基于 GPU 的并行自适应笛卡儿网格建模方法与仿真应用研究综述

任丽欣[1,2], 刘树森[3], 郭煜中[3], 何小伟[3], 吴恩华[1,4)*]

[1)] (中国科学院软件研究所计算机科学国家重点实验室　北京　100190)
[2)] (中国科学院大学　北京　100049)
[3)] (中国科学院软件研究所人机交互技术与智能信息处理实验室　北京　100190)
[4)] (澳门大学科技学院　澳门　999078)
(weh@ios.ac.cn)

摘　要: 自适应笛卡儿网格因其自适应性、正交性和可扩展性, 在仿真领域扮演着至关重要的角色. 首先对自适应笛卡儿网格的理论基础进行全面的梳理, 包括网格分类、基于 GPU 的并行网格构建算法、网格离散化方案以及微分算子的模板设计等方面; 然后从静态模型和动态仿真 2 个维度, 对比分析不同类型网格在空间复杂度以及不同构建算法在时间复杂度上的表现; 在非平衡网格构建中, 基于 GPU 的最大并行策略展现出显著效率优势, 然而为了生成平衡网格, 后续必须执行的平衡细分过程计算开销较大, 耗时为非平衡网格构建过程的 173.46%~356.09%, 显著地增加了整体计算成本; 通过构建实际应用中的浅水波模拟、基于布尔运算的实体建模以及弹性体碰撞等多个实例, 充分展示了自适应笛卡儿网格的广泛适用性和巨大潜力; 最后对自适应笛卡儿网格技术的未来发展方向进行展望, 包括算法优化和多物理场耦合的研究, 以提高其效率和适用性; 同时指出潜在的研究路径和挑战, 如算法的准确性和简洁性.

关键词: 自适应笛卡儿网格; 网格建模; 物理仿真; GPU 并行
中图法分类号: TP391.41　　**DOI:** 10.3724/SP.J.1089.2024-00765

Research on GPU-Based Parallel Adaptive Cartesian Grid Modeling Methods and Simulation Applications

Ren Lixin[1,2], Liu Shusen[3], Guo Yuzhong[3], He Xiaowei[3], and Wu Enhua[1,4)*]

[1)] (*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing*　100190)
[2)] (*University of Chinese Academy of Sciences, Beijing*　100049)
[3)] (*Laboratory of Human-Computer Interaction, Institute of Software, Chinese Academy of Sciences, Beijing*　100190)
[4)] (*Faculty of Science and Technology, University of Macau, Macao*　999078)

Abstract: Adaptive Cartesian grids play a crucial role in simulation fields due to their adaptiveness, orthogonality, and scalability. This work first provides a comprehensive review of the theoretical foundation of adaptive Cartesian grids, including grid classification, GPU-based parallel grid construction algorithms, grid discretization schemes, and template design for differential operators. Then, from the perspectives of static models and dynamic simulations, it compares and analyzes the performance of different grid types in terms of spatial complexity and the time complexity of grid construction algorithms. In non-balanced grid generation, the GPU-based maximum parallel strategy demonstrates significant efficiency advantages. However, to generate a balanced grid, the subsequent refinement process incurs a substantial computational cost, with

收稿日期: 2024-12-24; 修回日期: 2025-06-30. **基金项目**: 国家自然科学基金(62332015, 62302490); 中国科学院软件研究所基础研究项目(ISCAS-JCMS-202403). **任丽欣**(1995—), 女, 博士研究生, 主要研究方向为计算机图形学、流体仿真; **刘树森**(1988—), 男, 博士, 主要研究方向为计算机图形学、物理仿真; **郭煜中**(1996—), 男, 硕士, 助理工程师, 主要研究方向为数字媒体艺术; **何小伟**(1985—), 男, 博士, 研究员, 博士生导师, 主要研究方向为计算机图形学、物理仿真; **吴恩华**(1947—), 男, 博士, 研究员, 博士生导师, CCF 会员, 论文通信作者, 主要研究方向为真实感图像合成、物理仿真、虚拟现实等.

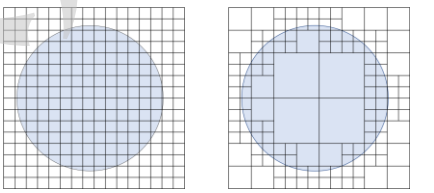
execution time reaching 173.46% to 356.09% that of the non-balanced grid generation process, significantly increasing the overall computational overhead. Furthermore, this work demonstrates the broad applicability and great potential of adaptive Cartesian grids through a series of practical examples, including shallow water wave simulation, Boolean operation-based solid modeling, and elastic collision simulations. Finally, this work outlines future directions for the development of adaptive Cartesian grid technology, including algorithmic optimization and research on multiphysics coupling to enhance its efficiency and applicability. Potential research paths and challenges are also discussed, such as improving algorithmic accuracy and simplicity.

Key words: adaptive Cartesian grid; grid modeling; physical simulation; GPU parallelism

1 概 述

基于物理的模拟仿真技术(简称为仿真)广泛应用于计算机辅助工程[1]、航空航天[2]、医学[3,4]、虚拟现实[5]、影视特效[6]等领域. 网格作为仿真计算的基础数据结构, 其构建质量与效率是保证仿真计算准确性与高效性的关键. 在仿真中, 网格的应用主要包含3方面: (1) 作为计算网格[7], 通过离散化方法将偏微分方程转化为可求解的代数方程组; (2) 作为辅助数据结构, 优化光线追踪等计算密集型任务[8]; (3) 作为仿真场景中物体几何建模的基础结构[9]. 为了在保证仿真精度的同时显著地降低仿真任务的计算量, 可动态地调整网格密度的自适应网格被广泛使用[10].

自适应笛卡儿网格是一种基于笛卡儿坐标系排列, 并根据需求动态调整局部网格分辨率的网格结构, 如图 1b 所示. 与图 1a 所示的均匀网格相比, 自适应笛卡儿网格通过局部加密捕捉流场的特征, 能够更灵活地适应流场中的变化[10]. 通常, 自适应笛卡儿网格的构建基于树结构, 基于 CPU 的算法从根节点出发, 逐个递归细分不满足约束的节点, 但其效率受限于串行执行模式. 随着 GPU 等并行计算硬件的飞速发展, 利用 GPU 大幅度提高自适应网格生成的效率, 成为近些年的研究热点之一[11]. 直观的 GPU 构建算法虽然可以在同一层次内并行生成所有节点, 但层次之间仍然保持串行[12]. 最大并行策略[13]的提出突破 GPU 算法对层次的依赖性, 实现了不同层级中间节点的并行构建. 然而, 当网格需满足更多约束(如相邻单元尺寸比例不得超过 2∶1)时, 必须对网格进行局部加密或粗化, 该过程可能引发非相邻单元的调整, 形成全局依赖关系. 在GPU并行架构中, 这类非局部依赖带来的全局内存访问需求与 SIMD (single instruction multiple data)特性产生冲突, 限制了算法的并行性和效率. 本文通过设计不同的实验, 对上述构建算法及过程进行深入对比和分析.



a. 均匀网格 b. 自适应笛卡儿网格

图 1 不同网格结构

本文对近年来自适应笛卡儿网格领域的研究进展进行系统地梳理, 从网格分类体系、构建方法与离散方案等维度对理论基础与方法体系进行讨论. 本文基于静态模型与动态仿真场景的对比分析, 揭示不同类型网格在空间效率上的差异, 以及各类构建算法的时间开销特征; 针对 GPU 技术对仿真效率的革新性影响, 重点探讨 GPU 加速的网格构建算法之间的对比, 及其与 CPU 方案相比在并行计算性能上的显著优势; 通过典型应用案例, 展示该技术在复杂几何建模与动态特征捕捉中的实践价值. 最后对自适应笛卡儿网格技术的未来发展方向进行展望, 并指出该领域可能的研究路径和面临的挑战. 本文对自适应笛卡儿网格相关工作的分类梳理及应用展示结构如图 2 所示.

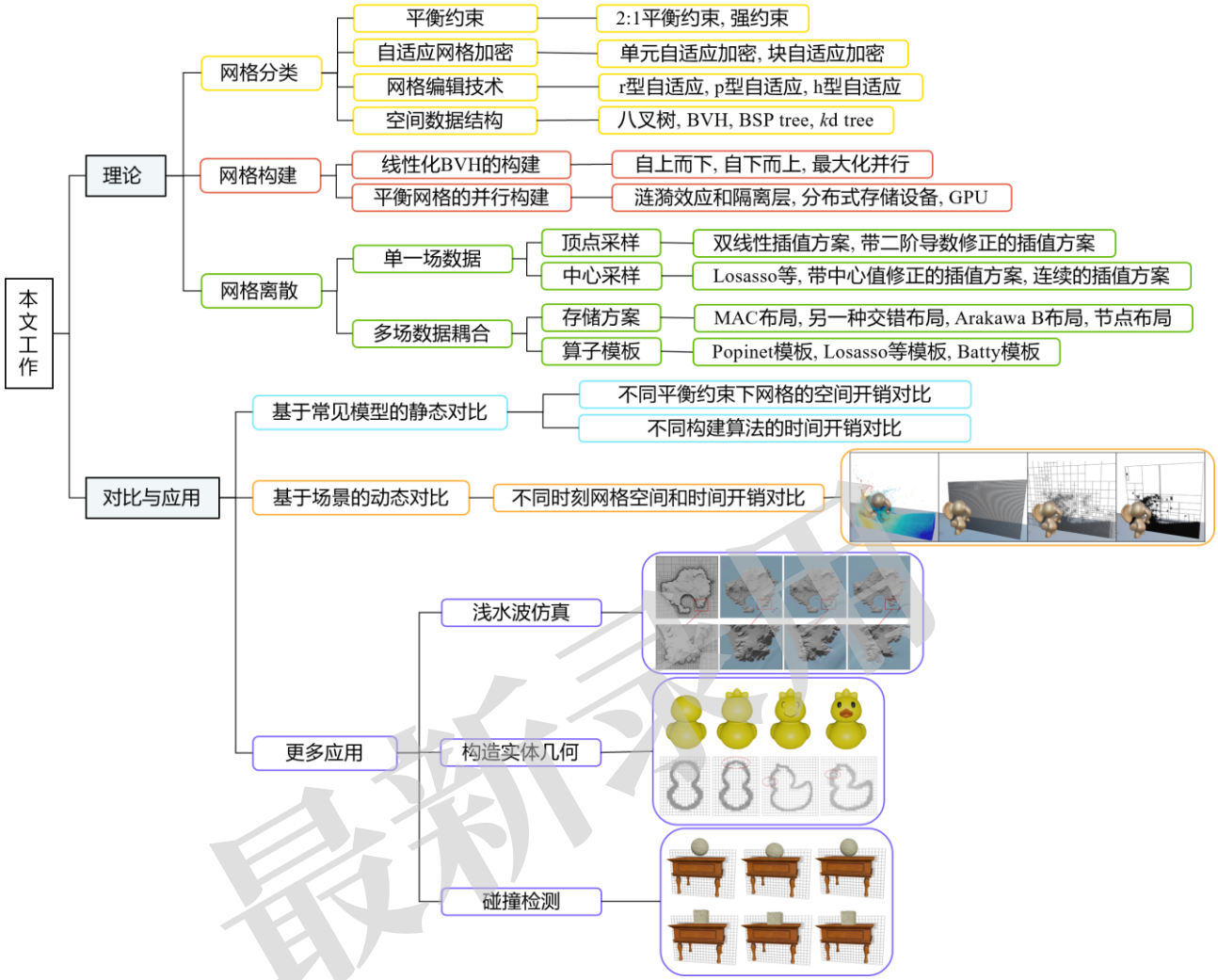


图 2　本文工作的分类梳理及应用展示

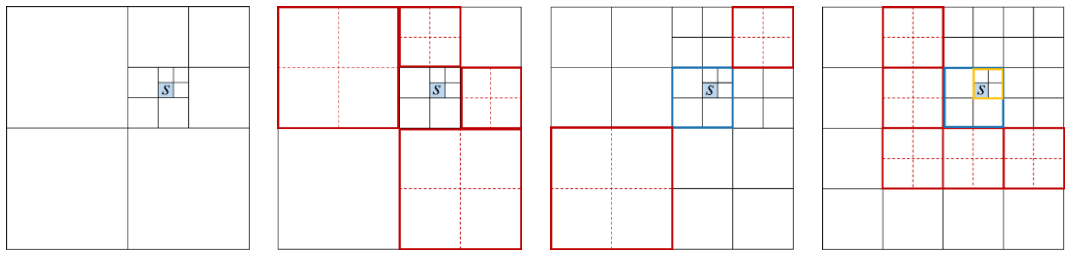
2　网格分类

自适应笛卡儿网格的应用广泛，不同的应用场景对网格的需求也不同. 例如，流体等材料的仿真要求网格铺满计算域且不重叠；考虑到离散求解的简洁性和准确性，有些应用会进一步要求计算网格满足平衡约束；在模型体素化应用中，体素网格只需要分布在沿模型边界的窄带区域[6,14]. 对网格的不同需求衍生出不同种类的自适应笛卡儿网格，下面结合应用对不同类型网格的实现方式及其特点进行讨论.

2.1　平衡约束

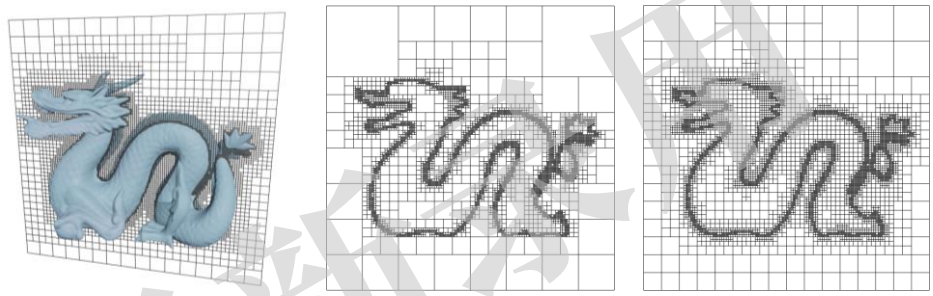
很多应用[7,15-16]要求自适应笛卡儿网格表示的计算域相对平滑，即相邻网格之间的尺寸没有急剧的大小变化，因此需要对网格施加 2∶1 平衡约束，该平衡约束要求相邻网格之间尺寸最多相差 2 倍，其中，“相邻”网格包括共享面、边和顶点的网格. 采用 Sundar 等[17]提出的按照顶点、边和面的维度进行分类，则分为 3 类: (1) 2-平衡网格. 网格中所有共享面的邻居满足 2∶1 平衡约束; (2) 1-平衡网格. 网格中所有共享面和边的邻居满足 2∶1 平衡约束; (3) 0-平衡网格. 网格中所有共享面、边和顶点的邻居满足 2∶1 平衡约束. 类似地，二维空间中，1-平衡网格要求所有共享边的邻居满足 2∶1 平衡约束, 0-平衡网格要求所有共享边和顶点的邻居满足 2∶1 平衡约束. 之后, Kim 等[18]提出一种略强于 2∶1 平衡条件的平衡约束，被称为强平衡约束，以简化基于网格中心的插值过程. 强平衡约束要求在基于八叉树的网格中，任一网格对应的叶节点的同一深度邻居是叶节点或其子节点是叶节点，这里的“邻居”包括所有共享面、边和顶点的网格. 二维自适应笛卡儿网格从非平衡约束到强平衡约束的过程如图 3 所示: 首先基于种子网格 s 构建初始非平衡网格(如图 3a 所示)，此时相邻网

格的尺寸没有任何约束；然后通过细分与共享边的邻居尺寸相差超过 2 倍的网格(如红框网格所示)，生成满足 1-平衡约束的网格(如图 3b 所示)；进一步，细分与共享顶点邻居不满足平衡条件的网格(如红框网格所示)，实现 0-平衡约束(如图 3c 所示)；为了满足强平衡约束(如图 3d 所示)，继续细分红框内的网格，因为这些网格的同一深度邻居(如蓝框所示)不是叶节点，且邻居节点的子节点(如黄框所示)也不是叶节点. 图 4 所示为基于三维 Dragon 模型构建的不同类型平衡网格的二维截面图.



a. 非平衡 b. 1-平衡 c. 0-平衡 d. 强平衡

图 3 不同平衡约束下二维自适应笛卡儿网格



a. Dragon 模型 b. 非平衡 c. 2-平衡

d. 1-平衡 e. 0-平衡 f. 强平衡

图 4 不同平衡约束下自适应笛卡儿网格截面图

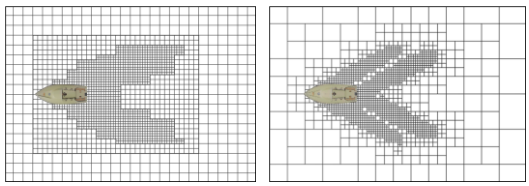
施加平衡约束的自适应笛卡儿网格称为平衡网格(也被称为等级网格)，该网格为方程的离散求解提供了一系列有利条件：简化了插值、微分、邻域查找以及许多其他操作；在更高精度要求的应用中，有助于实施更复杂的离散化方案，并减少不同细化级别区域边界处的数值不稳定性问题.

2.2 自适应网格加密

自适应网格加密(adaptive mesh refinement, AMR)[19]是一种动态调整网格密度的数值方法，旨在以最小计算成本实现高精度流场解析. 该方法最初由 Berger 等[20]提出，用于求解一维和二维双曲问题. 笛卡儿网格 AMR 可以分为块自适应与单元自适应 2 类. 块 AMR[21]通过局部细化规则的结构化网格块实现(如图 5a 所示)，核心是将待加密区域定义为规则的立方体块，采用多层嵌套网格结构既保留结构化网格的相邻关系简单、高阶算法兼容性等优势，又能显著地减少计算量并提升并行效率. 单元 AMR[2]则基于流场特征(如压力梯度、速度梯度或地形高度场)构造加密准则，对不满足阈值的单元递归细分(如图 5b 所示)，其优势在于直接针对物理量变化剧烈的单个单元操作，支持灵活且高分辨率的局部加密.

单元 AMR 通过制定不同的加密准则更灵活地追踪流场的细节. 陈祎等[22]在模拟车身周围复杂流场的过程中，将总压力场的梯度、散度运算结果作为加密准则；陈浩等[23]采用贴体网格和自适应笛卡儿网格重叠方法模拟三段翼绕流问题时，将重叠区网格尺寸比例系数作为网格加密准则，实

现背景笛卡儿网格的自适应加密. 块 AMR 具备的块结构化特点可以更好地与多重网格求解器等结合，在应用中结合实际的问题选择合适的方法. 肖中云等[21]采用重叠网格方法模拟旋翼尾涡，其中，背景笛卡儿网格采用块 AMR; Vanella 等[24]采用块 AMR 策略求解不可压层流和湍流，在不同空间分辨率的层次子网格进行单块求解，其中，每个子网格块都具有块结构化特征.



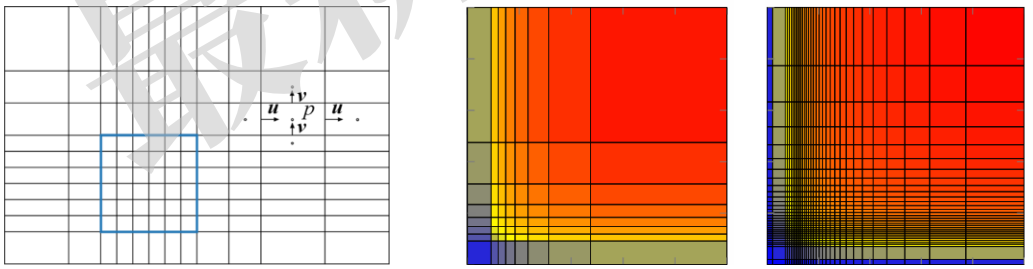
a. 块　　b. 单元

图 5　AMR

2.3　网格编辑技术

自适应笛卡儿网格构建过程中可以根据需要对网格进行编辑，常用的编辑方法包括 r 型和 h 型. h 型通过将现有单元拆分成更小的单元网格，或者将现有单元合并成更大的单元网格实现对网格的编辑，基于八叉树的自适应笛卡儿网格就是典型的 h 型自适应数据结构. r 型在不改变网格拓扑结构的情况下，通过移动网格边界满足网格的自适应需求. 实际应用中，还可以通过调整数值格式(如多项式次数[25])而不是编辑网格获得自适应的精度，被称为 p 型自适应. 由于 p 型自适应不涉及网格编辑，因此本文不进行详细介绍，相关研究可参考文献[25].

Zhu 等[26]采用 r 型自适应方法提出一种远场网格数据结构，如图 6a 所示，从一个统一笛卡儿网格构成的兴趣域出发(如蓝框区域所示)，在每个维度上动态地移动网格线以生成拉伸单元；拉伸单元被组织成不同的层次，每个层次上的单元边长是前一层次单元边长的 2 倍；为了高效地查找任意点所在的单元，提出一种将均匀网格单元对应的远场网格单元信息预先计算并存储在数组中，便于在模拟过程中快速访问的优化方法. Omella 等[27]提出一种深度神经网络(deep neural network, DNN)方法构建 r 型自适应网格，给定一个初始网格，该方法可以提供最优的节点位置，构建一个最优的 r 型自适应网格，并在该数据结构上求解偏微分方程. 针对椭圆问题，图 6b 和图 6c 所示为基于 8×8 和 32×32 的初始网格构建的 r 自适应网格，可以看出，在梯度变化剧烈的区域网格更精细.



a. 基于蓝框内的兴趣域构造的 r 型自适应网格[26]　b. 8×8 的 r 型自适应网格[27]　c. 32×32 的 r 型自适应网格[27]

图 6　r 型自适应网格

2.4　空间数据结构

自适应笛卡儿网格蕴含的空间划分思想经常被运用到加速优化中. 用于加速优化的空间数据结构包括八叉树(二维中是四叉树)、层次包围盒(bounding volume hierachy, BVH)结构，BSP tree (binary space partitioning tree)和 *k*d tree 等，它们通常作为加速辅助结构用于光线追踪[28]、碰撞检测[29]、体素化[30-31]和基于粒子的仿真[32]等场景. 八叉树的基本思想是每个节点的对应空间被分为 8 个相等的子空间. BVH 通常也是树状空间结构，其每个节点对应一个包围盒，常用的包围盒类型有轴对齐包围盒、有向包围盒和包围球等[33]. BSP tree 是一种二叉树，其每个节点表示一个有向超平面，将空间划分为 2 部分[34]. *k*d tree 也是一棵二叉树，其每个节点对应一个垂直于坐标轴的超平面，该超平面将空间分为 2 部分[35]. 实际上，*k*d tree 是轴对齐的 BSP tree.

空间数据结构的质量和构建速度之间的权衡取决于不同的应用. 例如，在光线追踪计算[8]中可能涉及数百万条光线与空间物体的相交计算，而高质量的空间数据结构为查询过程带来收益，因此 Goldsmith 等[36]提出的表面面积启发式(surface area heuristic, SAH)被认为是评价数据结构划分质量的标准方法；在物理仿真的宽阶段碰撞检测[37]中，由于需在每个时间步长重建空间数据结构，因此构建效率更重要. 还有一些工作努力在数据结构的质量和构建速度之间取得平衡. Zhou 等[38]在构建 *k*d tree 时，对处于上层的大节点使用中值分

割等低成本的方法确定分割平面，在靠近树底部的小节点上采用精确的方法确定分割平面；Garanzha 等[39]和 Pantaleoni 等[40]在构建 BVH 结构时，对根节点附近数量相对较少的节点使用高质量的划分算法，而对树的其余部分使用快速的划分算法.

3 网格构建

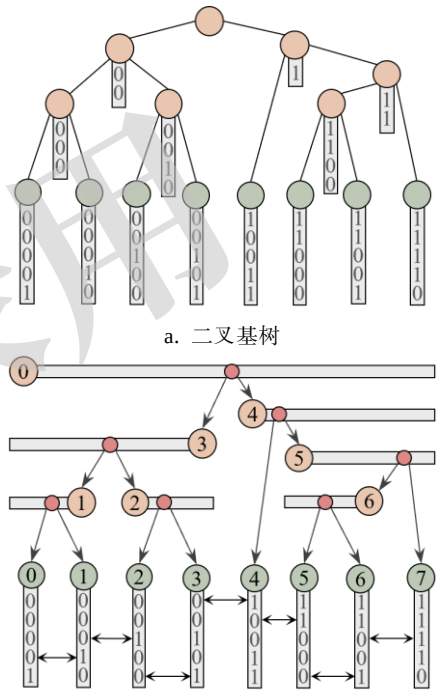
高效地构建自适应笛卡儿网格是计算网格或加速数据结构的关键. 近年来，针对百万级单元网格的实时生成需求，GPU 并行构建方法不断涌现. 首先梳理主流并行构建方法，然后重点探讨平衡约束引入对网格生成提出的新挑战与技术应对策略.

3.1 线性化 BVH 的并行构建

自适应笛卡儿网格通常基于树结构，常见的构建算法包括自上而下[38,41]和自下而上[12,42]这 2 类. 自上而下方法从根节点出发，递归地细分不满足约束的节点，直至达到终止条件或最大树深；自下而上方法则从底层节点逐层聚合，直至形成完整的树结构. 2 种策略的常规 GPU 并行实现均采用逐层生成方式，即在各层内部并行地处理节点；然而，这种实现方式存在层次间强依赖性：每层节点的生成必须等待其父节点或子节点处理完成，导致靠近根节点时大量线程闲置，显著地降低了并行资源利用率.

Lauterbach 等[41]首先提出线性 BVH，为每个基础单元分配一个莫顿码，按莫顿码对基础单元进行排序，将基础单元“线性化”为一维数组；然后按照莫顿码不同的比特位递归分裂，生成一个树结构，其中，每个子树都对应有序基础单元中的一段线性区间. 按照这一思想构建的树结构被称为二叉基树[13]. 二叉基树具有以下性质：(1) 没有链结构，具有 n 个叶节点的二叉基树包含 $(n-1)$ 个中间节点；(2) 叶节点有序且不重复，保证每个中间节点对应叶节点的线性区间 $[i,j]$，每个叶节点都是某一中间节点对应线性区间的左端点或者右端点；(3) 中间节点 $[i,j]$ 线性区间内的任意 $i',j'\in[i,j]$ 有 $\delta(i',j')\geqslant\delta(i,j)$，其中，$\delta$ 表示线性区间内叶节点的最长公共前缀的长度；(4) 中间节点 $[i,j]$ 根据 $\delta(i,j)$ 后第 1 个不同的比特位分裂为 2 个节点，分别对应 $[i,k]$ 和 $[k+1,j]$，其中，k 表示分裂位置且有如下性质 $\delta(k,k+1)=\delta(i,j)$，$\delta(i,k)>\delta(i,j)$ 和 $\delta(k+1,j)>\delta(i,j)$.

如图 7a 所示为一个具有 8 个叶节点的二叉基树，有 7 个中间节点. 二叉基树的根节点对应全部叶节点 $[0,7]$，且有 $\delta(0,7)=0$. 由于节点 3 和节点 4 的第 1 个比特位不同，因此根节点从此处分裂为 2 个孩子节点，分别对应 $[0,3]$ 和 $[4,7]$，且有 $\delta(3,4)=\delta(0,7)$，$\delta(0,3)=2>\delta(0,7)$ 和 $\delta(4,7)=1>\delta(0,7)$.



a. 二叉基树

b. a 的并行构建

图 7 二叉基树及其并行构建过程[13]

Karras 等[13]提出最大并行策略构造二叉基树. 图 7b 中，基于二叉基树的性质 2 可知，对于任意一个叶节点，只需要找到以该节点为端点的线性区间，就可以构建出该区间对应的中间节点. 基于二叉基树性质 1 分配 $(n-1)$ 个线程，每个线程从对应的叶节点开始，基于二叉基树性质 3 和二叉基树性质 4 执行以下操作：(1) 确定叶节点为线性区间的左端点还是右端点，如果是左端点则向右搜索，如果是右端点则向左搜索；(2) 继续搜索确定线性区间的另一端点，构建该线性区间对应的中间节点；(3) 查找该区间的分裂位置，找到该节点的孩子节点.

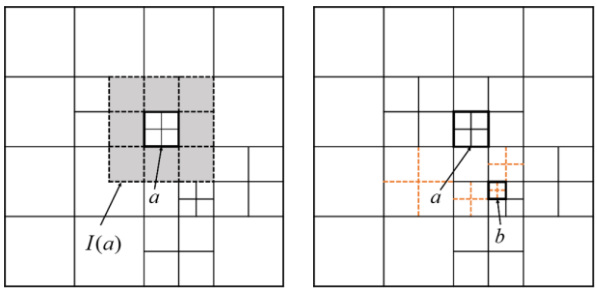
基于二叉基树，可以进一步构建 BVH、八叉树和 *k*d tree 等数据结构[13]. 对于八叉树，二叉基树中满足 $\lfloor\delta_{\text{child}}/3\rfloor-\lfloor\delta_{\text{parent}}/3\rfloor>0$ 的那些节点为八

叉树中的节点，只需要将这些节点保存并基于二叉基树构建出父子关系，即可完成八叉树的构建. Chen 等[29]提出一种基于GPU加速的自适应距离场构建方法，利用上述完全并行的二叉基树生成一个基于八叉树的自适应距离场，然后利用连续帧之间的相关性和对八叉树节点的排序加速距离查询；值得注意的是，这样构建的八叉树是有“洞”的，构建能够铺满整个空间的八叉树还需要额外的操作.

3.2　平衡网格的并行构建

基于树结构的自适应笛卡儿网格构建算法不能生成满足平衡约束的网格结构. 下面对平衡网格并行构建算法的相关研究进行总结和分析，包括基于分布式存储设备的方法[17,43-44]和基于 GPU 的方法[11].

平衡网格并行构建的难点是“涟漪效应”：一个节点的细分可能导致一系列节点的细分，即使这个节点不与细分的节点直接相邻. 所以，平衡细分本质上是一个迭代过程，这无疑为算法的并行实现带来了困难. Sundar 等[17]观察到，涟漪的影响是在有限的区域内. 对于任意节点 n，其同一层的所有潜在邻居节点组成该节点的隔离层 $I(n)$，其外任何节点的细分都不能迫使节点 n 细分；同理，节点 n 细分也不能迫使隔离层 $I(n)$ 外的节点细分. 图 8 中，虚线所示的节点组成了节点 a 的隔离层 $I(a)$，隔离层外节点 b 的细分不会对节点 a 产生影响.



a. 节点 a 的隔离层 $I(a)$　　b. 节点 b 细分

图 8　平衡网格的隔离层

Sundar 等[17]基于分布式存储设备提出 2 种算法，一个是从一系列点构建完整的线性八叉树，另一个是在完整线性八叉树上执行 2∶1 平衡约束. 在平衡约束问题中，基于上述观察提出两阶段均衡策略：第 1 阶段在各处理器内进行局部均衡；第 2 阶段平衡处理器间的边界，避免迭代通信. Isaac 等[44]提出一种通过判断任意 2 个给定的节点是否保持一致的距离或大小关系以减少通信开销的方法. 最近，Wang 等[11]指出，Sundar 等虽然在局部平衡过程中依赖哈希表减少节点的重复获得最佳性能，但是在具有大规模并行性的 GPU 硬件上缺乏高效且线程安全的哈希结构；提出 Sundar 等方法的GPU并行实现，其中利用GPU硬件的多种优化，如使用共享内存等，在不依赖任何哈希表结构的情况下实现了有效的树构建和平衡.

现有的平衡网格的并行构建算法普遍基于如下步骤实现：首先构建一个完整的线性八叉树，然后进行平衡细分. 其中，平衡细分过程中依然需要按顺序逐层实现，限制了算法在 GPU 上的并行效率.

4　网格离散

在流体仿真和水平集方法的应用中，首先要利用数值网格对计算域进行离散化处理. 在自适应笛卡儿网格的框架下，数据存储方式多样，包括网格中心、网格顶点或网格面等不同方案[18]，每种存储方案都对应特定的插值方法和算子模板，可以根据具体问题的性质和需求选择最合适的方法. 在水平集方法中，通常只需关注单一物理量的离散化和插值，如距离场[45]. 然而在流体仿真中，往往需要同时处理速度场、压力场等多个物理数据，此时必须考虑这些场数据之间的耦合效应. 下面详细探讨单一场数据和多场数据的存储方案、离散化方法和插值策略.

4.1　单一场数据

通常，单一物理场存储在网格中心或者网格顶点. 网格顶点采样可以简化插值和算子的离散过程[46]. 网格中心点采样无需记录网格到网格顶点的索引，避免额外的开销，而且中心采样有利于实现多分辨率网格之间的插值[18].

4.1.1　顶点采样

存储在网格顶点的数据可以自然地选择双线性插值方案(在三维中是三线性插值方案)[47]. 如图 9b 所示，对于二维网格 $C=[0,1]^2$ 中的任意一点 $(x,y)\in C$，采用双线性插值可以得到该点的值 $\phi(x,y)$ 为

$$\phi(x,y)=\phi(0,0)(1-x)(1-y)+\phi(0,1)(1-x)y+\phi(1,0)x(1-y)+\phi(1,1)xy;$$

其中，$\phi(x,y)$ 表示水平集函数在 (x,y) 点的值. 考虑到双性插值方案对取值不连续问题过于敏感，

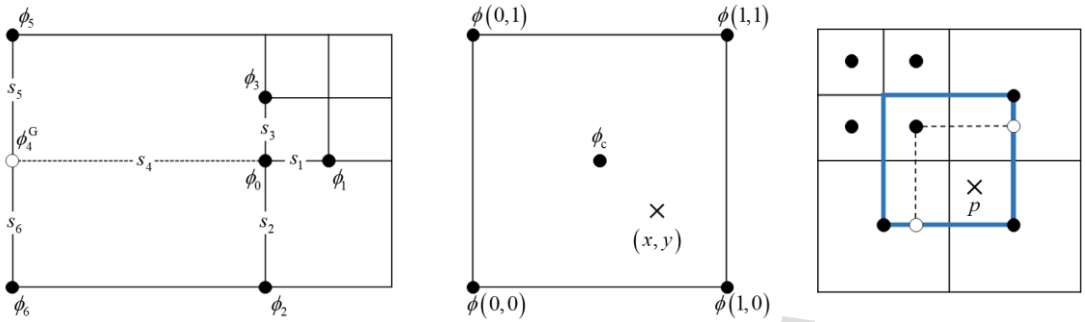
Min 等[48]提出通过二阶导数修正上述插值的方案

$$\phi(x,y)=\phi(0,0)(1-x)(1-y)+\phi(0,1)(1-x)y+\phi(1,0)(1-y)+\phi(1,1)xy-\phi_{xx}\frac{x(1-x)}{2}-\phi_{yy}\frac{y(1-y)}{2};$$

其中，

$$\phi_{xx}=\min_{v\in\text{vertices}(C)}\left(\left|D_{xx}\phi_v\right|\right)\ \phi_{yy}=\min_{v\in\text{vertices}(C)}\left(\left|D_{yy}\phi_v\right|\right).$$

由于通常存储的物理场(如距离场)都是分段可微的，因此选择绝对值最小的导数可以提高插值的数值稳定性.



a. 悬挂点　　b. 顶点采样　　c. 中心采样

图 9　单一场数据

在自适应笛卡儿网格结构中，构建算子模板的主要困难是悬挂点，这些点往往在某些方向缺少邻居. 解决该问题常使用的策略是为悬挂点构建虚拟的邻居节点[49]，如图 9a 所示，具有三阶精度的虚节点的值 ϕ_4^G 为[48]

$$\phi(x,y)=\phi_4^G=\frac{\phi_5 s_6+\phi_6 s_5}{s_5+s_6}-\frac{s_5 s_6}{s_2+s_3}\left(\frac{\phi_2-\phi_0}{s_2}+\frac{\phi_3-\phi_0}{s_3}\right).$$

有了虚节点后，可以采取与规则网格相同的方式处理悬挂点，根据标准中心有限差分格式，可以计算 X 方向的一阶导数和二阶导数为

$$\begin{cases}\mathrm{D}_x\phi_0=\dfrac{\phi_4^G-\phi_0}{s_4}\cdot\dfrac{s_1}{s_1+s_4}+\dfrac{\phi_0-\phi_1}{s_1}\cdot\dfrac{s_4}{s_1+s_4}\\ \mathrm{D}_{xx}\phi_0=\dfrac{\phi_4^G-\phi_0}{s_4}\cdot\dfrac{2}{s_1+s_4}-\dfrac{\phi_0-\phi_1}{s_1}\cdot\dfrac{2}{s_1+s_4}\end{cases}.$$

4.1.2　中心采样

当物理场存储在网格中心或者网格面中心时，Losasso 等[50]首先将值平均到网格顶点上，然后采用上述双线性或三线性方法进行插值，但这无疑会带来数值耗散[51]；针对此问题，Setaluri 等[51]提出了改进方案：在将网格中心或者网格面中心的值平均到网格顶点这一操作的基础上，对每个网格顶点上来自不同分辨率网格的值进行加权，其中，权重与网格尺寸成反比；然后根据网格中心点和网格顶点的值进行插值

$$\phi(x,y)=\phi(0,0)(1-x)(1-y)+\phi(0,1)(1-x)y+\phi(1,0)x(1-y)+\phi(1,1)xy+2\delta\phi_c\cdot\min\{x,1-x,y,1-y\};$$

其中，$\delta\phi_c=\phi_c-\phi_c^I$，$\phi_c$ 表示网格中心点的值，$\phi_c^I=\left(\phi(0,0)+\phi(0,1)+\phi(1,0)+\phi(1,1)\right)/4$. 当点 (x,y) 位于网格中心时，$\phi(x,y)=\phi_c$.

Kim 等[18]基于强平衡约束网格，对存储在网格中心的数据构建了连续的插值方案. 在插值过程中，先找到包含样本点的轴对齐插值盒，计算插值盒顶点的值；然后通过双线性或者三线性插值得到样本点的值. 如图 9c 所示，对点 p 进行插值时，首先将蓝色矩形按黑色虚线分割，通过比较点 p 和虚线的坐标确定包含点 p 的插值盒；然后在这个插值盒中进行双线性插值，其中，插值盒顶点的值是网格中心值的加权和. Kim 等[18]通过施加强平衡约束，将局部连通类型的数量限制在很小的范围内(四叉树为 5 种，八叉树为 21 种)，然后通过预先计算插值的权重提高插值的性能.

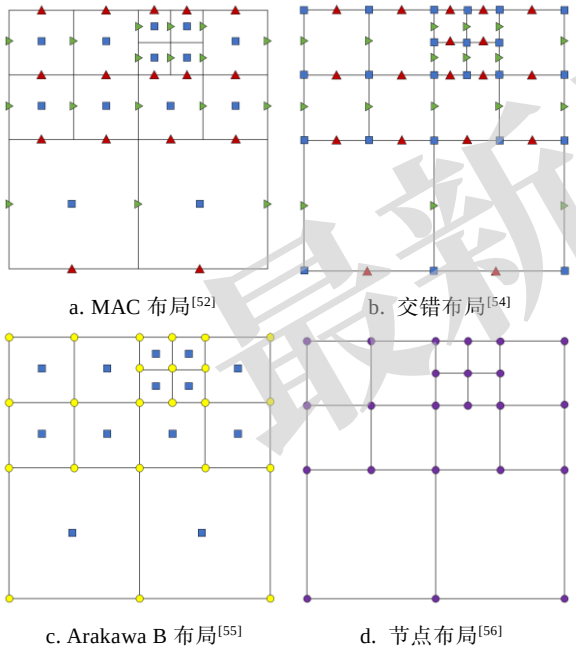
4.2　多场数据耦合

在不可压缩 Navier-Stokes 方程的数值求解中，多物理场离散常采用交错网格(marker and cell, MAC)方案. 该方法由 Harlow 等[52]于 1965 年提出，将压力场置于网格中心，速度场存储于网格面中心，这种空间解耦策略通过分离变量存储位置，有效地抑制了压力-速度与节点插值引发的振荡不稳定现象[53]，但需要布置大量标记点，计算量庞大. 近年来，研究人员提出了多种存储方案，同时，算子模板的优化进一步提升了计算效率和精度.

4.2.1　存储方案

图 10 所示为 4 种不同的存储方案，每种方案都有其特点和优势. 图 10a 所示为 MAC 布局，投影步骤通过速度场在无散度空间上的正交投影实

施无散度约束，算子的解析性质被保留，因此 MAC 方案一直被认为是流体仿真中求解不可压缩 NS 方程的理想选择. Gómez 等[54]提出一种交错布局，其是基于 0-平衡网格，压力存储在网格顶点，速度存储在网格面片，如图 10b 所示，该布局保证压力梯度和速度的自然对齐，简化有限差分算子的构造，提高了计算精度；并基于这种布局提出 3 种离散化方法，提供了不同级别的准确性和复杂性；如图 10c 所示，Arakawa 等[55]提出 Arakawa B 网格布局，速度场存储在网格顶点，压力场存储在网格中心，允许对流和扩散项使用紧凑的模板进行离散，并且具有一致的截断误差. Blomquist 等[56]提出将压力和速度均存储在网格顶点，如图 10d 所示，这种存储方案简单直观，不需要多种数据结构和求解器，而且可以使用紧凑模板实现二阶有限差分.



a. MAC 布局[52]　b. 交错布局[54]　c. Arakawa B 布局[55]　d. 节点布局[56]

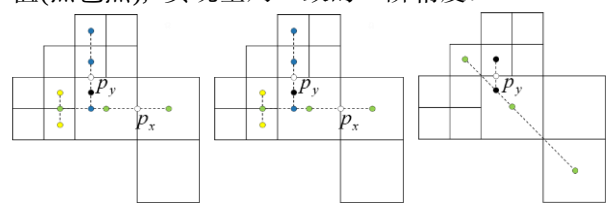
图 10　4 种速度压强耦合存储方案

4.2.2　算子模板

为了实现流体的不可压缩性，通常需要求解压力泊松方程

$$\nabla\cdot\frac{\Delta t}{\rho}\nabla p=\nabla\cdot u \tag{1}$$

其中，Δt 表示仿真时间步长，ρ 表示流体密度场，p 表示流体压强场，u 表示流体速度场. 通过离散式(1)中的梯度算子、散度算子和拉普拉斯算子可以将其转化为线性系统，求解出满足不可压缩条件的压力场. 与均匀网格相比，采用自适应笛卡儿网格离散求解会引入悬挂点，而在这些悬挂点处不能直接应用标准的中心有限差分/体积模板，因此如何保持数值解和梯度的二阶精度是一个挑战. Popinet[57]基于 0-平衡网格，通过插值构建缺失值，实现数值解的二阶精度. 如图 11a 所示，通过使用三点二次插值模板(绿色点和蓝色点)构建出缺失值(黑色点)，得到二阶精度的梯度值 p_y；但是，在插值过程中需要使用线性插值(如黄色点)恢复必要的数据来完成二次插值，线性插值的存在将计算梯度的精度在局部降低到一阶. 如图 11b 所示，Losasso 等[50]基于非平衡网格采用不准确的梯度估计保证矩阵系统的对称正定性，实现数值解的一阶精度；之后，Losasso 等[58]在细单元网格也使用粗单元网格的压力梯度，使无散度投影后的细单元网格与粗单元网格保持相同的速度，该方案在保证系数矩阵的对称正定性的同时实现了数值解的二阶精度. Lee 等[59]通过严格分析，证明了 Losasso 等提出的 2 种方法的数值解的收敛阶数与梯度的收敛阶数相差 1/2，而不是通常预期的 1，即证明了 Losasso 等提出的方法的有效性. 但是，Losasso 等提出的方法不能保证数值解的梯度具有二阶精度. 在各种应用中，保证梯度的准确性也是很重要的，如完成上述压力泊松方程的求解后，压力梯度被用来更新流体的速度场. Batty[60]提出一种基于非平衡网格的有限体积离散化方法，实现了数值解和梯度的二阶精度，如图 11c 所示，通过沿对角线采用三点二次插值(绿色点)构建出缺失值(黑色点)，实现全局一致的二阶精度.



a. Popinet[57]　b. Losasso 等[50,58]　c. Batty[60]

图 11　3 种算子模板

5　实验与应用

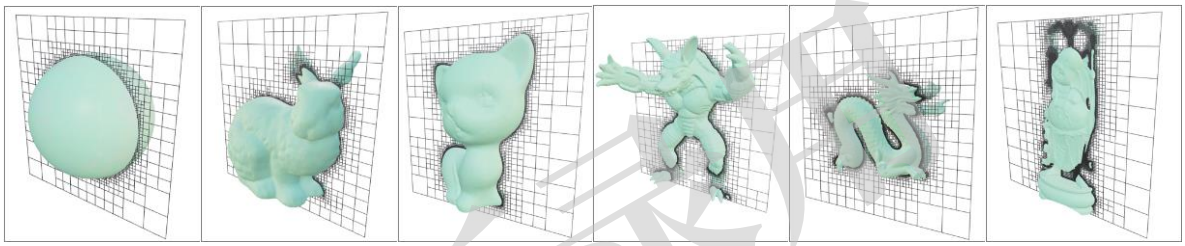
通过实验，对上文提到的网格类型以及基于 CPU 和 GPU 的网格构建算法进行对比和分析，包括静态对比和基于场景的动态对比. 本节中的算法均是基于开源的物理仿真引擎 PeriDyno[①]实现. 实验平台为 Intel Xeon W2245 CPU, 64 GB RAM,

① https://github.com/peridyno/peridyno

24 GB 内存的 NVIDIA GeForce RTX 3090 GPU 的 PC 上进行.

5.1 基于常见模型的对比

与均匀网格相比，自适应笛卡儿网格结构的优势是显著地降低计算的空间复杂度. 在不同平衡约束条件下，对图 12 所示的 6 种常见模型的自适应网格的单元数目，以及尺度相当的均匀网格的单元数目进行统计，结果如表 1 所示. 其中，树深表示基于树结构构建的自适应网格的层次数目. 可以看出，在尺度相当的情况下，均匀网格的空间开销是强平衡约束下自适应网格的 590%~3 314%，充分体现了自适应网格在空间利用效率方面的显著优势. 从图 4 和表 1 可以看出，随着约束条件的增强，网格的数目也随之增多. 图 13a 所示为网格数目随约束条件的变化趋势. 表 1 中还统计了平衡网格与非平衡网格的百分比，定义 R_0，R_1，R_2，R_3；其中，R_0 表示 0-平衡与非平衡网格的百分比，R_1 表示 1-平衡与非平衡网格的百分比，R_2 表示 2-平衡与非平衡网格的百分比，R_3 表示强平衡与非平衡网格的百分比. 另外，定义 R_4，表示均匀网格与强平衡网格的百分比. 可以看出，与非平衡网格相比，2-平衡约束使网格数目增加 11%~13%，1-平衡网格数目增加 17%~20%，0-平衡网格数目增加 19%~22%，强平衡网格数目增加 37%~45%.



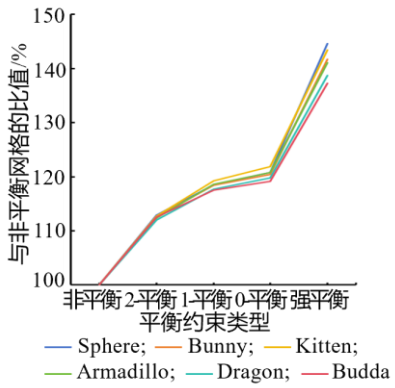
a. Sphere b. Bunny c. Kitten d. Armadillo e. Dragon f. Budda

图 12 6 种常见模型及其自适应笛卡儿网格截面

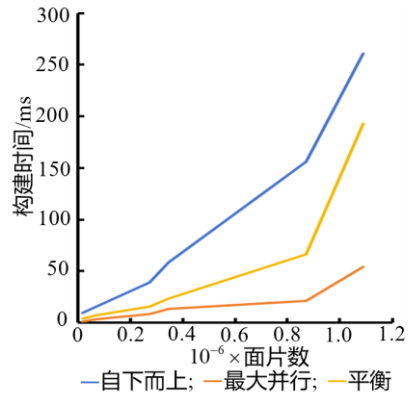
表 1 不同平衡约束下 6 种网格的空间开销对比

模型	树深	非平衡	2-平衡	R_2	1-平衡	R_1	0-平衡	R_0
Sphere	7	97 224	109 768	112.90	115 200	118.49	117 412	120.76
Bunny	8	386 086	434 869	112.64	457 087	118.39	464 836	120.40
Kitten	9	1 512 995	1 705 439	112.72	1 803 691	119.21	1 844 529	121.91
Armadillo	9	1 867 937	2 098 510	112.34	2 214 864	118.57	2 256 346	120.79
Dragon	10	4 573 353	5 121 236	111.98	5 382 210	117.69	5 480 014	119.82
Budda	10	5 691 883	6 404 028	112.51	6 687 101	117.48	6 781 048	119.14

模型	树深	非平衡	强平衡	R_3	均匀网格	R_4
Sphere	7	97 224	140 624	144.64	830 584	590.64
Bunny	8	386 086	547 156	141.72	6 487 677	1185.71
Kitten	9	1 512 995	2 169 553	143.39	38 175 216	1759.59
Armadillo	9	1 867 937	2 635 501	141.09	87 316 704	3313.10
Dragon	10	4 573 353	6 344 346	138.72	140 877 990	2220.53
Budda	10	5 691 883	7 810 475	137.22	150 959 012	1932.78



a. 6 个模型的空间开销对比



b. 3 种构建算法的时间开销对比

图 13 空间开销和时间开销对比

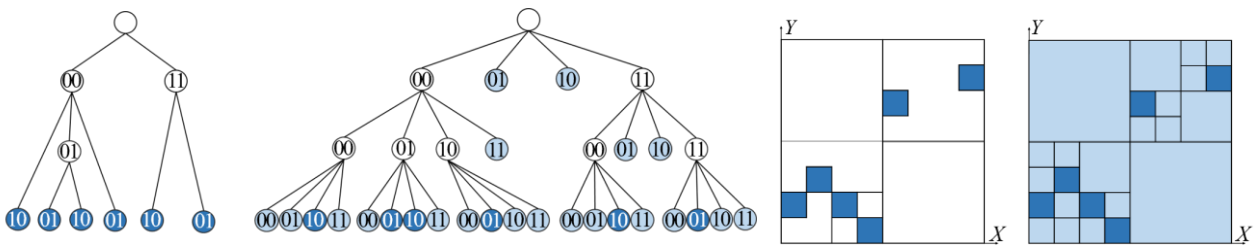
为了评估 CPU 与 GPU 架构下不同网格构建算

法的效率差异，本文实现了 3 种构建算法并统计其时间开销，结果如表 2 所示．其中，CPU 方法采用自上而下方法[41]递归细分网格；自下而上方法基于 Zhou 等方法[12]实现；最大并行方法基于 Karras 等方法[13]实现，其耗时包括二叉基树的构建时间 T_1 和补足缺失网格的时间 T_2，2 个阶段的总耗时为 $T_{\text{all}}=T_1+T_2$；平衡方法则基于 Teunissen 等方法[61]实现，逐层细分不满足平衡约束的节点．同时，定义 R_5，R_6，R_7；R_5 表示自上而下方法与 CPU 方法的百分比，R_6 表示平衡方法与最大并行方法 T_{all} 的百分比，R_7 表示最大并行方法 T_{all} 与自下而上方法的百分比．特别地，由于 Karras 方法生成的基于八叉树(二维中是四叉树)的自适应网格存在孔洞，本文在原始方法的基础上增加了网格补全模块，确保不同策略生成网格的完整性一致．图 14 所示为补足缺失网格前后的网格结构．图 14a 和图 14c 中，基于 Karras 等方法[13]构建的四叉树的叶节点均为输入的种子点(深蓝色)，该结构通常作为辅助数据结构[62]；图 14b 和图 14d 中，首先为每个节点生成其兄弟节点，然后去除重复的节点，即可补足缺失的叶节点(浅蓝色)，自适应网格结构可以铺满整个计算域，该结构通常作为计算网格用于求解偏微分方程[51]．对于平衡约束，本文以 1-平衡为例，对该条件下逐层实现平衡细化所需的时间开销进行统计，图 13b 所示为随着模型越来越复杂，不同构建方法时间开销的变化趋势．从表 2 和图 13b 可以看出，与基于 CPU 的方法相比，自下而上方法的时间开销降低到 2.45%~7.79%；与自下而上方法相比，最大并行方法的时间开销进一步降低到 13.80%~23.18%．构建平衡网格的过程中，需要在使用最大并行方法生成非平衡网格之后，进一步执行额外的平衡细分操作，该平衡细分阶段的耗时为采用最大并行方法生成非平衡网格过程的 173.46%~356.09%．

表 2　4 种构建算法的时间开销对比

模型	顶点数	面片数	树深	CPU 方法[41]/ms	自下而上方法[12]/ms	R_5	平衡方法[61]/ms	R_6
Sphere	9 002	18 000	7	125.00	9.74	7.79	4.07	292.81
Bunny	34 834	69 664	8	453.00	15.88	3.51	7.31	216.27
Kitten	137 098	274 196	9	1 438.00	39.25	2.73	15.46	178.11
Armadillo	172 974	345 944	9	2 250.00	58.68	2.61	23.59	173.46
Dragon	434 856	869 928	10	6 375.00	155.99	2.45	66.44	308.74
Budda	543 652	1 087 716	10	6 734.00	261.02	3.88	193.00	356.09

模型	顶点数	面片数	树深	最大并行方法[13]/ms			R_7
				T_1	T_2	T_{all}	
Sphere	9 002	18 000	7	0.79	0.60	1.39	14.27
Bunny	34 834	69 664	8	2.04	1.34	3.38	21.28
Kitten	137 098	274 196	9	6.64	2.04	8.68	22.11
Armadillo	172 974	345 944	9	8.93	4.67	13.60	23.18
Dragon	434 856	869 928	10	15.67	5.85	21.52	13.80
Budda	543 652	1 087 716	10	35.72	18.48	54.20	20.76



a. 四叉树　b. 补足缺失网格后的四叉树　c. a 的网格　d. b 的网格

图 14　补足 Karras 等方法[13]构建的网格结构

5.2　基于场景的对比

以光滑粒子流体动力学(smoothed particle hydrodynamics，SPH)方法为代表的无网格粒子方法，是实现流体、弹塑性固体等材料的重要且基础的方法之一[63-64]，具有能量耗散小[65]、易于划分边界[66-68]、易于保证体积与质量守恒[69]、易于处理材

料剧烈形变和碎裂[70-73]等诸多优势. 尽管无网格粒子法无需处理材料的拓扑网格, 但其邻域查找计算(即查找每个粒子的邻近粒子)的过程中仍然会使用网格结构以保证计算效率[74]. 图 15 所示为借助笛卡儿网格实现粒子邻域查找的过程. 可以看出, 有了网格结构后, 只需查找每个粒子邻近网格内的粒子, 无需遍历仿真域内全部的粒子.

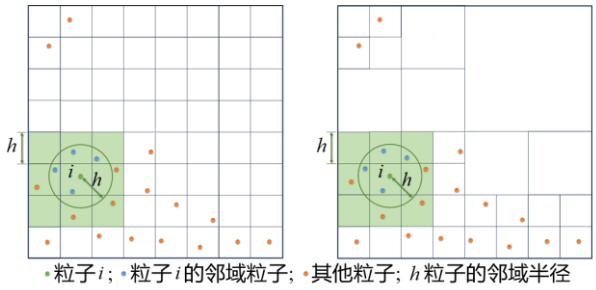
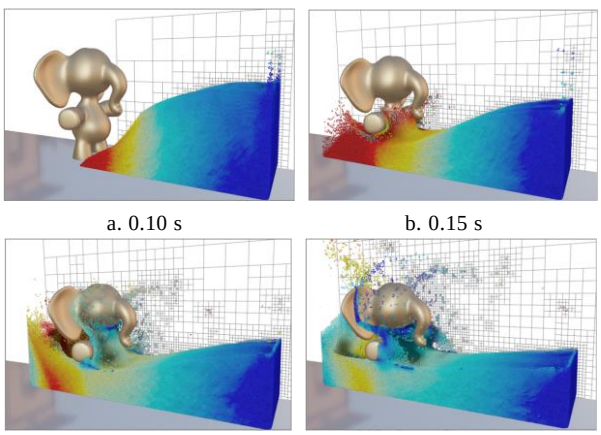


图 15 借助笛卡儿网格实现粒子的邻域查找

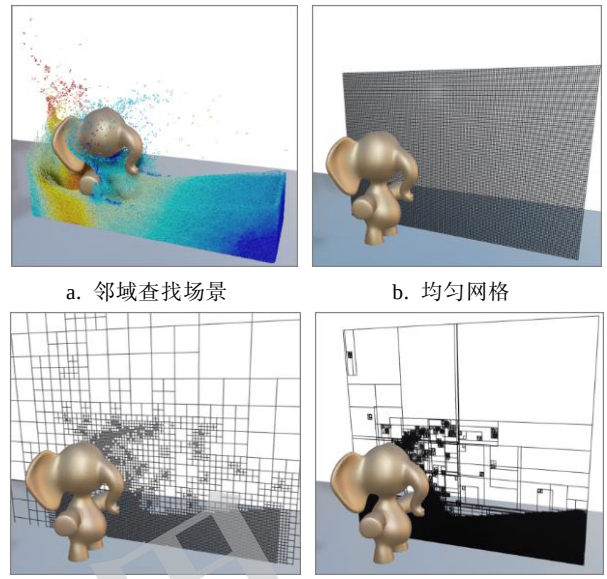
图 16 所示为基于 SPH 方法[75]搭建的流体仿真测试场景中不同时刻粒子的飞溅情况, 其中, 粒子数目为 826 281; 该场景可采用不同的数据结构实现邻域查找, 包括均匀网格、八叉树和 BVH 等, 如图 17 所示; 图 18 所示为该场景下, 不同时刻均匀网格和自适应网格的空间开销和时间开销, 图 18a 中, 自适应网格结构选取的是八叉树, 图 18b 中, 自适应网格结构选取的是 BVH. 从图 16 和图 18a 可以看出, 随着仿真的进行, 粒子飞溅的范围逐渐扩大, 涉及的计算域范围也相应增大, 均匀网格的数目急剧增加, 但自适应网格的数目只呈现出轻微增长; 从图 18b 可以看出, 自适应结构需要花费更多的时间构建. 综上所述, 均匀网格和自适应网格各有优势, 在小空间范围内, 可以使用均匀网格计算求解, 而在较大问题域中进行流体仿真计算时, 使用自适应数据结构能够显著地降低计算的空间开销.



a. 0.10 s b. 0.15 s

c. 0.20 s d. 0.25 s

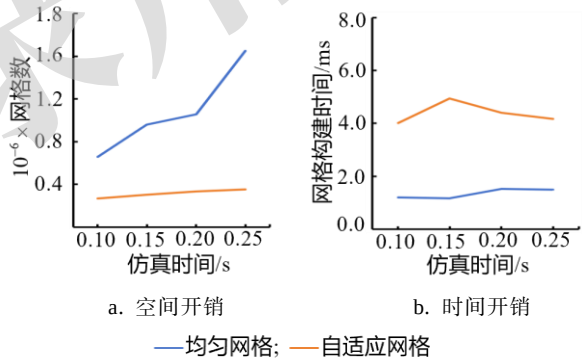
图 16 不同时刻粒子的飞溅情况



a. 邻域查找场景 b. 均匀网格

c. 八叉树 d. BVH

图 17 用于邻域查找的不同数据结构的截面图



a. 空间开销 b. 时间开销

均匀网格; 自适应网格

图 18 基于测试场景的空间开销和时间开销

5.3 更多应用

5.3.1 浅水波仿真

在欧拉视角方法的流体仿真中, 流场通常被离散到笛卡儿网格中, 并通过合适的方法(如 LBM、投影法等)求解对应的方程(如 NS 方程、浅水波方程等), 以更新网格中的流场. 按照网格的类型, 可以将这些方法分为 3 类: (1) 基于单一自适应网格结构. 这种情况下需要解决悬空节点的处理[15]; (2) 基于重叠网格结构. 这种情况下通常采用贴体网格追踪物体表面的流场细节, 采用自适应笛卡儿网格作为背景网格求解[76]; (3) 自适应网格与其他方法结合. 如网格与粒子结合的代表性方法 PIC(particle in cell)[77]和 MPM(material point method)[78]等.

本文以求解浅水波方程为例, 展示基于自适应笛卡儿网格的海洋仿真的流程和效果, 如图 19 所示. 浅水波方程是描述浅水环境中流体运动的方程, 被广泛应用于台风、海啸、溃坝等现象的模

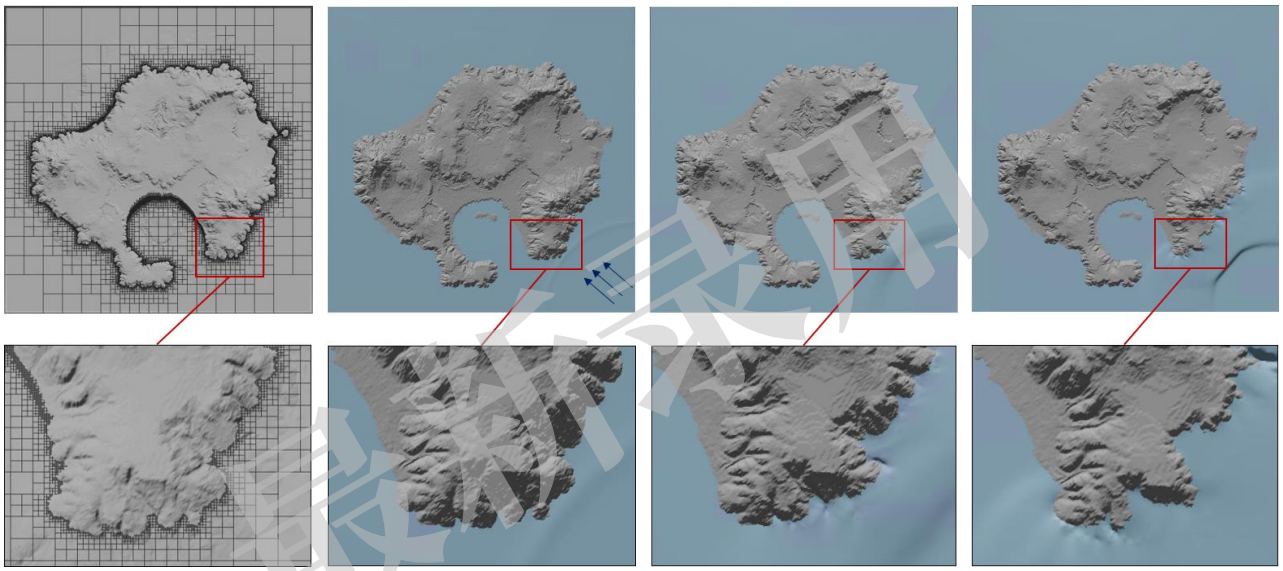
拟和预测．在不考虑摩擦力和黏性的情况下，浅水波方程[79]可以简化为

$$\frac{\partial h}{\partial t}+\frac{\partial}{\partial x}(hu)+\frac{\partial}{\partial y}(hv)=0;$$

$$\frac{\partial}{\partial t}(hu)+\frac{\partial}{\partial x}\left(\frac{hu^2}{2}\right)+\frac{\partial}{\partial y}(huv)=-gh\frac{\partial s}{\partial x};$$

$$\frac{\partial}{\partial t}(hv)+\frac{\partial}{\partial x}(huv)+\frac{\partial}{\partial y}\left(\frac{hv^2}{2}\right)=-gh\frac{\partial s}{\partial y}.$$

其中，h 表示水高；u,v 分别表示 X 和 Y 方向的水流速度；s 表示水面垂直坐标，即坡底高程 d 和水高 h 之和 $s=h+d$．如图 19a 所示，计算域被离散为 1-平衡网格，海岛陆地与水面的交界区域为精细网格，远离交界区域为粗糙网格．为了保证求解的稳定性和准确性，本文采用二阶中心迎风格式[80]对浅水波方程进行离散求解．图 19b~图 19c 所示为通过施加沿箭头方向的风力，使得海水涌向海岛，浸没海岛的部分海岸的过程；图 19c~图 19d 所示为撤销风力后海水回流，海面波浪扩散的过程．



a. 自适应笛卡儿网格　b. 施加风力　c. 海水浸没海岸　d. 撤销风力

图 19　浅水波方程模拟风力影响下海岛附近海水的运动

5.3.2　构造实体几何

构造实体几何(constructive solid geometry, CSG)通过布尔运算对几何对象进行组合，构造复杂的实体模型，为物理仿真提供几何基础．基于均匀体网格存储的有向距离场(signed distance field, SDF)进行布尔运算具有鲁棒性和高效性等优点，可以有效支持 CSG 操作．然而，基于均匀网格的 SDF 通常占用大量的存储空间，在一定程度上限制了其应用范围；相比之下，自适应笛卡儿网格凭借其灵活的分辨率调整能力，能够显著优化存储效率，无疑是更加理想的选择．下面通过具体样例，展示基于自适应笛卡儿网格存储的 SDF 进行 CSG 操作，构造复杂模型的全过程．

基于自适应笛卡儿网格，本文采用快速迭代方法(fast iterative method, FIM)[81]求解程函方程(Eikonal equation)[82]，得到全局的 SDF

$$|\nabla\phi(x)|=1;$$

其中，$\phi(x)$ 为有向距离函数，表示网格中心点到计算域的距离，其在计算域外为正值，计算域内为负值，边界上为 0．然后按照解析定义[83]对 SDF 进行布尔运算:

(1) 并集

$$\phi_3=\phi_1|\phi_2=\min(\phi_1,\phi_2);$$

(2) 交集

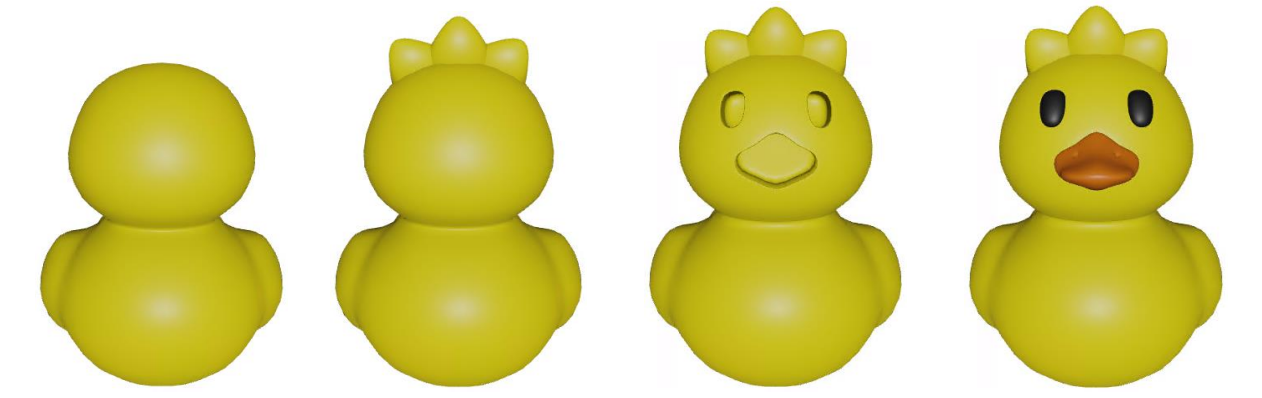
$$\phi_3=\phi_1\&\phi_2=\max(\phi_1,\phi_2);$$

(3) 差集

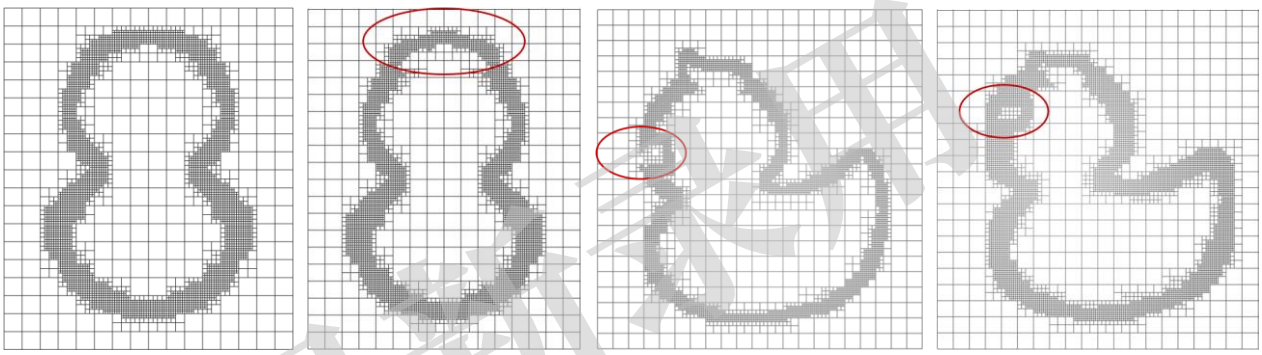
$$\phi_3=\phi_1\setminus\phi_2=\phi_1\&(-\phi_2).$$

最后采用经典的等值面提取(marching cubes, MC)算法[84]重建模型表面．图 20 所示为基于自适应笛卡儿网格取并集和差集，构建完整的小黄鸭模型的过程．从图 20a 主体模型出发，通过取并集可以为小黄鸭添加头发(如图 20b 所示)；然后通过差集操作构造出放置眼睛和嘴巴的凹槽(如图 20c 所示)；最后将眼睛和嘴巴添加到对应的凹槽里，即

可得到最终的小黄鸭模型(如图 20d 所示); 图 20e~图 20h 所示为并集和差集操作为自适应网格结构带来的变化.



a. 小黄鸭主体 b. 取并集后的模型 c. 取差集后的模型 d. 小黄鸭的完整模型

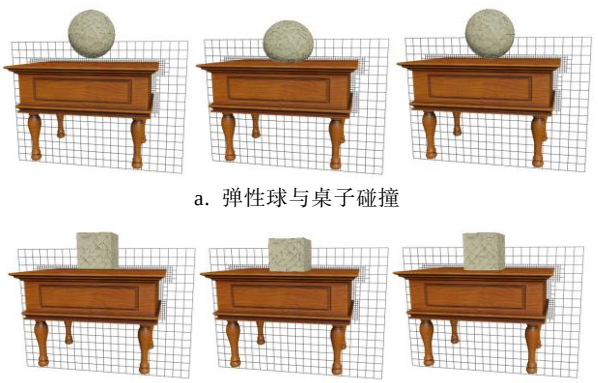


e. a 的自适应网格 f. b 的自适应网格 g. c 的自适应网格(嘴巴) h. d 的自适应网格(眼睛)

图 20 基于自适应笛卡儿网格存储的 SDF 进行布尔运算

5.3.3 碰撞检测

SDF 作为一种形状表示方法, 具有查询高效、结果稳定等优势[85], 广泛应用于碰撞检测. 与基于三角形对的最近距离计算或连续碰撞检测方法相比[86], SDF 能够提供平滑变化的穿透距离和碰撞方向, 适用于大多数碰撞处理算法; 此外, SDF 还能够提供可靠且稳定的内部/外部信息, 对于弹性体仿真中的穿透现象具有针对性的效果. 在弹性体仿真中, 碰撞通常发生在物体表面, 且对尖锐网格特征较为敏感, 因此, 基于均匀网格的 SDF 常面临昂贵的存储开销和粗糙的表面精度之间的权衡问题. 由于自适应笛卡儿网格使用精细网格表示尖锐特征, 粗糙网格描述物体内部, 因此在保持 SDF 优势的前提下能够有效地解决这一问题. 在图 21 所示的场景中, 本文使用四面体网格表示弹性体, 并采用自适应网格存储 SDF 表示刚性边界, 模拟二者的碰撞过程. 可以看出, 通过自适应网格的点查询操作能够直接获得弹性体各顶点的穿透信息, 进行碰撞处理. 图 21a 所示为弹性球体与桌子碰撞前、碰撞中和碰撞后的连续状态, 图 21b 所示为弹性立方体碰撞前、碰撞中和碰撞后的状态.



a. 弹性球与桌子碰撞

b. 弹性立方体与桌子碰撞

图 21 基于自适应笛卡儿网格存储 SDF 进行碰撞检测

5.4 总结与讨论

本节对各类笛卡儿网格的空间开销及构建算法效率进行了系统的对比. 与均匀网格相比, 自适应笛卡儿网格通过局部加密显著地降低存储开销, 尤其在百万级单元规模下, 其存储量可缩减至均匀网格的 1/5~1/34. 平衡约束对网格性能的影响呈现双刃剑特性, 约束的增加可简化相关方程的离散求解, 这种约束会引入额外的空间开销, 且约束强度与存储需求呈正相关, 需在算法效率与资源消耗间权衡取舍. 例如, 基于网格的仿真中, 有限

体积法常采用 2-平衡网格平衡通量计算与存储效率；而有限元法则偏好 1-平衡网格以适应复杂几何形态；0-平衡与强平衡网格因存储成本过高，实际应用较为有限[44].

在网格构建算法方面，与传统 CPU 方案相比，基于 GPU 的方法具有显著效率优势. 其中，最大并行策略通过深度挖掘 GPU 并行性，在构建非平衡计算网格或加速空间数据结构时表现突出；经典自下而上算法虽然具备结构稳定性，但由于其对层次的依赖性难以充分利用 GPU 并行计算资源，因此在处理大规模网格生成任务时效率相对低下；平衡网格构建需执行额外的平衡细分过程，约束平衡步骤成为主要耗时瓶颈，凸显出平衡网格生成算法仍存在优化空间.

基于自适应笛卡儿网格的浅水波模拟、实体建模与碰撞检测研究，凸显了该网格体系在兼顾计算精度与资源效率间的核心优势，验证了其在流体力学、工程仿真与交互计算等领域的应用潜力.

6　结　语

本文对自适应笛卡儿网格在网格分类、构建和离散等方面的最新研究工作进行综述，并通过对浅水波仿真、实体建模和弹性碰撞等应用场景的搭建，展示自适应笛卡儿网格的广泛应用. 基于当前的研究进展和本文的深入分析，未来，自适应笛卡儿网格的研究方向可能包括以下方面：(1) 当前，构建满足平衡约束的自适应笛卡儿网格通常采用先构建无约束网格，再进行平衡细化来满足约束条件的方法，平衡细化过程采用自底向上逐层处理. 基于 GPU 实现时，小负载的串行处理环节往往成为限制算法效率的瓶颈，突破此效率限制，实现带约束自适应笛卡儿网格的完全并行构建，将是研究的重点之一. (2) 基于自适应笛卡儿网格的离散插值方案需要在准确性和简洁性之间找到平衡，尤其是在处理多场数据耦合时，为了提高准确性，可能需要使用对角线邻居节点甚至不相邻节点的离散算子模板，而仅涉及简单邻居的模板则可能牺牲部分准确性. 因此，探索能够同时确保准确性和简洁性的离散方案，也将是未来研究的重点之一.

致谢　感谢罗旭锟同学在搭建应用场景时提供的支持和帮助！

参考文献(References):

[1] Cao Yi, Wang Huawei, Xia Fang, *et al*. Interactive visual analysis engine for high-performance CAE simulations[J]. Journal of Computer-Aided Design & Computer Graphics, 2021, 33(12): 1803-1810(in Chinese)
(曹轶, 王华维, 夏芳, 等. 面向高性能工业仿真的交互可视分析引擎[J]. 计算机辅助设计与图形学学报, 2021, 33(12): 1803-1810)

[2] Chen Hao, Hua Ruhao, Yuan Xianxu, *et al*. Simulation of flow around fly-wing configuration based on adaptive Cartesian grid[J]. Acta Aeronautica et Astronautica Sinica, 2022, 43(8): Article No.125674(in Chinese)
(陈浩, 华如豪, 袁先旭, 等. 基于自适应笛卡尔网格的飞翼布局流动模拟[J]. 航空学报, 2022, 43(8): Article No.125674)

[3] Ren L X, Wan S, Wei Y, *et al*. Towards a non-invasive diagnosis of portal hypertension based on an Eulerian CFD model with diffuse boundary conditions[C] //Proceedings of the 24th International Conference on Medical Image Computing and Computer Assisted Intervention. Heidelberg: Springer, 2021: 107-116

[4] Qi X L, An W M, Liu F Q, *et al*. Virtual hepatic venous pressure gradient with CT angiography (CHESS 1601): a prospective multicenter study for the noninvasive diagnosis of portal hypertension[J]. Radiology, 2019, 290(2): 370-377

[5] Shi Jiajun, Li Chen, Wang Changbo. Small-scale condensation simulation on complicated surface[J]. Journal of Computer-Aided Design & Computer Graphics, 2021, 33(4): 601-608(in Chinese)
(施佳俊, 李晨, 王长波. 复杂表面小尺度凝结现象真实感仿真[J]. 计算机辅助设计与图形学学报, 2021, 33(4): 601-608)

[6] Museth K. VDB: high-resolution sparse volumes with dynamic topology[J]. ACM Transactions on Graphics, 2013, 32(3): Article No.27

[7] Ando R, Batty C. A practical octree liquid simulator with adaptive surface resolution[J]. ACM Transactions on Graphics, 2020, 39(4): Article No.32

[8] Meister D, Ogaki S, Benthin C, *et al*. A survey on bounding volume hierarchies for ray tracing[J]. Computer Graphics Forum, 2021, 40(2): 683-712

[9] Museth K, Breen D E, Whitaker R T, *et al*. Level set surface editing operators[J]. ACM Transactions on Graphics, 2002, 21(3): 330-338

[10] Tang Jing, Cui Pengcheng, Zhang Jian, *et al*. Review of mesh adaptation for fluid numerical simulation[J]. Advances in Mechanics, 2023, 53(3): 661-692(in Chinese)
(唐静, 崔鹏程, 张健, 等. 流体数值模拟网格自适应技术研究进展[J]. 力学进展, 2023, 53(3): 661-692)

[11] Wang L, Witherden F, Jameson A. An efficient GPU-based h-adaptation framework via linear trees for the flux reconstruction method[J]. Journal of Computational Physics, 2024, 502: Article No.112823

[12] Zhou K, Gong M M, Huang X, *et al*. Data-parallel octrees for surface reconstruction[J]. IEEE Transactions on Visualization and Computer Graphics, 2011, 17(5): 669-681

[13] Karras T. Maximizing parallelism in the construction of BVHs, octrees, and *k*-d trees[C] //Proceedings of the 4th ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics. Aire-la-Ville: Eurographics Association Press, 2012: 33-37

[14] Kim D, Lee M, Museth K. NeuralVDB: high-resolution sparse volume representation using hierarchical neural networks[J]. ACM Transactions on Graphics, 2024, 43(2): Article No.20

[15] Goldade R, Wang Y P, Aanjaneya M, *et al*. An adaptive variational finite difference framework for efficient symmetric octree viscosity[J]. ACM Transactions on Graphics, 2019, 38(4): Article No.94

[16] Yu Wangyang, Ma Jianming, Yin Yueming, *et al*. 2D hydrodynamic model based on adaptive grid[J]. China Flood & Drought Management, 2022, 32(3): 66-72(in Chinese)
(于汪洋, 马建明, 尹岳明, 等. 基于自适应网格的二维水动

力模型[J]. 中国防汛抗旱, 2022, 32(3): 66-72)
[17] Sundar H, Sampath R S, Biros G. Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel[J]. SIAM Journal on Scientific Computing, 2008, 30(5): 2675-2708
[18] Kim B, Tsiotras P, Hong J M, *et al*. Interpolation and parallel adjustment of center-sampled trees with new balancing constraints[J]. The Visual Computer, 2015, 31(10): 1351-1363
[19] Zhang Heng, Sun Chong, Cai Youlin, *et al*. CFD study on wake field of submerged waterjet based on adaptive mesh refinement[J]. Shipbuilding of China, 2024, 65(4): 242-249(in Chinese)
(张恒, 孙翀, 蔡佑林, 等. 基于自适应网格技术的浸没式喷水推进尾流场 CFD 研究[J]. 中国造船, 2024, 65(4): 242-249)
[20] Berger M J, Oliger J. Adaptive mesh refinement for hyperbolic partial differential equations[J]. Journal of Computational Physics, 1984, 53(3): 484-512
[21] Xiao Zhongyun, Guo Yongheng, Zhang Lu, *et al*. Block structured adaptive mesh refinement for rotor flows[J]. Acta Aerodynamica Sinica, 2022, 40(5): 158-165(in Chinese)
(肖中云, 郭永恒, 张露, 等. 旋翼流动的块结构化网格自适应方法[J]. 空气动力学学报, 2022, 40(5): 158-165)
[22] Chen Yi, Qin Ling, Xiao Junfu, *et al*. The application study of adaptive mesh refinement in automotive CFD[J]. Journal of Chongqing University of Technology (Natural Science), 2024, 38(5): 95-101(in Chinese)
(陈祎, 秦玲, 肖竣夫, 等. 网格自适应加密在汽车 CFD 中的应用[J]. 重庆理工大学学报(自然科学), 2024, 38(5): 95-101)
[23] Chen Hao, Qi Long, Hua Ruhao, *et al*. An overset cartesian grid methodology for simulation of flow around three-element airfoils[J]. Journal of Computer-Aided Design & Computer, 2024: 1-7(in Chinese)
(陈浩, 齐龙, 华如豪, 等. 面向三段翼绕流模拟的重叠笛卡儿网格生成方法[J]. 计算机辅助设计与图形学学报, 2024: 1-7)
[24] Vanella M, Rabenold P, Balaras E. A direct-forcing embedded-boundary method with adaptive mesh refinement for fluid–structure interaction problems[J]. Journal of Computational Physics, 2010, 229(18): 6427-6449
[25] Koschier D, Deul C, Bender J. Hierarchical hp-adaptive signed distance fields[C] //Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Aire-la-Ville: Eurographics Association Press, 2016: 189-198
[26] Zhu B, Lu W L, Cong M, *et al*. A new grid structure for domain extension[J]. ACM Transactions on Graphics, 2013, 32(4): Article No.63
[27] Omella Á J, Pardo D. r-Adaptive deep learning method for solving partial differential equations[J]. Computers & Mathematics with Applications, 2024, 153: 33-42
[28] Viitanen T, Koskela M, Jääskeläinen P, *et al*. MergeTree: a fast hardware HLBVH constructor for animated ray tracing[J]. ACM Transactions on Graphics, 2017, 36(5): Article No.169
[29] Chen X R, Tang M, Li C, *et al*. BADF: bounding volume hierarchies centric adaptive distance field computation for deformable objects on GPUs[J]. Journal of Computer Science and Technology, 2022, 37(3): 731-740
[30] Zellmann S, Schulze J P, Lang U. Binned *k*-d tree construction for sparse volume data on multi-core and GPU systems[J]. IEEE Transactions on Visualization and Computer Graphics, 2021, 27(3): 1904-1915
[31] Ströter D, Mueller-Roemer J S, Stork A, *et al*. OLBVH: octree linear bounding volume hierarchy for volumetric meshes[J]. The Visual Computer, 2020, 36(10): 2327-2340
[32] Fernández-Fernández J A, Westhofen L, Löschner F, *et al*. Fast octree neighborhood search for SPH simulations[J]. ACM Transactions on Graphics, 2022, 41(6): Article No.242
[33] Tang Yong, Yang Sisi, Lu Mengya, *et al*. Collision detection for cloth based on adaptive enclosing ellipsoids[J]. Journal of Computer-Aided Design & Computer Graphics, 2013, 25(10): 1589-1596(in Chinese)
(唐勇, 杨偲偲, 吕梦雅, 等. 自适应椭球包围盒改进织物碰撞检测方法[J]. 计算机辅助设计与图形学学报, 2013, 25(10): 1589-1596)
[34] Lysenko M, D'Souza R, Shene C-K. Improved binary space partition merging [J]. Computer-Aided Design, 2008, 40(12): 1113-1120
[35] Vinkler M, Havran V, Bittner J. Performance comparison of bounding volume hierarchies and Kd-trees for GPU ray tracing[J]. Computer Graphics Forum, 2016, 35(8): 68-79
[36] Goldsmith J, Salmon J. Automatic creation of object hierarchies for ray tracing[J]. IEEE Computer Graphics and Applications, 1987, 7(5): 14-20
[37] Ericson C. Real-time collision detection[M]. Amsterdam: Elsevier, 2005
[38] Zhou K, Hou Q M, Wang R, *et al*. Real-time KD-tree construction on graphics hardware[J]. ACM Transactions on Graphics, 2008, 27(5): Article No.126
[39] Garanzha K, Pantaleoni J, McAllister D. Simpler and faster HLBVH with work queues[C] //Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. New York: ACM Press, 2011: 59-64
[40] Pantaleoni J, Luebke D. HLBVH: hierarchical LBVH construction for real-time ray tracing of dynamic geometry[C] //Proceedings of the Conference on High Performance Graphics. Aire-la-Ville: Eurographics Association Press, 2010: 87-95
[41] Lauterbach C, Garland M, Sengupta S, *et al*. Fast BVH construction on GPUs[J]. Computer Graphics Forum, 2009, 28(2): 375-384
[42] Ajmera P, Goradia R, Chandran S, *et al*. Fast, parallel, GPU-based construction of space filling curves and octrees[C] //Proceedings of the Symposium on Interactive 3D Graphics and Games. New York: ACM Press, 2008: Article No.10
[43] Burstedde C, Wilcox L C, Ghattas O. p4est: scalable algorithms for parallel adaptive mesh refinement on forests of octrees[J]. SIAM Journal on Scientific Computing, 2011, 33(3): 1103-1133
[44] Isaac T, Burstedde C, Ghattas O. Low-cost parallel algorithms for 2: 1 octree balance[C] //Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium. Los Alamitos: IEEE Computer Society Press, 2012: 426-437
[45] Liu F C, Kim Y J. Exact and adaptive signed distance FieldsComputation for rigid and deformablemodels on GPUs[J]. IEEE Transactions on Visualization and Computer Graphics, 2014, 20(5): 714-725
[46] Strain J. Fast tree-based redistancing for level set computations[J]. Journal of Computational Physics, 1999, 152(2): 664-686
[47] Strain J. A fast modular semi-Lagrangian method for moving interfaces[J]. Journal of Computational Physics, 2000, 161(2): 512-536
[48] Min C, Gibou F. A second order accurate level set method on non-graded adaptive Cartesian grids[J]. Journal of Computational Physics, 2007, 225(1): 300-321
[49] Gibou F, Fedkiw R, Osher S. A review of level-set methods and some recent applications[J]. Journal of Computational Physics, 2018, 353: 82-109
[50] Losasso F, Gibou F, Fedkiw R. Simulating water and smoke with an octree data structure[C] //Proceedings of the ACM SIGGRAPH Papers. New York: ACM Press, 2004: 457-462
[51] Setaluri R, Aanjaneya M, Bauer S, *et al*. SPGrid: a sparse paged grid structure applied to adaptive smoke simulation[J]. ACM Transactions on Graphics, 2014, 33(6): Article No.205
[52] Harlow F H, Welch J E. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface[J]. Physics of Fluids, 1965, 8(12): 2182-2189
[53] Beltman R, Anthonissen M, Koren B. Conservative mimetic cut-cell method for incompressible Navier-Stokes equations[C] //Proceedings of the Numerical Mathematics and Advanced Applications ENUMATH. Heidelberg: Springer, 2019: 1035-1043
[54] Gómez P, Zanzi C, López J, *et al*. Simulation of high density ratio interfacial flows on cell vertex/edge-based staggered octree grids with second-order discretization at irregular nodes[J]. Journal of Computational Physics, 2019, 376: 478-507
[55] Arakawa A, Lamb V R. Computational design of the basic dynamical processes of the UCLA general circulation model[J]. Methods in Computational Physics: Advances in Research and Applications, 1977, 17: 173-265
[56] Blomquist M, West S R, Binswanger A L, *et al*. Stable nodal

projection method on octree grids[J]. Journal of Computational Physics, 2024, 499: Article No.112695
[57] Popinet S. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries[J]. Journal of Computational Physics, 2003, 190(2): 572-600
[58] Losasso F, Fedkiw R, Osher S, *et al*. Spatially adaptive techniques for level set methods and incompressible flow[J]. Computers & Fluids, 2006, 35(10): 995-1010
[59] Lee B, Kim J, Min C. Super-convergence analysis on two symmetric Poisson solvers in octree grids[J]. Journal of Computational Physics, 2022, 464: Article No.111324
[60] Batty C. A cell-centred finite volume method for the Poisson problem on non-graded quadtrees with second order accurate gradients[J]. Journal of Computational Physics, 2017, 331: 49-72
[61] Teunissen J, Ebert U. Afivo: a framework for quadtree/octree AMR with shared-memory parallelization and geometric multigrid methods[J]. Computer Physics Communications, 2018, 233: 156-166
[62] Laine S, Karras T. Efficient sparse voxel octrees[C] //Proceedings of the ACM SIGGRAPH symposium on Interactive 3D Graphics and Games. New York: ACM Press, 2010: 55-63
[63] Koschier D, Bender J, Solenthaler B, *et al*. Smoothed particle hydrodynamics techniques for the physics based simulation of fluids and solids[C] //Proceedings of the 40th Annual Conference of the European Association for Computer Graphics. Aire-la-Ville: Eurographics Association Press, 2019: 1-41
[64] Liu Shusen, He Xiaowei, Wang Wencheng, *et al*. Survey on fluid simulation using smoothed particle hydrodynamics[J]. Journal of Software, 2024, 35(1): 481-512(in Chinese)
(刘树森, 何小伟, 王文成, 等. 光滑粒子流体动力学流体仿真技术综述[J]. 软件学报, 2024, 35(1): 481-512)
[65] Fei Y, Guo Q, Wu R D, *et al*. Revisiting integration in the material point method: a scheme for easier separation and less dissipation[J]. ACM Transactions on Graphics, 2021, 40(4): Article No.109
[66] Müller M, Schirm S, Teschner M, *et al*. Interaction of fluids with deformable solids[J]. Computer Animation and Virtual Worlds, 2004, 15(3-4): 159-171
[67] Keiser R, Adams B, Gasser D, *et al*. A unified lagrangian approach to solid-fluid animation[C] //Proceedings of the Eurographics/IEEE VGTC Symposium Point-Based Graphics. Los Alamitos: IEEE Computer Society Press,, 2005: 125-148
[68] Gissler C, Peer A, Band S, *et al*. Interlinked SPH pressure solvers for strong fluid-rigid coupling[J]. ACM Transactions on Graphics, 2019, 38(1): Article No.5
[69] Liu G R, Liu M B. Smoothed particle hydrodynamics: a meshfree particle method[M]. Singapore: World Scientific, 2003
[70] Akbay M, Nobles N, Zordan V, *et al*. An extended partitioned method for conservative solid-fluid coupling[J]. ACM Transactions on Graphics, 2018, 37(4): Article No.86
[71] He X W, Wang H M, Wu E H. Projective peridynamics for modeling versatile elastoplastic materials[J]. IEEE Transactions on Visualization and Computer Graphics, 2018, 24(9): 2589-2599
[72] Huang L B, Hädrich T, Michels D L. On the accurate large-scale simulation of ferrofluids[J]. ACM Transactions on Graphics, 2019, 38(4): Article No.93
[73] Gissler C, Henne A, Band S, *et al*. An implicit compressible SPH solver for snow simulation[J]. ACM Transactions on Graphics, 2020, 39(4): Article No.36
[74] Ihmsen M, Orthmann J, Solenthaler B, *et al*. SPH fluids in computer graphics[C] //Proceedings of the 35th Annual Conference of the European Association for Computer Graphics. Aire-la-Ville: Eurographics Association Press, 2014: 24-42
[75] Macklin M, Müller M. Position based fluids[J]. ACM Transactions on Graphics, 2013, 32(4): Article No.104
[76] Carson A M, Banks J W, Henshaw W D, *et al*. High-order accurate implicit-explicit time-stepping schemes for wave equations on overset grids[J]. Journal of Computational Physics, 2025, 520: Article No.113513
[77] Jiang C F F, Schroeder C, Selle A, *et al*. The affine particle-in-cell method[J]. ACM Transactions on Graphics, 2015, 34(4): Article No.51
[78] Hu Y M, Fang Y, Ge Z H, *et al*. A moving least squares material point method with displacement discontinuity and two-way rigid body coupling[J]. ACM Transactions on Graphics, 2018, 37(4): Article No.150
[79] Zhu K X, He X W, Li S, *et al*. Shallow sand equations: real-time height field simulation of dry granular flows[J]. IEEE Transactions on Visualization and Computer Graphics, 2021, 27(3): 2073-2084
[80] Kurganov A, Petrova G. A second-order well-balanced positivity preserving central-upwind scheme for the Saint-Venant system[J]. Communications in Mathematical Sciences, 2007, 5(1): 133-160
[81] Jeong W K, Whitaker R T. A fast iterative method for eikonal equations[J]. SIAM Journal on Scientific Computing, 2008, 30(5): 2512-2534
[82] Weber O, Devir Y S, Bronstein A M, *et al*. Parallel algorithms for approximation of distance maps on parametric surfaces[J]. ACM Transactions on Graphics, 2008, 27(4): Article No.104
[83] Pasko A, Adzhiev V, Sourin A, *et al*. Function representation in geometric modeling: concepts, implementation and applications[J]. The Visual Computer, 1995, 11(8): 429-446
[84] Lorensen W E, Cline H E. Marching cubes: a high resolution 3D surface construction algorithm[M] //Wolfe R. Seminal Graphics: Pioneering Efforts That Shaped the Field. New York: ACM Press, 1998: 347-353
[85] Macklin M, Erleben K, Müller M, *et al*. Local optimization for robust signed distance field collision[J]. Proceedings of the ACM on Computer Graphics and Interactive Techniques, 2020, 3(1): Article No.8
[86] Lin M C. Efficient collision detection for animation and robotics[D]. Berkeley: University of California, Berkeley, 1993