doi：103969/j. issn. 0490-6756. 2016. 03. 009

基于外存的场景加速数据结构快速构建算法

刘 森，吴志红
（四川大学计算机学院视觉合成图形图像技术国防重点学科实验室，成都 610065）

摘 要：大规模场景的绘制问题一直以来都是图形学中的重要研究课题之一. 其难点在于场景本身占用内存资源多；其次图形绘制过程计算量巨大. 因此本文以大规模场景为研究对象，设计了一种多级层次包围盒用于管理场景数据，并利用 GPU 的并行计算能力，加速多级层次包围盒的构建，提高绘制效率. 本文算法的贡献在于，利用莫顿编码将场景分块问题转化为图元排序问题，从而快速完成场景分块，并以此构造多级层次包围盒；同时，针对多级层次包围盒，使用分段遍历策略，以初始阶段的遍历结果进行 I/O 调度，有效地减少了遍历时间. 实验结果证明了该算法的正确性与可靠性，与 CPU 的遍历效率相比，提高 10x 以上.

关键词：大规模场景；GPU；遍历策略；多级层次包围盒
中图分类号：TP391 **文献标识码**：A **文章编号**：0490-6756(2016)02-0289-06

Fast construction algorithm of out-of-core based scene acceleration data structure

LIU Sen, *WU Zhi-Hong*
(National Key Laboratory of Fundamental Science on Synthetic Vision,
College of Computer Science, Sichuan University, Chengdu 610065, China)

Abstract: Large-scale scene's rendering is one of the major research topics in computer graphics. First, it needs lots of memory resource; second, rendering process requires a great amount of computation. This paper focuses on large-scale scene and proposes a novel algorithm which named Multi-Level Bounding Volume Hierarchies to manage scene data, and then using GPU to accelerate construction and traversal process. The algorithm has two contributions. First using spatial Morton codes to sort triangle primitives, then chapping scene into blocks and constructing MLBVH; at the same time, the traverse efficient have significantly improved by using a two phase traverse tactics, which using the first stage result to control the second stage traverse. Finally, the authors have done some experiments to prove the algorithm correctness and reliability. Traverse efficiency has improved 10x compared with CPU.

Key words: Large-sacle scene; GPU; Traversal strategy; Multi-level bounding volume hierarchies

1 引 言

生成相片级的图像是计算图形学长期以来的目标，这也是光线跟踪[1,2]算法的研究初衷. 随着光线跟踪技术的逐渐发展，并已广泛应用于游戏，但对于电影、军事仿真训练等大规模场景应用，其高效真实感绘制一直以来是图形绘制研究的难题. 特别是当场景规模超过内核内存容量，需要采用基于外存的存储方式时，数据组织与数据调度的效率成为制约绘制计算的重要瓶颈之一.

收稿日期：2015-06-01
基金项目：国家科技支撑计划(2012BAH62F03)；国家自然科学基金(61472261)；863 计划(2015AA016405)
作者简介：刘森(1990－)，男，重庆人，硕士研究生，研究方向为计算机图形学/虚拟现实. E-mail：1284902513@qq. com
通信作者：吴志红. E-mail：wuzhihong@scu. edu. cn

光线跟踪的算法核心是加速结构,通过将三角面片重新进行组织,可以极大的提高遍历效率. 目前已广泛应用的加速结构有:层次包围盒[3,4](BVH)、KD树[5,6]等. 其中KD树的性能最好,但构造过程耗时,因此在处理大规模场景时, BVH是一个更合适的选择.

Wald等[7]实现了多核架构下的层次包围盒的构建. MacDonald等[8]引入表面积启发函数,改善了构建的质量. Slusallek等[9]使用多核CPU构建KD树,但受硬件限制,性能提升幅度不大.

Teller等[10]提出了一种基于外存的辐射度系统用于管理大规模场景. Demarle等[11]提出了一种分布式虚拟内存系统加速光线跟踪速度. Navratil等[12]提出了一种利用缓存的有限粒度光线调度算法.

同时,一些基于GPU的并行构建算法也被提出. Zhou等[13]提出了实时的KD树构建算法. 基于GPU的BVH构建算法[14,15]也被广泛使用,杨鑫等[16]和Lauterbach等[17]分别提出了两种不同的基于GPU的BVH构建算法. Pantaleoni等[18]通过改进LBVH算法提出了HLBVH算法,而后改进的HLBVH算法陆续被提出[19,20],使构建和遍历效率都得到了极大提高. 尽管这些算法在一定程度上提高了光线跟踪的效率,但是对于大规模场景的处理还是有些不足.

本文在众多研究者已经提出并实现的层次包围盒构建算法基础上,结合大规模场景的特点,实现了基于GPU的大规模场景的多级层次包围盒(Multi-Level Bounding Volume Hierarchies, ML-BVH)构建与遍历算法. 该方法通过利用空间填充曲线的莫顿编码快速对数据进行分块,进而充分利用当前GPU的海量并行能力来提高计算的效率. 该算法贡献有两点:(1)受莫顿编码的启发,将场景分块问题转化为排序问题,从而快速对场景数据进行分块,而后使用异步调度的方式流水化加速MLBVH的构建;(2)实现了一种分段遍历策略,用前一阶段的遍历结果优化后一阶段的遍历顺序,极大的减少了遍历的时间. 由于以上两点,使得可以在不更新已有硬件的情况下,利用GPU的计算能力快速处理大规模场景数据.

2 MLBVH构建算法描述

虽然GPU有很高的并行计算能力,但是对于大规模场景的处理,由于数据量过大,显存有限,不能直接利用GPU进行处理. 因此本文采用分而治之的策略将构建过程分为两个处理阶段:(1)场景数据分块;(2)MLBVH加速数据结构的构建.

2.1 场景数据分块

分块质量将会影响整个加速结构的质量,而决定分块质量的因素有两个:其一是分块的大小,较小的块会增加I/O的次数,较大的块会增加构建的时间;其二是块间AABB重叠率,随着重叠率的增加,会增加遍历的块数(当光线相交于块的重叠部分时,需要分别与重叠块进行相交测试),从而降低遍历的效率. 因此,为了快速得到高质量的分块,引入空间填充曲线的莫顿编码,将场景分块问题转化为排序问题. 算法先对输入的场景三角面片排序,然后再进行分块,而排序算法的复杂度至少为$O(N \log N)$,GPU的高并发能力可以突破这个限制.

算法1描述了场景数据分块的整个流程. 整个流程由排序和分块两个部分构成.

算法 1 数据分块过程

```
1) // 对输入图元进行分块
2) function chipPrimitive2Blocks()
3) // 根据图元数,计算出分块所需二进制位
4) // 对图元进行排序
5) sortPrimitives()
6) // 将图元分为 2^SplitBits 块,SplitBits 是根据
      显存大小动态决定的
7) chip2BlocksUseSplitBits()
8) // 分块后存在分块不均的情况,需要将较小
      的块合并为较大的块,以减少 I/O 次数
9) mergeAdjacentSmallBlocks()
10) end function
```

排序:排序由两个步骤组成,段内排序和段间归并,段内高位用基数排序和低位用奇偶排序,段间采用归并排序. 段内排序的基本流程为:首先根据图元AABB的中心坐标计算莫顿编码,共3n位(每个维度n位),图1展示了2D空间中计算莫顿编码的过程,两位数字表示每个维度用一位($n=1$)进行量化,四个数字表示每个维度用两位($n=2$)进行量化. 量化每个维度后,根据中心点所在位置计算出每个维度的量化值,最后交叉排列二进制位,即为图元的莫顿编码. 然后按段内最高$3m$位排序,如图2所示,2D情况下的图元排序($m=2$),若相邻图元$i-1$和i的最高两位不等,则将标记置为

1，否则为 0. 对标记数组按公式

$$Arr[i]=\begin{cases}1, i=0\\ Arr[i-1]+Flag[i-1], \\ \quad i>0\end{cases} \quad (1)$$

的方式进行一次扫描得到扫描数组. 压缩步骤就是依赖于以上两个数组，压缩操作的结果是小段的开始位置和小段的长度. 接着按 MSD 的方式完成小段排序. 最后的解压步骤将还原排序后的图元顺序. 此时只是完成了最高的 $3m$ 位排序，剩下 $3(n-m)$位还是无序的. 为了完成排序，将图元按最高 $3m$ 位是否相同进行分组，然后是用 GPU 的 wrap 的局部存储空间快速的完成分组的内部排序. 与全局的基数排序相比，本文中的排序算法有以下优点：(1) 减少了 GPU 中线程的同步开销；(2) 局部的奇偶排序使得排序效率有较大提高与基数排序.

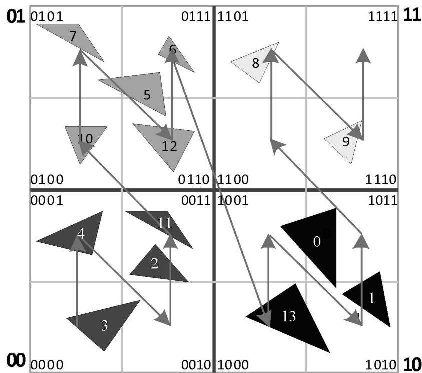


图 1　莫顿曲线顺序及图元莫顿编码

Fig. 1　Morton curve ordering and the corresponding Morton codes

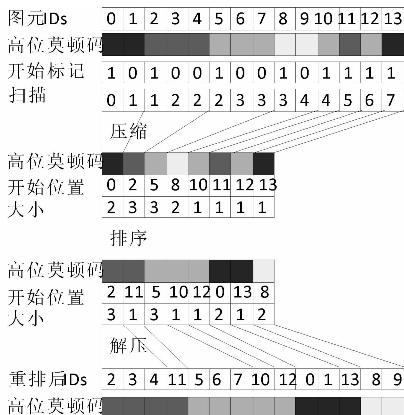


图 2　图元排序整体流程

Fig. 2　The overall primitive sorting scheme

分块：为快速完成有序图元序列的分块，即决定分块的开始和结束位置. 换句话说，对于图元 i 和 $i+1$，我们需要判断它们是在同一分块，还是不同分块. 根据观察与分析，从 3m 位的最高位开始，若相邻图元该位(第 h 位)值不同，那么在 h 到 $3m$ 的每个层次都会存在块的边界，本文中将其保存为(i,h)、$(i,h+1)$、……、$(i,3m)$. 当计算出所有的分割对后，按分割的层次进行排序(分割对的第二个元素)，即可快速的计算出分块的开始与结束位置. 在上述的分块中，存在一个较大的缺陷，即分块的粒度不均匀，因此需要一个后处理步骤将较小的块合并为较大的块，以减少 I/O 次数，提高效率，如图 3 所示.

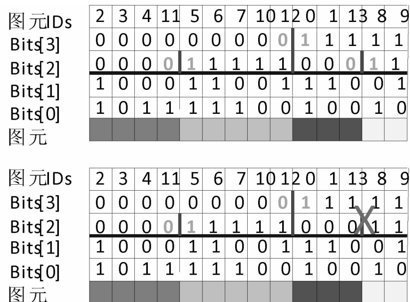


图 3　分块流程

Fig. 3　The overall blocking process

2.2　MLBVH 构建流程

表面启发代价函数由于其能够为光线与物体求交提供一定的预测性，且已被应用于多种加速数据结构的构建算法中，因此本文中的构建算法亦是基于该代价函数实现.

MLBVH 构建流程如算法 2 所示，对于每个分块使用 SBVH 算法构建其层次结构. SBVH 算法在基于 SAH 优化算法的基础上，引入裁剪桶加速 SAH 代价估算，同时根据当前待划分结点自适应地选择对象划分还是空间划分. 但该算法实现于 CPU 上，因此本文对算法稍作改进并移植到 GPU 上，使得构建速度得到了极大的提高. 完成所有分块的 BVH 构建后，以每个分块层次结构的根结点作为数据元素，使用 LBVH 方法构建上层层次结构. LBVH 算法的优势是可并行性高、速度块，非常适合在层次结构划分的上层结点使用. 下面对算法的每个步骤进行简要的分析.

函数 chipPrimitive2Blocks 在 2.1 小节已有详细描述，这里不再赘述. 接着依次加载数据块到显存，函数 constructSBVH、creatAndStreamoutTree 分别完成层次结构的构建和层次结构从 GPU 到 CPU 的导出，同时保存根结点到 TreeList 中. 最后由函数 constructHighLevelBVH 完成上层层次结构的构建，得到大场景的多级层次结构. 如图 4(a)和(b)所示.

算法 2 多级层次包围盒构建流程

```
1) function buildMLBVH()
2) chipPrimitive2Blocks()
3) while ReadBlock() ! = EOF
4) constructSBVH()
5) creatAndStreamoutTree ()
6) //root is the root node
7) TreeList ← root
8) end while
9) constructHighLevelBVH(TreeList)
10) end function
```

3 MLBVH 遍历

遍历目的是快速判断场景对光线的遮挡关系. 本文中的 GPU 光线遍历与普通的光线遍历有所不同,MLBVH 是多层次结构,因此本文提出了一种分步遍历策略,有预相交测试、光线聚类、局部光线图元求交三个部分. 预相交测试是与高层次 BVH 进行相交测试,然后根据测试结果对光线分类,最后与低层次 BVH 求交.

预相交测试阶段将所有待求交光线与 MLBVH 的高层次 BVH 进行求交,并记录光线求交结果为(RayID,TreeID),其中 RayID 是光线的索引,TreeID 是指低层次 BVH 的索引(即由每个分块构建的层次结构),若光线与场景不相交,则删除该记录. 光线聚类目的是利用空间相干性,将空间属性相似的光线存储在连续的内存空间,以减少低层次 BVH 的 I/O 调度次数. 聚类过程的具体实现为构建一个 Hash 表,对 TreeID 进行 Hash,每个 Hash 值所对应的光线即为一个光线聚类,称为光束. 局部光线图元求交阶段,以光束为单位,调度相应的场景块到显存,计算具体的相交三角面片. 具体流程如图 4(c)(d)所示.

4 结果分析

4.1 实验环境

处理器:Intel(R) Core(TM) i7-5930K CPU @ 3.50 GHz 3.50 GHz; 内存: 64.0GB; 显卡: GTX980; 操作系统: Windows 7.

4.2 实验结果

本文的算法使用 CUDA 语言实现. CUDA 提供了通用的类 C 语言的编程接口,以利用 GPU 的并行计算能力. 实验主要分为两个部分,分别为算法的构建效率分析和算法的遍历效率对比实验.

本文的测试场景有 4 个,表 1 中是各个场景的详细属性信息,最小测试场景有约 1600 万个三角面片,最大测试场景有约 4600 万个三角面片. 图 5 是测试场景的展示效果,从(a)到(d)测试场景的规模逐渐增大.

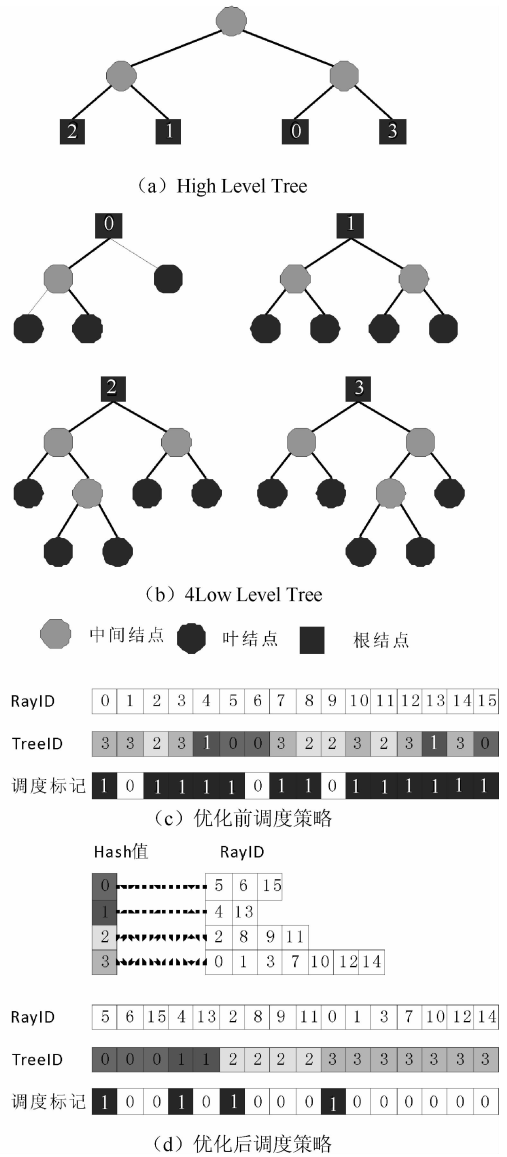
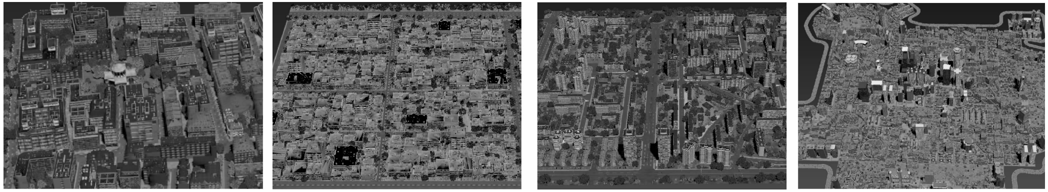


图 4 MLBVH 遍历
Fig. 4 MLBVH Traveling

4.2.1 构建效率分析 表 2 是 MLBVH 构建算法在 4 个不同场景上的构建效率测试结果. 以行观察,从表中可以看出,随着模型的增大,每一步需要的时间也在增加. 但是对于每个步骤耗时增加的原因却各有不同. 生成莫顿编码、构建低层次 BVH 的时间增加,主要由模型增大产生的耗时增加和分块数量增多引起的主存与显存间 I/O 耗时增加两部分导致. 而排序与分块的时间增加主要因



(a) 场景 1　(b) 场景 2　(c) 场景 3　(d) 场景 4

图 5　MLBVH 遍历

Fig. 5　MLBVH Traveling

表 1　实验场景信息

Tab. 1　Information of experimental scene

	场景 1	场景 2	场景 3	场景 4
三角面片(K)	16,185	22,475	37,161	46,169
大小(MB)	1,740	2,390	3,900	5,210
分块数目(块)	67	88	144	210

表 2　不同模型的构建效率

Tab. 2　Constructing efficiency of different scenes

	场景 1(ms)	场景 2(ms)	场景 3(ms)	场景 4(ms)
生成莫顿编码	806	1312	1807	2047
排　序	147	198	319	429
分　块	197	301	475	724
低层次 BVH	63392	78565	152671	192097
高层次 BVH	4	5	5	8
总时间	64546	80381	155277	195297

素是由数据规模增大导致. 以列观察,MLBVH 的构建过程时间开销主要集中于构建低层次 BVH. 构建低层次 BVH 主要受两个因素影响,其一构建 BVH 本身比较耗时,其二需要将构建完成的 BVH 数据从显存拷贝至主存,分块数越多,相应数据拷贝次数和拷贝量也会增加,从而增加耗时. 因此本文使用多个 CUDA 流,不同流之间进行流水化工作.

由于 LBVH 和 HLBVH 算法只适用于小规模模型的层次结构构建,而本文中处理的是大规模的场景数据,因此没有与 LBVH 和 HLBVH 进行构建和遍历效率的对比.

4.2.2　遍历效率分析　在渲染场景计算光照时,需要判断场景是否对当前光线有遮挡,以采取不同的光照计算公式计算光照值. 即给定大量光线,计算出光线与场景的交点. 表 3 是 CPU 与 GPU 遍历效率对比,测试模型为场景 4,光线由随机算法生成. 从对比结果可以看出,随着光线数量的增加,CPU 算法的遍历时间与 GPU 算法的遍历时间均呈线性增长,但 GPU 遍历速度在相同光线数目下是 CPU 遍历速度的 10x 以上,这是 GPU 的并行计算能力的体现.

表 3　求交效率对比

Tab. 3　Contrasting of intersection efficiency

光线数目(万条)	Ours(ms)	CPU(ms)
100	3122	72481
200	4600	127687
400	7932	278735
800	14439	539584

5　结束语

本文通过分析大规模场景绘制的难点,场景数据量过大,显存不足以完全载入进行并行运算,因此本文实现了一种基于外存的 MLBVH 算法快速构建大规模场景的加速数据结构. 该算法同时利用了主存的大容量和 GPU 高并行计算能力,在保证 BVH 结构质量的前提下,其构建效率和遍历效率都得到了较大提升.

本文中的算法还有待完善的地方(1)若场景达到几百 GB 甚至更大时,主存也不能一次载入场景数据,需要考虑 GPU 显存、CPU 主存以及硬盘这三者间的数据 I/O 调度,这对调度算法将是一个很大的挑战;(2)优化低层次 BVH 的构建算法,以减少整个构建过程的时间消耗.

参考文献:

[1]　Wald I, Slusallek P, Benthin C, *et al*. Interactive rendering with coherent ray tracing. [J]. Comput Graph Forum, 2001, 20(3): 153.

[2]　Reshetov A, Soupikov A, Hurley J. Multi-level ray tracing algorithm[J]. Acm Trans Graph, 2005, 24 (3): 1176.

[3]　冯立颖. 碰撞检测技术研究综述[J]. 计算机时代, 2014 (5): 9.

[4] Eloe N W, Steurer J A, Leopold J L, *et al*. Dual graph partitioning for Bottom-Up BVH construction [J]. J Visual Lang Comput, 2014, 25(6): 764.

[5] 张琴，蔡勇，常伟杰. 基于空间分割的局部KD树动态构建算法[J]. 机械工程师，2010(12)：30.

[6] Wu Z, Zhao F, Liu X, *et al*. SAH KD-tree construction on GPU [C]//Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. Jinan, China: ACM, 2011.

[7] Wald I, Boulos S, Shirley P. Ray tracing deformable scenes using dynamic bounding volume hierarchies[J]. Acm Trans Graph, 2007, 26(1): 1.

[8] Macdonald J D, Booth K S. Heuristics for ray tracing using space subdivision [J]. Visual Comput, 1990, 6(3): 153.

[9] Slusallek P, Seidel H P, Gunther J, *et al*. Experiences with streaming construction of SAH KD-Trees[C]// Proceedings of the 2006 IEEE on Symposium on Interactive Ray Tracing. Salt Lake City , USA: IEEE 2006.

[10] Teller S, Fowler C, Funkhouser T, *et al*. Partitioning and ordering large radiosity computations[J]. Acm Siggraph, 1994, 13(4): 443.

[11] Demarle D E, Gribble C P, Parker S G. Memory-savvy distributed interactive ray tracing[J]. Egpgv, 2004, 25(2): 93.

[12] Navratil P A, Fussell D S, Lin C, *et al*. Dynamic Ray Scheduling to Improve Ray Coherence and Bandwidth Utilization[C]//Proceedings of the IEEE Symposium on Interactive Ray Tracing. Ulm, Germany: IEEE Computer Society, 2007.

[13] Zhou K, Hou Q, Wang R, *et al*. Real-time kd-tree construction on graphics hardware[J]. Acm Transactions on Graphics, 2013, 27(5): 126.

[14] Doyle M J, Fowler C, Manzke M. A hardware unit for fast SAH-optimised BVH construction [J]. ACM Trans Graph, 2013, 32(4): 139.

[15] Yin M, Li S. Fast BVH construction and refit for ray tracing of dynamic scenes[J]. Multimed Tools Appl, 2014, 72(2): 1823.

[16] 杨鑫，王天明，许端清. 基于GPU的层次包围盒快速构造方法[J]. 浙江大学学报:工学版，2012，46(1)：84.

[17] Lauterbach C, Garland M, Sengupta S, *et al*. Fast BVH Construction on GPUs[J]. Comput Graph Forum, 2009, 28(2): 375.

[18] Pantaleoni J, Luebke D. HLBVH: hierarchical LBVH construction for real-time ray tracing of dynamic geometry[C]//Proceedings of the High Performance Graphics'10. Saarbrücken, Germany: Eurographics Association, 2010.

[19] 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, 2011.

[20] Karras T, Aila T. Fast parallel construction of high-quality bounding volume hierarchies[C]// Proceedings of the 5th High-Performance Graphics Conference. Anaheim, USA: ACM, 2013.