文章编号:1674-2974(2018)10-0148-07 DOI:10.16339/j.cnki.hdxbzkb.2018.10.020

用于光线跟踪的高并行度表面积启发式(SAH)KD树构建*

李建锋[1,2†],谭耀华[1],廖胜辉[1]
(1.中南大学 信息科学与工程学院,湖南 长沙 410083;2.吉首大学 信息科学与工程学院,湖南 吉首 416000)

摘 要:提出一种用于光线跟踪的SAH-KD树构建方法,解决当前KD树并行算法并行度不高且效率低的问题.算法首先对所有图元包围盒在三个维度按坐标轴左值排序,得到三维上有序的包围盒索引.然后使用层次遍历构建KD树,根据每个节点包围盒选择要划分的维度,并在当前层生成所有节点在该维度下的候选划分点序列.最后计算每个节点的空间树,在GPU中计算每个候选点的SAH值,选择每个节点的最小SAH值点进行划分.实验中采用4个常用场景进行测试算法性能,并同时比较了当前高效串行与并行算法,结果证明本文提出的算法在生成同等质量KD树的情况下达到对比串行方法4~6倍以及对比并行方法的1.3~1.5倍的计算速度,并且能在线程数成倍增加时达到相近倍数的加速比.

关键词:SAH-KD树;空间树;并行计算
中图分类号:TP309.7 **文献标志码**:A

Highly Parallel SAH-KD-tree Construction for Raytracing

LI Jianfeng[1,2†], TAN Yaohua[1], LIAO Shenghui[1]
(1. School of Information Science and Engineering, Central South University, Changsha 410083, China;
2. College of Information Science and Engineering, Jishou University, Jishou 416000, China)

Abstract: This paper proposed a SAH-KD tree construction method for ray tracing to solve the problem of low parallel degree and low efficiency in the existing algorithms. The algorithm first obtains the ordered index on the three latitudes by sorting the primitive bounding boxes according to the left value. Then, according to the node bounding box, we choose the dimension to be divided and generate the candidate partition points of each node under this dimension. Finally, the SAH value of each candidate point was calculated based on the spatial tree in GPU, and the minimum SAH value of each node was selected for partitioning. Four common scenarios were used for testing performance of the algorithm, and compared the efficient serial and parallel GPU algorithm. The results show that the proposed algorithm can achieve 4~6 and 1.3~1.5 times faster than the contrast method when generating the same quality KD tree. When the number of GPU cores fold increases, the speedup ratio reaches a near multiple.

Key words: SAH-KD-tree; space tree; parallel computing

* **收稿日期**:2017-11-27
基金项目:国家自然科学基金资助项目(61562029),National Natural Science Foundation of China(61562029);湘西州科技计划项目(2018SF5013)
作者简介:李建锋(1979—),男,土家族,张家界市人,吉首大学教授,博士
† 通讯联系人,E-mail:7684557@qq.com

K 维树(KD 树)是一类二进制分割树,作为加速结构广泛地应用于图形与查询领域,如光线跟踪中图元与光线的相交检测[1-2]和最近邻查询等. 为了高效遍历 KD 树,通常在构建时使用表面积启发式划分法(SAH),通过最小化每个节点的期望遍历成本选择划分点. 如何快速高效地构造 SAH-KD 树一直是研究重点,主要研究方向集中在两种方法上.

一是串行构造方法,最简单的思路是通过全遍历计算 SAH-KD 树所有划分点的 SAH 值,其计算复杂度为 $O(N2)$,由于计算复杂度较高,使得该方法无法应用于大场景. 针对该问题,文献[3]和[4]提出将划分点进行预排序,通过迭代的方法依次求出每个划分点的 SAH 值,把每个节点的时间复杂度降低到 $O(N\log N)$,从而使得整棵树的构建复杂度降低至 $O(N\log 2N)$. Wald 和 HaVRan 在文献[3]和文献[5]的基础上,提出只在根节点排序,非根节点通过遍历其父节点序列得到有序序列,使得整棵树的构建复杂度降低至 $O(N\log N)$[5],此时已经达到了 SAH-KD 树串行构建的理论下限. 文献[6]提出一种自调整参数的通用方法,通过随机搜索来得到满足条件的划分点. 该方法虽然可以提高构造速度,但无法得到稳定的结果.

二是并行化构建方法,文献[7]和[8]提出在上层大节点通过一个高性能线程计算,直到每个节点计算量足够小,才将它们分配到多核系统中的构造. 具体实践中,在 4 核 CPU 系统中能达到单核构造 2.5 倍的速度[9]. Shevtsov 等[10]通过并行方法计算中位图元取代 SAH 的计算来寻找上层的大节点的划分点,虽然在遍历性能较 SAH 方法有 15%的退化,但该方法应用在 Larrabee[11]的 4 路并行光线追踪器时达到 3.9 倍的构造与遍历的总性能. 文献[12]先把图元进行层次分组,之后按组并行构建 SAH-KD 树,在层次分明的场景中,12 线程下性能最好时达到文献[4]中算法的 130 倍. 该方法虽然平分了各个线程的计算量,但对于层次不明确的场景无法达到理想效果. 自从 CUDA 与 OPENCL 等并行架构提出来后,GPU 上 SAH-KD 树的构建得到了更多的研究与改进,Kun Zhou[13]提出在大节点直接使用图元的中点划分,对于小节点则进行多路 GPU 构建. 该方法在 128 线程上达到了单核构建的 6～15 倍性能,在 252K 个三角形的场景下达到了 16 线程的 3～6 倍. 该方法虽然通过顶层退化提高了 SAH-KD 树的构建效率,但在超大型场景遍历时有不同程度的性能损失. 文献[14]在前者的基础上加入了 CUDA 自带的 reduction 和原子操作,在实际应用中构建效率得到了部分提升,但依然存在顶层退化的问题.

本文提出一种基于 CPU/GPU 混合编程的 SAH-KD 树构建方法. 该方法使用宽度优先搜索(BFS)构建 SAH-KD 树,充分利用 GPU 多核的特性,并在每层构建时基于当前层次所有候选划分点同时并行计算,减少内存与显存间的拷贝次数,提高构建效率. 同时使用空间树高效计算 SAH 值,在整体构建上达到较高的性能.

1　表面积启发式划分法

表面积启发式划分法(Surface Area Heuristic:SAH)[7,15]相比于中点划分法,得到的结果遍历时更稳定,平均时间复杂度更低. SAH 使用图元包围盒加速计算,其方式有很多种,比如最小包围球,包围球的好处是在进行查询时计算简单,但是球体占用的空间体积太大,光线会穿过大量无关的包围球,因此在应用中并不常见. 作为计算性能与查询性能的折中,通常使用 AABB 包围盒,其特点是可以快速求出各个图元与节点的包围盒,且在空间中不需要占用太大体积. 每个节点的期望遍历成本为一根任意分布的直线,穿过该节点包围盒的概率乘以该节点上图元数. 根据上面的描述给出遍历成本的具体描述:对于任意分布的直线,该直线穿越某凸空间几何体的子区域的概率[16]为:

$$P[V_S \mid V] = \frac{\mathrm{SA}(V_S)}{\mathrm{SA}(V)} \tag{1}$$

其中 $\mathrm{SA}(V)$表示凸几何体 V 的表面积,于是在划分点 p 处的 SAH 值为该节点的遍历成本与其两个子节点 SAH 值之和:

$$\mathrm{SAH}_V(p) = K_t + P[V_L \mid V]\mathrm{SAH}_{V_L}(p) + P[V_R \mid V]\mathrm{SAH}_{V_R}(p) \tag{2}$$

其中左式表示几何体 V 在 p 点划分时的 SAH 值,K_t 表示光线遍历非叶节点所产生的计算成本.

1.1　局部贪婪的 SAH 估计

根据公式(2)可知,如果要精确计算一个节点的最优 SAH 值,必须通过后序遍历计算出它子节点的 SAH 值. 但是后序遍历需要计算所有可能的划

分情况,当场景中图元的数量增加时,可行划分的数量也会呈指数级别增加,所以寻求全局最优解至今还是不可行的.一种解决方案是通过估计来求出局部最优解,用对应节点的图元数来代替上述 K_t 和子节点的 SAH 值,这种方法称为局部贪婪的 SAH.当给出划分点时,我们可以计算出子空间包含的图元数,于是算法可以改成先根遍历,算出每个节点上的最优划分,再对其子节点进行计算,即每次一划分都是局部最优的.由此,将更改后的公式(2)可以化简为:

$$\mathrm{SAH}_V(p) = N_{VL}(p)\mathrm{SA}(V_L) + N_{VR}(p)\mathrm{SA}(V_R) \tag{3}$$

其中 N_{VL} 和 N_{VR} 表示节点在 p 点划分点后其左右子空间中包含的图元数.这种改进将指数级的算法降低至多项式级别的算法,虽然无法求出全局最优解,但是在实际应用中效果良好.

1.2 候选划分点序列

SAH 在节点包围盒空间范围内选择划分点,在连续空间中,理论上存在无限多的划分点.由于没有好的数值方法逼近极值,一般是通过采样的方式来生成划分点候选序列.当采样频率很大时,相邻两个采样点左右子空间包含的图元数通常不会变,改变的仅仅是两边的表面积,如公式(4)所示:

$$\mathrm{SAH}_V(p) = N_{VL}\mathrm{SA}(V_L) + N_{VR}\mathrm{SA}(V_R) \tag{4}$$

其中 N_{VL} 和 N_{VR} 分别为位于划分点左右的图元数.根据公式(4),由于相邻的采样点 N_{VL} 和 N_{VR} 不变,且两子空间表面积相加为常数,其极值必然存在于左右子空间图元数改变的划分点上.由于 SAH-KD 树可能在三维中任何一个维度进行划分,一种简单的采样方法是按维度用节点上图元包围盒的两个端点生成候选划分点序列.

2 并行 SAH-KD 树构造

并行化 SAH-KD 树的构造,现有的并行方式都是基于单个节点,每个线程串行处理一个节点,并单独使用一个线程进行同步.由于上层节点的巨大计算量,通常会使用大小节点的方式分别计算:对于大节点,使用一种退化的方式划分,比如空间中点划分以及图元中点划分等,对于小结点,则通过上述高效的串行方法.理想的并行方法需要基于候选划分点,这样可以实现较高的并行度和每个线程极低的负载.

为了设计一棵基于候选划分点并行的 SAH-KD 树,必须解决两个问题.第一是合理的同步机制.由于 SAH 值和空间树的计算分别位于 GPU 和 CPU 中,因此计算前后的同步工作变得尤为重要.高效的同步方案应该减少空间分配以及内存与显存间互相拷贝的次数.第二是每个线程使用高效的线面相交检测的方法,由于 GPU 单核性能并不突出,如果直接使用暴力算法,在大场景中效率会十分低下,从而影响整体的构造性能.

本文提出的方法大致思想如下:首先生成一层每个节点在三个维度上图元包围盒的有序索引,然后根据每个节点包围盒选择一个维度进行划分,生成每个节点在该维度下候选划分点序列,最后一次性在 GPU 中并行计算一层每个划分点的 SAH 值,并选择每个节点最优的 SAH 值的点进行划分.

2.1 算法步骤

本文方法的算法步骤如下.

1)预处理:在 SAH-KD 树的构建之前,对根节点图元进行预处理,其中包括在 CPU 中计算图元和根节点的 AABB 包围盒,之后在三个维度按包围盒左值的升序排列所有图元包围盒.

2)按层次递归构建 KD 树,对于该层的每个节点:

(a)数据准备:在 CPU 中根据节点包围盒选择划分的维度,生成该维度下的候选划分点序列和空间树,并传入 GPU.

(b)计算 SAH 值:在 GPU 中利用空间树并行计算出每个划分点的 SAH 值并传回内存.

(c)图元指派:在 CPU 中选出最小 SAH 值对应的划分点,生成两个新节点,在三个维度将图元划入对应的节点中,得到新节点的有序图元包围盒.

2.2 计算候选划分点的 SAH 值

计算候选划分点的 SAH 值主要是为每个节点选出最优划分点.由于 KD 树每次只选择一个维度进行划分,所以第一步是找出要进行划分的维度.为了简便计算,通常对当前节点包围盒进行分析,选择包围盒最长的维度作为要划分的维度.

计算 SAH 值时需要统计划分点左右两侧图元数量,本文算法通过空间树来加速这一过程.空间树通过有序图元包围盒构造:有序序列可以看成一棵平衡二叉树,而空间树是对平衡二叉树加入一个副

值来二次判断. 以划分 X 轴为例,包围盒有两个坐标值 Xleft 和 Xright,用来代表包围盒左右两端,此时加入一个辅助值 T,其定义为:

1)如果该节点是叶节点,T 定义为 Xright;

2)如果该节点是非叶节点,T 定义为两个子节点的 T 值与自己 Xright 的最大值.

很显然 T 值需要后序自底向上遍历,使用有序序列转换为平衡二叉树的算法实现 T 值的计算. 为了节省函数调用的开销,本文使用非递归的方式实现. 计算出空间树后,每个候选划分点都需要遍历相应节点的空间树求出其 SAH 值,主要复杂度在计算 NL、NR 和 NP 上,其中 NL,NR,NP 分别为位于候选划分点左,右以及被候选划分点穿过的图元数. 过程中对于坐标为 pos 的候选划分点有:

1)如果当前节点的 T 值小于 pos,则从 left 到当前节点都在划分点的左边,将 NL 加上当前区间图元的个数,并计算其右子节点.

2)如果当前节点的 Xleft 大于 pos,则位于该节点右边的节点都在划分点的右边,将 NR 置为当前节点序号,并计算其左子节点.

3)否则,判断当前节点的图元包围盒与 pos 的位置关系,相应地修改 NL 和 NP,并遍历其左右子节点.

4)得到 NL,NR,NP 后计算出 SAH 值.

上述空间树的遍历方法需要使用辅助栈,然而如果每个线程都使用辅助内存,在大场景下无法正常进行计算. 在实际操作中可以使用二叉搜索树的思想进行改造,在构造空间树时得到节点遍历的顺序并设立一个状态标识,不但可以节省显存开销,同时使各个线程计算更加高效.

2.3　图元指派

SAH-KD 树只在叶节点保存图元,因此需要把非叶节点中的图元指派到两个子节点中. 在并行计算出所有的 SAH 值后,使用 CPU 进行同步,顺序找出每个节点最小 SAH 值对应的划分点,以及其左右子节点包含的图元数 NL 和 NR,并生成两个新节点. 在该过程中,我们需要判断新生成的节点是否需要继续划分:通过设置一个阈值,当 NL 或 NR 小于阈值时,对应子节点将不会再进行划分且复制一份属于它的图元. 由于本文算法一次处理 SAH-KD 树一层的所有节点,需要一次性对所有需要划分的新节点分配空间. 将不需要再分裂的节点对应的 N 重置为 0,新一层图元的总和为:

$$\mathrm{NT}=\sum_i \mathrm{NL}_i+\mathrm{NR}_i \tag{5}$$

之后依次在三个维度遍历每一个节点的所有图元,并按照其实际左右子节点的归属依次指派到新分配的空间中. 具体做法是先迭代计算出每个节点在新的一层空间中的偏移 D:

$$D_n=\begin{cases}0 & n=0\\ \sum_i^{n-1}\mathrm{NL}_i+\mathrm{NR}_i & n>0\end{cases} \tag{6}$$

之后遍历上层节点的图元包围盒序列,根据图元的位置将该图元指派到对应的位置. 需要注意的是,如果新节点不再需要进行划分,则该节点的图元不会指派到新分配的空间中. 由于其父节点是有序序列,所以得到的子节点也是有序序列,从而省略了排序的步骤,直接进行下一层候选划分计算.

2.4　复杂度分析

本文方法的特点一是按层级同时计算每一层节点的最优划分点. 按层级计算可以使分配内存的次数以及 GPU 与 CPU 的交互次数只与层数有关,并且可以使用连续内存空间保存全部需要计算的图元,而不需要使用队列的辅助. 在进行同步操作时,只需要在每个维度遍历一次上层节点就能得到下层节点的有序候选序列,避免了多次遍历. 二是使用基于空间树的快速计数方法,使每个线程可以根据快速计算出划分点对应的 NL、NR 和 NP,其计算复杂度和划分点穿过的图元数有关,考虑实际场景这个值总是远远小于总的图元数,使得计算速度大大提高.

本节对本文提出方法的复杂度从三个方面进行分析:

1)KD 树构造前,需要先对图元包围盒在三个维度进行排序,其复杂度是 $O(N\log N)$.

2)在 KD 树构造期间,对于每一个非根节点,CPU 端首先需要构造出三个维度的有序图元包围盒和节点包围盒. 节点包围盒可以根据最优划分点在 $O(1)$ 的时间得出,将图元顺序指派至新的节点中,需要在三个维度上遍历所有图元包围盒序列一次,复杂度为 $O(N)$. 得出有序的图元包围盒后,层次递归每个节点的图元包围盒序列构造该节点的空间树,每一层总的复杂度为 $O(N)$. 在 GPU 端,根据空间树的特性,它需要搜索每个区间,如果区间内所

有图元都在划分点的同一侧,则停止搜索这个区间.所以只有该区间包括了被划分点穿过的图元包围盒时,空间树才会在该区间的二分子区间中去搜索.又因为树的层高不会超过($\log N$),于是每次计算SAH值的复杂度最坏情况下不会超过$O(NP+\log N)$.其中NP是该划分点穿过的图元数,正常场景中无论NP还是$\log N$都是远远小于N的值.

3)整体分析:整体结合来看,每一层复杂度为$O(N+NP+\log N)$,而树的高度为($\log N$),因此该构造方法复杂度仍为$O(N\log N)$.由于CPU端的任务主要是指派图元与空间树构建,而不涉及任何浮点数计算,所以计算效率很高.处理中排序对算法性能影响很大,考虑到真实场景中一般都会有一个初始状态,我们可以基于该状态下提前离线进行预排序,从而节省初始的排序计算量.

3 实验结果及讨论

为了验证方法的有效性,我们使用统一计算设备架构(Compute Unified Device Architecture:CUDA)框架实现本文提出的算法.算法运行平台为:四核八线程INTEL酷睿E3处理器,GPU型号为GTX750Ti.测试数据为Toys、Bunny、Dragon、buddha四个常用的场景,这些场景包含的图元的数从11 k到1 M不等.在最多64线程下本文算法与3种算法进行比较,分别是Wald提出的串行算法[2]、文献[14]算法,以及不使用空间树的本文算法(朴素GPU算法)做对比,算法时间(并行算法64线程)比较如表1所示.

表1 不同场景下本文与对比算法构建时间

Tab.1 The construction time of this paper and the contrast algorithm under different scenes

场景	图元数	时间/ms			
		文献[14]	Wald[2](串行)	朴素GPU算法	本文算法
Toys	11 k	31	37	27	28
Bunny	69 k	87	233	240	54
Dragon	252 k	193	641	810	110
buddha	1 M	790	2 554	2 840	410

表1结果表明,本文算法的构造时间在各个场景都优于其他算法,其主要原因是:本文算法是对图元包围盒进行排序,而不是候选点,使得需要排序的序列长度从$2N$减少到了N,在计算上又通过图元级并行处理,虽然算法均摊下来不如Wald方法中$O(1)$简单,但从每个计算核心来看本文算法复杂度为$O(NP+\log N)$优于Wald方法的$O(N)$.朴素GPU算法虽然不需要进行预排序,但每个线程都需要对当前所有图元进行相交测试,时间复杂度为$O(N)$,复杂度也远高于本文算法,对比于Wald方法,二者复杂度虽然相近,但由于GPU单核心计算能力较低,所以速度较慢.文献[14]中算法虽然是退化的SAH-KD树,但在实际场景中依然慢于本文算法,这归根于单节点运算的低并行度.此外,每次划分求出的SAH值是KD树性能的重要指标,但由于后三种方法都是通过寻找最优候选划分点,所以求出来的SAH值都差不多,并不存在遍历性能上的差别.

在具体的场景中,场景Toys的处理结果表明,在小场景下,本文算法与高效的串行算法差距并不大,主要原因是本文算法中指派图元是在CPU中完成,而少量的浮点数计算不会明显影响串行算法的计算时间.朴素GPU算法不需要最开始的预排序,且在小场景下空间树的效果并不明显,最终两者的结果相近.

其他三个场景中随着图元数量的增加,本文算法达到4~6倍于串行算法的时间效率,且随着图元数变大稳步增加.串行算法在每层构建时通常需要$O(N)$级别的浮点数计算,当数据量大时计算量远超过条件判断并且越来越明显.空间树的效果也在大场景中体现出来,对比于朴素GPU算法,由于条件判断较多消耗了大量线程的计算能力,而空间树可以大大降低这一过程,代价仅需在CPU端对有序的图元进行一次遍历.

构建SAH-KD树过程中,随着层次的加深,每个节点所包含的图元数越少,但遍历时需要更多的

相交测试才能达到叶节点,为了平衡这一现象,在实际应用中都会制定相应的早停策略. 图 1 列出 buddha 场景下 SAH-KD 树计算 1～7 层时,通过公式(7)计算得到的算法时间效率(文献[14]的算法在上层使用中位图元划分,不适用于这类分析).

$$P_n = \frac{T_n - T_{\text{arr}}}{T_1 - T_{\text{arr}}} \tag{7}$$

其中 T_n 为计算到第 n 层所用的时间,T_{arr} 为排序所用的时间. 可以看出 Wald 串行算法的时间消耗随着层数上升显著提高. 虽然每层计算后被划分点穿过的图元会被复制一份,但是本文算法中这些计算量会被平摊到每个线程中,并不会明显影响计算速度,从图 1 可以看出算法耗时基本和层次数量成线性关系.

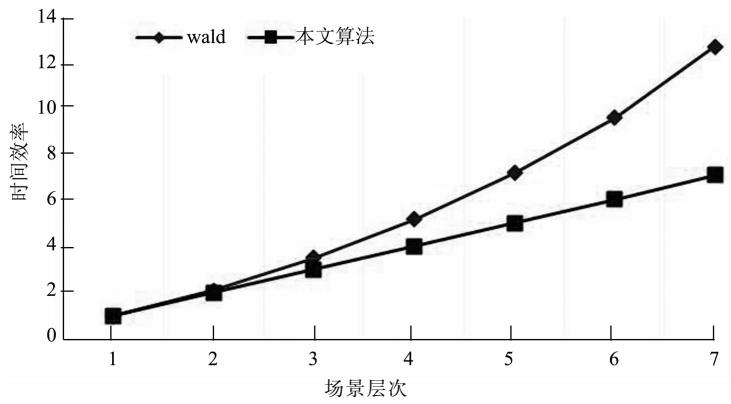

图 1　buddha 场景下 Wald 串行算法与本文算法在不同层次上的时间效率

Fig. 1　Time efficiency of Wald serial algorithm and our algorithm at different levels in Buddha scene

线程数对并行算法的影响也非常重要,图 2 是本文算法在使用 8～64 个线程时以 8 核为基准,在四种场景下与对比算法的性能加速比较. 可以看出本文算法随着核心数的增长,加速比基本和倍率成正比关系,能够充分利用 GPU 的多核架构,并且随着场景的增大,越可以有效地利用多核性能. 在小场景 Toys 中,64 核与 8 核的加速比为 5 倍左右,而在较大的场景 buddha 中达到了 7.5 倍左右的性能. 文献[13]提出的并行算法由于大小节点区分计算,在 128 线程只能达到 16 线程 3～6 倍的加速比,而本文算法在内存分配与图元的指派每层只在 CPU 中进行一次,所以不会给算法性能带来负担,同时由于较高的负载均衡度,使得算法可以高效率利用每个线程.

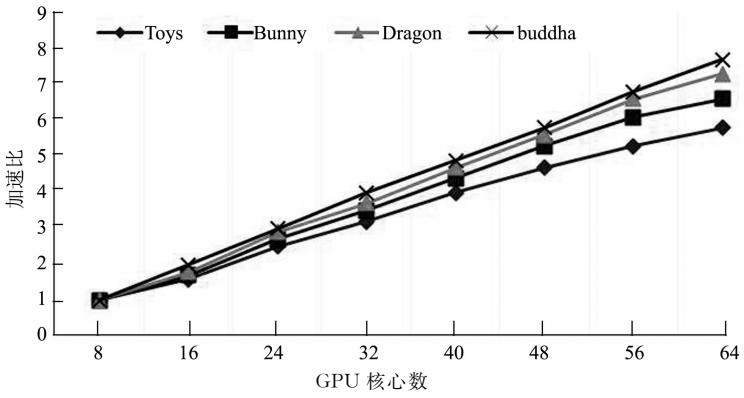

图 2　本文算法在不同线程数时的加速比

Fig. 2　Acceleration ratio of our algorithm in different threads

4 总 结

本文提出了一种使用 CPU 和 GPU 并行构造一棵基于层次的 SAH-KD 树,解决了当前 SAH-KD 树并行度较低和需要区分大小节点的问题.由于 KD 树的构建是一个静态过程,本文所有测试都是在静态场景中进行.但是真实场景许多是动态的,以往研究者通常是对静态场景进行重复计算来模拟动态场景.我们接下来的研究方向是如何在动态场景中快速生成 SAH-KD 树,基于本文算法在预排序阶段进行相关的优化,期望达到实时光线跟踪的效果.

参考文献

[1] HUSSAIN S, GRAHN H. Fast kd-tree construction for 3D-rendering algorithms like ray tracing[C]//Lecture Notes in Computer Science. Berlin: Springer, 2004,4842:681－690.

[2] WALD I. Realtime ray tracing and interactive global illumination[J]. Information Technology,2004,48 (4):242－245.

[3] MACDONALD J D, BOOTH K S. Heuristics for raytracing using space subdivision[J]. Visual Computer, 1990,6 (3):153－165.

[4] SZECSI L. An effective implementation of the kd-tree[M]. Rockland: Charles River Media Inc,2003:315－326.

[5] WALD I, HAVRAN V. On building fast kd-trees for ray tracing and ondoing that in $O(N\log N)$[C]//Symposium on Interactive Ray Tracing 2006(RT). Salt Lake City:IEEE,2006 :61－69.

[6] TILLMANN M, PFAFFE P, CKAAG W F. Online-auto tuning of parallel SAH kd-trees[C]//International Parallel and Distributed Processing Symposium (2016). Chicago: IEEE, 2016:628－637.

[7] LEE J, SHIN Y. Real-time ray tracing on coarse-grained reconfigurable processor[J]. International Conference on Field-programmable Technology,2014, 149 (3):192－197.

[8] POPOV S, GUNTHER J. HP seidel. experiences with streaming construction of SAH kd-trees[C]//Symposium on Interactive Ray Tracing (2006). Salt Lake City:IEEE,2006:89－94.

[9] HUNT W,MARK W R,STOLL G. Fast kd-tree construction with an adaptive error-bounded heuristic[C]//Symposium on Interactive Ray Tracing (2006). Salt Lake City:IEEE,2006:81－88.

[10] SHEVTSOV M, SOUPIKOV A, KAPUSTIAN A. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes[J]. Computer Graphics Forum, 2007,26 (3):395－404.

[11] SPJUT J, BOULOS S, KOPTA D. A multi-threaded architecture for real-time ray tracing[J]. Symposium on Application Specific Processors,2008,28 (12) :108－114.

[12] KANG Y S, NAH J H, PARK W C. gkDtree: A group-based parallel update kd-tree for interactive ray tracing[J]. Journal of Systems Architecture the Euromicro Journal , 2013,59 (3):166－175.

[13] ZHOU K, HOU Q, WANG R. Real-time kd-tree construction on graphics hardware[J]. Acm Transactions on Graphics, 2008,27 (5):1－11.

[14] CHANGH B,SEOH W,IHM I. On the efficient Implementation of a real-time kd-tree construction algorithm[M]. Singapore: Springer,2015:207－219.

[15] MACDONALD J D, BOOTH K S. Heuristics for raytracing using space subdivision[J]. Visual Computer,1990,6 (3):153－166.

[16] SANTALÓ L A. Integral geometry and geometric probability [M]. London: Addison-Wesley Pub Co,1976 :91－113.