虚拟场景的一种快速优化 Kd-Tree 构造方法

过 洁,徐晓旸,潘金贵
(南京大学计算机软件新技术国家重点实验室,江苏南京 210093)

摘 要: Kd-tree 因其具有场景自适应划分、低存储消耗和快速遍历等优势成为使用最为广泛的加速结构.本文提出一种快速优化的 kd-tree 构造方法,该方法通过分析场景的 SAH 函数,将模拟退火技术使用到最优分割平面搜索过程中加快搜索过程,从而加速 kd-tree 的构造过程.实验表明,通过本文的方法可以在保证构造的 kd-tree 的质量情况下有效加快构造速度.同时,本文实现了该方法的一个多核并行扩展,利用多核 CPU 的并行处理能力,进一步加快了 kd-tree 的构造过程.

关键词: 虚拟场景; kd-tree; 加速结构; 模拟退火; 并行计算
中图分类号: TP391 **文献标识码:** A **文章编号:** 0372-2112 (2011) 08-1811-07

Build Kd-Tree for Virtual Scenes in a Fast and Optimal Way

GUO Jie, XU Xiao-yang, PAN Jin-gui
(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu* 210093, *China*)

Abstract: Kd-tree, due to its adaptive division, low memory consumption and fast traversal, has become the most widely used accelerating structure of the virtual scenes. A fast and optimal kd-tree construction method for virtual scenes is proposed in this paper. By analyzing the SAH function of the scene, simulated annealing is used in the optimal split plane search procedure so as to speed up this procedure. In this way, the kd-tree construction is finally accelerated. Experimental results show that the construction of kd-tree is efficiently accelerated with no loss of kd-tree quality. Furthermore, a multi-core parallel computing extension is implemented in this paper, and with the use of multi-core CPU's parallel processing capabilities, the kd-tree construction is further accelerated.

Key words: virtual scene; kd-tree; accelerating structure; simulated annealing; parallel computing

1 引言

大部分全局光照算法,如光线跟踪、光子映射等,涉及对虚拟场景元素(通常为三角面片)的遍历分析.如果使用强力(brute force)方式依次遍历场景中的所有元素,则将花费大量的时间,以至于大大降低用户与虚拟场景的交互性.解决上述问题的一个有效方法就是为虚拟场景构造空间细分结构,亦称为加速结构.常见的加速结构有均匀网格、八叉树、BVH、BSP、kd-tree 等[1,2].各种加速结构各具优劣,根据文献[1]的测试结果分析,kd-tree 在基于 CPU 的光线跟踪中获得了最好的加速效果.本文重点解决虚拟场景的快速优化 kd-tree 构造问题.

Kd-tree 是一种轴对齐的 BSP 树,其具有场景自适应划分、低存储消耗和快速遍历等优势.特别是对虚拟场景中存在类似"Teaport in the stadium"[3]的空间表示(例如虚拟博物馆某个空旷的展示台上一件精密的展品)具有很好的自适应划分效果.

目前常见的构造 kd-tree 的高效算法主要分为两类,一类是基于三角形排序的算法,另一类是不需要排序的 BIN 算法.前者需要对场景中所有的三角形面片进行排序,然后选取每个三角形面片所对应的轴向包围盒的六个平面作为分割候选平面.利用这种方式构造的 kd-tree 的时间复杂度通常为 $O(N\log^2 N)$,最优情况下可以达到 $O(N\log N)$[4],即三角形面片排序需要的时间复杂度,可以证明,这是排序算法构造时间复杂度的下限.后者无需对场景中的三角形排序,而只需要对所有三角形进行一次遍历,确定每个三角形所在的 BIN,BIN 的边界作为候选分割平面.如果选择 BIN 的数目为 M,则理论上该方法确定最优分割平面的时间复杂度为 $O(MN)$.在 $M << N$ 的情况下,该类方法可以加速构造过程,但是构造的 kd-tree 质量欠佳[5,6].

当确定了候选分割平面之后,需要对每个候选分

收稿日期:2010-05-29;修回日期:2011-01-28
基金项目:江苏省自然科学基金(No.BK2010373);计算机软件新技术国家重点实验室海外开放课题(No.KFKT2009A13)

割平面进行评估.从候选分割平面集中选择最优的分割位置是构造 kd-tree 的核心工作[7].其中 SAH(Surface Area Heuristic)[8]是使用最为广泛的评估函数,接下来一节将具体分析.

2 SAH 分析

SAH 基于几何概率理论,是目前使用最为广泛的评估 kd-tree 分割效果的代价函数.通过 SAH 选取的分割平面具有包围盒表面积最小同时所包含的场景元素最多的特点[9].这样,光线以比较小的概率击中含有大量元素的包围盒,降低求交测试代价.SAH 基于以下三项假设条件[4]:

假设条件 1 场景中的光线是随机均匀分布的直线;

假设条件 2 节点的遍历代价 K_t 和直线与三角形的相交测试代价 K_i 已知;

假设条件 3 与 N 个三角形相交的代价为 NK_i,即叶节点的相交测试代价与其包含的三角形数目成正比.

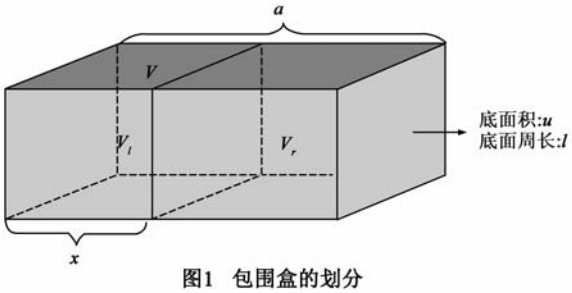


图1 包围盒的划分

如图 1,当前节点包围盒为 V,其分割后的左子节点为 V_l,右子节点为 V_r.根据假设条件 1,在光线击中当前节点的情况下,击中左、右子节点的条件概率分别为

$$P_l = SA(V_l)/SA(V) \tag{1}$$

$$P_r = SA(V_r)/SA(V) \tag{2}$$

其中 $SA(V)$为包围盒 V 的表面积.

根据假设条件 2,如图 1 所示某个节点的总的代价可以表示为

$$C(V) = K_t + P_lC(V_l) + P_rC(V_r) \tag{3}$$

式(3)是一个递归表达式,展开可得构建整棵树的代价为

$$C(V_{\text{root}}) = \sum_{n\in \text{中间节点}} \frac{SA(V_n)}{SA(V_{\text{root}})} K_t + \sum_{l\in \text{叶节点}} \frac{SA(V_l)}{SA(V_{\text{root}})} K_i \tag{4}$$

最优的 kd-tree 就是使式(4)最小的 kd-tree.但是直接求解式(4)计算量太大,在场景图元较多并且无规则分布的情况下基本不可能.一种简化的方法称为局部贪心算法(local greedy algorithm).使用局部贪心算法时,结合假设条件 3,分别以左右子节点中所含的场景图元数目代替其子节点代价,并乘上一个相交测试系数 K_t,即有

$$\begin{aligned}\tilde{C}(V) &\approx K_t + P_l|T_l|K_i + P_r|T_r|K_i \\ &= K_t + K_i\left(\frac{SA(V_l)}{SA(V)}|T_l| + \frac{SA(V_r)}{SA(V)}|T_r|\right)\end{aligned} \tag{5}$$

其中 $|T_l|$ 和 $|T_r|$ 分别表示左右子节点中的图元数目.

定理 1 如果场景分割能够减小 SAH 代价,则使用局部贪心算法计算的节点 SAH 代价不低于其真实代价.

证明 局部贪心算法假设每个分割后的子节点不再分割,从而成为叶节点.根据假设,子节点的分割会降低每个叶节点的 SAH 代价,即

$$C(V_l) \leqslant |T_l|K_i \tag{6}$$

$$C(V_r) \leqslant |T_r|K_i \tag{7}$$

所以

$$\tilde{C}(V) \geqslant C(V) \tag{8}$$

也就是说通过局部贪心法计算的代价比事实上的代价高.上述等号成立条件为某个节点只分割为两个叶节点,或者就是叶节点本身. 证毕.

确定了一条分割轴后,需要在分割轴的开区间(不包括两个端点)内确定一个最佳的分割位置,也就是使式(5)取最小值的位置.一般情况下,式(5)随分割位置的变化而不规则的变化,所以这样的最值点很难通过解析的方式获得.图 2 显示了几个不同的虚拟场景下,SAH 函数随分割位置的变化,相对应的场景来自图 7(a)、(b),横轴为分割位置(场景的 X 轴),纵轴为根据式(5)计算的代价值.

从图 2 中可以看出这些 SAH 函数具有类似 U 形的形状,一般 U 形的谷底即最值点的位置.我们的目的就是以较快的速度找到该最值点.

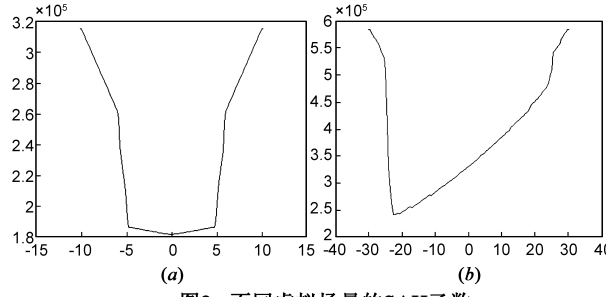


图2 不同虚拟场景的SAH函数

定理 2 如果场景中的三角形面片随机均匀分布,则其 SAH 函数呈现抛物线形状,并在场景分割轴中点取最小值.

证明 假设沿着某轴长为 a 的分割轴分割场景,分割位置离左边界为 x,分割面对应的底面面积为 u,周长为 l,如图 1 所示.同时假设该包围盒对应的场景中包含 T 个三角形面片,其左子结点包含 t 个三角形,则右子节点的三角形片近似为 $T-t$,这样式(5)可以表示为

$$C(x,t)=K_t+K_i\left(\frac{lx+2u}{la+2u}t+\frac{l(a-x)+2u}{la+2u}(T-t)\right)$$

$$=K_t+K_i\left(\frac{l}{la+2u}(2xt-at-Tx)+T\right) \quad (9)$$

如果场景中的三角形随机均匀分布,则

$$t=(T/a)x \quad (10)$$

即左子节点中的三角形数目随分割位置线性变化.综合式(9)和式(10)可得

$$C(x)=K_t+K_iT\left(\frac{2l}{la+2u}\left(\frac{1}{a}x^2-x\right)+1\right) \quad (11)$$

式(11)表示一条抛物线,并在 $a/2$ 取得最小值,该位置也是理想中的最优分割位置. 证毕.

如图 2 所示,现实的 SAH 函数是相当复杂的,一般情况下很难得到它的解析式.而且由于式(5)中 $|T_l|$ 和 $|T_r|$ 的不规则变化,其 SAH 有时严重偏离理想的抛物线形状,复杂 SAH 函数的最值点通常通过离散采样近似获得.三角形排序算法所选取的采样位置为各个三角形的包围盒边界,在三角形数目很多的情况下,这样的采样点很多,影响构造速度,但是可以保证 kd-tree 的质量,因为通常三角形的边界即为 SAH 函数发生非单调变化的位置(即拐点的位置);BIN 方法可以减少采样点,其在开区间内等间距或自适应非等间距[10]设置采样点,并根据这些采样点获得比较好的分割位置.这种情况下一般不会得到最优的分割位置,但是可以大大提高构造时间,只要所选取的采样点远小于场景中的三角面片数目.

3　启发式搜索最优分割平面

Kd-tree 构造的关键步骤是分割平面的确定,前面提到的 BIN 算法会在选定的分割轴上盲目搜索最优分割平面,其搜索空间只是离散有限的几个点.文献[7]提出一种基于 BIN 算法并利用 SAH 函数的启发式搜索最佳分割平面的算法,该方法首先启发式地快速确定包含潜在最佳分割平面的子区间,然后对该子区间进一步细化采样,从而逼近 SAH 最小值.本文提出的启发式搜索算法同样有效利用了场景的 SAH 函数,加快搜索最优分割平面的过程.首先在场景中选取有限的分割位置做采样,并记录这些采样点的相关信息.然后利用类似模拟退火的方法,结合这些采样信息,在分割轴开区间内启发式搜索最优分割位置,理想情况下,可以在较少的搜索候选位置找到接近最优的分割平面.BIN 算法只能在有限位置评估分割效果,而本文的启发式算法通过插值可以在分割轴开区间的任意位置评估分割效果,即分割平面的搜索空间包括整条分割轴区间.

3.1　场景采样

类似 BIN 算法,本文的算法首先需要对场景空间进行采样.当选定了一条分割轴之后,对该分割轴对应的开区间等距离划分,然后依次扫描整个场景包围盒中的所有三角形面片,记录每个面片所对应的 BIN 范围.扫描过程需要记录两个不同的 BIN,分别称为 START-BIN 和 END-BIN. START-BIN 中的每个 BIN 元素记录三角形面片起点位于该 BIN 元素对应的分割采样点左侧的三角形面片数;相反,END-BIN 中的每个 BIN 元素记录三角形面片终点位于该 BIN 元素对应采样点右侧的三角形面片数.

采样可以采用两种方式:一种方式只对包围盒最长轴采样,这样记录的采样点较少,速度较快,但是最终生成的 kd-tree 质量无法保证,因为很多情况下最长轴不一定是最优的分割平面方向;另一种方式即对三个轴向都进行采样,这样虽然产生了三倍的采样点,但是可以保证每次都能找到接近最优的分割位置,最终效果比单向采样好.

本文采用对三个轴向都采样,每条采样轴上 START-BIN 和 END-BIN 记录过程如图 3 所示.

```
1.  procedure RecordBIN()
2.  for each bin i
3.      SB[i] = EB[i] = 0
4.      for each aabb //轴向包围盒
5.          for each split plane sp[j]
6.          if (aabb.boxmin ≤ sp[j])
7.              SB[j] ++
8.          if (aabb.boxmax > = sp[j])
9.              EB[j] ++
```

图 3　记录 BIN 的过程

利用 START-BIN 和 END-BIN 中信息即可统计出式(5)中对应的 $|T_l|$ 和 $|T_r|$.

3.2　计算任意位置的 SAH

根据 START-BIN 和 END-BIN 中信息可以很快计算出每个采样点的 SAH 值,而非采样点的 SAH 值可以通过插值的方式获得.本文不直接插值 SAH 值,而是对 $|T_l|$ 和 $|T_r|$ 进行插值间接计算非采样点的 SAH 值.

假设某个分割候选位置 p 介于分割平面 $sp[i]$ 和 $sp[i+1]$ 之间,p 处的 $|T_l|$ 和 $|T_r|$ 以以下方式线性插值计算

$$\alpha=\frac{p-sp[i]}{sp[i+1]-sp[i]} \quad (12)$$

$$|T_l^p|=SB[i]+\alpha(SB[i+1]-SB[i]) \quad (13)$$

$$|T_r^p|=EB[i]+\alpha(EB[i+1]-EB[i]) \quad (14)$$

将式(13)、(14)代入式(5)就可以计算出 p 点对应的 SAH 值.

3.3　搜索最优分割平面

本文采用类似模拟退火[11~13]的方式启发式搜索

最优分割平面.模拟退火方法是局部搜索方法的扩展,它与局部搜索不同之处在于其以一定的概率接受邻域中代价大的位置,从而可以有效地避免局部极值点.模拟退火算法具有渐进收敛性,已在理论上被证明是一种以某种概率收敛于全局最优解的全局优化算法.本文将该思想引入 SAH 函数的优化求最优值过程:首先在某个分割轴对应的开区间内随机确定一个初始分割候选平面,通过式(5)计算其 SAH,并记录为当前状态.然后在其邻域内随机确定另一个候选分割平面,并评估新候选平面和当前候选平面之间的优劣,如果前者 SAH 小于后者,则前者立即被接受;相反,虽然当前候选平面更优,但是新候选平面依然有一定的概率被接受,如此即可避免局部极小值.如图 4 所示.

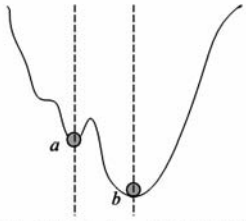


图4 局部极值点a和最值点b,采用模拟退火算法可以从局部极值点a跳到最值点b

使用模拟退火算法启发式求解最优分割位置的主要步骤如图 5 所示.

```
1.  procedure FindClipPlane()
2.  currentPositon = RandomPosition()
3.  currentCost = SAH(currentPosition)
4.  for each axis in {x,y,z} {//三个轴求解
5.      T = T0
6.      while (T > 1){
7.          n = 0
8.          while (n < N_MAX){
9.              newPosition = RandomPosition()
10.             newCost = SAH(newPosition)
11.             if (newCost < currentCost){
12.                 currentPosition = newPosition
13.                 currentCost = newCost
14.             }else{
15.                 r = exp(-(newCost - currentCost)/T)
16.                 if (random(0,1) < r){
17.                     currentPosition = newPosition
18.                     currentCost = newCost}}
19.             n = n + 1}
20.         T = CoolingSchedule (T)}}
```

图 5 模拟退火算法启发式求解最优分割平面的过程

该部分主要用来对 SAH 函数(即式(5))求最小值,从而找到最优的分割平面.温度 T 体现了系统当前的能量状态.当系统处于高能量时,系统不稳定,搜索的过程就是随机化,即使出现 newCost > currentCost,newPosition 也有较高的概率被系统接受,这个体现在第 15、16 步.当温度比较高时,exp(-(newCost - currentCost)/T)比较大,接近于 1,所以 16 步判断成立的可能性非常高,newPosition 被接受.当温度降低时,T 减小,exp(-(newCost - currentCost)/T)也减小,并趋向于 0,所以若 newCost > currentCost,newPosition 被舍弃.这时可以基本确定当前选定的位置已经非常靠近最优分割平面,即系统随着温度的降低收敛于最优分割平面.

RandomPosition()函数在当前候选平面的邻域中随机选择一个新的平面,邻域的确定方式影响搜索的效率.针对本文算法,一种简单的邻域确定方法是使整个分割轴开区间都作为当前候选平面的邻域.这样,每次寻找下一个候选分割平面时,都会从整个分割轴上随机选择一个位置.假设当前分割轴为 a,其左右端点分别为 min(a)、max(a).搜索邻域空间由二元组(astart, alength)表示,astart 表示搜索邻域空间在当前分割轴上的起始位置,alength 表示搜索邻域空间大小.简单的邻域确定方法设定二元组为固定值(min(a), max(a) - min(a)).为了提高搜索效率,可以利用以下方式不断更新二元组(astart, alength),从而缩小搜索空间:

初始化:

(astart, alength) = (min(a), max(a) - min(a))

更新规则:

(1)如果 newCost < currentCost,并且 newPosition < currentPosition, 则 (astart, alength) = (astart, currentPosition - astart);

(2)如果 newCost < currentCost,并且 newPosition > currentPosition,则(astart, alength) = (currentPosition, alength - (currentPosition - astart));

(3)如果 newCost > currentCost,并且 newPosition 被接受,则(astart, alength) = (min(a), max(a) - min(a)).

通过这种方式可以不断地缩小搜索邻域空间,加快收敛速度.

CoolingSchedule()函数用于控制温度的下降变化,随着温度的下降,系统从高温的不稳定状态趋向于低温稳定,搜索过程也将趋向于最优值.常用的降温函数有比例下降法:$T(i+1) = \alpha T(i)$,对数下降法:$T(i) = T_0/\log i$,Cauchy 下降法:$T(i) = T_0/i$,指数下降法:$T(i) = T_0\exp(-C_i)$等[12].实验测试说明,这些不同的降温函数对本文的算法的速度影响不大,所以本文选择简单的比例下降法.初始温度一般选的比较高,这样可以使初始搜索空间尽可能覆盖整个解空间.

3.4 并行设计

随着硬件技术的发展,多核 CPU 已经成为商用电脑配置的主流,多核并行计算[14]可以显著提升计算速度,因此已经在很多方面应用.本文的方法可以有效地

扩展到多核并行处理,以得到性能提升.Kd-tree 并行构造算法已经被很多研究人员提出.文献[6]实现了一种多线程的 BIN 算法,由于其 BIN 是均匀分布的,所以虽然构造速度很快,但是质量不佳.文献[10]提供了使用 SSE 指令的并行构造方法.文献[15]提供了一种在 GPU 上实现的 kd-tree 并行构造方法,获得了非常好的构造效率.文献[9]总结了以前的串行和并行构造算法,提出了两种基于多核计算的并行 kd-tree 构造新算法:嵌套并行算法(nested parallel algorithm)和原地并行算法(in-place parallel algorithm).

本文提供的并行设计方法主要包括场景粗划分和并行细分两个阶段,最后通过组装将所有的 kd-tree 子树组装为一棵完整的场景 kd-tree,如图 6.

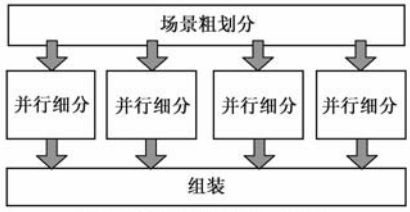


图6　并行构造kd-tree过程

粗划分是并行构造的第一步,其主要目的是快速地分割原始场景,产生多个不相交的子场景,为下一步的并行细分做准备.粗划分既要保证速度,也要在一定程度上保证质量.划分初期所要处理的场景图元很多,如果采用排序算法,则会产生大量的分割候选平面,从而影响速度;使用 BIN 算法不一定能找到最佳的分割点,但是候选平面较少,无需对场景图元排序,所以计算速度快,而且,靠近根节点处的场景分割结果对最终 kd-tree 的效果影响较小.基于此,我们使用 BIN 算法进行粗划分.

第二步,对每个子场景使用本文的算法并行构造子 kd-tree.子 kd-tree 在各个独立的线程中构造,而不需要任何交互.

最后,将所有子 kd-tree 组装起来,构成整个场景的 kd-tree.

4　实验结果及分析

我们在图形工作站上实现了本文的方法,并使用一些虚拟场景进行了测试.图形工作站的硬件配置为 CPU:Intel 至强(Xeon)四核,2.93GHz;RAM:12GB;GPU:NVIDIA Quadro FX 580.操作系统使用 Windows 7,64 位.编译环境为 VS2008.测试用虚拟场景如图 7 所示.

4.1　本文算法与排序算法和 BIN 算法的比较

我们对比了三种 kd-tree 构造算法.排序算法我们采用的是文献[4]提出的时间复杂度为 $O(N\log N)$的算法.BIN 算法实现参考文献[6],BIN 数目 M 设为 64.满足如下之一时达到终止条件,递归构造结束,形成叶节点:

$(a)\ |T| \leqslant TRIANGLE\_TARGET$

$(b)\ depth(V) \geqslant MAX\_DEPTH$

$(c)\ C(V) > |T| \cdot K_i$

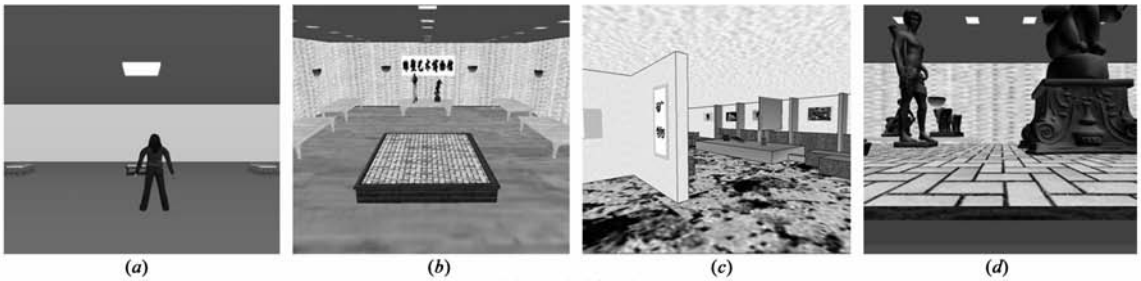
条件(a)表示当前节点中的三角面片个数达到某个阈值;(b)表示当前节点的深度达到最大深度;(c)表示当前节点满足局部最优,无法通过此次分割降低代价.实验中,我们设置 *TRIANGLE_TARGET* 为 1,*MAX_DEPTH* 为 30.运行时间的实验结果参见表 1.

渲染时间表示使用构造的 kd-tree 作为加速结构对图 7 场景进行光线跟踪所消耗的时间,本实验设计的光线跟踪每条屏幕光线只对场景一次求交,而不递归跟踪次生光线,屏幕采样分辨率为 1024 × 1024.渲染时间体现 kd-tree 的质量,渲染时间越短,所构造的 kd-tree 质量越好.

从表 1 中数据可以看出三种方法中 BIN 算法的构造时间最快,但是该方法构造的 kd-tree 质量不佳,而且

表 1　本文算法与排序算法和 BIN 算法运行时间对照

场景	三角面片数	排序算法		BIN 算法		本文算法	
		构造时间/s	渲染时间/ms	构造时间/s	渲染时间/ms	构造时间/s	渲染时间/ms
(a)	63,172	1.71	4691	0.64	6830	0.85	4797
(b)	117,374	2.45	5156	1.15	7406	2.25	5083
(c)	696,228	11.64	4973	4.71	6890	4.67	5236
(d)	1,085,536	17.55	5749	11.53	6879	14.60	5783



(a)　(b)　(c)　(d)

图7　测试场景

不稳定,对某些场景构造的 kd-tree 质量可能很差(如表 1中场景(b)).使用排序算法基本可以得到最优的 kd-tree,但是由于涉及到三角面片排序,同时其候选平面包含所有三角面片的包围盒,所以构造时间较长.使用本文的方法无需遍历所有候选平面,所以可以在排序算法的基础上降低构造时间,而比 BIN 算法使用的时间略大,但其基本保证了构造的 kd-tree 具有与排序算法一样较高的质量.

4.2 并行实现

并行处理部分使用 OpenMP[16] 实现.OpenMP 是对 C 语言的一个支持并行程序设计的扩展,其在 C 程序中加入了 OpenMP 的编译指示,这些编译指示描述了程序应该以何种方式并行执行.

由于本图形工作站 CPU 包含四个核,所以本实验在场景粗划分阶段将原始场景分割为四个子场景,然后生成四个线程分别执行一个子场景的细分过程.表 2 和图 8 分别表示本文原始算法与并行算法构造时间和渲染时间的比较.

表 2 原始算法和并行算法的构造时间比较

场景	原始算法构造时间/s	并行算法构造时间/s	加速比
(a)	0.857	0.307	2.79
(b)	2.281	1.098	2.08
(c)	4.675	2.295	2.04
(d)	14.655	7.214	2.03

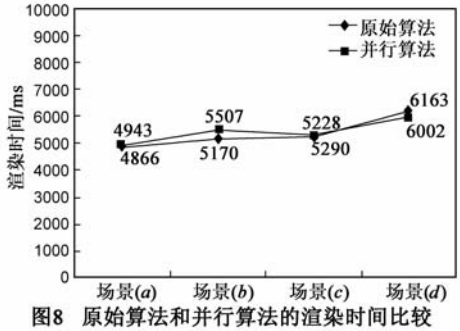


图8 原始算法和并行算法的渲染时间比较

由表 2 和图 8 可以看出,使用并行算法之后可以大大提高 kd-tree 的构造速度,构造时间加速比可以达到 2 以上,同时基本保持 kd-tree 的质量不变.

5 结论与展望

本文提出了一种快速优化的 kd-tree 构造方法,该方法基于 BIN 方法思想,将模拟退火优化计算方法引入到 kd-tree 分割评估的 SAH 函数的求最优值过程中,以加快求解过程.根据一组虚拟场景实验表明最终生成的 kd-tree 在保证质量的情况下,其构造时间有明显下降.同时,本文又实现了该方法的多核并行扩展,通过多核并行计算,kd-tree 的构造速度可以大大加快,而且基本没有损失最终构造的 kd-tree 的质量.

随着 GPU 技术的发展,在 GPU 上实现 kd-tree 的构造和遍历[15,17,18]也成为新的研究方向.由于 GPU 的特殊结构,在线程很多的情况下才能发挥其优势(通常需要 $10^3 \sim 10^4$ 个线程),所以需要特殊设计其并行计算过程.下一步,我们将把本文的方法移植到 GPU 端,以利用其强大的并行计算能力进一步提升 kd-tree 构造速度.

参考文献

[1] Havran V. Heuristic Ray Shooting Algorithms[D]. Ph D Dissertation, Czetch Technical University, Prague, Czetch Republic, 2001.

[2] Pharr M, Humphreys G. Physically Based Rendering: From Theory to Implementation[M]. USA: Morgan Kaufmann, 2004.

[3] J Lext, U Assarsson, T Möller. A benchmark for animated ray tracing[J]. IEEE Computer Graphics & Applications, 2001, 21(2): 22 – 31.

[4] I Wald, V Havran. On building fast kd-Trees for ray tracing, and on doing that in $O(N\log N)$[A]. Proceedings of the IEEE Symposium on Interactive Ray Tracing[C]. USA: IEEE Computer Society, 2006. 61 – 69.

[5] Hurley J, Kapustin A, Reshetov A, Soupikov A. Fast ray tracing for modern general purpose CPU[A]. Proceedings of the GraphiCon[C]. Nizhny Novgorod, Russia, 2002. 1 – 8.

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

[7] 范文山,王斌.启发式探查最佳分割平面的快速 KD-Tree 构建方法[J].计算机学报,2009,32(2):185 – 192.
Fan Wenshan, Wang Bin. A fast KD-Tree construction method by probing the optimal splitting plane heuristically[J]. Chinese Journal of Computers, 2009, 32(2): 185 – 192. (in Chinese)

[8] J D MacDonald, K S Booth. Heuristics for ray tracing using space subdivision[J]. Visual Computer, 1990, 6(3): 153 – 166.

[9] C Byn, K Rakesh, L Victor, S Hyojin, et al. Parallel SAH k-D Tree Construction for Fast Dynamic Scene Ray Tracing[R]. 2009.

[10] W Hunt, W R Mark, G Stoll. Fast kd-tree construction with an adaptive error bounded heuristic[A]. Proceedings of the IEEE Symposium on Interactive Ray Tracing[C]. USA: IEEE Computer Society, 2006. 81 – 88.

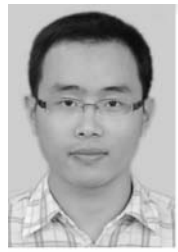
[11] D Bertsimas, J Tsitsiklis. Simulated Annealing[J]. Statistical Science, 1993, 8(1): 10 – 15.

[12] B Suman, P Kumar. A survey of simulated annealing as a tool for single and multiobjective optimization[J]. Journal of the Operational Resarch Society, 2006, 57(10): 1143 – 1160.

[13] 邢文训,谢金星.现代优化计算方法(第 2 版)[M].北

京:清华大学出版社,2005.
[14] Mirman I. Concepts in Multicore Programming[EB/OL]. http://www.cilk.com/multicore-blog/,2009-1-20.
[15] Kun Z, Qiming H, Rui W, Baining G. Real-time KD-tree construction on graphics hardware[A]. Proceedings of ACM SIGGRAPH Asia[C]. New York: ACM, 2008. 126:1 – 126:11.
[16] Quinn M J 著. 陈文光等译. MPI 与 OpenMP 并行程序设计:C 语言版[M]. 北京:清华大学出版社,2004.
[17] Foley T, Sugerman J. KD-tree acceleration structures for a GPU ray tracer[A]. Proceedings of the ACM Siggraph/Eurographics Conference on Graphics Hardware (HWWS)[C]. New York: ACM, 2005. 15 – 22.
[18] Popov S, Gunther J, Seidel H-P, Slusallek P. Stackless kd-tree traversal for high performance GPU ray tracing[J]. Computer Graphics Forum, 2007, 26(3):415 – 424.

作者简介



过　洁　男,1986 年出生于江苏无锡. 2008 年毕业于南京大学计算机科学与技术系,现为南京大学计算机科学与技术系博士生,主要研究领域为虚拟环境和实时绘制.
E-mail: guojie_022@163.com

徐晓旸　男,1981 年出生于江苏常州. 现为南京大学计算机科学与技术系博士生,主要研究领域为虚拟环境和实时绘制.