, depthY, depthZ, partitionSkip, nodeIdx, xN, yN, zN, numSiblings， single_point_eligible_flag_per_depth[depth]) {	
if (single_point_eligible_flag_per_depth [depth] && (!GeomSingleNodeControlFlag)) {	
if (GeometryNodeOccupancyCnt[depth][xN][yN][zN] == 1 && GeomSingleEligibleFlag[depth - 1]) {	
geom_single_flag = 0	
} else {	
geom_single_flag	ae(v)
}	
if (geom_single_flag) {	
for (i = ChildNodeSizeXLog2 - 1; i >= 0; --i) {	
point_offset_x[i]	ae(v)
}	
for (i = ChildNodeSizeYLog2-1; i >= 0; --i) {	
point_offset_y[i]	ae(v)
}	
for (i = ChildNodeSizeZLog2-1; i >= 0; --i) {	
point_offset_z[i]	ae(v)
}	
}	
} else {	
if (!geom_single_flag) {	
occupancy_code	ae(v)
if (depthX >= MaxNodeSizeXLog2 − 1 && depthY >= MaxNodeSizeYLog2 – 1 && depthZ >= MaxNodeSizeZLog2 – 1) {	
if (!geomRemoveDuplicateFlag) {	
for (child = 0; child < GeometryNodeChildrenCnt; child++) {	
num_duplicated_points_eq1	ae(v)
if (!num_duplicated_points_eq1)	
num_duplicated_points_minus2	ae(v)
}	
}	
}	
}	
}	
}	

7.1.4　属性数据

7.1.4.1　通用属性语法

通用属性语法见表 28。

表28　通用属性语法

伪代码	描述符
general_attribute_data_bitstream( ) {	
if (next_start_code( ) == color_slice_payload_start_code)	
general_color_data_bitstream( )	
if (next_start_code( ) == refl_slice_payload_start_code)	
general_refl_data_bitstream( )	
}	

7.1.4.2　通用颜色语法

通用颜色语法见表 29。

表29　通用颜色语法

伪代码	描述符
general_color_data_bitstream( ) {	
color_slice_payload_start_code	f(32)
attribute_data_color( )	
byte_alignment( )	
}	

7.1.4.3　通用反射率语法

通用反射率语法见表 30。

表30　通用反射率语法

伪代码	描述符
general_refl_data_bitstream( ) {	
refl_slice_payload_start_code	f(32)
if (attribute_data_num_set_minus1[1] > 1 && multi_attr_group_num > 1) {	
attribute_data_multiple_refl( )	
} else {	
attribute_data_refl( )	
}	
byte_alignment( )	
}	

7.1.4.4　颜色数据语法

颜色数据语法定义见表 31。

表31　颜色数据

伪代码	描述符
attribute_data_color ( ) {	
isLengthControl = false	
residual_zero_run_length = zero_run_length_code( )	
for (i = 0; i < slice_num_points; i ++) {	
if (residual_zero_run_length = maxLatency) {	
isLengthControl = true	
}	
if (residual_zero_run_length) {	
residual_zero_run_length--	
} else {	
if (isLengthControl) {	
residual_zero_run_length = zero_run_length_code( )	
isLengthControl = false	
} else {	
color_residual_correlation_code( )	
residual_zero_run_length = zero_run_length_code( )	
}	
}	
}	
termination_bit_one /* equals to 1 */	ae(v)
}	

7.1.4.5　反射率数据语法

反射率数据语法定义见表 32。

表32　反射率数据

伪代码	描述符
attribute_data_refl( ) {	
isLengthControl = false	
residual_zero_run_length = zero_run_length_code()	
for (i = 0; i < slice_num_points; i++) {	
if (residual_zero_run_length = maxLatency) {	
isLengthControl = true	
}	

表 32　反射率数据（续）

伪代码	描述符
if (residual_zero_run_length) {	
-- residual_zero_run_length	
abs_reflectance = 0	
} else {	
if (isLengthControl) {	
residual_zero_run_length = zero_run_length_code( )	
isLengthControl = false	
} else {	
if (!isDuplicatePoint) {	
residual_sign	ae(v)
}	
refl_minus1_parity	ae(v)
refl_minus1_div2_eq0	ae(v)
if (!refl_minus1_div2_eq0) {	
refl_minus1_div2_eq1	ae(v)
if (!reflabs_level_minus1_div2_eq1) {	
reflabs_level_minus1_div2_minus2	ae(v)
abs_reflectance = (refl_minus1_div2_minus2 + 2) << 1 + 1 + refl_minus1_parity	
} else {	
abs_reflectance = 3 + refl_minus1_parity	
}	
} else {	
abs_reflectance = 1 + refl_minus1_parity	
}	
residual_zero_run_length = zero_run_length_code( )	
}	
}	
}	
termination_bit_one /* equals to 1 */	ae(v)
}	

7.1.4.6　零游程编码语法

零游程编码语法定义见表 33。

表33　零游程编码

伪代码	描述符
zero_run_length_code( ) {	
zero_run_length_eq0	ae(v)

表 33　零游程编码（续）

伪代码	描述符
if (!zero_run_length_eq0) {	
zero_run_length_minus1	ae(v)
}	
}	

7.1.4.7　颜色残差编码语法

颜色残差编码语法定义见表 34。

表34　颜色残差编码

伪代码	描述符
color_residual_correlation_code( ) {	
color_first_comp_zero	ae(v)
if (!color_first_comp_zero) {	
color_component[0] = color_component_code(true, 0)	
color_component[1] = color_component_code(false, 1)	
color_component[2] = color_component_code(false, 2)	
if (!(isDuplicatePoint && order_switch == 0))	
color_component_sign[0]	ae(v)
if (color_component[1] != 0 && !(isDuplicatePoint && order_switch == 1))	
color_component_sign[1]	ae(v)
if (color_component[2] != 0)	
color_component_sign[2]	ae(v)
} else {	
color_component[0] = 0	
color_second_comp_zero	ae(v)
if (!color_second_comp_zero) {	
color_component[1] = color_component_code(true, 1)	
color_component[2] = color_component_code(false, 2)	
if (!(isDuplicatePoint && order_switch == 1))	
color_component_sign[1]	ae(v)
if (color_component[2] != 0)	
color_component_sign[2]	ae(v)
} else {	
color_component[1] = 0	
color_component[2] = color_component_code(true, 2)	
color_component_sign[2]	ae(v)
}	
}	
}	

7.1.4.8 颜色分量编码语法

颜色分量编码定义见表 35。

表35 颜色分量编码

伪代码	描述符
color_component_code(isComponentNoneZero, i) {	
if (!isComponentNoneZero) {	
color_eq0	ae(v)
if (!color_eq0) {	
color_eq1	ae(v)
if (!color_eq1) {	
color_parity	ae(v)
color_minus2_div2_eq0	ae(v)
if (!color_minus2_div2_eq0){	
color_minus2_div2_minus1	ae(v)
color_component[i] = (color_minus2_div2_minus1 + 1) << 1 + 2 + color_parity	
} else {	
color_component[i] = 2 + color_parity	
}	
} else {	
color_component[i] = 1	
}	
} else {	
color_component[i] = 0	
}	
} else {	
color_minus1_eq0	ae(v)
if (!color_minus1_eq0) {	
color_minus1_eq1	ae(v)
if (!color_minus1_eq1) {	
color_parity	ae(v)
color_minus1_minus2_div2_eq0	ae(v)
if (!color_minus1_minus2_div2_eq0){	
color_minus1_minus2_div2_minus1	ae(v)
color_component[i] = (color_minus2_minus1_div2_minus1 + 1) << 1 + 3 + color_parity	
} else {	
color_component[i] = 3 + color_parity	
}	
} else {	

表 35　颜色分量编码（续）

伪代码	描述符
color_component[i] = 2	
}	
} else {	
color_component[i] = 1	
}	
}	
}	

7.1.4.9　多反射率数据编码语法

多反射率数据编码语法定义见表 36。

表36　多反射率数据编码

伪代码	描述符
attribute_data_multiple_refl( ) {	
isLengthControl = false	
residual_zero_run_length = zero_run_length_code( )	
for (i = 0; i < slice_num_points; i ++) {	
for (j = 0; j < multi_attr_group_num; j++) {	
if (residual_zero_run_length = maxLatency) {	
isLengthControl = true	
}	
}	
}	
termination_bit_one /* equals to 1 */	ae(v)
}	

7.1.4.10　用户数据

用户数据语法定义见表 37。

表37　用户数据定义

伪代码	描述符
userdata_bitstream( ) {	
user_data_start_code	f(32)
while (next_bits(24) != 0000 0000 0000 0000 0000 0001) {	
user_data	u(8)
}	
}	

7.2 语义描述

7.2.1 序列头

档次标号 profile_id

4 位无符号整数，取值范围为[1,15]。表示位流符合的档次。

级别标号 level_id

8 位无符号整数，取值范围为[1,255]。表示位流符合的级别。

档次和级别见附录B。

帧率代码 frame_rate_code

4 位无符号整数，取值范围为[0,15]。规定帧率，用于点云解码呈现，见表 38。

表38 帧率代码

frame_rate_code 的值	帧率
0000	禁止
0001	10
0010	20
0011	30
0100～1111	保留

几何移除重复点标志 geom_remove_duplicate_flag

二值变量。值为 0 表示几何编码前不去除重复点，即几何位置相同的点；值为 1 表示去除重复点。

属性存在标志 attribute_present_flag

二值变量。值为 0 表示码流不包含属性编码,即不包含属性头，属性片头，和属性数据；值为 1 表示码流包含属性编码。

最大属性数减一 max_num_attributes_minus1

7 位无符号整数，取值范围为[0,127]。加 1 表示码流支持的最大属性种类的编码数目。当 max_num_attributes_minus1 不出现在码流的时候，max_num_attributes_minus1 的默认值为-1。

多属性信息开启标志 multi_attributes_set_flag

二值变量。值为 0 表示不支持同一属性数据使用多套属性信息，同一属性数据只可以使用一套属性信息；值为 1 表示支持同一属性数据使用多套属性信息。当 multi_attributes_set_flag 不出现在码流时，其默认值为 0。

7.2.2 字节对齐

对齐位 alignment_bit_equal_to_one

填充位。值为 1。

7.2.3 几何头

几何量化步长有效值 geometry_quant_step_significand

21 位无符号整数，取值范围为[0,2^{21}-1]。表示几何量化步长有效值。

几何量化步长指数 geometry_quant_step_exponent

5 位无符号整数，取值范围为[0,20]。表示几何量化步长有效值中小数部分位数。

几何预测树最大点数 geom_max_tree_size_log2_minus8

无符号整数。几何预测树的最大点数为 geom_max_tree_size = 1 << (geom_max_tree_size_log2_minus8 + 8)。

几何隐式划分标志 implicit_geom_partition_flag

二值变量。值为 0 表示关闭几何隐式划分；值为 1 表示打开几何隐式划分。

几何孤立点编码模式标志 single_mode_flag

二值变量。值为 0 表示关闭几何孤立点编码模式；值为 1 表示打开几何孤立点编码模式。

几何占位信息搜索范围边长大小 occupancy_search_range_side_log2

无符号整数。其值指示占位码编码时邻域节点占位信息的搜索范围边长。该搜索范围大小为 occupancy_search_range = 1 << (3 * occupancy_search_range_side_log2)。

几何编码状态存储标志 save_state_flag

二值变量。值为 0 表示不存储编码状态，即熵编码上下文和几何编码的哈希表信息；值为 1 表示存储编码状态。

几何宏块编码状态依赖标志 lcu_dependency_flag

二值变量。值为 0 表示几何宏块编码相互独立；值为 1 表示几何宏块编码状态具有依赖。默认值为 0，即默认几何宏块编码相互独立。

7.2.4 属性头

属性数据存在标志 attribute_data_present_flag[attrIdx]

二值变量。值为 0 表示码流不包含第 attrIdx 属性编码；值为 1 表示码流包含第 attrIdx 属性编码。attrIdx 取值范围为[0,15]。其含义见表 39。其默认值为 0。

表39 attrIdx 含义

attrIdx	属性数据
0	Color
1	Reflectance
2～15	保留

属性数据数目减一 attribute_data_num_set_minus1[attrIdx]

无符号整数，取值范围为[0,127]。加 1 表示码流中由属性索引 attrIdx 确定的属性支持的该属性

多数据集的数目，当 attribute_data_num_set_minus1[attrIdx]不出现在码流的时候，其默认值为-1。

多属性分组参数 multi_attr_group_id

无符号整数数组，取值范围为[0,127]。当 attribute_data_num_set_minus1[1]大于 0 时，表示支持多维反射率属性维度分组，同一维度组内的多维属性需要按组解码码流。multi_attri_group_id[i]表示第 i 个维度属性属于第 multi_attri_group_id[i]属性维度分组，取值介于 0 到 127 之间。multi_attr_group_id[0]的取值为 0，multi_attr_group_id[i+1]减 multi_attr_group_id[i]等于 0 或 1。定义 multi_attri_group_num 表示第 multi_attri_group_id[i]属性维度分组内的属性维度个数，可通过累加连续相同 multi_attri_group_id[i]的属性维度个数得到。

属性信息开启标志 multi_data_set_flag[attrIdx]

二值变量。值为 0 表示由属性索引 attrIdx 确定的属性不支持使用多套属性信息，该属性数据只可以使用一套属性信息；值为 1 表示由属性索引 attrIdx 确定的属性支持使用多套属性信息。当 multi_data_set_flag[attrIdx]不出现在码流时，其默认值为 0。

属性信息数目减一 attribute_info_num_set_minus1[attrIdx]

无符号整数，取值范围为[0,127]。加 1 表示码流中由属性索引 attrIdx 确定的属性支持的属性信息的数目，当 attribute_info_num_set_minus1[attrIdx]不出现在码流的时候，其默认值为 0。

属性输出位深减一 output_bit_depth_minus1

无符号整数，取值范围为[0,31]，用于表示属性输出位深度。属性输出位深为 outputBitDepth = output_bit_depth_minus1 + 1。如果该语法元素不出现在码流时，默认值为 0。

属性量化参数 attr_quant_param

无符号整数，取值范围为[0,127]，用于表示属性的量化参数。最小属性量化参数 minQp = 0，最大属性量化参数 maxQp = 127。亮度通道的量化参数 lumaQp = attr_quant_param。

色度通道 Cb 量化参数偏移量 chroma_qp_offset_cb

有符号整数，取值范围为[-16,16]。用于表示 Cb 通道量化参数。如果该语法元素不出现在码流时，默认值为 0。色度通道 Cb 的量化参数 chromaQpCb = Clip(minQp, maxQp, attr_quant_param + chroma_qp_offset_cb)。

色度通道 Cr 量化参数偏移量 chroma_qp_offset_cr

有符号整数，取值范围为[-16,16]。用于表示 Cr 通道量化参数。如果该语法元素不出现在码流时，默认值为 0。色度通道 Cr 的量化参数 chromaQpCr = Clip(minQp, maxQp, attr_quant_param + chroma_qp_offset_cr)。

颜色残差编码顺序开关 order_switch

二值变量。值为 0 表示颜色残差解码顺序是 YUV/RGB，颜色第一分量是 Y（颜色空间是 YUV）或者 R（颜色空间是 RGB），颜色第二分量是 U（颜色空间是 YUV）或者 G（颜色空间是 RGB），颜色第三分量是 V（颜色空间是 YUV）或者 B（颜色空间是 RGB）；值为 1 表示颜色残差解码顺序是 UYV/GRB，颜色第一分量是 U（颜色空间是 YUV）或者 G（颜色空间是 RGB），颜色第二分量是 Y（颜色空间是 YUV）或者 R（颜色空间是 RGB），颜色第三分量是 V（颜色空间是 YUV）或者 B（颜色空间是 RGB）。

颜色重排序模式 color_reorder_mode

无符号整数，取值范围为[0,2]。用于表示颜色属性的重排序模式。值为 0 表示原始顺序，值为 1 表示希尔伯特排序，值为 2 表示莫顿排序。当颜色属性和反射率属性同时存在且跨属性预测存在时，颜色属性和反射率属性必须有相同的重排序。

颜色指数哥伦布阶数 color_golomb_numColorGolombNum

无符号整数，取值范围为[0,8]。用于表示颜色预测残差或变换系数在进行解码时，采用的 K 阶指数哥伦布的阶数 K = color_golomb_num。

自适应指数哥伦布编码滑动窗口大小 golomb_group_size_log2

无符号整数，取值范围为[0,8]。用于表示属性预测残差或变换系数的自适应指数哥伦布编码滑动窗口大小。自适应指数哥伦布编码滑动窗口大小为 golomb_sliding_window_size = 1 << golomb_group_size_log2。

空间偏倚系数减一 axis_bias_minus1

无符号整数，取值范围为[0,15]。用于表示属性预测值计算中在 Z 方向上的加权因子。Z 方向上的加权因子为 axisBias = axis_bias_minus1 + 1。

反射率重排序模式 refl_reorder_mode

无符号整数，取值范围为[0,2]，用于表示反射率属性的重排序模式。值为 0 表示原始顺序，值为 1 表示希尔伯特排序，值为 2 表示莫顿排序。当颜色属性和反射率属性同时存在且跨属性预测存在时，颜色属性和反射率属性必须有相同的重排序。

反射率指数哥伦布阶数 refl_golomb_num

无符号整数，取值范围为[0,8]。用于表示反射率预测残差或变换系数在进行解码时，采用的 K 阶指数哥伦布的阶数 K = refl_golomb_num。

反射率属性预测精度值 pred_fixed_point_frac_bit

无符号整数，取值范围为[0,30]。用于表示反射率预测时的定点化运算的移位精度。值为 0 表示不使用定点化运算。

属性变换算法标志 transform

无符号整数，取值范围[0,2]，用于表示属性编码方法。值为 0 时表示预测方法；值为 1 时表示多层变换方法；值为 2 时表示预测变换方法。

最大搜索的邻居点数对数值减七 max_num_of_neighbours_log2_minus7

无符号整数，取值范围为[0,3]。用于表示搜索的最大已编码邻居点数。最大搜索的邻居点数为 maxNumOfNeighbours = 1 << (max_num_of_neighbours_log2_minus7 + 7)。当 cross_attr_type_pred = 1 时，颜色属性和反射率属性有相同的最大搜索邻居点数。

属性残差二次预测 cross_component_pred

二值变量。值为 0 表示不使用属性残差二次预测；值为 1 表示使用属性残差二次预测。

最近邻点预测参数一 nearest_pred_param1 和最近邻点预测参数二 nearest_pred_param2

无符号整数，取值范围为[0,32]。用于表示最近邻点预测的阈值的参数 1 和参数 2。最近邻点预测的阈值为 attr_quant_param * nearest_pred_param1 + nearest_pred_param2。

属性值变化统计窗口大小 pred_dist_weight_group_size_log2

无符号整数，取值范围为[0,7]。用于表示属性值变化统计窗口的大小。统计窗口大小 pred_dist_ weight_group_size = 1 << pred_dist_weight_group_size_log2。

属性变换点数高 16 位部分 transform_segment_size_upper 和属性变换点数低 16 位部分 transform_segment_size_lower

16 位无符号整数。表示属性变换点数 32 位无符号整数的高 16 位和低 16 位。属性变换点数为 transform_segment_size = transform_segment_size_upper << 16 + transform_segment_size_ lower。

属性变换精度值 k_frac_bits

无符号整数，取值范围为[1,48]。用于表示属性变换时的定点化运算的移位精度。

属性变换系数量化参数差值 attr_transform_qp_delta

无符号整数，取值范围为[0,32]。用于表示属性变换系数与属性预测残差的量化参数的差值。属性变换系数量化参数为 attrTransformQp = attr_quant_param + attr_transform_qp_delta。

变换残差层标志 trans_res_layer

二值变量。值为 0 表示不使用属性残差补偿；值为 1 表示使用属性残差补偿。

最大缓存限制参数 max_num_of_coeff_log2_minus8

无符号整数,取值范围为[0,10]。用于表示变换系数的最大缓存参数。变换系数的最大缓存为 maxNumofCoeff = 1 << (max_num_of_coeff_log2_minus8 + 8)。当 max_num_of_coeff_log2_minus8 不出现在码流时，maxNumofCoeff 的默认值为 1。

预测变换 DC 系数量化参数偏移量 qp_offset_dc

有符号整数，取值范围为[-16,16]。用于表示预测变换中颜色的亮度分量或是反射率对应的 DC 系数的量化参数偏移量，默认值为 0。

预测变换 AC 系数量化参数偏移量 qp_offset_ac

有符号整数，取值范围为[-16,16]。用于表示预测变换中颜色的亮度分量或是反射率对应的 AC 系数的量化参数偏移量，默认值为 0。

颜色最大变换阶数 color_max_trans_numColorGolombNum

无符号整数，取值范围为[0,8]。用于表示当前颜色预测变换中的最大变换阶数。

色度通道 DC 系数量化参数偏移量 chroma_qp_offset_dc

有符号整数，取值范围为[-16,16]。用于表示预测变换中颜色的色度分量对应的 DC 系数的量化参数偏移量，默认值为 0。

色度通道 DC 系数量化参数偏移量 chroma_qp_offset_ac

有符号整数，取值范围为[-16,16]。用于表示预测变换中颜色的色度分量对应的 AC 系数的量化参数偏移量，默认值为 0。

点云自适应量化工具标志 color_qp_adjust_flag

二值变量。值为 0 表示不使用点云自适应量化；值为 1 表示使用点云自适应量化。

反射率最大变换阶数 refl_max_trans_numRefGolombNum

无符号整数，取值范围为[0,8]。用于表示当前反射率预测变换中的最大变换阶数。

反射率同组共用第一个点预测值标志 refl_group_pred_flag

二值变量。值为 0 表示不使用反射率同组共用第一个点预测值；值为 1 表示使用反射率同组共用第一个点预测值。

最大延迟限制参数 coeff_length_control_log2_minus8

无符号整数，取值范围为[0,9]。用于表示在属性变换编码中变换参数的最大延迟点数。coeffLengthControl =(1 << (coeff_length_control_log2_minus8 + 8))。最大延迟点数为 maxLatency = maxNumofCoeff * coeffLengthControl，maxLatency 的最大值为 2^{17}。

跨类型的属性预测 cross_attr_type_pred

二值变量。值为 0 表示不允许跨类型的属性预测；值为 1 表示允许跨类型的属性预测。若当前点云只存在一组颜色和一组反射率属性时，可使用跨类型的属性预测工具。

属性编码顺序 attr_coding_order

二值变量。值为 0 表示先编码颜色，再编码反射率；值为 1 表示先编码反射率，再编码颜色。

跨类型的属性预测权重参数 1 cross_attr_type_pred_param1

15 位无符号整数。用于跨类型的属性预测中，计算几何信息距离和属性信息距离的权重参数 1。

跨类型的属性预测权重参数 2 cross_attr_type_pred_param2

21 位无符号整数。用于跨类型的属性预测中，计算几何信息距离和属性信息距离的权重参数 2。

7.2.5 点云帧头

帧序号 frame_idx

无符号整数，取值范围为[0,65535]。用于表示点云帧在点云序列中的序号，每个点云帧的序号为前一个点云帧序号加 1，第一个点云帧序号为 0。当某个点云帧序号达到 65535 后，其后一帧点云帧序号重置为 0。

帧内点云片数量减一 frame_num_slice_minus1

无符号整数，取值范围为[0,65535]。当前点云帧内点云片的数量为 slice_num = frame_num_slice_minus1 + 1。用于判断当前点云帧是否已解码所有点云片。

对数几何宏块的节点大小减一 lcu_node_size_log2_minus1

无符号整数。值为 0 表示关闭块结构编码；值大于 0 表示打开块结构编码，并且定义了宏块的对数几何节点大小为 lcu_node_size_log2 = lcu_node_size_log2_minus1 + 1。

点云帧内点数高 16 位部分 geom_num_points_upper 和点云帧内点数低 16 位部分 geom_num_points_lower

16 位无符号整数。表示点云帧内经可能进行的几何量化和去重后点的个数 32 位无符号整数的高 16 位和低 16 位。点云帧内点的个数即为 geom_num_points= geom_num_points_upper << 16 + geom_num_points_lower。

包围盒原点 x 坐标高 16 位部分 bounding_box_offset_x_upper 和包围盒原点 x 坐标低 16 位部分 bounding_box_offset_x_lower

16 位无符号整数。表示包围盒 x 坐标 32 位有符号整数的高 16 位和低 16 位。包围盒原点 x 坐标为 bounding_box_offset_x = bounding_box_offset_x_upper << 16 + bounding_box_offset_x_ lower。

包围盒原点 y 坐标高 16 位部分 bounding_box_offset_y_upper 和包围盒原点 y 坐标低 16 位部分 bounding_box_offset_y_lower

16 位无符号整数。表示包围盒 y 坐标 32 位有符号整数的高 16 位和低 16 位。包围盒原点 y 坐标为 bounding_box_offset_y = bounding_box_offset_y_upper << 16 + bounding_box_offset_y_ lower。

包围盒原点 z 坐标高 16 位部分 bounding_box_offset_z_upper 和包围盒原点 z 坐标低 16 位部分 bounding_box_offset_z_lower

16 位无符号整数。表示包围盒 z 坐标 32 位有符号整数的高 16 位和低 16 位。包围盒原点 z 坐标为 bounding_box_offset_z = bounding_box_offset_z_upper << 16 + bounding_box_offset_z_ lower。

包围盒宽度高 16 位部分 bounding_box_size_width_upper 和包围盒宽度低 16 位部分 bounding_box_size_width_lower

16 位无符号整数。表示包围盒宽度 32 位无符号整数的高 16 位和低 16 位。包围盒宽度为 bounding_box_size_width = bounding_box_size_width_upper << 16 + bounding_box_size_width_lower。

包围盒高度高 16 位部分 bounding_box_size_height_upper 和包围盒高度低 16 位部分 bounding_box_size_height_lower

16 位无符号整数。表示包围盒高度 32 位无符号整数的高 16 位和低 16 位。包围盒高度为 bounding_box_size_height = bounding_box_size_height_upper << 16 + bounding_box_size_height_lower。

包围盒深度高 16 位部分 bounding_box_size_depth_upper 和包围盒深度低 16 位部分 bounding_box_size_depth_lower

16 位无符号整数。表示包围盒深度 32 位无符号整数的高 16 位和低 16 位。包围盒深度为 bounding_box_size_depth = bounding_box_size_depth_upper << 16 + bounding_box_size_depth_lower。

7.2.6 几何片头

片标号 slice_id

无符号整数,取值范围为[0,65535]。表示片的标号。定义的片的标号在几何头信息中有相同的片的标号。

片几何上下文模式标志 context_mode

二值变量。点云片根据值选择不同的几何八叉树编码中的上下文模式。

在八叉树划分前最大四叉树/二叉树划分数量 max_num_implicit_qtbt_before_ot

无符号整数。表示在几何隐式划分中，在八叉树划分之前，最大被允许的四叉树或者二叉树划分数量。

四叉树/二叉树划分的最小尺寸 min_size_implicit_qtbt

无符号整数。表示在几何隐式划分中，四叉树或者二叉树划分最小被允许的划分尺寸。当几何隐式划分标志为 1 时，max_num_implicit_qtbt_before_ot 和 min_size_implicit_qtbt 需要根据根节点对数尺寸进行限制。

片几何孤立点编码模式标志 gsh_single_mode_flag

二值变量。值为 0 表示关闭当前点云片几何孤立点编码模式；值为 1 表示打开当前点云片几何孤立点编码模式。当码流中没有该符号时，其默认值为 0。

片几何平面模式标志 planar_mode

二值变量。值为 0 表示关闭当前点云片平面编码模式；值为 1 表示打开当前点云片几何平面模式。

片包围盒原点 x 坐标高 16 位部分 slice_bounding_box_offset_x_upper 和片包围盒原点 x 坐标低 16 位部分 slice_bounding_box_offset_x_lower

16 位无符号整数。表示片包围盒 x 坐标 32 位有符号整数的高 16 位和低 16 位。片包围盒原点 x 坐标为 slice_bounding_box_offset_x = slice_bounding_box_offset_x_upper << 16 + slice_bounding_box_offset_x_lower。

片包围盒原点 y 坐标高 16 位部分 slice_bounding_box_offset_y_upper 和片包围盒原点 y 坐标低 16 位部分 slice_bounding_box_offset_y_lower

16 位无符号整数。表示片包围盒 y 坐标 32 位有符号整数的高 16 位和低 16 位。片包围盒原点 y 坐标为 slice_bounding_box_offset_y = slice_bounding_box_offset_y_upper << 16 + slice_bounding_box_offset_y_lower。

片包围盒原点 z 坐标高 16 位部分 slice_bounding_box_offset_z_upper 和片包围盒原点 z 坐标低 16 位部分 slice_bounding_box_offset_z_lower

16 位无符号整数。表示片包围盒 z 坐标 32 位有符号整数的高 16 位和低 16 位。片包围盒原点 z 坐标为 slice_bounding_box_offset_z = slice_bounding_box_offset_z_upper << 16 + slice_bounding_box_offset_z_lower。

片包围盒 X 方向对数尺寸 slice_bounding_box_sizeXLog2

6 位无符号整数,取值范围为[0,32]。表示片包围盒 X 方向对数尺寸。

片包围盒 Y 方向对数尺寸 slice_bounding_box_sizeYLog2

6 位无符号整数,取值范围为[0,32]。表示片包围盒 Y 方向对数尺寸。

片包围盒 Z 方向对数尺寸 slice_bounding_box_sizeZLog2

6 位无符号整数,取值范围为[0,32]。表示片包围盒 Z 方向对数尺寸。

片所含点数高 16 位部分 slice_num_points_upper 片所含点数低 16 位部分 slice_num_points_lower

16 位无符号整数。表示片所含点数的高 16 位和低 16 位。片所含点数为 slice_num_points = slice_num_points_upper << 16 + slice_num_points_lower。

7.2.7 属性片头

片标号 slice_id

无符号整数，取值范围为[0,65535]。表示片的标号。定义的片的标号在几何头信息中有相同的片的标号。

属性信息索引 attribute_id

无符号整数，取值范围为[0,127]。用于表示属性片使用的属性头中相应类型属性的第 attribute_id 组属性信息，这是一个 0 到 attribute_info_num_set_minus1[attrIdx]+1 间的数字（当属性片为颜色属性片时，attrIdx 对应为 0，当属性片为反射率属性片时，attrIdx 对应为 1）。当该语法元素在码流中不存在的时候，其默认值为 0。

属性量化参数偏移量 qp_offset

有符号整数，取值范围为[-32,32]。用于表示色亮度属性或反射率属性量化参数的片内偏移量。

颜色初始预测变换比例 color_init_pred_trans_ratio

有符号整数，取值范围为[-16,16]。用于表示属性压缩的多层变换算法中，控制构建预测变换树时颜色所用初始距离阈值的大小。

反射率初始预测变换比例 refl_init_pred_trans_ratio

有符号整数，取值范围为[-16,16]。用于表示属性压缩的多层变换算法中，控制构建预测变换树时反射率所用初始距离阈值的大小。

点云几何量化前后点数比值 color_qp_adjust_scalar

无符号整数，取值范围为[0,127]。用于表示点云自适应量化工具的距离阈值。

7.2.8 几何片信息

7.2.8.1 几何片数据语义

八叉树层级孤立点模式标志 single_point_eligible_flag_per_depth[depth]

二值变量。值为 0 表示在几何八叉树划分深度为 depth 时，该八叉树层节点不满足孤立点模式；值为 1 表示在几何八叉树划分深度为 depth 时，该八叉树层节点满足孤立点模式。对于 depth 为 0 时，该标识符的默认值为 0。

几何树类型 geom_tree_type

二值变量。值为 0 表示八叉树编码；值为 1 表示预测树编码。

宏块(LCU)点数值占用字节数 num_bits_in_lcu_num_points

5 位无符号整数。表示宏块中点数值所用的位数。

宏块(LCU)点数值字节 lcu_num_points[i]

二值变量。表示宏块点数值的第 i 位。

几何残差为零标志 is_geom_residual_zero[n][k]

二值变量。表示第 n 个几何残差的第 k 分量是否为 0，k = 0,1,2。值为 0 表示残差值不等于 0；值为 1 表示残差值等于 0。

几何残差相对符号序号 geom_residual_ord_rel_sign[n][j]

二值变量。表示第 n 个几何残差的相对符号在所有可能状态中的序号的第 j 位，j = 0,1,2。可能状态最多为 geom_residual_max_rel_sign[n]个。这里 geom_residual_max_rel_sign[n]是 3 位无符号整数,依据 9.2.6 确定第 n 个几何残差的相对符号的所有可能状态的对应值。

几何残差绝对值减一除二占用位数 num_bits_geom_residual_minus1[n][k]

4 位无符号整数。表示第 n 个几何残差的第 k 分量绝对值减一除二所占用的位数的值，k = 0,1,2。

几何残差绝对值减一除二 geom_residual_minus1_div2[n][k][j]

二值变量。表示第 n 个几何残差的第 k 分量绝对值减一除二值的第 j 位的值，k = 0,1,2。

几何残差绝对值减一除二余数 geom_residual_minus1_div2_remain[n][k]

二值变量。表示第 n 个几何残差的第 k 分量绝对值减一除二的余数值，k = 0,1,2。

终止位 termination_bit_one

终止位。值为 1。

7.2.8.2 几何片节点语义

几何孤立点模式标志符 geom_single_flag

二值变量。值为 0 当前节点不按照孤立点模式编码；值为 1 表示当前节点按照孤立点模式编码。如果该语法元素不在码流中出现，其缺省值为 0。

几何占位码 occupancy_code

8 位无符号整数。8 个位分别表示当前节点的 8 个子节点是否被占用，某一位值为 0 表示该位对应的子节点没有被占用，值为 1 表示该位对应的子节点被占用。

孤立点 X 方向相对坐标的第 i 位 point_offset_x[i]

在孤立点编码中，表示其相对于其所属节点 X 坐标的相对位置的第 i 位的值。

孤立点 Y 方向相对坐标的第 i 位 point_offset_y[i]

在孤立点编码中，表示其相对于其所属节点 Y 坐标的相对位置的第 i 位的值。

孤立点 Z 方向相对坐标的第 i 位 point_offset_z[i]

在孤立点编码中，表示其相对于其所属节点 Z 坐标的相对位置的第 i 位的值。

几何子节点中重复点标志 num_duplicated_points_eq1

表示当前几何子节点中重复点数目是否为 1。值为 0 表示子节点中包含多于 1 个点；值为 1 表示子节点中只有 1 个点。

几何子节点中重复点数目减二 num_duplicated_points_minus2

表示当前几何子节点中重复点数目为 num_duplicated_points_minus2 + 2。该语法元素由 0 阶指数哥伦布二值化后编码。

7.2.9 属性片数据语义

零游程为零标志 zero_run_length_eq0

二值变量。值为 0 表示零游程值不为 0；值为 1 表示零游程值为 0。当该语法元素不在码流中，其默认值为 0。

零游程减去一 zero_run_length_minus1

无符号整数。表示零游程值减去一，当该语法元素不在码流中，其默认值为 0。该语法元素由二阶指数哥伦布二值化后编码。

颜色分量残差符号 color_component_sign[k]

二值变量。值为 0 表示颜色第 k 分量残差值为负；值为 1 表示颜色第 k 分量残差值为正，k = 0,1,2。

颜色值为零标志 color_eq0

二值变量。值为 0 表示颜色值不为 0；值为 1 表示颜色值为 0。

颜色值为一标志 color_eq1

二值变量。值为 0 表示颜色值不等于 1；值为 1 表示颜色值等于 1。

颜色值奇偶标志 color_parity

二值变量。值为 0 表示颜色值减去二或颜色值减去一再减去二是偶数；值为 1 表示颜色值减去二或颜色值减去一再减去二是奇数。

颜色值减去二除以二等于零 color_minus2_div2_eq0

二值变量。值为 0 表示颜色值减去二除以二不等于 0；值为 1 表示颜色值减去二除以二等于 0。

颜色值减去二除以二大于零 color_minus2_div2_minus1

无符号整数。表示颜色值减去二除以二后再减去一的值。该语法元素由 color_golomb_num 阶指数哥伦布二值化后编码。

颜色值减去一为零标志 color_minus1_eq0

二值变量。值为 0 表示颜色值减去一不为 0；值为 1 表示颜色值减去一为 0。

颜色值减去一为一标志 color_minus1_eq1

二值变量。值为 0 表示颜色值减去一不等于 1；值为 1 表示颜色值减去一等于 1。

颜色值减去一再减去二除以二等于零 color_minus1_minus2_div2_eq0

二值变量。值为 0 表示颜色值减去一再减去二除以二不等于 0；值为 1 表示颜色值减去一再减去二除以二等于 0。

颜色值减去一再减去二除以二大于零 color_minus1_minus2_div2_minus1

无符号整数。表示颜色值减去一再减去二除以二后再减去一的值。该语法元素由 color_golomb_num 阶指数哥伦布二值化后编码。

反射率残差符号 residual_sign

二值变量。值为 0 表示反射率残差值为负；值为 1 表示反射率残差值为正。

反射率残差绝对值减一的奇偶标志 refl_minus1_parity

二值变量。值为 0 表示反射率残差绝对值减一后为偶数；值为 1 表示反射率残差绝对值减一后为奇数。当该语法元素不出现的时候，其默认值为 0。

反射率残差绝对值减一除二等于零标志 refl_minus1_div2_eq0

二值变量。值为 0 表示反射率残差绝对值减一再除二后值不等于 0；值为 1 表示反射率残差绝对值减一再除二后值等于 0。

反射率残差绝对值减一除二等于一标志 refl_minus1_div2_eq1

二值变量。值为 0 表示反射率残差绝对值减一再除二后值不等于 1；值为 1 表示反射率残差绝对值减一再除二后值等于 1。

反射率残差绝对值减一除二后减二 refl_minus1_div2_minus2

无符号整数。表示反射率残差绝对值减一除二后再减二。该语法元素由 refl_golomb_num 阶指数哥伦布二值化后编码。

颜色第一分量残差值为零标志 color_first_comp_zero

二值变量。值为 0 表示颜色第一分量残差值不为 0；值为 1 表示颜色第一分量残差值为 0。当 order_switch = 0,颜色第一分量是 Y（颜色空间是 YUV）或者 R（颜色空间是 RGB）；当 order_switch = 1，颜色第一分量是 U（颜色空间是 YUV）或者 G（颜色空间是 RGB）。

颜色第二分量残差值为零标志 color_second_comp_zero

二值变量。值为 0 表示颜色第二分量残差值不为 0；值为 1 表示颜色第二分量残差值为 0。当 order_switch = 0,颜色第二分量是 U（颜色空间是 YUV）或者 G（颜色空间是 RGB）；当 order_switch = 1，颜色第二分量是 Y（颜色空间是 YUV）或者 R（颜色空间是 RGB）。

终止位 termination_bit_one

终止位。值为 1。

7.2.10 用户数据

用户数据字节 user_data

8 位整数。用户数据的含义由用户自行定义。

用户数据起始码 user_data_start_code

位串‘0x00000105’。标识用户数据的开始。用户数据连续存放，直到下一个起始码。

8 解析过程

8.1 k 阶指数哥伦布码

解析 k 阶指数哥伦布码时，首先从位流的当前位置开始寻找第一个非零位，并将找到的零位个数记为 leadingZeroBits，然后根据 leadingZeroBits 计算 CodeNum。用伪代码描述如下：

```
leadingZeroBits = -1
for (b = 0; !b; leadingZeroBits++)
  b = read_bits(1)
CodeNum = 2^(leadingZeroBits + k) – 2^k + read_bits(leadingZeroBits + k)
```

表 40 给出了 0 阶、1 阶、2 阶和 3 阶指数哥伦布码的结构。指数哥伦布码的位串分为“前缀”和“后缀”两部分。前缀由 leadingZeroBits 个连续的 0 和一个 1 构成。后缀由 leadingZeroBits 加 k 个位构成，即表中的 x_i 串，x_i 的值为 0 或 1。

表40　k 阶指数哥伦布码表

阶数	码字结构	CodeNum 取值范围
k = 0	1	0
	0 1 x0	1～2
	0 0 1 x1 x0	3～6
	0 0 0 1 x2 x1 x0	7～14
	……	……
k = 1	1 x0	0～1
	0 1 x1 x0	2～5
	0 0 1 x2 x1 x0	6～13
	0 0 0 1 x3 x2 x1 x0	14～29
	……	……
k = 2	1 x1 x0	0～3
	0 1 x2 x1 x0	4～11
	0 0 1 x3 x2 x1 x0	12～27
	0 0 0 1 x4 x3 x2 x1 x0	28～59
	……	……
k = 3	1 x2 x1 x0	0～7
	0 1 x3 x2 x1 x0	8～23
	0 0 1 x4 x3 x2 x1 x0	24～55
	0 0 0 1 x5 x4 x3 x2 x1 x0	56～119
	……	……

8.2　ue(v)和 se(v)的解析过程

ue(v)和 se(v)描述的语法元素使用 0 阶指数哥伦布码，其解析过程为：

ue(v)：语法元素的值等于 CodeNum；

se(v)：根据表 41 给出的有符号指数哥伦布码的映射关系求语法元素的值。

表41　se(v)与 CodeNum 的映射关系

CodeNum 值	语法元素值
0	0
1	1
2	- 1
3	2
4	- 2
5	3
6	- 3
k	$(-1)^{k+1} \times \left\lceil \frac{k}{2} \right\rceil$

8.3 ae(v)的解析过程

8.3.1 初始化二元符号模型

解码器应保存全部二元符号模型,每个二元符号模型 ctx 需要初始化三个变量 mps、cycno 和 lgPmps。mps 的位宽为 1 位，cycno 的位宽为 2 位，lgPmps 的位宽为 11 位。mps 和 cycno 的值应初始化为 0，lgPmps 的值应初始化为 1023。

8.3.2 初始化熵编码解码器

rS1、rT1、bFlag、cFlag、valueS、boundS、valueT 和 valueD 是用于熵编码解码器的变量。boundS 是一个大于 0 的整数。valueD 的值为 0 或 1，bFlag 的值为 0 或 1，cFlag 的值为 0 或 1。valueS 和 boundS 的位宽是大于或等于 Log(boundS+1)的最小整数，rS1 的位宽是大于或等于 Log(boundS+2)的最小整数，rT1 的位宽是 8 位，valueT 的位宽是 9 位。rS1 的值应初始化为 0，rT1 的值应初始化为 0xFF。如果 boundS 的值为 254，则 valueS 和 rS1 的位宽是 8 位。

（注：valueS 记录了预先连续读进多少位才会使 valueT 的最高位为 1。这一功能可用于快速读取位流及并行解码。在极端的状况下，valueS 的值有可能超过 16 位可表示的范围。boundS 限制了连续读进 0 的个数，从而对 valueS 的位宽进行合理的控制。）

valueS、valueT 和 valueD 的初始化过程用伪代码描述如下：

```
valueS = 0
valueT = read_bits(9)
valueD = 1
```

8.3.3 二元符号串解析

8.3.3.1 概述

解析二元符号串的步骤如下：

a) 设二元符号的索引号 binIdx 的值为-1，二元符号串为空。

b) binIdx 的值加 1，然后进行以下操作：

1) 置 BypassFlag 和 StuffingBitFlag 的值为 0，根据 binIdx 得到每个二元符号对应的唯一的 ctxIdx，并根据 ctxIdx 导出二元符号模型 ctx（见 8.3.3.2 ）；

2) 解析当前二元符号（见 8.3.3.3 ）；

3) 将 2)得到的二元符号加入二元符号串的尾部，得到更新的二元符号串；

4) 将 3)得到的二元符号串与 8.3.4 中对应的表格进行比较。如果该二元符号串与表格中某个二元符号串相匹配，则完成二元符号串的解析；否则回到步骤 1)，继续解析下一个二元符号。

8.3.3.2 确定二元符号模型

8.3.3.2.1 概述

二元符号模型 ctx 等于 ctxArray[ctxIdx]，其中 ctxArray 是保存二元符号模型的数组，ctxIdx 是数组的索引值，语法元素的每个二元符号的 ctxIdx = ctxIdxInc + ctxIdxStart。各语法元素对应的 ctxIdxStart 及每个二元符号对应的 ctxIdxInc 见表 42。

表42　语法元素对应的 ctxIdxStart 和 ctxIdxInc

语法元素	ctxIdxInc	ctxIdxStart	ctx 的数量
occupancy	见 8.3.3.2.2	0	194
num_duplicated_points_eq1	0	194	1
single_point_eligible_flag_per_depth	0	195	1
geom_single_flag	0	196	1
occupancy	见 8.3.3.2.2	197	290
geom_tree_type	0	487	1
is_geom_residual_zero	见 9.2.6	488	3
num_bits_geom_residual_minus1	见 9.2.6	491	30
geom_residual_ord_rel_sign	见 9.2.6	521	3
zero_run_length_eq0	0	524	1
zero_run_length_minus1	见 8.3.3.2.4	525	5
color_eq0	见 8.3.3.2.3	530	8
color_eq1	见 8.3.3.2.3	538	4
color_minus2_div2_eq0	见 8.3.3.2.3	542	4
color_minus2_div2_minus1	见 8.3.3.2.5	546	6
color_parity	见 8.3.3.2.3	552	4
color_minus1_eq0	见 8.3.3.2.3	558	6
color_minus1_eq1	见 8.3.3.2.3	564	3
color_minus1_minus2_div2_eq0	见 8.3.3.2.3	567	3
color_minus1_minus2_div2_minus1	见 8.3.3.2.5	546	6
color_first_comp_zero	0	570	1
color_second_comp_zero	0	571	1
refl_minus1_parity	见 8.3.3.2.3	552	1
refl_minus1_div2_eq0	见 8.3.3.2.3	538	1
refl_minus1_div2_eq1	见 8.3.3.2.3	542	1
refl_minus1_div2_minus2	见 8.3.3.2.5	546	4

8.3.3.2.2　确定 occupancy 的 ctxIdxInc

occupancy 语法元素包含 8 个位 $b_7b_6b_5b_4b_3b_2b_1b_0$，分别对应一个节点的八个子块（子节点），这八个子块是按照莫顿码排序的。

根据已解码的节点信息，首先确定 ctxIdxInc 的偏移值 ctx_offset，偏移值有 0、1 和 2 这三个值。当 planar_mode 为 1 时，执行以下操作，否则 ctx_offset 默认为 0。

```
if (LowOccNum > 1 && HighOccNum < 1 && !!(childIdx & 1) && (!(codedOccupancyCode & 0xaa)))
  ctx_offset = 0
else if (LowOccNum < 1 && HighOccNum > 1 && (!(childIdx & 1)) && (!(codedOccupancyCode & 0x55)))
  ctx_offset = 1
else
  ctx_offset = 2
```

其中 LowOccNum 和 HighOccNum 表示当前节点的邻居节点的子节点中，位于较低层的被占据子节点个数和位于较高层的被占据子节点个数，由 9.2.3.2 得到。childIdx 表示待解码子块的标号（childIdx = 0,1,2,…,7），codedOccupancyCode 表示当前节点中已解码的占位码。

当 context_mode 等于 0 时，根据以下方法确定 occupany 的 ctxIdxInc，其中 bit_ctx、ChildNei、ctx_ParentIdx、ctxChild 和 ctxFromMemory 由 9.2.3.2 得到：

a) 若 ChildNei 不为 0，则 ctxIdxInc = ctx_ParentIdx *（ChildNei-1）。

b) 若 ChildNei 为 0，则判断 ctxChild 的值来确定 occupany 的 ctxIdxInc：

1) 若 ctxChild 不为 0，则 ctxIdxInc = ctxFromMemory * ctx_ParentIdx。

2) 若 ctxChild 为 0，则 ctxIdxInc = bit_ctx+ctx_ParentIdx。

当 context_mode 等于 1 时，根据以下方法确定 occupancy 的 ctxIdxInc，其中 bit_ctx 与 context 由 9.2.3.2 得到：

ctxIdxInc = bit_ctx + context。

对 ctxIdxInc 进行更新，ctxIdxInc = (ctx_offset < 2) ? ctx_offset : (ctxIdxInc+ctx_offset)。

8.3.3.2.3 确定 color_eq0, color_eq1, color_minus2_div2_eq0, color_parity, color_minus1_eq0, color_minus1_eq1, color_minus1_minus2_div2_eq0 的 ctxIdxInc

表43 color 语法元素对应的 ctxIdxInc

语法元素	ctxIdxInc	第一属性系数为 0	第二属性系数为 0	第一属性系数小于等于第二属性系数	对应的属性系数
color_eq0	0	√	×		对应的属性是颜色中的第三属性系数
	1	×			对应的属性是颜色中的第二属性系数
	2	×		√	对应的属性是颜色中的第三属性系数
	3	√	×		对应的属性是重复点的颜色中的第三属性系数
	4	×			对应的属性是重复点的颜色中的第二属性系数
	5	×		√	对应的属性是重复点的颜色中的第三属性系数
	6	×		×	对应的属性颜色中的第三属性系数
	7	×		×	对应的属性是重复点的颜色中的第三属性系数

表 43 color 语法元素对应的 ctxIdxInc（续）

语法元素	ctxIdxInc	第一属性系数为 0	第二属性系数为 0	第一属性系数小于等于第二属性系数	对应的属性系数
color_eq1	0	√	×		对应的属性是颜色中的第三属性系数
	1	×			对应的属性是颜色中的第二属性系数
	2	×		√	对应的属性是颜色中的第三属性系数
	3	×		×	对应的属性是颜色中的第三属性系数
color_minus2_div2_eq0	0	√	×		对应的属性是颜色中的第三属性系数
	1	×			对应的属性是颜色中的第二属性系数
	2	×		√	对应的属性是颜色中的第三属性系数
	3	×		×	对应的属性是颜色中的第三属性系数
color_minus1_eq0	0	√	√		对应的属性是颜色中的第三属性系数
	1	×			对应的属性是颜色中的第一属性系数
	2	√	×		对应的属性是颜色中的第二属性系数
	3	√	√		对应的属性是重复点的颜色中的第三属性系数
	4	×			对应的属性是重复点的颜色中的第一属性系数
	5	√	×		对应的属性是重复点的颜色中的第二属性系数
color_minus1_eq1	0	√	√		对应的属性是颜色中的第三属性系数
	1	×			对应的属性是颜色中的第一属性系数
	2	√	×		对应的属性是颜色中的第二属性系数

表 43 color 语法元素对应的 ctxIdxInc（续）

语法元素	ctxIdxInc	第一属性系数为 0	第二属性系数为 0	第一属性系数小于等于第二属性系数	对应的属性系数
color_minus1_minus2_div2_eq0	0	√	√		对应的属性是颜色中的第三属性系数
	1	×			对应的属性是颜色中的第一属性系数
	2	√	×		对应的属性是颜色中的第二属性系数
color_parity	0	√	×		对应的属性是颜色中的第三属性系数
		√	√		对应的属性是颜色中的第三属性系数
	1	×			对应的属性是颜色中的第二属性系数
		×			对应的属性是颜色中的第一属性系数
	2	×		√	对应的属性是颜色中的第三属性系数
		√	×		对应的属性是颜色中的第二属性系数
	3	×		×	对应的属性是颜色中的第三属性系数

8.3.3.2.4 确定 zero_run_length_minus1 的 ctxIdxInc

根据以下方法确定 zero_run_length_minus1 的 ctxIdxInc：

设二值化指数哥伦布码为 $b_0b_1\cdots b_mb_{m+1}\cdots b_{2m+k}$，其中 $b_0b_1\cdots b_m$ 为前缀码部分 prefix，$b_{m+1}\cdots b_{2m+k}$ 为后缀码部分 suffix，k 为指数哥伦布阶数；

zero_run_length_minus1 指数哥伦布码的前缀的 ctxIdxInc 为 0、1 和 2，指数哥伦布码的后缀的 ctxIdxInc 为 3 和 4，对应 ctxIdxInc 如下表 44 所示。

表44 语法元素对应的 ctxIdxInc

位置	二值化变量	ctxIdxInc
前缀码	b_0	0
	b_1	1
	$b_2\cdots b_m$	2
后缀码	b_{m+1}	3
	$b_{m+2}\cdots b_{2m+k}$	4

8.3.3.2.5 确定 refl_minus1_div2_minus2, color_minus2_div2_minus1, color_minus1_minus2_div2_minus1 的 ctxIdxInc

根据以下方法确定 color_minus2_div2_minus1、color_minus1_minus2_div2_minus1 和 refl_minus1_div2_minus2 的 ctxIdxInc：

将 color_minus2_div2_minus1、color_minus1_minus2_div2_minus1、refl_minus1_div2_minus2 由 k 阶指数哥伦布二值化，其中 k 由 9.3.10 属性解码过程计算所得；设二值化后为 $b_0b_1\cdots b_mb_{m+1}\cdots b_{2m+k}$，其中 $b_0b_1\cdots b_m$ 为前缀码部分 prefix，$b_{m+1}\cdots b_{2m+k}$ 为后缀码部分 suffix，k 为指数哥伦布阶数；

如表 45 所示，refl_minus1_div2_minus2 的前四个 bin 采用基于上下文编码方式（基于上下文编码的 bin 的上下文索引对应情况如表 46 所示），其余 bin 采用旁路编码方式；

如表 45 所示，color_minus2_div2_minus1 的前四个 bin 采用基于上下文编码方式（基于上下文编码的 bin 的上下文索引对应情况如表 46 所示），其它 bin 采用旁路编码方式；

如表 45 所示，color_minus1_minus2_div2_minus1 的前四个 bin 采用基于上下文编码方式（基于上下文编码的 bin 的上下文索引对应情况如表 46 所示），其余 bin 采用旁路编码方式；

refl_minus1_div2_minus2,color_minus2_div2_minus1、color_minus1_minus2_div2_minus1 共享上下文。

表45 语法元素的 bin 的编码方式

语法元素	binIdx				
	0	1	2	3	>= 4
refl_minus1_div2_minus2	ctx	ctx	ctx	ctx	bypass
color_minus2_div2_minus1	ctx	ctx	ctx	ctx	bypass
color_minus1_minus2_div2_minus1	ctx	ctx	ctx	ctx	bypass

表46 语法元素 ctxctxIdxInc

语法元素	binIdx					
	0	1	[2,m]	m+1	m+2	[m+3, 2m+k]
refl_minus1_div2_minus2	0	1	2	3	4	5
color_minus2_div2_minus1[0]	0	0	0	3	3	3
color_minus2_div2_minus1[1]	1	1	1	4	4	4
color_minus2_div2_minus1[2]	2	2	2	5	5	5
color_minus1_minus2_div2_minus1 [0]	0	0	0	3	3	3
color_minus1_minus2_div2_minus1 [1]	1	1	1	4	4	4
color_minus1_minus2_div2_minus1 [2]	2	2	2	5	5	5

8.3.3.3 二元符号解析

8.3.3.3.1 解析过程

二元符号的解析过程如下：

第一步，解析二元符号值 binVal：

a) 如果 BypassFlag 的值为 1，执行 decode_bypass 过程（见 8.3.3.3.4 ）；

b) 否则，如果 StuffingBitFlag 的值为 1，则执行 decode_aec_stuffing_bit 过程（见 8.3.3.3.3 ）；

c) 否则，令 cFlag 为 1，执行 decode_decision 过程（见 8.3.3.3.2 ）。

第二步，如果 binVal 的值为 0，则二元符号为 0；如果 binVal 的值为 1，则二元符号为 1。

8.3.3.3.2 decode_decision 过程

decode_decision 过程的输入是 bFlag、cFlag、rS1、rT1、valueS、valueT、valueD 以及上下文模型 ctx。decode_decision 过程的输出是二元符号值 binVal。decode_decision 过程用伪代码描述如下：

```
decode_decision( ) {
    predMps = 0
    if (valueD || (bFlag == 1 && rS1 == boundS )) {
                    rS1 = 0
            valueS = 0
        while (valueT < 0x100 && valueS < boundS) {
            valueS++
            valueT = (valueT << 1) | read_bits(1)
            }
            if (valueT < 0x100)
                bFlag = 1
            else
                bFlag = 0
            valueT = valueT & 0xFF
        }
        if (rT1) {
            rS2 = rS1
            rT2 = rT1 - 1
        }
        else {
            rS2 = rS1 + 1
            rT2 = 255
        }
        if (rS2 > valueS || (rS2 == valueS && valueT >= rT2) && bFlag == 0) {
            binVal = ! predMps
            if (rS2 == valueS)
                valueT = valueT – rT2
            else
                valueT = 256 + ((valueT << 1) | read_bits(1)) – rT2
            valueT = (valueT << 8) | read_bits(8)
            rT1 = 0
            valueD = 1
        }
         else {
             binVal = predMps
```

```
                rS1 = rS2
                rT1 = rT2
                valueD = 0
            }
        return (binVal)
}
```

8.3.3.3.3 decode_aec_stuffing_bit 过程

decode_aec_stuffing_bit 过程的输入是 bFlag、cFlag、rS1、rS2、valueS、valueT 和 valueD。decode_aec_stuffing_bit 过程的输出是二元符号值 binVal。ctx0 是二元符号模型，令 ctx0->lgPmps 等于 4，ctx0->mps 等于 0。令 cFlag 等于 0，ctx 等于 ctx0，带入 decode_decision 过程实现 decode_aec_stuffing_bit 过程。decode_aec_stuffing_bit 过程用伪代码描述如下：

```
decode_aec_stuffing_bit( ) {
    predMps = 0
    if (valueD || (bFlag == 1 && rS1 == boundS)) {
                rS1 = 0
            valueS = 0
        while (valueT < 0x100 && valueS < boundS) {
            valueS++
            valueT = (valueT << 1) | read_bits(1)
            }
            if (valueT < 0x100)
                bFlag = 1
            else
                bFlag = 0
            valueT = valueT & 0xFF
        }
        if (rT1) {
            rS2 = rS1
            rT2 = rT1 - 1
        }
        else {
            rS2 = rS1 + 1
            rT2 = 255
        }
        if (rS2 > valueS || (rS2 == valueS && valueT >= rT2) && bFlag == 0) {
            binVal = ! predMps
            if (rS2 == valueS)
                valueT = valueT – rT2
            else
                valueT = 256 + ((valueT << 1) | read_bits(1)) – rT2
            valueT = (valueT << 8) | read_bits(8)
```

```
                rT1 = 0
                valueD = 1
            }
             else {
                  binVal = predMps
                  rS1 = rS2
                  rT1 = rT2
                  valueD = 0
              }
             return (binVal)
}
```

8.3.3.3.4 decode_bypass 过程

decode_bypass 过程的输入是 bFlag、cFlag、rS1、rS2、valueS、valueT 和 valueD。decode_bypass 过程的输出是二元符号值 binVal。ctx1 是二元符号模型，令 ctx1->lgPmps 等于 1024，ctx1->mps 等于 0。令 cFlag 等于 0，ctx 等于 ctx1，带入 decode_decision 过程实现 decode_bypass 过程。decode_bypass 过程用伪代码描述如下：

```
decode_bypass( ) {
            predMps = 0
            if (valueD || (bFlag == 1 && rS1 == boundS)) {
                        if（rS1 == boundS）  {
                                    rS1 = 0
            valueS = 0
                        while (valueT < 0x100 && valueS < boundS) {
                                    valueS++
                                    valueT = (valueT << 1) | read_bits(1)
                        }
                  if (valueT < 0x100)
                      bFlag = 1
                  else
                      bFlag = 0
                  valueT = valueT & 0xFF
    }
             }
            if (valueD || (bFlag == 1 && rS1 == boundS)) {
                  rS1 = 0
                  valueD = 1
                  valueT = (valueT << 1 ) | read_bits(1)
                  if (valueT >=(256 + rT1)) {
                       binVal = !predMps
                       valueT -= (256 + rT1)
                   }
```

```
                else
                binVal = predMps
            }
            else {
                rS2 = rS1+1
                if (rS2 > valueS || (rS2 == valueS && valueT >= rT1) && bFlag == 0) {
                binVal = !predMps
                if (rS2 == valueS)
                    valueT = valueT – rT1
                else
                    valueT = 256 + ((valueT << 1) | read_bits(1)) – rT1
                rS1 = 0
                valueD = 1
              }
              else {
                  binVal = predMps
                  rS1= rS2
                  valueD = 0
               }
            }
            return (binVal)
}
```

8.3.3.3.5 update_ctx 过程

update_ctx 过程的输入是 binVal 和 ctx。update_ctx 过程的输出是更新后的 ctx。update_ctx 过程用伪代码描述如下：

```
update_ctx( ) {
    if (ctx->cycno <= 1)
      cwr = 3
    else if (ctx->cycno == 2)
      cwr = 4
    else
      cwr = 5
    if (binVal != ctx->mps) {
      if (ctx->cycno <= 2)
        ctx->cycno = ctx->cycno + 1
      else
        ctx->cycno = 3
    }
    else if (ctx->cycno == 0)
      ctx->cycno = 1
    if (binVal == ctx->mps)
```

```
        ctx->lgPmps = ctx->lgPmps – (ctx->lgPmps >> cwr) - (ctx->lgPmps >> (cwr+2))
    else {
        switch (cwr) {
            case 3:
                ctx->lgPmps = ctx->lgPmps + 197
                break
            case 4:
                ctx->lgPmps = ctx->lgPmps + 95
                break
            default:
                ctx->lgPmps = ctx->lgPmps + 46
        }
        if (ctx->lgPmps > 1023) {
            ctx->lgPmps = 2047 - ctx->lgPmps
            ctx->mps = !(ctx->mps)
        }
    }
    return (ctx)
}
```

8.3.4 反二值化方法

8.3.4.1 概述

本条定义语法元素的反二值化方法。

8.3.4.2 采用截断一元码的反二值化方法

由二元符号串根据表 47 得到 synElVal 的值。

表47　synElVal 与二元符号串的关系（截断一元码）

synElVal	二元符号串							
0	1							
1	0	1						
2	0	0	1					
3	0	0	0	1				
4	0	0	0	0	1			
5	0	0	0	0	0	1		
…	0	0	0	0	0	0	…	
maxVal-1	0	0	0	0	0	0	…	1
maxVal	0	0	0	0	0	0	…	0
binIdx	0	1	2	3	4	5	…	maxVal-1

8.3.4.3　采用一元码的反二值化方法

由二元符号串根据表 48 得到 synElVal 的值。

表48　synElVal 的值与二元符号串的关系（一元码）

synElVal	二元符号串					
0	1					
1	0	1				
2	0	0	1			
3	0	0	0	1		
4	0	0	0	0	1	
5	0	0	0	0	0	1
…						
binIdx	0	1	2	3	4	5

8.3.4.4　采用标记位的反二值化方法

由二元符号串根据表 49 得到 synElVal 的值。

表49　synElVal 的值与二元符号串的关系（标记位）

synElVal	二元符号串
0	0
1	1
binIdx	0

9 解码过程

9.1 解码过程概述

序列解码过程如下：

第一步，解码序列：

1.1）解码序列头；

1.2）解码几何头；

1.3）若 attr_present_flag = 1,解码属性头；

第二步，解码帧头；

第三步，解码几何片信息：

3.1）解码几何片头；

3.2）解码几何数据（见 9.2 ）；

第四步，若 attr_present_flag = 1,解码属性片信息：

4.1）解码属性片头；若当前属性片头起始码是 0x07,表示颜色对应的点云属性片头；若当前属性片头起始码是 0x08，表示反射率对应的点云属性片头；

4.2）解码属性数据（见 9.3 ）；

第五步，执行以下操作：

5.1）如果遇到几何片头起始码，继续执行第三步；

5.2）如果遇到属性片头起始码，继续执行第四步；

5.3）如果遇到帧起始码，生成重建点云（见 9.4 ），继续执行第二步；

5.4）如果遇到序列结束码，生成重建点云（见 9.4 ）。

9.2 几何数据解码过程

9.2.1 概述

几何数据解码过程如下：

几何划分解码（见 9.2.2 ）。

几何节点解码（见 9.2.3 ）。

几何数据解码过程还包括如下：

宏块几何划分解码（9.2.4 ）。

八叉树宏块几何节点解码（9.2.5 ）。

预测树宏块解码（9.2.6 ）。

如果该宏块的 geom_tree_type = 0，按照 9.2.3 描述的过程进行解码。

如果该宏块的 geom_tree_type = 1，按照 9.2.5 及 9.2.6 描述的过程进行解码。

9.2.2 几何划分解码

首先定义下述变量：

几何划分最大深度 MaxGeometryOctreeDepth：表示几何划分的最大深度值。

几何划分深度 depth：表示几何划分的深度，根节点的深度定义为 0，每划分一次深度值增加 1。

几何划分在 X 方向的深度 depthX：表示几何划分在 X 方向的深度，根节点的深度定义为 0，每划分一次深度值增加 1。

几何划分在 Y 方向的深度 depthY：表示几何划分在 Y 方向的深度，根节点的深度定义为 0，每划分一次深度值增加 1。

几何划分在 Z 方向的深度 depthZ：表示几何划分在 Z 方向的深度，根节点的深度定义为 0，每划分一次深度值增加 1。

几何划分深度下的节点总数 NumNodesAtDepth[depth]：表示在几何划分深度为 depth 下的节点总数。根节点的节点数初始化为 1，即 NumNodesAtDepth[0] = 1。

几何节点中子节点的数目 GeometryNodeOccupancyCnt[depth][xN][yN][zN]：表示在几何划分深度为 depth、三维位置为(xN, yN, zN)的节点中包含的子节点的个数。

几何节点占位信息可用性 AvailNodeOccupancy[depth][xA][yA][zA]：表示待解码节点(xN, yN, zN)在几何划分深度为 depth、三维位置为(xA, yA, zA)的邻域节点的占位信息。

```
AvailNodeOccupancy[depth][xA][yA][zA] =
GeometryNodeOccupancyCnt[depth][xA][yA][zA] == 0 ? 0:
(abs(xA - xN) >> occupancy_search_range_side_log2 == 0 &&
abs(yA - yN) >> occupancy_search_range_side_log2 == 0 &&
abs(zA - zN) >> occupancy_search_range_side_log2 == 0) == 0?
GeometryNodeOccupancyCnt[depth][xA][yA][zA] : 0
```

几何节点在 X 方向的坐标 NodeX[depth][idx]：表示在几何划分至第 depth 层，第 idx 个节点的 X 方向坐标。初始化为 NodeX[0][0] = 0。

几何节点在 Y 方向的坐标 NodeY[depth][idx]：表示在几何划分至第 depth 层，第 idx 个节点的 Y 方向坐标。初始化为 NodeY[0][0] = 0。

几何节点在 Z 方向的坐标 NodeZ[depth][idx]：表示在几何划分至第 depth 层，第 idx 个节点的 Z 方向坐标。初始化为 NodeZ[0][0] = 0。

几何划分在 X 方向最大对数尺寸 MaxNodeSizeXLog2：表示在几何划分中，X 方向的最大尺寸的以 2 为底的对数，即 MaxNodeSizeXLog2 = slice_bounding_box_sizeXLog2。

几何划分在 Y 方向最大对数尺寸 MaxNodeSizeYLog2：表示在几何划分中，Y 方向的最大尺寸的以 2 为底的对数，即 MaxNodeSizeYLog2 = slice_bounding_box_sizeYLog2。

几何划分在 Z 方向最大对数尺寸 MaxNodeSizeZLog2：表示在几何划分中，Z 方向的最大尺寸的以 2 为底的对数，即 MaxNodeSizeZLog2 = slice_bounding_box_sizeZLog2。

几何节点占位码 GeometryNodeOccupancyCode[depth][xN][yN][zN]：表示在几何八叉树划分深度为 depth、三维位置为(xN, yN, zN)的节点占位码。

几何划分类型标识 partitionSkip：3 位标识符。表示当前节点的划分类型为八叉树、四叉树或者二叉树。其中 3 位从高位到低位分别表示 X,Y,Z 方向是否跳过划分，其中 0 表示不跳过划分， 1 表示跳过划分。具体划分类型如下表 50 所示。

表50 partitionSkip 与几何划分类型的对应关系

标识	沿x-y轴四叉树划分	沿x-z轴四叉树划分	沿y-z轴四叉树划分	八叉树划分	沿x轴二叉树划分	沿y轴二叉树划分	沿z轴二叉树划分
partitionSkip	001	010	100	000	011	101	110

变量 partitionSkip 的解码过程如下：

```
partitionSkip = 0
NodeSizeXLog2 = MaxNodeSizeXLog2 – depthX
NodeSizeYLog2 = MaxNodeSizeYLog2 – depthY
NodeSizeZLog2 = MaxNodeSizeZLog2 – depthZ
```

```
MinNodeSizeLog2 = min{NodeSizeXLog2, NodeSizeYLog2, NodeSizeZLog2}
MaxNodeSizeLog2 = max{NodeSizeXLog2, NodeSizeYLog2, NodeSizeZLog2}
if (MinNodeSizeLog2 == MaxNodeSizeLog2)
  min_size_implicit_qtbt = 0
if (max_num_implicit_qtbt_before_ot > depth || min_size_implicit_qtbt == MinNodeSizeLog2) {
  if (NodeSizeXLog2 < MaxNodeSizeLog2)
    partitionSkip |= 4
  if (NodeSizeYLog2 < MaxNodeSizeLog2)
    partitionSkip |= 2
  if (NodeSizeZLog2 < MaxNodeSizeLog2)
    partitionSkip |= 1
}
```

9.2.3 几何节点解码

9.2.3.1 概述

首先定义下述变量：

孤立点 X 方向相对坐标 PointOffsetX：在孤立点编码中，表示其相对于其所属节点 X 坐标的相对位置。

孤立点 Y 方向相对坐标 PointOffsetY：在孤立点编码中，表示其相对于其所属节点 Y 坐标的相对位置。

孤立点 Z 方向相对坐标 PointOffsetZ：在孤立点编码中，表示其相对于其所属节点 Z 坐标的相对位置。

几何划分节点 X 方向对数尺寸 NodeSizeXLog2：表示几何划分中，当前节点在 X 方向的对数尺寸。

几何划分节点 Y 方向对数尺寸 NodeSizeYLog2：表示几何划分中，当前节点在 Y 方向的对数尺寸。

几何划分节点 Z 方向对数尺寸 NodeSizeZLog2：表示几何划分中，当前节点在 Z 方向的对数尺寸。

几何划分子节点 X 方向对数尺寸 ChildNodeSizeXLog2：表示几何划分中，当前节点的子节点在 X 方向的对数尺寸。

几何划分子节点 Y 方向对数尺寸 ChildNodeSizeYLog2：表示几何划分中，当前节点的子节点在 Y 方向的对数尺寸。

几何划分子节点 Z 方向对数尺寸 ChildNodeSizeZLog2：表示几何划分中，当前节点的子节点在 Z 方向的对数尺寸。

当前几何节点的非空子节点个数 GeometryNodeChildrenCnt：当前几何节点的非空子节点个数。

当前几何节点尺寸最小维度 minDime：表示几何划分中，当前节点在 X、Y 和 Z 方向中边长尺寸最小的维度。

当前几何节点及已解码相邻邻居的子节点占位情况 encodedChildNode[idx]：表示当前节点的已解码邻居以及当前节点的子节点的占位情况。当前节点的已解码邻居包括：三个共线节点，空间位置分别为(xN-1, yN-1, zN)，(xN, yN-1, zN-1)和(xN-1, yN, zN-1)；一个共点节点，空间位置为(xN-1, yN-1, zN-1)。idx 的排序是 5 个节点按莫顿顺序，从共点节点开始，至当前节点结束。

几何孤立点模式是否被限制使用的标识 GeomSingleNodeControlFlag：表示当前节点的孤立点模式是否被暂时限制使用，0 表示孤立点模式，1 表示跳过孤立点模式直接进行占位码解码。

几何孤立点模式中的模式标识 GeomSingleNodeControlMode：0 代表正常孤立点编码，1 代表间隔

孤立点编码。GeomSingleNodeControlMode 在解码时初始化为 1。

孤立点模式可用节点个数 singlePointEligibleNodeCnt：表示孤立点模式可用节点的个数。

真实孤立点节点个数 singlePointNodeCnt：表示正常孤立点编码模式下 geom_single_flag 为 1 的节点个数。

被占据子节点数为 1 的节点数 bit_count_equ1_num：表示间隔孤立点编码模式下被占据子节点个数为 1 的节点个数。

滑动窗口中取值为 1 的位的个数 popcnt8(memoryVal)：表示大小为 K(K = 8)的滑动窗口 memoryVal 中的 K 个符号中取值为 1 的个数。

9.2.3.2 占位码解码

解码占位码 OccupancyCode 的过程如下：

```
OccupancyCode = 0
maxNumOccupiedBins[8] = {9, 4, 4, 2, 4, 2, 2,1}
maxCodedBins = maxNumOccupiedBins[partitionSkip]
numCodedBins = 0
for (childIdx = 0; childIdx < 8; ++ childIdx ) {
  if ((partitionSkip & 1) && (childIdx & 1))
    continue
  if ((partitionSkip & 2) && (childIdx & 2))
    continue
  if ((partitionSkip & 4) && (childIdx & 4))
    continue
  bit = 1
  if (OccupancyCode || (numCodedBins < maxCodedBins - 1)) {
    bit = decode_decision( )
  }
  OccupancyCode |= bit << childIdx
  ++numCodedBins
}
```

解码占位码的位的过程需要根据几何上下文模式标志（context_mode）来获取周围已解码节点的占用信息，从而选择需要的上下文。

定义变量 LowOccNum，HighOccNum 并将初始值设为 0。LowOccNum 表示当前节点的邻居节点中，低平面子节点被占据的个数；HighOccNum 表示当前节点的邻居节点中，高平面子节点被占据的个数。当 planar_mode 为 1 时，对于标号为 childIdx 的子节点，执行以下步骤；否则 LowOccNum，HighOccNum 置为 0。获取 LowOccNum、HighOccNum 后，通过 8.3.3.2.2 确定对应的上下文模型。

```
preNodePlanarNum = 0
planarNodeOffset[5][3] = {{-1, 0, 0}, {-2, 0, 0}, {0, -1, 0}, {0, -2, 0}, {-1, -1, 0}}
for (j = 0; j <= 4; j++) {
  neighborOccupancyCode = GeometryNodeOccupancyCode[depth][xN + planarNodeOffset[j][0]][yN +
  planarNodeOffset[j][1]][zN + planarNodeOffset[j][2]]
  neighborOccupancyHighCode = !(neighborOccupancyCode & 0x55) && (!!(neighborOccupancyCode & 0xaa))
```

```
    neighborOccupancyLowCode = (!!(neighborOccupancyCode & 0x55)) && !(neighborOccupancyCode & 0xaa)
    if (neighborOccupancyHighCode || neighborOccupancyLowCode)
      preNodePlanarNum += 1
    for (i = 0; i < 8; i++) {
      LowOccNum += (neighborOccupancyLowCode >> i) & 1
      HighOccNum += (neighborOccupancyHighCode >> i) & 1
    }
  }
  LowOccNum = preNodePlanarNum > 2 ? LowOccNum : 0
  HighOccNum = preNodePlanarNum > 2 ? HighOccNum : 0
```

当 context_mode 等于 0 时，对于标号为 childIdx 的子节点，选择上下文的过程如下：

```
  xC = (xN << !(partitionSkip & 4)) + ( childIdx & 4 == 1)
  yC = (yN << !(partitionSkip & 2)) + ( childIdx & 2 == 1)
  zC = (zN << !(partitionSkip & 1)) + ( childIdx & 1 == 1)
  childNeiX = AvailNodeOccupancy[depth + 1][xC - 1][yC][zC] ? 1 : 0
  childNeiY = AvailNodeOccupancy[depth + 1][xC][yC - 1][zC] ? 1 : 0
  childNeiZ = AvailNodeOccupancy[depth + 1][xC][yC][zC - 1] ? 1 : 0
  ChildNei = childNeiX + (childNeiY << 1) + (childNeiZ << 2)
  parentNeiX = AvailNodeOccupancy[depth][xN + deltaX [childIdx][0]][yN][zN] ? 1 : 0
  parentNeiY = AvailNodeOccupancy[depth][xN][yN + deltaY [childIdx][1]][zN] ? 1 : 0
  parentNeiZ = AvailNodeOccupancy[depth][xN][yN][zN + deltaZ [childIdx][2]] ? 1 : 0
  ctx_ParentIdx = parentNeiX + (parentNeiY << 1) + (parentNeiZ << 2)
  adjChildIdx = adjacentCIdx[i]
  for (idx = 0; idx < 4; ++idx) {
    ctxChild |= !!(encodedChildNode[adjChildIdx[idx] >> 3] & (1 << (adjChildIdx[idx] & 7))) << idx
  }
  ctxChild |= !!(encodedChildNode[minDime2ParentNeiIndex[minDime]] & (1 << (childIdx))) << 4
  ctxChild |= !!(encodedChildNode[3] & (1 << (childIdx))) << 5
  ctxChild |= !!(encodedChildNode[5] & (1 << (childIdx))) << 6
  ctxChild |= !!(encodedChildNode[6] & (1 << (childIdx))) << 7
  ctxChild |= !!(encodedChildNode[1] & (1 << (childIdx))) << 8
  ctxChild |= !!(encodedChildNode[2] & (1 << (childIdx))) << 9
  if (ChildNei) {
    bit_ctx = ctx_combineParentIdx[ctx_ParentIdx]
    context = ChildNei-1
  } else if (ctxChild) {
    bit_ctx = combineSlideWindowIdx[ctx_ParentIdx]
    memoryVal = memoryChannel[ctxChild]
    ctxFromMemory = popcnt8(memoryVal)
    context = ctxFromMemory
  } else {
    bit_ctx = ctx_compute[idx]
```

```
        context = ctx_ParentIdx
    }
```

其中 childIdx 表示待解码子块的标号（childIdx = 0,1,2,…,7）。其中，bit_ctx 表示每种情况下选择的上下文起始地址，ctxChild 表示当前待解码子块相邻块的占位情况，ctx_ParentIdx 表示当前待解码子块父节点的三个共面相邻块（即空间位置分别为(xN-1, yN, zN)，(xN, yN-1, zN)以及(xN, yN, zN-1)）的占用情况。memoryChannel[ctxchild]表示子节点组成的 1024 个上下文状态，ctxFromMemory 表示当前待解码子块的占位情况经过状态转换后最近已解码的 8 个符号的取值，这 1024 个状态的初始值设置为 15，popcnt8(memoryVal)表示在 1024 个状态中 memoryVal 状态对应的 8 个已解码符号中 1 的个数。ctx_combineParentIdx[ctx_ParentIdx],combineSlideWindowIdx[ctx_ParentIdx]表示三个共面父节点组成的上下文索引，ctx_compute[i]表示第 i 个子节点的上下文起始地址。以 bit_ctx + context 表示最终选择的上下文。adjacentCIdx[i]表示各子节点选择的邻居节点的索引。

对于当前待解码占位码的位，其对应的滑动窗口 memoryChannel 的更新方法等于 1 时，如下：

```
    memoryVal = ((memoryVal) << 1) | bit
```

其中 deltaX、deltaY 和 deltaZ 的取值如表 51、表 52 和表 53 所示：

表51　deltaX 查找表

deltaX[childIdx][i]	i = 0	i = 1	i = 2
childIdx = 0	-1	0	0
childIdx = 1	-1	0	0
childIdx = 2	-1	0	0
childIdx = 3	-1	0	0
childIdx = 4	1	0	0
childIdx = 5	1	0	0
childIdx = 6	1	0	0
childIdx = 7	1	0	0

表52　deltaY 查找表

deltaY[childIdx][i]	i = 0	i = 1	i = 2
childIdx = 0	0	-1	0
childIdx = 1	0	-1	0
childIdx = 2	0	1	0
childIdx = 3	0	1	0
childIdx = 4	0	-1	0
childIdx = 5	0	-1	0
childIdx = 6	0	1	0
childIdx = 7	0	1	0

表53 deltaZ 查找表

deltaZ[childIdx][i]	i = 0	i = 1	i = 2
childIdx = 0	0	0	-1
childIdx = 1	0	0	1
childIdx = 2	0	0	-1
childIdx = 3	0	0	1
childIdx = 4	0	0	-1
childIdx = 5	0	0	1
childIdx = 6	0	0	-1
childIdx = 7	0	0	1

当 context_mode 等于 1 时，选择上下文的过程如下：

```
parentNeiX = AvailNodeOccupancyCnt[depth][xN + deltaX[childIdx][0]][yN][zN] ? 1 : 0
parentNeiY = AvailNodeOccupancyCnt[depth][xN][yN + deltaY[childIdx][1]][zN] ? 1 : 0
parentNeiZ = AvailNodeOccupancyCnt[depth][xN][yN][zN + deltaZ[childIdx][2]] ? 1 : 0
ctx26Parent = parentNeiX + (parentNeiY << 1) + (parentNeiZ << 2)
for ( i = 3; i < 6; ++i ) {
  ctx26Parent += (AvailNodeOccupancyCnt[depth][x + deltaX[childIdx][i]][y + deltaY[childIdx][i]][z + deltaZ
[childIdx][i]] ? 1 : 0) << i
}
ctxFrom6Nei = (!!(ctx6Parent & 0x0003)) && (!!(ctx6Parent & 0x000c)) && (!!(ctx6Parent & 0x0030))
ctxHashMap[8][8] = {{0, 1, 1, 1, 1, 1, 1, 1},{2, 5, 5, 9, 4, 10, 10, 13},
                    {2, 5, 4, 10, 5, 9, 10, 13},{3, 7, 8, 11, 8, 11, 12, 14},
                    {2, 4, 5, 10, 5, 10, 9, 13},{3, 8, 7, 11, 8, 12, 11, 14},
                    {3, 8, 8, 12, 7, 11, 11, 14},{6, 15, 15, 15, 15, 15, 15, 15}}
for (int j = 0; j < 3; j++) {
  ctxFrom3FaceNei |= (!!((ctx26Parent[i] >> j) & 0x01) << (2 - j))
  ctxFrom3EdgeNei |= (!!((ctx26Parent[i] >> (j + 3)) & 0x01) << (2 - j))
}
ctxFromParent = ctxHashMap [neiFromFaceMode][neiFromEdgeMode] * 2 + ctxFrom6Nei
adjChildIdx = adjacentCIdx[i]
for (size_t idx = 0; idx < 7; ++idx) {
  childInformation |=
  !!(encodedChildNode[adjChildIdx[idx] >> 3] & (1 << (adjChildIdx[idx] & 7))) << idx
}
childInformation |= !!(encodedChildNode[3] & (1 << (i))) << 7
childInformation |= !!(encodedChildNode[5] & (1 << (i))) << 8
childInformation |= !!(encodedChildNode[6] & (1 << (i))) << 9
memoryVal = memoryChannel[childInformation]
ctxFromMemory = popcnt8(memoryVal)
context = ctxFromParent * 9 + ctxFromMemory
```

其中,通过映射哈希表 ctxHashMap 得到与当前子节点直接接触的 6 个共面、共线相邻节点的占用信息；ctxFrom3FaceNei 表示与当前子节点直接接触的 3 个共面相邻节点的占据情况，完成计算后有效位为 3 位；ctxFrom3EdgeNei 表示与当前子节点直接接触的 3 个共线相邻节点的占用信息，完成计算后有效位为 3 位。ctx6Parent 表示与当前待解码节点父节点直接接触的 6 个共面节点的占位情况。memoryChannel[childInformation]表示子节点组成的 1024 个上下文状态，这 1024 个状态的初始值设置为 15。ctxFromParent 表示当前待解码子块父节点相邻块的占位情况，ctxFromMemory 表示当前待解码子块的占位情况经过状态转换后最近已解码的 8 个符号的取值，以 bit_ctx + context 表示最终选择的上下文。这里 bit_ctx 是上下文的起始地址，adjacentCIdx 表示各子节点选择的邻居节点的索引。

对于当前待解码占位码的位,其对应的滑动窗口 memoryChannel 的更新方法如下：

```
memoryChannel[childInformation] = (memoryChannel[childInformation] << 1) | bit
```

其中deltaX、deltaY和deltaZ的取值如表54、表55和表56所示：

表54　deltaX 查找表

deltaX[childIdx][i]	i = 0	i = 1	i = 2	i = 3	i = 4	i = 5
childIdx = 0	-1	0	0	-1	-1	0
childIdx = 1	-1	0	0	-1	-1	0
childIdx = 2	-1	0	0	-1	-1	0
childIdx = 3	-1	0	0	-1	-1	0
childIdx = 4	1	0	0	1	1	0
childIdx = 5	1	0	0	1	1	0
childIdx = 6	1	0	0	1	1	0
childIdx = 7	1	0	0	1	1	0

表55　deltaY 查找表

deltaY[childIdx][i]	i = 0	i = 1	i = 2	i = 3	i = 4	i = 5
childIdx = 0	0	-1	0	-1	0	-1
childIdx = 1	0	-1	0	-1	0	-1
childIdx = 2	0	1	0	1	0	1
childIdx = 3	0	1	0	1	0	1
childIdx = 4	0	-1	0	-1	0	-1
childIdx = 5	0	-1	0	-1	0	-1
childIdx = 6	0	1	0	1	0	1
childIdx = 7	0	1	0	1	0	1

表56 deltaZ 查找表

deltaZ[childIdx][i]	i = 0	i = 1	i = 2	i = 3	i = 4	i = 5
childIdx = 0	0	0	−1	0	−1	−1
childIdx = 1	0	0	1	0	1	1
childIdx = 2	0	0	−1	0	−1	−1
childIdx = 3	0	0	1	0	1	1
childIdx = 4	0	0	−1	0	−1	−1
childIdx = 5	0	0	1	0	1	1
childIdx = 6	0	0	−1	0	−1	−1
childIdx = 7	0	0	1	0	1	1

9.2.3.3 几何节点尺寸解码

几何节点对数尺寸和子节点对数尺寸解码如下：

```
NodeSizeXLog2 = MaxNodeSizeXLog2 – depthX
NodeSizeYLog2 = MaxNodeSizeYLog2 – depthY
NodeSizeZLog2 = MaxNodeSizeZLog2 – depthZ
if (!(partitionSkip & 4)
   ChildNodeSizeXLog2 = NodeSizeXLog2 – 1
else
   ChildNodeSizeXLog2 = NodeSizeXLog2
if (!(partitionSkip & 2)
   ChildNodeSizeYLog2 = NodeSizeYLog2 – 1
else
   ChildNodeSizeYLog2 = NodeSizeYLog2
if (!(partitionSkip & 1)
   ChildNodeSizeZLog2 = NodeSizeZLog2 – 1
else
   ChildNodeSizeZLog2 = NodeSizeZLog2
```

9.2.3.4 孤立点解码控制模式

a) 计算 GeomSingleNodeControlFlag。

对 singlePointEligibleNodeCnt 进行更新，方法如下：

```
if (GeomSingleEligibleFlag[depth])
   singlePointEligibleNodeCnt++
```

计算 GeomSingleNodeControlFlag 的值，方法如下：

```
if (GeomSingleNodeControlMode == 0) {
   GeomSingleNodeControlFlag = 0
} else {
```

```
    if (singlePointEligibleNodeCnt == 5 && bit_count_equ1_num >= 4)
      GeomSingleNodeControlFlag = 0
    else
      GeomSingleNodeControlFlag = 1
  }
```

b） 根据解码结果进行更新。

若 GeomSingleNodeControlFlag = 0，则进行 9.2.3.4 孤立点解码，否则进行 9.2.3.2 占位码解码。

节点完成解码之后，根据解码结果进行以下更新：

若 GeomSingleNodeControlMode = 1 且 singlePointEligibleNodeCnt < 10，则根据孤立点解码结果 geom_single_flag 更新 singlePointNodeCnt，方法如下：

```
  if (geom_single_flag)
    singlePointNodeCnt++
```

若 GeomSingleNodeControlMode = 0 且 singlePointEligibleNodeCnt < 5，则进行占位码解码，以及根据节点是否只包含一个被占据的子节点更新 bit_count_equ1_num，方法如下：

```
  if (GeometryNodeChildrenCnt == 1)
    bit_count_equ1_num++
```

若 GeomSingleNodeControlMode = 0 且 singlePointEligibleNodeCnt = 10，则更新 GeomSingleNodeControlMode，并将 singlePointEligibleNodeCnt、singlePointNodeCnt 置零，方法如下：

```
  if (singlePointEligibleNodeCnt == 10) {
      GeomSingleNodeControlMode = !(singlePointNodeCnt >= 3)
      singlePointEligibleNodeCnt = 0
      singlePointNodeCnt = 0
  }
```

若 GeomSingleNodeControlMode = 1 且 singlePointEligibleNodeCnt = 5，则更新 GeomSingleNodeControlMode，以及将 singlePointEligibleNodeCnt、bit_count_equ1_num 置零，方法如下：

```
  if (singlePointEligibleNodeCnt == 5) {
      GeomSingleNodeControlMode = bit_count_equ1_num >= 4 && !geom_single_flag
      GeomSingleNodeControlFlag = GeomSingleNodeControlMode
      singlePointEligibleNodeCnt = 0
      bit_count_equ1_num = 0
  }
```

9.2.3.5 孤立点几何位置解码

孤立点相对于其所属节点的相对位置的解码过程如下：

```
    PointOffsetX = PointOffsetY = PointOffsetZ = 0
    for (i = 0; i < ChildNodeSizeXLog2; i++) {
      PointOffsetX += point_offset_x[i] << i
```

```
}
for (i = 0; i < ChildNodeSizeYLog2; i++) {
  PointOffsetY += point_offset_y[i] << i
}
for (i = 0; i < ChildNodeSizeZLog2; i++) {
  PointOffsetZ += point_offset_z[i] << i
}
```

当 geom_single_flag 等于 1 时，孤立点几何坐标解码过程如下：

```
PointPos[PointCount][0] = xN + PointOffsetX[i]
PointPos[PointCount][1] = yN + PointOffsetY[i]
PointPos[PointCount][2] = zN + PointOffsetZ[i]
++PointCount
```

在解码孤立点后，采用当前节点的占位码 OccupancyCode 信息更新几何上下文。

9.2.3.6 非空子节点个数的解码

当 OccupancyCode 不等于 0 时，当前几何节点的非空子节点个数的解码过程如下：

```
childCnt = 0
for (childIdx = 0; childIdx < 8; childIdx++) {
if (!(OccupancyCode & (1 << childIdx)))
  continue
GeometryNodeChildren[childCnt] = childIdx
++ childCnt
}
GeometryNodeChildrenCnt = childCnt
GeometryNodeOccupancyCnt[depth][xN][yN][zN] = childCnt
```

9.2.3.7 叶子节点几何位置解码

当 OccupancyCode 不等于 0 时，且到达叶子节点，即 depthX >= MaxNodeSizeXLog2 - 1 && depthY >= MaxNodeSizeYLog2 - 1 && depthZ >= MaxNodeSizeZLog2 - 1 时，叶子节点的几何坐标解码过程如下：

```
for (child = 0; child < GeometryNodeChildrenCnt; child++) {
  childIdx = GeometryNodeChildren[child]
  if (!(OccupancyCode & ( 1 << childIdx)))
    continue
  xC = (xN << !(partitionSkip & 4)) + ( childIdx & 4 == 1)
  yC = (yN << !(partitionSkip & 2)) + ( childIdx & 2 == 1)
  zC = (zN << !(partitionSkip & 1)) + ( childIdx & 1 == 1)
  for (i = 0; i < GeometryNodeDupPoints[child] + 1; ++i) {
    PointPos[PointCount][0] = xC
```

```
            PointPos[PointCount][1] = yC
            PointPos[PointCount][2] = zC
            ++PointCount
        }
    }
```

9.2.3.8 下一层节点信息更新

当 OccupancyCode 不等于 0 时，更新下一层划分节点信息的过程如下：

```
    nodeIdx = NumNodesAtDepth[depth + 1]
    for (child = 0; child < GeometryNodeChildrenCnt; child++) {
        childIdx = GeometryNodeChildren[child]
        if (!(OccupancyCode & (1 << childIdx)))
            continue
        xC = NodeX[depth+1][nodeIdx] = (xN << !(partitionSkip & 4)) + (childIdx & 4 == 1)
        yC = NodeY[depth+1][nodeIdx] = (yN << !(partitionSkip & 2)) + (childIdx & 2 == 1)
        zC = NodeZ[depth+1][nodeIdx] = (zN << !(partitionSkip & 1)) + (childIdx & 1 == 1)
        GeometryNodeOccupancyCnt[depth+1][xC][yC][zC] = 1
        nodeIdx++
    }
    NumNodesAtDepth[depth + 1] = nodeIdx
```

9.2.4 宏块几何划分解码

几何划分是在宏块内进行，同 9.2.2 的过程一致。因此下述变量：

MaxGeometryOctreeDepth, depthX, depthY, depthZ, NumNodesAtDepth[depth], GeometryNodeOccupancyCnt[depth][xN][yN][zN], NodeX[depth][idx], NodeY[depth][idx], NodeZ[depth][idx], MaxNodeSizeXLog2, MaxNodeSizeYLog2, MaxNodeSizeZLog2, partitionSkip 改成在宏块中定义（将宏块作为八叉树的根节点进行划分），改成下述变量：

lcuMaxGeometryOctreeDepth, lcuDepthX, lcuDepthY, lcuDepthZ, lcuNumNodesAtDepth[depth], lcuGeometryNodeOccupancyCnt[depth][xN][yN][zN], lcuNodeX[depth][idx], lcuNodeY[depth][idx], lcuNodeZ[depth][idx], lcuMaxNodeSizeXLog2, lcuMaxNodeSizeYLog2, lcuMaxNodeSizeZLog2, lcuPartitionSkip。

9.2.5 八叉树宏块几何节点解码

几何宏块状态保存过程如下：

a) 当 save_state_flag = 1 时，在进入宏块解码前，保存当前熵编码上下文和几何解码的哈希表信息（存储邻居占位信息）；在对每个宏块解码时，恢复所存的熵编码上下文和几何解码的哈希表信息。这样几何解码中宏块之间没有参考依赖关系，可以实现宏块之间独立解码。

b) 当 save_state_flag = 0 时，在进入宏块解码前，会关闭熵编码上下文和几何解码的哈希表信息的存储和恢复。

当 lcu_dependency_flag = 1 时，指示所有宏块允许熵依赖，即当前宏块解码完，会保存当前熵编码上下文和几何解码的哈希表信息给下一个宏块使用。此时，宏块之间是串行解码。

当 lcu_dependency_flag = 0 时，所有宏块不允许熵依赖，即每个宏块独立解码，每个熵编码上下文和几何解码的哈希表信息会重新初始化。此时，宏块之间是并行解码，这样可以减少内存的消耗和信息存储和恢复的操作。

9.2.6 预测树宏块解码

首先定义下述变量：

预测树宏块中的节点个数 lcuNumNodes[idx]：表示第 idx 个宏块中的预测树几何节点个数。

预测树节点在 X 方向的残差 lcuResidualX[idx][n]：表示在第 idx 个宏块中，第 n 个点的 X 方向坐标残差。

预测树节点在 Y 方向的残差 lcuResidualY[idx][n]：表示在第 idx 个宏块中，第 n 个点的 Y 方向坐标残差。

预测树节点在 Z 方向的残差 lcuResidualZ[idx][n]：表示在第 idx 个宏块中，第 n 个点的 Z 方向坐标残差。

预测树节点在 X 方向的坐标 PointPosX[idx][n]：表示在第 idx 个宏块中，第 n 个点的 X 方向坐标。

预测树节点在 Y 方向的坐标 PointPosY[idx][n]：表示在第 idx 个宏块中，第 n 个点的 Y 方向坐标。

预测树节点在 Z 方向的坐标 PointPosZ[idx][n]：表示在第 idx 个宏块中，第 n 个点的 Z 方向坐标。

预测树节点在 X 方向的起始坐标 lcuNodeX[idx]：表示第 idx 个宏块在 X 方向的起始坐标。

预测树节点在 Y 方向的起始坐标 lcuNodeY[idx]：表示第 idx 个宏块在 Y 方向的起始坐标。

预测树节点在 Z 方向的起始坐标 lcuNodeZ[idx]：表示第 idx 个宏块在 Z 方向的起始坐标。

如果第 idx 个宏块的 geom_tree_type = 1，其解码过程如下：

a） 解析表示第 idx 个宏块中的预测树几何节点个数 lcuNumNodes[idx] = lcu_num_points。

b） 获得 geom_max_tree_size，将该宏块按 geom_max_tree_size 点个数进行解码，当最后一组的点数小于 geom_max_tree_size 时，与其前一组点合并解析。

c） 确定预测树的起始点信息，以第 idx 个宏块的几何坐标作为预测树解码的起始几何坐标，逐个解析码流中的预测树节点在 X，Y，Z 方向的几何残差信息，由起始几何坐标与解析的第一个残差信息相加，获得该宏块中第一个预测树节点几何坐标。由前一点几何坐标与当前点残差信息相加，逐个获得当前宏块内所有预测树节点的几何坐标重建值。确定下一组的起始点的几何信息为前一组的最后一个重建点的几何信息。重复该过程，确定每个组的单链预测树。

```
PointPosX[idx][0] = lcuNodeX[idx] + lcuResidualX[idx][0]
PointPosY[idx][0] = lcuNodeY[idx] + lcuResidualY[idx][0]
PointPosZ[idx][0] = lcuNodeZ[idx] + lcuResidualZ[idx][0]
for (nodeIdx = 1; nodeIdx < lcuNumNodes[idx]; ++n) {
 PointPosX[idx][n] = PointPosX[idx][n - 1] + lcuResidualX[idx][n]
 PointPosY[idx][n] = PointPosY[idx][n - 1] + lcuResidualY[idx][n]
 PointPosZ[idx][n] = PointPosZ[idx][n - 1] + lcuResidualZ[idx][n]
}
```

待解码顶点P_i的几何坐标重建值计算过程如下：

1） 计算待解码顶点P_i到前序顶点在X,Y,Z三个坐标分量上的残差值的绝对值absResidual[n][k]，其中k = 0,1,2表示对应的几何位置坐标X，Y，Z；i表示点数，i = 0,1,…,N-1；N是宏块的点数。其步骤如下：

解析is_geom_residual_zero[n][k]并确定当前坐标分量的残差绝对值是否为0。

若解析对应残差绝对值非零，解析对应的除二位数num_bits_geom_residual_minus1[n][k]，采用二值化形式$b_m...b_4b_3b_2b_1b_0$逐位解析，所需位数为numbits_Lcusize_log2[k]，表示当前宏块对应的几何节点尺寸第k分量所占用位数的值（k = 0,1,2）。

ctxIdx = 0，解析b_0；

ctxIdx = 1 + b_0，解析b_1；

ctxIdx = 3 + b_1，解析b_2；

ctxIdx = 5+ b_2 + b_1 << 1，解析b_3；

ctxIdx = 9，解析$b_m...b_4$；

计算num_bits_geom_residual_minus1[n][k]，方法如下：

num_bits_geom_residual_minus1[n][k] = b_0 + b_1 << 1 + b_2 << 2 + …+ b_m << m

基于位数num_bits_geom_residual_minus1[n][k]逐位解析得到当前坐标分量的几何残差绝对值减一除二的值geom_residual_minus1_div2[n][k]，方法如下：

geom_residual_minus1_div2[n][k] = geom_residual_minus1_div2[n][k][0] + geom_residual_minus1_div2[n][k][1] << 1 + …+ geom_residual_minus1_div2[n][k][j] << j

计算每个坐标分量上的几何残差值绝对值absResidual[n][k]，方法如下：

absResidual [n][k] = geom_residual_minus1_div2[n][k] << 1 + geom_residual_minus1_div2_remain[n][k] + 1

2） 计算待解码顶点P_i到前序顶点的残差符号位lcuResidualSign[n][k]，其中k = 0,1,2表示对应的几何位置坐标X，Y，Z。根据待解码顶点P_i的残差值的绝对值x_r，y_r，z_r与前序顶点P_i到前序顶点的父顶点P_i的曼哈顿距离d_0，对码流解码得到残差符号位。其步骤如下：

计算待解码顶点P_i的前序顶点P_i到前序顶点的父顶点P_i曼哈顿距离d_0见式（6）。

$$d_0 = |x_{i-1} - x_{i-2}| + |y_{i-1} - y_{i-2}| + |z_{i-1} - z_{i-2}| \quad (6)$$

式中：

d_0 ——曼哈顿距离；

$x_{i-1}, y_{i-1}, z_{i-1}$ ——顶点P_i的X，Y，Z坐标；

$x_{i-2}, y_{i-2}, z_{i-2}$ ——顶点P_i的X，Y，Z坐标。

0代表正号、1代表负号，则X,Y,Z三个坐标分量上的残差值的符号组合情况为000，001，010，011，100，101，110，111，共8种。残差值的符号结合残差值的绝对值x_r, y_r, z_r，得到8种可能的残差值组合(x_r, y_r, z_r)，$(x_r, y_r, -z_r)$，$(x_r, -y_r, z_r)$，$(x_r, -y_r, -z_r)$，$(-x_r, y_r, z_r)$，$(-x_r, y_r, -z_r)$，$(-x_r, -y_r, z_r)$，$(-x_r, -y_r, -z_r)$。

8种可能的残差值组合分别和前序顶点P_i坐标值$(x_{i-1}, y_{i-1}, z_{i-1})$相加，得到8个可能的顶点$P_i$及其坐标值$(x_j, y_j, z_j)$

计算每个可能的顶点P_i到前序顶点的父顶点P_i的曼哈顿距离d_j见式（7）

$$d_j = |x_j - x_{i-2}| + |y_j - y_{i-2}| + |z_j - z_{i-2}| \quad (7)$$

式中：

d_j ——曼哈顿距离；

x_j, y_j, z_j ——顶点P_i的X，Y，Z坐标；

$x_{i-2}, y_{i-2}, z_{i-2}$ ——顶点P_i的X，Y，Z坐标。

比较d_j和d_0，删除残差值的符号的组合中使得d_j小于d_0的情况，剩余残差值的符号的可行情况为 M = geom_residual_max_rel_sign[n]个，并按顺序编号为 0,1,...N-1,M-1。

根据残差值的符号的可行情况及其编号对码流解码得到该顶点P_i的残差值的符号。以二进制形式从高位到低位解码残差值的符号的相对编号。解码相对编号的第 j 位 geom_residual_ord_rel_sign[n][j]时，令第 j 位 geom_residual_ord_rel_sign[n][j]取 1，剩余未解码位取 0，将得到的数字的十进制值记为 m，若$n > N-1$，则相对编号的第 j 位 geom_residual_ord_rel_sign[n][j]为 0，否则进行基于上下文的算术解码得到相对编号的第 j 位的值 geom_residual_ord_rel_sign[n][j]；继续解码下一位直到解码 3 位后得到残差值的符号的相对编号；该相对编号与前序顶点P_i残差值的实际编号异或得到其实际编号，根据该实际编号在残差值的符号的可行情况中找到对应的残差值的符号，得到该顶点P_i残差值的符号。

3) 结合待解码顶点P_i的残差值的绝对值 absResidual[n][k]和该顶点P_i的残差符号位 lcuResidualSign[n][k]，得到待解码顶点P_i的残差 lcuResidual[n][k]。

4) 将待解码顶点P_i的残差与前序顶点P_i的坐标值相加，得到待解码顶点P_i的几何坐标重建值。

9.3 属性数据解码过程

9.3.1 概述

属性数据解码过程如下：

第一步，几何解码点重排序；

第二步，邻居预测点查找；

第三步，预测值计算；

第四步，属性残差解码；

第五步，属性残差绝对值反量化；

第六步，属性重建值解码。

该标准包含三种属性解码过程。

基于预测算法的属性解码过程一：

根据 9.3.2 ，9.3.3 ，9.3.4 获得属性解码顺序，根据 9.3.5 ，9.3.6 ，9.3.7 和 9.3.8 获得当前待解码点的邻居预测点，并进一步根据 9.3.9 计算当前待解码点的属性预测值；根据 9.3.10 逐点解码属性残差值，并按照 9.3.11 对其反量化；最后根据 9.3.12 解码当前待解码点的属性重建值。

基于变换的属性解码过程二：

根据 9.3.13 对属性值解码。

基于预测变换的属性解码过程三：

根据 9.3.14 对属性值解码。

对多属性点云，在解码每个属性时，需要先初始化熵编码器。

9.3.2 莫顿码重排序

将几何解码后的点的三维笛卡尔坐标转换成莫顿码（参见 5.7.2 ），根据莫顿码按递增的顺序排序点云中的点。将排序后的点表示为数组大小为 slice_num_points 的一维数组，Morder[slice_num_points]= { $P_0, P_1, \cdots, P_{idx}, P_{idx+1}, \cdots P_{slice\_num\_points-1}$}，其中 P_{idx} 是第 idx 个索引对应点

的莫顿码，P_{idx}大于或等于P_{idx-1}，对数组 Morder 内的点按索引从小到大的顺序进行属性解码。

9.3.3 希尔伯特码重排序

将几何解码后的点的三维笛卡尔坐标转换成希尔伯特码（参见 0），根据希尔伯特码按递增的顺序排序点云中的点。将排序后的点表示为数组大小为 slice_num_points 的一维数组，Horder[slice_num_points]= {$P_0, P_1, \cdots, P_{idx}, P_{idx+1}, \cdots P_{slice\_num\_points-1}$}，其中$P_{idx}$是第 idx 个索引对应点的希尔伯特码，$P_{idx}$大于或等于$P_{idx-1}$，对数组 Horder 内的点按索引从小到大的顺序进行属性解码。

9.3.4 原始顺序

经几何解码后点的重建顺序。

9.3.5 基于替换最远点的邻居预测点全搜索（颜色属性）

9.3.5.1 确定邻居预测点

设当前待解码组有 L 个点，当前待解码点 Pi 为根据 color_reorder_mode 确定的顺序（原始顺序或莫顿或希尔伯特）重排序后的第 i 个点，且 Pi 为组内第 n 个点（0<=n<L），点 Ps 为重排序后的第 s 个点（0<=i,s<N，N 为待解码点的总数），且 Ps 为组内第 0 个点。对于基于预测算法的解码过程一，当前待解码组只有 1 个点，此时 i=s，Pi 和 Ps 为同一个点。

设预测参考点集为 Sp，长度上限为 maxNumOfNeighbours。Sp 中包含了当前点索引之前的最多 maxNumOfNeighbours 个已解码点的坐标重建值与颜色重建值。对于当前待解码点 P_i，邻居预测点的计算方法如下：

如果$s = 0$,则无需查找，将{128，128，128}作为预测值；

如果$s = 1$,则 P_0为邻居预测点；

如果$s = 2$,则 P_0, P_1为邻居预测点；

如果$3 <= s <= maxNumOfNeighbours$,则 P_{i-1}、P_{i-2}、P_{i-3}为邻居预测点；

如果$s > maxNumOfNeighbours$,在 Sp 中遍历查找当前待解码点 $P_i(x_i, y_i, z_i)$的 3 个距离最近点及最多 13 个等距离点作为当前待解码点的邻居预测点。对于 Sp 中任一点 $P_j(x_j, y_j, z_j)$，距离 d 计算采用曼哈顿距离，定义为：$d = |x_i - x_j| + |y_i - y_j| + |z_i - z_j|$。如果两个点距离相同，则优先选择最先遍历的顶点。

3 个距离最近点及最多 13 个等距离点作为当前点的邻居预测点的搜索方式如下：

a） 距离最近点集合初始化为空集。当邻居预测点集合中的点数小于 3 个，即在处理当前点的 Sp 中的前 3 个点 P_{i-1}、P_{i-2}、P_{i-3}时，按照索引从大到小的顺序将这 3 个点插入距离最近点集合，且每插入一个点之后，都要根据距离最近点集合中的点与当前待解码点之间的距离，对距离最近点集合中的点进行升序排序，使得排序之后的三个点的距离满足$d_1 <= d_2 <= d_3$。这里d_1、d_2和d_3分别表示距离最近点集合中第 1 点、第 2 点和第 3 点与当前待解码点的距离；

b） 继续在 Sp 中按照索引递减顺序搜索第 j 点，当距离最近点集合中的点数等于 3 个时，若第 j 点与当前待解码点之间的距离d_j等于d_3，则将第 n 点加入到等距离点集合中；若第 j 点与当前待解码点之间的距离$d_j < d_3$，则将 d_3对应的点移出距离最近点集合并令$d_{temp} = d_3$，并将 j 点加入距离最近点集合，加入新点后再次根据距离最近点集合中的点与当前待解码点之间的距离对距离最近点集合中的点进行升序排序，然后将重排序后的 d_3与被移出点的距离d_{temp}进行比较，若d_{temp}等于d_3，则将被移出的点加入到等距离点集合中，否则将等距离点集合清空；

c） 直到搜索完 Sp 中的所有点，最终获得 3 个距离最近点以及最多 13 个等距离点作为当前点的邻居预测点。

9.3.5.2 确定最远点

如果$s > maxNumOfNeighbours$，在 Sp 内$[n, maxNumOfNeighbours - 1]$查找距离 P_i 最远的点，假设距离 P_i 最远点的距离为 D_{max}，若存在多个与 P_i 距离为 D_{max} 的点，按照 Sp 中的索引从小到大排序，确定第一个为最远点进行交换，将该最远点的坐标重建值及颜色重建值与 Sp 的第 n 个点进行交换。

9.3.5.3 更新预测参考点集

在当前组 L 个点全部预测、重构完成后，更新预测参考点集：

如果$0 <= s <= maxNumOfNeighbours$，将当前待解码组的 L 个点的坐标重建值及颜色重建值放入 Sp，以当前点 P_i 为例，将其放入 Sp 的第 m 个点，$m = i\%maxNumOfNeighbours$。

如果$s > maxNumOfNeighbours$，用当前待解码组 L 个点的坐标重建值及颜色重建值替换 Sp 的前 L 个点的信息，以当前点 P_i 为例，用其替换 Sp 的第 n 个点。

9.3.6 基于替换最远点和空间偏倚的邻居预测点全搜索（反射率属性）

9.3.6.1 确定邻居预测点

将几何解码后的点的三维笛卡尔坐标中的 z 轴坐标乘以空间偏倚系数 axisBias，x 轴、y 轴坐标不变，再进行几何解码点重排序。

设当前待解码组有 L 个点，当前待解码点 Pi 为根据 refl_reorder_mode 确定的顺序（原始顺序或莫顿或希尔伯特）重排序后的第 i 个点，且 Pi 为组内第 n 个点（0<=n<L），点 Ps 为重排序后的第 s 个点（0<=i,s<N，N 为待解码点的总数），且 Ps 为组内第 0 个点。对于基于预测算法的解码过程一，当前待解码组只有 1 个点，此时 i=s，Pi 和 Ps 为同一个点。

设预测参考点集为 Sp，长度上限为 aps 中的 maxNumOfNeighbours。Sp 中包含了当前点索引之前的最多 maxNumOfNeighbours 个已解码点的坐标重建值与反射率重建值。对于当前待解码点 P_i，邻居预测点的计算方法如下：

如果$s = 0$，不进行邻居预测点查找，将{0}作为预测值；

如果$s = 1$，则 P_0 为邻居预测点；

如果$s = 2$，则 P_0, P_1 为邻居预测点；

如果$3 <= s <= maxNumOfNeighbours$，则 P_{i-1}、P_{i-2}、P_{i-3} 为邻居预测点；

如果$s > maxNumOfNeighbours$，在 Sp 中遍历查找与当前待解码点 $P_i(x_i, y_i, z_i)$ 距离最近的 3 个点作为当前待解码点的邻居预测点 $P_j(x_j, y_j, z_j)$，对于 Sp 中任一点 $P_j(x_j, y_j, z_j)$，距离 d 计算采用曼哈顿距离，定义为：$d = |x_i - x_j| + |y_i - y_j| + |z_i - z_j|$。

9.3.6.2 确定最远点

如果$s > maxNumOfNeighbours$，在 Sp 内$[n, maxNumOfNeighbours - 1]$查找距离 P_i 最远的点，假设距离 P_i 最远点的距离为 D_{max}，若存在多个与 P_i 距离为 D_{max} 的点，按照 Sp 中的索引从小到大排序，确定第一个为最远点进行交换，将该最远点的坐标重建值及反射率重建值与 Sp 的第 n 个点进行交换。

9.3.6.3 更新预测参考点集

在当前组 L 个点全部预测、重构完成后，更新预测参考点集：

如果$0 <= s <= maxNumOfNeighbours$，将当前待解码组 L 个点的坐标重建值及反射率重建值放入 Sp，以当前点 P_i 为例，将其放入 Sp 的第 m 个点，$m = i\%maxNumOfNeighbours$。

如果$s > maxNumOfNeighbours$，用当前待解码组 L 个点的坐标重建值及反射率重建值替换 Sp 的前 L 个点的信息，以当前点 P_i 为例，用其替换 Sp 的第 n 个点。

9.3.7 多属性点云的邻居预测点查找

对于包含多种属性类型的点云，维护一个长度上限为 maxNumOfNeighbours 的预测参考点集，记为 Sp，Sp 中包含了顶点的坐标，颜色，反射率。令当前待解码组有 L 个点，当前待解码点 P_i 为根据 color_reorder_mode 确定的顺序（原始顺序或莫顿或希尔伯特）重排序后的第 i 个点，且 P_i 为组内第 n 个点（0 <= n < L），点 P_s 为重排序后的第 s 个点（0 <= i,s < N，N 为待解码点的总数），且 P_s 为组内第 0 个点$L > n \geq 0$。$L > n \geq 0$对于基于预测算法的解码过程一，当前待解码组只有 1 个点，此时 i = s，P_i 和 P_s 为同一个点。

当 cross_attr_type_pred = 0 时，分别采用 9.3.5 和 9.3.6 的方法对颜色属性和反射率属性进行邻居预测点搜索。

当 cross_attr_type_pred = 1 时，在属性预测时，对于先预测的属性采用 9.3.5.1 和 9.3.6.1 的对应方法对颜色属性和反射率属性进行邻居预测点搜索，对于后预测的属性采用 9.3.5.1 和 9.3.6.1 的对应方法对颜色属性和反射率属性进行邻居预测点搜索，其中确定邻居预测点所用的距离改用综合距离 dis。并用 9.3.5.2 和 9.3.6.2 确定最远点，在后预测属性完成预测后统一进行最远点交换及更新。

综合距离 dis 的计算如下所示：

a） 计算几何信息距离 disGeom，几何信息距离为待解码点与邻居候选点之间的曼哈顿距离。

b） 计算属性信息距离 disAttr，属性信息距离为待解码点与邻居候选点之间的已解码属性类型的残差的绝对值。

反射率属性信息距离见式（8）：

$$disAttr = |curRefl - neighRefl| \quad (8)$$

式中：

disAttr ——属性信息距离；

curRefl ——待解码点的反射率；

neighRefl ——邻居候选点的反射率。

颜色属性信息距离：

$$disAttr = |curColor[0] - nColor[0]| + |curColor[1] - nColor[1]| + |curColor[2] - nColor[2]| \quad (9)$$

式中：

disAttr ——属性信息距离；

curColor ——待解码点的颜色；

nColor ——邻居候选点的颜色。

c） 根据几何信息距离 disGeom 和属性信息距离 disAttr 计算综合距离 dis，其中综合距离 dis 的计算见式（10）。

$$dis = disGeom + \left(\left(\left(\lambda * \mathrm{Round}\left(\frac{maxGeom<<10}{maxAttr}\right) + 2^{19}\right) >> 20\right) * disAttr\right) >> 10 \quad (10)$$

式中：

dis ——综合距离；

disGeom ——几何信息距离；

disAttr ——属性信息距离；

maxGeom ——几何信息的最大值；

maxAttr ——属性信息的最大值；

λ ——属性信息距离在综合距离计算过程中的权重。

其中，几何信息的最大值的计算见式（11）：

$$maxGeom = bounding\_box\_size\_width + bounding\_box\_size\_height + bounding\_box\_size\_depth \quad (11)$$

式中：

maxGeom ——几何信息的最大值；

bounding_box_size_width ——点云包围盒的宽度值；

bounding_box_size_height ——点云包围盒的高度值；

bounding_box_size_depth ——点云包围盒的深度值。

属性信息的最大值的计算见式（12）：

对于反射率属性信息：

$$maxAttr = 2^{reflOutputDepth} - 1 \quad (12)$$

式中：

maxAttr ——反射率的最大值；

reflOutputDepth ——反射率的位深。

对于颜色属性信息见式（13）：

$$maxAttr = 2^{colorOutputDepth} - 1 \quad (13)$$

式中：

maxAttr ——颜色的最大值；

colorOutputDepth——颜色的位深。

属性信息距离在综合距离计算过程中的权重计算见式（14）：

$$\lambda = -crossAttrTypePredParam1 * attrQuantParm + crossAttrTypePredParam2 \quad (14)$$

式中：

λ ——属性信息距离在综合距离计算过程中的权重；

crossAttrTypePredParam1 ——权重计算过程中的参数1；

crossAttrTypePredParam2 ——权重计算过程中的参数2；

attrQuantParm ——后预测属性的属性量化参数。

在后预测属性的邻居预测点寻找结束后，如果 $s > maxNumOfNeighbours$，在 Sp 内 $[n, maxNumOfNeighbours - 1]$ 查找距离 P_i 几何信息距离 disGeom 最远的点，假设距离 P_i 最远点的距离为 D_{max}，若存在多个与 P_i 距离为 D_{max} 的点，按照 Sp 中的索引从小到大排序，确定第一个为最远点进行交换，将该最远点的坐标与反射率重构值、颜色重构值与 Sp 的第 n 个点进行交换。

在当前组 L 个点全部预测、重构完成后，更新预测参考点集：

如果 $0 <= s <= maxNumOfNeighbours$，将当前待解码组 L 个点的坐标与反射率重构值、颜色重构值放入 Sp，以当前点 P_i 为例，将其放入 Sp 的第 m 个点，$m = i\%maxNumOfNeighbours$。

如果 $s > maxNumOfNeighbours$，用当前待解码组 L 个点的坐标与反射率重构值、颜色重构值替换 Sp 的前 L 个点的信息，以当前点 P_i 为例，用其替换 Sp 的第 n 个点。

9.3.8 重复点查找

设当前待解码点为 P_j，其坐标为 (x_j, y_j, z_j)，其前一个已解码的点为 P_{j-1}，其坐标为 $(x_{j-1}, y_{j-1}, z_{j-1})$。

如果 (x_j, y_j, z_j) 与 $(x_{j-1}, y_{j-1}, z_{j-1})$ 相等，定义标志位 isDuplicatePoint = 1，则 P_{j-1} 作为 P_j 的邻居预测点。

9.3.9 预测值计算

9.3.9.1 根据距离的加权预测计算

当前待解码点 P_j 的几何坐标为 (x_j, y_j, z_j)。

查找到的 k 个邻居预测点的坐标为：$(x_{jn}, y_{jn}, z_{jn})_{n=1,2,\ldots,k}$，属性重建值为 $(\hat{A}_{jn})_{n=1,2,\ldots,k}$。邻居预测点

的距离计算方法如下。

对于颜色属性计算见式（15）：

$$d_{jn} = |x_j - x_{jn}| + |y_j - y_{jn}| + |z_j - z_{jn}| \quad (15)$$

式中：

d_{jn} ——当前待解码点与邻居预测点n的曼哈顿距离；

x_j, y_j, z_j ——点P_j的X，Y，Z坐标；

x_{jn}, y_{jn}, z_{jn} ——邻居预测点 n 的 X，Y，Z 坐标。

对反射率属性计算见式（16）：

$$d_{jn} = |x_j - x_{jn}| + |y_j - y_{jn}| + \theta|z_j - z_{jn}| \quad (16)$$

式中：

d_{jn} ——当前待解码点与邻居预测点n的曼哈顿距离；

x_j, y_j, z_j ——点P_j的X，Y，Z坐标；

x_{jn}, y_{jn}, z_{jn} ——邻居预测点 n 的 X，Y，Z 坐标；

θ ——Z方向上的加权因子，等于axisBias。

k 个邻居预测点的属性重建值$(\widehat{A}_{jn})_{n=1,2,\ldots,k}$，k 个邻居预测点的权重计算见式（17）：

$$w_{jn} = \prod_{\substack{l=1 \\ l \neq n}}^{k} d_{jl} \quad (17)$$

式中：

w_{jn} ——邻居预测点n的权重；

d_{jl} ——当前待解码点与邻居预测点l的曼哈顿距离。

当前待解码点的属性预测值$\widetilde{A}_j$计算见式（18）：

$$\widetilde{A}_j = \text{Round}\left(\frac{\sum_{n=1}^{k} w_{jn}\widehat{A}_{jn}}{\sum_{n=1}^{k} w_{jn}}\right) \quad (18)$$

式中：

$\widetilde{A}_j$ ——当前待解码点P_j的属性预测值；

w_{jn} ——邻居预测点n的权重；

$\widehat{A}_{jn}$ ——邻居预测点n的属性重建值。

当 pred_fixed_point_frac_bit ＞ 0 时，上述公式中 k 个邻居预测点的权重计算使用定点化运算实现见式（19）：

$$w_{jn} = \text{Round}\left(\frac{\text{distanceScale}}{d_{jn}}\right) \quad (19)$$

式中：

w_{jn} ——邻居预测点n的权重；

d_{jn} ——当前待解码点与邻居预测点n的曼哈顿距离。

distanceScale ——缩放因子，计算见式（20）：

$$\text{distanceScale} = 2^{\text{pred\_fixed\_point\_frac\_bit}} - 1 \quad (20)$$

式中：

distanceScale ——缩放因子；

pred_fixed_point_frac_bit ——反射率属性预测精度值。

9.3.9.2 基于最近邻点的属性预测

由 9.3.5 方法查找到 k 个邻居预测点，其对应的属性重建值表示为$(\widehat{A}_n)_{n=1,2,\ldots,k}$，如果满足下述公式的条件，则选择$\widehat{A_1}$作为当前点的属性预测值，计算见式（21）。

$$\left|\widehat{A_1} - \widehat{A_k}\right| \geq \text{attr\_quant\_param} * \text{nearest\_pred\_param1} + \text{nearest\_pred\_param2} \quad (21)$$

式中：

$\widehat{A_1}$ ——k个邻居预测点中与当前点最近的点的属性重建值；

$\widehat{A_k}$ ——k个邻居预测点中与当前点最远的点的属性重建值。

9.3.9.3 基于同距离点个数和量化步长的加权预测优化

当前待解码点 P_j 的几何坐标为(x_j, y_j, z_j)。

查找到的p个邻居预测点中，距离小于最大距离值的邻居预测点有k个，坐标为：$(x_{jn}, y_{jn}, z_{jn})_{n=1,2,\ldots,k}$，属性重建值为$(\widehat{A}_{jn})_{n=1,2,\ldots,k}$。

k 个距离小于最大距离值的邻居预测点的权重见式（22）：

$$w_{jn} = \frac{1}{|x_j - x_{jn}| + |y_j - y_{jn}| + |z_j - z_{jn}|} \quad (22)$$

式中：

w_{jn} ——邻居预测点n的权重；

x_j, y_j, z_j ——当前待解码点P_j的X，Y，Z坐标；

x_{jn}, y_{jn}, z_{jn} ——距离小于最大距离值的邻居预测点 n 的 X，Y，Z 坐标。

查找到的 p 个邻居预测中，距离等于最大距离值的同距离邻居预测点有 r 个，坐标为：$(x_{jm}, y_{jm}, z_{jm})_{m=1,2,\ldots,k}$，属性重建值为$(\widehat{A}_{jm})_{m=1,2,\ldots,r}$。

r 个同距离邻居预测点的优化权重见式（23）：

$$w_{jm} = \frac{d_{wm}}{|x_j - x_{jm}| + |y_j - y_{jm}| + |z_j - z_{jm}|} \quad (23)$$

式中：

w_{jm} ——邻居预测点m的权重；

x_j, y_j, z_j ——当前待解码点P_j的X，Y，Z坐标；

x_{jm}, y_{jm}, z_{jm} ——同距离邻居预测点 m 的 X，Y，Z 坐标；

d_{wm} ——优化系数，当 attr_quant_param = 0 时，$d_{wm} = \frac{1}{r}$；当 attr_quant_param > 0 时，$d_{wm} = 1$。

当前待解码点的属性预测值$\widetilde{A}_j$计算见式（24）：

$$\widetilde{A}_j = \text{Round}\left(\frac{\sum_{n=1}^{k} w_{jn}\widehat{A}_{jn} + \sum_{m=1}^{r} w_{jm}\widehat{A}_{jm}}{\sum_{n=1}^{k} w_{jn} + \sum_{m=1}^{r} w_{jm}}\right) \quad (24)$$

式中：

$\widetilde{A}_j$ ——当前待解码点P_j的属性预测值；

w_{jn} ——k个距离小于最大距离值的邻居预测点中的点n的权重；

$\widehat{A}_{jn}$ ——k个距离小于最大距离值的邻居预测点中的点n的属性重建值；

w_{jm} ——r个同距离邻居预测点中的点m的优化权重；

$\widehat{A}_{jm}$ ——r个同距离邻居预测点中的点m的属性重建值。

9.3.9.4 基于重复点的预测方法

在当前待解码点为重复点时，针对反射率属性，其属性预测值为前一个点的属性重建值，无需解析其符号位，即默认为非负数。

针对颜色属性，其 Y/R 分量属性预测值为前一个点的属性重建值，无需解析其符号位，即默认为非

负数。

基于重复点的预测值方法在多种属性（同时具有颜色和反射率等）情况下，处于关闭状态。

9.3.9.5 基于属性值变化的邻居预测点查找

采用 9.3.6 的方法对反射率属性进行邻居预测点寻找。其中查找距离最近的点所用的距离采用了基于属性值变化的加权距离d_{avg}。

基于属性值变化的加权距离d_{avg}的计算方式如下：使用固定窗口统计已解码的前$N=2^{aps.log2_pred_dist_weight_group_size}$个点在三个方向上的属性值变化程度，计算权重$w_x, w_y, w_z$使用该权重对距离进行加权计算，每 N 个点使用同一组权重$[w_x, w_y, w_z]$，解码完 N 个点后更新权重。步骤如下：

a) 前 n 个已解码点的属性重建值为$(\hat{A}_k)_{k=1,2,\ldots,n}$，设 X，Y，Z 方向的属性变化$Ref_x = Ref_y = Ref_z = 0$，X，Y，Z 方向的点数$Num_x = Num_y = Num_z = 0$，当前待解码点为$P_i$。

b) 计算前 n 个已解码点的中的最大距离maxDist见式（25）：

$$maxDist = Max(|x_{i-im} - x_{i-im-1}|, |y_{i-im} - y_{i-im-1}|, |z_{i-im} - z_{i-im-1}|) \quad (25)$$

式中：

maxDist ——最大距离；

x_{i-im}，y_{i-im}，z_{i-im} ——点$i-im$的X，Y，Z坐标；

x_{i-im-1}，y_{i-im-1}，z_{i-im-1} ——点$i-im$的前一点的X，Y，Z坐标。

若 $maxDist = |x_{i-im} - x_{i-im-1}|$，计算见式（26）和式（27）：

$$Ref_x = Ref_x + \left(|\hat{A}_{i\text{-}im} - \hat{A}_{i\text{-}im\text{-}1}| \ll coef\right) / maxDist \quad (26)$$

$$Num_x = Num_x + 1 \quad (27)$$

式中：

Ref_x ——X方向的属性变化；

$\hat{A}_{i\text{-}im}$ ——点$i-im$的属性重建值；

$\hat{A}_{i\text{-}im\text{-}1}$ ——点$i-im$的前一点的属性重建值；

coef ——缩放系数；

maxDist ——最大距离；

Num_x ——X方向的点数。

若 $maxDist = |y_{i-im} - y_{i-im-1}|$，计算见式（28）和式（29）：

$$Ref_y = Ref_y + \left(|\hat{A}_{i\text{-}im} - \hat{A}_{i\text{-}im\text{-}1}| \ll coef\right) / maxDist \quad (28)$$

$$Num_y = Num_y + 1 \quad (29)$$

式中：

Ref_y ——Y方向的属性变化；

$\hat{A}_{i\text{-}im}$ ——点$i-im$的属性重建值；

$\hat{A}_{i\text{-}im\text{-}1}$ ——点$i-im$的前一点的属性重建值；

coef ——缩放系数；

maxDist ——最大距离；

Num_y ——Y方向的点数。

若 $maxDist = |z_{i-im} - z_{i-im-1}|$，则计算见式（30）和式（31）

$$Ref_z = Ref_z + \left(|\hat{A}_{i\text{-}im} - \hat{A}_{i\text{-}im\text{-}1}| \ll coef\right) / maxDist \quad (30)$$

$$Num_z = Num_z + 1 \quad (31)$$

式中：

Ref_z ——Z方向的属性变化；

$\hat{A}_{i\text{-}im}$ ——点$i-im$的属性重建值；

$\hat{A}_{i\text{-}im\text{-}1}$ ——点i − im的前一点的属性重建值；

coef ——缩放系数；

maxDist ——最大距离；

Num_z ——Z方向的点数。

其中，缩放系数coef计算见式（32）：

$$coef = \lceil \log_2(boundingBoxSize / geom\_num\_points) \rceil \quad (32)$$

式中：

coef ——缩放系数；

geom_num_points ——点云点数。

boundingBoxSize ——点云包围盒表面积，计算方式见式（33）：

$$\begin{aligned} boundingBoxSize = (&bounding\_box\_size\_width * bounding\_box\_size\_height + \\ &bounding\_box\_size\_height * bounding\_box\_size\_depth + \\ &bounding\_box\_size\_depth * bounding\_box\_size\_width) * 2 \end{aligned} \quad (33)$$

c） 计算属性值变化均值：

$$RefAvg_x = Ref_x / Num_x \quad (34)$$

$$RefAvg_y = Ref_y / Num_y \quad (35)$$

$$RefAvg_z = Ref_z / Num_z \quad (36)$$

式中：

$RefAvg_x$, $RefAvg_y$, $RefAvg_z$ ——X，Y，Z方向的属性变化均值；

Ref_x, Ref_y, Ref_z ——X，Y，Z方向的属性变化；$Ref_x = Ref_y = Ref_z = 0$

Num_x, Num_y, Num_z $Num_x = Num_y = Num_z = 0$ ——X，Y，Z方向的点数。

d） 计算权重见式（37）、式（38）、式（39）：

$$w_x = (RefAvg_x \ll 7) / (RefAvg_x + RefAvg_y + RefAvg_z) \quad (37)$$

$$w_y = (RefAvg_y \ll 7) / (RefAvg_x + RefAvg_y + RefAvg_z) \quad (38)$$

$$w_z = (RefAvg_z \ll 7) / (RefAvg_x + RefAvg_y + RefAvg_z) \quad (39)$$

式中：

w_x, w_y, w_z ——X，Y，Z方向的权重；

$RefAvg_x, RefAvg_y, RefAvg_z$ ——X，Y，Z方向的属性变化均值。

e） 计算加权距离见式（40）：

$$d_{avg} = (w_x|x_i - x_j| + w_y|y_i - y_j| + w_z|z_i - z_j|) \gg 3 \quad (40)$$

式中：

d_{avg} ——加权距离；

x_i, y_i, z_i ——当前待解码点P_i的X，Y，Z坐标；

x_j, y_j, z_j ——邻居预测点P_j的X，Y，Z坐标；

w_x, w_y, w_z ——X，Y，Z方向的权重。

若计算得到的权重w_x, w_y, w_z都等于 0，则使用曼哈顿距离计算距离d。

9.3.10 属性残差值解码

属性残差 residual 的解码过程如下：

zero_run_length 为属性残差零游程值，即属性预测残差连续分布为 0 的个数。

zero_run_length_eq0 代表的是 zero_run_length 是否等于 0。

zero_run_length_minus1 代表 zero_run_length 减 1 后的值。

属性残差零游程值的解码过程为：

解码 zero_run_length_eq0，如果 zero_run_length_eq0 解码值等于 1，则属性残差零游程值等于 0，解码完成；

如果 zero_run_length_eq0 标志位等于 0，表示属性残差零游程值不等于 0，则继续解码 zero_run_length_minus1，即 zero_run_length = zero_run_length_minus1 + 1。

```
if (zero_run_length_equal_zero)
    zero_run_length = 0
else {
    zero_run_length = zero_run_length_minus1 + 1
}
```

计算最大延迟限制点数 maxLatency，取值范围为[1, 2^{17}]，maxLatency = 1 << (coeff_length_control_log2_minus8 + 8)。

获得零游程值 zero_run_length 并解析，具体步骤如下：

a) 当 zero_run_length 等于 maxLatency 时，则当前点对应的残差值为 0，同时零游程值减 1，继续解析零游程值直到零游程值为 0；

b) 当 zero_run_length 大于 0 且不等于 maxLatency 时，则当前点对应的残差值为 0，同时零游程值减 1，直到零游程值为 0；

c) 当 zero_run_length 等于 0 时，按以下方法解析当前点对应的残差值：

1) 利用索引值选择上下文解码一个标志位代表第一解码系数是否等于 0，利用第一解码系数得到第一属性系数。

2) 如果第一属性系数等于 0，继续利用索引值选择上下文解码一个标志位代表第二解码系数是否等于 0，利用第二解码系数得到第二属性系数。

3) 如果第一属性系数以及第二属性系数都等于 0，利用索引值选择上下文解码第三解码系数，利用第三解码系数得到第三属性系数。

4) 如果第一属性系数等于 0 但第二属性系数不等于 0，利用索引值选择上下文解码第二解码系数，利用第二解码系数得到第二属性系数，继续利用索引值选择上下文解码第三解码系数，利用第三解码系数得到第三属性系数。

5) 如果第一属性系数不等于 0，利用索引值选择上下文解码第一解码系数，利用第一解码系数得到第一属性系数，继续利用索引值选择上下文解码第二解码系数，利用第二解码系数得到第二属性系数。

6) 利用第一属性系数绝对值和第二属性系数绝对值的大小关系自适应选择索引值，利用选择的上下文解码第三解码系数，利用第三解码系数得到第三属性系数。

color_level 代表颜色值。当 isComponentNoneZero = true,颜色值等于颜色分量残差绝对值减一；当 isComponentNoneZero = false,颜色值等于颜色分量残差绝对值。

color_level_eq0 代表的是颜色值是否等于 0。

color_component_sign 表示颜色分量残差值的符号。

color_level_eq1 表示颜色分量残差值是否等于 1。

color_level_minus2_parity 表示颜色分量残差值减去二的奇偶性。

color_level_minus2_div2_eq0 表示颜色分量残差值减去二除以二后是否等于 0。

color_level_minus2_div2_minus1 表示颜色分量残差值减去二除以二后再减去一的值。

color_level_golomb 表示采用 K 阶指数哥伦布解析所得颜色相关值。

refl_minus1_parity 表示反射率残差绝对值减一后的奇偶性。

refl_minus1_div2_eq0 表示反射率残差绝对值减一再除二后值是否等于 0。

refl_minus1_div2_eq1 表示反射率残差绝对值减一再除二后值是否等于 1。

refl_minus1_div2_minus2 表示反射率残差绝对值减一除二后再减二的值。

residual_sign 代表的是反射率残差的符号位。

p_ctxPrefix 代表的是指数哥伦布解码的前缀码。

p_ctxSufffix 代表的是指数哥伦布解码的后缀码。

attr_type 表示解码属性信息类别。attr_type 的值为 0、1、2 分别表示 Y（或 R）、U（或 G）、V（或 B），值为 3 表示反射率 R。

K[attr_type]表示指数哥伦布编码的阶数。解码颜色属性时 K[attr_type]的值默认等于 color_golomb_num （attr_type = 0,1,2），解码反射率属性时 K[3]的值等于 refl_golomb_num。

current_window_size 表示当前自适应指数哥伦布编码滑动窗口的元素个数。

golomb_k_for_val_upper 表示颜色属性编码的默认指数哥伦布编码阶数对应的编码值上界，初始化方法：golomb_k_for_val_upper = 1 << (color_golomb_num - 1) + 1 << (color_golomb_num - 2)。

golomb_k_for_val_lower 表示颜色属性编码的默认指数哥伦布编码阶数对应的编码值下界，初始化方法：若 color_golomb_num > 1，则 golomb_k_for_val_lower = 1<< (color_golomb_num - 1) - 1<< (color_golomb_num - 2)；若 color_golomb_num <= 1,则 golomb_k_for_val_lower = 1。

input_sliding_window_sum 表示滑动窗口内元素的和，由计算当前解码元素的前 golomb_sliding_window_size 个使用指数哥伦布编码的残差值或变换系数的和得到，首先将其初始化 input_sliding_window_sum = 0；滑动窗口的大小为 golomb_sliding_window_size，当滑动窗口未填满时，向滑动窗口末尾添加当前使用指数哥伦布编码的残差值或变换系数 rear,更新 input_sliding_window_sum = input_sliding_window_sum + rear；当滑动窗口填满时，先删除滑动窗口首位元素 first，更新 input_sliding_window_sum = input_sliding_window_sum - first,再向滑动窗口末尾添加当前使用指数哥伦布编码的残差值或变换系数 rear,更新 input_sliding_window_sum = input_sliding_window_sum + rear。

input_sliding_window_avg 表示滑动窗口内元素的平均值，input_sliding_window_avg = input_sliding_window_sum >> golomb_group_size_log2。

当 order_switch = 0 时，颜色残差解码顺序是 YUV/RGB；

color_first_comp_zero 表示 Y 或者 R 分量残差值是否为 0；

color_component[0]表示 Y/R 分量残差值；

color_component[1]表示 U/G 分量残差值；

color_component[2]表示 V/B 分量残差值；

color_second_comp_zero 表示 U 或者 G 分量残差值是否为 0。

当 order_switch = 1 时，颜色残差解码顺序是 UYV/GRB；

color_first_comp_zero 表示 U 或者 G 分量残差值是否为 0；

color_component[0]表示 U/G 分量残差值；

color_component[1]表示 Y/R 分量残差值；

color_component[2]表示 V/B 分量残差值；

color_second_comp_zero 表示 Y 或者 R 分量残差是否为 0。

9.3.11 属性残差绝对值反量化

输入：量化参数 attr_quant_param，属性残差绝对值 absolute_residual

输出：反量化属性残差绝对值 dequantized_absolute_residual

属性残差绝对值反量化过程如下：

```
offset = 1 << (decoderShiftBit - 1)
qStepSize = DriveInerseQuantstepSize[attr_quant_param]
absolute_dequantized_residual=(absolute_residual * qStepSize + offset) >> decoderShiftBit
```

其中 decoderShiftBit = 6，量化步长查找表 DriveInerseQuantstepSize 的查找表定义如下：

```
const uint32_t DriveInerseQuantstepSize[180] = {
    64,        70,        76,        83,        91,        99,        108,       117,
    128,       140,       152,       166,       181,       197,       215,       235,
    256,       279,       304,       332,       362,       395,       431,       470,
    512,       558,       609,       664,       724,       790,       861,       939,
    1024,      1117,      1218,      1328,      1448,      1579,      1722,      1878,
    2048,      2233,      2435,      2656,      2896,      3158,      3444,      3756,
    4096,      4467,      4871,      5312,      5793,      6317,      6889,      7512,
    8192,      8933,      9742,      10624,     11585,     12634,     13777,     15024,
    16384,     17867,     19484,     21247,     23170,     25268,     27554,     30048,
    32768,     35734,     38968,     42495,     46341,     50535,     55109,     60097,
    65536,     71468,     77936,     84990,     92682,     101070,    110218,    120194,
    131072,    142935,    155872,    169979,    185364,    202141,    220436,    240387,
    262144,    285870,    311744,    339959,    370728,    404281,    440872,    480774,
    524288,    571740,    623487,    679917,    741455,    808563,    881744,    961548,
    1048576,   1143480,   1246974,   1359835,   1482910,   1617125,   1763488,   1923097,
    2097152,   2286960,   2493948,   2719670,   2965821,   3234251,   3526975,   3846194,
    4194304,   4549753,   4971027,   5478275,   5965232,   6547206,   7064091,   7669584,
    8388608,   9256395,   9942054,   10737418,  11671107,  12782640,  14128182,  15790321,
    16777216,  18295684,  19951585,  21757357,  23726566,  25874004,  28215802,  30769550,
    33554432,  36591368,  39903169,  43514715,  47453133,  51748008,  56431603,  61539100,
    67108864,  73182735,  79806339,  87029429,  94906266,  103496017, 112863206, 123078199,
    134217728, 146365470,  159612677, 174058859, 189812531,  206992033, 225726413, 246156398,
    268435456, 292730940,  319225354, 348117717}
```

9.3.12 属性重建值解码

9.3.12.1 概述

当前待解码点的解码残差值表示为$\widehat{R}_j$，当前待解码点的属性预测值表示为$A_j^{'}$，当前待解码点的反射率属性或颜色的每个通道（R，G，B 通道之一或 Y，U，V 通道之一）的属性重建值$\widehat{A}_j$计算见式（41）：

$$\widehat{A}_j = \mathrm{Clip}(\mathrm{MinRange}, \mathrm{MaxRange}, \widehat{R}_j + \widetilde{A}_j) \quad (41)$$

式中：

$\widehat{A}_j$ ——当前待解码点的属性重建值；

$\widehat{R}_j$ ——当前待解码点的解码残差值；

$\widetilde{A}_j$ ——当前待解码点的属性预测值；

MinRange ——属性重建值下界；

MaxRange ——属性重建值上界。

当 outputBitDepth < 16 时，MinRange = 0，MaxRange = (1 << outputBitDepth)-1；否则重建值的范围不作限制。

9.3.12.2 颜色属性残差二次预测解码

当前待解码点颜色属性的解码残差值表示为 residual[i]，i = 0,1,2 分别对应 R，G，B 通道。

当前待解码点颜色属性的属性预测值表示为 predictor[i]，i = 0,1,2 分别对应 R，G，B 通道。

当前待解码点颜色属性的属性重建值表示为 reconValue[i]，i = 0,1,2 分别对应 R，G，B 通道。

如果 cross_component_pred = 1，当前待解码点的属性重建值解码如下：

```
int residualPrevComponent = 0
for (int i = 0; i < 3; i++) {
  reconValue[i] = Clip (0, 255, residual + predictor[i] + residualPrevComponent)
  if (i <= 1) {
    residualPrevComponent = reconValue[i] - predictor[i]
  }
}
```

9.3.13 变换算法

9.3.13.1 概述

基于变换的属性解码过程如下：

第一步，几何解码点重排序；

第二步，熵解码；

第三步，反量化；

第四步，预测变换树构建；

第五步，属性变换系数解码；

第六步，属性变换系数的反变换与预测；

第七步，属性重建值计算。

属性解码过程：

根据 9.3.2 ，9.3.3 ，9.3.4 对几何解码点进行重排序，根据 9.3.10 进行熵解码获得量化后的属性变换系数，残差系数和直流残差系数，根据 9.3.13.2 进行反量化获得属性变换系数，残差系数和直流残差系数，根据 9.3.13.3 构建预测变换树结构，并基于预测变换树结构根据 9.3.13.4 解码变换系数，9.3.13.5 对解析的变换系数反变换及预测，最后根据 9.3.13.6 完成属性重建值的计算。

9.3.13.2 反量化

属性变换系数绝对值，直流残差系数绝对值和属性残差系数绝对值的反量化过程如下：

属性变换系数绝对值和直流残差系数绝对值的量化参数 coeffQuantParam 等于 aps 中的属性量化参数 attr_quant_param。如果存在属性残差层，即变换残差层标志 trans_res_layer = 1，coeffQuantParam = attr_quant_param + attr_transform_qp_delta。属性残差系数绝对值的量化参数 resLayerQuantParam 等于属性量化参数 attr_quant_param。计算方式如下：

```
coeffQuantParam = attr_quant_param
```

```
if (trans_res_layer) {
    coeffQuantParam += attr_transform_qp_delta
}
resLayerQuantParam = attr_quant_param
```

输入量化参数 Qp 和待反量化的系数 residual，通过量化步长查找表和移位计算获得反量化后的值 dequantizedValue。对于属性变换系数绝对值和直流残差系数绝对值，Qp = coeffQuantParam；对于属性残差系数绝对值，Qp = resLayerQuantParam。反量化过程具体如下：

```
index = Qp
offset = 1 << (decoderShiftBit - 1)
quantstepSize = DriveInerseQuantstepSize[index]
dequantizedValue = (residual * quantstepSize + offset) >> decoderShiftBit
```

其中 decoderShiftBit = 6，量化步长查找表 DriveInerseQuantstepSize 的 LUT 定义如下：

```
const uint32_t DriveInerseQuantstepSize[180] = {
    64,        70,        76,        83,        91,        99,        108,       117,
    128,       140,       152,       166,       181,       197,       215,       235,
    256,       279,       304,       332,       362,       395,       431,       470,
    512,       558,       609,       664,       724,       790,       861,       939,
    1024,      1117,      1218,      1328,      1448,      1579,      1722,      1878,
    2048,      2233,      2435,      2656,      2896,      3158,      3444,      3756,
    4096,      4467,      4871,      5312,      5793,      6317,      6889,      7512,
    8192,      8933,      9742,      10624,     11585,     12634,     13777,     15024,
    16384,     17867,     19484,     21247,     23170,     25268,     27554,     30048,
    32768,     35734,     38968,     42495,     46341,     50535,     55109,     60097,
    65536,     71468,     77936,     84990,     92682,     101070,    110218,    120194,
    131072,    142935,    155872,    169979,    185364,    202141,    220436,    240387,
    262144,    285870,    311744,    339959,    370728,    404281,    440872,    480774,
    524288,    571740,    623487,    679917,    741455,    808563,    881744,    961548,
    1048576,   1143480,   1246974,   1359835,   1482910,   1617125,   1763488,   1923097,
    2097152,   2286960,   2493948,   2719670,   2965821,   3234251,   3526975,   3846194,
    4194304,   4549753,   4971027,   5478275,   5965232,   6547206,   7064091,   7669584,
    8388608,   9256395,   9942054,   10737418,  11671107,  12782640,  14128182,  15790321,
    16777216,  18295684,  19951585,  21757357,  23726566,  25874004,  28215802,  30769550,
    33554432,  36591368,  39903169,  43514715,  47453133,  51748008,  56431603,  61539100,
    67108864,  73182735,  79806339,  87029429,  94906266,  103496017, 112863206, 123078199,
    134217728, 146365470, 159612677, 174058859, 189812531, 206992033, 225726413, 246156398,
    268435456, 292730940, 319225354, 348117717}
```

对反量化后的属性变换系数绝对值，直流残差系数绝对值左移 k_frac_bits 位，dequantizedValue = dequantizedValue << k_frac_bits。

9.3.13.3 预测变换树构建

基于排序点云数据以及排序点云数据之间的距离构建多层预测变换树结构过程如下：

对于有 N 个几何点的点云，构建 M 层预测变换结构。将 N 个点作为最低层（M 层）的节点，计算当前点 i 与下一点 i+1 的欧式距离d_i。如果$d_i^2 = (x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 + (z_i - z_{i+1})^2$小于距离阈值$th_m$，点 i 和点 i+1 为变换模式，两点合并构成它们在 M-1 层的父节点。这些父节点构成 M-1 层的节点，按照合并的先后顺序排列，父节点的几何坐标为其两个子节点的中点位置。如果d_i^2大于距离阈值th_m，点 i 为预测模式，继续处理下一个点。

对 M-1 层的所有节点，按照上述步骤合并，构成 M-2 层的节点，以此类推，对每一层的节点进行合并。当第 m 层的节点数少于 128 或者第 m 层的节点数大于等于 m+1 层的节点数的一半时，对所有节点进行两两合并，无需判断距离关系。当一层内没有节点合并时停止。

距离阈值的初始值，即最底层的距离阈值th_M计算方法见式（42）：

$$th_M = \mathrm{Max}(1, (\mathrm{meanBB} * \mathrm{meanBB}/\mathrm{geom\_num\_points})/\mathrm{ratio}) \quad (42)$$

式中：

th_M ——最底层的距离阈值；

meanBB ——点云平均空间包围盒边长；

geom_num_points ——点云点数；

ratio ——预设比例，对于颜色属性$\mathrm{ratio} = 2^{\mathrm{color_init_pred_trans_ratio}}$，对于反射率属性$\mathrm{ratio} = 2^{\mathrm{refl_init_pred_trans_ratio}}$。

第m层的距离阈值th_m的更新方法见式（43）：

$$th_m = \mathrm{Max}(1, th_{m+1} * N_{m+1}/N_m) \quad (43)$$

式中：

th_m ——第m层的距离阈值；

th_{m+1} ——第（m+1）层的距离阈值；

N_{m+1} ——第（m+1）层的节点数；

N_m ——第m层的节点数。

在构建预测变换树的过程中，同一层的距离阈值th_m也在更新。在第 m 层的构建过程中，统计连续出现预测模式节点和变换模式节点的个数，记为N_p和N_t。对于点 i，如果N_p大于阈值T_p，更新距离阈值$th_m = th_m * 2$；如果N_t大于阈值T_t且距离阈值th_m不为 1，更新距离阈值$th_m = \mathrm{Max}(1, th_m/2)$。使用更新后的距离阈值$th_m$对后续点间距离进行比较。

划分结束后得到一个 M 层预测变换树结构，基于此结构，进行分层变换和预测。

9.3.13.4 属性变换系数解码

按 9.3.10 进行熵解码并按照 9.3.13.2 进行反量化，得到属性变换系数，自上而下逐层获取对应的属性变换系数。首先获取第一层的重建 DC 系数，并按各点的处理顺序和对应的解码方式，依次获取其他层对应的 AC 系数或 DC 残差系数。

9.3.13.5 属性变换系数的反变换与预测

基于 M 层预测变换树结构以及对应的解码方式分别对各节点进行点云属性解码。根据 9.3.13.4 得到的属性变换系数（第一层的重建 DC 系数和其他层的重建 AC 系数），和反量化后的直流残差系数（重建 DC 残差系数）进行反变换和预测。

a） 对属性变换系数（第一层的重建 DC 系数和其他层的重建 AC 系数），直流残差系数（重建 DC 残差系数）进行基于层的矫正。

分别对第一层节点的重建DC系数乘以$(C_{weight})^{M-1}$，对第m层节点的重建AC系数乘以$(C_{weight})^{M-m}$，

对第 m+1 层节点的重建 DC 残差系数乘以$(C_{weight})^{M-m-1}$，(m=1,2,…M-1)，其中$C_{weight} = N >>$ k_frac_bits。N的查找表见表 57：

表57 k_frac_bits 和 N 查找表

k_frac_bits	1	2	3	4	5	6	7	8	9	10	11	12	13
N	2	5	11	22	45	90	181	362	724	1448	2896	5792	11585

b） 对矫正后的属性变换系数进行反变换与预测。

属性变换系数的反变换：对第 m 层的第 j 个节点，输入为 j 节点的属性变换系数，逆变换输出对应两个子节点的重建 DC 系数a_1和a_2，见式（44）和式（45）：

$$a_1 = (b_1 + b_2)/2 \quad (44)$$

$$a_2 = (b_1 - b_2)/2 \quad (45)$$

式中：

a_1 ——对应子节点1的重建DC系数；

a_2 ——对应子节点2的重建DC系数；

b_1 ——第m层的第j个节点的重建DC系数；

b_2 ——第m层的第j个节点的重建AC系数。

如此，遍历 m 层的所有节点。

c） 直流残差系数的预测补偿。

对第 m+1 层的第 j 个节点，如果其没有父节点，则在层内搜索其距离最近的 K（K=3）个已经计算过重建 DC 系数的节点，搜索范围为前向 128 点和后序 128 点，利用 9.3.9.1 方法计算 j 节点的 DC 预测值。将 DC 预测值与重建直流残差系数相加得到 j 节点的重建 DC 系数。如此，遍历 m+1 层的所有节点。

若已重建 DC 系数的节点为 k_1 个，步骤如下：

1） 当 k_1 = 0 时，颜色属性测值为{128, 128, 128}，反射率属性的预测值为 0；

2） 当 k_1 <= K 时，该 m 个节点为邻居预测点；

3） 当 k_1 > K 时，颜色属性采用 K（K = 3）个距离最近点及最多 13 个等距离点作为当前点的邻居预测点（参考 9.3.5.1）；反射率属性采用 K（K = 3）个距离最近点作为当前点的邻居预测点；

基于 m=1, 2, …, M-2, M-1，自上而下的遍历 M 层结构的每一层并且循环执行上述步骤 2）和步骤 3），进行相关计算。最终在预测变换树的第 M 层，将每个节点的重建 DC 系数DC_{rec}通过下述公式右移 k_frac_bits 位，得到每个节点的属解码值A_{rec}，见式（46）。

$$A_{rec} = Sign(DC_{rec}) * \left((Sign(DC_{rec}) * DC_{rec} + 2^{k\_frac\_bits-1}) >> k\_frac\_bits \right) \quad (46)$$

9.3.13.6 属性重建值计算

如果 aps.trans_res_layer = 1，则根据 9.3.10 进行熵解码获得每个点的残差层残差值，并与 9.3.13.5 中得到的属性解码值逐点相加进行截断，反之 aps.trans_res_layer = 0，则直接对 9.3.13.5 中得到的属性解码值进行截断，获得属性重建值。

输入：当前待解码点的属性解码值 inverse_value；当前待解码点的属性残差值 residual。

输出：当前待解码点的属性重建值 reconstruct_value。

属性重建值的计算过程如下：

```
if (trans_res_layer)
    reconstruct_value = Clip(MinRange, MaxRange, inverse_value+ residual)
else
    reconstruct_value = Clip(MinRange, MaxRange, inverse_value)
```

9.3.13.7 基于跨属性参考的多层预测变换解码方法

对于包含多种属性类型的点云，可以利用先解码的属性信息辅助解码后解码的属性。在根据 9.3.13.3 构建预测变换树时，使用综合距离替代原有的几何距离。

当前点 i 与下一点 i+1 的综合距离d_i^2的算法方式见式（47）：

$$d_i^2 = disGeom_i^2 + \left(\left(\left(\lambda * Round\left(\frac{maxGeom \ll 10}{maxAttr}\right) + 2^{19}\right) \gg 20\right) * disAttr_i^2\right) \gg 10 \quad \text{(47)}$$

式中：

d_i^2 ——综合平方距离

$disGeom_i^2$ ——几何信息平方距离；

$disAttr_i^2$ ——属性信息平方距离；

$maxGeom$ ——几何信息的最大值；

$maxAttr$ ——属性信息的最大值；

λ ——属性信息距离在综合距离计算过程中的权重。

其中，按照 9.3.7 中方法和参数设置计算$maxGeom$，$maxAttr$和λ。

几何信息平方距离$disGeom_i$的计算方法见式（48）：

$$disGeom_i^2 = (x_i - x_{i+1})^2 + (y_i - y_{i+1})^2 + (z_i - z_{i+1})^2 \quad \text{(48)}$$

式中：

$disGeom_i^2$ ——几何信息平方距离；

x_j, y_j, z_j ——当前点i的X，Y，Z坐标；

$x_{i+1}, y_{i+1}, z_{i+1}$ ——下一点i+1的X，Y，Z坐标。

属性信息平方距离$disAttr_i^2$的计算方法见式（49）：

反射率属性：

$$disAttr_i^2 = |Refl_i - Refl_{i+1}|^2 \quad \text{(49)}$$

式中：

$disAttr_i^2$ ——反射率属性信息平方距离；

$Refl_i$ ——当前点i的反射率重建值；

$Refl_{i+1}$ ——下一点i+1的反射率重建值。

颜色属性见式（50）：

$$disAttr_i^2 = |Color[0]_i - Color[0]_{i+1}|^2 + |Color[1]_i - Color[1]_{i+1}|^2 + |Color[2]_i - Color[2]_{i+1}|^2 \quad \text{(50)}$$

式中：

$disAttr_i^2$ ——颜色属性信息平方距离；

$Color_i$ ——当前点i的颜色重建值；

$Color_{i+1}$ ——下一点i+1的颜色重建值。

在根据 9.3.13.5 进行属性变换系数的反变换与直流残差系数的预测补偿时，使用 9.3.7 中的多属性点云邻居预测点查找方法和参数设置替换原有的邻居预测点查找方法。

9.3.14 预测变换算法

9.3.14.1 概述

基于预测变换算法的属性解码过程如下：

第一步，几何解码点重排序；

第二步，几何解码点分组；

第三步，分组变换系数解码；

第四步，属性变换系数的反量化；

第五步，属性变换系数的反变换；

第六步，属性预测值计算；

第七步，属性重建值获取。

属性解码过程：

根据 9.3.2 ，9.3.3 ，9.3.4 获得的顺序，将点云点按照 9.3.14.2 或 9.3.14.3 所述方法进行分组，按顺序依次解码每组点。根据 9.3.14.4 解码点云点的属性变换系数并按照 9.3.14.5 对其进行反量化，对反量化后的变换系数按照 9.3.14.6 对其进行反变换获取当前解码点云的属性残差值，对于当前解码点云，根据 9.3.14.7 获得当前解码点云的属性预测值，最后根据 9.3.14.8 获得属性重建值。

9.3.14.2 几何解码点分组(颜色属性)

对重排序之后的点云几何点依次分组，通过计算得到初始右移位数L_{base}，根据L_{base}获取当前右移位数 L，根据当前右移位数 L 进行分组，设定最大变换阶数为 color_max_trans_num，具体的分组规则如下：

a） 初始右移位数L_{base}的计算公式见式（51）：

$$L_{base} = \mathrm{Max}\left(3, 3 * \left(\mathrm{maxNodeSizeLog2} - \mathrm{Round}\left(\frac{\left\lceil \log_2\left(\frac{\mathrm{slice\_num\_points}}{K_{mean}}\right)\right\rceil}{2}\right)\right)\right) \quad \text{.....(51)}$$

式中：

L_{base} ——初始右移位数；

maxNodeSizeLog2 ——片包围盒最大边长对数尺寸；

slice_num_points ——当前点云片的点数；

K_{mean} ——分组后的平均点数，设为4。

其中，maxNodeSizeLog2计算方式见式（52）：

$$\mathrm{maxNodeSizeLog2} = \mathrm{Max}\left(\mathrm{NodeizeLog2[0]}, \mathrm{Max}\left(\mathrm{NodeSizeLog2[1]}, \mathrm{NodeSizeLog2[2]}\right)\right) \quad \text{.....(52)}$$

式中：

maxNodeSizeLog2 ——片包围盒最大边长对数尺寸；

NodeSizeLog2[0] ——片包围盒X方向对数尺寸，等于slice_bounding_box_sizeXLog2；

NodeSizeLog2[1] ——片包围盒Y方向对数尺寸，等于slice_bounding_box_sizeYLog2；

NodeSizeLog2[2] ——片包围盒Z方向对数尺寸，等于slice_bounding_box_sizeZLog2。

b） 设定L的初始值设为L_{base}，将希尔伯特码右移L位后相同的点，划分为同一个宏块，若当前块内的点数小于等于最大变换阶数 color_max_trans_num，则该宏块内的点成为一组，若当前块内的点大于 color_max_trans_num 个点，则对该块内的点进行细分组，细分组的规则如下：

细分组时，获取当前右移位数L，当前细分组的右移位数$L_1 = L - 1$，然后将希尔伯特码右移L_1位后相同的点归为新的一细分组，继续判断该新的一细分组内的点数是否大于 color_max_trans_num，若大

于 color_max_trans_num，则继续重复操作$L_1 = L_1 - 1$，直至每个细分组内的点数都小于等于color_max_trans_num。

c） 动态调整L的大小，具体的规则为，统计 3 个连续分组组内的点数的和并将其值右移 3 位后记作B，如果B的值小于 2，则$L = L + 1$；如果B的值大于 8，则$L = L - 1$；否则，L不变。L为大于等于 3 的整数，每 32 个分组调整一次L的值。

d） 判断当前点与上一个已分组的点是否为重复点，若当前点为重复点则截断该组，之后的重复点依次独立分组，即按点数为 1 进行分组。

9.3.14.3 几何解码点分组(反射率属性)

对重排序之后的几何点依次分组，通过计算得到初始右移位数，根据当前右移位数进行分组，设定最大变换阶数为 refl_max_trans_num，当 outputBitDepth = 8 时，最大变换阶数 refl_max_trans_num 设置为 4；当 outputBitDepth = 16 时，最大变换阶数 refl_max_trans_num 设置为 2。具体的分组规则如下：

a） refl_max_trans_num 不等于 4 时，初始右移位数$L_{base} = 3$；refl_max_trans_num 等于 4 时，初始右移位数L_{base}的计算公式见式（53）：

$$L_{base} = Max(3, (maxBits - minBits)/3 * 3) + shift \quad (53)$$

式中：

L_{base} ——初始右移位数；

maxBits ——最大位数；

minBits ——最小位数；

shift ——初始偏移。

其中，最大位数maxBits为点云包围盒的对数尺寸之和，见式（54），并设置上限值为 32：

$$maxBits = Min(32, NodeSizeLog2[0] + NodeSizeLog2[1] + NodeSizeLog2[2]) \quad (54)$$

式中：

maxBits ——最大位数；

NodeSizeLog2[0] ——片包围盒X方向对数尺寸，等于slice_bounding_box_sizeXLog2；

NodeSizeLog2[1] ——片包围盒Y方向对数尺寸，等于slice_bounding_box_sizeYLog2；

NodeSizeLog2[2] ——片包围盒Z方向对数尺寸，等于slice_bounding_box_sizeZLog2。

最小位数minBits为点云点数slice_num_points取对数，见式（55）：

$$minBits = \lceil \log_2 slice\_num\_points \rceil \quad (55)$$

式中：

minBits ——最小位数；

slice_num_points ——当前点云片的点数。

初始偏移shift通过判断属性量化参数attrQuantParam获得，如果attrQuantParam大于等于 32 则shift等于 12，否则shift等于-6。attrQuantParam = aps.attr_quant_param + ash.qp_offset。

b） 将希尔伯特码右移位L_{base}位后相同的点划分到同一个宏块。对每个宏块中的点进行分组，若当前点为重复点则截断该组，之后的重复点依次独立分组，即按点数为 1 进行分组。

若第 j 个组内点的个数K_j大于 refl_max_trans_num，则通过以下方法将第 j 个块进一步分组：依次取 refl_max_trans_num 个点，直至该块内各个分组中的点数都小于等于 refl_max_trans_num。

c） refl_max_trans_num > 2 且 shift > 0 时，动态调整右移位数L。

设定L的初始值设为L_{base}，统计前序 N 组的总几何点个数（N = 8），计算 N 组的平均点数，如果平均点数小于 2，则$L = L + 1$；如果平均点数大于 refl_max_trans_num，则$L = L - 1$；否则，L不变。L为大于等于 1 的整数，每 8 个分组调整一次L的值。

9.3.14.4 分组变换系数解码

基于 9.3.14.2 或 9.3.14.3 中的分组方法可获得各个组对应的点数，每次解码变换系数时，总系数个数N_{All}小于等于变换系数最大缓存限制参数 maxNumofCoeff。设对应 DC 系数的个数为N_{DC}，则 AC 系数的个数为$N_{All}-N_{DC}$。按顺序解析各组的变换参数：对偶数次的缓存单元内的变换系数，按$[DC_1,...,DC_c,\{AC_1\},...,\{AC_c\}]$排列；对奇数次的缓存单元内的变换系数，按$[\{AC_1\},...,\{AC_c\},DC_1,...,DC_c]$排列（从第 0 次开始计数），其中$\{AC_i\}$表示对应第 i 组的所有 AC 系数。

计算最大延迟限制点数 maxLatency,取值范围为[1, 2^{17}]，等于变换系数最大缓存限制参数 maxNumofCoeff 与最大延迟限制参数 coeffLengthControl 的乘积，见式（56）。

$$maxLatency = maxNumofCoeff * coeffLengthControl \quad (56)$$

式中：

maxLatency ——最大延迟限制点数；

maxNumofCoeff ——变换系数最大缓存限制参数；

coeffLengthControl ——最大延迟限制参数。

按 9.3.10 解析对应的变换系数值。

9.3.14.5 预测残差反量化

属性变换系数反量化过程如下：

属性变换系数的量化参数 Qp 等于 aps 中的属性量化参数 attr_quant_param 加固定偏移值 72，再加量化参数偏移值 QpOffset，即 Qp = attr_quant_param + 72 + QpOffset。其中，对于颜色属性，对亮度变换直流系数设置 QpOffset = qp_offset_dc，对亮度变换交流系数设置 QpOffset = qp_offset_ac；对色度变换直流系数设置 QpOffset = chroma_qp_offset_dc，对色度变换交流系数设置 QpOffset = chroma_qp_offset_ac。

当 color_qp_adjust_flag = 1 时，开启逐点自适应量化工具。当前待解码组的最后一点与其最近邻居点的距离为 D，比较 D 与预设距离阈值 T 的大小关系。距离阈值 T 的计算方法见式（57）：

$$T = Max(4, 2^{groupShiftBits/3+4}/scalar) \quad (57)$$

式中：

T ——预设距离阈值；

groupShiftBits ——颜色初始右移位数，等于 9.3.14.2 中的颜色初始右移位数L_{base}

scalar ——点云几何量化前后点数比值，等于 ash.color_qp_adjust_scalar。

如果$D > T$，颜色属性 DC 变换系数的量化参数QP_{DC}自适应调整见式（58）：

$$QP_{DC} = Max(72, QP - 8) \quad (58)$$

式中：

QP_{DC} ——颜色属性DC变换系数的量化参数；

QP ——颜色属性的量化参数。

对于反射率属性，对反射率变换直流系数设置 QpOffset = qp_offset_dc，对反射率变换交流系数设置 QpOffset = qp_offset_ac。

反量化过程具体如下：

反量化过程具体如下，输入量化参数 Qp 和待反量化的系数 residual，通过量化步长查找表和移位计算获得反量化后的值 dequantizedValue。具体过程如下：

```
index = Qp
offset = 1 << (decoderShiftBit - 1)
quantstepSize = DriveInerseQuantstepSize[index]
```

```
dequantizedValue = (residual * quantstepSize + offset) >> decoderShiftBit
```

其中 decoderShiftBit = 6,量化步长查找表 DriveInerseQuantstepSize 的 LUT 定义如下：

```
const uint32_t DriveInerseQuantstepSize[180] = {
    64,        70,        76,        83,        91,        99,        108,       117,
    128,       140,       152,       166,       181,       197,       215,       235,
    256,       279,       304,       332,       362,       395,       431,       470,
    512,       558,       609,       664,       724,       790,       861,       939,
    1024,      1117,      1218,      1328,      1448,      1579,      1722,      1878,
    2048,      2233,      2435,      2656,      2896,      3158,      3444,      3756,
    4096,      4467,      4871,      5312,      5793,      6317,      6889,      7512,
    8192,      8933,      9742,      10624,     11585,     12634,     13777,     15024,
    16384,     17867,     19484,     21247,     23170,     25268,     27554,     30048,
    32768,     35734,     38968,     42495,     46341,     50535,     55109,     60097,
    65536,     71468,     77936,     84990,     92682,     101070,    110218,    120194,
    131072,    142935,    155872,    169979,    185364,    202141,    220436,    240387,
    262144,    285870,    311744,    339959,    370728,    404281,    440872,    480774,
    524288,    571740,    623487,    679917,    741455,    808563,    881744,    961548,
    1048576,   1143480,   1246974,   1359835,   1482910,   1617125,   1763488,   1923097,
    2097152,   2286960,   2493948,   2719670,   2965821,   3234251,   3526975,   3846194,
    4194304,   4549753,   4971027,   5478275,   5965232,   6547206,   7064091,   7669584,
    8388608,   9256395,   9942054,   10737418,  11671107,  12782640,  14128182,  15790321,
    16777216,  18295684,  19951585,  21757357,  23726566,  25874004,  28215802,  30769550,
    33554432,  36591368,  39903169,  43514715,  47453133,  51748008,  56431603,  61539100,
    67108864,  73182735,  79806339,  87029429,  94906266,  103496017, 112863206, 123078199,
    134217728, 146365470,  159612677, 174058859, 189812531,  206992033, 225726413, 246156398,
    268435456, 292730940,  319225354, 348117717}
```

9.3.14.6 属性变换系数的反变换

对解码获得的变换系数，按照分组每次取对应个数的点作为待变换系数，利用反变换矩阵进行反变换获得属性残差值。

对 9.3.14.5 反量化后获得的变换系数矩阵Q_{1*N}，按照 9.3.14.2 或 9.3.14.3 中的分组，每次取一个组中的点作为待反变换系数，利用对应维度的反变换矩阵T_{N*N}进行反变换获得属性残差矩阵R_{1*N}，见式（59）。

$$R_{1*N} = Q_{1*N} T_{N*N} \quad (59)$$

式中：

R_{1*N} ——属性残差矩阵；

Q_{1*N} ——属性变换系数矩阵；

T_{N*N} ——反变换矩阵。

属性残差矩阵R_{1*N}中的每一个元素r还需要进行如下的移位操作，获得最终的属性残差值 residual，见式（60）。

$$residual = (r + 2^{17}) >> 18 \quad (60)$$

式中：

residual ——属性残差值；

r ——属性残差矩阵中的元素。

不同维度的反变换矩具体如下：

```
int64_t DCTmatrix2[2][2] = {
   {362, 362},
   {362, -362}
}

int64_t DCTmatrix3[3][3] = {
   {296, 296, 296},
   {362, 0, -362},
   {209, -418, 209}
}

int64_t DCTmatrix4[4][4] = {
   {256, 256, 256, 256},
   {256, 256, -256, -256},
   {256, -256, -256, 256},
   {256, -256, 256, -256}
}

int64_t DCTmatrix5[5][5] = {
   {229, 229, 229, 229, 229},
   {308, 190, 0, -190, -308},
   {262, -100, -324, -100, 262},
   {190, -308, 0, 308, -190},
   {100, -262, 324, -262, 100}
}

int64_t DCTmatrix6[6][6] = {
   {209, 209, 209, 209, 209, 209},
   {286, 209, 77, -77, -209, -286},
   {256, -0, -256, -256, -0, 256},
   {209, -209, -209, 209, 209, -209},
   {148, -296, 148, 148, -296, 148},
   {77, -209, 286, -286, 209, -77}
}

int64_t DCTmatrix7[7][7] =
   {{194, 194, 194, 194, 194, 194, 194},
   {267, 214, 119, 0, -119, -214, -267},
```

```
    {247, 61, -171, -274, -171, 61, 247},
    {214, -119, -267, -0, 267, 119, -214},
    {171, -247, -61, 274, -61, -247, 171},
    { 119, -267, 214, -0, -214, 267, -119},
    { 61, -171, 247, -274, 247, -171, 61}
  }

  int64_t DCTmatrix8[8][8] = {
    {181, 181, 181, 181, 181, 181, 181, 181},
    {251, 213, 142, 50, -50, -142, -213, -251},
    {236, 98, -98, -236, -236, -98, 98, 236},
    {213, -50, -251, -142, 142, 251, 50, -213},
    {181, -181, -181, 181, 181, -181, -181, 181},
    {142, -251, 50, 213, -213, -50, 251, -142},
    {98, -236, 236, -98, -98, 236, -236, 98}
  }
```

9.3.14.7 属性预测值计算

对于颜色属性，采用 9.3.5 的邻居预测点搜索方法，9.3.9.1 和 9.3.9.3 的预测值计算方法，获得属性预测值。

对于反射率属性，采用 9.3.6 的邻居预测点搜索方法，9.3.9.1 和 9.3.9.2 的预测值计算方法，获得属性预测值。当 refl_group_pred_flag = 1 时，若第 j 个组内点的个数$K_j < 3$，则该组K_j个点共用该组第一个点的预测值，剩余$K_j - 1$个点不进行预测；若当前组的点数$K_j >= 3$，该组K_j个点分别进行预测。

对于多属性(颜色和反射率)，采用 9.3.7 的邻居预测点搜索方法，9.3.9.1 、9.3.9.2 、9.3.9.3 和 9.3.9.5 的预测值计算方法，获得属性预测值。

其中，对重复点采用 9.3.9.4 的预测值计算方法。

9.3.14.8 属性重建值计算

输入：当前待解码点的属性残差值 residual；当前待解码点的属性预测值 predictor。

输出：当前待解码点的属性重建值 reconstruct_value。

属性重建值的计算过程如下：

```
reconstruct_value = Clip(MinRange, MaxRange, predictor + residual)
```

当 outputBitDepth < 16 时，MinRange = 0，MaxRange = (1 << outputBitDepth) - 1。

9.4 生成重建点云

所有完成几何解码和属性解码后的片点云组成一个整体的几何量化点云，几何量化点云的坐标进行反量化生成反量化重建点云。几何量化点云的坐标为(X^k, Y^k, Z^k)，几何量化步长有效值为geometry_quant_step_significand，几何量化步长指数geometry_quant_step_exponent，表示几何量化步长有效值中小数部分位数，点云几何反量化后的坐标计算如下，符合IEEE Std 754™-2019中binary64浮点数格式，见式（61）、式（62）、式（63）：

$$\hat{x}^k = X^k * geometry\_quant\_step\_significand / 2^{geometry\_quant\_step\_exponent} \quad (61)$$

$$\hat{y}^k = Y^k * geometry\_quant\_step\_significand / 2^{geometry\_quant\_step\_exponent} \quad (62)$$

$$\hat{z}^k = Z^k * geometry\_quant\_step\_significand / 2^{geometry\_quant\_step\_exponent} \quad (63)$$

式中：

$\hat{x}^k, \hat{y}^k, \hat{z}^k$ ——点云几何反量化后的X，Y，Z坐标；

X^k, Y^k, Z^k ——几何量化点云的X，Y，Z坐标；

$geometry\_quant\_step\_significand$ ——几何量化步长有效值；

$geometry\_quant\_step\_exponent$ ——几何量化步长指数。

反量化后的点云坐标进行反平移。点云的平移位移为$(x^{origin}, y^{origin}, z^{origin})$，点云反平移后的坐标见式（64）、式（65）、式（66）：

$$x^k = \hat{x}^k + x^{origin} \quad (64)$$

$$y^k = \hat{y}^k + y^{origin} \quad (65)$$

$$z^k = \hat{z}^k + z^{origin} \quad (66)$$

式中：

x^k, y^k, z^k ——点云反平移后的X，Y，Z坐标；

$\hat{x}^k, \hat{y}^k, \hat{z}^k$ ——点云几何反量化后的X，Y，Z坐标；

$x^{origin}, y^{origin}, z^{origin}$ ——点云的平移位移的XYZ分量，x^{origin} = frame_header.bounding_box_offset_x，y^{origin} = frame_header.bounding_box_offset_y，z^{origin} = frame_header.bounding_box_offset_z。

重建点云输出得到解码点云。

附 录 A
（规范性）
伪起始码方法

本附录定义防止在位流中出现伪起始码的方法。起始码的形式、含义，以及为了使起始码字节对齐而进行填充的方法见 7.1.1 和 5.9.2 。

解码时应按以下方法处理：每读入一个字节时，检查前面读入的两个字节和当前字节，如果这三个字节构成位串‘0000 0000 0000 0000 0000 0010’，丢弃当前字节的最低两个有效位。丢弃一个字节最低两个有效位可采用任意等效的方式，本文件不做规定。

在解码时对于序列头、几何头、属性头、帧头、几何片头、属性片头的保留字节中的数据不应采用上述方法。

备注：为了防止出现伪起始码，编码时可按照以下方法处理：写入一位时，如果该位是一个字节的第二最低有效位，检查该位之前写入的 22 位，如果这 22 位都是‘0’，在该位之前插入‘10’，该位成为下一个字节的最高有效位。

附 录 B
（规范性）
档次和级别

B.1 概述

档次和级别提供了一种定义本文件的语法和语义的子集的手段。档次和级别对位流进行了各种限制，同时也就规定了对某一特定位流解码所需要的解码器能力。档次是本文件规定的语法、语义及算法的子集。符合某个档次规定的解码器应完全支持该档次定义的子集。级别是在某一档次下对语法元素和语法元素参数值的限定集合。在给定档次的情况下，不同级别往往意味着对解码器能力和存储器容量的不同要求。

本附录描述了不同档次和级别所对应的各种限制。所有未被限定的语法元素和参数可以取任何本文件所允许的值。如果一个解码器能对某个档次和级别所规定的语法元素的所有允许值正确解码，则称此解码器在这个档次和级别上符合本文件。如果一个位流中不存在某个档次和级别所不允许的语法元素，并且其所含有的语法元素的值不超过此档次和级别所允许的范围，则认为此位流在这个档次和级别上符合本文件。

profile_id 和 level_id 定义了位流的档次和级别。

B.2 档次

本文件定义的档次见表 B.1。

表 B.1 档次

profile_id的值	档次
0	禁止
1	基础档次 Base Profile
2	基准档次 Main Profile
3～15	保留

基础档次和基准档次的语法元素区别见表 B.2，其中 N/A 表示该档次不涉及此参数。

表 B.2 档次的语法元素

头信息	语法元素	档次	
		基础档次	基准档次
序列头信息	profile_id	0	1
属性头信息	transform	0	0,1,2
	transform_segment_size_upper	N/A	$0\sim2^{16}-1$
	transform_segment_size_lower	N/A	$0\sim2^{16}-1$
	k_frac_bits	N/A	1～48
	attr_transform_qp_delta	N/A	0～32
	trans_res_layer	N/A	0,1
	max_num_of_coeff_log2_minus8	N/A	0～10

表 B.2 档次的语法元素（续）

头信息	语法元素	档次	
		基础档次	基准档次
属性头信息	qp_offset_dc	N/A	-16～16
	qp_offset_ac	N/A	-16～16
	color_max_transNum	N/A	0～8
	chroma_qp_offset_dc	N/A	-16～16
	chroma_qp_offset_ac	N/A	-16～16
	color_qp_adjust_flag	N/A	0,1
	refl_max_trans_num	N/A	0～8
	refl_group_pred_flag	N/A	0,1
属性片头信息	color_init_pred_trans_ratio	N/A	-16～16
	refl_init_pred_trans_ratio	N/A	-16～16
	color_qp_adjust_scalar	N/A	0～127

B.3 级别

本文件定义的级别见表 B.3。

表 B.3 级别

level_id的值	级别
0	禁止
1	最大几何位深：20； 最大属性位深：8； 单通道最大属性个数：1； 三通道最大属性个数：0； 每片最大点数：2^{20}； 每秒最大帧数：10。
2	最大几何位深：20； 最大属性位深：8； 单通道最大属性个数：1； 三通道最大属性个数：0； 每片最大点数：2^{20}； 每秒最大帧数：20。
3	最大几何位深：20； 最大属性位深：8； 单通道最大属性个数：1； 三通道最大属性个数：0； 每片最大点数：2^{20}； 每秒最大帧数：30。

表 B.3 级别（续）

level_id的值	级别
4	最大几何位深：32； 最大属性位深：16； 单通道最大属性个数：1； 三通道最大属性个数：1； 每片最大点数：2^{20}； 每秒最大帧数：10。
5	最大几何位深：32； 最大属性位深：16； 单通道最大属性个数：1； 三通道最大属性个数：1； 每片最大点数：2^{20}； 每秒最大帧数：30。
6	最大几何位深：32； 最大属性位深：16； 单通道最大属性个数：1； 三通道最大属性个数：1； 每片最大点数：2^{30}； 每秒最大帧数：10。
7	最大几何位深：32； 最大属性位深：16； 单通道最大属性个数：1； 三通道最大属性个数：1； 每片最大点数：2^{30}； 每秒最大帧数：30。
8	最大几何位深：32； 最大属性位深：32； 单通道最大属性个数：128； 三通道最大属性个数：128； 每片最大点数：2^{20}； 每秒最大帧数：10。
9	最大几何位深：32； 最大属性位深：32； 单通道最大属性个数：128； 三通道最大属性个数：128； 每片最大点数：2^{30}； 每秒最大帧数：30。
10～255	保留

B.4 语义取值范围

序列头语法元素取值范围见表 B.4。

表 B.4 序列头取值范围

语法元素	表示	范围
profile_id	u(4)	1～15
level_id	u(8)	1～255
frame_rate_code	u(4)	0～15
geom_remove_duplicate_flag	u(1)	0,1
attribute_present_flag	u(1)	0,1
max_num_attributes_minus1	u(7)	0～127
multi_attributes_set_flag	u(1)	0～1

几何头语法元素取值范围见表 B.5。

表 B.5 几何头取值范围

语法元素	表示	范围
geometry_quant_step_significand	u(21)	$1\sim2^{21}-1$
geometry_quant_step_exponent	u(5)	0～20
geom_max_tree_size_log2_minus8	ue(v)	0～MaxSlicePointsLog2-8
implicit_geom_partition_flag	u(1)	0,1
single_mode_flag	u(1)	0,1
occupancy_search_range_side_log2	ue(v)	0～MaxSliceDimLog2-1
save_state_flag	u(1)	0,1
lcu_dependency_flag	u(1)	0,1

MaxSlicePointsLog2 是一个 slice 的最大点数取 log2 对数。

MaxSliceDimLog2 是一个 slice 的包围盒的最大边长取 log2 对数。

属性头语法元素取值范围见表 B.6。

表 B.6 属性头取值范围

语法元素	表示	范围
attribute_data_present_flag	u(1)	0,1
attribute_data_num_set_minus1	ue(v)	0～127
multi_attr_group_id	ue(v)	0～127
multi_data_set_flag	u(1)	0,1
attribute_info_num_set_minus1	ue(v)	0～127
output_bit_depth_minus1	ue(v)	0～31
attr_quant_param	ue(v)	0～127
chroma_qp_offset_cb	se(v)	-16～16
chroma_qp_offset_cr	se(v)	-16～16
order_switch	u(1)	0,1

表 B.6 属性头取值范围（续）

语法元素	表示	范围
color_reorder_mode	ue(v)	0,1,2
color_golomb_num	ue(v)	0～8
golomb_group_size_log2	ue(v)	0～8
axis_bias_minus1	ue(v)	0～15
refl_reorder_mode	ue(v)	0,1,2
refl_golomb_num	ue(v)	0～8
pred_fixed_point_frac_bit	ue(v)	0～30
transform	u(2)	0,1,2
max_num_of_neighbours_log2_minus7	u(2)	0～3
cross_component_pred	u(1)	0,1
nearest_pred_param1	ue(v)	0～32
nearest_pred_param2	ue(v)	0～32
pred_dist_weight_group_size_log2	ue(v)	0～7
transform_segment_size_upper	u(16)	$0\sim2^{16}-1$
transform_segment_size_lower	u(16)	$0\sim2^{16}-1$
k_frac_bits	ue(v)	1～14
attr_transform_qp_delta	ue(v)	0～32
trans_res_layer	u(1)	0,1
max_num_of_coeff_log2_minus8	ue(v)	0～9
qp_offset_dc	se(v)	-16～16
qp_offset_ac	se(v)	-16～16
color_maxtransNum	ue(v)	0～8
chroma_qp_offset_dc	se(v)	-16～16
chroma_qp_offset_ac	se(v)	-16～16
color_qp_adjust_flag	u(1)	0,1
refl_max_trans_num	ue(v)	0～8
refl_group_pred	u(1)	0,1
coeff_length_control_log2_minus8	ue(v)	0～9
cross_attr_type_pred	u(1)	0,1
attr_coding_order	u(1)	0,1
cross_attr_type_pred_param1	u(15)	$0\sim2^{15}-1$
cross_attr_Type_pred_param2	u(21)	$0\sim2^{21}-1$

帧头语法元素取值范围见表 B.7。

表 B.7 帧头取值范围

语法元素	表示	范围
frame_idx	ue(v)	0～2^{16}-1
frame_num_slice_minus1	ue(v)	0～2^{16}-1
lcu_node_size_log2_minus1	ue(v)	0～MaxSliceDimLog2-1
geom_num_points_upper	u(16)	0～2^{16}-1
geom_num_points_lower	u(16)	0～2^{16}-1
bounding_box_offset_x_upper	u(16)	0～2^{16}-1
bounding_box_offset_x_lower	u(16)	0～2^{16}-1
bounding_box_offset_y_upper	u(16)	0～2^{16}-1
bounding_box_offset_y_lower	u(16)	0～2^{16}-1
bounding_box_offset_z_upper	u(16)	0～2^{16}-1
bounding_box_offset_z_lower	u(16)	0～2^{16}-1
bounding_box_size_width_upper	u(16)	0～2^{16}-1
bounding_box_size_width_lower	u(16)	0～2^{16}-1
bounding_box_size_height_upper	u(16)	0～2^{16}-1
bounding_box_size_height_lower	u(16)	0～2^{16}-1
bounding_box_size_depth_upper	u(16)	0～2^{16}-1
bounding_box_size_depth_lower	u(16)	0～2^{16}-1

几何片头语法元素取值范围见表 B.8。

表 B.8 几何片头取值范围

语法元素	表示	范围
slice_id	ue(v)	0～2^{16}-1
context_mode	u(1)	0,1
max_num_implicit_qtbt_before_ot	ue(v)	0～MaxSliceDimLog2
min_size_implicit_qtbt	ue(v)	0～MaxSliceDimLog2
gsh_single_mode_flag	u(1)	0,1
planar_mode	u(1)	0,1
slice_bounding_box_offset_x_upper	u(16)	0～2^{16}-1
slice_bounding_box_offset_x_lower	u(16)	0～2^{16}-1
slice_bounding_box_offset_y_upper	u(16)	0～2^{16}-1
slice_bounding_box_offset_y_lower	u(16)	0～2^{16}-1
slice_bounding_box_offset_z_upper	u(16)	0～2^{16}-1
slice_bounding_box_offset_z_lower	u(16)	0～2^{16}-1
slice_bounding_box_sizeXLog2	u(6)	0～32
slice_bounding_box_sizeYLog2	u(6)	0～32
slice_bounding_box_sizeZLog2	u(6)	0～32
slice_num_points_upper	u(16)	0～2^{16}-1
slice_num_points_lower	u(16)	0～2^{16}-1

属性片头语法元素取值范围见表 B.9。

表 B.9 属性片头取值范围

语法元素	表示	范围
slice_id	ue(v)	$0\sim2^{16}-1$
attribute_id	ue(v)	0～127
qp_offset	se(v)	-32～32
color_init_pred_trans_ratio	se(v)	-16～16
refl_init_pred_trans_ratio	se(v)	-16～16
color_qp_adjust_scalar	ue(v)	0～127