•智能交叉科学与工程•

DOI:10.12454/j.jsuese.202400163

本刊网刊

基于粒子索引排序算法的kd-tree缓存优化问题研究

张　挺,林震寰,杨丁颖*,王宗锴,陈轶凡

(福州大学 土木工程学院,福建 福州 350116)

摘　要:在使用kd-tree进行大规模随机粒子近邻搜索时,可能出现计算域内索引值相近的粒子在空间上距离较远而导致kd-tree搜索路径在短时间内产生较大差异等问题,使得节点数据的访问效率降低,最终影响kd-tree近邻搜索的效率。为解决该问题,本文引入了主成分分析中最大离散度降维的思想,采用平均绝对差作为离散度衡量指标,提出了基于平均绝对差粒子索引值排序的缓存优化策略MAD-index-sort,通过计算粒子集群平均绝对差最大的维度来实现数据降维,进而完成粒子的索引值重排序,并应用具有自动终止准则的ATC-kd-tree进行近邻搜索。为验证MAD-index-sort缓存优化策略的可行性,设计了不同维度和离散度对照组进行近邻搜索效率对比实验。结果表明,MAD-index-sort能根据粒子集群的离散度自动改变排序方向,具有更强的适应性,相较于未排序的情况性能最高可提升30.3%。

关键词:kd-tree;粒子近邻搜索;缓存优化;粒子索引值排序

中图分类号:O35　　**文献标志码:**A　　**文章编号:**2096-3246(2026)01-0313-11

kd-tree是一种特殊的k维二叉树结构[1],因能够高效求解近邻搜索问题而被广泛应用于粒子运动学[2]、计算流体力学[3–5]、光线追踪[6]、图像识别[7]等领域。针对kd-tree的改进主要围绕回溯策略[8]、参数率定[9–10]等方面,其中:回溯策略改进主要是对kd-tree近邻搜索过程中的回溯路径进行重组[11],进而减少搜索过程中不必要的距离运算次数;参数率定则是针对kd-tree的可变参数,即最大深度$d_{\max}$和叶子节点大小阈值n_0进行率定,从而使kd-tree近邻搜索效率最高[12–14]。

kd-tree的近邻搜索效率不仅与回溯搜索的方式和参数取值有关,还与搜索过程中的缓存优化[15]问题存在关联。缓存优化问题涉及如何最大限度地利用高速缓存,以减少程序在数据访问过程中产生的缓存未命中(cache miss)次数[16]。当kd-tree在搜索过程中访问节点时,如果所需数据不在高速缓存中,就需要从较慢的主存中获取,将会增加访问时间,而通过优化数据的存储结构和访问模式,使kd-tree在进行近邻搜索时,能更高效地使用缓存,从而减少访问主存的次数[17],缩短节点数据的访问时间,进而减小近邻搜索的总时间成本。例如在光线追踪时,随着场景复杂度的增加,kd-tree的节点访问频率显著提高,Liang等[18]提出了一种改进的kd-tree构建算法,通过平衡节点的访问频率,优化了缓存命中率,显著提高了光线追踪的性能。在三维点云配准领域,Shi等[19]研究了不同kd-tree构建方法对缓存命中的影响,提出了一种基于层次化数据分布的kd-tree构建方法,有效减少了缓存未命中次数,提高了点云配准的精度和速度。Choi等[20]则在该研究的基础上提出了近似搜索的回溯策略,通过限制kd-tree回溯次数,进一步缩短搜索时间成本。Nuchter等[21]提出的近似缓存优化策略,将kd-tree在第i次近邻搜索过程中访问的叶子节点存储在节点指针队列中,而后在第i+1次近邻搜索过程中优先访问节点指针队列,进而减少不必要的下溯搜索,缩短数据访问时间,优化缓存命中。

事实上,在计算流体力学中,kd-tree同样存在缓存优化问题。以流体运动模拟问题为例,粒子的初始分布状态为均匀布点,在受到外部激励后,速度、空间位置等属性将随运算时间延长而不断改变,出现随机布

收稿日期:2024 – 03 – 01　　**修回日期:**2024 – 08 – 14　　**网络出版日期:**2024 – 10 – 16

基金项目:国家自然科学基金项目(52079032);福建省水利科技项目(MSK202215;MSK202333)

作者简介:张　挺(1977—),男,教授,博士. 研究方向:水力学及河流动力学. E-mail:zhangting@fzu.edu.cn

* **通信作者:**杨丁颖,讲师,E-mail:ydytcl@fzu.edu.cn

点状态，可能导致计算域内索引值相近的粒子在空间上距离较远。这种情况下极易引起kd-tree在短时间内的搜索路径存在较大差异，降低节点数据的访问效率，最终对kd-tree近邻搜索效率产生影响。值得注意的是，在流体数值模拟中，针对近邻搜索算法缓存优化问题的研究多集中于均匀网格法(UGM)[22]，该方法划分的空间子区域按等大层次排布。基于均匀网格法的这种特点，可通过对其划分的空间子区域进行索引值排序，进而减少缓存未命中次数[23−24]，如Ihmsen等[25]提出的“Z”字索引排序(Z-index-sort)策略，将均匀网格法划分的空间子区域按照“Z”字形进行排序，提高粒子空间分布的连续性以及算法的缓存命中率。然而，对于kd-tree而言，空间子区域的大小与粒子集群的空间分布有关，不同子区域的大小互不相等[26]，故难以对kd-tree划分的空间子区域进行索引值排序，Z-index-sort策略不再适用。因此，如何提高kd-tree近邻搜索效率，实现缓存优化成为一个值得关注的问题。

为解决该问题，本文提出了基于平均绝对差粒子索引值排序(mean absolute deviation index sort，MAD-index-sort)的缓存优化策略，引入了主成分分析(PCA)中最大离散度降维[27]的思想，采用平均绝对差(MAD)[28−31]作为离散度衡量指标，通过计算粒子集群平均绝对差最大的维度来实现数据降维，进而完成粒子的索引值重排序。在此基础上，构建kd-tree树形结构，应用kd-tree自动终止准则(ATC-kd-tree)[12]，求解近邻搜索问题，进而优化近邻搜索过程中的缓存命中情况。选取流体力学中常见布点形状进行系列实验，分析粒子均匀布点与随机布点下kd-tree近邻搜索效率存在差异的原因，并且设置两种近邻搜索对比方案，验证其性能。

1 运算性能衡量指标与缓存优化问题

在粒子近邻搜索问题中，kd-tree通过粒子的唯一索引值快速确定其空间位置，并按照索引值大小进行近邻点搜索。在均匀布点情况下，粒子呈现等间距层次分布规律，索引值相近的粒子往往在空间上相邻，支持域有较大概率重叠，近邻点较多。而在随机布点情况下，粒子则呈现不等间距任意分布的规律，索引值相近的粒子空间距离较远，支持域交集较少，导致kd-tree的缓存命中率降低，近邻搜索效率下降。

1.1 运算性能衡量指标

在对近邻搜索算法的缓存优化问题进行分析前，首先要明确衡量kd-tree近邻搜索效率的指标以及CPU缓存命中情况的计算方法。

本文采用实测的kd-tree算法耗时，即kd-tree总时间成本C作为效率衡量指标，该指标可精确反映kd-tree运算效率的变化趋势[12]，则kd-tree的时间成本评价模型如下：

$$C=\lambda_1(C_{\mathrm{di}}+C_{\mathrm{tr}})+\lambda_2 C_{\mathrm{di}}+C_{\mathrm{co}} \tag{1}$$

式中：C_{tr}为父节点下溯到子节点的时间成本；C_{di}为单次距离运算的时间成本；λ_1为下溯过程中经过的内部节点个数，即近邻搜索过程中在内部节点进行的距离运算次数；λ_2为近邻搜索过程中在叶子节点进行的距离运算次数；C_{co}为构建kd-tree算法的时间成本。基于该模型，可对比分析不同布点形状和方式下近邻搜索效率和距离运算次数的差异，揭示布点方式对kd-tree近邻搜索效率存在影响的关键原因。

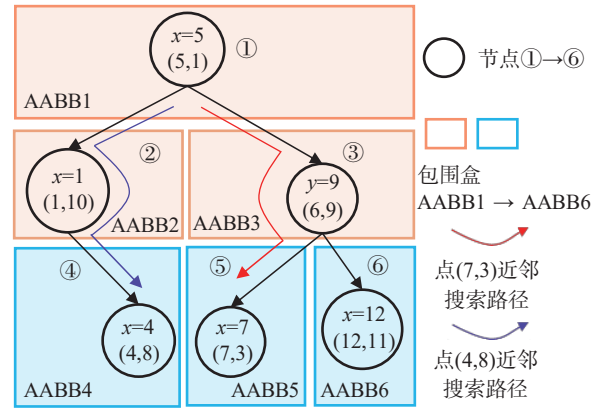
同时，为了对比分析缓存优化策略的效率，本文采用Cachegrind[28]模拟CPU中一级缓存和三级缓存的命中指标及读写次数等5个指标，包括：一级数据缓存的未命中次数D_1、一级指令缓存的未命中次数I_1、三级缓存未命中次数M_3，数据缓存读写次数D和指令缓存读写次数I。其中：一级数据缓存的未命中次数D_1是影响缓存命中最重要的因素，直接反映了程序中数据访问的效率，D_1较小表示一级数据缓存更有效，有助于提高整体性能；D和I包括了命中和未命中情况下的读写次数，考虑了在程序执行期间所有对数据和指令的访问，为缓存效率和程序整体性能的评估提供了全面的视角。基于这些指标，可推算出kd-tree在近邻搜索过程中CPU缓存的命中情况，便于对比不同布点方式下kd-tree的缓存情况，相关的分析将在第3.1和3.2节中给出。

1.2 缓存优化问题

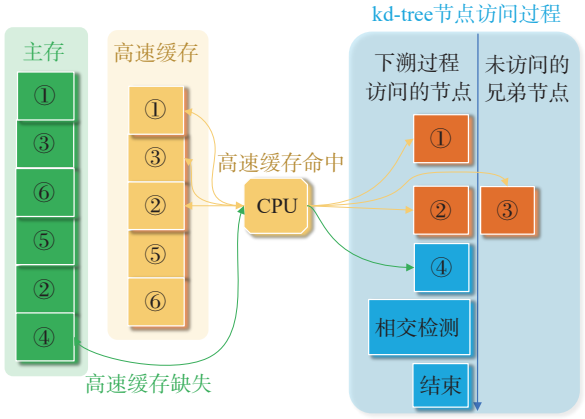
图1为kd-tree近邻搜索算法示意图。kd-tree近邻搜索算法是基于深度优先搜索原则的二分查找，现考虑如下近邻搜索问题：在二维粒子集群A_1={(1,10),(4,8),(5,1),(6,9),(7,3),(12,11)}中分别确定距离任意的粒子(8,3)和(1,10)最近的两个点。图1(a)为基于粒子集群A_1建立的kd-tree树形结构，基于该树形结构，kd-tree先确定出点(8,3)的近邻点。具体二分查找过程，可参考文献[12]。

值得一提的是，kd-tree近邻搜索算法需通过CPU来访问节点数据，对于CPU而言，数据存取的位置分别为主存和高速缓存，其中高速缓存的数据访问效率高于主存。如图1(b)所示，在完成点(8,3)的搜索任务后，搜索过程中访问的①、②、③、⑤、⑥号节点数据被更新到高速缓存中，随后kd-tree求解点(1,10)的近邻点。

不同于点(8,3)，点(1,10)的搜索路径以①号节点为起点，经过②号节点，最终下溯到④号节点，如图1(a)所示。并且，在搜索过程中CPU可直接从高速缓存中读取①、②、③号节点的数据，该现象即高速缓存命中。然而，kd-tree在上一次的搜索过程中并未访问④号节点，高速缓存中缺失了该节点，因此CPU在本次搜索



(a) kd-tree构建与近邻搜索过程（d_{max}=3，n_0=1）



(b) kd-tree近邻搜索数据访问过程

图1　kd-tree近邻搜索算法示意图

Fig. 1　Schematic diagram of kd-tree nearest neighbor search algorithm

过程中需从主存中读取④号节点，上述现象即高速缓存缺失，如图1(b)所示。

不难发现，kd-tree的数据访问效率与CPU高速缓存中是否存储相应的节点数据有关。当kd-tree完成索引值为i的粒子的近邻搜索任务后，搜索过程中访问的节点数据将被存储于高速缓存中。而后在求解索引值为i+1的粒子的近邻搜索任务时，CPU将先从高速缓存中查找本次搜索过程中需访问的节点数据，若高速缓存中存在相应的节点，则可显著提高数据读取的效率。进一步地，对于随机布点而言，其索引值相近的粒子在空间上距离较远，这导致kd-tree在前后两次近邻搜索过程中的下溯路径存在较大差异，访问的kd-tree节点不同，更容易出现缓存缺失现象，影响时间成本评价模型中C_{tr}和C_{di}的大小，使得C增大。

需要强调的是，在实际问题中，粒子的布点方式多为随机布点，该情况下粒子分布较为分散，近邻搜索算法的缓存未命中次数更多，如何减少kd-tree在近邻搜索过程中的缓存未命中次数，实现近邻搜索算法的缓存优化，减小C_{tr}和C_{di}，从而减小C将是一个值得关注的问题。

2　基于粒子索引排序算法的kd-tree缓存优化

为了提高kd-tree在随机布点条件下的近邻搜索效率，本文在分析Z-index-sort缓存优化策略适用性的基础上，提出一种基于最大离散度降维思想的MAD索引值排序缓存优化方法，用于改善kd-tree的数据局部性，实现缓存优化。

2.1　Z-index-sort策略

Z-index-sort[23]为均匀网格法缓存优化策略。仍以粒子集群A_1为例，图2为Z-index-sort策略示意图。如图2所示，计算域被划分为16份，每个空间子区域等大层次排布，区域内或不存在粒子，或存在多个粒子。子区域的索引值按“Z”字箭头形式分布，从箭头的开始为①号子区域，而后为②号子区域，依次递增直到“Z”字形的末尾。在搜索进行至⑬号子区域时，其范围内存在点(4,8)和(6,9)，均匀网格法在完成上述两个点的近邻搜索任务后，将⑬号子区域以及点(4,8)和(6,9)存储于高速缓存中，随后切换至⑭号子区域，该区域内存在点(1,10)，因此开始搜索点(6,9)的近邻点。因为在上一次的搜索过程中，已经访问过了点(4,8)和(6,9)，所以本次搜索点(1,10)近邻点的过程中，采用均匀网格法可以从高速缓存中直接访问点(4,8)和(6,9)，从而实现缓存优化，提高近邻搜索效率。可以看出，当空间子区域等大层次排布时，Z-index-sort通过对空间子区域进行索引值排序，使相邻空间子区域的索引值相近，进而实现均匀网格法近邻搜索的缓存优化。

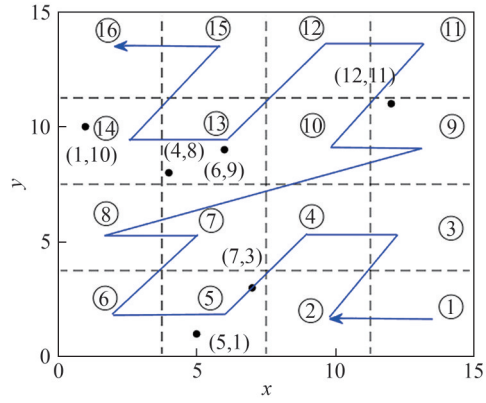


图2　Z-index-sort策略示意图

Fig. 2　Schematic diagram of Z-index-sort strategy

然而，对于kd-tree而言，Z-index-sort存在明显的局限性。首先，Z-index-sort主要适用于均匀网格法划分的空间子区域，其依赖于空间子区域等大的特性。然而，kd-tree划分的空间子区域大小是不等的，其取决于粒子集群的空间分布。其次，Z-index-sort假设相邻索引值的粒子在空间上也相邻，这在kd-tree的应

用中并不成立。随机布点情况下，相邻索引值的粒子在空间上可能相距甚远，导致 kd-tree 搜索路径差异显著，增加了缓存未命中的次数，降低了数据访问效率。

2.2 MAD-index-sort 策略

为了解决 Z-index-sort 在 kd-tree 应用中的局限性，本文提出了 MAD-index-sort 缓存优化策略。该策略引入了 PCA 中的最大离散度降维思想，通过线性变换将数据投影到离散度最大方向的低维空间中，同时尽可能保留数据的主要变异信息，使得在低维空间中仍能表达出原始数据的重要特征。常用的离散度衡量指标包括方差、极差以及 MAD 等。其中，MAD 能够有效反映粒子集群的离散度，指导数据降维和索引值重排序。通过计算粒子集群中 MAD 最大的维度来实现数据降维，并完成粒子的索引值重排序，使空间位置相邻的粒子索引值相近。这一选择基于以下考虑：首先，MAD 能够捕捉到粒子集群中不同维度的离散程度，有助于确定最佳的降维方向；其次，该指标比方差、极差有更强的稳定性，不易受离群点影响；最后，采用 MAD 作为衡量指标有助于动态调整排序策略，适应粒子集群的变化，提高缓存命中率。MAD-index-sort 能够动态调整排序方向，更好地适应粒子集群的空间分布特点，显著提高缓存命中率和近邻搜索效率。

图 3 为 MAD-index-sort 策略示意图。仍考虑粒子集群 A_1，其初始分布如图 3(a)所示，可见 A_1 中索引值相近的粒子在空间上距离较远，且粒子在 x 方向和 y 方向不存在明显的离散或聚集现象，即粒子集群在 x 方向和 y 方向上的离散度近似相等。此时，MAD-index-sort 策略将任选一个维度作为基准维度 M，并基于粒子在 M 维度上的大小进行索引值排序，使索引值相近的粒子在空间上相邻，从而提高粒子分布的空间连续性。以 x 方向为基准维度，如图 3(b)所示，根据粒子集群 A_1 中粒子的 x 值大小进行索引值排序，将 x 值最小的点(1,10)的索引值设定为 1，x 值第二小的点(4,8)的索引值设定为 2，依此类推，直至最后一个点(12,11)的索引值被设定。同理，若以 y 方向为基准维度，粒子的索引值排序如图 3(c)所示，其索引值依据粒子的 y 值大小递增。不难发现，MAD-index-sort 的目标是对粒子的索引值进行重排序，使相邻粒子的索引值相近。该方法与 Z-index-sort 策略中相邻空间子区域的索引值相近的思想类似，本质上都是使近邻搜索过程中访问的数据点尽可能在高速缓存中命中。

然而，在实际问题中，粒子集群在每个维度上的离散度存在差异，按不同基准维度进行索引值排序后，近邻搜索的缓存命中情况存在差异，会对 kd-tree 近邻搜索的效率产生影响。本文引入 PCA 中最大离散度降维的思想，旨在通过选择最具变异性的维度优化数据处理和算法性能。

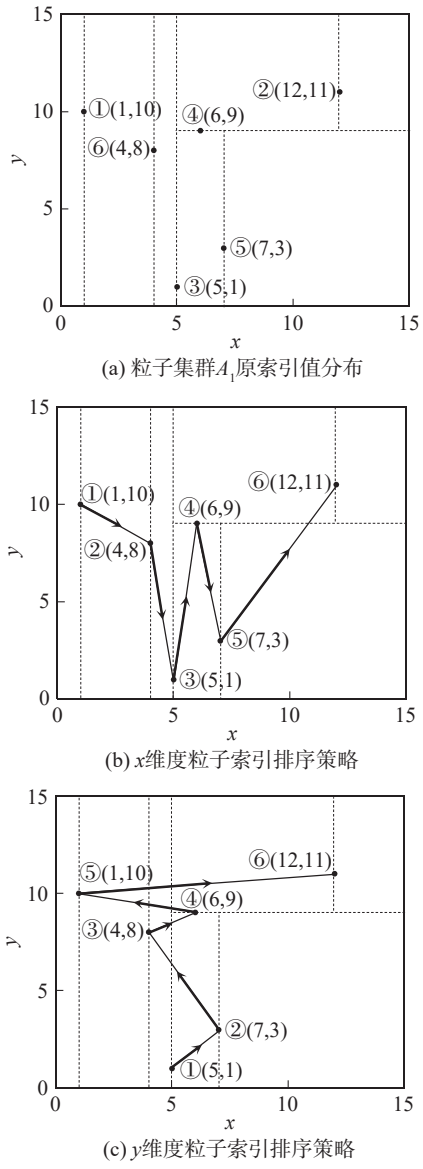


图 3 MAD-index-sort 策略示意图

Fig. 3 Schematic diagram of MAD-index-sort particle index value sorting schem

首先，选取稳定性最强的 MAD 作为离散度衡量指标，计算粒子集群每个维度的 MAD。以粒子总数为 N 的 m 维粒子集群 P 为例，如图 4 所示，MAD-index-sort 策略先计算粒子集群 P 在每个维度的最大平均差，并形成 MAD 的向量 $\boldsymbol{K}_{\mathrm{MAD}}$。

$$K_{\mathrm{MAD}\,i}=\frac{\sum_{j=1}^{N}\left|P_j^i-\overline{P^i}\right|}{N},\ i=1,2,\cdots,m \tag{2}$$

式中：$K_{\mathrm{MAD}i}$ 为粒子集群 P 在第 i 维上的 MAD；P_j^i 为索引值为 j 的粒子在第 i 维上的大小；$\overline{P^i}$ 为粒子集群第 i 维上的平均值，可用式(3)计算。

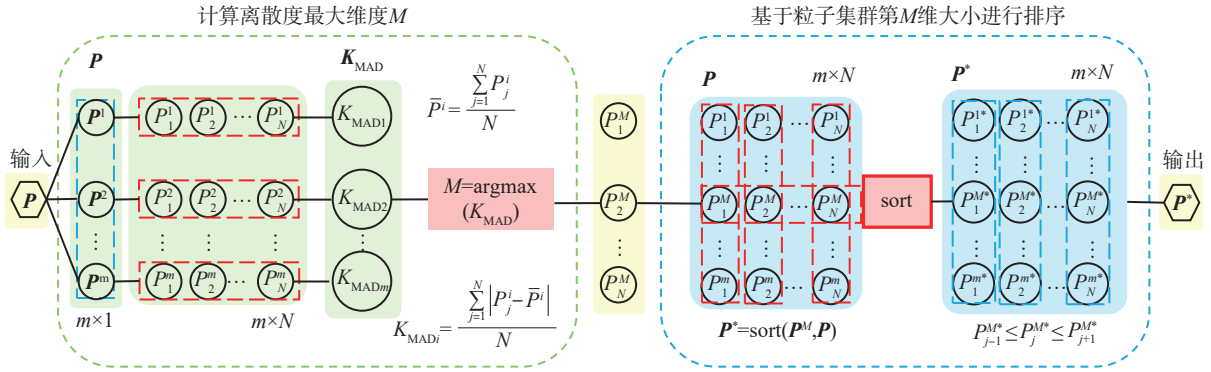


图4　MAD-index-sort粒子索引排序算法示意图

Fig. 4　Schematic diagram of MAD-index-sort particle index sorting algorithm

$$\overline{P^i}=\frac{\sum_{j=1}^{N}P_j^i}{N} \tag{3}$$

$K_{\mathrm{MAD}i}$越大则说明粒子集群中的各个数据点与平均值之间的绝对偏差越大，表明粒子集群的分散程度越大，即离散度越大；反之，$K_{\mathrm{MAD}i}$越小则意味着该方向上的布点越集中，即聚集度越高，随着$K_{\mathrm{MAD}i}$的进一步减小，可能会导致空间布点形成聚集或簇状分布，从而影响kd-tree的局部近邻搜索效率。

然后，确定$\boldsymbol{K}_{\mathrm{MAD}}$中最大值所在的维度$M$，选取离散度最大的维度，这个维度对应的数据分布最为分散，具有最大的变异性。将粒子集群降维到第M维上，以第M维的向量$\boldsymbol{P}^M$为基准进行索引值排序。将第M维值最小的粒子的索引值设定为1，依次递增，直到第M维值最大的粒子的索引值被设定为N为止。

最后，基于该维度的值对粒子进行索引值排序，形成新的粒子集群坐标值矩阵$\boldsymbol{P}^*$，该矩阵中的元素满足如下关系式：

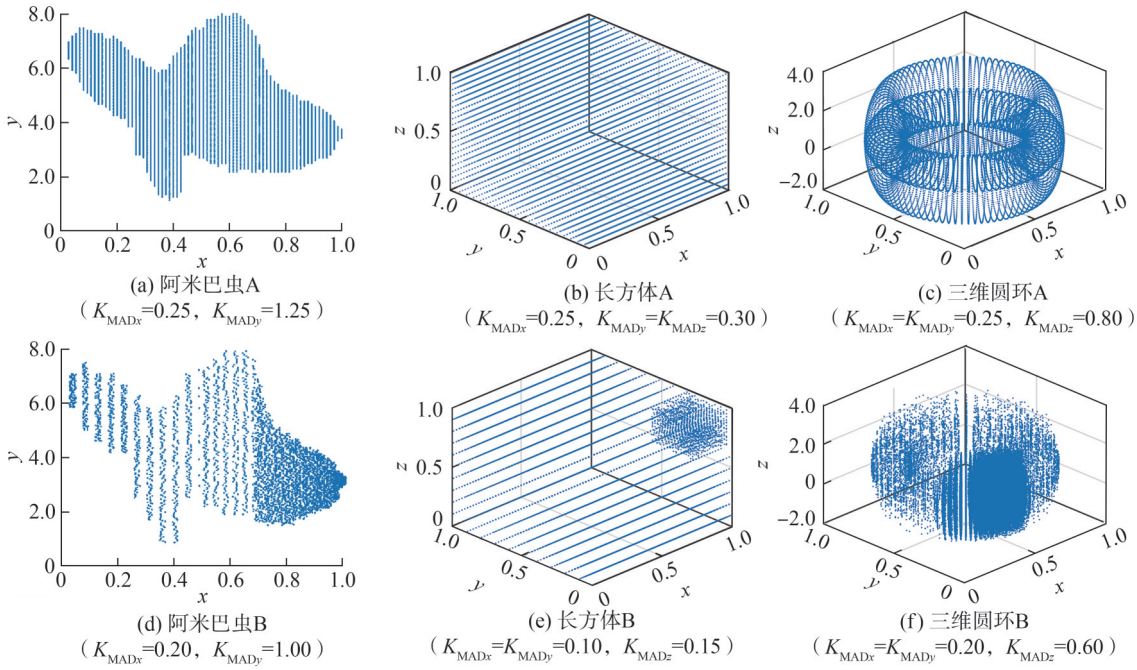
$$P_{j-1}^{M*}\le P_j^{M*}\le P_{j+1}^{M*} \tag{4}$$

式中，P_j^{M*}为索引值排序后索引值为j的粒子在第M维的大小。

在完成粒子索引值排序后，构建如图1(a)所示的树形结构，并应用ATC-kd-tree进行近邻搜索，该过程即为本文提出的kd-tree缓存优化策略，进而减少近邻搜索过程中的缓存未命中次数。ATC-kd-tree近邻搜索的详细过程可参考文献[12]。

3　离散度差异性分析与效率对比实验

为验证MAD-index-sort缓存优化策略的可行性，设计效率对比实验，图5为布点形状示意图。如图5所示，实验布点形状为阿米巴虫、长方体、三维圆环3种[32–33]，考虑到布点形状的复杂性，粒子会呈现区域



(a) 阿米巴虫A（$K_{\mathrm{MAD}x}$=0.25，$K_{\mathrm{MAD}y}$=1.25）

(b) 长方体A（$K_{\mathrm{MAD}x}$=0.25，$K_{\mathrm{MAD}y}$=$K_{\mathrm{MAD}z}$=0.30）

(c) 三维圆环A（$K_{\mathrm{MAD}x}$=$K_{\mathrm{MAD}y}$=0.25，$K_{\mathrm{MAD}z}$=0.80）

(d) 阿米巴虫B（$K_{\mathrm{MAD}x}$=0.20，$K_{\mathrm{MAD}y}$=1.00）

(e) 长方体B（$K_{\mathrm{MAD}x}$=$K_{\mathrm{MAD}y}$=0.10，$K_{\mathrm{MAD}z}$=0.15）

(f) 三维圆环B（$K_{\mathrm{MAD}x}$=$K_{\mathrm{MAD}y}$=0.20，$K_{\mathrm{MAD}z}$=0.60）

图5　布点形状示意图

Fig. 5　Schematic diagram of point layout shape

性聚集和离散现象，即每个维度的离散度会存在一定的差异。为探究离散度对 kd-tree 运算性能的影响，每个布点形状分为 A、B 两个离散度不同的对照组，采用 $K_{\mathrm{MAD}i}$ 作为衡量指标，$K_{\mathrm{MAD}i}$ 的值越小则说明该维度上的布点越集中，即聚集度越高。实验环境如下：CPU 为 Intel®Core™i7-10700，内存 16 GB，编译器为 MinGW。缓存命中情况采用 Cachegrind[28] 统计，一级数据缓存和一级指令缓存大小均为 256 kB，三级缓存大小为 16 MB。

3.1 近邻搜索效率及布点方式的差异性分析

为了深入了解不同搜索方法在各种布点形状下的效率差异，对比了穷举法、均匀网格搜索法和 kd-tree 这 3 种方法的近邻搜索总时间成本。其中，均匀网格搜索法的网格密度为 10×10×10，按照坐标轴均匀划分。表 1 为粒子总数 N=1×10^6 时不同搜索方案近邻搜索总时间成本对比。由表 1 可见，无论是阿米巴虫、长方体还是三维圆环布点形状，kd-tree 方法均表现出最优的时间效率。具体而言，在阿米巴虫 A 布点形状下，穷举法的时间成本为 87.12 s，而均匀网格搜索法为 6.34 s，而 kd-tree 的时间成本仅为 0.72 s。类似的趋势也出现在其他布点形状中，如长方体 A 和三维圆环 A，穷举法的时间成本分别为 209.88 s 和 279.27 s，而 kd-tree 的时间成本仅为 1.06 s 和 1.07 s。实验结果表明，kd-tree 在处理大规模数据时，能够显著降低搜索时间成本，特别是在复杂布点形状下优势更加明显。此外，通过对比不同布点形状的时间成本，可以观察到布点形状对 kd-tree 性能的影响。例如，对于三维圆环 B，总时间成本为 1.78 s，相较于三维圆环 A 的 1.07 s 有所增加。这一变化表明，布点形状的复杂度和离散度对 kd-tree 的性能产生了一定影响。在相同的粒子总数下，更复杂的布点形状会导致更高的时间成本，这进一步验证了 kd-tree 在处理不同布点形状时的适应性和性能优势。

表 1 不同搜索方案近邻搜索总时间成本（N=1×10^6）

Tab. 1 Total time cost of nearest neighbor search for different search schemes (N=1×10^6)

布点形状	C/s		
	穷举法	均匀网格搜索法	kd-tree
阿米巴虫 A	87.12	6.34	0.72
阿米巴虫 B	117.04	7.85	0.88
长方体 A	209.88	9.14	1.06
长方体 B	309.89	11.08	1.33
三维圆环 A	279.27	9.48	1.07
三维圆环 B	461.02	15.22	1.78

为了分析不同离散度情况下 kd-tree 近邻搜索效率存在差异的原因，实验中并未采用任何排序策略，而是在各个坐标方向上均进行随机布点。表 2 为粒子总数 N=1×10^6 时，不同布点形状对应的离散度与运算性能指标，包括距离运算次数 λ_1、λ_2，总时间成本 C 和一级数据缓存的未命中次数 D_1。总体上看，总时间成本 C 与 λ_1、λ_2、D_1 有关，而 λ_1、λ_2、D_1 值又与粒子集群的维度、布点形状、离散度等因素密切相关。对比不同离散度布点情况（A、B 对照组），可见离散度对 D_1 的影响十分显著，随着离散度指标 $K_{\mathrm{MAD}i}$ 的减小，即布点聚集度越高，D_1 越大。以三维圆环布点形状为例，三维圆环 B 布点情况下，距离运算次数 λ_1、λ_2 与三维圆环 A 相差不超过 5.4%，而 D_1 增大了 167.1%，这导致三维圆环 B 布点情况下的总时间成本 C 比三维圆环 A 高出约 66.4%。

表 2 不同布点形状对应的离散度与运算性能指标（N=1×10^6）

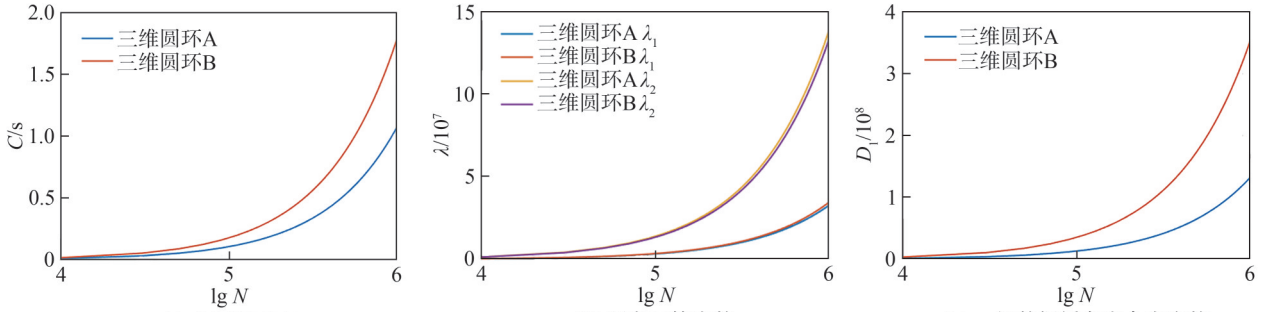
Tab. 2 Dispersion and computing performance indicators corresponding to different point shapes (N=1×10^6)

布点形状	离散度指标			运算性能指标			
	$K_{\mathrm{MAD}x}$	$K_{\mathrm{MAD}y}$	$K_{\mathrm{MAD}z}$	C/s	λ_1/10^6	λ_2/10^6	D_1/10^6
阿米巴虫 A	0.25	1.25		0.72	19.52	47.11	89.50
阿米巴虫 B	0.20	1.00		0.88(+22.2%)	20.69	52.78	174.07(+66.4%)
长方体 A	0.25	0.30	0.30	1.06	28.10	119.02	282.90
长方体 B	0.10	0.10	0.15	1.33(+25.5%)	29.05	95.98	355.55(+25.7%)
三维圆环 A	0.25	0.25	0.80	1.07	32.80	139.01	131.80
三维圆环 B	0.20	0.20	0.60	1.78(+66.4%)	34.57	133.57	351.98(+167.1%)

注：括号中数据为形状 B 相对于形状 A 的运算性能指标变化率。

为了分析不同总点数对搜索效率的影响规律，进一步以三维圆环形状为例进行实验。图 6 为不同总点数下离散度差异对 kd-tree 近邻搜索效率的影响对比，图 6 中给出了粒子总数 N 条件下 kd-tree 总时间成本 C、距离运算次数 λ_1、λ_2 以及一级数据缓存的未命中次数 D_1 的对比。显然，kd-tree 近邻搜索的总时间成本 C 与 λ_1、λ_2、D_1 三者都存在关联，随着粒子总数的增加，总时间成本 C、距离运算次数 λ_1、λ_2 与一级数据缓存的未命中次数 D_1 均呈

现出指数级增大的趋势。另外，由图6可见，离散度变化对λ_1、λ_2的影响相对较小，而对D_1影响非常明显，成为影响总时间成本C的最关键因素。空间中粒子集的聚集度越高（即$K_{\mathrm{MAD}i}$值较小），D_1越大，从而使得时间成本评价模型中的C_{tr}、C_{di}的值增大，导致离散度小的情况下kd-tree搜索效率下降。综合上述分析可知，对于kd-tree，提高其近邻搜索效率的关键点在于提高近邻搜索过程中的缓存命中率，即减小D_1，以实现缓存优化。



(a) 总时间成本　(b) 距离运算次数　(c) 一级数据缓存未命中次数

图6　不同粒子总数下离散度差异对kd-tree近邻搜索效率的影响对比

Fig. 6　Comparison of the impact of dispersion differences on kd-tree nearest neighbor search efficiency under different total number of points

3.2　MAD-index-sort效率分析

为验证MAD-index-sort缓存优化策略的高效性，对比3种粒子索引值排序策略方案：Sort Ⅰ[12]使用ATC-kd-tree（$d_{\max}$=lb N-2，n_0=19），不对粒子的索引值进行排序；Sort Ⅱ[34]不考虑粒子集群在各个维度的离散度差异，在Sort Ⅰ的基础上仅根据粒子的x值大小对索引值进行排序；Sort Ⅲ在Sort Ⅰ的基础上再根据粒子集群中MAD最大维度进行索引值排序，即本文提出的MAD-index-sort策略。

表3为N=1×10^6^时不同粒子索引值排序策略的近邻搜索效率，其中变化率为Sort Ⅲ和Sort Ⅱ相对Sort Ⅰ的变化率。总体上看，MAD-index-sort策略的近邻搜索效率高于Sort Ⅰ和Sort Ⅱ，其中三维圆环B的近邻搜索效率提升最明显，相较于Sort Ⅰ，总时间成本C缩短约30.3%。这说明在粒子索引值排序过程中基准维度M的选择尤为重要，Sort Ⅱ在排序过程中不考虑粒子集群在各方向上的离散度差异，仅沿着x方向进行排序，而本文中所用的布点形状在y和z方向上有更大的离散度差异，MAD-index-sort策略使得D_1和C进一步减少。

表3　不同粒子索引值排序策略近邻搜索效率（N=1×10^6^）

Tab. 3　Efficiency of nearest neighbor search with different particle index value sorting strategies (N=1×10^6^)

布点形状	Sort Ⅰ		Sort Ⅱ				Sort Ⅲ			
	C/s	D_1/10^6^	C/s	C变化率/%	D_1/10^6^	D_1变化率/%	C/s	C变化率/%	D_1/10^6^	D_1变化率/%
阿米巴虫A	0.72	89.50	0.58	−19.4	87.70	−2.0	0.51	−29.2	53.61	−40.1
阿米巴虫B	0.88	174.07	0.69	−21.6	168.05	−3.5	0.65	−26.1	109.92	−36.9
长方体A	1.06	282.90	0.86	−18.9	167.76	−40.7	0.82	−22.6	161.54	−42.9
长方体B	1.33	355.55	1.05	−21.1	195.88	−44.9	0.99	−25.6	179.51	−49.5
三维圆环A	1.07	131.80	0.95	−11.2	127.06	−3.6	0.78	−27.1	114.40	−13.2
三维圆环B	1.78	351.98	1.59	−10.7	340.43	−3.3	1.24	−30.3	302.27	−14.1

从布点形状的角度来看，形状A和形状B在不同排序策略下的表现存在差异。具体而言，形状B的总时间成本C和一级数据缓存未命中次数D_1都比形状A更高，这是因为形状B的粒子集群在空间中的分布更加复杂，导致计算和缓存需求增加。Sort Ⅲ情况下，在对形状B进行近邻搜索时的优化效果更为显著。例如，长方体B的总时间成本在Sort Ⅲ下减少了25.6%，一级缓存的未命中次数降低了49.5%，比相应的长方体A形状的优化幅度更大。这进一步证明了MAD-index-sort缓存优化策略在处理复杂粒子分布形状时的优势。

为进一步分析3种粒子索引值排序策略方案存在效率差异的原因，统计了各方案的缓存工作情况。表4为N=1×10^6^时，不同粒子索引值排序策略缓存未命中次数D_1、I_1、M_3、数据缓存和指令缓存读写次数D、I。在不同布点形状下均可发现Sort Ⅲ的一级数据缓存未命中次数D_1和三级缓存未命中次数M_3均低于其余方案，而一级指令缓存未命中次数I_1则不存在明显的差

异。同时，3 个方案的 D_1 都远高于 I_1 和 M_3。该规律说明减小 D_1 正是提高近邻搜索效率的关键。

观察表 4 中 3 种方案近邻搜索过程中的数据缓存以及指令缓存的读写次数 D、I，可以明显看出，近邻搜索过程中指令缓存读写次数 I 大于数据缓存读写次数 D。同时，Sort Ⅲ的 D、I 值总是略低于 Sort Ⅰ、Sort Ⅱ，这说明了本文提出的 MAD-index-sort 策略不但减小了 D_1，还略微减小了 D、I，近邻搜索效率更高。

表 4 不同粒子索引值排序策略缓存工作情况（N=1×10^6）

Tab. 4 Cache behavior of different particle index value sorting strategies (N=1×10^6)

布点形状	Sort Ⅰ					Sort Ⅱ					Sort Ⅲ				
	D_1/10^6	I_1/10^6	M_3/10^6	D/10^9	I/10^9	D_1/10^6	I_1/10^6	M_3/10^6	D/10^9	I/10^9	D_1/10^6	I_1/10^6	M_3/10^6	D/10^9	I/10^9
阿米巴虫B	174.07	5.52	5.92	195.23	322.94	168.05	5.34	5.58	194.72	322.14	109.92	5.34	5.23	194.24	321.39
长方体B	355.55	1.36	83.02	238.97	396.20	195.88	1.35	21.26	237.41	395.37	179.51	1.35	6.74	236.84	394.44
三维圆环B	351.98	2.53	73.54	316.75	530.12	340.43	2.46	21.81	315.34	526.89	302.27	2.46	7.17	314.63	525.78

综合上述分析，可得出如下结论：1）MAD-index-sort 策略减小了一级数据缓存的未命中次数 D_1，同时减小了数据缓存和指令缓存读写次数 D、I，从而缩短了总时间成本 C；2）kd-tree 近邻搜索的总时间成本 C 与缓存读写次数 D、I 有关，维度越高的布点形状 D、I 值越大，导致近邻搜索效率降低；3）在粒子索引值排序过程中，基准维度 M 的选择非常关键，本文提出的 MAD-index-sort 策略基于 MAD 来确定排序的基准维度 M，其排序方向能根据粒子集群的离散情况自动改变，具有更强的适应性，相较于沿着固定方向排序的 Sort Ⅱ有更高的搜索效率。

4 结 论

针对粒子集群离散度变化情况下 kd-tree 近邻搜索效率下降的问题，提出了 MAD-index-sort 策略。结果表明，随着粒子总数 N 的增加，总时间成本 C、距离运算次数 λ_1、λ_2 与一级数据缓存的未命中次数 D_1 均呈现出指数级增大的趋势。对于相同的粒子总数 N，离散度差异的变化对 λ_1、λ_2 的影响相对较小，而对 D_1 影响非常明显，表明 D_1 是影响总时间成本 C 的最关键因素。

本文提出的 MAD-index-sort 策略有效减小了一级数据缓存的未命中次数 D_1，缩短了总时间成本 C；同时，维度越高的布点形状缓存读写次数 D、I 值越大，kd-tree 近邻搜索效率越低。MAD-index-sort 的排序方向能根据粒子集群的离散情况自动改变，具有更强的适应性。

值得注意的是，MAD-index-sort 策略并未改变 kd-tree 近邻搜索流程，距离运算次数 λ_1、λ_2 仍与 Sort Ⅰ、Sort Ⅱ相等，如何减小 λ_1、λ_2，进一步提高 kd-tree 近邻搜索效率有待进一步研究。

参考文献：

[1] Rafiee M, Abbasi M. Pruned Kd-tree: A memory-efficient algorithm for multi-field packet classification[J]. SN Applied Sciences,2019,1(12):1537.

[2] Plaza E,Di G Sigalotti L,Pérez L,et al.Efficiency of particle search methods in smoothed particle hydrodynamics: A comparative study (part Ⅰ) [J]. Progress in Computational Fluid Dynamics,an International Journal,2021,21(1):1.

[3] Guo Zhongzhou, He Zhiqiang, Zhao Wenwen, et al. Efficient mesh deformation and flowfield interpolation method for unstructured mesh[J]. Acta Aeronautica et Astronautica Sinica,2018,39(12):122411.[郭中州,何志强,赵文文,等. 高效非结构网格变形与流场插值方法[J]. 航空学报,2018,39(12):122411.]

[4] Duan Xingfeng, Ren Hongxiang, Shen Helong. Fluid modeling and simulation using CUDA-based weakly compressible SPH[J]. Computer Engineering & Science, 2018, 40(8):1375–1382. [段兴锋,任鸿翔,神和龙. 基于 CUDA 的弱可压 SPH 流体建模与仿真[J]. 计算机工程与科学,2018,40(8):1375–1382.]

[5] Wang Dongpo,Qu Huanan, Shen Wei, et al. Dynamic response analysis for debris flow dam with sediments[J]. Advanced Engineering Sciences,2021,53(2):1–9.[王东坡,瞿华南,沈伟,等. 考虑坝后淤积的泥石流冲击拦挡坝动力响应研究[J]. 工程科学与技术,2021,53(2):1–9.]

[6] Zhao Weiming. Application of automatic analysis of image data based on kd-tree in ray tracing technology[J]. MATEC Web of Conferences,2022,365:01028.

[7] Hu Linjia,Nooshabadi S. High-dimensional image descriptor matching using highly parallel kd-tree construction and approximate nearest neighbor search[J]. Journal of Parallel and Distributed Computing,2019,132:127–140.

[8] Mao Chengying,Zhan Xuzheng,Tse T H,et al.KDFC-ART: A KD-tree approach to enhancing fixed-size-candidate-set Adaptive Random Testing[J]. IEEE Transactions on Reliability,2019,68(4):1444–1469.

[9] Xiang Feng, Li Zhongzhi, Xiong Xi, et al. Inverse distance weight interpolation algorithm based on particle swarm local optimization[J]. Journal of Computer Applications,

2023,43(2):385–390.[向峰,李中志,熊熙,等.粒子群局部优化的反距离权重插值算法[J].计算机应用,2023,43(2):385–390.]

[10] Gidaspov V Y,Morozov A Y,Reviznikov D L.Adaptive interpolation algorithm using TT-decomposition for modeling dynamical systems with interval parameters[J]. Computational Mathematics and Mathematical Physics, 2021, 61(9):1387–1400.

[11] Ma Jing, Li Li. Optimization of distributed K-means algorithm with RDD-based extended index layer[J]. Computer Engineering and Applications,2019,55(1):161–167.[马菁,李力.RDD上扩展索引层优化的分布式K-means算法[J].计算机工程与应用,2019,55(1):161–167.]

[12] Zhang Ting, Wang Zongkai, Lin Zhenhuan, et al. Improved kd-tree particle nearest neighbor search based on automatic termination criterion[J]. Advanced Engineering Sciences,2024,56(6):217–229.[张挺,王宗锴,林震寰,等.基于自动终止准则改进的kd-tree粒子近邻搜索研究[J].工程科学与技术,2024,56(6):217–229.]

[13] Guo Zhongzhou,He Zhiqiang,Xia Chenchao,et al.KD tree method for efficient wall distance computation of mesh[J]. Journal of National University of Defense Technology, 2017,39(4):21–25. [郭中州,何志强,夏陈超,等.高效计算网格壁面距离的KD树方法[J].国防科技大学学报,2017,39(4):21–25.]

[14] Nie Feiping, Xue Jingjing, Wu Danyang, et al. Coordinate descent method for k-means[J].IEEE Transactions on Pattern Analysis and Machine Intelligence,2022,44(5):2371–2385.

[15] Mahmud M P,Wiedenhoeft J,Schliep A.Indel-tolerant read mapping with trinucleotide frequencies using cache-oblivious kd-trees[J].Bioinformatics,2012,28(18):i325–i332.

[16] Ni Yufei,Deng Yangdong,Li Zonghui.Agglomerative memory and thread scheduling for high-performance ray-tracing on GPUs[J].IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems, 2022, 41(2): 334–345.

[17] Shi Zhiyuan,Xu Weiming,Meng Hao.A point cloud simplification algorithm based on weighted feature indexes for 3D scanning sensors[J].Sensors,2022,22(19):7491.

[18] Liang Xiao, Yang Hongyu, Zhang Yanci, et al. Efficient kd-tree construction for ray tracing using ray distribution sampling[J]. Multimedia Tools and Applications,2016,75 (23):15881–15899.

[19] Shi Xiaojing, Liu Tao, Han Xie. Improved Iterative Closest Point (ICP) 3D point cloud registration algorithm based on point cloud filtering and adaptive fireworks for coarse registration[J]. International Journal of Remote Sensing, 2020,41(8):3197–3220.

[20] Choi W S, Oh S Y. Fast nearest neighbor search using approximate cached k-d tree[C]//Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. Vilamoura:IEEE,2012:4524–4529.

[21] Nuchter A,Lingemann K,Hertzberg J.Cached k-d tree search for ICP algorithms[C]//Proceedings of the Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM 2007).Montreal:IEEE,2007:419–426.

[22] Wu Wei, Li Hongping, Su Tianyun, et al. GPU-accelerated SPH fluids surface reconstruction using two-level spatial uniform grids[J]. The Visual Computer,2017,33(11):1429–1442.

[23] Alduán I,Tena A,Otaduy M A. DYVERSO:A versatile multi-phase position-based fluids solution for VFX[J]. Computer Graphics Forum,2017,36(8):32–44.

[24] Tsuzuki S, Aoki T. Effective dynamic load balance using space-filling curves for large-scale SPH simulations on GPU-rich supercomputers[C]//Proceedings of the 2016 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA). Salt Lake City: IEEE, 2016:1–8.

[25] Ihmsen M,Akinci N,Becker M,et al.A parallel SPH implementation on multi-core CPUs[J]. Computer Graphics Forum,2011,30(1):99–112.

[26] Jin Kaizhong, Zhang Xiaojian, Peng Huili. KD-TSS: Accurate method for private spatial decomposition[J]. Journal of Frontiers of Computer Science and Technology,2017,11 (10):1579–1590.[金凯忠,张啸剑,彭慧丽.KD-TSS:精确隐私空间分割方法[J].计算机科学与探索,2017,11(10):1579–1590.]

[27] Yoshida R, Zhang L, Zhang Xu. Tropical principal component analysis and its application to phylogenetics[J]. Bulletin of Mathematical Biology,2019,81(2):568–597.

[28] Verde S, Milani S, Calvagno G. Phylogenetic analysis of software using cache miss statistics[C]//Proceedings of the ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Brighton:IEEE,2019:2552–2556.

[29] Pal A K,Mondal P K,Ghosh A K.High dimensional nearest neighbor classification based on mean absolute differences of inter-point distances[J]. Pattern Recognition Letters,2016,74:1–8.

[30] Koga S, Komikado A, Murayama Y, et al. OL07-1 Time-dependent sensor performance is sustained during 4 years: Accuracy measurement with mean absolute difference (MAD) in continuous glucose monitoring (CGM)[J]. Diabetes Research and Clinical Practice,2016,120:S55.

[31] Klongdee W, Pongkitwitoon M. Forecasting techniques based on absolute difference for small dataset to predict the SET index in Thailand[J].Engineering and Applied Science Research,2018,45(4):308–311.

[32] Zhang Ting,Ren Yufei,Yang Zhiqiang,et al.Generalized finite difference method for non-linear free-surface problems of liquid sloshing[J]. Journal of Sichuan University (Engineering Science Edition),2016,48(1):8–14.[张挺,任聿飞,杨志强,等.基于广义有限差分法求解非线性自由液面的液体晃动问题[J].四川大学学报(工程科学版),2016,48(1):8–14.]

[33] Zhang Ting,Zhang Siqian,Yang Dingying,et al.Numerical investigation on competitive mechanism between internal and external effects of submarine pipeline undergoing vortex-induced vibration[J].Ocean Engineering,2022,266:112744.

[34] Li Jianfeng,Tan Yaohua,Liao Shenghui.Highly parallel SAH-KD-tree construction for raytracing[J].Journal of Hunan University (Natural Sciences), 2018, 45(10): 148–154.[李建锋,谭耀华,廖胜辉.用于光线跟踪的高并行度表面积启发式(SAH)KD树构建[J].湖南大学学报(自然科学版),2018,45(10):148–154.]

Research on kd-tree Cache Optimization Based on Particle Index Sorting Algorithm

ZHANG Ting, *LIN Zhenhuan*, *YANG Dingying**, *WANG Zongkai*, *CHEN Yifan*

(College of Civil Engineering, Fuzhou University, Fuzhou 350116, China)

Abstract:

Objective When utilizing a kd-tree for large-scale random particle nearest neighbor search, particles with closely aligned index values within the computational domain can become spatially distant, resulting in significant variations in the kd-tree search path over a short time frame. Therefore, this divergence reduces the efficiency of accessing node data and ultimately impedes the effectiveness of kd-tree nearest neighbor searches. This inefficiency becomes particularly evident in non-uniform or randomized particle distributions, where traditional cache optimization strategies show limited effectiveness. Maximum dispersion dimensionality reduction in principal component analysis (PCA) is introduced to address this issue. It employs the mean absolute difference (MAD) as the dispersion measure and proposes a novel cache optimization strategy named MAD-index-sort. MAD-index-sort improves cache utilization and enhances the overall performance of kd-tree-based nearest neighbor searches by dynamically reordering particle index values based on spatial distribution.

Methods The MAD-index-sort strategy introduced in this study utilized the dimensionality reduction principles found in PCA to achieve optimal data reordering. Specifically, the method employed MAD as a dispersion measure, which served as the key metric for determining the most appropriate dimension along which the particle data should be reordered. By calculating the dimension with the highest MAD value, identifying the axis along which particle distribution exhibited the greatest variance, the algorithm ensured that particles with closer spatial proximity were assigned similar index values, thus optimizing data access during kd-tree searches. This reordering process, in turn, enabled more efficient cache usage because it minimized cache misses and increased the likelihood of retrieving relevant data from faster memory. The proposed approach was then integrated with the automatic termination criterion for the kd-tree framework, which further streamlined the nearest neighbor search process by automatically halting the search once optimal criteria were met, reducing unnecessary computations. A series of rigorous experimental trials was conducted using various particle distribution models, including uniform, random, and clustered configurations to comprehensively evaluate the effectiveness of this approach. Key performance metrics, such as cache miss rates, search path divergence, and total computational time, were carefully monitored and compared against baseline methods, including Z-index-sort, which was traditionally employed for uniform grid methods, and the standard unsorted kd-tree approach.

Results and Discussions The experimental results revealed that the MAD-index-sort strategy consistently surpassed both traditional unsorted kd-tree methods and alternative sorting strategies such as Z-index-sort across various particle distribution scenarios. Specifically, MAD-index-sort improved search efficiency by dynamically adjusting to particle dispersion characteristics, which led to a marked reduction in cache misses and overall computational time. When applied to randomized particle distributions with high spatial divergence, MAD-index-sort significantly decreased the cache miss rate. Compared to the unsorted kd-tree, the cache miss rate was reduced by 24.2%, while compared to Z-index-sort, the reduction was 18.6%. In addition, the improved performance was particularly evident in computational fluid dynamics (CFD) simulations, where particle positions frequently shifted irregularly due to external forces. In this context, MAD-index-sort dynamically adapted to the shifting particle arrangement, reduced cache miss rates by 20%, and simultaneously cut the total computational time by 15% compared to the unsorted kd-tree method. The algorithm was versatile enough to handle both static and dynamic conditions, making it highly reliable for real-time applications where particle distributions tended to change unpredictably. In addition, the results indicated that MAD-index-sort performed exceptionally well in scenarios involving highly dispersed particle clusters. In these cases, the search time was shortened by 28.5% compared to the unsorted kd-tree

and by 22% compared to Z-index-sort. This efficiency was attributed to MAD-index-sort's ability to reorder particles based on the dimension with the greatest variance, ensuring that spatially close particles were assigned similar index values. Therefore, the cache access pattern was optimized, which led to fewer cache misses and improved search times. In addition, MAD-index-sort demonstrated a significant performance boost, achieving search times 32.8% faster than unsorted kd-tree methods, with a 30% reduction in cache miss rates. These findings indicated that MAD-index-sort was effective in handling irregular particle distributions and further demonstrated its versatility and adaptability. Further analysis revealed that MAD-index-sort consistently improved the cache hit rates across all tested particle distribution models. For example, in scenarios characterized by extreme spatial variance, the strategy attained an average 27.3% reduction in computational time, coupled with a 17% improvement in cache hit rates, compared to traditional methods. This broad applicability of the MAD-index-sort strategy highlighted its effectiveness in a wide range of computational tasks that required efficient data access, particularly in systems where memory hierarchy played a crucial role in performance. Accordingly, the experimental results validated the efficiency and flexibility of the MAD-index-sort strategy, showing that it significantly reduced cache misses and improved search performance across a wide variety of particle distribution scenarios. MAD-index-sort optimized cache utilization and minimized computational overhead by dynamically adjusting the index sorting process based on particle dispersion, making it a practical tool for kd-tree-based nearest neighbor searches in different environments.

Conclusions The introduction of the MAD-index-sort strategy represents a meaningful enhancement in kd-tree-based nearest neighbor search optimization, particularly for scenarios involving random or highly dispersed particle distributions. The strategy improves cache efficiency, reduces computational overhead, and enhances overall search performance by incorporating PCA-inspired dimensionality reduction and using the MAD metric to dynamically reorder particle indices. The experimental results indicate that MAD-index-sort can improve search efficiency by up to 30.3% compared to unsorted kd-tree searches, making it a highly adaptable and versatile tool for a wide range of computational applications. The implications of this research extend beyond kd-tree nearest neighbor searches, as the underlying principles of dynamic index reordering and cache optimization can be applied to other computational problems where data access efficiency is critical. Future work can investigate integrating MAD-index-sort with advanced cache management techniques, such as predictive caching algorithms or hybrid data structures, to enhance performance. In addition, further research on reducing the computational complexity of distance calculations, particularly in high-dimensional spaces, can yield greater efficiency gains and make kd-tree-based methods more suitable for large-scale, real-time applications.

Key words: kd-tree; particle nearest neighbor search; cache optimization; particle index value sorting

（编辑　陈　雪）

引用格式: Zhang Ting,Lin Zhenhuan,Yang Dingying,et al.Research on kd-tree cache optimization based on particle index sorting algorithm[J].Advanced Engineering Sciences,2026,58(1):313–323.[张挺,林震寰,杨丁颖,等.基于粒子索引排序算法的kd-tree缓存优化问题研究[J].工程科学与技术,2026,58(1):313–323.]