ICS 35.040

CCS L 71

团体标准

信息技术 高效图形数据编码

第2部分：点云

Information Technology Efficient Graphical Data Coding

Part 2: Point Cloud

|  |
| --- |
| （征求意见稿） |
| 在提交反馈意见时，请将您知道的相关专利连同支持性文件一并附上。 |

XXXX - XX - XX发布

XXXX - XX - XX实施

新一代人工智能产业技术创新战略联盟

中关村视听产业技术创新联盟 发布

|  |
| --- |
| T/AI |

目次

[前  言 II](#_Toc162878809)

[引  言 III](#_Toc162878810)

[1 范围 1](#_Toc162878811)

[2 规范性引用文件 1](#_Toc162878812)

[3 术语和定义 1](#_Toc162878813)

[4 缩略语 3](#_Toc162878814)

[5 约定 3](#_Toc162878815)

[5.1 概述 3](#_Toc162878816)

[5.2 算术运算符 4](#_Toc162878817)

[5.3 逻辑运算符 4](#_Toc162878818)

[5.4 关系运算符 4](#_Toc162878819)

[5.5 位运算符 4](#_Toc162878820)

[5.6 赋值 5](#_Toc162878821)

[5.7 数学函数 5](#_Toc162878822)

[5.8 结构关系符 10](#_Toc162878823)

[5.9 位流语法、解析过程和解码过程的描述方法 10](#_Toc162878824)

[6 编码位流的结构 13](#_Toc162878825)

[6.1 点云序列 14](#_Toc162878826)

[6.2 点云片 14](#_Toc162878827)

[7 位流的语法和语义 14](#_Toc162878828)

[7.1 语法描述 14](#_Toc162878829)

[7.2 语义描述 30](#_Toc162878830)

[8 解析过程 42](#_Toc162878831)

[8.1 k阶指数哥伦布码 42](#_Toc162878832)

[8.2 ue(v)和se(v)的解析过程 43](#_Toc162878833)

[8.3 ae(v)的解析过程 44](#_Toc162878834)

[9 解码过程 56](#_Toc162878835)

[9.1 解码过程概述 56](#_Toc162878836)

[9.2 几何信息解码过程 56](#_Toc162878837)

[9.3 属性信息解码过程 70](#_Toc162878838)

[9.4 生成重建点云 93](#_Toc162878839)

[附　录　A （规范性）伪起始码方法 95](#_Toc162878840)

[附　录　B （规范性）档次和级别 96](#_Toc162878841)

[B.1 概述 96](#_Toc162878842)

[B.2 档次 96](#_Toc162878843)

[B.3 级别 97](#_Toc162878844)

[B.4 语义取值范围 99](#_Toc162878845)

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 XXX《信息技术 高效图形数据编码》的第2部分。

请注意本文件的某些内容可能涉及专利。本文件的发布机构不承担识别专利的责任。

本文件由新一代人工智能产业技术创新战略联盟/中关村视听产业技术创新联盟AVS标准工作组提出并归口。

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。

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）、何堅、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 《信息技术 高效图形数据编码》标准拟由四部分组成：

第1部分：系统。旨在定义点云编码码流文件格式规范。

第2部分：点云。旨在定义高效的点云解码过程和解码规范。

第3部分：点云编码参考软件。旨在提供符合规范的点云解码参考软件和点云编码参考软件。

第4部分：点云编码符合性测试。旨在提供点云解码符合性测试描述。

本文件为T/AI 《信息技术 高效图形数据编码》的第2部分。

本文件的发布机构提请注意，声明符合本文件时，可能涉及点云编解码技术相关的专利的使用。

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保证，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：

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 高效图形数据编码 第2部分：点云

1. 范围

本标准规定了适应多种点数、数据位深、属性个数和质量要求的点云压缩方法的解码过程。

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。

1. 规范性引用文件

下列文件对于本文件的应用是必不可少的。凡是注日期的引用文件，仅注日期的版本适用于本文件。凡是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 754TM-2019 IEEE Standard for Floating-Point Arithmetic

1. 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 mortonCode

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

3.24

希尔伯特码 hilbertCode

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

3.25

片 slice

一系列的语法元素，用于表示一部分或者整个点云。

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

有限个二进制比特位的有序序列，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

颜色 color

点对应的三维RGB或YUV信号。

3.31

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。

3.32

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。

3.33

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。

3.34

语法元素 syntax element

位流中的数据单元解析后的结果。

3.35

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，其中每个比特的值对应该节点的一个子节点的占用情况。

3.36

字节 byte

8位的位串。

3.37

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。

3.38

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。

1. 缩略语

下列缩略语适用于本文件。

AEC 算术熵编码（Arithmetic Entropy Code）

APS 属性参数集（Attribute Parameter Set）

ASH 属性片头（Attribute Slice Header）

Attr 属性（Attribute）

Cnt 数量（Count）

FPS 帧率（Frame Per Second）

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）

Pred 预测（Prediction）

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）

Trans 变换（Transform）

1. 约定
   1. 概述

本文件中使用的数学运算符和优先级参照C语言。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。

* 1. 算术运算符

算术运算符定义见表1。

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 |
| ab | 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 |
|  | 除法运算，不做截断或四舍五入 |
|  | 自变量i取由a到b（含b）的所有整数值时，函数的累加和 |
| 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 |

* 1. 逻辑运算符

逻辑运算符定义见表2。

1. 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 |

* 1. 关系运算符

关系运算符定义见表3。

1. 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 |

* 1. 位运算符

位运算符定义见表4。

1. 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 |

* 1. 赋值

赋值运算定义见表5。

1. 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 |

* 1. 数学函数
     1. 通用函数

数学函数定义见式（1）～式（4）。

(1)

式中：

x ——自变量x；

i ——下界；

j ——上界。

(2)

式中：

x ——自变量x；

y ——自变量y。

(3)

式中：

x ——自变量x；

y ——自变量y。

(4)

式中：

x ——自变量x。

* + 1. 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 = xn-1xn-2…x1x0

Y = yn-1yn-2…y1y0

Z = zn-1zn-2…z1z0

输出：对应的三维莫顿码M的二进制表示即为：

M = xn-1yn-1zn-1xn-2yn-2zn-2…x1y1z1x0y0z0

下表6举例展示了该转换过程：

1. 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 |
| xn-1…x0 | yn-1…y0 | zn-1…z0 | xn-1yn-1zn-1…x0y0z0 | ... |

* + 1. 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 = xn-1xn-2…x1x0

Y = yn-1yn-2…y1y0

Z = zn-1zn-2…z1z0

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查询表如下：

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}}}

* 1. 结构关系符

结构关系符定义见表7。

1. 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 |

* 1. 位流语法、解析过程和解码过程的描述方法
     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。

1. 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conditon为TRUE，则重复执行primary语句和subsequent语句直到condition不为TRUE。\*/ |  |
| 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。

* + 1. 函数
       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。

* + - 1. 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。

* + - 1. 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。

* + - 1. 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。

* + - 1. 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。

1. 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。

* + - 1. 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。

1. 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 |  |
| } |  |

* + - 1. 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。

* + 1. 描述符

描述符表示不同语法元素的解析过程，见表11。

1. 描述符

|  |  |
| --- | --- |
| 描述符 | 说明 |
| ae(v) | 算术熵编码的语法元素。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。 |

* + 1. 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。

1. 编码位流的结构
   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。

* 1. 点云片

点云片由几何片头、几何信息、属性片头和属性信息组成，在本标准中片及点云片具有相同的含义。

1. 位流的语法和语义
   1. 语法描述
      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。

1. 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B定义的方法处理伪起始码问题。

* + 1. 点云序列，几何片，属性片

点云序列定义见表13。

1. 点云序列定义

|  |  |
| --- | --- |
| PCC\_sequence( ) { | Descriptor |
| 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) |  |
| } |  |

几何片定义见表14。

1. 几何片定义

|  |  |
| --- | --- |
| geometry\_slice( ) { | Descriptor |
| geometry\_slice\_header( ) |  |
| general\_geometry\_data\_bitstream( ) |  |
| } |  |

属性片定义见表15。

1. 属性片定义

|  |  |
| --- | --- |
| attribute\_slice( ) { | Descriptor |
| attribute\_slice\_header( ) |  |
| general\_attribute\_data\_bitstream( ) |  |
| } |  |

* + 1. 序列头

序列头SPS定义见表16。

1. 序列头定义

|  |  |
| --- | --- |
| sequence\_header( ) { | Descriptor |
| **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( ) |  |
| } |  |

* + 1. 字节对齐

字节对齐定义见表17。

1. 字节对齐定义

|  |  |
| --- | --- |
| byte\_alignment( ) { | Descriptor |
| while (!byte\_aligned( )) |  |
| alignment\_bit\_equal\_to\_one /\* equal to 1 \*/ |  |
| } |  |

* + 1. 几何头

几何头GPS定义见表18。

1. 几何头定义

|  |  |
| --- | --- |
| geometry\_header( ) { | Descriptor |
| **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( ) |  |
| } |  |

* + 1. 属性头

属性头APS定义见表19。

1. 属性头定义

|  |  |
| --- | --- |
| attribute\_header( ) { | Descriptor |
| **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) |
| } |  |
| 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)) { |  |
| **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( ) |  |
| } |  |

* + 1. 点云帧头

帧头frame\_header定义见表20。

1. 帧头定义

|  |  |
| --- | --- |
| frame\_header( ) { | Descriptor |
| **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) |
| **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( ) |  |
| } |  |

* + 1. 几何片头

几何片头GSH定义见表21。

1. 几何片头定义

|  |  |
| --- | --- |
| geometry\_slice\_header( ) { | Descriptor |
| **geometry\_slice\_header\_start\_code** | f(32) |
| **slice\_id** | ue(v) |
| **marker\_bit** | f(1) |
| **context\_mode** | ue(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) |
| **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( ) |  |
| } |  |

* + 1. 属性片头

属性片头ASH定义见表22。

1. 属性片头定义

|  |  |
| --- | --- |
| attribute\_slice\_header( ) { | Descriptor |
| **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( ) |  |
| } |  |

* + 1. 几何信息
       1. 通用几何语法

通用几何语法见表23。

1. 通用几何语法

|  |  |
| --- | --- |
| general\_geometry\_data\_bitstream( ) { | Descriptor |
| **geometry\_slice\_payload\_start\_code** | f(32) |
| geometry\_data( ) |  |
| byte\_alignment( ) |  |
| } |  |

* + - 1. 几何数据语法

几何数据语法见表24。

1. 几何数据语法

|  |  |
| --- | --- |
| geometry\_data( ) { | Descriptor |
| 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 |  |
| 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) |
| } |  |

八叉树数据语法见表25。

1. 八叉树数据语法

|  |  |
| --- | --- |
| lcu\_octree\_data( ) { | Descriptor |
| 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) |
| } |  |

预测树数据语法见表26。

1. 预测树数据语法

|  |  |
| --- | --- |
| lcu\_predtree\_data( ) { | Descriptor |
| **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) |
| } |  |

* + - 1. 几何节点语法

几何节点语法见表27。

1. 几何节点语法

|  |  |
| --- | --- |
| geometry\_node(depth, depthX, depthY, depthZ, partitionSkip, nodeIdx, xN, yN, zN, numSiblings，single\_point\_eligible\_flag\_per\_depth[depth]) { | Descriptor |
| 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) |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

* + 1. 属性信息
       1. 通用属性语法

通用属性语法见表28。

1. 通用属性语法

|  |  |
| --- | --- |
| general\_attribute\_data\_bitstream( ) { | Descriptor |
| 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( ) |  |
| } |  |

通用颜色语法见表29。

1. 通用颜色语法

|  |  |
| --- | --- |
| general\_color\_data\_bitstream( ) { | Descriptor |
| **color\_slice\_payload\_start\_code** | f(32) |
| attribute\_data\_color( ) |  |
| byte\_alignment( ) |  |
| } |  |

通用反射率语法见表30。

1. 通用反射率语法

|  |  |
| --- | --- |
| general\_refl\_data\_bitstream( ) { | Descriptor |
| **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( ) |  |
| } |  |

* + - 1. 属性数据语法

属性数据语法定义见下述各表。

1. 颜色数据

|  |  |
| --- | --- |
| attribute\_data\_color ( ) { | Descriptor |
| 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) |
| } |  |

1. 反射率数据

|  |  |
| --- | --- |
| attribute\_data\_refl( ) { | Descriptor |
| 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 { |  |
| if (!isDuplicatePoint) { |  |
| **residual\_sign** | ae (v) |
| } |  |
| **abs\_level\_minus1\_parity** | ae(v) |
| **abs\_level\_minus1\_div2\_eq0** | ae(v) |
| if (!abs\_level\_minus1\_div2\_eq0) { |  |
| **abs\_level\_minus1\_div2\_eq1** | ae(v) |
| if (!abs\_level\_minus1\_div2\_eq1) { |  |
| **abs\_level\_minus1\_div2\_minus2** | ae(v) |
| } |  |
| } |  |
| residual\_zero\_run\_length=zero\_run\_length\_code( ) |  |
| } |  |
| ­} |  |
| } |  |
| **termination\_bit\_one** /\* equals to 1 \*/ | ae(v) |
| } |  |

1. 零游程编码

|  |  |
| --- | --- |
| zero\_run\_length\_code( ) { | Descriptor |
| **zero\_run\_length\_eq0** | ae(v) |
| if (!zero\_run\_length\_eq0) { |  |
| **zero\_run\_length\_minus1** | ae(v) |
| } |  |
| } |  |

1. 颜色残差编码

|  |  |
| --- | --- |
| color\_residual\_correlation\_code( ) { | Descriptor |
| **color\_first\_comp\_zero** | ae(v) |
| if (!color\_first\_comp\_zero) { |  |
| color\_component[0] = color\_component\_code(true) |  |
| color\_component[1] = color\_component\_code(false) |  |
| color\_component[2] = color\_component\_code(false) |  |
| 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) |  |
| color\_component[2] = color\_component\_code(false) |  |
| 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) |  |
| **color\_component\_sign[2]** | ae(v) |
| } |  |
| } |  |
| } |  |

1. 颜色分量编码

|  |  |
| --- | --- |
| color\_component\_code(isComponentNoneZero) { | Descriptor |
| 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) |
| } |  |
| } |  |
| } 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) |
| } |  |
| } |  |
| } |  |
| } |  |

1. 多反射率数据编码

|  |  |
| --- | --- |
| attribute\_data\_multiple\_refl( ) { | Descriptor |
| 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) |
| } |  |

* + 1. 用户数据

用户数据语法定义见表37。

1. 用户数据定义

|  |  |
| --- | --- |
| userdata\_bitstream( ) { | Descriptor |
| **user\_data\_start\_code** | f(32) |
| while (next\_bits(24) != 0000 0000 0000 0000 0000 0001) { |  |
| **user\_data** | u(8) |
| } |  |
| } |  |

* 1. 语义描述
     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。

1. 帧率代码

|  |  |
| --- | --- |
| frame\_rate\_code的值 | 帧率 |
| 0000 | 禁止 |
| 0001 | 10 fps |
| 0010 | 20 fps |
| 0011 | 30 fps |
| 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，和属性信息；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。

* + 1. 字节对齐

字节对齐比特值1 alignment\_bit\_equal\_to\_one

填充位。值为1。

* + 1. 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[0,221-1]。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。

* + 1. 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。

1. attrIdx含义

|  |  |
| --- | --- |
| attrIdx | 属性信息 |
| 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。

属性输出比特深度减一 output\_bit\_depth\_minus1

无符号整数，取值范围为[0,31]，用于表示属性输出比特深度。属性输出比特深度为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\_num

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)。

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比特部分 transform\_segment\_size\_upper和属性变换点数低16比特部分 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比特位和低16比特位。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\_num

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\_num

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217。

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。

* + 1. 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比特位部分 geom\_num\_points\_upper和点云帧内点数低16比特位部分 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比特位和低16比特位。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比特位部分 bounding\_box\_offset\_x\_upper和包围盒原点x坐标低16比特位部分 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比特位和低16比特位。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比特位部分 bounding\_box\_offset\_y\_upper和包围盒原点y坐标低16比特位部分 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比特位和低16比特位。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比特位部分 bounding\_box\_offset\_z\_upper和包围盒原点z坐标低16比特位部分 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比特位和低16比特位。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比特位部分 bounding\_box\_size\_width\_upper和包围盒宽度低16比特位部分 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比特位和低16比特位。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比特位部分 bounding\_box\_size\_height\_upper和包围盒高度低16比特位部分 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比特位和低16比特位。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比特位部分 bounding\_box\_size\_depth\_upper和包围盒深度低16比特位部分 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比特位和低16比特位。包围盒深度为

bounding\_box\_size\_depth = bounding\_box\_size\_depth\_upper << 16 + bounding\_box\_size\_depth\_lower。

* + 1. 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比特位部分 slice\_bounding\_box\_offset\_x\_upper和片包围盒原点x坐标低16比特位部分 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比特位和低16比特位。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比特位部分 slice\_bounding\_box\_offset\_y\_upper和片包围盒原点y坐标低16比特位部分 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比特位和低16比特位。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比特位部分 slice\_bounding\_box\_offset\_z\_upper和片包围盒原点z坐标低16比特位部分 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比特位和低16比特位。片包围盒原点z坐标为slice\_bounding\_box\_offset\_z = gsh\_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比特位部分 slice\_num\_points\_upper片所含点数低16比特位部分 slice\_num\_points\_lower

16位无符号整数。表示片所含点数的高16比特位和低16比特位。片所含点数为slice\_num\_points = slice\_num\_points\_upper << 16 + slice\_num\_points\_lower。

* + 1. 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。

* + 1. 几何信息
       1. 几何数据语义

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比特数。

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位比特。

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比特，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比特数 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比特数的值，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比特数的值，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。

终止比特 termination\_bit\_one

终止位。值为1。

* + - 1. 几何节点语义

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个比特分别表示当前节点的8个子节点是否被占用，某一位比特值为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个比特 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个比特值。

孤立点Y方向相对坐标的第i个比特 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个比特值。

孤立点Z方向相对坐标的第i个比特 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个比特值。

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。

* + 1. 属性信息
       1. 属性数据语义

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 abs\_level\_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 abs\_level\_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 abs\_level\_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 abs\_level\_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）。

终止比特 termination\_bit\_one

终止位。值为1。

* + 1. 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。

1. 解析过程
   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 = 2leadingZeroBits + k – 2k + 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xi串，xi的值为0或1。

1. 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 |
| ...... | ...... |

* 1. 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。

1. 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. ae(v)的解析过程
     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。

* + 1. 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

* + 1. 二元符号串解析
       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。

* + - 1. 导出二元符号模型
         1. 导出二元符号模型

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。

1. 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 |  | 194 | 1 |
| single\_point\_eligible\_flag\_per\_depth |  | 195 | 1 |
| geom\_single\_flag |  | 196 | 1 |
| occupancy | 见8.3.3.2.2 | 197 | 290 |
| geom\_tree\_type |  | 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 |  | 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 |  | 570 | 1 |
| color\_second\_comp\_zero |  | 571 | 1 |
| abs\_level\_minus1\_parity | 见8.3.3.2.3 | 552 | 1 |
| abs\_level\_minus1\_div2\_eq0 | 见8.3.3.2.3 | 538 | 1 |
| abs\_level\_minus1\_div2\_eq1 | 见8.3.3.2.3 | 542 | 1 |
| abs\_level\_minus1\_div2\_minus2 | 见8.3.3.2.5 | 546 | 4 |

* + - * 1. 确定occupancy的ctxIdxInc

occupancy语法元素包含8个比特b7b6b5b4b3b2b1b0,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9.2.3.1　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9.2.3.1　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9.2.3.1　得到：

ctxIdxInc = bit\_ctx + context。

对ctxIdxInc进行更新，ctxIdxInc = (ctx\_offset < 2) ？ctx\_offset ：(ctxIdxInc+ ctx\_offset)。

* + - * 1. 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

1. 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 |
| 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 |
| 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 |

* + - * 1. 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b0b1…bmbm+1…b2m+k，其中b0b1…bm为前缀码部分prefix，bm+1…b2m+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。

1. 语法元素对应的ctxIdxInc

|  |  |  |
| --- | --- | --- |
|  | 二值化变量 | ctxIdxInc |
| 前缀码 | b0 | 0 |
|  | b1 | 1 |
|  | b2…bm | 2 |
| 后缀码 | bm+1 | 3 |
|  | bm+2…b2m+k | 4 |

* + - * 1. 确定abs\_level\_minus1\_div2\_minus2, color\_minus2\_div2\_minus1[0], color\_minus2\_div2\_minus1[1], color\_minus2\_div2\_minus1[2], color\_minus1\_minus2\_div2\_minus1[0], color\_minus1\_minus2\_div2\_minus1[1], color\_minus1\_minus2\_div2\_minus1[2]的ctxIdxInc

根据以下方法确定color\_minus2\_div2\_minus1、color\_minus1\_minus2\_div2\_minus1和abs\_level\_minus1\_div2\_minus2的ctxIdxInc：

将color\_minus2\_div2\_minus1、color\_minus1\_minus2\_div2\_minus1、abs\_level\_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b0b1…bmbm+1…b2m+k，其中b0b1…bm为前缀码部分prefix，bm+1…b2m+k为后缀码部分suffix，k为指数哥伦布阶数；

如表45所示，abs\_level\_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；

abs\_level\_minus1\_div2\_minus2,color\_minus2\_div2\_minus1、color\_minus1\_minus2\_div2\_minus1共享上下文。

1. 语法元素的bin的编码方式

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 语法元素 | binIdx | | | | |
| 0 | 1 | 2 | 3 | >= 4 |
| abs\_level\_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 |

1. 语法元素ctxctxIdxInc

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 语法元素 | binIdx | | | | | |
| 0 | 1 | [2,m] | m+1 | m+2 | [m+3, 2m+k] |
| abs\_level\_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 |

* + - 1. 二元符号解析
         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。

* + - * 1. decode\_decision

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)

}

* + - * 1. decode\_aec\_stuffing\_bit

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)

}

* + - * 1. decode\_bypass

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)

}

* + - * 1. update\_ctx

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)

}

* + 1. 反二值化方法
       1. 概述

本条定义语法元素的反二值化方法。

* + - 1. 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。

1. 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 |

* + - 1. 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。

1. 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 |

* + - 1. 采用标记位的反二值化方法

由二元符号串根据表49表49　得到synElVal的值。

1. 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 |

1. 解码过程
   1. 解码过程概述

序列解码过程如下：

第一步，解码序列头；

第二步，解码几何头；

第三步，解码属性头；

第四步，解码帧头；

第五步，解码几何片头；

第六步，解码几何信息；

第七步，解码属性片头；

第八步，解码属性信息；

第九步，生成重建点云。

* 1. 几何信息解码过程
     1. 概述

几何信息解码过程如下：

几何划分解码（见9.2.2　）。

几何节点解码（见9.2.3　）。

* + 1. 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比特标识符。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个比特从高位到低位分别表示X,Y,Z方向是否跳过划分，其中比特0表示不跳过划分，比特1表示跳过划分。具体划分类型如下表50所示。

1. 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 |
| partitionSkip | 001 | 010 | 100 | 000 |
| 标识 | 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 | 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

}

* + 1. 几何节点解码

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的比特个数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。

* + - 1. 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

}

解码占位码比特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)

adjChidIdx = adjacentCIdx[i]

for (idx = 0; idx < 4; ++idx) {

ctxChild |= !!(encodedChildNode[adjChidIdx[idx] >> 3] & (1 << (adjChidIdx[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比特,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的取值如下表所示：

1. 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 |

1. 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 |

1. 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

adjChidIdx = adjacentCIdx[i]

for (size\_t idx = 0; idx < 7; ++idx) {

childInformation |=

!!(encodedChildNode[adjChidIdx[idx] >> 3] & (1 << (adjChidIdx[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，完成计算后有效比特位为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，完成计算后有效比特位为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比特,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的取值如下表所示：

1. 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 |

1. 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 |

1. 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 |

* + - 1. 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

* + - 1. 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9.2.3.3　孤立点解码，否则进行9.2.3.1　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

}

* + - 1. 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。

* + - 1. 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

* + - 1. 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

}

}

* + - 1. 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

* + 1. 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。

* + 1. 八叉树宏块几何节点解码过程

如果该宏块的geom\_tree\_type = 0，按照9.2.3　描述的过程进行解码。

* + - 1. 几何宏块状态保存

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。

* + 1. 预测树宏块解码过程

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Pi的几何坐标重建值计算过程如下：

1. 计算待解码顶点Pi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，解析对应的除二比特位数num\_bits\_geom\_residual\_minus1[n][k]，采用二值化形式bm...b4b3b2b1b0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分量所占用比特数的值（k = 0,1,2）。

ctxIdx = 0，解析b0；

ctxIdx = 1 + b0，解析b1；

ctxIdx = 3 + b1，解析b2；

ctxIdx = 5+ b2 + b1 << 1，解析b3；

ctxIdx = 9，解析bm...b4；



计算num\_bits\_geom\_residual\_minus1[n][k]，方法如下：

num\_bits\_geom\_residual\_minus1[n][k] = b0 + b1 << 1 + b2 << 2 + …+ bm << m

基于比特位数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



1. 计算待解码顶点Pi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Pi的残差值的绝对值，，与前序顶点Pi-1到前序顶点的父顶点Pi-2的曼哈顿距离，对码流解码得到残差符号位。其步骤如下：

计算待解码顶点Pi的前序顶点Pi-1到前序顶点的父顶点Pi-2曼哈顿距离。

(5)

式中：

——曼哈顿距离；

,, ——顶点Pi-1的X，Y，Z坐标；

,, ——顶点Pi-2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,,，得到8种可能的残差值组合,,,,,,,。

8种可能的残差值组合分别和前序顶点Pi-1坐标值相加，得到8个可能的顶点Pj及其坐标值(,,)

计算每个可能的顶点Pj到前序顶点的父顶点Pi-2的曼哈顿距离；

(6)

式中：

——曼哈顿距离；

,, ——顶点Pj的X，Y，Z坐标；

,, ——顶点Pi-2的X，Y，Z坐标。

比较和，删除残差值的符号的组合中使得小于的情况，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,...,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Pi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m > M-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Pi-1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Pi残差值的符号。

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

4) 将待解码顶点Pi的残差与前序顶点Pi-1的坐标值相加，得到待解码顶点Pi的几何坐标重建值。

* 1. 属性信息解码过程
     1. 概述

属性信息解码过程如下：

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。

* + 1. 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]= { P0,P1,…,Pidx ,Pidx+1,… Pslice\_num\_points-1},其中Pidx是第idx个索引对应点的莫顿码，Pidx大于或等于Pidx-1，对数组Morder内的点按索引从小到大的顺序进行属性解码。

* + 1. 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5.7.3　），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]= {P0,P1,…,Pidx ,Pidx+1,… Pslice\_num\_points-1},其中Pidx是第idx个索引对应点的希尔伯特码，Pidx大于或等于Pidx-1，对数组Horder内的点按索引从小到大的顺序进行属性解码。

* + 1. 原始顺序

经几何解码后点的重建顺序。

* + 1. 基于替换最远点的邻居预测点全搜索（颜色属性）

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（），点Ps为重排序后的第s个点（，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，Pi和Ps为同一个点。

* + - 1. 确定邻居预测点

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Pi，邻居预测点的计算方法如下：

如果,则无需查找，将{128，128，128}作为预测值；

如果,则P0为邻居预测点；

如果,则P0,P1为邻居预测点；

如果,则Pi-1、Pi-2、Pi-3为邻居预测点；

如果,在Sp中遍历查找当前待解码点Pi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Pj，距离d计算采用曼哈顿距离，定义为：。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Pi-1、Pi-2、Pi-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。这里、和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等于，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，则将d3对应的点移出距离最近点集合并令，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d3与被移出点的距离进行比较，若等于，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。

* + - 1. 确定最远点

如果，在Sp内[查找距离Pi最远的点，假设距离Pi最远点的距离为Dmax，若存在多个与Pi距离为Dmax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。

* + - 1. 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，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Pi为例，将其放入Sp的第m个点，。

如果，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Pi为例，用其替换Sp的第n个点。



* + 1. 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）

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（），点Ps为重排序后的第s个点（，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，Pi和Ps为同一个点。



* + - 1. 确定邻居预测点

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Pi，邻居预测点的计算方法如下：

如果,不进行邻居预测点查找，将{0}作为预测值；

如果,则P0为邻居预测点；

如果,则P0,P1为邻居预测点；

如果,则Pi-1、Pi-2、Pi-3为邻居预测点；

如果,在Sp中遍历查找与当前待解码点Pi(距离最近的3个点作为当前待解码点的邻居预测点Pj，对于Sp中任一点Pj，距离d计算采用曼哈顿距离，定义为：。

* + - 1. 确定最远点

如果，在Sp内[]查找距离Pi最远的点，假设距离Pi最远点的距离为Dmax，若存在多个与Pi距离为Dmax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。

* + - 1. 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，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Pi为例，将其放入Sp的第m个点，。

如果，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Pi为例，用其替换Sp的第n个点。



* + 1. 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（），点Ps为重排序后的第s个点（，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，Pi和Ps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：

(7)

式中：

——属性信息距离；

——待解码点的反射率；

——邻居候选点的反射率。

颜色属性信息距离：

(8)

式中：

——属性信息距离；

——待解码点的颜色；

——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的计算如下。

(9)

式中：

——综合距离；

——几何信息距离；

——属性信息距离；

——几何信息的最大值；

——属性信息的最大值；

——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公式如下：

(10)

式中：

——几何信息的最大值；

——点云包围盒的宽度值；

——点云包围盒的高度值；

——点云包围盒的深度值。

属性信息的最大值的计算公式如下：

对于反射率属性信息：

(11)

式中：

——反射率的最大值；

——反射率的比特深度。

对于颜色属性信息：

(12)

式中：

——颜色的最大值；

——颜色的比特深度。

属性信息距离在综合距离计算过程中的权重计算公式如下：

(13)

式中：

——属性信息距离在综合距离计算过程中的权重；

——权重计算过程中的参数1；

——权重计算过程中的参数2；

——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，在Sp内[]查找距离Pi几何信息距离disGeom最远的点，假设距离Pi最远点的距离为Dmax，若存在多个与Pi距离为Dmax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，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Pi为例，将其放入Sp的第m个点，。

如果，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Pi为例，用其替换Sp的第n个点。



* + 1. 重复点查找

设当前待解码点为Pj，其坐标为,其前一个已解码的点为Pj-1，其坐标为。

如果与相等，定义标志位isDuplicatePoint = 1，则Pj-1作为Pj的邻居预测点。

* + 1. 预测值计算
       1. 根据距离的加权预测计算

当前待解码点Pj的几何坐标为。

查找到的k个邻居预测点的坐标为：，属性重建值为。邻居预测点的距离计算方法如下。

对于颜色属性：

(14)

式中：

——当前待解码点与邻居预测点n的曼哈顿距离；

——点Pj的X，Y，Z坐标；

——邻居预测点n的X，Y，Z坐标。

对反射率属性：

(15)

式中：

——当前待解码点与邻居预测点n的曼哈顿距离；

——点Pj的X，Y，Z坐标；

——邻居预测点n的X，Y，Z坐标；

——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为，k个邻居预测点的权重计算如下：



(16)

式中：

——邻居预测点n的权重；

——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为：

(17)

式中：

——当前待解码点Pj的属性预测值；

——邻居预测点n的权重；

——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如下：

(18)

式中：

——邻居预测点n的权重；

——当前待解码点与邻居预测点n的曼哈顿距离。

——缩放因子，计算方式如下：

(19)

式中：

——缩放因子；

——反射率属性预测精度值。

* + - 1. 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，如果满足下述公式的条件，则选择k个邻居预测点中与当前点最近的点的属性重建值作为当前点的属性预测值。

(20)

式中：

——k个邻居预测点中与当前点最近的点的属性重建值；

——k个邻居预测点中与当前点最远的点的属性重建值。

* + - 1. 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Pj的几何坐标为。

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：，属性重建值为。

k个距离小于最大距离值的邻居预测点的权重为：

(21)

式中：

——邻居预测点n的权重；

——当前待解码点Pj的X，Y，Z坐标；

——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：，属性重建值为。



r个同距离邻居预测点的优化权重为：

(22)

式中：

——邻居预测点m的权重；

——当前待解码点Pj的X，Y，Z坐标；

——同距离邻居预测点m的X，Y，Z坐标；

——优化系数，当attr\_quant\_param = 0时，；当attr\_quant\_param > 0时，。

当前待解码点的属性预测值为：

(23)

式中：

——当前待解码点Pj的属性预测值；

——k个距离小于最大距离值的邻居预测点中的点n的权重；

——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；

——r个同距离邻居预测点中的点m的优化权重；

——r个同距离邻居预测点中的点m的属性重建值。

* + - 1. 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。

* + - 1. 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。

基于属性值变化的加权距离的计算方式如下：使用固定窗口统计已解码的前个点在三个方向上的属性值变化程度，计算权重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,解码完N个点后更新权重。步骤如下：

1. 前n个已解码点的属性重建值为，设X，Y，Z方向的属性变化，X，Y，Z方向的点数，当前待解码点为。
2. 计算前n个已解码点的中的最大距离：

(24)

式中：

——最大距离；

——点的X，Y，Z坐标；

——点的前一点的X，Y，Z坐标。

若,则：

(25)

(26)

式中：

——X方向的属性变化；

——点的属性重建值；

——点的前一点的属性重建值；

——缩放系数；

——最大距离；

——X方向的点数。

若,则：

(27)

(28)

式中：

——Y方向的属性变化；

——点的属性重建值；

——点的前一点的属性重建值；

——缩放系数；

——最大距离；

——Y方向的点数。

若,则

(29)

(30)

式中：

——Z方向的属性变化；

——点的属性重建值；

——点的前一点的属性重建值；

——缩放系数；

——最大距离；

——Z方向的点数。

其中，缩放系数计算如下：

(31)

式中：

——缩放系数；

——点云点数。

——点云包围盒表面积，计算方式如下：

(32)

1. 计算属性值变化均值：

(33)

(34)

(35)

式中：

——X，Y，Z方向的属性变化均值；

——X，Y，Z方向的属性变化；

——X，Y，Z方向的点数。

d) 计算权重：

(36)

(37)

(38)

式中：

——X，Y，Z方向的权重；

——X，Y，Z方向的属性变化均值。

e) 计算加权距离：

(39)

式中：

——加权距离；

——当前待解码点Pi的X，Y，Z坐标；

——邻居预测点Pj的X，Y，Z坐标；

——X，Y，Z方向的权重。

若计算得到的权重都等于0，则使用曼哈顿距离计算距离。

* + 1. 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, 217]，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。

abs\_level\_minus1\_parity表示反射率残差绝对值减一后的奇偶性

abs\_level\_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

abs\_level\_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。

abs\_level\_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。

* + 1. 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}

* + 1. 属性重建值解码

当前待解码点的解码残差值表示为，当前待解码点的属性预测值表示为，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）的属性重建值计算如下：

(40)

式中：

——当前待解码点的属性重建值；

——当前待解码点的解码残差值；

——当前待解码点的属性预测值；

——属性重建值下界；

——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。

* + - 1. 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]

}

}

* + 1. 变换算法
       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。

* + - 1. 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 = l，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。

* + - 1. 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。如果小于距离阈值，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。如果大于距离阈值，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计算方法如下：

(41)

式中：

——最底层的距离阈值；

——点云平均空间包围盒边长；

——点云点数；

——预设比例，对于颜色属性，对于反射率属性。

第m层的距离阈值的更新方法如下：

(42)

式中：

——第m层的距离阈值；

——第（m+1）层的距离阈值；

——第（m+1）层的节点数；

——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，记为和。对于点i，如果大于阈值，更新距离阈值；如果大于阈值且距离阈值不为1，更新距离阈值。使用更新后的距离阈值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。

* + - 1. 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。

* + - 1. 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，对第m层节点的重建AC系数乘以，对第m+1层节点的重建DC残差系数乘以，(m=1,2,…M-1)。其中。的查找表见表57：

1. 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系数和：

(43)

(44)

式中：

——对应子节点1的重建DC系数；

——对应子节点2的重建DC系数；

——第m层的第j个节点的重建DC系数；

——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k1个，步骤如下：

1) 当k1 = 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k1 <= K时，该m个节点为邻居预测点；

3) 当k1 > 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。最终预测变换树的第M层，每个节点的重建DC系数即为该点的属性解码值。

* + - 1. 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)

* + - 1. 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的综合距离的算法方式如下：

(45)

式中：

——综合平方距离

——几何信息平方距离；

——属性信息平方距离；

——几何信息的最大值；

——属性信息的最大值；

——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，和。

几何信息平方距离的计算方法为：

(46)

式中：

——几何信息平方距离；

,, ——当前点i的X，Y，Z坐标；

,, ——下一点i+1的X，Y，Z坐标。

属性信息平方距离的计算方法为：

反射率属性：

(47)

式中：

——反射率属性信息平方距离；

——当前点i的反射率重建值；

——下一点i+1的反射率重建值。

颜色属性：

(48)

式中：

——颜色属性信息平方距离；

——当前点i的颜色重建值；

——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。

* + 1. 预测变换算法
       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。

* + - 1. 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，根据获取当前右移位数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) 初始右移位数的计算公式如下：

(49)

式中：

——初始右移位数；

——片包围盒最大边长对数尺寸；

——当前点云片的点数；

——分组后的平均点数，设为4。

其中，计算方式如下：

(50)

式中：

——片包围盒最大边长对数尺寸；

——片包围盒X方向对数尺寸，等于slice\_bounding\_box\_sizeXLog2；

——片包围盒Y方向对数尺寸，等于slice\_bounding\_box\_sizeYLog2；

——片包围盒Z方向对数尺寸，等于slice\_bounding\_box\_sizeZLog2。

b） 设定的初始值设为，将希尔伯特码右移位后相同的点，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，当前细分组的右移位数，然后将希尔伯特码右移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，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，如果的值小于2，则；如果的值大于8，则；否则，不变。为大于等于3的整数，每32个分组调整一次的值。

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。

* + - 1. 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；refl\_max\_trans\_num等于4时，初始右移位数的计算公式如下：

(51)

式中：

——初始右移位数；

——最大比特数；

——最小比特数；

——初始偏移。

其中，最大比特数为点云包围盒的对数尺寸之和，并设置上限值为32：

(52)

式中：

——最大比特数；

——片包围盒X方向对数尺寸，等于slice\_bounding\_box\_sizeXLog2；

——片包围盒Y方向对数尺寸，等于slice\_bounding\_box\_sizeYLog2；

——片包围盒Z方向对数尺寸，等于slice\_bounding\_box\_sizeZLog2。

最小比特数为点云点数取对数：

(53)

式中：

——最小比特数；

——当前点云片的点数。

初始偏移通过判断属性量化参数获得，如果大于等于32则等于12，否则等于-6。 = + 。

b） 将希尔伯特码右移位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。

设定的初始值设为，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；如果平均点数大于refl\_max\_trans\_num，则；否则，不变。为大于等于1的整数，每8个分组调整一次的值。

* + - 1. 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，则AC系数的个数为。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[DC1,...,DCC,{AC1},...,{ACC}]排列；对奇数次的缓存单元内的变换系数，按[{AC1},...,{ACC}，DC1,...,DCC]排列（从第0次开始计数），其中{ACi}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, 217]，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。

(54)

式中：

——最大延迟限制点数；

——变换系数最大缓存限制参数；

——最大延迟限制参数。

按9.3.10解析对应的变换系数值。

* + - 1. 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的计算方法如下：

(55)

式中：

——预设距离阈值；

——颜色初始右移位数，等于9.3.14.2中的颜色初始右移位数

——点云几何量化前后点数比值，等于ash.color\_qp\_adjust\_scalar。

如果，颜色属性DC变换系数的量化参数自适应调整为：

(56)

式中：

——颜色属性DC变换系数的量化参数；

——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}

* + - 1. 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，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进行反变换获得属性残差矩阵。

(57)

式中：

——属性残差矩阵；

——属性变换系数矩阵；

——反变换矩阵。

属性残差矩阵中的每一个元素还需要进行如下的移位操作，获得最终的属性残差值residual。

(58)

式中：

——属性残差值；

——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}

}

* + - 1. 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，则该组个点共用该组第一个点的预测值，剩余个点不进行预测；若当前组的点数，该组个点分别进行预测。

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。

* + - 1. 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。

* 1. 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，几何量化步长有效值为，几何量化步长指数，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：

(59)

(60)

(61)

式中：

——点云几何反量化后的X，Y，Z坐标；

——几何量化点云的X，Y，Z坐标；

——几何量化步长有效值；

——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，点云反平移后的坐标计算如下：

(62)

(63)

(64)

式中：

——点云反平移后的X，Y，Z坐标；

——点云几何反量化后的X，Y，Z坐标；

——点云的平移位移的X，Y，Z分量，；；。

重建点云输出得到解码点云。

1. （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。

1. （规范性）档次和级别
   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。

* 1. 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 | 0～216-1 |
| transform\_segment\_size\_lower | N/A | 0～216-1 |
| 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 |
| 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 |

* 1. 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：220；  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：220；  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：220；  每秒最大帧数：30。 |
| 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：220；  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：220；  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：230；  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：230；  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：220；  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：230；  每秒最大帧数：30。 |
| 10～255 | 保留 |

* 1. 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) | 1～221-1 |
| 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 |
| 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) | 0～216-1 |
| transform\_segment\_size\_lower | u(16) | 0～216-1 |
| 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) | 0～215-1 |
| cross\_attr\_Type\_pred\_param2 | u(21) | 0～221-1 |

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) | 0～216-1 |
| frame\_num\_slice\_minus1 | ue(v) | 0～216-1 |
| lcu\_node\_size\_log2\_minus1 | ue(v) | 0～MaxSliceDimLog2-1 |
| geom\_num\_points\_upper | u(16) | 0～216-1 |
| geom\_num\_points\_lower | u(16) | 0～216-1 |
| bounding\_box\_offset\_x\_upper | u(16) | 0～216-1 |
| bounding\_box\_offset\_x\_lower | u(16) | 0～216-1 |
| bounding\_box\_offset\_y\_upper | u(16) | 0～216-1 |
| bounding\_box\_offset\_y\_lower | u(16) | 0～216-1 |
| bounding\_box\_offset\_z\_upper | u(16) | 0～216-1 |
| bounding\_box\_offset\_z\_lower | u(16) | 0～216-1 |
| bounding\_box\_size\_width\_upper | u(16) | 0～216-1 |
| bounding\_box\_size\_width\_lower | u(16) | 0～216-1 |
| bounding\_box\_size\_height\_upper | u(16) | 0～216-1 |
| bounding\_box\_size\_height\_lower | u(16) | 0～216-1 |
| bounding\_box\_size\_depth\_upper | u(16) | 0～216-1 |
| bounding\_box\_size\_depth\_lower | u(16) | 0～216-1 |

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) | 0～216-1 |
| 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) | 0～216-1 |
| slice\_bounding\_box\_offset\_x\_lower | u(16) | 0～216-1 |
| slice\_bounding\_box\_offset\_y\_upper | u(16) | 0～216-1 |
| slice\_bounding\_box\_offset\_y\_lower | u(16) | 0～216-1 |
| slice\_bounding\_box\_offset\_z\_upper | u(16) | 0～216-1 |
| slice\_bounding\_box\_offset\_z\_lower | u(16) | 0～216-1 |
| 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) | 0～216-1 |
| slice\_num\_points\_lower | u(16) | 0～216-1 |

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) | 0～216-1 |
| 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 |