第41卷第3期 计 算 力 学 学 报 Vol. 41, No. 3
2024 年 6 月 Chinese Journal of Computational Mechanics June 2024

DOI: 10.7511/jslx20221101002

一种基于莫顿码及镜像编码的平衡八叉树模型

袁 瑶[1], 徐 骏*[1], 顾剑锋[1,2]

(1. 上海交通大学 材料改性与数值模拟研究所, 上海 200240;
2. 上海交通大学 材料基因组联合研究中心, 上海 200240)

摘 要: 在接触分析和动画模拟等网格规模庞大、需要实时更新的应用场景下, 普遍采用莫顿码实现包围盒层次树结构的快速重构。但现有的层次树由于结构平衡性差, 普遍存在搜索效率不稳定的问题, 为此本文在莫顿码法的基础上提出了一种兼顾构建与搜索效率的平衡八叉树模型 BOT 树(Balanced Octree)。设计了镜像编码来保证树的上层节点均有 8 个分支, 且同层树节点所含三角面数之差不超过 1。实际算例表明, BOT 树与现有模型 OIOT 树在 CUDA 并行框架下对比, 构建加速比最高可达 1.29×, 且网格规模越大, BOT 树构建效率的优势越明显。同时, 与 OIOT 树相比 BOT 树的筛除率更高, 在凸体接触和边缘接触算例中加速比分别达到 1.13×和 1.06×。

关键词: 层次包围盒树; 平衡八叉树; cuda 并行框架; 莫顿码
中图分类号: TP183 **文献标志码**: A **文章编号**: 1007-4708(2024)03-0467-07

1 引 言

层次包围盒(BVH)是一种以树形式存储网格信息的经典模型, 广泛应用于接触搜索、碰撞测试及动画模拟等领域。常见的包围盒有 AABB[1]、OBB[2]和 k-dops[3]。如何降低构建与更新包围盒树的时间成本、提高树的搜索效率始终是层次包围盒算法的核心问题。如在隐式接触分析中, 树的节点信息在每个迭代步后都将随着网格几何的改变而更新, 大变形条件下甚至需要重构。故在此类分析中, 基于莫顿码(Morton code)[4]的自底向上构建算法因其高效性得以广泛应用。该方法通过排序[5]来获得节点向上合并的路径, 首先根据中心点或包围盒顶点[6]计算所有三角面的莫顿码; 然后以莫顿码为键值进行排序; 最后借助莫顿码各位数字对三角面序列进行递归的二叉划分[5], 划分所得集合对应包围盒树的各层节点。

该方法操作简单, 但是树节点分布的平衡性与具体模型相关, 容易出现局部分支深度过大的情况, 影响接触搜索的效率。为此, Fernandes 等[7]使用桶排序法合并莫顿码数值相近的叶节点。Vinkler 等[8]将包围盒对角线的长作为莫顿码的第四个计算参数, 使局部区域内尺寸相近的三角面更倾向于合并。此外, Gu 等[9]则针对接触物体具体尺寸设置莫顿码的位数, 使更大物体对应的节点处于树的上层。Apetrei[10]在 Karras[11]的基数树基础上设计了经典的紧密二叉树模型, 通过相邻莫顿码的最高相同位数确定叶节点的共同祖先节点, 有效实现了树节点的连续存储、避免了删除空节点等后续处理。Chitalu 等[12]通过虚拟节点概念提出了隐式二叉树, 保证大部分内部节点具有两个分支, 不仅在构建阶段实现节点信息连续存储, 还有极高的构建速度。该树与文献[10]的加速比最高可达 6×, 但仍不能彻底解决均衡性问题。

为了提高和维持树的平衡性, 本文基于 AABB 包围盒, 设计了一种平衡的八叉树模型(Balanced Octree, 以下简称为 BOT 树), 并提出辅助镜像编码的概念, 使得, (1) 除去叶节点层及其最近层, 所有上层内部节点均有 8 个子节点; (2) 所有兄弟节点包含的三角面数量相差不超过 1。与其他现有模型相比, 该模型在保持良好构建速度的同时最大程度降低了树的总层数, 维持了极佳的平衡性, 非

收稿日期: 2022-11-10; **修改稿收到日期**: 2023-01-05.
基金项目: 国家重点研发计划(2018YFA0702900)资助项目.
作者简介: 徐 骏* (1978-), 男, 博士, 助理研究员(E-mail: xuj@sjtu.edu.cn).

引用本文: 袁 瑶, 徐 骏, 顾剑锋. 一种基于莫顿码及镜像编码的平衡八叉树模型[J]. 计算力学学报, 2024, **41**(3): 467-473.
YUAN Yao, XU Jun, GU Jian-feng. A balanced octree based on morton code and mirror code [J]. *Chinese Journal of Computational Mechanics*, 2024, **41**(3): 467-473.

常适合于以接触检测为目的的并行层级搜索。

2 BOT树构建算法与层级结构特性

2.1 节点信息与存储

平衡八叉树的节点信息包括,(1) 几何参数*AABB*_data,即包围盒顶点三个方向上坐标的最值;(2) 所含三角面个数 n _of_seg;(3)叶节点所含三角面序号或是内部节点首个子节点的存储位置 id_seg。为了节省存储空间,将 BOT 树所有节点信息按层依次连续存储在 *aabb*_ls,n _of_seg_ls 和 id_seg_ls 三个数组中。除此之外,BOT 树还有 k_0,k_1 和 k_2 三个重要参数。其中参数 k_2 的数值由三角面总数 n 决定,用于划分树的各层节点

$$k_2=\lceil \log_8 n \rceil \tag{1}$$

式中 中括号表示向下取整。BOT 树的节点层可分为内部节点层(internal level)、起始层(entry level)和叶节点层(leaf level)。树的第 0 层至第 k_2-1 层为内部节点层,层内每个节点均有 8 个分支,故第 k 层($0\leqslant k\leqslant k_2-1$)有 8^k 个节点,第 i 个节点的首个子节点存储位置 pos 可由式(2)计算获得

$$\text{pos}=8^k+8\cdot i \tag{2}$$

式(1,2)表明 BOT 树的形态与网格的几何形态或拓扑关系基本无关,仅与网格数量有关。

第 k_2 层由叶节点的父节点(或叶节点)构成,衔接了内部节点层和叶节点层。在 BOT 树的自底向上构建算法中,该层与叶节点层属于最先计算的部分,所以命名为起始层。起始层有 8^{k_2} 个节点,每个节点所含三角面的个数(即内部节点的分支数)等于参数 k_1 或 k_1+1。由于每个节点至多有 8 个分支,且该层的下一层为叶节点层,参数 k_1 必满足

$$1\leqslant k_1\leqslant 7 \tag{3}$$

起始层的构建算法为,首先以中心点莫顿码为键值对三角面排序;将三角面均匀划分到 8^{k_2} 个集合中。当三角面数 n 不是 8^{k_2} 的整数倍时,必然存在一些集合比其他集合多出一个三角面的情况;最后令每个起始层节点对应一个集合,依次从序列中提取相应数量的三角面,进行子(叶)节点及自身包围盒信息的计算。在此将有 k_1+1 个三角面的起始层节点称为余节点,参数 k_0 则为余节点的总数,满足

$$k_0<8^{k_2} \tag{4}$$

参数 k_0 和 k_1 可结合式(1,5)计算得到

$$n=k_0+k_1\times 8^{k_2} \tag{5}$$

具体地,当 $k_1>1$,起始层所有节点均为内部节点,第 k_2+1 层有 n 个叶节点,分别对应一个三角面。而当 $k_1=1$ 时,起始层由叶节点和只含两个三角面的余节点构成,第 k_2+1 层只有 $2\cdot k_0$ 个叶节点。

2.2 BOT树起始层构建方法

确定叶节点与其父节点的对应关系是 BOT 树构建中的关键。第 l 个起始层节点首个子节点的位置由该层前 l 个节点的子节点总数决定,也即取决于第 k_2 层中余节点的位置。因此,余节点的位置还影响了 BOT 树上层节点内三角面的平衡分布。本文制定了算法 1 所示的余节点定位方法。

算法 1 平衡八叉树三角面插入算法

输入:八叉树内部节点的 n _of_seg 数据、待插入三角面数 n _insertion
输出:更新后的八叉树内部节点的 n _of_seg 数据
(1) 八叉树的第 0 至 k_2 层的 n _of_seg 数据展开为平衡二叉树
(2) for $i=1$ to n _insertion
(3) node = root//从二叉树根节点 root 开始
(4) node.n _of_seg = node.n _of_seg + 1
(5) for $j=1$ to k_2-1
(6) if (node.leftchild.n _of_seg == node.rightchild.n _of_seg)
(7) node = node.rightchild//进入当前节点的右子节点
(8) node.n _of_seg = node.n _of_seg + 1
(9) else
(10) node = node.leftchild//进入当前节点的左子节点
(11) node.n _of_seg = node.n _of_seg + 1
(12) End if
(13) End for
(14) End for
(15) 二叉树还原为八叉树第 0 至 k_2 层的 n _of_seg 数据

以参数 $k_0=0$,$k_1=2$ 和 $k_2=1$ 的平衡八叉树为例,提取出内部节点的 n _of_seg 数据,再将其展开为图 1 的二叉树。此时向体系依次加入 2 个新的三角面,插入过程中规定二叉树所有右节点的 n _of_seg 数据必须大于等于其左兄弟节点,且两者之差不超过 1。完成插入操作后,将二叉树压缩还原为八叉树即可得到起始层余节点分布。

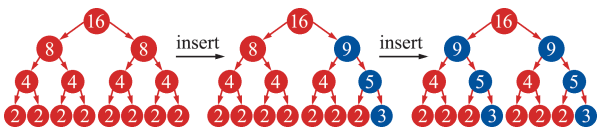


图 1 三角面插入过程
Fig. 1 Process of inserting triangles

类似于 Zhou 等[13]使用莫顿码表示八叉树的节点路径,此处三角面的插入路径(同时也是余节点在第 k_2 层的局部序号)也可用一个整数的二进制格式来记录。令 0 和 1 分别表示左分支和右分支,图 1 中第 2 个三角面的路径为 011,即整数 3。总结上述规则,当参数 $k_1>1$ 时第 2 个三角面插入路径为

$$[011\cdots1]_2=[8^{k_2}/2-1]_{10} \tag{6}$$

式中 下标 10 和 2 表示进制。对于任意三角面,已知其插入序号 i ($i>0$),通过镜像编码(mirror

code）即可求其插入路径。第 k_2 层节点的局部编号 l 满足

$$0 \leqslant l \leqslant 8^{k_2}-1 \tag{7}$$

则编号 l 的二进制格式有 $3 \cdot k_2$ 个有效位数，把 l 各位的数值反向重排，可以得到与之一一对应的数字 l'，称为 l 的镜像编码。在上述插入法则下，插入序号 i 的增加相当于对路径的镜像编码进行降位，故 i 与镜像编码之和为 8^{k_2}。因此可以通过插入序号 i 和镜像编码反推余节点的位置。仍以 $k_2=2$ 时为例，当 $i=2$，镜像编码为 62，反转得到余节点局部编号 31，即

$$[62]_{10}=[111110]_2 \cdot [011111]_2=[31]_{10} \tag{8}$$

式(8)展示的关系在插入序号 $i=1,\cdots,8^{k_2}-1$ 时始终成立。实际构建起始层时，只需计算 $8^{k_2}-1$，$8^{k_2}-2$，…，$8^{k_2}-k_0$ 共 k_0 个数，再进行镜像反转即可得到第 k_2 层中所有余节点的位置。

2.3　BOT 树的并行构建算法

为了进一步提高 BOT 树构建的效率，本文利用 CUDA 11.4 框架设计了基于 GPU 运算的 BOT 树并行构建算法。

算法 2　平衡八叉树并行构建算法

输入：三角面数 n、网格节点坐标以及三角面内节点编号
输出：八叉树参数以及数组 *aabb*_ls、*n*_of_seg_ls 和 id_seg_ls
(1) 并行计算各三角面中心点坐标及其 Morton 码
(2) 以中心点的 Morton 码为键值，对三角形面编号进行排序
(3) 计算八叉树参数、镜像编码以及起始层节点内三角面个数 *n*_of_seg
(4) 计算起始层节点所含首个三角面在排序序列中的位置 first_tri
(5) 调用核函数 kernel_entry_leaf 并行构建起始层、叶节点层；
(6)　for $i=k_2-1$ to 1
(7)　　调用核函数 kernel_internal，并行构建第 i 层
(8)　End for
(9) 计算根节点包围盒信息

其中构建起始层、叶节点层和内部节点层的核函数伪代码分别如算法 3 和算法 4 所示。

算法 3　核函数 kernel_entry_leaf

输入：节点信息数组、三角面序列、起始层的 first_tri、*n*_of_seg 数据
为起始层的所有节点各分配一个线程(thread)，每个线程：
(1) 提取对应节点的 first_tri 数据
(2) 提取对应节点所含三角面个数 *n*_of_seg
(3)　if(*n*_of_seg==1)//当前节点为叶节点
(4)　　通过 first_tri 数据查找对应三角面编号，构建当前叶节点
(5)　　//构建一个节点时需要更新八叉树的三个数组相应位置的信息
(6)　else//当前节点为内部节点
(7)　　for $j=0$ to *n*_of_seg−1
(8)　　通过 first_tri+j 数据查找对应三角面编号，构建叶子节点
(9)　End for
(10)　通过所有叶子节点信息计算当前节点包围盒信息
(11)　End if

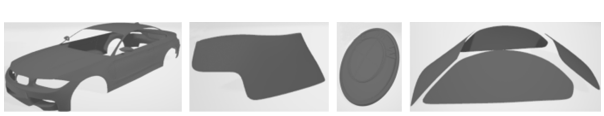
算法 4　核函数 kernel_internal

输入：八叉树的三个节点信息数组、当前层序号 i
为第 i 层的所有节点各分配一个线程(thread)，每个线程：
(1) 通过式(2)计算对应节点的 id_seg 数据
(2)　for $j=0$ to 7
(3)　　通过 id_seg+j 数据查找第 j 个子节点的信息
(4)　End for
(5) 通过所有子节点信息构建当前节点包围盒

多数二叉树的 GPU 并行构建算法仅调用一个核函数，并给每个叶节点分配一个线程以进行父节点的查找和更新。通过原子操作使得多个线程串行地访问修改同一个父节点，最后到达的线程将使用父节点信息继续计算上层节点。但是由于八叉树分支数远大于二叉树，大量使用原子操作会在串行运算的基础上增加线程调度的时间成本。故本文对每层都调用一个核函数，利用核函数之间本身存在的同步机制。包围盒的信息计算则是从父节点向下查找已经完成计算的子节点。并且由于八叉树节点信息是连续存储的，串行访问子节点不会因为跳转和乱序读取而导致额外的时间增加。

2.4　BOT 树的层级结构特性

对 BOT 树进行搜索操作时，因具体算法而异，树节点包围盒的相交判断对象可以是来自同一或不同 BOT 树的包围盒，也可以是网格节点、棱边等接触元素的包围盒。但是基于 BOT 树的算法均满足若上层节点与搜索对象不相交，则其子代节点都不再与该对象进行接触判断。节点所在层越接近根节点，则其可以避免的无效接触判断越多，对搜索效率的提升作用越明显。虽然其他八叉树也具有层级结构，但是其节点分布平衡性较为随机，一旦存在局部分支深度大于其他部分，层级结构的作用便会减弱。局部分支深度越大，其实质越接近于线性结构。与之相比，BOT 树的层级结构是固定的，可以充分发挥树结构筛除无效搜索元素的特性。



(a) 车架 (a) Carshell　(b) 车灯 (b) Taillight　(c) 车标 (c) Roundel　(d) 车窗 (d) Window

图 2　汽车模型
Fig. 2　Car model

为了分析 BOT 树层级结构对搜索效率的影响，将其与典型的线性算法桶排序法进行对比。从一个汽车模型的车架部分提取出 99488 个三角面构建 BOT 树，再从其车灯部分提取 3835 个节点进行桶

排序划分，每个方向上的桶数为10。将桶作为AABB包围盒与BOT树进行深度优先搜索，最后将桶中所含的节点与相交的BOT树叶节点包围盒进行判断（以下简称BOT-桶排序方法）。结果显示，桶和节点包围盒分别与BOT树包围盒进行的相交判断次数总量为33313和105460。同时使用战修广[14]的点找面式桶排序搜索法作为对比，总计进行了179689次节点包围盒与三角面包围盒的相交判断，比BOT-桶排序方法的总和多出28.5%。

并且从图3可以看出，使用BOT-桶排序方法时，单个节点包围盒参与判断的次数也明显普遍低于传统桶排序方法。以上结果说明BOT-桶排序方法在搜索时比传统桶排序法更高效，充分体现了层级结构对搜索速度的提升作用。考虑到桶排序与BOT树更新数据的计算复杂度分别与接触元素总量 m 及其对数 $\log_8 m$ 线性相关，若结合数据初始化和更新的计算成本，BOT树的优势将更加显著。

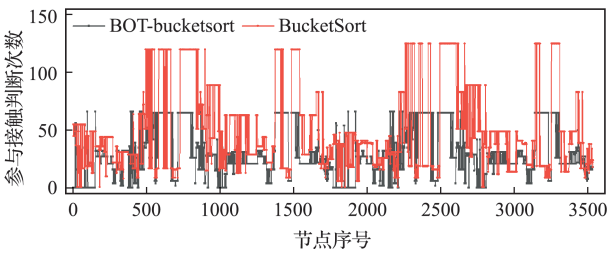


图3 串行算例中各节点参与接触判断的次数

Fig. 3 Number of overlapping test of each node in the serial test

3 实验与对比

3.1 BOT树的并行构建时间及对比

为了研究BOT树并行构建算法的实际效率，将Chitalu等[12]提出的二叉树模型推广为八叉树（Ostensibly Implicit Octrees，以下简称OIOT树）进行对比。同时为了避免偶然性，采用了多种不同规模和形状的网格模型（图4）。各模型的八叉树信息列入表1。

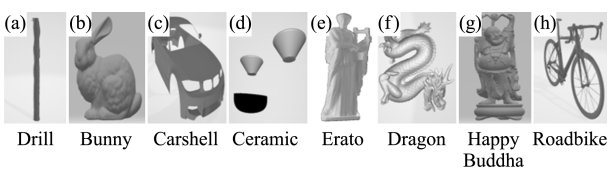


图4 模型3D图

Fig. 4 3D figure of each model

两种树模型构建过程的耗时情况如图5所示。可以看出，两种树前三层的构建时间基本一致。这是因为前3层调用的线程总数至多为512，所有线程实际上都能同时开启，各层处理时间主要与单个线程的计算耗时相关。其中模型drill的网格数量最少，第3层即为起始层，除了内部节点还需要进行叶节点的计算，耗时反而最多。对于第4层至第6层，两种树的计算时间差异以及同种树各层之间的差异逐渐增大。因为此时的线程规模极大，最多可达262144。这些线程无法同时开启，而是分成若干批依次运行。故实际计算时间由单个线程计算量和各层节点个数共同决定。同时，单个线程计算量又与该层节点最大分支数相关。以起始层的构建为例，除了最后一个节点，OIOT树起始层节点的分支数均为8，故单个线程计算的包围盒个数为固定值9。

表1 各模型BOT树参数及八叉树节点数

Tab. 1 Parameters of BOT and node number of octree

模型	三角面数	BOT树参数			节点总数	
		参数 k_0	参数 k_1	参数 k_2	BOT树	OIOT树
a	3855	271	7	3	4440	4407
b	69451	3915	2	5	106900	79376
c	99488	1184	3	5	136937	113704
d	240632	11256	7	5	278081	275009
e	412669	150525	1	6	712262	471624
f	871414	84982	3	6	1171007	995904
g	1087716	39140	4	6	1387309	1243108
h	1677520	104656	6	6	1977113	1917169

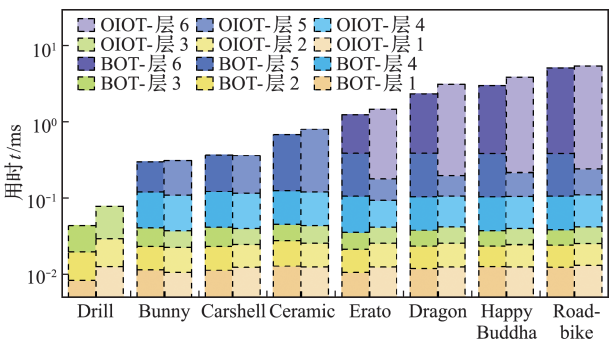


图5 BOT树和OIOT树各层的构建时间

Fig. 5 Construction time of each level of BOT and OIOT

如图6所示，OIOT树起始层构建时间与节点数基本呈线性关系（图中拟合均方差分别为0.9979和0.9985）。反之，当参数 k_2 一定时，BOT树起始层的节点数不变，总计算时间由该层最大分支数，

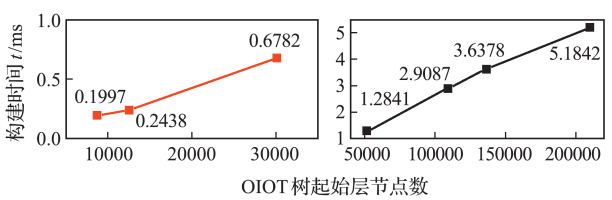


图6 OIOT树起始层节点数与起始层构建核函数运行时间的曲线

Fig. 6 Number of OIOT entry node vs running time of kernel_entry_leaf

即参数 k_1 决定。如图 7 所示，构建时间与参数 k_1 也基本呈线性关系，拟合均方差分别为 0.9991 和 0.9926。

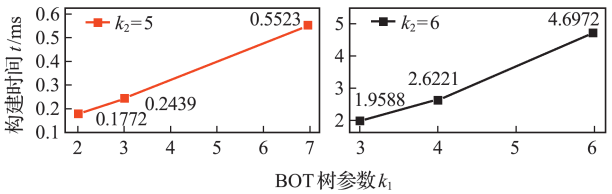


图 7　BOT 树参数 k_1 与起始层构建核函数运行时间的曲线

Fig. 7　Parameter k_1 of BOT vs running time of kernel_entry_leaf

最终，各模型的两种八叉树的预处理时间、各层构建总时间以及 BOT 树相对于 OIOT 树的加速比如图 8 所示。其中 BOT 树的预处理内容包含镜像编码的计算及相关数组的处理。结合图 5 和图 8，各模型用于 BOT 树各层构建的时间之和均少于 OIOT 树。但是在网格规模较小的情况下，BOT 树的预处理时间更多，产生了较长的构建总时间。综上，三角面总数小于 10^5 时，BOT 树相对于 OIOT 树的加速比均略小于 1，构建效率略逊。但当网格规模较大时（10^5 及以上），BOT 树则显现出更高的构建效率。鉴于 OIOT 树一向以构建速度快而著称，因此可以认为 BOT 树同样具有十分突出的构建效率。

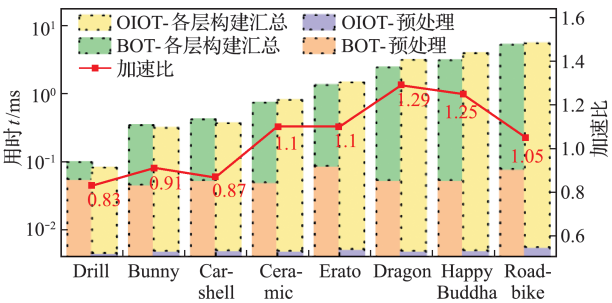


图 8　各模型的预处理时间及各层构建时间之和

Fig. 8　Preprocess time and sum of construction time of each model

3.2　BOT 树的接触搜索时间及对比

3.2.1　并行接触搜索方法

本文使用 Chitalu 等[15]的并行搜索方法，对相同算例分别基于 BOT 树和 OIOT 树逐层开展接触搜索。将一对待判定包围盒的存储序号合并称为包围盒对（bounding box pair，以下简称 BBP）。若一对包围盒相交，则将两者的子代包围盒或本身组合得到子包围盒对 CBBP。并行接触搜索的核函数输入上一轮的 CBBP 序列，为其中每个 BBP 分配一个线程，分别进行接触判定、统计并写入新的 CBBP。第一轮的输入 CBBP 序列通过组合八叉树第 1 层所有节点得到，而最后一轮所得的 CBBP 序列将用于精细接触判断。此处将一轮搜索中单个 BBP 最多能产生的 CBBP 数定义为搜索扩展率，描述搜索量的增长速率，各种情况下两类八叉树的搜索扩展率如表 2 所示。其中默认树 1 的总层数大于等于树 2，k_1^1 和 k_1^2 分别表示两树的参数 k_1。

表 2　BOT 树及 OIOT 树的搜索扩展率

Tab. 2　Search expansion rate of BOT and OIOT

	树 1 当前层	树 2 当前层	BOT	OIOT
1	内部节点层	内部节点层	64	64
2	起始层	起始层	$(k_1^1+1)\cdot(k_1^2+1)$	64
3	内部节点层	起始层	$8\cdot(k_1^2+1)$	64
4	内部节点层	叶节点层	8	8
5	起始层	叶节点层	k_1^1+1	8

每轮的搜索扩展率直接影响了单个线程的计算量、运行时间以及下一轮的线程总数。因为具有更好的平衡结构，BOT 树在起始层和叶节点层的搜索扩展率显著小于 OIOT 树，相应的搜索计算量也明显更少。并且由于这两层的搜索规模在搜索总量中的占比非常大，BOT 树的搜索总时间将会更短。

另外，在内部节点层的搜索阶段，BOT 树实际发生接触的 BBP 个数在待测 BBP 总数中的比例（在此定义为接触率）比 OIOT 树更低。因此 BOT 树能更快地缩小搜索范围，并在一定程度上弥补搜索初期待测 BBP 绝对数量大的劣势。

3.2.2　接触搜索算例基本信息

接触分析中，任意复杂的接触系统均可分解为若干个简单的二体系统。本文从图 2（a，c，d）中汽车模型的三个部分分别提取了一套粗糙三角面集合，组合为两个二体接触算例来验证 BOT 树的搜索效率。各集合的网格尺寸及八叉树相关信息列入表 3。其中尺寸为三角面各顶点到重心的平均距离，由于来自同一原始模型，长度单位省去。下面将集合 1 与集合 2、集合 3 分别组成算例 1 和算例 2，对应二体接触中最典型的凸体接触以及两套网格在边缘处相接的极端情况（以下简称边缘接触）。

表 3　三角面集合的网格尺寸及八叉树信息

Tab. 3　Mesh size and octree information of sets

集合	平均尺寸	BOT 树参数			节点总数	
		参数 k_0	参数 k_1	参数 k_2	BOT 树	OIOT 树
1	19.4264	1184	3	5	136937	113704
2	1.7349	544	1	4	5769	5306
3	27.4887	480	3	3	2601	2305

3.2.3　算例 1 接触搜索结果

对算例 1 进行 6 轮搜索，每轮待测的 BBP 个数、最终判定为发生接触的 BBP 个数列入表 4，每

轮的搜索时间具体如图 9 所示。

表 4 算例 1 各层搜索结果

Tab. 4 Search result of each level in test 1

	待测 BBP 个数		判为接触 BBP 个数		接触率	
	BOT	OIOT	BOT	OIOT	BOT	OIOT
1	64	8	56	6	0.875	0.75
2	3584	240	1791	160	0.500	0.667
3	114624	9344	24759	3363	0.216	0.360
4	1584576	213984	286514	38625	0.181	0.181
5	2611832	2472000	446927	439505	0.171	0.178
6	1352010	3516040	199453	199453	0.148	0.057

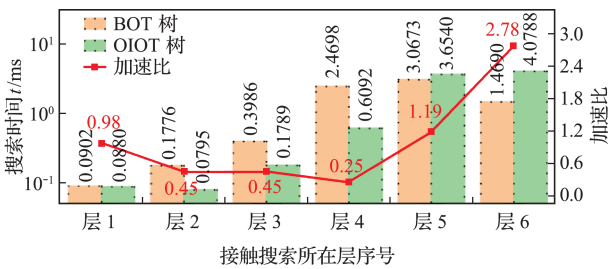


图 9 算例 1 各层接触搜索用时及加速比

Fig. 9 Contact-searching time and speedup of each level of test 1

集合 2 的八叉树高度比集合 1 低，起始层为第 4 层。在第 1 至 4 层的测试中，BOT 树每层待测的 BBP、判定为接触的 BBP 的个数均多于 OIOT 树，检测用时也更长。而在第四轮搜索中，BOT 树的搜索扩展率降为 16，远小于 OIOT 树的 64，因此两者的待测 BBP 数之比从第 4 层的 7.41 降为第 5 层的 1.06。并且由于第 5 层为集合 1 的起始层，BOT 树的搜索扩展率继续减小为 4，而 OIOT 树则为 8，单个线程的运行时间更长。因此在 BOT 树中实际接触的 BBP 数比 OIOT 树多出 1.69%的情况下，第 5 层的运行时间却少了 16.1%。而在叶子节点层，BOT 树待测 BBP 数仅为 OIOT 树的 38.5%，单层加速比达到了最大值 2.78×。

由表 4 可知，BOT 树在第 1 层的接触率高于 OIOT 树，这是因为集合 2 的三角面平均尺寸仅为集合 1 的 9%，与其上层包围盒有极高的重叠性。在此基础上 BOT 树的接触率仍在第 2、3 层迅速下降，并低于 OIOT 树，充分体现了 BOT 树的优势。在第 6 层的搜索中，BOT 树的待测 BBP 数在搜索扩展率的影响下小于 OIOT 树，接触率必然更大。

综合 6 轮搜索结果，使用 BOT 树搜索的总运行时间为 7.6724 ms，少于 OIOT 树的 8.6885 ms。在网格尺寸差异大、重叠性高这一不利于发挥优势的情况下，BOT 树仍取得了总加速比 1.13×。

3.2.4 算例 2 接触搜索结果

算例 2 的 6 轮搜索中，待测的 BBP、发生接触的 BBP 个数以及接触率列入表 5，各轮的搜索用时和加速比如图 10 所示。

表 5 算例 2 各层搜索结果

Tab. 5 Search result of each level in test 2

	待测 BBP 个数		判为接触 BBP 个数		接触率	
	BOT	OIOT	BOT	OIOT	BOT	OIOT
1	64	16	64	16	1.000	1.000
2	4096	800	2420	681	0.591	0.851
3	154880	41988	34874	12997	0.225	0.310
4	1098016	827208	249801	173300	0.228	0.209
5	1998408	1384596	601973	384312	0.301	0.278
6	1831511	3074496	1033110	1033110	0.564	0.336

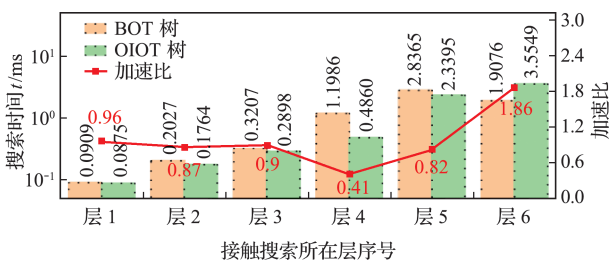


图 10 算例 2 各层接触搜索用时及加速比

Fig. 10 Contact-searching time and speedup of each level of test 2

由于该算例模拟边缘接触情况，集合 3 的网格几乎完全嵌在集合 1 的内部，两集合八叉树上层的重叠程度也非常大，BOT 树和 OIOT 树在第 1 层的接触率同时为 1。在第 2 层至第 3 层，BOT 树的接触率明显低于 OIOT 树，但是待测 BBP 数是 OIOT 树的 3.7 倍至 5.1 倍，实际的搜索计算量更大。由表 5 和图 10 可见，在到达集合 1 的起始层之前，即第 1 层至第 5 层搜索中，BOT 树的待测 BBP 个数和搜索用时始终超过 OIOT 树。最终在搜索扩展率减小为 4 之后，BOT 树在第 6 层的待测 BBP 数降为 OIOT 树的 38.5%，用时仅占后者的 53.7%。综上，BOT 树和 OIOT 树的搜索总用时分别为 6.5570 ms 和 6.9341 ms，BOT 树取得了 1.06×的总加速比。

4 结 论

本文设计了一种基于 AABB 包围盒的平衡八叉树(Balanced Octree)模型，并引入 GPU 并行机制以加速其构建效率。与表面隐式八叉树对比发现：

(1) 在网格规模较小的条件下(10^5 以下)，BOT 树的构建时间与表面隐式八叉树(OIOT 树)基本相当；而网格规模较大时(10^5 及以上)，BOT 树模型具有更高的构建效率，且规模越大，效率优势越显著。

(2) 在并行接触搜索方面，BOT 树因起始层节点的分支数小于 8 的特性，与 OIOT 树相比接触率更小，且搜索范围缩减速度更快。在凸体接触和边缘接触算例中，BOT 树相比 OIOT 树均取得了

更短的耗时，加速比分别达到 1.13× 和 1.06×。值得注意的是，BOT 树低接触率的优势在凸体接触算例中受到了抑制。推测在模型网格尺寸较为相近时，BOT 树在接触搜索中的优越性更显著。

参考文献(References)：

[1] van den Bergen G. Efficient collision detection of complex deformable models using AABB trees[J]. *Journal of Graphics Tools*, 1997, **2**(4): 1-13.

[2] Gottschalk S, Lin M C, Manocha D. OBBTree: A hierarchical structure for rapid interference detection [A]. Proceedings of 23rd Annual Conference on Computer Graphics and Interactive Techniques[C]. 1996.

[3] Klosowski J T, Held M, Mitchell J S B, et al. Efficient collision detection using bounding volume hierarchies of k-DOPs[J]. *IEEE Transactions on Visua-lization and Computer Graphics*, 1998, **4**(1): 21-36.

[4] Morton G. *A Computer Oriented Geodetic Data Base and A New Technique in File Sequencing*[M]. Hinneburg: International Business Machines Com-pany, 1966.

[5] Lauterbach C, Garland M, Sengupta S, et al. Fast BVH construction on GPUs[J]. *Computer Graphics Forum*, 2009, **28**(2): 375-384.

[6] 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.

[7] Fernandes I B, Walter M. A bucket LBVH construction and traversal algorithm for volumetric sparse data[A]. 33rd SIBGRAPI Conference on Graphics, Patterns and Images[C]. 2020.

[8] Vinkler M, Bittner J, Havran V. Extended morton codes for high performance bounding volume hierarchy construction [A]. High-Performance Graphics (HPG) Conference[C]. 2017.

[9] Gu F, Jendersie J, Grosch T. Fast and dynamic construction of bounding volume hierarchies based on loose octrees [A]. Proceedings of the Conference on Vision, Modeling, and Visualization[C]. 2018.

[10] Apetrei C. Fast and simple agglomerative LBVH construction[A]. Computer Graphics and Visual Computing[C]. 2014.

[11] Karras T. Maximizing parallelism in the construction of BVHs, octrees, and *k-d* trees[A]. Proceedings of the Fourth ACM SIGGRAPH/Eurographics Conference on High-Performance Graphics[C]. 2012.

[12] Chitalu F M, Dubach C, Komura T. Binary ostensibly-implicit trees for fast collision detection[J]. *Computer Graphics Forum*, 2020, **39**(2): 509-521.

[13] 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.

[14] 战修广. 基于 SiPESC. FEM 的接触搜索算法研究及软件开发[D]. 大连理工大学, 2021. (ZHAN Xiuguang. Research on Contact Search Algorithm and Software Development Based on SiPESC. FEM[D]. Dalian University of Technology, 2021. (in Chinese))

[15] Chitalu F M, Dubach C, Komura T. Bulk-synchronous parallel simultaneous BVH traversal for collision detection on GPUs [A]. Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games[C]. 2018.

A balanced octree based on morton code and mirror code

YUAN Yao[1], XU Jun*[1], GU Jian-feng[1,2]

(1. Institute of Materials Modification and Modeling, Shanghai Jiao Tong University, Shanghai 200240, China;
2. Materials Genome Initiative Center, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: In contact analysis and animation simulation, there are massive and frequent re-mesh operations. A series of algorithms based on Morton code method have been established to carry out fast construction of BVH trees. Unfortunately, search efficiency of BVH trees constructed by these algorithms is unstable due to their unbalance. Therefore, an alternative algorithm of the balanced octree (BOT) based on Morton code, targeting both fast construction and stable search efficiency, is proposed in this paper. A mirror code method is designed to ensure that each upper node of BOT contains 8 branches and the amount difference of triangle units belonging to those nodes on the same tree level is no larger than one. Experiments showed that, the parallel construction of BOT achieved 1.29× speedup over that of the existing OIOT under the CUDA framework. In addition, construction efficiency of BOT increase with larger mesh size. Meanwhile, a BOT tree has a higher filter rate. In parallel contact searching, it has achieved 1.13× and 1.06× speedup over OIOT in convex-contact and edge-contact cases, respectively.

Key words: BVH; balanced octree; cuda; Morton code