面向移动平台的异构三维模型压缩方法

作者 1[1,2], 作者 2[1)*], 作者 3[2)]

[1)] (中文工作单位名称　中文城市名　邮编)
[2)] (中文工作单位名称　中文城市名　邮编)
(通讯作者的 E-mail 地址)

摘　要: 随着移动设备的广泛应用和性能的日益增强，三维模型在移动平台上的应用需求不断扩大. 与传统的台式机和游戏主机相比，移动设备的存储和计算资源相对有限，因此三维模型的高效压缩尤为重要. 针对这一需求，本文提出了一种基于 CPU 的两阶段流水线大小核异构压缩方案，有效利用了移动设备中的异构多核架构. 通过在流水线的第一阶段使用超大核进行网格拓扑数据的压缩，而第二阶段由大核完成对顶点数据的压缩，本方案显著优化了计算资源分配和压缩效率. 与传统单核 CPU 压缩方法相比，实验结果显示，我们的方法在压缩性能上平均提高了 22%. .

关键词: 异构算法; 压缩; MPSoC; 三维模型
中图法分类号: TP391.41　　**DOI:** 10.3724/SP.J.1089.2024.论文编号

Heterogeneous 3D model compression method for mobile platforms

Author 1[1,2], Author 2[1)*], and Author 3[2)]

[1)] (*英文工作单位, 英文城市名*　邮编)
[2)] (*英文工作单位, 英文城市名*　邮编)

Abstract: With the widespread application and increasing performance of mobile devices, the demand for the application of 3D models on mobile platforms continues to expand. Compared with traditional desktop computers and game consoles, mobile devices have relatively limited storage and computing resources, so efficient compression of 3D models is particularly important. In response to this demand, this paper proposes a CPU-based two-stage pipeline heterogeneous compression scheme for large and small cores, which effectively utilizes the heterogeneous multi-core architecture in mobile devices. By using very large cores to compress grid topology data in the first stage of the pipeline, and using large cores to compress vertex data in the second stage, this solution significantly optimizes computing resource allocation and compression efficiency. Compared with traditional single-core CPU compression methods, experimental results show that our method improves compression performance by an average of 22%.

Key words: Heterogeneous algorithms; compression; MPSoC; 3D model

收稿日期: 20**-**-**; 修回日期: 20**-**-**.

.

三维模型在现代生活中扮演着越来越重要的角色，它们广泛应用于多个领域，极大地丰富和改进了我们的日常经验．在娱乐行业，三维模型是创建引人入胜的视频游戏和电影特效的基础，提供了视觉上的震撼和更深层的沉浸感．在教育和培训领域，三维模型通过模拟复杂的生物结构、历史遗迹或科学概念，帮助学生和专业人士以直观的方式学习和理解复杂信息．此外，三维模型在医疗领域中也非常关键，它们用于精确模拟人体部位，辅助手术计划和医生的训练，提高手术的成功率和安全性．在工程和制造业，三维模型用于产品设计、故障分析和模拟测试，加速产品开发流程并优化性能．随着技术的进步，三维模型还在城市规划、虚拟现实以及增强现实等领域中展现出其独特的价值，成为现代社会不可或缺的技术元素．随着移动设备的广泛应用和性能的日益增强，三维模型在移动平台上的应用需求不断扩大，特别是在游戏、虚拟现实以及增强现实等领域．这种需求的增长背景主要是由于移动技术的快速发展以及消费者对于更加丰富多彩和互动性更强体验的追求．移动平台上的三维应用不仅提升了用户的交互体验，也推动了新型娱乐和教育内容的创新发展．然而，与传统的台式机和游戏主机相比，移动设备在处理高质量三维图形时，由于其存储和计算资源的限制，面临着更为复杂的挑战．

当前，面向移动平台的三维模型压缩研究主要关注于如何在保证模型质量的同时，有效减少数据的存储占用和加速模型的加载与渲染速度．现阶段的研究展示了多种压缩技术的发展，其中就包括几何结构压缩方法[1, 2, 3, 4, 5, 6, 7]，根据不同应用场景下的具体需求，这些方法大致可以分为几类：多边形简化技术[2]，是指通过减少模型中的顶点数和面数来降低模型复杂度；顶点量化和索引压缩[3]，是指通过简化模型的表示形式来减少所需的存储空间；预测压缩和基于变换的压缩方法[4]，则是指利用数学和统计技术优化数据的存储；而簇化方法[5]，则是指通过将模型划分为多个子集来实现高效的数据管理．除了几何结构的压缩，研究者还对拓扑结构以及具体的编码格式进行了许多的探索．这些技术努力在不牺牲太多渲染质量的前提下，尽可能地压缩数据量，但许多方法在实际应用中仍面临着执行效率或兼容性问题．此外，三维模型数据的复杂性和多样性也使得找到一种既通用又高效的解决方案变得更加困难．除此以外，异构计算技术的发展为优化移动设备上的三维数据处理提供了新的可能．当前市场上的移动设备越来越多地采用了包含多种处理单元的异构架构，例如集成有 GPU 和多核 CPU 的系统．这种硬件的多样性提供了执行并行计算的可能，从而有助于提高应用的响应速度和处理能力．然而，如何设计适应异构计算环境的压缩算法，以充分利用移动设备的计算资源，仍是一个亟待解决的问题．这要求研究者不仅需要深入理解移动设备的硬件架构，同时还需充分考虑算法的可移植性和效率，确保在不同设备上都能获得最佳性能．

本文提出了一种基于 CPU 的两阶段流水线大小核异构压缩方案，并与单核 CPU 的情况进行了对比．该方案优化了数据处理流程，通过大小核的协同工作，提高了计算效率．研究结果表明，本文提出的方法的压缩性能平均提升了 22%.

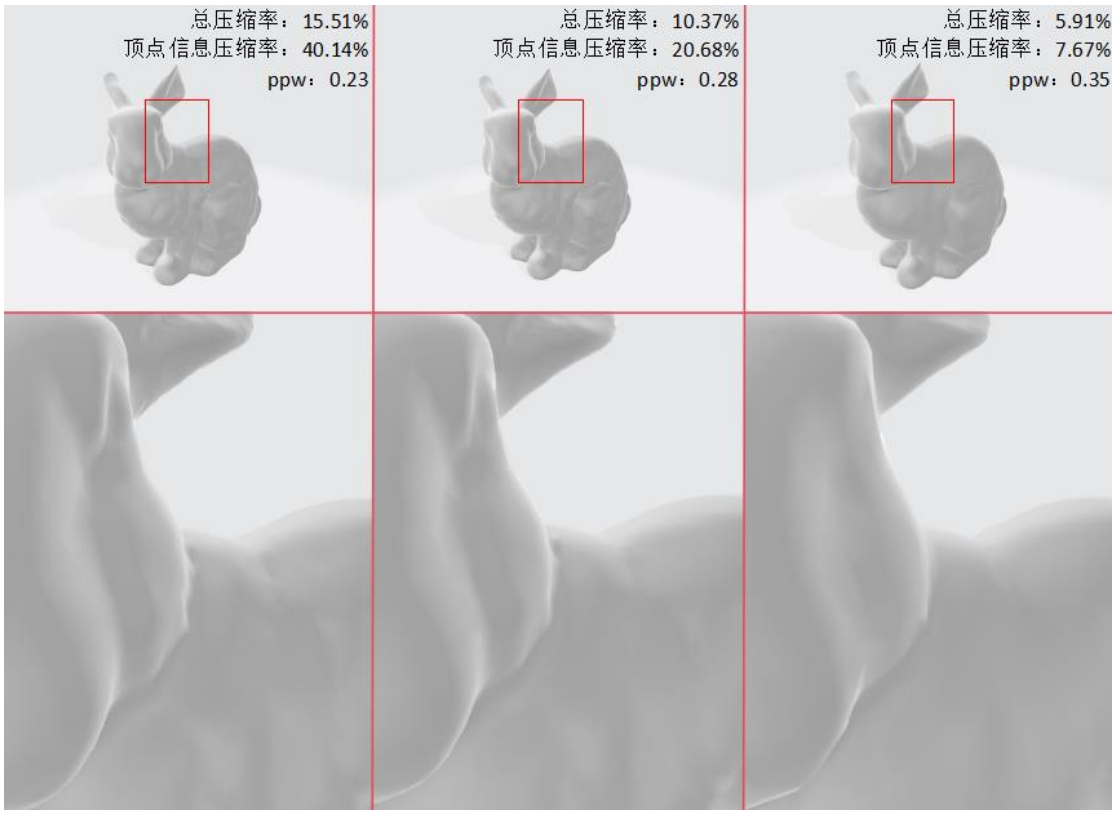

图 1　不同压缩率下的模型与局部放大图

1　相关工作

1.1　三维模型中几何数据的压缩

三维模型是现实或虚构物体在数字环境中的三维表示. 一个完整的三维模型通常包含以下几种类型的数据: 几何数据、材质与纹理、法线数据、动画数据、场景数据等. 本文主要关注模型的几何数据.

模型的几何结构即构成模型的顶点、边、面的数据，这些数据定义了模型的基础形状和外观. 研究者们提出了多种几何结构压缩方法[1, 2, 3, 4, 5, 6, 7]，旨在解决不同应用场景下的具体需求. 这些方法大致可以分为几类: 多边形简化技术[2]通过减少模型中的顶点数和面数来降低模型复杂度；顶点量化和索引压缩[3]通过简化模型的表示形式来减少所需的存储空间；预测压缩和基于变换的压缩方法[4]则利用数学和统计技术优化数据的存储；而簇化方法[5]则通过将模型划分为多个子集来实现高效的数据管理. 每种方法都有其独特的优点和应用场景，对于推动三维模型处理技术的发展起到了关键作用.

Garland 等人首次提出了使用二次误差度量(Quadric Error Metrics, QEM)的方法来进行网格简化[2]. 这种方法的核心是为每个顶点定义一个错误度量二次方程，该方程衡量了将顶点移动到任意位置所产生的视觉误差. 通过这种方式，算法能够评估合并任意两个顶点所引入的视觉误差，并选择误差最小的顶点对进行合并. 随后算法不断重复这个过程，每次都移除误差最小的顶点对，直到达到用户定义的简化程度或顶点数目. 这种方法特别注重保持模型的原始外观，使得简化后的模型在视觉上与原模型保持尽可能接近.

Chou 等人提出了一种基于向量量化的顶点压缩技术[3]，他们证明了向量量化(Vector Quantization, VQ)是一种有效的三维模型顶点数据压缩技术. 向量量化通过使用

非结构化码本(Unstructured Code Books)和乘积码金字塔向量量化器(Product Code Pyramid Vector Quantizer)来实现预测向量量化方法. 该技术与大多数现有的网格连接编码方案兼容，且不需要使用熵编码. 此外，作者还指出他们的向量量化方案不仅可以用于减少几何数据的比特流大小，从而降低带宽需求和缓解传输瓶颈，还可以通过加速线性顶点变换的计算来用于复杂度降低. 结果表明，采用该压缩方案的顶点集合的解码和变换时间大约只需传统未压缩方法的60%. 这项工作为三维模型的高效传输和处理提供了一种有效的解决方案，并且展示了向量量化在减少三维网格顶点数据处理时间和空间上的潜力.

Taubin提出了一种基于多分辨率表示的压缩方法[4]，通过对模型执行一系列平滑和细化操作，建立了模型的多分辨率层次结构. 在这个过程中，每一层的几何细节被编码为前一层的残差，这些残差信息可以用于恢复高分辨率模型. 通过仅存储较低分辨率的基础网格和必要的残差信息，可以显著减少所需存储空间，实现几何数据的压缩. 此外，Taubin还探讨了如何利用网格拉普拉斯算子进行几何压缩，该方法通过分析网格的光滑特性来优化残差编码，进一步提高压缩效率. 这种方法的关键在于能够在保持模型视觉质量的同时，有效减少表示模型所需的数据量.

Alexa 等人提出了一种簇化方法[5]，通过移动最小二乘法(Moving Least Squares, MLS)来处理点集数据，并生成平滑的流形表面. 这种方法的核心是局部地重新采样点集，以此来定义和优化表面的几何表示. 他们利用点的上采样和下采样技术调整点集的密度，从而在不牺牲表面的几何和视觉特性的前提下减少数据量. 此外，通过引入多分辨率表示和使用高效的数据结构如八叉树，进一步优化了数据存储和查询过程，有效达到了数据压缩的目的. 这种簇化和重采样方法有效地将点集数据转化为更为平滑且连续的表面表示，使得最终的表面模型不仅几何上更加精确，还能在视觉上提供更高质量的渲染效果.

此外，研究者们还提出了许多拓扑数据的压缩方法. 与直接处理顶点和面数据的几何压缩技术不同，这种方法通过编码网格的结构来实现数据的压缩. Rossignac 提出了一种称为 Edgebreaker 的压缩方法[8]，专门用于三角网格的拓扑数据. Edgebreaker 通过递归地”剥离”三角形来压缩网格，生成一个小型的编码序列，这一序列反映了网格的拓扑结构. 该方法高效地将三角网格的连接性压缩到接近理论最小限度，通常每个三角形仅需约 2 位存储空间，大幅减少了数据的存储和传输需求. 具体来说，Edgebreaker 通过对三角网格执行一个简单的遍历过程，在此过程中，每步操作都按照如何从网格中剥离一个三角形来记录. 这些操作被编码成一个紧凑的字符串，可以在解码时重构整个网格的拓扑.

1.2 异构计算

异构计算是一种使用不同类型的处理器或核心来优化计算资源的方法，目的是提升计算效率和能效. 在这种计算模型中，不同的处理单元各自承担最适合其硬件特性的计算任务[9, 10, 11, 12, 13].

Ali 等人详细地探讨了异构计算系统中任务映射和调度的复杂问题并给出了对应的解决方案[13]，特别是如何将具有不同计算需求的任务有效地分配给包含多种机器类型的异构环境，以优化特定的性能度量，比如整体任务的执行时间. 异构计算环境涵盖了从大型分布式共享内存机器到小型共享内存机器的多种机器类型，这些通过高速连接组成一个元计算机或计算网格. Ali 等人的

核心贡献在于详细介绍了任务与机器的匹配问题，即如何将任务映射到最合适的机器上以提升系统性能．映射问题被定义为一个核心挑战，因为不同机器的处理能力和特性差异较大．为此，作者提出了静态和动态映射策略：静态映射在任务执行前做出决策，适用于任务特性和系统资源状态已知的情况；动态映射则在任务执行过程中根据实时信息调整映射决策，适用于任务到达时间和机器可用性不确定的环境．这些详细探讨和解决方案为设计和优化异构计算系统提供了理论基础和实践指导，使得不同类型的计算资源可以更有效地协同处理复杂的计算任务．

近年来，异构计算与深度学习的结合已成为推动计算性能提升的关键因素．深度学习模型因其复杂的数据处理和计算密集型的特性，特别适合异构计算环境，如利用GPU、TPU 和其他专用硬件加速计算．这种结合使得训练和推理过程大幅加速，支持了更大规模的模型和数据集，从而在视觉识别、自然语言处理和其他 AI 领域实现了显著的进步．异构计算平台提供的并行处理能力和高效的数据流动性，为深度学习应用带来了前所未有的速度和效率，使得实时数据处理和复杂模型应用成为可能．

Wang S 等人提出了一种名为”Pipe-it”的框架[14]，该框架通过流水线设计优化 CNN 层的分配，以提高推理吞吐量．这种方法能够有效利用多核处理器的全部计算资源，而不仅仅是分配计算任务到各个核心．通过对比传统的核心层平行处理策略，Pipe-it 在平均情况下实现了 39%的吞吐量提升. Wang 等人还研究了一个性能预测模型，用于预测不同核心配置下每个 CNN 层的执行时间，进而指导流水线设计和层分配的最优化．这种方法不仅提高了处理速度，还通过减少不同处理核心间的通信需求来优化能效．

Wang M 等人提出了一种名为 AsyMo 的新技术[15]，这是一种线程池实现，用于优化深度学习框架的推理性能和能效．AsyMo 的核心设计原则是利用深度学习推理的执行确定性，通过联合考虑模型结构和硬件特性，离线构建最优的执行计划．这包括基于成本模型的分区和考虑不对称性的任务调度，以适当地划分和公平地调度任务．对于节能，AsyMo 根据模型的数据重用率确定最低能耗频率．这一技术在不同的模型和深度学习框架上进行了评估，显示了显著的性能和能效改进，例如在卷积主导的模型中，AsyMo 展示了高达 46%的性能和 37%的能效提升．

2 背景

2.1 MPSoC 的异构性

MPSoC 中各种形式的异构性大致可分为两类：性能异构性和功能异构性． 前者是指将具有相同功能(相同指令集架构或 ISA)但不同功耗性能特性的内核集成在一起，而在后者是指将具有截然不同功能(不同 ISA)的内核散布在同一芯片上．

核心之间性能异构性的差异源于不同的微架构特征，例如有序核心与无序核心．ARM big.LITTLE 架构是一个代表性的例子，它将高性能的乱序 ARM Cortex-A15 内核(大内核)与低功耗的顺序 ARM Cortex-A7 内核(小内核)集成在一起．这种性能异构性可以使 MPSoC 在保持低功耗的同时，更好地满足性能需求．

功能异构多核由具有不同功能的核组成．这在嵌入式领域相当常见，其中 MPSoC 由通用 CPU 内核、GPU 内核、DSP 模块和各种硬件加速器或 IP 模块(例如视频编码器/解码器、成像等)组成．这里引入异构性是为了满足严格的功耗预算下的性能需求，因为某些工作负载更适合 GPU、DSP

或固定功能加速器.

最新的移动平台(智能手机、平板电脑和可穿戴设备)在同一芯片中同时采用性能和功能异构性. 以我们研究中使用的MPSoC(Qualcomm Snapdragon 855)为例，从图 2 可以看出，它由 1 个高性能CPU(Qualcomm Kryo 485 Gold@2841 MHz, 基于 A76)、3 个中性能 CPU(Qualcomm Kryo 485 Gold @ 2419 MHz，基于 A76)和四个低性能CPU(Qualcomm Kryo 485 Silver @ 1785 MHz，基于 A55)组成. GPU 方面，搭载 Adreno(TM) 640.

图 2 高通骁龙 855 体系结构图

2.2 动态电压频率调整技术

动态电压频率调整技术（Dynamic Voltage and Frequency Scaling, DVFS）是一项在运行时根据需求动态调整 CPU 电压和频率的技术[16, 17]. 该技术在确保处理器稳定运行的基础上，根据处理器当前的工作负载和功耗自动调整电压与频率，实现性能提升与节能的双重目标. 在 DVFS 技术的应用过程中，处理器会根据实时负载自动选择最合适的电压和频率设置. 当处理器负载较低时，通过降低电压和频率来减少功耗和热量，达到节能效果；当负载较高时，则通过提升电压和频率来增强性能，以满足处理器的运行需求. 此外，DVFS 技术还可以依据处理器的温度和电源等环境因素进行调整，进一步优化处理器的性能和稳定性.

在现代计算机系统中，DVFS 技术被广泛应用于各种处理器和电子设备中，包括中央处理器(CPU)、图形处理器(GPU)、系统芯片组和存储控制器等. 它是一种重要的节能技术，能够在不影响计算机性能的情况下，有效地降低系统的功耗和温度，从而提高系统的可靠性和稳定性.

处理器内核的频率会影响其性能、能耗和热行为. 现有研究表明，任务在核心上的执行时间 T 可以使用如下线性关系进行建模:

$$T = \frac{\alpha}{f} + \beta \tag{1}$$

其中 f 为频率，α 、β 为运行时在两个极值点插值得到的常数. 随着频率的增加，功耗呈二次方增加:

$$P = ACV^2 f + P_i \tag{2}$$

其中，A 为活跃因子，C 为电容，V 为电压，f 为频率，P_i 为相应频率设置下的空闲功耗. 核心温度随着频率的增加而增加得更快，尽管无法准确指定关系. 动态电压频率调节(DVFS)算法可以根据当前工作负载动态调整工作频率和相应的电压，是控制核心性能、节能和热节流的最广泛使用的方案之一.

3 本文方法

本文方法是由 Edgebreaker 和熵编码两个阶段组成流水线，从而达到加速压缩过程的目的. 由于三维网格的拓扑信息压缩和顶点信息压缩具有一定的独立性，因此我们提出基于 CPU 的大小核异构方案，以更好地利用 CPU 的不同内核. 具体来说，我们将 Edgebreaker 算法放在超大核上运行，将熵编码放在大核上运行，实现并行化加速.

Edgebreaker 在压缩网格的过程中会不断遇到新的顶点，我们采用平行四边形预测法[18]对新遇到的顶点进行预测，然后计算预测顶点与真实顶点的差值，并存储这些预测残差. 这些预测残差将会作为熵编码的输入进行进一步的压缩. 我们方法的流程图如下图所示:

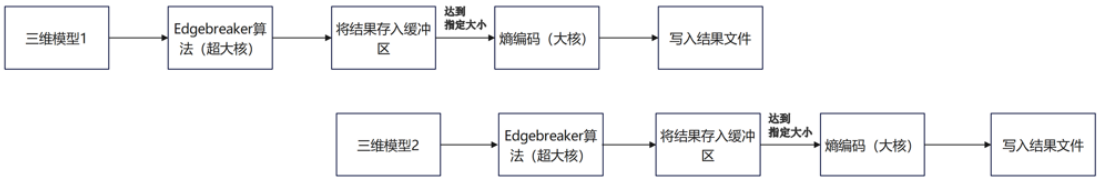

图 3　基于 CPU 的大小核调度方案

考虑到我们希望压缩的过程尽可能短，同时又尽可能节约能耗. 我们可以定义如下目标函数来解决如何进行负载的分配和各个内核的频率设置问题:

$$\arg\min_{f_S, f_B} \omega_t \cdot \left\| T_{EB} - T_E \right\|^2 + \omega_p \left(E_S T_S^2 + E_B T_B^2 \right) \tag{3}$$

其中，T_{EB} 是 Edgebreaker 执行的时间，T_E 是熵编码执行的时间，E_S 为超大核上的能耗，T_S 为超大核上执行的时间，E_B 为大核上的能耗，T_B 为大核上执行的时间，f_S 和 f_B 分别是超大核和大核的频率，ω_t 和 ω_p 分别是代表性能和功耗的权重.

基于以上分析，我们通过解决一个优化问题，确定任务在不同内核上的分配比例以及各个内核的最佳运行频率，以实现功耗最小化和性能最大化的目标. 同时，实时监测设备的负载和功耗，根据预测的最佳频率动态调整设备的工作频率，以实现最佳性能功耗平衡.

3.1　Edgebreaker

Edgebreaker 算法是一种专为压缩 3D 三角网格而设计的复杂方法，通过对连接性和几何信息进行编码，有效降低存储和传输要求. 在算法执行前，我们量化所有网格顶点，将它们的坐标从浮点表示转换为整数索引，从而为高效压缩准备几何数据. 此预处理步骤对于促进后续编码过程至关重要.

初始化完成后，Edgebreaker 算法将遍历三角形网格. 在此遍历过程中，每个三角形根据其与网格边界的拓扑关系被分为几种类型(C、L、R、E、S)之一. 在介绍这五种三角形之前，我们先介绍门(gate)的概念: 对于一个有边界的网格，门是处于网格边界上的一个边，且具有方向，在一个三角形中，默认的方向为逆时针方向.

接下来简要介绍这五种三角形的含义:

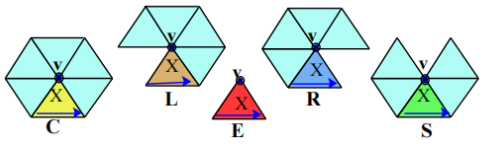

图 4　Edgebreaker 中的五种三角形

“C” 代表 Connect. 当处理到一个新的三角形时，如果此时的门与这个门相对的顶点(即不与这个边相连的第三个顶点)尚未被包含在已处理区域内，那么这个三角形被标记为 C. 它实际上是向外扩展已处理区域的边界. 这个操作使得处理的边界向前推进，同时新增一个未处理的顶点.

“R” 代表 Right. 如果与此时门相对的顶点位于边界上，且这个顶点在边界循环的顺序上紧邻此时门后面，则该三角形标记为 R. 这通常意味着新的三角形沿着已处理边界的”右转”.

“L”代表 Left，与 R 相反. 如果与此时门相对的顶点位于边界上，且这个顶点在边界循环的顺序上紧邻此时门的前面，则该三角

形标记为 L．这表示新三角形沿已处理边界的”左转”.

“S”代表 Start，这是用来标记一个新开始的区域的第一个三角形．在开始处理整个网格或一个独立的网格部分时使用．S 类型的三角形是处理序列中的起点，表示一个全新处理序列的开始．如果与此时的门相对的顶点位于边界上，且该顶点既不紧邻门的前面也不紧邻门的后面，则该三角形被标记为 S.

“E”代表 End．这个标记用于那些通过两个边与已处理区域相连的三角形，如果此时与门相对的顶点在边界循环上既紧邻门之前又紧邻门之后(即边界缩小到该三角形的边界)，则该三角形标记为 E．此标记的三角形表明了一个处理的封闭区域的结束，这通常意味着当前处理的”口袋” 或子区域已经完全封闭，可以开始回溯或者处理新的区域.

这些分类至关重要，因为它们不仅代表了三角形之间的空间和拓扑关系，还决定了算法的遍历路径和压缩策略．通过这种方式，Edgebreaker 可以有效地将一个复杂的三角形网格转换成一串较为简单的符号序列，这些序列在存储和传输时占用的空间远小于原始数据，且可以通过解码过程完整地恢复原始网格结构.

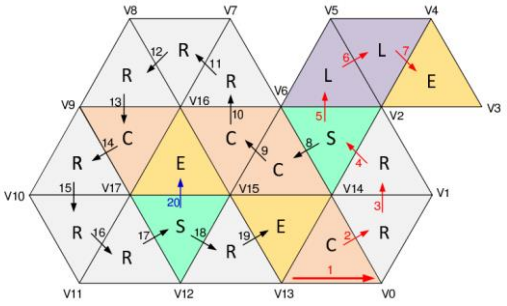

图 5 具有一个边界循环且不具有把手和孔的网格

下面以具有一个边界循环且不具有把手和孔的网格为例说明算法的流程．选取 $\overrightarrow{V_{13}V_0}$ 作为起始边，由于与 $\overrightarrow{V_{13}V_0}$ 相对的点 V_{14} 并未包含在已处理区域内，所以第一个三角形被标记为 C．将它剔除后，$\overrightarrow{V_0V_{14}}$ 变成了一个门，由于 V_1 在边界循环上处于该门之后，所以第二个三角形被标记为 R．当处理到第四个三角形时，此时的门为 $\overrightarrow{V_{14}V_2}$，与其相对的顶点 V_6 在边界循环上既不紧邻门之前，也不紧邻门之后，属于在边界上的其他位置，所以第四个三角形被标记为 S.

在剔除第四个三角形之后，原来属于一个整体的网格被分割成了两个部分，成为了左子网格和右子网格.

第五个三角形属于右子网格的一部分，此时的门为 $\overrightarrow{V_6V_2}$，与其相对的顶点为 V_5，该顶点在边界循环的顺序紧邻门的前方，所以第五个三角形被标记为 L．再向后处理到第七个三角形时，右子网格已经只剩一个三角形，此时门为 $\overrightarrow{V_4V_2}$，与门相对的顶点为 V_3，所以该三角形被标记为 E，随后转而处理左子网格．以此流程处理后，原网格被编码为 CCRRRSLCRSERRELCRRRCRRRE.

在剔除三角形的过程中，对于新遇到的顶点(不在已处理的网格中)，我们采用平行四边形预测方案来估计新顶点的位置，再对预测结果与实际结果取差值，得到预测残差．该方案依赖于网格的局部几何形状倾向于类似于平行四边形的假设．这种几何预测与残差编码步骤相结合，显着提高了压缩效率.

上述过程的算法步骤如下:

输入．三角形网格 ***T***.

输出．编码序列 ***S*** 与顶点预测残差 ***R***.

Step1．量化所有网格顶点，以生成整数量化索引.

Step2．当网格未被完全编码时依次执行下列步骤，否则算法结束.

Step3．找到当前三角形的类型，并将其添加到操作代码序列中.

Step4．预测在当前三角形中新遇到的顶点

的位置.

Step5. 计算预测残差.

Step6. 移动到一个特定的相邻三角形，并回到算法的第二步.

一旦网格内的所有三角形都被遍历并适当编码，压缩过程就结束，从而产生包括操作码序列和编码的预测残差序列的紧凑表示. Edgebreaker 算法能够压缩连接性和几何信息，这使其对于大型 3D 模型特别有效，从而大幅减少存储和传输所需的数据量. 这使得 Edgebreaker 成为 3D 建模领域的宝贵工具，有助于有效处理和分布复杂的几何数据.

3.2 熵编码

对于 Edgebreaker 中产生的预测残差，我们采用熵编码的方式进一步进行压缩.

3.2.1 算术编码

算术编码作为一种高效的数据压缩技术[19]，常用于文件压缩和通信系统中. 这种方法的基本思想是将一个消息(字符串)转换成一个单一的数字，该数字在 0 和 1 之间，并且可以精确地表示原始数据.

算术编码的工作原理是利用整个消息的统计特性. 首先，它需要一个概率模型，用于预测各个符号的出现概率. 然后，根据这些概率，算术编码将消息映射到一个逐渐缩小的数值区间. 每读入一个新的符号，就基于符号的概率缩小当前的区间. 具体来说，每个符号对应区间的大小是依据该符号的出现概率来确定的. 例如，如果某个符号在消息中出现的概率很高，则它对应的区间也比较大；相反，出现概率低的符号则对应一个较小的区间. 随着更多符号的读入，最终形成的数值区间将变得越来越小，精确地表示了原始的消息. 在实际编码过程中，算术编码通过不断细分区间，最终生成一个位于最后区间内的任意数值，这个数值可以通过二进制形式高效地存储或传输. 解码时，接收方只需利用相同的概率模型和这个数值，就可以逐步恢复出原始的消息.

3.2.2 Gzip

我们还选择了通用压缩算法 GZip[20, 21]，用于压缩 Edgebreaker 阶段产生的预测残差. GZip 算法由两个主要步骤组成：LZ77 编码和 Huffman 编码.

LZ77 算法是 Abraham Lempel 与 Jacoc Ziv 在 1977 年共同提出的一种基于字典的典型压缩算法. 它是基于一种简单的数据压缩思想：在一个字符串中，某些子串可能在另一个位置上已经出现过，那么就可以用之前出现过的子串来替换现在出现的子串，从而减少重复的数据量. 即：构建一个字典，用索引来代替重复出现的字符或字符串.

由于 LZ77 压缩算法的大部分时间都会花在字符串匹配上. 如何高效查找最长匹配串便成为提升 LZ77 压缩算法性能和效率的关键所在. 如果采用直接匹配的方式，将每个数据都与前面出现过的数据一一比对，所需的时间复杂度为 $O(n^2)$，当待压缩文件比较大时，会严重影响压缩的速度. 而采用哈希映射的方式可以将时间复杂度降为 $O(1)$，大幅度提高查找效率.

在寻找匹配串的过程中，使用”哈希桶”保存每三个相邻字符构成的字符串中首字符的窗口索引. 压缩过程中每当遇到新字符时，进行如下操作:

(1)利用哈希函数计算该字符与紧跟其后的两个字符构成字符串的哈希地址.

(2)将该字符串中首字符在窗口中的索引插入上述计算出哈希位置的哈希桶中，返回插入之前该桶的状态.

(3)根据(2)返回的状态检测是否找到匹配串，如果当前桶为空，说明未找到匹配，将该字符作为源字符写到压缩文件中；否则表示找到匹配，定位到匹配串位置开始找最长匹配，找到后将<距离，长度>对写到压缩

文件中.

在”哈希桶”的设计时，每三个字符为一组进行插入，三个字符便有 2^{24} 种取值，即：桶的个数需要 2^{24} 个. 而每个字符占 2 字节，桶总共需要占 32M 字节，是一个非常大的开销，会降低程序的运行效率，因此我们将哈希桶的个数缩小为： 2^{15} (32K).

通过多对一的映射方式，大大缩小查找的范围，但是由于哈希地址多于哈希表的位数，无法避免哈希冲突的发生. 解决哈希冲突的方法一般有：开放寻址法、链地址法(拉链法)、再哈希法、建立公共溢出区等方法. 如果使用链地址法解决哈希冲突，链表中的节点要不断申请与释放，而且浪费空间，影响查找效率. 此处采用开放地址法，开辟与之前动态滑动窗口大小一样的 64K 内存空间，也将其分为两部分，如下图所示:

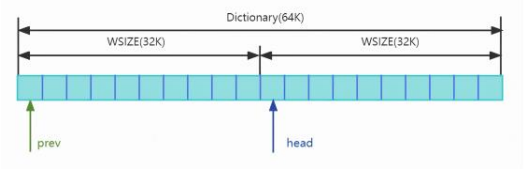

图 6 哈希表实现结构

prev 指向整个内存的起始位置，其中 head = prev + WSIZE . head 指向的数组用来保存通过哈希函数计算出的首字符的下标，prev 指向的数组专门来解决哈希冲突. 当发生哈希冲突时，首先将冲突位置原有 head 的值取出来，再将新的值插入到 head 中，以 head 中新值作为 prev 数组的下标，将原有的值放入 prev 数组，这样就将所有相同字符串链接起来. 当需要查找字符串时，通过 head 数组找到对应字符串的首字符下标，然后在 prev 数组中链式查找其他此字符串出现的位置.

Huffman 编码是对经过 LZ77 压缩后得到的<长度，距离>对数组进行进一步的压缩. 其编码理论依据是通过统计字符出现次数，对字符按照出现频率从小到大进行排序，构造 Huffman 二叉树，从树的叶子节点到根节点进行变长编码，出现频率多的字符分配较短编码，而出现频率少的字符分配较长编码，实现数据总体大小的减小，以达到压缩的目的.

4 实验

4.1 测试环境

本文的测试平台为: Google Pixels 4，搭载的 MPSoC 高通骁龙 855 的架构是 ARM big.LITTLE 架构的变体，测试所用的硬件及软件环境如下表所示:

表 1 实验测试环境

实验环境	内容
操作系统	Android 11
运行内存	6GB
CPU	高通 Kryo 485Gold@2841MHz, 基于 A76(一个)
	高通 Kryo 485 Gold@2419 MHz, 基于 A66(三个)
	高通 Kryo 485Silver@1785 MHz, 基于 A55(四个)
GPU	Adreno(TM)640

4.2 性能测试

本文将 Edgebreaker 算法放在一个超大核上运行，将 Gzip 放在一个大核上运行. 对于不同大小的输入(off 文件)进行测试，得到的结果如下表:

表 2 测试结果对比(压缩率)

文件大小(KB)	超大核压缩(KB)	流水线压缩(KB)	压缩率
10	3	3	30.00%
91	13	13	14.29%
547	62	62	11.33%
2337	142	142	6.08%

表 3 测试结果对比(耗时)

文件大小(KB)	超大核耗时(s)	流水线耗时(s)	加速比
10	0.013	0.011	1.18
91	0.062	0.050	1.24
547	0.183	0.150	1.22
2337	0.593	0.475	1.25

从表 2 中可以看出，采用优化后的压缩算法和原算法相比，压缩率相同，说明在优化过程中并未破坏原有算法的逻辑结构，从而保证了优化后压缩算法的正确性.

表 3 则对比了单核运行压缩算法和优化后基于 CPU 大小核异构方案的耗时情况. 从表中数据可以看出，优化后的算法平均快了 1.22 倍，但是由于输入的网格中，各种类型的三角形数量不同，从而导致优化算法的效果不尽相同.

4.3 能耗测试

我们使用每焦耳的吞吐量，即 ppw，来评估能源效率. 结合前面提到的功耗计算公式, ppw 的具体定义如下:

$$ppw = \frac{1}{ET} \tag{4}$$

其中，E 表示能耗，T 表示算法执行的总时间. ppw 既限制了算法的运行时间，也限制了算法执行时的能量消耗，因而 ppw 越高，说明算法的能耗越低，越节能.

以 91KB 的 off 文件为例，测试了四组单核压缩算法和大小核压缩算法执行过程中的能耗情况，如下表所示:

表 4 测试结果对比(ppw)

测试序列	时间(s)		能耗(J)		ppw		提升比率
	超大核	流水线	超大核	流水线	超大核	流水线	
1	0.062	0.050	0.066	0.067	244.38	298.51	1.22
2	0.061	0.050	0.094	0.098	174.40	212.77	1.22
3	0.062	0.051	0.095	0.100	169.78	196.08	1.15
4	0.063	0.052	0.097	0.102	163.64	188.54	1.15

从上表可以看出，大小核异构算法的 ppw 与单核情况下相比有所提升，但是提升效果差异较大. 前两次测试的提升较高，可能是由于刚开始测试时手机的电压与电流不稳定. 后续测试中，大小核异构算法的提升比例稳定在 15%左右.

5 结 语

本文提出了一种基于 CPU 的两阶段流水线大小核异构压缩方案，并与单核 CPU 的情况进行了对比，结果表明本文提出的方法的压缩性能平均提升了 22%.

参考文献(References):

[1] Maglo A, Lavoué G, Dupont F, et al. 3d mesh compression: Survey, comparisons, and emerging trends[J]. ACM Computing Surveys (CSUR), 2015, 47(3): 1-41.

[2] Garland M, Heckbert P S. Surface simplification using quadric error metrics[C]//Proceedings of the 24th annual conference on Computer graphics and interactive techniques. 1997: 209-216.

[3] Chou P H, Meng T H. Vertex data compression through vector quantization[J]. IEEE Transactions on Visualization and Computer Graphics, 2002, 8(4): 373-382.

[4] TaubinÝ G. Geometric signal processing on polygonal meshes[C]//Proceedings of EUROGRAPHICS. 2000.

[5] Alexa M, Behr J, Cohen-Or D, et al. Computing and rendering point set surfaces[J]. IEEE Transactions on visualization and computer graphics, 2003, 9(1): 3-15.

[6] Taubin G, Rossignac J. Geometric compression through topological surgery[J]. ACM Transactions on Graphics (TOG), 1998, 17(2): 84-115.

[7] Deering M. Geometry compression[C]//Proceedings of the 22nd annual conference on Computer graphics and interactive techniques. 1995: 13-20.

[8] Rossignac J. Edgebreaker: Connectivity compression for triangle meshes[J]. IEEE transactions on visualization and computer graphics, 1999, 5(1): 47-61.

[9] Khokhar A A, Prasanna V K, Shaaban M E, et al. Heterogeneous computing: Challenges and opportunities[J]. Computer, 1993, 26(6): 18-27.

[10] Mittal S, Vetter J S. A survey of CPU-GPU heterogeneous computing techniques[J]. ACM Computing Surveys (CSUR), 2015, 47(4): 1-35.

[11] Topcuoglu H, Hariri S, Wu M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE transactions on parallel and distributed systems, 2002, 13(3): 260-274.

[12] Maheswaran M, Braun T D, Siegel H J. Heterogeneous distributed computing[J]. Encyclopedia of electrical and electronics engineering, 1999, 8: 679-690.

[13] Ali S, Braun T D, Siegel H J, et al. Heterogeneous computing[J]. Encyclopedia of Distributed Computing. Kluwer Academic Publishers, Norwell, MA, 2002.

[14] Wang S, Ananthanarayanan G, Zeng Y, et al. High-throughput cnn inference on embedded arm big. little multicore processors[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 39(10): 2254-2267.

[15] Wang M, Ding S, Cao T, et al. Asymo: scalable and efficient deep-learning inference on asymmetric mobile cpus[C]//Proceedings of the 27th Annual International Conference on Mobile Computing and Networking. 2021: 215-228.

[16] Basireddy K R, Singh A K, Al-Hashimi B M, et al. AdaMD: Adaptive mapping and DVFS for energy-efficient heterogeneous multicores[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 39(10): 2206-2217.

[17] 王胜，徐文祥，尹志杰. 基于异构多核处理器的 DVFS 技术研究[J]. 通信技术, 2017, 50(3): 565-570.

[18] Touma C, Gotsman C. Triangle mesh compression[C]//Proceedings-Graphics Interface. Canadian Information Processing Society, 1998: 26-34.

[19] Rissanen J, Langdon G G. Arithmetic coding[J]. IBM Journal of research and development, 1979, 23(2): 149-162.

[20] Ouyang J, Luo H, Wang Z, et al. FPGA implementation of GZIP compression and decompression for IDC services[C]//2010 international conference on field-programmable technology. IEEE, 2010: 265-268.

[21] Rigler S. FPGA-based lossless data compression using GNU Zip[D]. University of Waterloo, 2007.