自底向上构建高效 BVH 的研究①

唐 宇[1,2], 于 放[2], 孙 咏[2], 王丹妮[3]

[1](中国科学院大学, 北京 100049)
[2](中国科学院沈阳计算技术研究所, 沈阳 110168)
[3](国网辽宁省电力有限公司信息通信分公司, 沈阳 110000)

摘 要: 光线追踪中, 加速结构对于减少光线与几何体求交的计算量起到了不可或缺的作用, 找到一种高效的加速结构仍然是本领域的研究热点之一. 传统的加速结构如 BVH, 都是自顶向下的构建方式, 由于其本身的限制, 构造出的加速结构往往不是最优的. 为了改进自顶向下的缺陷, 提出了一种自底向上 BVH 的构建方式, 它使用了 BVH 和空间划分的混合结构, 并引入了一种改进的表面积启发代价函数和多个构建参数来优化最后的构建结果. 最后实验表明, 此文中提出的方式比传统的BVH与KD-Trees有更好的加速效果, 可以实现对前两者5%-10%的提速. 最后, 文中还给出了一些加速方案来减少构建时间.

关键词: 光线追踪, 自底向上, KD-Trees, BVH, SAH

Research on Bottom-Up Approach to Build an Effective BVH

TANG Yu[1,2], YU Fang[2], SUN Yong[2], WANG Dan-Ni[3]

[1](University of Chinese Academy of Sciences, Beijing 100049, China)
[2](Shenyang Institute of Computing Technology, Chinese Academy of Sciences, Shenyang 110168, China)
[3](Liaoning Electric Power Company Limited, Shenyang 110000, China)

Abstract: In ray tracking, acceleration structure is indispensable for reducing computation time of ray-geometry intersection. Finding an efficient and high quality acceleration structure is still an active area of research. The top-down approach is widely used in building acceleration structure, such as BVH. However, due to its limitation, the result tree is not always optimal. This paper presents a bottom-up approach to build an effective BVH which is a hybrid between BVH and Space Partitioning. An improved surface area heuristic cost function and multiple parameters have also been used to optimize this model. Experiments show that the approach mentioned above is more effective than the conventional BVH and KD-Trees. It can realize 5%-10% performance improvement. Furthermore, the way to reduce the building time has been investigated in this paper.

Key words: ray tracking; bottom-up; KD-Trees; BVH; SAH

1 引言

光线追踪是一种十分强大的图像合成技术, 可以生成真实感十分强的图像, 它已经被广泛应用于电影、动画等离线渲染领域. 近年来, 随着计算机计算能力的提升, 实时的光线追踪成为可能. 然而光线追踪仍是一种计算密集的任务, 不得不用一些预处理的数据结构来加速其计算过程[1].

常见的加速结构有 KD-Trees, BVH(Bounding volume hierarchies), 和 Grid, 其中 KD-Trees 是光线追踪中最常用的加速结构. 然而上述提到的加速结构都是基于自顶向下式的构建方式, 本文提出了一种自底向上构建BVH的方式, 这种方式结合了传统空间加速划分和针对对象的划分, 并使用自底向上的启发式构建的方法, 生成的树结构往往质量更高, 包围盒更加紧密.

2 相关工作

文献[2]提出了被广泛应用的表面积启发式算法,

① 收稿时间:2015-04-29;收到修改稿时间:2015-06-03

即 SAH. 尽管最初此算法被用在 BVH 上, 后续的研究将其应用在 KD-Trees 上.

加速结构的改进工作主要围绕着两个方向进行. 一种为改进的数据结构, 如为了有效的解决场景中图元分布不均匀导致 BVH 效率不高的情况, 文献[3]提出了一种针对 BVH 的 Early Split Clipping 算法. 它放松了每个 BVH 叶子节点中只能存放一个图元的限制条件, 在构建 BVH 之前, 通过对较大图元的包围盒(并不是图元本身)进行分割, 对图元创建了更加紧凑的包围盒. 每个包围盒根据它最长轴的中点来递归的划分, 直到划分的包围盒面积到达某个用户定义的阈值 SA_{max} 停止. 其余的图元按照常规的方式来进行构建. 这种方式构建的 BVH 往往有更高的效率.

与上面提到的算法类似, 文献[4]提出了 Edge Volume Heuristic 来减少节点之间的重叠, 与上边不同的是, 它来分割三角形而不是包围盒, 划分递归的进行, 直到三角形三条边的包围盒中, 最大的那个小于某个阈值.

文献[5]提出了 Spatial Splits Volume Hierarchy, 此算法综合了传统的 BVH 构建和空间划分算法, 首先通过 SAH 算法找到最合适的图元划分候选位置, 然后通过 Chopped Binning 找出空间划分的候选位置, 对于一个重叠的三角形, 其有三种可能, 左右都划分, 全部划分为左边与全部划分到右边, 通过计算其代价, 来选择最优的划分. 对于重叠的部分, 此算法判定如果相交部分小于某个阈值, 则忽略.

另一种改进为用并行的方式来加快树的构建, 文献[6]提供了快速构建 BVH 与 KD-trees 的方法. 文献[7]提出了一种无栈遍历 BVH 的方法, 文献[8]提出了 GPU 加速的方式, 文献[9]的方法使得加速结构适用到动态场景成为可能, 文献[10]给出了一种硬件加速的 BVH 设计.

最后, 文献[11]做了加速结构构建与场景渲染并行进行的研究.

3 高效BVH构建

大多数的加速结构都是基于层次的划分, 按照划分对象的不同, 加速结构可分为俩种, 其一, 基于空间的划分技术(如 KD-Trees, octrees 等), 这种划分技术的典型特征为叶子节点互不相交, 并且一个图元信息(如三角形)可能存在多个对其的引用. 另一种加速结构为, 基于对象的划分技术(如 BVH), 它确保了每个图元信息只在叶子节点存储一次.

为了构建更加高效的 BVH, 本文使用了两种划分结合的方式.

3.1 空间分割

算法的第一个步骤, 使用一个叫做向下分割的过程. 此处的算法与 KD-Trees 类似, 每次选取所有图元包围盒中最长轴为参考轴, 然后根据标准的表面积启发式算法来选择划分的位置, 将划分出来的两个平面作为左右子树.

当某一个节点的图元数量小于某个阈值 δ, 则划分结束. 与 KD-Trees 不同的是, 这时的算法并未停止, 而是以当前的节点为根节点, 自底向上的构建 BVH, 由于 BVH 的构建并不影响当前的空间划分, 这里可以创造一个子线程来执行相关的任务, 详细的算法见(表 1).

表 1 空间划分算法

```
1.  // N: leaf node to split
2.  // P: primitives set
3.  BEGIN Split(N,P):
4.      IF P.size < delta
5.          T = createThread()
6.          T.parameter = BuildBVH(N,P)
7.          T.run()
8.          return
9.      ENDIF
10.
11. findBestSplit(N,P,axis,pos)
12. NL = leftChild(N)
13. NR = rightChild(N)
14. BL = NL.BBox
15. BR = NR.BBox
16.
17. FOR i range 1 to P.Size
18.     B = P[i].Bbox
19.     IF B.OverLaps(BL)
20.         append(PL,P[i]);
21.     ENDIF
22.     IF B.OverLaps(BR)
23.         append(PR,P[i]);
24.     ENDIF
25. ENDFOR
26.
27. Split(NL,PL)
28. Split(NR,PR)
29. END
```

3.2 改进的代价函数

对于 KD-trees 和 BVH 都需要找出最佳的划分位置, 为了评估某个划分位置的优劣, Goldsmith 等人假设光线是均匀分布的, 并且不与场景中的任何图元相交, 提出了表面积启发代价函数[2]

$$C = C_t + \frac{SA(B_1)}{SA(B)} N_1 C_i + \frac{SA(B_2)}{SA(B)} N_2 C_i$$

式中, C_t 是光线的传播代价, C_i 为光线与图元相交的计算代价. N_1、N_2 分别是树中左右子节点中的图元数量. 与此对应的 B_1、B_2 为左右节点的包围盒. B 为当前节点的包围盒, 其中 B1、B2 ⊆B. SA 函数计算了某包围盒对应的表面积. 此式的前半部分给出了左子节点的代价, 对应的后半部分计算右子节点的代价. 有了代价函数, 可以十分高效的评判一个位置是否是合适的划分点.

由于自底向上的构建方式是从叶子节点开始, 必须引入一个判别方式来判定给定的两个包围盒是否更加合适用来构造新的父节点. 与文献[12]中提出的只计算距离不同, 这里使用了改进的 SAH 算法

$$C = a\left\{C_t + \frac{SA(B_1 \cup B_2)}{SA(B_{all})}(N_1 + N_2)C_i\right\} + (1-\alpha)d^2$$

其中 B_{all} 为当前网格中所有图元的包围盒, B_1、B_2 为需要判断代价的包围盒, N_1、N_2 为包围盒中图元的数量. d 为两个包围盒之间的距离. α⊆ [0,1], 经过实验, α 取 0.6 时具有更好的效果.

算法开始时, 以每个包围盒为一个簇, 依次计算任意两个簇的代价, 选取代价最小的一对簇来生成他们的父节点, 使用父节点的包围盒生成一个新的簇. 然后重新开始算法, 直到集合中只有一个簇结束. 详细的算法见表 2.

表 2 BVH 构建

```
1.  // N: leaf node to split
2.  // P: primitives set
3.  BEGIN BuildBVH(N,P):
4.  Clusters
5.  Cost
6.
7.  FOREACH primitive in P
8.      c = new Cluster(primitive)
9.      appendTo(Clusters,c)
10. ENDFOREACH
11.
12. FOREACH Ci,Cj in Clusters
13.     Cost[i,j] = calculate cost between Ci and Cj
14. ENDFOREACH
15.
16. WHILE Clusters.size > 1
17.     Ci,Cj = Find lowest cost in Cost
18.     Left = Ci
19.     Right = Cj
20.     c = new Cluster(Left,Right)
21.     C = C – Ci – Cj + c
22.     Update Cost
23. ENDWIHLE
24. N.BVHroot = Clusters
25. END
```

4 结构遍历

由于加速结构混合了空间划分与 BVH 树, 对于任意一条光线, 先使用空间划分的结果来遍历, 如果遇到某一结点为叶子节点, 后续的遍历则与普通的 BVH 遍历没有区别. 详细的算法可见表 3.

表 3 结构遍历

```
1.  // Ray: the ray want to be traversed
2.  BEGIN Traverse(Ray)
3.  Stack.push(TreeRoot)
4.  WHILE(!Stack.empty())
5.      N = Stack.pop_back();
6.      IF N is leaf
7.          root = N.BVHroot
8.          Stack2.push(root)
9.          WHILE(!Stack2.empty())
10.         n = Stack2.pop_back();
11.         IF n is leaf
12.             n.primitive->intersectP(Ray)
13.         ELSE
14.             IF n.BBox->IntersectP(Ray)
15.             Stack2.push(n->right)
16.             Stack2.push(n->left)
17.             ENDIF
18.         ENDIF
19.         ENDWIHLE
20.     ELSE
21.         IF N.BBox->IntersectP(Ray)
22.             Stack.push(N->right)
23.             Stack.push(N->left)
24.         ENDIF
25.     ENDIF
26. ENDWHILE
27. END
```

5　实验及其结论

本文实现了上述的算法，并对比了本文算法相对于传统的 BVH 与 KD-Tree 的执行效率. 工作站使用了 Intel Core I5 的处理器，并配有 8GB 的内存.

5.1 值 α 的选取

首先为了取得最优的代价函数，这里选取了不同的 α 值来测定对于相同的场景下，不同参数对于渲染时间的影响，场景选择为图 3(a). 实验结果见图 1. 对于式 2 中 C_t 的值取 1，C_i 选取 80,这是由于，相对于求交的代价，光线的传播代价总是非常小的. 当 α 为 1 和 0，代价函数分别只考虑了面积因素和距离因素，效果较差. 最后，由实验结果得出，α 取 0.6 时构造出的代价函数是最优的.

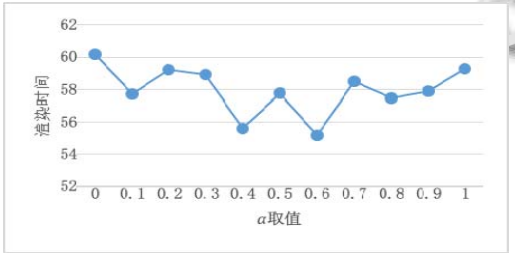


图 1　取值对渲染时间的影响

5.2 阈值 δ 的选取

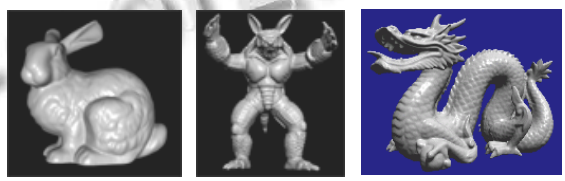
阈值 δ 应该谨慎的选取，过小的值导致加速不起作用，过大的值又会使得构建时间太长. 为了判定最优的阈值 δ，在固定场景下，实验了不同 δ 的构建时间与渲染时间，此处选择了 α=0.6，并以 δ= 1 为基准，场景为图 3(a). 最后实验如图(2)所示，阈值的选择需要权衡构建时间与渲染效率，δ 取 32 或 64 时较为合适，值再高时，加速效果不再明显，但是构建的时间却过长.

5.3 结论

为了对比本文算法与传统 KD-Trees 和 BVH 算法，这里选择了 3 个场景，如图 3 所示. 在生成图像分辨率都为 640*480 的情况下，实验结果为表(4).

本文算法相对于 BVH 加速为 6%~9%，但是传统的 BVH 有着更快的构建速度，在场景 b 中，本文中的构建算法比其慢了 6 倍. 本文算法相对于 KD-Trees 的加速为 4%~6%，这是由于本文的前期构建与 KD-Trees 相同，所以加速效果并不明显. 但是相对于 KD-Trees，由于每次自底向上的构建 BVH 并不影响后续的空间划分，本文的构建时间只比其慢 2 倍左右.

图 2　阈值 δ 对构造时间与渲染加速的影响



(a)Bunny　(b)Armadillo　(c)Dragon

图 3　测试场景

表 4　不同算法渲染时间对比

场景	三角形面片数	KD-Tree 渲染 (s)	BVH 渲染 (s)	本文渲染 (s)
(a)	69,451	57.4	60.8	55.2
(b)	1,132,830	96.3	99.2	90.3
(c)	345,944	75.2	77.2	71.9

对于本文中算法的有效改进为，避免使用栈结构来遍历节点，这是由于进出栈的操作很消耗时间，一种可行性为用数组来模拟栈的行为，或者改进文献[7]中提出的无栈遍历的方法,这些操作对于减少渲染时间是很有益处的.

最后，本文中算法虽然在渲染时较传统的 BVH 与 KD-Trees 有更高的效率，但缺点在于构建的时间较长. 这是由于自底向上构建不得不使用聚类的方式. 所幸的是，每一个节点中图元的数量都不多，这样可以在相对较小的时间内来构建. 一种更加有效的方式是使用现代的 GPU 中通用计算的能力，来并行的聚类，这种方式为改进本文中的算法提供了良好的思路.

参考文献

1 M. Pharr,G. Humphreys. Physically Based Rendering.3rd ed, San Francisco: Morgan Kaufmann, 2010: 221–298.

2 GOLDSMITH J, SALMON J. Automatic creation of object hierarchies for ray tracing. IEEE Computer G-raphics and

Applications, 1987, 7(5): 14–20.

3 Ernst M, Greiner G. Early split clipping for bounding volume hierarchies. 2007 IEEE/EG Symposium on Interactive Ray Tracing. 2007. 73–78.

4 Dammertz H, Keller A. The edge volume heuristi-crobust triangle subdivision for improvedBVH performance. 2008 IEEE/EG Symposiumon Interactive Ray Tracing. 2008. 155–158.

5 Stich M, Friedrich H, Dietrich A. Spatial splits in bounding volume hierarchies. Proc. of the Conference on High Performance Graphics. 2009. 7–13.

6 Wald I. On fast Construction of SAH-based bounding volume hierarchies. 2007 IEEE/EG Symposium on Interactive Ray Tracing. 2007. 33–40.

7 Áfra At, Szirmay-Kalos L. Stackless multi-BVH traversal for CPU, MIC and GPU ray tracing. Computer Graphics Forum, 2014, 33(1): 129–140.

8 杨鑫,王天明,许端清.基于 GPU 的层次包围盒快速构造方法,浙江大学学报(工学版),2012,46(1):84–89.

9 Barringer R, Akenine-Müller T. Dynamic ray stream traversal. ACM Trans. on Graphics, 2014, 33(33): 1–9.

10 Doyle MJ, Fowler C, Manzke M. A hardware unit for fast SAH-optimised BVH construction. ACM Trans. on Graphics, 2013, 32(4): 139:1–139:10.

11 Ize T, Wald I, Parker SG. Asynchronous BVH construction for ray tracing dynamic scenes on parallel multicore architectures. Proc. of the 7th Eurographics Conferenceon Parallel Graphics and Visualization. Aire-la-Ville, Switzerland. 2007. 101–108.

12 Gu Y, He Y, Fatahalian K. Efficient BVH construction via approximate agglomerative clustering. Proc. of the 5th High-Performance Graphics Conference. New York. 2013. 81–88.