软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(Suppl.(2)):158−167

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

一种快速高质量 k-D 树构建算法*

钱银玲[1,2,4], 梁 晓[2,3,4], 张严辞[2,4+]

[1](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[2](四川大学 计算机学院,四川 成都 610065)

[3](西南石油大学 计算机科学学院,四川 成都 610500)

[4](四川大学 视觉合成图形图像技术国防重点学科实验室,四川 成都 610065)

Fast High Quality k-D Tree Construction Algorithm

QIAN Yin-Ling[1,2,4], LIANG Xiao[2,3,4], ZHANG Yan-Ci[2,4+]

[1](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[2](College of Computer Science, Sichuan University, Chengdu 610065, China)

[3](College of Computer Science, Southwest Petroleum University, Chengdu 610500, China)

[4](State Key Laboratory of Fundamental Science on Synthetic Vision, Sichuan University, Chengdu 610065, China)

+ Corresponding author: E-mail: yczhang@scu.edu.cn

Qian YL, Liang X, Zhang YC. Fast high quality k-D tree construction algorithm. *Journal of Software*, 2012,23(Suppl.(2)):158−167 (in Chinese). http://www.jos.org.cn/1000-9825/12035.htm

Abstract: This paper proposes a high quality k-D tree constructing algorithm for interactive dynamic scene ray tracing. Combined with the fact that scene primitives were usually well distributed, a reasonable formula is derived from the basic k-D tree traversal cost function to represent the traversal cost for sequential positions in node. After dividing the bounding box of splitting node to uniform bin series, the best splitting position is directly computed by resolving the analytic solution of the cost formula. To guarantee superior rendering performance for different ray tracing applications, the study raised several sensible functions for sub-space number computing in different situations. Experimental results proved that this algorithm was suitable for scenes with various kinds of primitive distribution, and it can be used for full k-D tree construction. Great construction efficiency was obtained with the best splitting quality maintained.

Key words: ray tracing; k-D tree; space splitting; acceleration structure

摘 要: 提出了一种适用于交互式动态场景光线跟踪的高质量k-D树构建算法.结合基本的k-D树遍历代价函数和场景一般具有较均匀分布的特征,推导出合理表示节点中连续分割面遍历代价的计算公式.在计算过程中将待划分节点包围盒划分成均匀子空间,采用直接求解函数解析解的方式进行节点分割面计算.为了保证各种光线跟踪应用中较高的渲染效率,提出了不同情况下合理的空间划分数量计算函数.实验结果表明,该算法适用于各种不同图元分布的场景 k-D 树构建,并可以应用于整个构建过程,效率有了很大的提升,同时构建结果保持了与最优划分相近的质量.

* 基金项目: 国家自然科学基金(60903118, 60832011, 61103137)
收稿时间: 2012-05-20; 定稿时间: 2012-09-29

关键词: 光线跟踪;k-D 树;空间划分;加速结构

光线跟踪作为生成相片级虚拟图像的重要全局光照算法之一,一直是计算机图形学的研究热点.在光线跟踪算法中,设计实现高效的加速数据结构是提高光线与复杂场景数据求交效率的重要方法.目前被广泛采用的加速结构有基本网格、k-D 树、层次包围盒等.对于静态场景来说,k-D 树是公认最快的加速结构[1],而其构建方式对场景遍历效率有很大影响.利用空间或图元中分的方法构建 k-D 树,虽然速度较快,但构建结果质量较低,会导致场景遍历速度大幅下降[2,3].表面积启发规则(surface area heuristic,简称 SAH[2])是目前应用最广泛的 k-D 树构建规则.其中,最优 SAH 构建方法[4]计算每个图元在分割轴上最小和最大投影位置的 SAH 代价,并选取代价最小的位置为分割面.这种规则构建的 k-D 树能够对各种空间分布的场景进行合理划分,但其巨大的计算量难以满足交互式光线跟踪的要求.为了提高构建效率,一些简化的 SAH 构建方法[5−8]被提出来.它们虽然达到了降低构建代价的效果,但遍历效率也受到一定的影响.对于动态场景,以上两类构建方法都不能很好地满足要求.

本文提出了基于 Bin 的 k-D 树构建算法,通过采用合理的函数计算最优分割位置,在构建过程中也无须利用最优 SAH 构建算法,从根本上避免了算法中大量候选分割面的代价计算,并保持与最优 k-D 树相近的质量.本文的主要贡献是:利用合理的函数表示每个 Bin 中的连续位置的 SAH 代价,以快速计算最优 SAH 分割面;针对静态或动态场景以及不同的绘制要求(是否具有阴影、反射和折射光线),提出了不同的 Bin 数量计算方法,从而保证较高的渲染效率.

1 相关工作

SAH 是应用最为广泛的加速结构遍历代价评估模型,其关键在分割平面的选择和左右两边图元数的统计.Wald[4]提出记录所有候选分割面,按照位置进行排序,将最优 SAH 构建算法的复杂度从 Pharr[9]提出的 $O(N\log^2 N)$降低为 $O(N\log N)$(N 为场景中的图元数),但整个过程需要进行大量内存读写操作.Tseng[10]扩展 $O(N\log N)$算法用于直接处理三角形簇,进一步提高了构建效率.Kang[11]将场景分成很多 Group,建立各自的 k-D 树,并在上层构建层次结构,能进行快速更新和遍历.

为了避免大量候选分割面代价计算,学者们提出了各种简化的 k-D 树构建算法.Hurley[12]首次提出将连续空间离散分割采样来减小候选分割平面集,快速求解全局最小值,也就是基本的 Bin 构建算法.Hunt[5]提出了二次混合的均匀采样方法,对第一划分中图元数量变化较大的子空间进行二次划分,最终生成分段二次函数来近似代价函数,将分段函数的误差限定在一定范围内.Popov[6]使用类似方法,多次划分节点包围盒以满足构建精度的要求.Zhou[13]提出了一种新的混合构建方法,采用空间中分或者划分不含图元的空间.针对离散采样中搜索分割平面的盲目性且设置采样点的有限性,Fan[7]采用启发式的方法定位当前节点分割平面所在子区间,并在探查到的子区间进一步细化采样,以得到更优的分割平面.Guo[14]结合模拟退火算法快速搜索最优分割平面.Havran[15]和 Wachter[16]使用 k-D 树和层次包围盒构建混合加速结构的方式减小计算量.以上方法为了避免遍历代价的大幅提高,在构建底层都利用最优 k-D 树构建算法进行划分.

为了进一步提高 k-D 树构建效率,并且保证 k-D 树质量,出现了很多结合场景预知信息的构建算法.Gunther[17]研究可变形场景中的半结构化运动,例如骨骼动画,采用运动分解[18,19]的思想对场景运动分类,再利用运动的帧间连续性建立实时加速结构.Djeu[20]使用预定义场景图(scene graph)来加速构建.Hunt[21]利用场景树信息,在对场景做出合理假设的情况下,时间复杂度为 $O(n)$.但这一类方法局限于某种场景的应用或需要预定义信息,不具有通用性.

随着多核 CPU 和 GPU 的快速发展,并行构建逐渐成为研究热点.并行方法的难点在于并行模块之间实现负载均衡,减少通信和异步操作[22].Shevtsov[8]率先成功地在多核 CPU 上实现了 k-D 树的并行构建,并利用负载均衡算法提高了构建效率.Zhou[13]在 GPU 上实现了空间中分和无图元空间划分的混合构建算法,相对于 Hunt[5]提出的简化的构建算法有了较大的性能提升.Choi[23]提出了合理的并行计算策略:在节点较少时进行广度划分,当节点数达到一定个数时进行深度划分,将 Wald[4]中复杂度为 $O(N\log N)$的算法移植到多核 CPU 上,并提出一种

“In-Place”算法减少内存操作以提高并行扩展性能.Danilewski[22]将基本 Bin 算法移植到 GPU,根据空间中图元数将构建过程分为 5 个阶段,以充分利用 GPU 的并行计算性能.Wu[24]在 GPU 上实现了最优 SAH 算法,对子事件列表的生成过程进行了优化.

2 基于 Bin 的 k-D 树构建算法

如式(1)所示,利用 k-D 树进行光线跟踪的代价 C_{RT} 可以表达为构建代价 C_{Build} 和遍历代价 C_{Trav} 之和.

$$C_{RT}=C_{Build}+C_{Trav} \tag{1}$$

C_{Build} 和 C_{Trav} 是两个矛盾的目标,往往构建代价越低的 k-D 树,其遍历代价越高.但对于静态场景来说,一般忽略 C_{Build} 而只需考虑最小化 C_{Trav};而动态场景则需要在 C_{Build} 和 C_{Trav} 之间折中.

本文算法的基本思想是,在尽量不降低 C_{Trav} 的同时,尽量减少 C_{Build} 代价,以实现交互式的动态场景光线跟踪.算法建立在以下场景特征的基础上:一般场景局部具有较均匀的图元分布.可以利用这一特征建立局部连续分割面的 SAH 代价计算函数快速计算最优分割面.

为了衡量 C_{Trav},对于包围盒表面积为 $SA(V)$的待划分节点,假设分割面 P 左右的图元数分别为 N_L 和 N_R,对应包围盒表面积分别 $SA(V_L)$和 $SA(V_R)$,分割面 P 对应的遍历代价 C_P 为式(2).

$$C_P=C_T+C_I\left(\frac{SA(V_L)}{SA(V)}N_L+\frac{SA(V_R)}{SA(V)}N_R\right) \tag{2}$$

其中,C_T 和 C_I 分别是节点的遍历代价和光线图元的求交代价,$\frac{SA(V_L)}{SA(V)}$ 和 $\frac{SA(V_R)}{SA(V)}$ 为光线与左右子空间的相交概率[25].

本文将节点包围盒划分成一系列均匀的 Bin,对于图元分布较均匀的空间,利用拟合函数求解最优划分位置可以得到更好的划分结果.为了推导出合理的代价函数,本文以基本 SAH 代价计算公式为基础,结合图元变化规律计算公式中的相关变量,得到用于求解最优分割面的目标函数,计算最优分割面转化为目标函数最小化的问题,避免了最优 SAH 构建中大量候选面的代价计算,对应的解析解也就是所需求解的最优分割面.

2.1 Bin的SAH代价计算公式

如图 1 所示的场景,从 P_0 到 P_1 之间任何位置的左右图元数不变,如果光线不是均匀分布并且穿过整个场景,则 SAH 代价不是关于分割位置的线性函数,此时最优分隔位置则可能是 P_0 和 P_1 之间的位置 P',而并非如文献[1]所述一定是 P_0 或 P_1 中的一个.

针对图 1,利用函数拟合分割位置的 SAH 代价则可能得到代价更小的分割位置.如图 2 中分割面 A 和 B 之间宽为 $step$ 的 Bin,根据 START 和 END 事件[4]数列表可以知道当前 Bin 中的事件数 N_S 和 N_E,同时可以计算得到分割面 A 的左、右图元数(计算过程见第 2.2.2 节),分别设为 N_{la} 和 N_{ra}.因为假设局部空间中事件均匀分布,所以对于图 2 中的分割面 Split,到分割面 A 的距离为 d,左、右图元数 N_l 和 N_r 如式(3)所示.

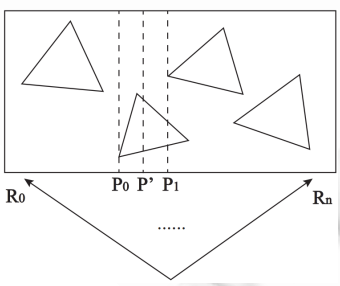


Fig.1 Least traversal cost split position

图 1 最小遍历代价分割位置

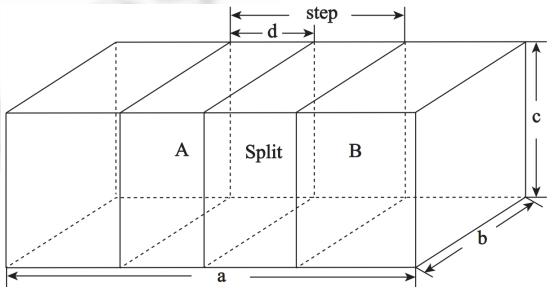


Fig.2 Compute SAH value

图 2 计算 SAH 代价

$$N_l = N_{la} + N_S \times \frac{d}{step},\ N_r = N_{ra} - N_E \times \frac{d}{step} \tag{3}$$

根据节点包围盒在分割面 A 的左、右侧的子包围盒表面积 S_{la} 和 S_{ra},可以计算分割面 Slice 左右子包围盒的表面积 S_l 和 S_r,如式(4)所示.

$$S_l = S_{la} + d \times bc,\ S_r = S_{ra} - d \times bc \tag{4}$$

将式(3)和式(4)代入式(2),整理可得分割轴上的近似 SAH 代价,如式(5)所示.

$$C_{\text{Split}} = C_T + \frac{C_I}{S_A}\left\{\frac{bc}{step}\left(N_S + N_E\right)d^2 + \left[bc\left(N_{la} - N_{ra}\right) + \frac{N_S S_{la} - N_E S_{ra}}{step}\right]d + N_{la}S_{la} + N_{ra}S_{ra}\right\} \tag{5}$$

式(5)即表示当前 Bin 中分割轴上任意位置的 SAH 代价,可以很容易地求出最小值,即为当前 Bin 中的最小 SAH 代价,根据此时的 d 值计算得到最优分割面的位置.

计算式(5)的最小值可以做一定的优化.当 Bin 中 START 和 END 事件数均为 0 时,式(5)可以简化为式(6).

$$C = C_T + \frac{C_I}{S_A}\left[bc\left(N_{la} - N_{ra}\right)d + N_{la}S_{la} + N_{ra}S_{ra}\right] \tag{6}$$

式(6)是关于分割面偏移值 d 的线性函数,当利用式(6)进行 SAH 计算时,只需考虑 d 的系数符号,最优分割面为 A 或 B 之一,降低了一定的计算量.

2.2 算法实现细节

和一般 k-D 树构建方法一样,本文算法采用自上而下的迭代方式进行节点的划分,包含生成 Bin 事件数列表、计算最优分割面和生成子节点图元列表这 3 个步骤.考虑到一般情况下最小 SAH 分割面处于节点空间的最长轴上,所以节点划分的所有步骤都是针对节点包围盒最长轴进行.

2.2.1 生成事件数列表

事件数列表包含每个 Bin 中 START 事件和 END 事件的数量.为了避免复杂的图元事件计算,本文和 Shevtsov[8]及 Benthin[26]一样,利用图元的紧凑包围盒计算事件数列表,统计方式如图 3 所示.

将节点包围盒均分为 N_{sl} 个 Bin,N_{sl} 值的确定将在第 3.2 节做详细分析,扫描节点中图元的包围盒列表,在划分轴上,将每个包围盒最小位置所处 Bin 的 START 数目加 1,相应地将最大位置对应 Bin 的 END 数目加 1.具体实现见算法 1 中的伪代码.

算法 1. 计算事件数列表.

1: *ComputeSliceEventNumber*(*Box*;□;*axis*;*step*;N_{sl})
return (N_S;N_E)
{*Box*—图元包围盒列表,□—节点包围盒}
{*axis*—分割轴,*step*—Bin 间距,N_{sl}—Bin 总数}
2: $N_S \leftarrow 0$; $N_E \leftarrow 0$ {重置计数器列表}
3: **for all** box_i □ *Box* **do**
4: index$\leftarrow \left\lfloor \frac{box_i.\min[axis] - \square.\min[axis]}{step} \right\rfloor$
{处理开始事件}
5: **if** *index*≥0 **then**
6: **inc** N_S[*index*+1]
7: **else**
8: **inc** N_S[0]
9: **end if**

10: $index \leftarrow \left\lfloor \frac{box_i.\max[axis] - \square.\min[axis]}{step} \right\rfloor$

{处理结束事件}

11: **if** $index<N_{sl}$ **then**

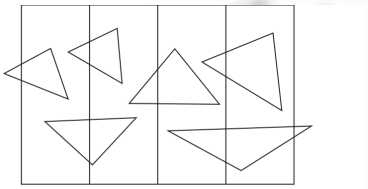
12: **inc** $N_E[index+1]$

13: **end if**

14: **end for**

15: **return** $(N_S;N_E)$

算法 1 中第 4 行~第 9 行计算包围盒 START 事件所处 Bin 序号,然后将相应 Bin 的 START 数加 1,当序号小于 0 时,将 $N_S[0]$加 1.在最终的计算结果中,$N_S[0]$表示将包围盒最小位置作为分割面时左侧的图元数,同时右侧图元数为节点中的图元总数,这是最优分割面计算的基准值.



START	1	2	1	2	0	0
END	0	1	2	1	1	0

Fig.3 Generate event count list

图 3 生成事件数列表

2.2.2 计算最优分割面

这一节将讨论 Bin 最优分割面的计算过程.根据 Bin 的 SAH 计算公式和事件数列表,遍历事件数列表,得到每个 Bin 起始位置的左右图元数,然后根据上一节中的 SAH 计算公式,得到 Bin 内部的最优分割面.计算过程见算法 2.

算法 2. 计算最优分割面.

1: FindBestPlane(2;*NS*;*NE*;*NT*;*Nsl*)

return $(C';pos)$ {N_T—节点图元总数}

2: $C' \leftarrow N_T * C_I$; $N_{la} \leftarrow N_S[0]$; $N_{ra} \leftarrow N_T$

3: $S_{la} \leftarrow 2bc$; $S_{ra} \leftarrow S_A$

4: **for** $i \leftarrow 1 \ldots N_{sl}$ **do**

5: **if** $N_S[i]+N_E[i]>0$ **then**

6: $(C;pos) \leftarrow LeastCost(N_{la};N_{ra};S_{la};S_{ra};N_S;N_E)$

7: **else**

8: $(C;pos) \leftarrow SimpleLeastCost(N_{la};N_{ra};S_{la};S_{ra})$

9: **end if**

10: **if** $C<C'$ **then**

11: $(C';pos) \leftarrow (C;pos)$

12: **end if**

13: $N_{la} \leftarrow N_{la}+N_S[i]$; $N_{ra} \leftarrow N_{ra}-N_E[i]$

14: $S_{la} \leftarrow S_{la}+step*bc$; $S_{ra} \leftarrow S_{ra}-step*bc$

15: **end for**

16: **return** (N_S;N_E)

算法 2 中第 2 行进行计算初始化,节点最小代价 C'的初始值为节点图元数乘以光线三角形求交代价,第 5 行~第 6 行和第 7 行~第 8 行分别根据式(5)和式(6)计算 Bin 最小 SAH 代价和相应分割面.第 11 行~第 12 行更新计算公式中参数值,用于计算下一个 Bin 的最优分割面计算.

计算得到最优分割面后,如果需要划分,则生成新的孩子节点和对应的图元列表,否则直接生成叶子节点.

2.2.3 确定划分终止条件

传统 k-D 树构建算法的终止条件可直接由算法本身确定,不存在最优分割面即停止划分.利用 Bin 进行划分时,即使底层节点中图元很少时仍会计算得到最优分割面,从而导致图元数迅速增长.为了解决这一问题,本文采用 Pharr[9]和 Kang[11]提出的方案,限定 k-D 树的最大深度为 $8+1.3\times\log_2 N$.

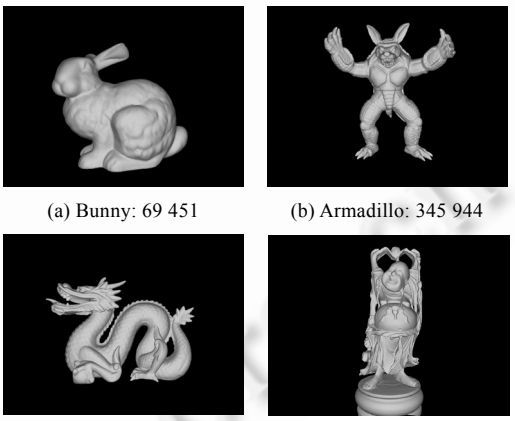
3 实验结果

本文的实验环境为 Intel I3-2100 3.10GHz CPU,4.0G 内存.所有计算都使用基本单线程,节点划分计算的图元阈值均设为 1,生成大小为 800×600 像素的平滑着色图像.用于比较的基准算法是 Wald[4]提出的复杂度为 $O(N\log N)$的最优 k-D 树构建算法.

3.1 静态场景测试

本文首先对文献[27]中的均匀模型进行了测试,渲染结果和相应的模型名称及图元数如图 4 所示.算法采用的切片数为 $0.4n$(n 为节点中图元数量),测试结果见表 1.

表 1 中的数据表明,本文算法的平均构建效率为基准算法的 2 倍,并且遍历效率高 10%左右.遍历效率提高是因为:具有最小 SAH 代价的分割面是理论上最优的 k-D 树划分位置,场景空间中光线均匀分布(SAH 代价计算公式的假设条件[2]之一)在通常情况下并不成立,所以利用式(2)计算得到的最小 SAH 代价位置不一定是实际最优分割位置.图 4 中的模型具有相对均匀的顶点分布,所以为了验证本文算法也适用于非均匀分布的场景,本文测试了一系列非均匀分布场景,其结果和模型信息如图 5 所示,实验数据见表 2.



(a) Bunny: 69 451 (b) Armadillo: 345 944

(c) Dragon: 871 414 (d) Buddha: 1 087 716

Fig.4 Well distributed scenes

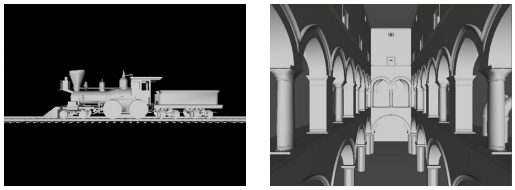
图 4 均匀分布场景

从表 2 可以看出,对于非均匀分布的场景,本文算法可以达到基准算法平均 2.5 倍的加速,同时遍历效率基本保持不变.对比表 1 和表 2 中的数据,非均匀场景本文算法能够得到更大的加速比,而遍历效率加速有所下降,这是因为利用基准算法进行 k-D 树构建对每个图元计算两个位置的 SAH 值,不考虑空间中图元分布情况,而本文算法的划分与空间中的图元数相关,能够更好地适应局部图元密度变化,同时因为图元分布不均匀,利用本文算法计算出的最优划分位置也就存在较大的偏差,导致遍历加速比下降.

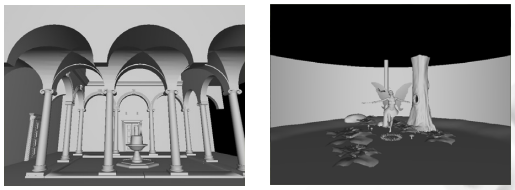
Table 1 Experimental data of well distributed scenes

表 1 均匀分布场景测试数据

场景名	构建时间(ms)		渲染时间(ms)		构建加速比	渲染加速比
	本文算法	基准算法	本文算法	基准算法		
Bunny	221	461	251	275	2.08	1.10
Armadillo	1 458	2 602	258	285	1.78	1.10
Dragon	2 774	5 886	304	333	2.12	1.09
Buddha	3 375	7 263	284	300	2.15	1.06



(a) Train: 6 598 (b) Sponza: 67 461



(c) Closier: 81 354 (d) Fairy: 173 397

Fig.5 Ill distributed scenes

图 5 非均匀分布场景

Table 2 Experimental data of ill distributed scenes

表 2 非均匀分布场景测试数据

场景名	构建时间(ms)		渲染时间(ms)		构建加速比	渲染加速比
	本文算法	基准算法	本文算法	基准算法		
Train	12	30	149	143	2.44	0.96
Sponza	151	366	987	1059	2.42	1.07
Closier	186	482	1300	1347	2.58	1.04
Fairy	426	932	507	521	2.19	1.03

3.2 确定Bin数量

为了解决基本 Bin 划分算法因采用固定数量分割面而不适应场景规模的问题,本文通过分析实验结果,给出不同场景复杂度或者光线跟踪要求下应该采用的划分数量计算函数,使得算法能够用于完整 k-D 树构建过程,并保持较高的构建质量.

首先本文详细测试了图 4 和图 5 中的所有模型.测试过程中只进行了主光线测试,测试的划分数量计算函数为 $\log_2 n$,n,$\log_2^2 n$ 和 $n\log_2 n$(系数分别取 1.0,0.4,0.3,0.1).C_{Build},C_{Trav} 及 C_{RT} 的测试结果分别如图 6~图 8 所示.

从图 6 可以看出,均匀场景使用 $n\log_2 n$ 进行包围盒划分具有最高构建效率,非均匀场景的构建加速比具有不稳定性,但相对均匀场景能得到较高的构建加速.图 7 表明,所有函数都能得到与最优划分相近或更高的遍历效率,也证明了利用与图元数相关的划分数量能够很好地解决基本 Bin 算法中构建质量下降的问题.对于图元非均匀分布的场景,各种划分函数对应的构建质量都波动较大,是因为图元分布很不均匀导致构建结果不稳定,但仍然保持较高的质量.分析图 8 可知,当对均匀场景只进行主光线跟踪时,数量为 n 的划分能够得到稳定的最高总效率,并且场景的复杂度直接影响总的提升效率,导致这一现象是由于图元数越多,k-D 树构建耗时所占比例就越大,而本文算法对构建效率有较大幅度的提升.

综合以上分析可以得出以下结论:

- 选取分割数量为 n 一般能够得到较高质量的 k-D 树,适用于静态场景和需要进行复杂次光线计算的动态场景光线跟踪;
- 利用 $\log_2^2 n$ 函数计算分割数量能够更好地适应不均匀的图元分布,保证在一定质量下,更大幅度地降低构建时间;
- 对具有相对均匀分布的静态场景,$n\log_2 n$ 数量的分割面能够得到较高的总效率.

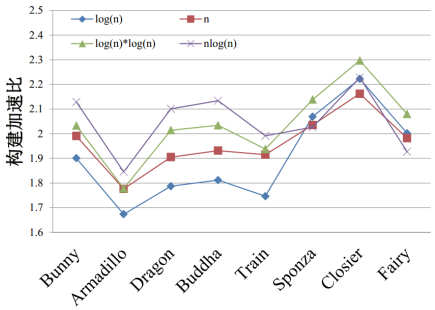


Fig.6 Construct acceleration ratio

图 6 构建加速比

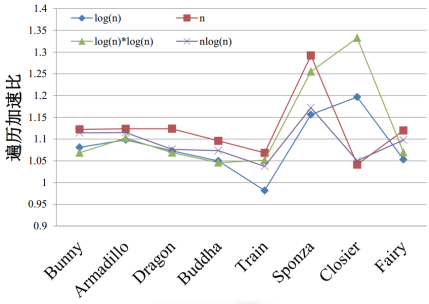


Fig.7 Traversal acceleration ratio

图 7 遍历加速比

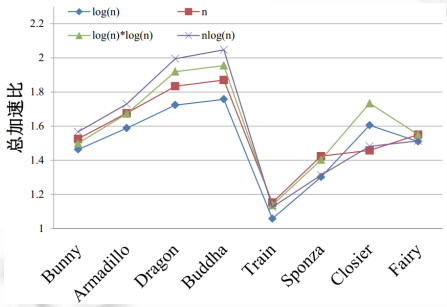


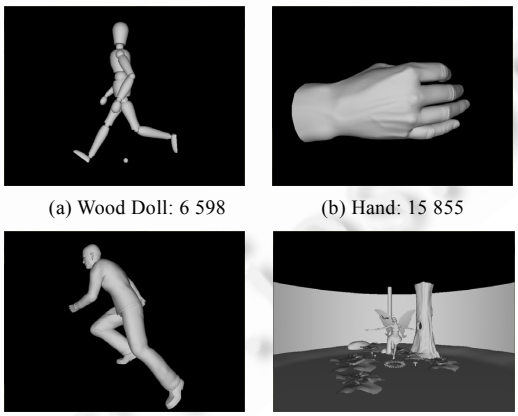
Fig.8 Construct acceleration ratio

图 8 总加速比

3.3 动态场景测试

为了验证本文算法对动态场景的适用性,我们对文献[28]中的动态模型进行了测试,渲染结果如图 9 所示.

从表 3 中的测试结果可以看出,对不同规模的场景,本文算法都得到了一定程度的效率提升,并且模型中图元数越多,提升效果越明显.



(a) Wood Doll: 6 598 (b) Hand: 15 855

(c) Ben: 78 029 (d) Fairy: 174 117

Fig.9 Dynamic scenes

图 9 动态场景

Table 3 Experimental data of dynamic scenes

表 3 动态场景测试数据

场景名	FPS		加速比
	本文算法	基准算法	
Wood Doll	7.07	6.50	1.09
Hand	3.68	3.24	1.14
Ben	2.87	1.85	1.50
Fairy Forest	1.11	0.74	1.51

3.4 多线程及GPU实现讨论

本文算法在对顶点进行划分计算时,分别计算每个图元所属的 Bin,更新相应事件数列表,然后针对每个 Bin 计算最优分隔位置,最终选取最小代价分割面.每一个步骤的计算都具有良好的独立性,所以能够很好地移植到多线程或者 GPU 实现.

4 结 论

本文提出了基于 Bin 的 k-D 树构建算法,分析了最优 SAH 构建算法中采样空间的实际缺陷,根据基本 SAH 代价计算公式推导出合理表示 Bin 中分割位置的表示函数,高效地构建了 k-D 树,并且得到与最优 k-D 树相近或者更高的质量.另外,为了扩展本文算法用于各种光线跟踪应用场景,给出了不同情况下 Bin 划分数量选取的合理意见.

References:

[1] Havran V. Heuristic ray shooting algorithms [Ph.D. Thesis]. Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.

[2] MacDonald JD, Booth KS. Heuristics for ray tracing using space subdivision. The Visual Computer, 1990,6:153−166.

[3] Goldsmith J, Salmon J. Automatic creation of object hierarchies for ray tracing. Computer Graphics and Applications, 1987,7(5): 14−20.

[4] Wald I, Havran V. On building fast kd-trees for ray tracing, and on doing that in $o(n\log n)$. In: Proc. of the IEEE Symp. on Interactive Ray Tracing 2006. 2006. 61−69.

[5] Warren Hunt, Mark WR, Stoll G. Fast kd-tree construction with an adaptive error-bounded heuristic. In: Proc. of the IEEE Symp. on Interactive Ray Tracing. 2006.

[6] Popov S, Günther J, Seidel HP, Slusallek P. Experiences with streaming construction of SAH KD-trees. 2006.

[7] Fan WS, Wang B. Fast KD-tree construction method by probing the optimal splitting plane heuristically. Chinese Journal of Computers, 2009,32(2).

[8] Shevtsov M, Soupikov A, Kapustin A. Highly parallel fast KD-tree construction for interactive ray tracing of dynamic scenes. Computer Graphics Forum, 2007,26(3):395−404.

[9] Pharr M, Humphreys G. Physically Based Rendering: From Theory to Implementation. Morgan Kaufmann Publishers, 2010.

[10] Tseng Q, Jeng SK. A parallel SAH KD-tree construction with fast sifting.

[11] Kang YS, Nah JH, Park WC, Yang SB. gkDtree: A group-based parallel update kd-tree for interactive ray tracing. Journal of Systems Architecture, 2011.

[12] Hurley J, Kapustin A, Reshetov A, Soupikov A. Fast ray tracing for modern general purpose CPU. In: Proc. of the Int'l Conf. Graphicon 2002. 2002. 22−25.

[13] Zhou K, Hou QM, Wang R, Guo BN. Real-Time KD-tree construction on graphics hardware. ACM Trans. on Graphics, 2008,27(5): 1−11.

[14] Guo J, Xu XY, Pan JG. Build Kd-tree for virtual scenes in a fast and optimal way. Acta Electronica Sinica, 2011,39(8).

[15] Havran V, Herzog R, Seidel HP. On the fast construction of spatial data structures for ray tracing. 2006. 71−80.

[16] Wachter C, Keller A. Instant ray tracing: The bounding interval hierarchy. In: Akenine-Moller T, Heidrich W, eds. Rendering Techniques, Eurographics Association. 2006. 139−149.

[17] Gunther J, Friedrich H, Wald I, Seidel HP, Slusallek P. Ray tracing animated scenes using motion decomposition. Computer Graphics Forum, 2006,25(3):517−525.

[18] Lext J, Akenine-Möller T. Towards rapid reconstruction for animated ray tracing. In: Proc. of the Eurographics 2001. 2001.

[19] Wald I, Benthin C, Slusallek P. Distributed interactive ray tracing of dynamic scenes. In: Proc. of the 2003 IEEE Symp. on Parallel and Large-Data Visualization and Graphics. Washington: IEEE Computer Society, 2003. 77−85.

[20] Stoll G, Mark WR, Djeu P, Wang R, Elhassan I. Razor: An architecture for dynamic multiresolution ray tracing. ACM Trans. on Graphics, 2011,30(5):83−108.

[21] Hunt W, Mark WR, Fussell D. Fast and lazy build of acceleration structures from scene hierarchies. In: Proc. of the IEEE Symp. on Interactive Ray Tracing 2007 (RT 2007). 2007. 47−54.

[22] Danilewski P, Popov S, Slusallek P. Binned SAH Kd-tree construction on a GPU. Technical Report, Saarland University, 2010.

[23] Choi B, Komuravelli R, Lu V, Sung H, Bocchino RL, Adve SV, Hart JC. Parallel SAH k-d tree construction for fast dynamic scene ray tracing. In: Proc. of the Conf. on High Performance Graphics. Eurographics Association, 2010. 77−86.

[24] Wu ZF, Zhao FK, Liu XG. SAH KD-tree construction on GPU. In: Proc. of the ACM SIGGRAPH Symp. on High Performance Graphics. ACM, 2011. 71−78.

[25] Santalo L. Integral Geometry and Geometric Probability. Cambridge University Press, 2004.

[26] Benthin C. Realtime ray tracing on current CPU architectures [Ph.D. Thesis]. Saarland University, 2006.

[27] Levoy M, Gerth J, Curless B, Pull K. The Stanford 3D scanning repository. 2005. http://www-graphics.stanford.edu/data/3dscanrep

[28] The Utah 3D animation repository. http://www.sci.utah.edu/~wald/animrep

钱银玲(1989－),男,安徽枞阳人,硕士生,主要研究领域为实时计算机图形学,交互式光线跟踪.

梁晓(1983－),女,博士生,讲师,主要研究领域为计算机图形学,真实感绘制.

张严辞(1975－),男,博士,副教授,主要研究领域为计算机图形学,虚拟现实.