班　级　1901015
学　号　19010100277

西安电子科技大学

本科毕业设计论文

题　　目　G-PCC Trisoup 点云几何信息编码优化

学　　院　通信工程学院

专　　业　通信工程

学生姓名　姚凯

导师姓名　张伟

摘要

关键词： 帧间预测 运动补偿 点云压缩 预测树

ABSTRACT

Keywords: Inter prediction Motion compensation Point cloud compression Prediction tree

目录

第一章 绪论

1.1 研究意义和背景

近年来，三维点云模型已经成为继音频、图像、视频之后的新一代数字化媒体。点云是三维物体和场景的主要数据表示形式之一，用 x、y 和 z 坐标描述几何形状，每个点都有额外的属性，如颜色、法线、透明度和材料属性。它们不需要像 Mesh 网格那样进行繁琐的三角计算，也不需要复杂的预处理[1]。因其具有相对于传统二维平面媒体更高的视觉自由度和细节表达优势，点云在沉浸和交互式应用中具有优势。与此同时，其巨大的数据量也对当前多媒体生态系统的能力提出了挑战。为了降低点云数据的存储和传输对多媒体信号处理系统的压力，点云数据的压缩编码至关重要。

点云数据的获取主要是通过激光扫描技术，由一套成像传感器或一套摄像机来生成一个多维图像阵列并创建点云。随着 3D 采集技术的飞速发展，3D 传感器成本逐步降低并得到更多的推广应用，成熟的产品包括各种类型的 3D 扫描仪、LiDAR 和 RGB-D 相机。由这些传感器获取的三维数据可以提供丰富的几何信息（形状、大小、三维空间位置等）、特征信息（颜色、不透明率、反射率、反照率等）。为此，国际和国内多个标准化组织均已开始三维点云压缩[2] 编码标准的制订工作。2017 年 4 月，国际标准化组织 ISO 下设的 MPEG 运动图像专家组正式开始了三维点云压缩编码 PCC 标准制订，此后 MPEG 一直致力于不断改进点云压缩的性能。在 2020 年批准了两种用于点云压缩的编码标准：V-PCC(基于视频的点云压缩) 和 G-PCC（基于几何的点云压缩）。其中，针对静态点云和动态获取点云的 G-PCC，它是直接对三维空间里的点云利用八叉树或者预测树按照几何信息进行编码，然后再用重建几何信息和原始点云进行重着色，在对重着色的点云进行属性编码。我国的数字音视频编解码技术标准工作组（AVS）设立了点云组，并于 2019 年 7 月、11 月相继发布了点云编码证据征集书和点云编码探索模型软件 PCEM v0.1。

在对几何信息编码的研究中，得益于小米提出的一系列有关 Trisoup 的改进提案[3]，几何信息编码性能有了新的提升。Trisoup 是基于八叉树编码的一种将对象表面表示为一系列三角形的编码方式，这样的处理大大降低了需要传输的点云几何信息，编码端仅需传输选取的用于构建表面三角形的顶点信息，利用这一信息重建出三角面片然后对三角面片进行射线追踪采样后便可重建出原始几何形状。另外，参考八叉树编码采用利用邻居信息来作为上下文，Trisoup 在编码顶点信息时也引入了上下文的概念，这对改善其几何信息编码后进行熵编码时的编码环境，性能都有一定的提升，但由于上下文模型成熟度不高，仍然存在很多冗余条件或者错误的邻居信息。

1.2 研究内容

现有点云压缩将从两方面进行，分别是几何信息压缩和属性信息压缩。其中几何信息的压缩是指利用某些编码方式来编码重建点云的几何坐标，并且在解码端可以恢复出具有三维位置信息的点。而属性信息的压缩针对重建点云中的点附带的一些属性信息，例如颜色、反射率等。目前 MPEG 批准的用于点云压缩的 G-PCC 编码标准中，首先进行几何编码，之后基于重建的几何信息再进行属性编码。本文重点研究几何信息编码。

目前 MPEG 针对 PCC 中的几何编码，采用的是八叉树或者 Trisoup 算法。八叉树编码主要是通过将点云不断进行八叉划分，划分出 8 个节点，并用 8 个比特表示，根据子节点的占据情况，决定 8 个比特的 0，1 分布。接着对于占据的子节点进一步划分，直到划分到最小的节点为止，并最终通过熵编码对产生的码流进行压缩。Octree 几何编码常用于 TMC3，即动态获取点云。Trisoup 几何编码方式常用于 TMC1，即静态获取且表面密集的点云。在 TMC1 几何信息编码中，常常先用八叉树划分到一定层次，然后应用 Trisoup 进行确定顶点和对顶点熵编码以达到压缩的目的，然后通过三角重建近似曲面进行解压缩，也就是说 Trisoup 是一种基于八叉树的几何编码方式。但是在当前几何信息编码标准测试环境中，对

于每一个点云文件在进行编码时八叉树应该划分到哪一层都有固定的值，由参数trisoupNodeSizeLog2 确定。

其中 Trisoup 算法为了使最后的熵编码更加的高效，编码得到的 bit 流更小，熵编码顶点信息过程中使用了动态 OBUF。顶点标识信息与顶点位置信息分开编码，用到了两组不同的上下文信息，同时上下文信息又分为主要信息与次要信息：主要信息用较少的 bit 数承载了绝大部分相关性信息，去除该相关性以后再进行熵编码，码流显然会减少；次要信息的动态使用则是动态 OBUF 的特点所在，它会根据熵编码输出的值更新 LUT 以及上下文使用数量。因此，次要信息中的上下文顺序与熵编码码流的大小有着直接的关联，最佳的上下文顺序可以使熵编码码流达到最小值，有效提高熵编码效率。本文主要分析研究 Trisoup 算法中的熵编码过程，在现有上下文基础上，通过测试次要信息中各个上下文的性能来决定其使用顺序。通过对比更改顺序前后在各个类型测试序列下的性能结果来评判上下文顺序是否得到优化。最终引入自适应的上下文顺序选择算法，为不同类型的点云序列选用最合适的上下文顺序，从而最大程度的减小码流，提高编码效率。

1.3 章节安排

本论文一共分为五个章节，各个章节主要内容如下所述：

第一章：主要介绍了点云编码的研究背景和意义，同时也对本文的研究内容进行了阐述，另外还对本篇论文的整体结构进行了介绍。

第二章：介绍了 G-PCC 编解码整体流程，并对获取点云数据、几何信息编码、属性信息编码、动态 OBUF 以及编码性能的评价标准进行了详细的介绍。

第三章：首先介绍了 Trisoup 算法中编码顶点标识信息和顶点位置信息时用到的上下文信；其次开始介绍各个上下文信息的信息熵测量；然后, 基于求得的信息熵，按照条件熵递增的顺序求得上下文排列顺序。求得局部最优的上下文顺序，用此来提高熵编码效率。

第四章：介绍了 TMC13v20 平台下的测试序列、测试条件以及对应条件下的

测试结果与分析。

第五章：对全文进行了一个总结，分析了本次实验带来的增益与不足，对于不足之处，提出了后续可以进行改进的方向与思路。

第二章 点云压缩

2.1　概述

点云是空间中一组用来表示三维物体、场景的空间结构和表面属性信息的无规则分布的离散点集。其中的每个点都包含几何位置的三维信息，同时也根据应用场景的不同，还会包括不同的属性信息，比如颜色、反射率等。点云的应用十分广泛，如虚拟现实游戏、自动导航系统、地理信息系统、文物数字化保护[4] 等领域。

近年来，随着 3D 扫描技术的迅速发展、采集设备的精度不断提高、图像传感器的改善，使得拥有数以百万计个点的点云模型得以实现，从而为用户带来更加逼真的视觉体验，但与此同时也对数据的存储和传输带来了巨大的压力。在现有的点云压缩编码方法中，国内由 AVS 提出的 PCEM 平台以及国外由 MPEG 提出的 G-PCC 标准中都是将点云几何信息和属性信息分开进行编码，并且属性信息的编码是建立在重建后的几何信息之上。因此，如何有效地对点云的几何与属性信息进行压缩，是目前科研工作的重点，也是目前研究的热点问题。

本章的内容安排如下：2.2 介绍了点云的获取方法，2.3 介绍了目前 MPEG 的 G-PCC 框架，2.4 介绍了几何信息编码方法，2.5 介绍了属性信编码方法，2.6 介绍了点云压缩性能评价标准。

2.2　点云的获取

截止目前，主流的点云获取方法有计算机生成、3D 激光扫描、3D 摄影测量这三种。获取方法的不同，得到的点云数据也不同：计算机生成的点云一般是表示虚拟物体或者场景，被称为静态点云；3D 激光扫描仪可以快速获取被扫描物体或场景的表面坐标、属性数据，为三维物体或场景建立静态点云模型，同时它也能生成动态获取点云，比如安装在无人驾驶汽车上的 LiDAR 扫描仪就是一种先进的 3D 激光扫描仪，它每秒获得的点云可达百万级；3D 摄影测量获取的点云则是

动态的点云模型，这样的点云能像视频一样播放，因此在医学影像领域备受青睐。

点云获取技术的不断更新与发展，使得获得的点云模型精度不断提高，数量也急剧增大。越来越大的行业需求压力，促使着点云压缩技术的发展与标准化工作的展开。其中，MPEG 提出的 G-PCC 标准主要针对静态点云和动态获取点云进行压缩，而 V-PCC 主要针对静态获取的动态点云进行压缩。

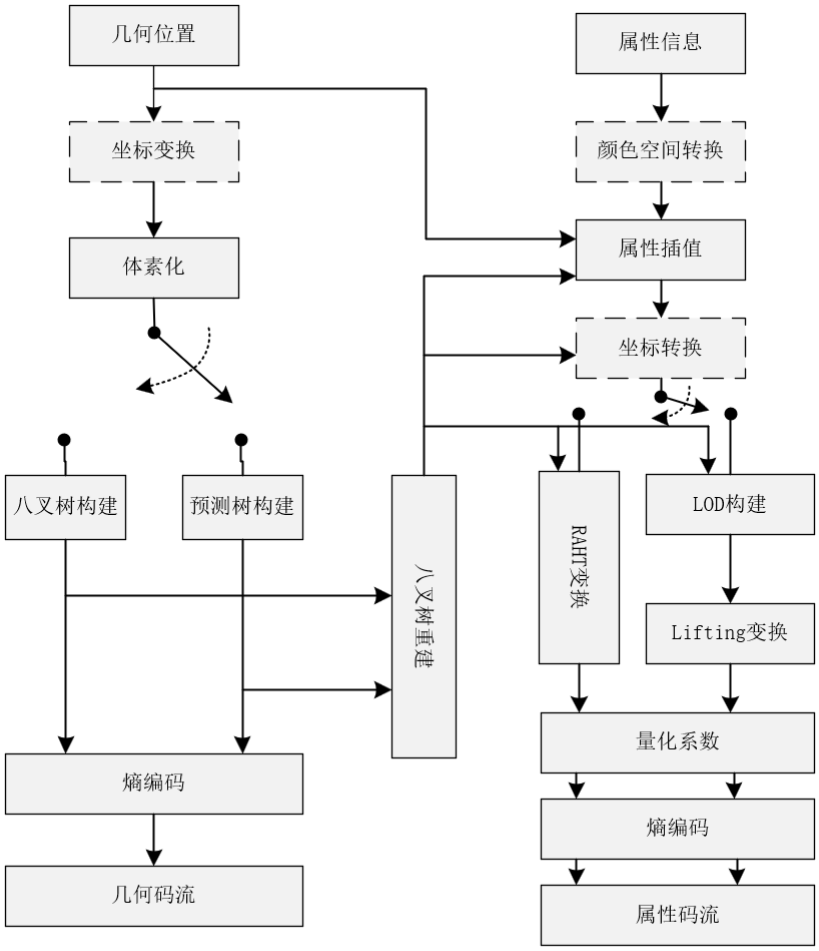


图 2.1　不同种类点云示意图

2.3　GPCC 框架

本文主要对国际动态图像专家组 MPEG 提出的几何点云压缩（G-PCC）展开研究，该组织在为了方便测试和研究，推出了 TMC13(Test Model for Category1 and Catgory3) 模型软件，作为公开探索实验的参考平台。这一平台的具体编解码框架如图2.2所示。

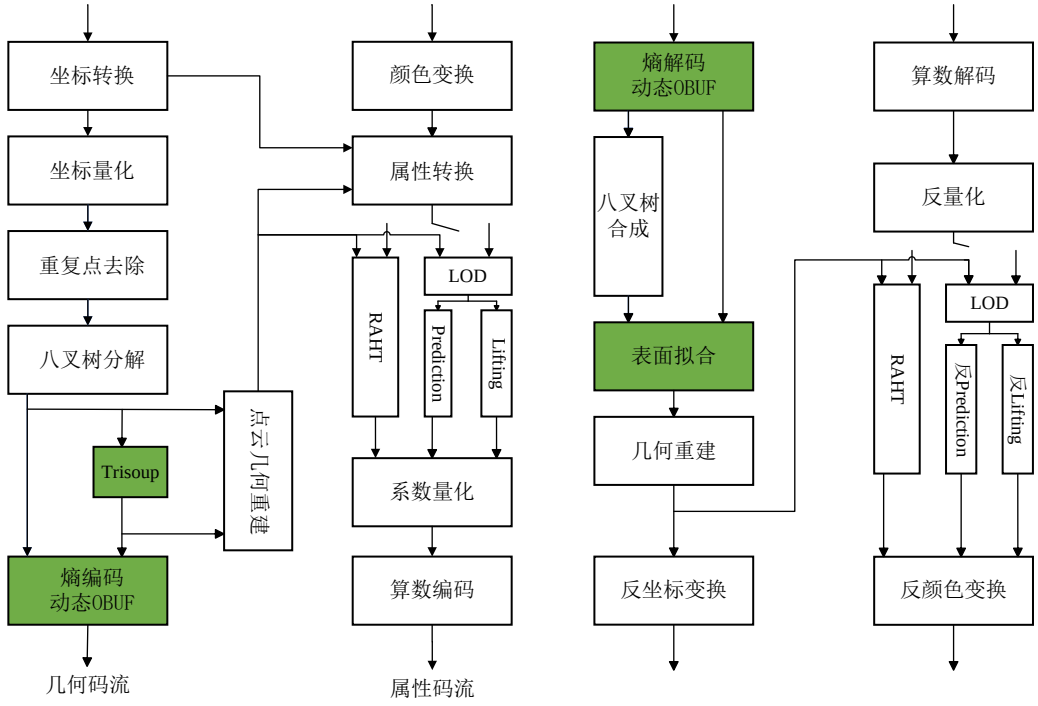


图 2.2　编解码流程图

从图中可以看出，不论是在编码端还是在解码端，几何信息和属性信息的编码都是分开进行的，以编码端为例：

几何信息编码首先对原始点云数据进行预处理，将点云几何信息进行坐标转换，使得变换后的点都在包围（BoundingBox）内再进行量化和重复点去除工作。然后对包围盒进行八叉树（Octree）划分，通过递归细分立方体的方法来构建八叉树结构。立方体每次八等分被划分成八个子立方体，之后再对非空的子立方体进行八等分，直到划分到体素大小为 1×1×1，使用占用码（occupancy code）表示一个立方体是否包含子立方体，1 表示被占据，0 表示不被占据，最后将占位信息送入熵编码器得到几何码流。Octree 几何编码常用于 TMC3，即动态获取点云，而针对表面稠密的静态点云（TMC1），Trisoup 几何编码方式更具优势。Trisoup 是一种基于八叉树的几何编码方式，该方法先用八叉树划分到一定体素大小，然后应用 Trisoup 进行确定顶点和对顶点熵编码以达到压缩的目的，然后通过三角重建近似曲面进行解压缩，后续的属性编码也是基于重建后的点云进行。

属性信息编码首先选择是否对对输入的属性信息进行颜色空间转换，目前 TMC13 平台支持将颜色属性从 GBR 转换成 YUV 空间。当几何有损时，需要对点

云进行属性转换，也就是对几何重建出来的点云重新赋予颜色信息。之后进行属性信息编码，根据不同的测试序列和条件选择适合的编码方式。接着将属性残差经过系数量化处理，最后通过熵编码得到最终的属性码流。

解码端对码流的处理流程即为编码端的逆过程。

另外，在现有的 G-PCC 标准中，使用 Trisoup 几何编码方式时，其熵编码过程用到了动态更新的最佳二值化技术（动态 OBUF），相比于 OBUf，该技术将上下文信息分为了主要信息和次要信息两部分，其中，主要信息用较少的 bir 位包含了上下文信息中最相关的部分，因此它不会变化；相反，次要信息会根据已使用的上下文信息能否将点云有效的区分开来进而来决定是否继续使用下一 bit 位上下文信息。这也是本文的主要研究内容。

2.4 几何信息编码

点云数据中一般存储着几何坐标、颜色、法线等信息。其中几何信息是必不可少的，它在物体的三维重建过程中起着决定性的作用。点云的几何形状通常用特定于应用程序的空间表示，其中点的位置可以用浮点数表示。因此，需要一个预处理步骤，将应用程序特定的空间转换为有限分辨率的内部空间。几何信息预处理的步骤一般包括：坐标转换，坐标量化、重复点去除。其具体过程如下：

2.4.1 坐标转换

获取到的原始点云中，各个点的位置通常是无规则的，并且通常由浮点数表示。因此，为了方便后续对几何信息进行压缩，首先将原始坐标系中点的位置坐标 $\mathrm{X}_{\mathrm{n}}^{orig}$ 用公式2-1转换为内部坐标系下的坐标 $\mathrm{X}_{\mathrm{n}}^{\mathrm{int}}$。

$$X_n = (\mathrm{X}_n^{orig} - T)/s \tag{2-1}$$

其中，$T = (t_x, t_y, t_z)$。参数 T 和 s 由包围盒 $[0, 2^d)^3$ 中的点位置坐标 $\mathrm{X}_{\mathrm{n}}^{\mathrm{int}}$,$n = 1, ..., N$（其中 D 是非负整数参数）决定。

当采用八叉树几何编码时,$1/s$ 由 $positionQuantizationScale$ 参数决定，$T =$

$(\min x_n^{orig}, \min y_n^{orig}, \min z_n^{orig})$，d 由公式2-2确定:

$$d = Ceil(Log_2(Max(x_n^{\text{int}}, y_n^{\text{int}}, z_n^{\text{int}}, n = 1, \ldots, N_{orig}) + 1)) \tag{2-2}$$

$[0, 2^d)^3$ 是最小的包围盒，所有的内部点位置坐标都在这个边长 2^d 的立方体内。

当采用 Trisoup 几何编码时，s 由参数 *trisoupQuantizationBits* 确定，T=[0，0，0]，d 由 *trisoupNodeSizeLog*2 参数确定。

2.4.2 坐标量化

在压缩之前，点位置坐标在内部被表示为 *d* 位非负整数。为了获得这些整数，点的位置坐标在内部坐标系中被量化取整。设 X_n^{int} 表示内部坐标系中的点位置坐标，量化公式如公式2-3所示:

$$\begin{cases} x^k = round\left(\frac{x'}{QS}\right) \\ y^k = round\left(\frac{y'}{QS}\right) \\ z^k = round\left(\frac{z'}{QS}\right) \end{cases} \tag{2-3}$$

其中，*QS* 为量化步长，这里 *round*(*s*) 函数的功能是返回距 *s* 最近的整数，定义如下：

$$round(s) = \begin{cases} int(s + 0.5) & if\, s \geq 0 \\ -int(-s + 0.5) & if\, s < 0 \end{cases} \tag{2-4}$$

2.4.3 重复点去除

量化之后会产生大量位置坐标相同的点，称为重复点。可以将具有相同量化坐标的重复点进行删除操作。坐标量化和删除重复点的过程称为体素化，也就是将一个体素内的所有点都量化到体素中心。如果一个体素内包含任意点云中的点，则称该体素被占用。

2.4.4 八叉树编码

八叉树是一种自顶向下的广度优先的树状数据结构，它可以用来存储表示三维空间物体和场景的几何信息。八叉树中的每个节点都代表着一个对应的正方体。

对于每个非空的节点，可以将其继续划分为八个子节点，直至划分到叶子节点，即指定精度下的三维空间的最小单元。在对点云几何信息进行预处理之后，将对点云几何信息进行几何编码，首先将点云中各点的坐标转换为二进制表示。如果各点的几何位置用三维笛卡尔坐标表示为 $(X,Y,Z),n=0,,N-1$, 其中 N 是点云的总点数，那需要将其 XYZ 坐标数值转化为 K 位比特表示的二进制数值：

$$\begin{cases} X^n = \left(x_{K-1}^n x_{K-2}^n \cdots x_1^n x_0^n\right) \\ Y^n = \left(y_{K-1}^n y_{K-2}^n \cdots y_1^n y_0^n\right) \\ Z^n = \left(z_{K-1}^n z_{K-2}^n \cdots z_1^n z_0^n\right) \end{cases} \tag{2-5}$$

按照莫顿码的生成规则，可以将点云中第 n 个点的三维几何坐标的三个维度上的 K 位比特表示的二进制数值逐位交叉排列，形成一个新的 $3K$ 位比特表示的二进制数，也就是对应的莫顿码，表示如下：

$$M^n = \left(x_{K-1}^n y_{K-1}^n z_{K-1}^n, x_{K-2}^n y_{K-2}^n z_{K-2}^n, \ldots x_1^n y_1^n z_1^n, x_0^n y_0^n z_0^n\right) \tag{2-6}$$

将每三个比特用八进制数表示 $m_n^k = \left(x_n^k x_n^k x_n^k\right), n = 0, 1, \ldots, N-1$, 则 K 点对应的莫顿码可以表示成：

$$M^n = \left(m_{K-1}^n m_{K-2}^n \ldots m_1^n m_0^n\right) \tag{2-7}$$

可以看出，每个点的坐标用莫顿码表示之后形成了 K 个八进制数排列组成的数据。由于八进制数的所有的取值情况可以表示 8 种情况，正好可以与八个子节点一一对应，而且随着 k 值的减小，每个八进制数都代表一层更深的八叉树划分。这样在每一层划分八叉树节点时，就可以很好的判断当前待划分的点要被划分至哪个子节点中。原始点云中的点被划分到不同的节点的实现过程如下:

1. 首先将所有的点根据莫顿码的第 1 个八进制数 m_{K-1}^n 的取值分到八个子节点中：

- 所有 $m_{K-1}^n = 0$ 的点，分配到第 0 个子节点 N_0^1 中;
- 所有 $m_{K-1}^n = 1$ 的点，分配到第 1 个子节点 N_1^1 中;

- ……;
- 所有 $m_{K-1}^{n}=7$ 的点，分配到第 7 个子节点 N_7^1 中;

至此点云中所有的点凭借莫顿码的第一个八进制数都被划分至根节点的八个子节点中，八叉树的第一层划分结束。

2. 用八比特 $B_0^0=(b_0b_1b_2b_3b_4b_5b_6b_7)$ 来表示根节点 N_0^0 的八个子节点是否被占用。如果子节点至少包含点云中的一个点，那么该子节点对应的占位比特 b_k 取 1。如果该子节点不包含任何点，那么占位比特 b_k 取 0。

3. 根据几何位置的莫顿码的第 2 个八进制数 m_{K-2}^{n}，对第一层中被占用的节点进一步进行划分。将每个节点中的点按照其莫顿码第二个八进制数的取值在该节点中选取自己被划分到的子节点，并继续使用八个比特表示节点的八个子节点的占用信息；而不占用的节点也就意味着该子节点不包含点，停止划分。

4. 再根据莫顿码中的第 t 个八进制数 m_{K-t}^{n}，对第 $t=3,\ldots,K-3$ 层中被占用的节点进一步划分成八个子节点；并用八个比特表示将其子节点的占用信息。

5. 对第 $t=K$ 层，所有的节点成为叶子节点，此时点云模型被划分至体素级别，即划分出的叶子节点边长为 1，无法继续划分，八叉树划分到此结束。

由于经过量化后一个节点内的不同点可能会重合成一个点，这时编码端会将重复点个数送入熵编码器，解码端通过算数解码得到。因此解码端在重建八叉树时，能够无损恢复每个节点中真实点的个数。

2.4.5 Trisoup 几何编码

Trisoup 几何编码方式常用于 TMC1，即静态获取且表面密集的点云。在几何信息编码中，常常先用八叉树划分到一定层次，然后应用 Trisoup 进行确定顶点和对顶点熵编码以达到压缩的目的，然后通过三角重建近似曲面进行解压缩，也就是说 Trisoup 是一种基于八叉树的几何编码方式。但是在当前几何信息编码标准测试环境中，对于每一个点云文件在进行编码时八叉树应该划分到哪一层都有固定的值，由参数 *trisoupNodeSizeLog*2 确定。

Trisoup 几何编码主要分四个步骤进行：

1、确定顶点

几何图形在每个块中表示为一个至多与块的每条边相交一次的曲面。由于一个块有 12 条边，所以在一个块中最多可以有 12 个这样的交点。每一个这样的交点称为顶点。当且仅当共享该边的所有块中与该边相邻的块中至少有一个已占用的体素时，才能检测到沿该边的顶点。检测到的顶点沿边的位置是共享该边的所有块中，所有与该边相邻的体素沿边的顶点的平均位置。

2、熵编码顶点信息

顶点信息包含两部分：一个是标识信息，即标识所有被占据的叶子节点中哪些边上有顶点，也称为边被占据，被占据标识为 1，不被占据标识为 0；另一个是顶点位置信息，对于每个被占据的边，顶点位置信息被量化为 2bit。最后利用算数编码器对标识信息与顶点位置信息进行熵编码。

3、三角面片构建

根据三角形投影面积大小确定主导轴、求出质心坐标以及质心偏移量、根据方位角信息对顶点进行排序。最后将顶点与质心组合，构建三角面片。如图2.3所示。

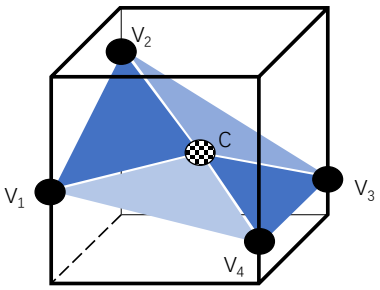


图 2.3　三角面片构建示意图

4、射线追踪采样

以三角面片在主轴方向上的投影面为射线起点，射线追踪的间隔由高层语法元素决定，同时会根据重建后的点云数量改变间隔大小。射线追踪采样的核心是利用 Muller-Trumbore 算法，判定射线是否与三角面片有交点，产生的交点便是重建点云。如图2.4所示。

为了使熵编码更加的高效，编码得到的 bit 流更小，熵编码顶点信息过程中使用了动态 OBUF。顶点标识信息与顶点位置信息分开编码，用到了两组不同的上

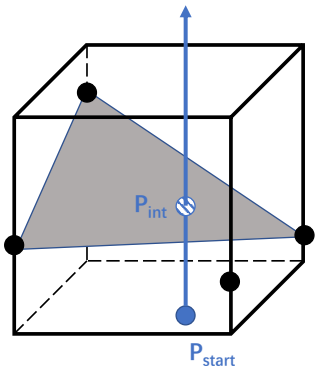


图 2.4　射线追踪采样示意图

下文信息，同时上下文信息又分为主要信息与次要信息：主要信息用较少的 bit 数承载了绝大部分相关性信息，去除该相关性以后再进行熵编码，码流显然会减少；次要信息的动态使用则是动态 OBUF 的特点所在，它会根据熵编码输出的值更新 LUT 以及上下文使用数量。

2.4.6 动态 OBUF

在 MPEG 给出的 TMC13v20 参考代码中，采用 Trisoup 几何编码方式时，最后将顶点的标识信息、顶点的位置信息进行熵编码时，采用的是基于动态更新的最佳二值化的熵编码器（动态 OBUF），其编码框架如图2.5所示：

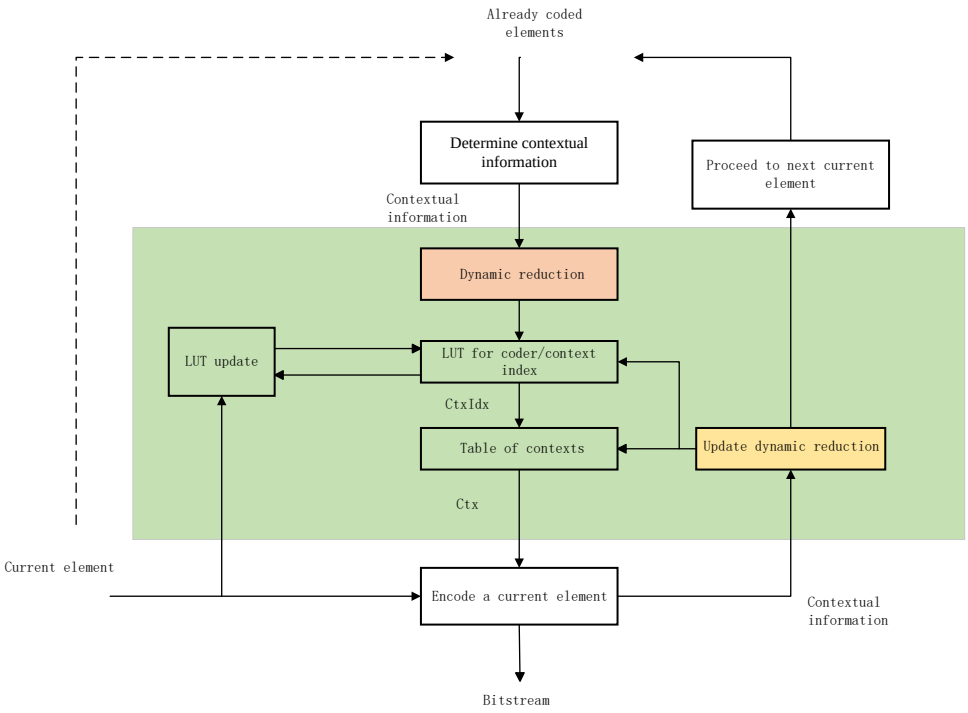


图 2.5　动态 OBUF 流程图

动态 OBUF 相比于 OBUF 不同之处在于它将上下文信息分为了主要信息（Primary Information）和次要信息（Second Information），其中主要信息以较少的 bit 数承载着强相关的上下文信息，主要信息是不会随着熵编码的不断进行而减少的；次要信息的使用是动态 OBUF 的特点所在，图中标识的部分具体细节如图2.6与2.7所示：

动态减少是针对次要信息进行的，整个动态减少的过程主要由计数器 N（i1，i2’）和决定次要信息使用 bit 位数的 K 两个参数控制，其中 N（i1，i2’）是记录整个编码过程访问当前上下文的次数，当计数器超过一定的阈值时，如图2.7所示，首先 DRn 将会更新为 DRn+1，这同样与熵编码器的索引表（LUT）相适应，最后还需要将计数器置零。另外，次要信息可以表示为 i2=。。。。，通过更新 K 值，再对次要信息进行移位操作，便可达到动态更新次要信息的效果。

这样设计的目的在于，在编码初期，熵编码只会使用到很少的次要信息甚至仅使用主要信息就可以高效率的将待编码语法元素编码完成，因此大大节省了熵编码的 bit 流，提高了编码效率。

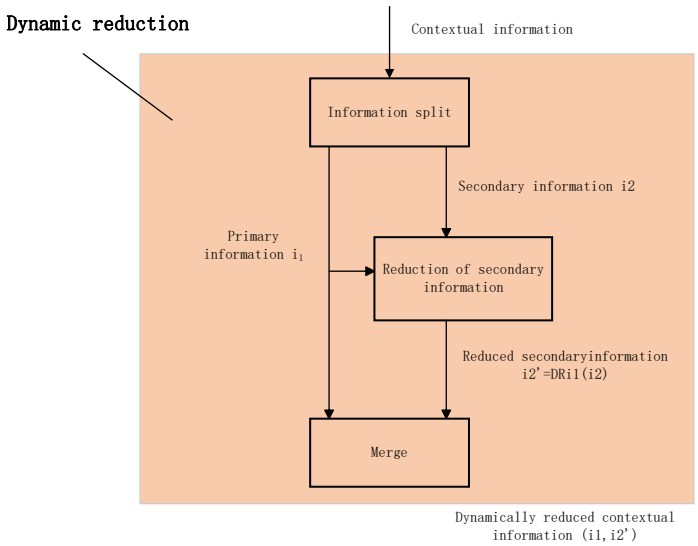


图 2.6　dynamicreduction 示意图

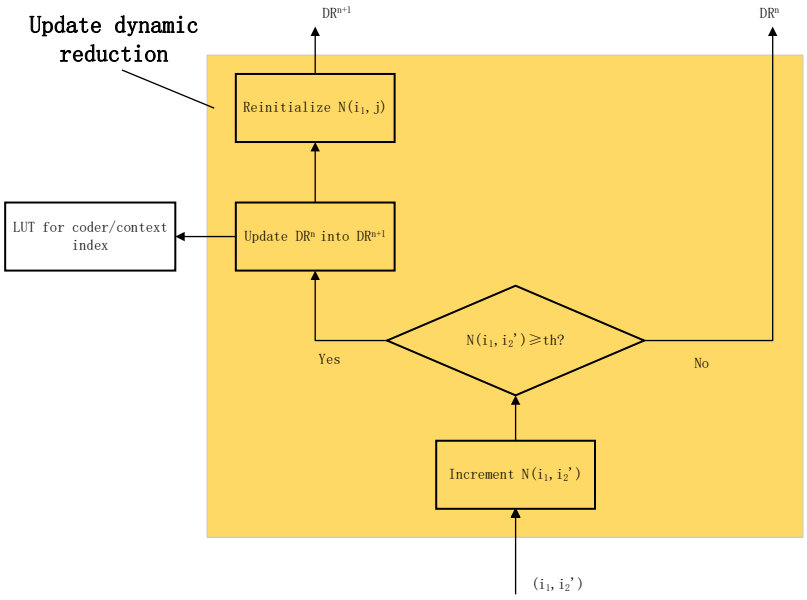


图 2.7　updatedynamicreduction 示意图

2.5　属性信息编码

由图2.2可以直观的看出，G-PCC 主要包括两种属性变换编码方法，第一种是基于插值的带更新/提升的分层最近邻预测（Lifting Transform），第二种是区域自适应分层变换（RAHT）。其中，与 Trisoup 几何编码方法配合使用的主要是第二种。另外，这两种变换编码方式通常用于第一类点云数据。

2.5.1 颜色变换

G-PCC 提供了颜色空间转换的功能，方便在不同场景下，应用合适的颜色空间。例如从 RGB 到 YUV 的颜色空间转换，具体公式如下所示：

$$\begin{cases} Y = 0.2126 \times R + 0.7152 \times G + 0.0722 \times B \\ U = -0.114572 \times R - 0.385428 \times G + 0.5 \times B + 128 \\ V = 0.5 \times R - 0.454153 \times G - 0.045847 \times B + 128 \end{cases} \tag{2-8}$$

转换的逆过程有：

$$\begin{cases} R = Y + 1.5748 \times (V - 128) \\ G = Y - 0.18733 \times (U - 128) - 0.46813 \times (V - 128) \\ B = Y + 1.85563 \times (U - 128) \end{cases} \tag{2-9}$$

其中，在整个颜色转换的过程中，使用的所有数据都是浮点型的，但是标准定义的输出结果统一转换为 $unit8_t$ 格式，因此经过颜色空间转换后，颜色信息会产生一定的偏差，这也就限定了颜色空间转换只适应于属性有损条件下进行。

2.5.2 属性转换

进行属性信息编码前，因为当几何编码是有损时，重建出来的点云坐标位置与原始几何坐标有偏差，所以重建的点云没有属性信息，这个时候需要将原始的属性信息进行属性转换，为点云重新着色，同时需要保证属性失真最小。目前，在 TMC13 中，可以通过计算原始点云值多个近邻的距离加权平均数来实现属性转换。具体的属性转换过程如下：

1、为重建点云 $(\widehat{X}_i)_{i=0\cdots N_{rec}-1}$ 建立起 KD-Tree，将三个维度中的最长边方向作为划分主轴，其次基于划分主轴方向对重建点云进行划分。然后遍历原始点云的每个点，在重建点云 KD-Tree 结构中进行最近邻居点 $Ref1$ 查找。由于是遍历的原始点云，因此最近邻居点会出现两种情况，一种是在重建点云中某个点作为原始点云中多个点的最近邻居点；第二种是原始点云中所有点都不会将某个重建点作为最近邻居点。

2、对原始点云 $(X_i)_{i=0\cdots N-1}$ 建立起 KD-Tree，将三个维度中的最长边方向作为划分主轴，其次基于划分主轴方向对重建点云进行划分。然后遍历重建点云中的每个点，在原始点云 KD-Tree 结构中进行最近邻居点 $Ref2$ 查找。但是由于是遍历重建点云的每一个点，每重建点都会找到一个原始点云作为其最近邻居。

3、首先判断 $Ref1$ 集合是否为空，如果为空，则当前重建点的属性信息由相应索引的 $Ref2$ 得到，否则当前重建点的属性由公式可得：

$$\hat{A}_i = \frac{1}{num}\sum_{k=1}^{Ref(i)} A_k(i) \tag{2-10}$$

2.5.3 RAHT 变换编码

在八叉树划分的结构下，相邻节点里的属性值可能会比较相近，即这些属性信号的交流分量几乎为 0，能量主要集中在直流分量上，通过变换将原始属性值

转换成对应的 AC，DC 系数，就可以更好的对信号进行压缩去冗余。在 GPCC 中 RATH 变换主要过程如下：

1. 构建和解析树

开始构建树时，需要对点云中所有的点按照莫顿序逐次进行扫描，通过莫顿码右移，不断将点云序列中的点进行合并，从而逐层构建树。在构建树的过程中，每个点的初始权重 $weight_{org}$ 设置为 1，初始颜色属性为 $Attr_{org}$, 每当两个点进行合并时，合并后点的属性值和权重分别为:

$$Attr_s = Attr_{org1} + Attr_{org2} \tag{2-11}$$

$$weight_s = weight_{org1} + wieght_{org2} \tag{2-12}$$

不断进行莫顿码右移和迭代，直至整个点云序列中只有一个点。解析树的过程是构建树的相反过程，基本规则相同。

2. 预测和变换

由于最高级父块处于树的顶部位置，并没有可以用于参考从而进行预测的父块或者祖父块，因此第 n-1 到 n-4 层不进行预测，当预测标识开启时，除第一级父块外的其余层都会进行预测。通过 occupancy 标记当前父节点中 8 个子节点的占用情况。通过父级块对当前块进行预测，首先要寻找当前块的父块的 19（1+6+12）个邻居节点，其中，1 为邻居节点是当前块的父节点，6 为与父节点共面的邻居节点，12 为与父节点共线的邻居节点，但这 19 个邻居节点不一定总是被占据，所以需要对邻居节点进行标识, 用“1”表示当前子节点被占据，用 0 表示当前子节点不被占据，邻居节点示意图如 2.8 所示：

查找完邻居节点后，就需要对子节点进行预测。不同类型的邻居节点，在预测中所占的权重是不一样的。首先是父节点对子节点预测效果最好，其次是共面的父邻居节点，再者是共线的父邻居节点。因此在设置预测权重时，权重值依次递减。

对属性信息和预测的属性信息进行 RAHT 变换，根据构造的树的结构，从最

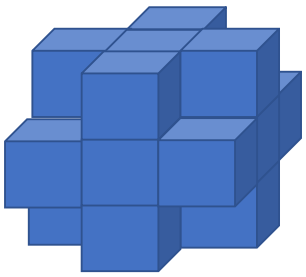


图 2.8　邻居节点的示意图

高层开始，每三层进行 RAHT 变换，其中每个父块作为变换块来依次进行 RAHT 变换。如果进行在该层进行预测，则需要对属性信息和预测属性信息都进行 RAHT 变换；如果不进行预测，则只需要对属性信息进行 RAHT 变换。用 w_1 和 w_2 分别表示两个待变换块包含的点数，用 c_1 和 c_2 表示两个待变换块的属性值，则 RATH 变换公式如2-13和2-14所示：

$$\mathrm{RAHT}\left(w_1, w_2\right)=\frac{1}{\sqrt{w_1+w_2}}\begin{bmatrix}\sqrt{w_1} & \sqrt{w_2}\\ -\sqrt{w_2} & \sqrt{w_1}\end{bmatrix} \tag{2-13}$$

$$\begin{bmatrix}DC\\ AC\end{bmatrix}=\mathrm{RAHT}\left(w_1, w_2\right)\begin{bmatrix}c_1\\ c_2\end{bmatrix} \tag{2-14}$$

其中 DC 系数为属性的加权平均值，AC 系数为相邻两点的属性残差。计算 AC 系数的残差，计算公式如2-15所示：

$$Attr_{res}=AttrCoff_{org}-AttrCoff_{pred} \tag{2-15}$$

若当前层不进行预测，则直接对属性信息变换系数进行量化和熵编码。若当前层需要进行预测时，对上述步骤得到的 AC 系数的残差进行量化和熵编码。

2.5.4 Lifting 预测变换编码

该方法利用点云中已编码点的属性值对待编码点进行预测，从而得到待编码点的属性预测值，并将待编码点的真实属性值与预测值相减得到预测残差，以达到去除空域冗余的目的。基于莫顿码排序选取预测点，一定程度上反映了空间近

邻的相关性，但是由于存在空间跳跃点等情况，预测效果并不是很好。因此引入了 LOD (Level Of Detail) 划分技术，也就是将点云划分成不同的层次结构，利用层次结构对属性进行有效预测。如图 2.9 所示：

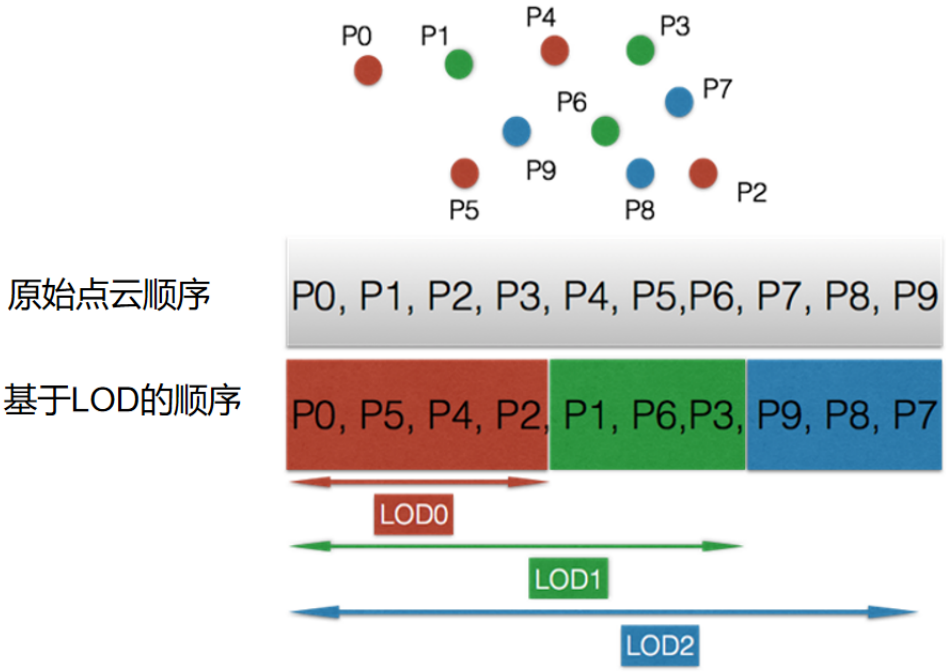


图 2.9　基于距离的 LOD 划分示意图

LOD 划分有三种方法，包括基于距离的 LOD 划分、基于采样率的 LOD 划和基于八叉树的 LOD 划分。基于距离的 LOD 划分适用于稠密均匀的点云，如 cat1 类型；基于采样率的 LOD 划分适用于分布稀疏不均匀的点云，如 cat3-frame 和 cat3-fused 类型；基于八叉树的 LOD 划分适用于 scalable 情况，对齐几何和属性的点数，使其与几何八叉树结构对齐。

Lifting 变换编码是一种依赖于多层 LOD 的属性编码方式，也就是将点云划分成不同的层次结构，利用层次结构对属性进行有效预测。其主要包含了分割、预测、更新三个部分，如图2.10

其中，$H(N)$ 表示高层次点云，$L(N)$ 表示低层次点云,$D(N)$ 表示预测残差。$L(N)$ 是与 $LODN$ 相关的属性集，$H(N)$ 是与细化层相关的属性集。

Lifting 变换建立在分层预测的基础上，引入了权重更新和自适应量化的概念。低层 LOD 层被频繁的用来预测高层 LOD 层中的点，所以更新量化权重的目的是给低层 LOD 层点云提升一定的权重值来表征其重要性，使其影响更大，量化损失

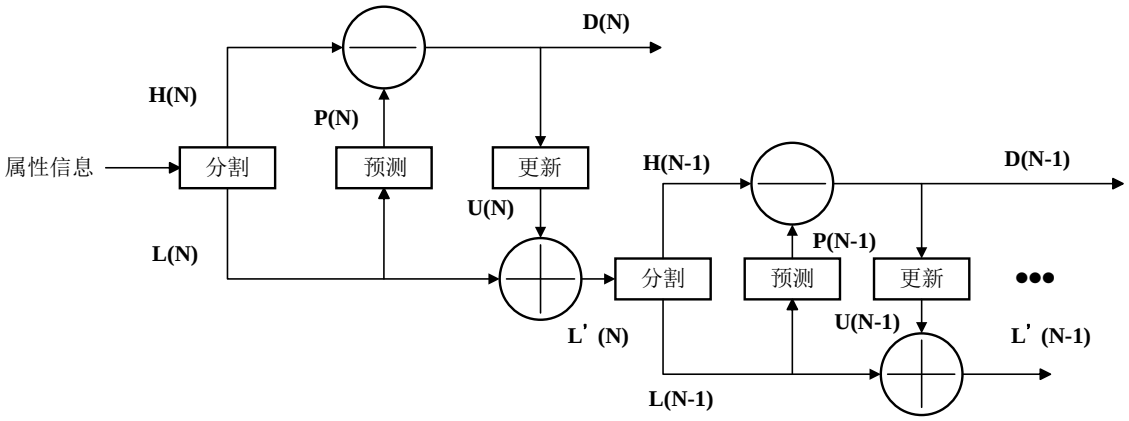


图 2.10　LOD 划分示意图

更小，预测结果更准确。Lifting 变换的关键在于预测和更新算法。预测算法得到的预测残差是变换过程中的高频分量，而通过更新算子来提升的属性信息是输入信号的低频分量。低频分量表征了点云属性的基本特征，反映出点云属性的整体情况，高频分量表征了点云属性的更多细节特征。Lifting 变换运算时间短，占用存储少，可准确地将点云属性划分为低频和高频两个部分，具有良好的压缩性能和重建效果

2.6　评价标准

评价一个点云压缩算法的性能好坏一般从三个方面出发：

1、编解码前后点云失真情况，具体又细分为几何失真与属性失真。两部分的失真计算都是基于对应点的，几何失真计算的是几何距离的失真，属性失真计算的是属性数值的失真。其中对应点就是原始点云的点在重建点云中的最近邻居点或者重建点云中的点在原始点云中的最近邻居点。

2、编码后比特流的大小。比特流大小可以通过 Bpip（bits per input point，即平均每个输入点所用比特数）来衡量，Bpip 越小，压缩性能越好。

3、编解码点云的时间复杂度。时间复杂度可以通过编解码过程耗费的时间来衡量，时间越短，压缩性能越好。

在无损压缩情况下，不存在点云的失真，只要考后两个方面来衡量性能。在有损压缩的情况下，点云的失真程度也是重要的考量，在衡量性能时三个方面都要

考虑。在实际评估过程中，可从 rate 模式和 PSNR 模式两方面进行比较。BD-rate 为负值时，表示相同 PSNR 的条件下，码率减小，性能提升；BD-PSNR 为负值时，表示相同码率的条件下，PSNR 增大，性能提升。

第三章 Trisoup 顶点标识与位置信息的熵编码优化

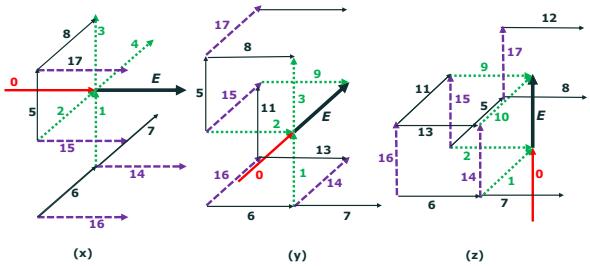
Trisoup 几何信息编码算法的基本思想是将编码八叉树划分到一定层次后的叶子节点中数量较多的原始点云信息压缩成了编码该叶子节点边上的顶点信息，其中顶点信息包括叶子节点的每条边是否被占据以及被占据边上顶点的位置信息，解码端在获取到编码生成的 bit 流后，按照标准进行解码操作，然后进行点云重建。当然，在实际编码过程中还对很多其他辅助信进行了编码，比如：质心坐标值、质心偏移值以及上层参数值。这样的操作虽然大大减少了需要编码的信息量，但是由于点云空间位置存在一定相关性，普通算数编码会编码很多冗余信息。因此 Trisoup 对顶点信的息编码采用了基于本文2.4.6所述的动态 OBUF 的熵编码方法，一定程度上消除了点云空间位置上的相关性，进一步压缩了码流。

然而，现有 MPEG 提出的编解码标准中为顶点标识信息与顶点位置信息构建的上下文未必是最佳。为此，本章通过对各个上下文信息熵与条件熵的测量，依照熵值越小，上下文信息越有效的原则，对现有上下文顺序进行了一定调整。

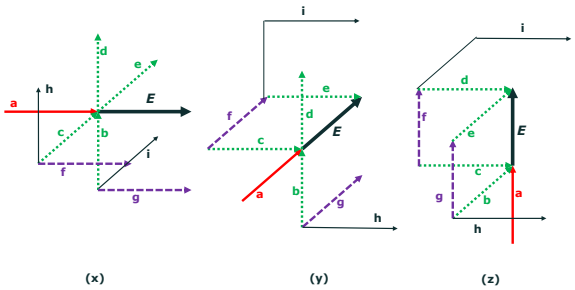
3.1　Trisoup 几何信息熵编码的上下文构造

在 TMC13v20 版本的参考代码中，MPEG 采纳了由 Sébastien 提出的拓展 Trisoup 中上下文所用到的邻居数量的提案。截至目前，Trisoup 在编解码顶点信息时将当前待编码边按照朝向的不同分为三种情况，分别为：平行于 x 轴、平行于 y 轴、平行于 z 轴。每条待编码边拥有各自的 18 条邻居边，但实际用于构建上下文的邻居边是按照不同朝向从 18 条边中选出的 9 条边，如图3.1所示。另外，由于上下文构建的需要，如果某一邻居边被占据，边上顶点位置在被量化为 2bit 的基础上再次细分出一个 *closest* 区间和 *close* 区间, 如图3.2所示。用于组成上下文的信息除了选取的 9 条边占据情况外，还包括当前待编码边周围节点的占据情况，如图3.3所示。

利用上述这些邻居边信息和邻居节点信息分别为编码顶点标识信息、顶点位



(a) 18 条邻居边



(b) 9 条邻居边

图 3.1　邻居边示意图

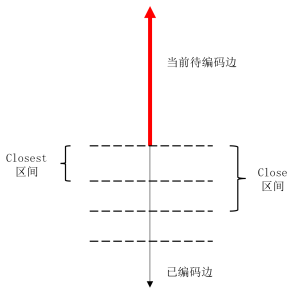


图 3.2　边的区间划分示意图

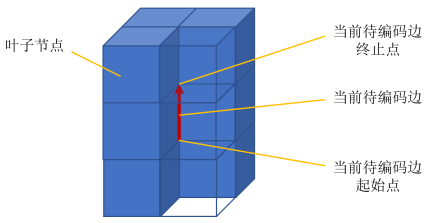


图 3.3　当前待编码边周围节点示意图

置信息的高 bit 位以及顶点位置的低 bit 位构建了三套不同的上下文。其中强相关的信息作为上下文的主要信息，次相关的信息作为上下文的次要信息。

3.1.1 顶点标识信息上下文结构

顶点标识信息简而言之就是用 1bit 位来标识当前待编码边是否被占据，如果被占据则该 bit 置“1”，如果不被占据则该 bit 置“0”。其上下文结构如下：

1、主要信息组成

```
// encode flag vertex
int ctxMap1 = std::min(nclosestPattern, 2) * 15 * 2 +  (neighbEdge-1) * 2 + ((ctx1 == 4));
```

图 3.4　组成标识信息上下文的主要信息代码截图

其中：

nclosestPattern : 表示 9 条边中的前 5 条边在 closest 区间的数量。

neighbEdge : 表示包含当前待编码边的 4 个节点的占据情况。

*ctx*1 : 表示包含当前待编码边起始点坐标的 4 个节点被占据的数量。

2、次要信息组成

```
int ctxMap2 = neighbEnd << 11;
ctxMap2 |= (patternClose & (0b00000110)) << 9 - 1 ; // perp that do not depend on direction = to start
ctxMap2 |= direction << 7;
ctxMap2 |= (patternClose & (0b00011000))<< 5-3; // perp that  depend on direction = to start or to end
ctxMap2 |= (patternClose & (0b00000001))<< 4;  // before
int orderedPclosePar = (((pattern >> 5) & 3) << 2) + (!!(pattern & 128) << 1) + !!(pattern & 256);
ctxMap2 |= orderedPclosePar;
```

图 3.5　组成标识信息上下文的次要信息代码截图

其中：

neighbEnd：表示包含当前待编码边终止点的 4 个节点的占据情况。

patternClose：9 条边在 close 区间被占据的情况。

direction：当前待编码边的朝向 $(0 = X, 1 = Y, 2 = Z)$。

pattern：9 条边的被占据的情况。

另外，如表3.1所示，本文按照次要信息的现有上下文顺序对其进行编号。

3.1.2 顶点位置信息上下文结构

在编码叶子节点中某一条被占据边上的顶点坐标时，首先需要对顶点坐标进行量化操作，现有的量化操作是将顶点坐标量化为 0,1,2,3 四个坐标值，因此需要

表 3.1 标识信息上下文编号

上下文名称	编号
missedCloseStart	ctx1
patternClosest & 1	ctx2
direction	ctx3
patternClose & (0b00011111)	ctx4
missedCloseStart	ctx5
missedCloseStart	ctx6

用 2bit 来表示量化后的顶点位置信息。由于两个 bit 位所承载的位置信息在物理含义上不一致，对于高 bit 位，它将量化后的顶点坐标可选范围缩减一半，顶点坐标取值要么为 0，1 要么为 2，3；低 bit 位则是在高 bit 位基础上，完全确定顶点坐标位置。因此分别为其构建不同的上下文模型。

对于高 bit 位：

1、主要信息组成

```
int ctxFullNbounds =(4 * (ctx0 <= 1 ? 0 : (ctx0 >= 3 ? 2 : 1)) + (std::max(1, ctx1) - 1))* 2+ (ctxE == 3);
int b = nbitsVertices - 1;

// first bit
ctxMap1 = ctxFullNbounds * 2 + (nclosestStart > 0);
```

图 3.6 组成位置信息上下文高 bit 的主要信息代码截图

其中：

*ctx*0: 表示包含当前待编码边终止点坐标的 4 个节点被占据的数量。

ctxE: 表示当前待编码边周围 4 各个节点被占据数量

nclosestStart: 表示 9 条边中的前 5 条边在 closest 区间被占据的数量

2、次要信息组成

```
ctxMap2 = missedCloseStart << 8;
ctxMap2 |= (patternClosest & 1) << 7;
ctxMap2 |= direction << 5;
ctxMap2 |= patternClose & (0b00011111);
```

图 3.7 组成位置信息上下文高 bit 的次要信息代码截图

其中：

missedCloseStart: 表示 4 条相邻垂直边不被占据的数量

patternClosest：表示 9 条边在 closest 区间被占据的情况

对于低 bit 位，其上下文模型的主要信息部分与高 bit 位一致，主要差别体现在次要信息更加丰富。下面给出该 bit 位上下文模型的次要信息组成：

```
ctxMap2 = missedCloseStart << 8;
ctxMap2 |= (patternClose & 1) << 7;
ctxMap2 |= (patternClosest & 1) << 6;
ctxMap2 |= direction << 4;
ctxMap2 |= (patternClose & (0b00011111)) >> 1;
ctxMap2 = (ctxMap2 << 4) + orderedPclosePar;
```

图 3.8　组成位置信息上下文低 bit 的次要信息代码截图

其中：

orderedPclosePar：表示不相邻的垂直边与平行边的占据情况

同样的对次要信息上下文按照其现有使用顺序进行编号，如表3.2所示。

表 3.2　位置信息上下文编号

位置信息	上下文名称	编号
高 bit	missedCloseStart	ctx1
	patternClosest & 1	ctx2
	direction	ctx3
	patternClose & (0b00011111)	ctx4
低 bit	missedCloseStart	ctx1
	patternClose & 1	ctx2
	patternClosest & 1	ctx3
	direction	ctx4
	patternClose & (0b00011111)	ctx5
	orderedPclosePar	ctx6

对于要编码的顶点标识信息与位置信息，有了这一系列的上下文模型后，采用动态 OBUF 技术进行熵编码操作，就可以达到去除点云空间坐标之间冗余信息的目的，从而有效降低了的码流大小，提高了编码效率。

3.2　上下文信息熵测量

虽然利用基于动态 OBUf 的熵编码技术能够有效的降低码流，但是其是否动态更新次要信息是通过判断当前已使用的上下文模型被选中的次数是否大于一定阈值来控制的，因此本文希望通过计算各个上下文的信息熵大小，来决定整套上下文模型的第一个上下文。其中依据的原理是，若某一上下文熵值越小，则说明它与当前待编码边的相关性越强，因此如果将其放在越高 bit 位使用，随着熵编码

过程的进行，所选择的熵编码器概率模型会越来越符合当前待编码值的实际概率分布，整个熵编码过程更加高效。基于这一理论，本节尝试对三组次要信息所有用到的所有上下文进行信息熵的测量与对比，从而选出最佳的第一上下文。

本文探究如何生成求解信息熵的可执行文件是在*basketball_player_vox*11_00000200 序列的 *r*01 码率点上进行的，首先在 TMC13v20 的参考代码中加入如图3.9所示的无序键值对，用于存储各上下文信息不同值与当前待编码 bit 不同值之间的匹配数量。

```
//yk add
std::unordered_map<int, std::unordered_map<int, int>> neighbEndCtx;
std::unordered_map<int, std::unordered_map<int, int>> patternCloseCtx;
std::unordered_map<int, std::unordered_map<int, int>> directionCtx;
std::unordered_map<int, std::unordered_map<int, int>> orderedPcloseParCtx;
std::unordered_map<int, std::unordered_map<int, int>> missedCloseStartCtx;
std::unordered_map<int, std::unordered_map<int, int>> patternClosestCtx;
```

图 3.9　无序键值对代码截图

例如计算标识信息时用到的键值对：$std :: unordered_map < int, std :: unordered_map < int, int >> directionCtx$ 存储了 *direction* 的 3 个不同取值 0,1,2 与当前待编码边是否被占据的 0,1 信息匹配的所有情况各自的次数，利用该键值对存储的数据，可以得到 $direction = 0$ 时，当前待编码边占据时的占比 *rate*1 与不占据时的占比 *rate*0，同时也可以得到 $direction = 0$ 在该类上下文中的占比 *countRate*。基于这些数据，利用如下信息熵求解公式3-1：

$$entropy = rate0 * \log \frac{1}{rate0} + rate1 * \log \frac{1}{rate1} \tag{3-1}$$

可以得出该类上下文在 direction=0 时，它的信息熵如公式3-2所示。

$$entropy0 = entropy * countRate \tag{3-2}$$

同理，$direction = 1$ 和 $direction = 2$ 时，可以求出其对应的信息熵 *entropy*1，*entropy*2. 由此便得到了 direction 这类上下文的信息熵 $entropy = entropy0 + entropy1 + entropy2$。重复这一计算过程，可以求得编码顶点标识信息时用到的各类上下文的信息熵。对于其他序列其他码率点的测试，本文采取的方法是：

1、将上述计算方法写入 TMC13v20 的参考代码，并且在参考代码中加入打印每个 slice 求得的信息熵的代码后生成可执行文件。

2、撰写 python 脚本文件将可执行文件用于通测最终测试性能所需的所有点云序列的所有码率点，并将得到的数据结构用 txt 文本输出。

3、撰写 python 脚本对所有生成的 txt 文本进行遍历，获取其中的信息熵并按不同序列的不同码率点输出到 excel 表格中。其中对于同一码率点的不同 slice，将不同 slice 得到的信息熵进行简单平均作为该码率点的信息熵。

4、重复上述步骤，得到所有测试点云序列的全部码率点下的各类上下文信息熵。

5、最后进行折线图绘制，并分析结果。

以标识信息为例，从四类测试点云序列 (solid,dense,sparse,scant) 中各取一个测试序列作图，得到其次要信息所用到的上下文的信息熵对比图如下所示：

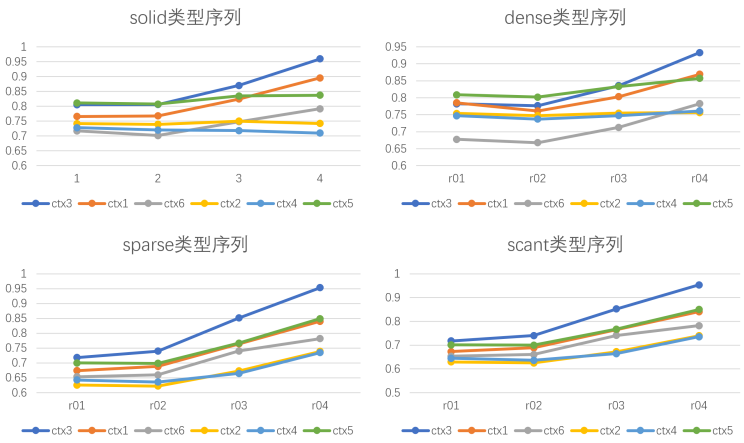
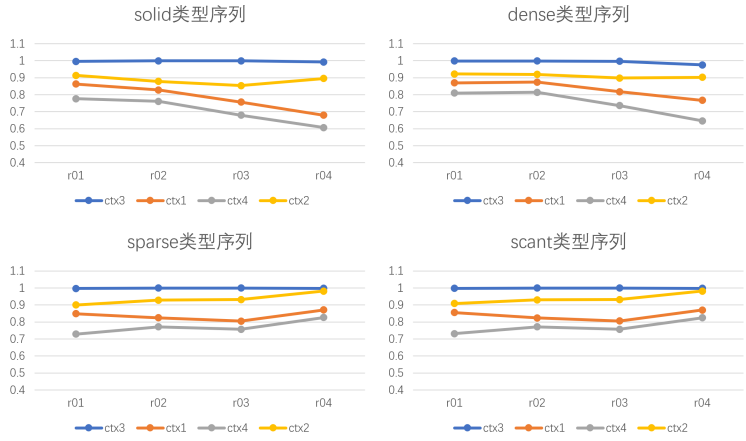


图 3.10　标识信息的信息熵对比图

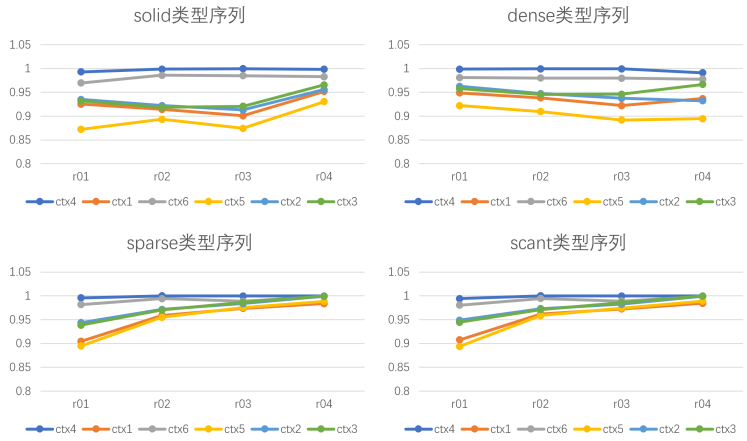
图中我们可以看出在各类点云序列下，ctx4 这一上下文基本处于最低熵值的位置，故选取 ctx4，即 patternClose&(0b00011000) 作为编码顶点标识信息所用到的次要信息的第一个上下文。

同理，对编码顶点位置信息用到的次要信息所包含的上下文信息熵进行测试，可得到如图3.11所示结果。

据此得出高 bit 位应选取 ctx4，即 patternClose&(0b00011111) 作为其第一类上



(a) 高 bit 位



(b) 低 bit 位

图 3.11　位置信息的信息熵对比图

下文；低 bit 位应选取 ctx5, 即 patternClose& (0b00011111) 作为其第一类上下文。

3.3　上下文条件熵测量

基于熵越小，上下文与当前待编码边相关性越强的理论基础，本文在三套上下文的第一类上下文已经确定的基础上对剩余上下文的条件熵进行测试，进而得到三套上下文模型的局部最优顺序。本文条件熵的测试将在 *basketball_player_vox*11_00000200 序列的 *r*01 码率点下进行。

以标识信息的上下文条件熵测量为例。依据信息熵的测量结果将 *ctx*4 作为首要条件，在此条件下对剩余上下文进行条件熵的测试。实际操作过程中，先将 *ctx*4 分别与剩余五类上下文进行组合构成五类新的上下文，如图3.12所示。

从而将对条件熵的测量转换为了对这新的五类熵下文进行信息熵的测量，目的是可以继续使用3.2节所介绍的方法。具体实验过程如下：

```
//yk add
// 求条件熵
 int ctxl_2=neighbEnd*4+(patternClose & (0b00000110)>>1);
 int ctxl_3=neighbEnd*4+direction;
 int ctxl_4=neighbEnd*4+(patternClose & (0b00011000)>>3);
 int ctxl_5=neighbEnd*2+(patternClose & (0b00000001));
 int ctxl_6=neighbEnd*16+orderedPclosePar;
```

图 3.12　组合测试所需上下文部分代码截图

1、以 ctx4 作为条件时，测试得到剩余五类上下文的条件熵如图3.13所示。

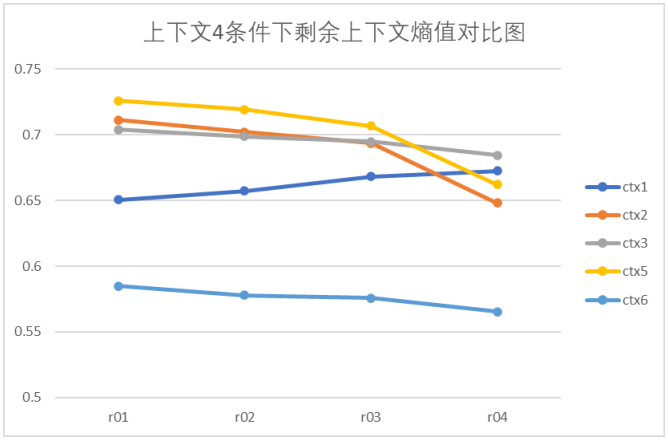


图 3.13　条件熵对比图 1

可以看出，ctx6 条件熵明显小于其他四类上下文条件熵，因此选择 ctx6 作为第二个条件。

2、以 ctx4、ctx6 作为条件时，测试得到剩余四类上下文的条件熵如图3.14所示。

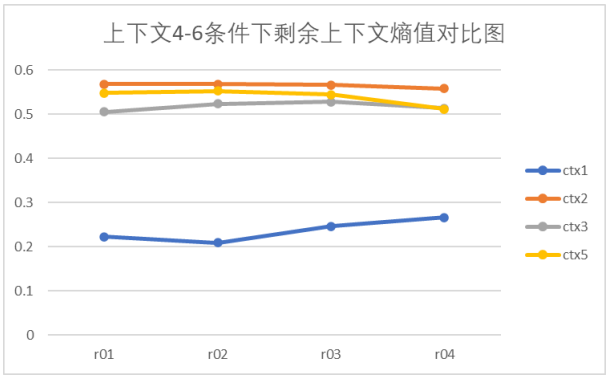


图 3.14　条件熵对比图 2

可以看出，ctx1 条件熵明显小于其他三类上下文条件熵，因此选择 ctx1 作为第三个条件。

3、以 ctx4、ctx6、ctx1 作为条件时，测试得到剩余三类上下文的条件熵如图3.15所示。

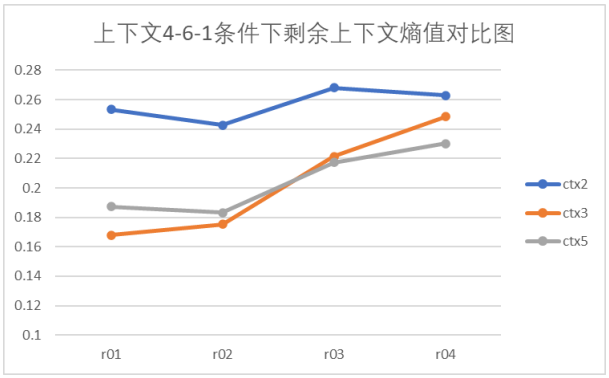


图 3.15　条件熵对比图 3

此时得到 ctx3 在 r01 与 r02 码率点下条件熵低于 ctx5，但是在 r03 与 r04 码率点下，ctx3 的条件熵比 ctx5 的条件熵高。对于这种情况，将四个码率点的条件熵取平均，然后比较其大小。最终得出 ctx3 在此条件下四个码率点的平均条件熵要小于 ctx5，因此选择 cxt3 作为第四个条件。

4、至此，仅剩下 ctx2 与 ctx5 未进行测试，在先前四类上下文的条件下，其条件熵如图3.16所示。

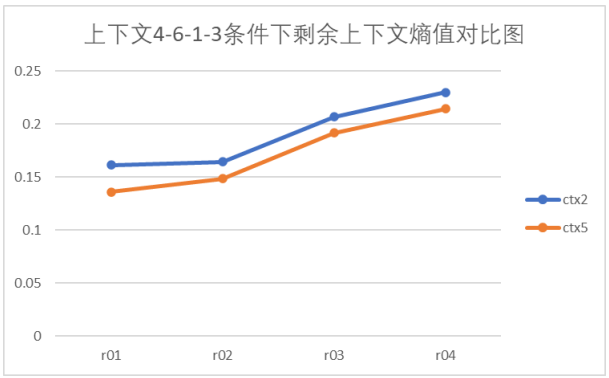
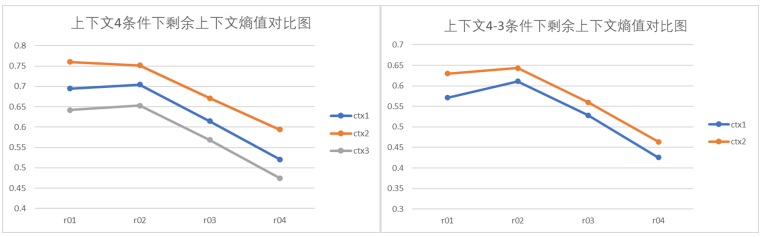


图 3.16　条件熵对比图 4

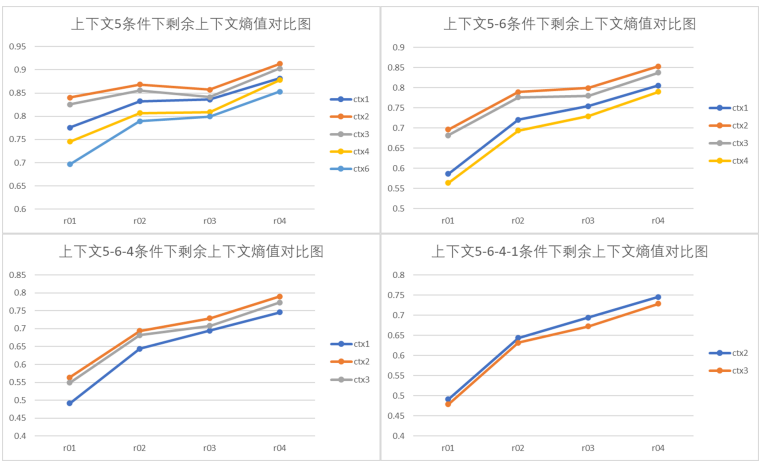
可以看出，ctx5 条件明显小于 ctx2 条件熵，因此 ctx5 将在 ctx2 前作为上下文被使用。

最终得到编码顶点标识信息时，其次要信息所包含的上下文的顺序应为 ctx4-ctx6-ctx1-ctx3-ctx5-ctx2。

对于 2bit 位置信息，它们条件熵的具体测试过程与标识信息一致，在此不再赘述，给出其测试过程的条件熵对比图如下所示：



(a) 高 bit 位置信息条件熵对比图



(b) 低 bit 位置信息条件熵对比图

图 3.17　位置信息条件熵对比图

由图可以得出，对于编码高 bit 位置信息时，它所用到的次要信息所包含的上下文的顺序为 ctx4-ctx3-ctx1-ctx2；对于编码低 bit 位置信息时，他所用用到的次要信息所包含的上下文的顺序为 ctx5-ctx6-ctx4-ctx1-ctx3-ctx2。

3.4　本章小结

本章首先介绍了 Trisoup 几何信息编码算法在利用基于 OBUF 的熵编码时所采用的上下文模型，同时分析了现有上下文模型可能存在的改进空间，然后通过计算并对比各个上下文模型的信息熵与条件熵的大小，得到了一套全新的局部最优的上下文模型。

第四章 实验条件与结果分析

MPEG 针对第一类静态点云序列和第三类动态点云序列构建了 TMC13 测试平台。并且针对不同版本的测试平台，发布了 Common test conditions for GPCC 的文件来统一测试条件。本次实验的参考测试平台是 TMC13v20.0，实验使用的测试条件、软件配置情况与 TMC13v20.0 的通用测试条件、软件参考配置保持一致。测试得到的性能结果也是相对于 TMC13v20.0 性能的增益情况。

4.1　实验配置

4.1.1 测试序列

本次实验测试的序列是根据点云的稠密程度划分的四类静态点云序列：具有连续表面的体素化点云（solid）、表面不完全连续的体素化点云（dense）、稀疏点云（sparse）、极其稀疏点云（scant）。下表 4.1 介绍了四类点云序列中各种序列的参数情况。点数表示每个点云序列包含的所有点的个数，几何精度指示每个点云序列的几何信息表示精度，帧数表示每个点云序列包含点云帧的个数，本次实验测试的都是单帧点云，所以帧数都为 1。属性格式指示的是点云属性的类型，有颜色 RGB 和反射率 reflectance 两种类型。

表 4.1　点云测试序列参数列表

序列种类	序列名	点数	几何精度/bit 位	帧数	属性格式
solid	basketball_player_vox11_00000200	2925514	11	1	RGB
	dancer_vox11_00000001	2592758	11	1	RGB
	facade_00064_vox11	4061755	11	1	RGB
	longdress_vox10_1300	857966	10	1	RGB
	loot_vox10_1200	805285	10	1	RGB
	queen_0200	1000993	10	1	RGB
	redandblack_vox10_1550	757691	10	1	RGB
	soldier_vox10_0690	1089091	10	1	RGB
	thaidancer_viewdep_vox12	3130215	12	1	RGB
dense	boxer_viewdep_vox12	3493085	12	1	RGB
	facade_00009_vox12	1596085	12	1	RGB
	facade_00015_vox14	8907880	14	1	RGB
	facade_00064_vox14	19702134	14	1	RGB
	frog_00067_vox12	3614251	12	1	RGB
	head_00039_vox12	13903516	12	1	RGB
	house_without_roof_00057_vox12	4848745	12	1	RGB
	landscape_00014_vox14	71948094	14	1	RGB
	longdress_viewdep_vox12	3096122	12	1	RGB
	loot_viewdep_vox12	3017285	12	1	RGB

序列种类	序列名	点数	几何精度/bit 位	帧数	属性格式
	vredandblack_viewdep_vox12	2770567	12	1	RGB
	soldier_viewdep_vox12	4001754	12	1	RGB
dense	arco_valentino_dense_vox12	1481746	12	1	RGB
	arco_valentino_dense_vox12	1481746	12	1	RGB
	egyptian_mask_vox12	272684	12	1	RGB
	palazzo_carignano_dense_vox14	4187594	14	1	RGB
	shiva_00035_vox12	1009132	12	1	RGB
	stanford_area_2_vox16	47062002	16	1	RGB
	stanford_area_4_vox16	43399204	16	1	RGB
	staue_klimt_vox12	499660	12	1	RGB
	ulb_unicorn_hires_vox15	63787119	15	1	RGB
	ulb_unicorn_vox13	1995189	13	1	RGB
dense	arco_valentino_dense_vox20	1530552	20	1	RGB
	egyptian_mask_vox20	272689	20	1	RGB
	facade_00009_vox20	1602990	20	1	RGB
	facade_00015_vox20	8929532	20	1	RGB
	facade_00064_vox20	19714629	20	1	RGB
	frog_00067_vox20	3630907	20	1	RGB
	head_00039_vox20	14025709	20	1	RGB
	house_without_roof_00057_vox20	5001077	20	1	RGB
	landscape_00014_vox20	72145549	20	1	RGB
	palazzo_carignano_dense_vox20	4203962	20	1	RGB
	shiva_00035_vox20	1010591	20	1	RGB
	stanford_area_2_vox20	47062018	20	1	RGB
	stanford_area_4_vox20	43399207	20	1	RGB
	staue_klimt_vox20	499886	20	1	RGB
	ulb_unicorn_hires_vox20	63864641	20	1	RGB
	ulb_unicorn_vox20	2000297	20	1	RGB

4.1.2 测试条件

GPCC 将通用测试条件根据几何、属性部分是否有损分为 4 种测试条件：C1(Lossless Geometry -Lossy attributes) 代表着几何无损、属性有损；C2(Near-lossless | Lossy Geometry -Lossy Attributes) 代表着几何有损、属性有损；CW(Lossless Geometry -Lossless Attributes) 代表着几何无损、属性无损；C4(Lossless Geometry -Near-lossless Attributes) 代表着几何无损、属性有限度有损。通过更改相应配置文件参数，可以实现不同条件的实验配置。本文对 Trisoup 几何信息编码的熵编码过程进行优化，测试条件是 C2，即几何有损，属性有损。

4.2 实验结果分析

在 GPCC 中衡量一个编解码器的性能主要从三个方面考虑，一是点云压缩后输出的比特流大小；二是编解码点云的时间复杂度；三是编解码前后点云的失真情况。由于本文仅对 trisoup 几何信息编码过程中的熵编码过程进行了改动，因此

最终的性能结果只需要关注点云压缩后输出的比特流的大小变化与编解码点云的时间复杂度的变化。D1，D2 用来衡量本文所提出的改进方案是否带来性能提升，其中，D1 用来表示点对点的几何失真，D2 用来表示点对面的几何失真。当其值小于零时，表明改进方案存在性能增益，当其值大于零时，表明改进方案相比于现有压缩方案带来了一定 loss。之所以可以用这一综合衡量指标代替比特流大小(bpip ratio) 来衡量性能变化，是因为本文所提出的改进方案不对几何重建产生任何影响，即不会影响几何失真。因此 D1，D2 的变化全部来自于比特流大小变化。

4.2.1 上下文改进结果

1、仅将编码顶点标识信息时基于动态 OBUF 的熵编码所用到的次要信息中的六类上下文顺序改为：ctx4-ctx6-ctx1-ctx3-ctx5-ctx2，然后对所有测试序列进行压缩性能测试，测试结果与 TMC13v20 的 anchor 进行对比，得到综合性能表如4.1所示：

C2_ai	lossy geometry, lossy attributes [all intra]					
	End-to-End BD-AttrRate [%]				Geom. BD-TotGeomRate [%]	
	Luma	Chroma Cb	Chroma Cr	Reflectance	D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	1.3%	1.3%
Dense average	0.0%	0.0%	0.0%	0.0%	0.9%	1.0%
Sparse average	0.0%	0.0%	0.0%	0.0%	-0.2%	-0.1%
Scant average	0.0%	0.0%	0.0%	0.0%	0.0%	0.1%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	0.5%	0.5%
Avg. Enc Time [%]	100%					
Avg. Dec Time [%]	100%					

图 4.1　仅修改标识信息上下文后性能表截图

2、仅将编码顶点位置信息的高 bit 位时次要信息上下文顺序改为:，然后对所有测试序列进行压缩性能测试，测试结果与 TMC13v20 的 anchor 进行对比，得到综合性能表如4.2所示：

C2_ai	lossy geometry, lossy attributes [all intra]					
	End-to-End BD-AttrRate [%]				Geom. BD-TotGeomRate [%]	
	Luma	Chroma Cb	Chroma Cr	Reflectance	D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	0.6%	0.6%
Dense average	0.0%	0.0%	0.0%	0.0%	0.4%	0.5%
Sparse average	0.0%	0.0%	0.0%	0.0%	0.6%	0.6%
Scant average	0.0%	0.0%	0.0%	0.0%	0.6%	0.6%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	0.5%	0.6%
Avg. Enc Time [%]	100%					
Avg. Dec Time [%]	100%					

图 4.2　仅修改标识信息上下文后性能表截图

3、仅将编码顶点位置信息的低 bit 位时次要信息上下文顺序改为:，然后对所有测试序列进行压缩性能测试，测试结果与 TMC13v20 的 anchor 进行对比，得到综合性能表如4.3所示：

C2_ai	lossy geometry, lossy attributes [all intra]					
	End-to-End BD-AttrRate [%]				Geom. BD-TotGeomRate [%]	
	Luma	Chroma Cb	Chroma Cr	Reflectance	D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	0.4%	0.4%
Dense average	0.0%	0.0%	0.0%	0.0%	0.2%	0.2%
Sparse average	0.0%	0.0%	0.0%	0.0%	0.6%	0.7%
Scant average	0.0%	0.0%	0.0%	0.0%	0.5%	0.5%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	0.4%	0.4%
Avg. Enc Time [%]	100%					
Avg. Dec Time [%]	100%					

图 4.3 仅修改标识信息上下文后性能表截图

4、将编码顶点标识信息和位置信息的上下文均按照条件熵递增的顺序修改后，我们得到本此研究实验的最终性能表如4.4所示：

C2_ai	lossy geometry, lossy attributes [all intra]					
	End-to-End BD-AttrRate [%]				Geom. BD-TotGeomRate [%]	
	Luma	Chroma Cb	Chroma Cr	Reflectance	D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	2.1%	2.1%
Dense average	0.0%	0.0%	0.0%	0.0%	1.2%	1.3%
Sparse average	0.0%	0.0%	0.0%	0.0%	0.2%	0.2%
Scant average	0.0%	0.0%	0.0%	0.0%	0.3%	0.3%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	0.9%	0.9%
Avg. Enc Time [%]	100%					
Avg. Dec Time [%]	100%					

图 4.4 仅修改标识信息上下文后性能表截图

4.2.2 结果分析

从4.2.1小节所展示的性能表可以看出，除了在修改标识信息上下文时，在 sparse 类型序列上整体有 0.2% 的增益外，不论是单独看修改每部分上下文后的性能表还是看将三套上下文模型均修改后的性能表，得到的均是负增益。但是在 C2 测试条件下，展开查看各个序列的性能结果后发现，有个别序列性能增益较大，如表4.2所示：

表 4.2 增益较大序列汇总表

序列所属类型	序列名称	D1	D2
dense	landscape_00014_vox14	-2.4%	-2.4%
multirow2*sparse	stanford_area_2_vox16	-3.3%	-3.3%
	stanford_area_4_vox16	-3.8%	-3.8%
multirow3*sparse	landscape_00014_vox20	-2.5%	-2.5%
	stanford_area_2_vox20	-3.4%	-3.2%
	stanford_area_4_vox20	-3.8%	-3.6%

由于同样也存在较多的负增益很大的序列，所以导致整体性能呈现负增益。本文在最初分析如何进行上下文改进时有提到，通过求解现有上下文的信息熵然后再基于信息熵最小的上下文一层一层按照条件熵递增的顺序排列剩余上下文的方法，得到的只能是局部的最优解，因为无法确定当选取信息熵较大的序列为第一个上下文后，其他上下文在此条件下会不会有更优异的性能，从而致使整体性

能优于本文所得到的上下文顺序。为此，本文进行了一项额外实验，假设现有上下文顺序是通过一定方法选择出来的较优顺序，那么不改变现有上下文模型中排在最前面的那类上下文的位置，而剩余上下文仍旧按照条件熵递增的顺序排列，得到如图4.5所示性能表。

C2_ai	lossy geometry, lossy attributes [all intra]					
	End-to-End BD-AttrRate [%]				Geom. BD-TotGeomRate [%]	
	Luma	Chroma Cb	Chroma Cr	Reflectance	D1	D2
Solid average	0.0%	0.0%	0.0%	0.0%	0.5%	0.5%
Dense average	0.0%	0.0%	0.0%	0.0%	-0.1%	0.0%
Sparse average	0.0%	0.0%	0.0%	0.0%	-1.1%	-1.1%
Scant average	0.0%	0.0%	0.0%	0.0%	-0.7%	-0.7%
Am-fused average	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!
Am-frame spinning average				#DIV/0!	#DIV/0!	#DIV/0!
Am-frame non-spinning average				#DIV/0!	#DIV/0!	
Overall average	0.0%	0.0%	0.0%	#DIV/0!	-0.4%	-0.4%
Avg. Enc Time [%]	100%					
Avg. Dec Time [%]	100%					

图 4.5　额外实验性能表截图

从表中可以看出，除了 solid 类型测试点云序列没有性能增益外，其他三类测试点云序列均有性能增益，并且 sparse 类型测试点云序列下甚至达到了 1.1% 的增益。这一结果佐证了了本文对提出的上下文模型出现性能负增益的结果分析。

另外，由于现有的动态 OBUF 技术的实现原理依据当前编码的 bit 位以及使用到的上下文信息，逐 bit 的更新次要信息，但是在 Trisoup 几何信息编码算法中，每类上下文模型的取值并不是二元的，因此动态 OBUF 可能并不能够很好的发挥其特点，甚至带来负面效果。这一猜想有待进一步实验验证。

第五章 总结与展望

5.1　总结

本文基于 MPGE 提出的 TMC13v20 测试平台，对 Trisoup 几何压缩算法进行了改进，主要体现在对熵编码过程所用到的上下文的顺序进行了调整，整体性能效果没达到预期目标，但是个别序列性能增益挺大，通过这一系列测试以及对结果的分析，发现了很多可能改进的方向。

文章首先介绍了 GPCC 中点云编解码的整体流程，其中对八叉树编码，Trisoup 几何编码，RATH 变换编码，Lifting 预测变换编码等几何编码和属性编码的常用方法进行了详细介绍，还对点云压缩算法的性能评价标准进行了介绍；然后通过对 Trisoup 几何信息熵编码的上下文构造的介绍，提出其中可以进行改进的方向，紧接着分小节对上下文信息熵与条件熵的具体测量过程进行了介绍，最终得到了一套全新的局部最优的上下文模型；本文第四章介绍了本次实验在 solid、dense、sparse、scant 四类测试序列，几何有损、属性有损的条件下得到的实验结果，结果表明本文提出的根据信息熵大小决定首位上下文，再根据条件熵大小决定后续上下文的方法并不理想，D1、D2 均有 0.9% 的负增益。

5.2　展望

通过对实验结果的分析，以及对动态 OBUF 的深入理解，得到以下改进方法：

1、目前所用到的各类上下文模型都不是二元取值的，都是由大于 1bit 位的信息构成，这样的上下文用于动态 OBUF 进行熵编码时，可能并不能够充分消除点云之间的空间相关性，甚至可能将熵编码的概率模型收敛至非理想值。这就导致最终的性能结果出现较大负增益。后续可以通过拆分某一上下文，然后交换该上下文多个 bit 位的位置来观察测试序列的压缩性能变化，从而判断是否可以从这一方向进行改进。

2、考虑到改进后的上下文模型对于某些序列存在着较大的增益，如表4.2所示。如果能找出这些序列之间的一个共性，并且能够有效的代表某一类点云序列，那么我们通过设置自适应判断的方法，为不同类型的点云序列采用不同的上下文模型，这样我们得到的性能增益一定是可观的。

Trisoup 几何压缩算法在 Ofinno，小米，OPPO 等公司的不断改进下，对于表面稠密点云展现出越来越好的压缩性能。截止目前，每次 MPEG 召开的国际会议上，都会有大量的有关 Trisoup 的改进提案涌现，同时针对表面连续的稠密点云（solid），MPEG 新发布了一套几何压缩测试模型（Ges-TM）。这暗示着这类几何压缩算法在未来能带来更加理想的性能结果。

致谢

参考文献

[1] J. Kammerl, N. Blodow, R. B. Rusu, et al. “Real-time compression of point cloud streams,” IEEE International Conference on Robotics and Automation, 2012, pp. 778-785.

[2] S. Schwarz et al. ”Emerging MPEG Standards for Point Cloud Compression,” in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 2019, March, vol. 9, no. 1, pp. 133-148.

[3] AVS 点云编码需求与技术讨论组. AVS 点云编码需求与技术分析报告 V0.1［C］// AVS_N2643，第 68 次会议. 中国青岛，2019 年 3 月.

[4] 万帅，杨付正. 新一代高效视频编码 H.265/HEVC：原理、标准与实现 [M]. 北京：电子工业出版社，2014. 142-152.

[5] S. Lasserre, D. Flynn, “[PCC] Global motion compensation for point cloud compression in TMC3,”ISO/IEC JTC1/SC29/WG11 Doc. M44751, Macau, October 2018.

[6] O. Nakagami, P. Chou, M. Krivokuća, et al. “Third Working Draft for G-PCC (Geometry-based PCC)”. ISO/IEC JTC1/SC29/WG11MPEG/ N17770, Ljubljana, July 2018.

[7] C.Cao, M.Preda, and T.Zaharia. “3D Point Cloud Compression: A Survey”. In The 24th International Conference on 3D Web Technology. Association for Computing Machinery, New York, NY, USA, 1–9.

[8] 程效军，贾东峰. 海量点云数据处理理论与技术 [M]. 上海：同济大学出版社，2014. 1-42.

[9] Fleishman S, Drori I, Cohen-Or D. Bilateral mesh denoising[C]//ACM transactions on graphics (TOG). ACM, 2003, 22(3): 950-953.

[10] MPEG 3D Graphics Coding, “G-PCC codec description,”ISO/IEC JTC 1/SC 29/WG 7 N0099, Online, April 2021.

[11] Z.Gao, D.Flynn, A.Tourapis, et al. “[G-PCC]Predictive Geometry Codin,” ISO/IEC JTC1/SC29/WG11 MPEG/m51012, Geneva, CH, October 2019.

[12] K.Mammou, D.Flynn, A.Tourapis, “[G-PCC] Optimization of the predictive coding scheme for Spinning Lidars,”ISO/IEC JTC1/SC29/WG11 MPEG/ m53618, Alpbach, AT, April 2020.

[13] D. Tian, H. Ochimizu, C. Feng, et al. “Geometric distortion metrics for point cloud compression,” 2017 IEEE International Conference on Image Processing (ICIP), 2017, pp. 3460-3464.

[14] A.Javaheri, C.Brites, F.Pereira, et al. “A Generalized Hausdorff Distance Based Quality Metric for Point Cloud Geometry,” 2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX), 2020, pp. 1-6.

[15] S.Lasserre, D.Flynn, “[PCC] On motion compensation for geometry coding in TM3,”ISO/IEC JTC1/SC29/WG11 MPEG2018/m42521, San Diego, CA, US, April 2018.

[16] S.Lasserre, D.Flynn, “[PCC] Global motion compensation for point cloud compression in TM3,”ISO/IEC JTC1/SC29/WG11 MPEG2018/m44751, Macau, China, October 2018.

[17] J.Kim, K.Kim (Kyung Hee University), J.Lee, et al. “[G-PCC] A method of applying the global motion matrix according to the compression condition,”ISO/IEC JTC1/SC29/WG7 MPEG/m57484, Online, July 2021.

[18] J.Kim, K.Kim (Kyung Hee University) J.Seo, et al. “[G-PCC] Road and objects segmentation for global motion prediction,” ISO/IEC JTC1/SC29/WG11 MPEG/m55390, Online, October 2020.

[19] L.Van, K.Cao, Adarsh K.Ramasubramonian, et al. “[G-PCC]Improved global motion estimation for G-PCC,”ISO/IEC JTC 1/SC 29/WG 7 m56113, Online, January 2021.

[20] Adarsh K. Ramasubramonian, G.Auwera, L.Van, et al. “[G-PCC] Additional results on inter prediction for predictive geometry,”ISO/IEC JTC 1/SC 29/WG 7 m56841, April 2021.

[21] K.Loi, T.Nishi, T.Sugio, “[G-PCC]Inter Prediction for Improved Quantization of Azimuthal Angle in Predictive Geometry Coding,”ISO/IEC JTC 1/SC 29/WG 7 m57351, Online, July 2021

[22] J.Taquet, S.Lasserre, S.Gao,et al. “[G-PCC]Improved Quantization of Azimuthal Angle in Predictive Geometry Coding,”ISO/IEC JTC 1/SC 29/WG 7 m55979, online, January 2021.

[23] MPEG 3D Graphics coding, “Common Test Conditions for G-PCC,” ISO/IEC JTC1/SC29/WG7, 134th MPEG meeting, OnLine, April 2021.

参考文献