


Lookof 's Wild

Last of the Wild

博客园 :: 首页 :: 新随笔 :: 联系 :: 订阅  :: 管理

58 随笔 :: 0 文章 :: 121 评论 :: 40万 阅读

公告

昵称: lookof
园龄: 16年3个月
粉丝: 87
关注: 3
+加关注

搜索

找找看

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

DirectX 3D(2)
学习笔记(2)
过程生成(2)
分形(2)
地形(2)
windows编程(1)
vc++(1)
shadow(1)
ogre1.7(1)
ogre(1)
更多

随笔分类

IOS(1)
叨叨叨(6)
计算机相关(6)
家(5)
赏析(1)
数学(10)
图形学(21)
映像(1)
游戏开发(6)
重温C++(5)

随笔档案

2014年2月(1)
2014年1月(3)
2013年9月(1)
2013年3月(2)
2013年2月(1)
2012年5月(2)
2012年2月(1)
2011年12月(2)
2010年3月(3)
2010年2月(1)
2010年1月(1)
2009年12月(3)
2009年11月(4)
2009年9月(4)
2009年8月(3)
更多

相册

pic(1)

Me

lookof's IW
豆瓣

友邻链接

熊哥
虚拟闲人
Gondalman

BVH with SAH (Bounding Volume Hierarchy with Surface Area Heuristic)

- BVH with SAH (Bounding Volume Hierarchy with Surface Area Heuristic) -

0. Overview

包围层次盒(BVH)是一种基于“物体”(object, 区别于“空间”, spatial)的场景管理技术, 广泛应用于碰撞检测、射线相交测试之类的应用场合中。

一般情况下BVH的性能都比较不错, 起码比基于格子(Grid)的场景管理技术要高。

BVH的数据结构其实就是一棵二叉树(Binary Tree)。它有两种节点(Node)类型: Interior Node 和 Leaf Node。前者也是非叶子节点, 即如果一个Node不是Leaf Node, 它必定是Interior Node。Leaf Node 是最终存放物体/们的地方, 而Interior Node存放着代表该划分(Partition)的包围盒信息, 下面还有两个子树有待遍历。

使用BVH需要考虑两个阶段的工作: 构建(Build)和遍历(Traversal)。

1. Build

构建工作考虑的是如何构造一棵可以有效描述当前场景信息的二叉树。这当中的关键是如何对(假定)毫无规律地散落在场景中的众物体进行划分(Partition), 即决定哪些物体该划分到左子树上, 哪些物体该划分到右子树上。我们可以把这个问题抽象成一个“划分策略(Partition Strategy)”——我们总会按照某种“策略”划分场景的, 待会再考虑具体有哪些策略。另外, 由于我们是在3D空间中工作, 为了将问题简化, 用分而治之的角度看, 我们可以首先建立一个“原则”(Principle): 即决定在哪根轴(x,y,z)上进行划分。“原则”(Principle)与“策略”(Strategy)的不同之处在于, 不管用何种“策略”, 总是遵守同一种“原则”。

1.1 Principle

决定在哪根轴(x,y,z)上进行划分, 取决于场景中的物体在各个轴上分布的“散度”。如果这些物体沿着某根轴分布得最为“松散”(即落在该轴上靠一侧最近的物体与另一侧最近的物体, 二者距离为最大), 那么就沿该轴进行划分。

1.2 Strategy

确定了以哪根轴进行划分, 接下来就要考虑“怎么划分”。这时有多种策略可以考虑: *pbrt*中应用到的有三种划分策略: 取中点划分(SPLIT_MIDDLE), 按等量划分(SPLIT_EQUAL_COUNT), 以及用表面积启发式算法划分(SPLIT_SAH, SAH=Surface Area Heuristic)。

1.2.1 SPLIT_MIDDLE

顾名思义, 取中点划分的意思就是在先前选取的轴上找到最靠两侧的物体的包围盒, 连接二者包围盒的重心得到一条线段, 取其中点作为划分点, 中点以左划分到左子树, 中点以右划分到右子树。顺便提一下使用std::partition算法可以快速实现这一目的。

这种划分实现起来最为简单, 但往往效果不是太好: 因为物体的分布往往不是均匀的。其中一种糟糕的情况(a)是, 某侧子树上可能会拥挤过多的物体, 而另一侧子树上却太少。这对查找效率影响很大: 想象一种极端情况, 对于有n个物体的场景, 通过这样的划分可能会有n-1个物体落在了左子树, 而1个物体落在了右子树。这种情况下BVH的管理技术基本等于形同虚设, 解决不了实际问题。

另外还有一种糟糕的情况(b), 就是包围盒之间互相“重叠”(overlapped)的情况。如果两棵子树对应的包围盒“重叠”的越多, 那么一条射线穿过该区域时同时击中两子树的概率也就越大, 这就意味着两棵子树都得进行相交测试。如果所有的子树都不得不进行相交测试, 那便失去了“场景管理”本该具有的作用。

1.2.2 SPLIT_EQUAL_COUNT

按数量平分两棵子树, 即左子树拥有的物体数量与右子树拥有的物体数量相等。使用std::nth_element算法可以快速实现。

此种策略是对第一种“按中点划分”策略的改进, 对于处理(a)情况有很大的改善。但是对于(b)情况, 依然束手无策。如果遇到(b)情况出现, 使用该策略性能也是非常低下的。

1.2.3 SPLIT_SAH

好东西总是留到最后面讲。表面积启发式算法(Surface Area Heuristic)应该是目前应付以上(a)(b)情况最好的算法了, 同时也是性能最高、成本最低的算法。该算法基于的理论是复杂度成本分析和概率论, 这种同时涉足了数学与计算机科学的交叉理论使得它披上了一点高大上的光环。

从复杂度成本分析的角度看, 假设场景中有1,2,...n个物体, 如果不做场景管理的话, 那就需要射线对每个物体都做相交测试。假设对于第i个物体, 其做射线相交测试的时间复杂度为t(i), 那么总体的时间复杂度就为t(1)+t(2)+...t(n) = Σt(i)。这个值当然越小越好。t(i)是个相对值, 代表所用的时间复杂度, 并不是真要求这个时间还是咋地。可以简单设这个值为单位1。*pbrt*中, 有一个简单的假设是对每个物体进行射线相交测试所用的时间复杂度都是相等的, 皆为1。所以Σt(i)可以用n来代替。

小明

阅读排行榜

- 1. 希腊字母表及其读音与意义(184006)
- 2. 常数变易法的解释(43116)
- 3. BVH with SAH (Bounding Volume Hierarchy with Surface Area Heuristic)(17784)
- 4. glibc下的内存管理(17110)
- 5. [翻译]随机分形地形生成(16533)

评论排行榜

- 1. 常数变易法的解释(28)
- 2. [翻译]随机分形地形生成(17)
- 3. 上帝的天空之岛(9)
- 4. 两块钱买来的常数变易法(8)
- 5. A Translation for Quaternion 一篇对四元数的翻译(7)

推荐排行榜

- 1. 常数变易法的解释(26)
- 2. 无限风光： 近来地形算法学习小结(5)
- 3. 希腊字母表及其读音与意义(5)
- 4. BVH with SAH (Bounding Volume Hierarchy with Surface Area Heuristic)(4)
- 5. Normal Map中的值， Tangent Space， 求算 Tangent 与 Binormal 与 TBN Matrix(4)

最新评论

- 1. Re:常数变易法的解释

仿佛是一场跨越时空的学习，看着评论区中10多年前的人们用着当时的流行语可可爱地与我对话，心中不禁感慨万千。

--龙冬蕾蕾

- 2. Re:常数变易法的解释

太牛了！！

--流浪猪头拯救地球

- 3. Re:Normal Map中的值， Tangent Space， 求算 Tangent 与 Binormal 与 TBN Matrix

谢谢清晰的讲解~

--weiwen1990

- 4. Re:glibc下的内存管理
楼主看下这个: Since glibc 2.8 this function frees memory in all arenas and in all chunks with whole free p...

--ysuhao

- 5. Re:[多媒体]算术编码、游程编码

能解释一下，算数编码会产生错误传播，但游程编码不会是因为什么？

--zer0_1s

再假设这样一种划分方式：将场景划分为两个区域A和B，物体散落在AB两区中。射线有可能会击中A区，也可能击中B区。在遍历前我们当然无法断言射线究竟会击中哪个区。但毫无疑问，击中哪个区，就要遍历散落在该区内所有的物体， $\Sigma t(i)$ in A or B。

射线究竟会击中A区还是B区，虽然无法断言，却可以“猜测”。这就用到概率论的理论了。假设击中A区的概率为 $p(A)$ ，击中B区的概率为 $p(B)$ 。那么综合以上的分析，就会得到总体的时间复杂度为

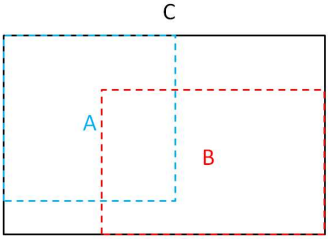
$$\begin{aligned} \text{cost}(A,B) &= p(A) * \Sigma t(i) \text{ in A } + p(B) * \Sigma t(j) \text{ in B } \\ &= p(A) * n \text{ in A } + p(B) * m \text{ in B } \end{aligned}$$

其中n为A区下的物体个数，m为B区下的物体个数

我个人的一点理解，觉得这玩意难道不就是数学中的期望么？（书中倒是没提这茬）。同理，这个值越小越好，也就是说所求得期望值越低越好（整体上复杂度的平均值降到最低，便是最优解。这是个人的一点揣测）。

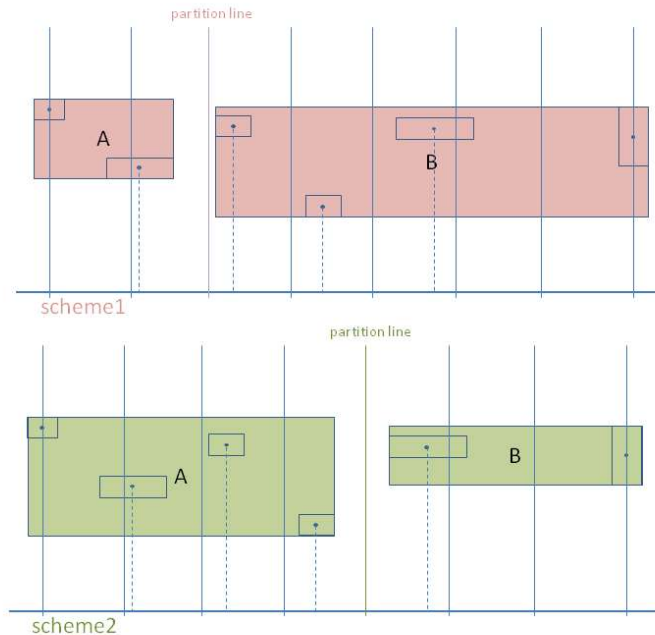
那么， $p(A)$ 和 $p(B)$ 怎么算呢？这时就要用表面积估算了。这根植于一个朴素的想法：如果某物的表面积越大，那么它被射线击中的可能性也就越大。举一个简单的例子，假设空间中有一个6面立方体，一条射线从中穿过。问击中其中一个面的概率是多少（1/6）？击中其他五个面呢（5/6）？这种情况下实际上可以简单把表面积比率（ratio）看作是被射线击中的概率。面积比越大，被击中的概率自然越大。我见到过图形学中很多算法，都有类似的做法。

具体到这个算法中，表面积是通过节点的包围盒的表面积（即长方体的表面积）求算的。仍然考虑一个父节点（C）下带有左（A）右（B）子节点（子树）这种情况。父节点C的表面积为 $S(C)$ ，左子节点A的表面积为 $S(A)$ ，右子节点B的表面积为 $S(B)$ 。那么击中父节点下的左子节点A的概率为 $p(A|C) = S(A)/S(C)$ ，击中父节点下的右子节点B的概率为 $p(B|C) = S(B)/S(C)$ 。



注意 $S(A)/S(C)+S(B)/S(C)$ 是有可能大于 $S(C)$ 的，因为 $S(A)$ 与 $S(B)$ 可能会发生“重叠”，这当然是我们不喜欢的情况，原因上面已经讲过。所以 $S(A)/S(C)+S(B)/S(C)$ 越小（越接近 $S(C)$ ）我们越喜欢。

在划分一个节点代表的空间区域时，可以通过不同的切法将该空间划分成两个子区域。切法可以平均等距地一刀一刀地切，也可以耍点小手段带点“智能”使其“自适应”。总之每次划分，都会得到两个子区域A和B。那么相应也会算出一个 $\text{cost}(A,B)$ 来。比较所有划分方案下所得的 $\text{cost}(A,B)$ ，取值最小的方案就是成本最低的划分方案，也作为划分该节点的最终/优方案。如下图所示，scheme1和scheme2就是两种不同的划分方案。

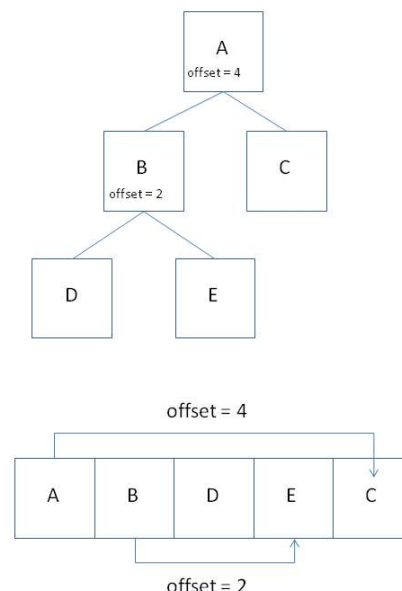


反思sah这种算法，想想为什么它能够有效应付糟糕情况（a）和（b）？我的理解是它综合考虑了“分布”（体现在因子 $\Sigma t(i)=n$ ）和“重叠”（体现在因子 $S(A)/S(C)=p(A|C)$ ）两种情况，二者的乘积最小，代表检测它所花的成本最小（我们当然喜欢某个区域内需待检测的物体越少越好，且该区域与其他区域发生的“重叠”也越小越好了）。但这只是我的胡思乱想，没有数学证明可以支持这一分析。

sah的做法并不能完全做到“不重叠”或者使划分后的分布就很“均匀”，但它每做一次划分，选取的都是当前情形下最优的方案。因此称它是一种“启发式”（Heuristic）算法。

1.3 Compact BVH

构建好一棵BVH后可以进一步优化。树的结构是通过指针寻址下一个子节点的，与连续空间的数组存放数据的方式相比，无论内存的整齐程度还是遍历的速度都逊色不少。那能不能把这棵BVH转换成一个数组呢？答案是肯定的，而且也不难。按深度优先顺序把遍历的节点包装一下（加点offset信息）依次放入数组中，就把这棵BVH“压平”到一个数组中。这样遍历起这个数组那速度可就快多了。

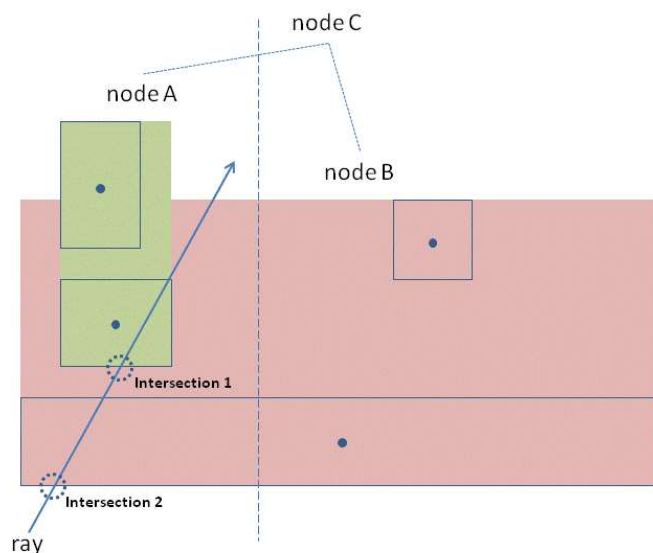


2. Traversal

遍历BVH差不多是件直截了当的事情。只不过这里注意下如果做了Compact BVH优化的话，其实是对一个数组进行遍历，这时要通过算好offset的值来找到对应的节点（或是叶子）。

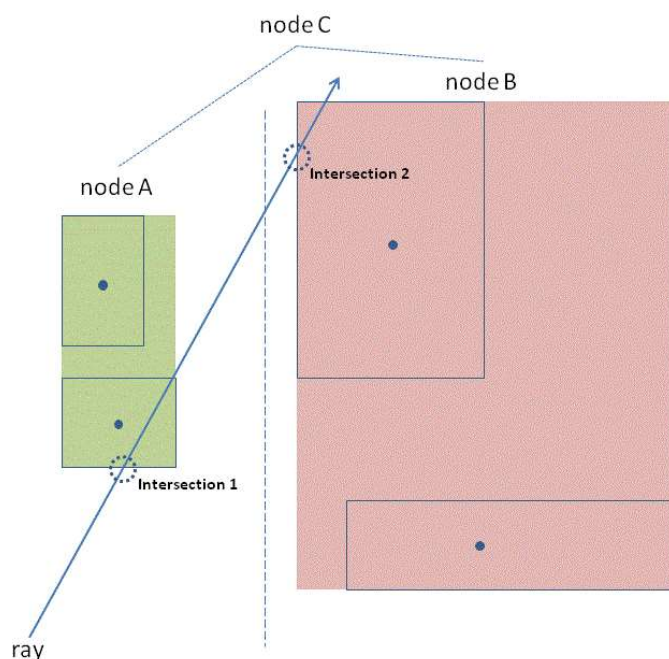
*pbrt*介绍了一种更加快速判断射线与包围盒相交的算法，但我没有细看。这里标记一下，留作以后研究。

另外要考虑的一个问题是，当发现射线与某个子节点相交的话，那么有无必要再检测下与另一子节点是否相交？答案是要的。因为两个节点无法保证完全“不重叠”，如下图所示，很有可能在检测另一子节点时发现了更近（closer）的交点。



*Intersection 1 is prior to intersection 2 to be visited
But Intersection 2 is closer than intersection 1*

还有一个问题是，当需要判断射线是否与子节点相交时，应该先检测左子节点呢还是右子节点？答案取决于射线有可能会先与谁相交。如果射线通过的方向是从左到右，那就应该先检测左子节点，反之就应该先检测右子节点。因为上面讲过两棵子节点都要检测（因为可能“重叠”），通过这种方法可以提高检测效率。因为如果不重叠的话，当判断到另外一棵子节点时就会立即返回了（不重叠的话就不可能有比当前相交点更近的值）。如下图所示。



*Intersection 1 is prior to intersection 2 to be visited
And Intersection 1 is closer than intersection 2
Because node A and node B are not overlapped*

(完)



lookof

粉丝 - 87 关注 - 3

+加关注

40

[升级成为会员](#)

« 上一篇: DDA, Bresenham line's algorithm and Voxel Traversal used in the Grid-Accelerator in PBRT
posted on 2014-02-12 23:06 lookof 阅读(17785) 评论(1) 编辑 收藏 举报

[刷新页面](#) [返回顶部](#)

登录后才能查看或发表评论, 立即 [登录](#) 或者 [逛逛](#) 博客园首页

- 【推荐】100%开源！大型工业跨平台软件C++源码提供，建模，组态！
- 【推荐】博客园社区专享云产品让利特惠，阿里云新客6.5折上折
- 【推荐】轻量又高性能的 SSH 工具 IShell：AI 加持，快人一步

C-阿里云

阿里云新用户钜惠

博客园社区专享 6.5 折优惠

立即领取

120+ 款云产品专属折扣 立享新用户 折上折

- 编辑推荐:
- C# 13(.Net 9) 中的新特性 - 半自动属性

· 聊聊向量数据库

· golang slice相关常见的性能优化手段

· 谈一谈 Netty 的内存管理，且看 Netty 如何实现 Java 版的 Jemalloc

· 修复一个kubernetes集群



华为云总经销杭云&博客园



购买云资源(服务器、数据库等)
享受优惠折扣

- 阅读排行:
- .NET 8.0 开源在线考试系统（支持移动端）

· 全中国有多少公网IP地址？

· C#使用Socket实现分布式事件总线，不依赖第三方MQ

· 基于wxpython的跨平台桌面应用系统开发

· Angular 19 "要" 来了 ⚡