

一种高效且忽略缓存的 ABT 树实现算法研究

杨 明, 刘宏义

(解放军边防学院, 陕西 西安 710108)

摘 要: 计算机的性能将会变得越来越依赖于内存访问, 而不是单纯的 CPU 原始性能. 实时仿真需要大量的数据集, 而内存瓶颈会成为它的限制, 尤其是那些不能充分利用多级存储结构的算法, 会有更多的缓存缺失. 但是基于 ABT 树的忽略缓存算法具有很好的适应性, 它为缓存感知算法提供了一个很好的替代方案, 且二者的性能不相上下.

关键词: 忽略缓存算法; ABT 树; 缓存缺失; Van Emde Boas 布局

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2013)08-0135-04

Research on an Efficient and Ignorable Cache Algorithm of ABT Tree

YANG Ming, LIU Hong-yi

(Border Defence Academy of P. L. A., Xi'an 710108, China)

Abstract: The computer's capability depends more memory access than the CPU's original ability. Real-time simulation needs lots of datasets, the shortage of memory will become the main limitation. Especially those algorithm which cannot take advantage of multilevel storage structure will bring much more lose of cache. But the ignorable cache algorithm based on ABT tree has great adaptability, it will offer a wonderful substitute scheme for the algorithm of cache-awareness. What's more, these tow methods have the same effect.

Key words: ignorable cache algorithm; ABT tree; cache loss; Van Emde Boas placement

1 引言

计算机体系结构所描述的多级存储系统正不断变得越来越慢、越来越大. 这个多级存储系统包括寄存器、一级缓存、二级缓存、主存和硬盘. 它们的访问时间从寄存器的一个存储周期, 增长到了缓存、主存和硬盘各自的 10、100 和 100 000 个存储周期. 从当前 CPU 及内存的发展情况来看, 那些不能充分利用这个多级存储结构的算法, 会有更多的缓存缺失, 从而造成更严重的后果. 为了解决这个问题, 文献[1]中提出了忽略缓存的算法. 该算法的核心思想是在不需要知道存储块大小的情况下优化 I/O 模型的方案.

2 内存结构

大多数算法都会忽略内存结构. 如二叉树, 在访

问它们的数据时会遭受严重的性能下降. 每一个内存层都是以相似的方式工作, 并由缓存块组成. 当前的内存结构使用的缓存行的大小大约是 32~64 个字节, 通过访问同一个缓存行中的数据, 可以获得明显的性能提升^[2]. 缓存的管理要选择合适的替换策略和关联策略. 有时, 缓存缺失是不可避免的, 例如强制缺失、容量缺失、冲突缺失^[3].

2.1 数据结构和缓存一致性

空间的排列对缓存的使用影响很大. 特别要注意的是, 密集使用指针的数据结构并不是好的选择. 可能需要一起访问的指针或数据, 最好也保存在一起, 这样可以优化内存的访问^[3]. 在某些情况下, 它们也许无法遵从面向对象编程的方法论.

2.2 Van Emde Boas 布局

Van Emde Boas 布局^[4]是一个内存中编排出平衡树的标准方法(见图 1). 在忽略缓存的模型中, 使

用它可以非常高效地遍历从根节点到叶子节点的每一条路径,其时间复杂度为 $O(4 * \log_B(n))$ 次内存传输,其中 $B = NbElement/CacheLine$.

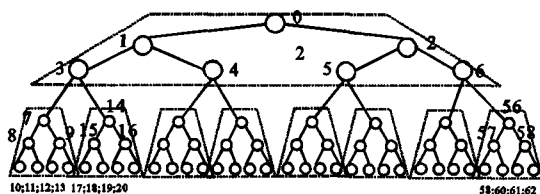


图1 Van Emde Boas 树的表示法

3 实现 ABT 树

ABT 树与 KD 树非常类似^[5]. 在每个构建步骤中,沿坐标轴的分割平面会分割出两个子节点. 与 KD 树有一点不同的是,ABT 树的算法可以将分割产生的子节点的 AABB 盒最小化,并将所有几何体独占式地存储在叶子节点中. 这样,每个节点就变成了空间中一个完全封闭的区域. 在这个空间中,我们会遍历这些内部节点,将场景中的不可见部分剔除.

3.1 ABT 树的创建

创建 ABT 树是从一个根节点开始的,这个根节点包含一个对整个场景 AABB 盒的引用. 递归创建法会用一个轴向上的分割平面,将局部当前节点的 AABB 盒分割成两个部分(2 个面),然后根据它们的中间层,把这 2 个面分别分配给两个子节点(见图 2).

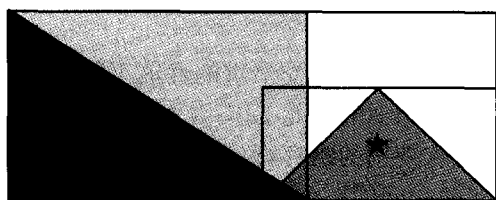


图2 沿坐标轴的包围盒的重新调整

一旦所有面都分配完毕,每个子节点就会重新计算它们各自的、含有这个层中独一无二的面的 AABB 盒. 结束条件取决于特定的 3D 场景和设想的硬件. 分割策略会尽量将下列属性最小化:

(1)空间局部化,参见式(1)

$$f_1(n) = \text{Min}(\text{Area}(\text{boxLeft}) + \text{Area}(\text{boxRight})) \quad (1)$$

(2)树的平衡性,参见式(2)和(3):

$$f_2(n) = \text{Min}[\Delta((\text{boxLeft}) - \text{Area}(\text{boxRight}))] \quad (2)$$

$$f_3(n) = \text{Min}[\Delta(\sum(\text{ExtendedArea}) - \sum(\text{Area}))] \leq \epsilon \quad (3)$$

(3)在性能出现显著降低之前,Epsilon 应该维

持在 5%~10% 以下. 在大多数情况下,将 AABT 场景复杂度扩展 5%,就可以包含 90% 的面,参见式(4)和(5):

$$f_4(n) = \text{Min}[\Delta(\sum \text{faces}(\text{boxLeft}) - \sum \text{faces}(\text{boxRight}))] \quad (4)$$

$$f_5(n) = \text{Min}[\Delta(\int \text{ressources}(\text{boxLeft}) dt - \text{ressources}(\text{boxRight}))] \quad (5)$$

最后的公式就变成了式(6):

$$f(n) = \omega_1 * f_1(n) + \omega_2 * f_2(n) + \omega_3 * f_3(n) + \omega_4 * f_4(n) + \omega_5 * f_5(n) \quad (6)$$

根据游戏引擎的瓶颈和场景的组织(场景图、特效等),公式中各项的权重会有所变化. 在预处理阶段和运行时,采用的方法也是不同的.

在运行时,在不同的渲染阶段(剔除、阴影和碰撞检测等),每个视点通常都会执行一个完全或部分的遍历. 从上一个局部根节点开始,递归函数会检测两个子节点是否位于视锥内部,如果确实位于视锥内部,那就继续执行树的遍历. 当遍历到一个叶子节点时,它包含的所有几何体都可以传送到渲染管道中的下一个阶段. 由于每个面都是独一无二的,所以不需要进行额外的测试(如碰撞检测). 我们还可以为叶子节点和局部素材维护一个很小的顶点缓冲器. 为了减少所使用的顶点缓冲器的数量,可以从邻居子节点的位置中获益. 因此,几个叶子节点可以共享一个单独的顶点缓冲器,只要不超出顶点缓冲器的容量限制即可(通常是 65K, 16 位索引). 这样一来,当几个邻居节点需要一起处理时,就可以获得更高效的分支处理,提高渲染性能^[6].

3.2 复杂度

和所有的二叉树一样,ABT 树也需要一个 $O(\log n)$ 的搜索时间^[7]. 通过使用 Van Boas 忽略缓存的布局表示法,搜索时间可以估算为 $O(4 * \log_B(n))$. 其中, $B = \text{CoupleNode} \div \text{CacheLines}$ ^[8] (见图 3).

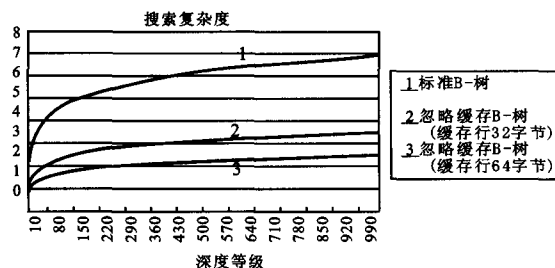


图3 搜索时间排序

每个节点需要存储各自的局部 AABB 盒和一个指向其两个子节点的指针(或索引). AABB 盒由 6 个浮点值表示是 6×4 个字节,指向两个子节点的

指针是 2×4 个字节(32 位 CPU),或者 2×8 个字节(64 位 CPU). 这样算来,每个节点的内存总需求是 32 个字节(32 位 CPU),或者 40 个字节(64 位 CPU). 如果使用一个不成熟的算法,内存消耗相对地就变的更重要了.

文献[9]告诉我们只需保存每个子节点 AABB 盒的相对范围(extent),可以把每个范围压缩成一个 8 位的整数值. 这个谨慎的估算方法只有 $1/255$,也就是约 0.4% 的相对误差,完全可以被平均 5%~10% 的 AABB 重叠率覆盖掉.

3.3 冗余技术

为减少内存开销,在每次分割时,定义子节点 AABB 盒 12 个范围(extent)中的 6 个直接来自父

节点,因为父节点包含的所有面都会被分割、传递给叶子节点. 我们不是单独为每个子节点保存几个字节的信息,而是把两个子节点保存为一个整体. 这样,就不用局部地保存每个子节点 AABB 盒的绝对数据,取代它的是每对子节点都会保存它们相对于父节点 AABB 盒的一个比例. 因此,需要有 6 个字节来表示子节点的相对比例. 通过不同特征指定的另外一个字节会使用两个子节点的相对位置,并再次使用父节点的相关数据. 在运行时的遍历过程中,递归方法会重新计算子节点局部 AABB 盒的位置. 因为在此阶段最后那个字节还有两位没有用到,所以就用这两位来说明左/右子节点是右节点还是叶子节点(见图 4).

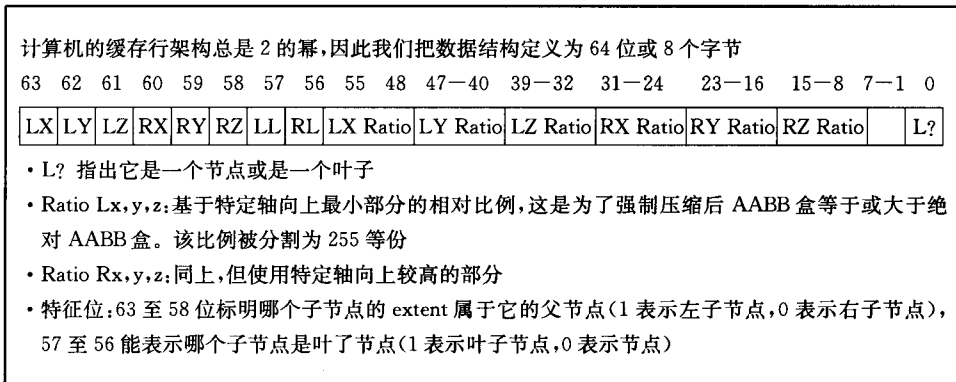


图 4 ABT 树的节点数据表示(一对子节点)

为了遵守忽略缓存算法的数据结构定义,我们把这些数据都保存为 8 个字节的一个值,为树本身留下 7 个未使用的位(bit). 例如,可以使用这 7 个位(bit)来标识是否加载了下列子树(subtree),这对流(streaming)应用非常有价值. 最后,因为子树总是 2 的幂减去 1,且缓存行的大小也总是 2 的幂,所以有 8 个字节可以用来把这棵子树链接到下一棵子树. 由于打算用隐式指针来表示这种分级,所以用 4 个字节来保存第 1 个可用子节点的索引. 另外 4 个字节中,有些位用来说明哪个端节点(end node)连接在子节点的子树上. 根据不同缓存行的大小,有些位可能仍然未被使用(见图 5).

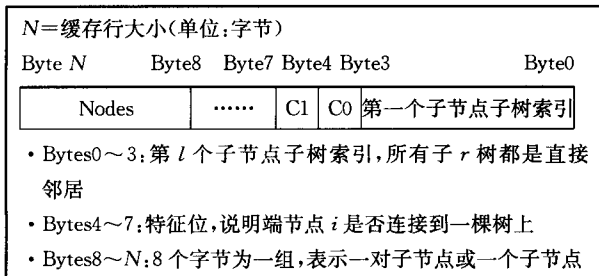
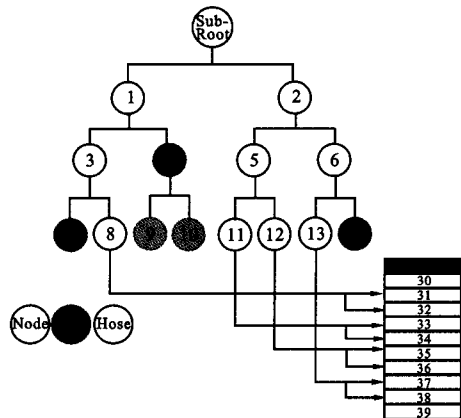


图 5 缓存行的组织结构

当上一棵子树的所有子树都成为直接邻居时,这个创建过程就完成了(见图 6). 叶子节点了,它也是由 8 个字节组成. 一个位用来表示它的条件,在 32 位系统中,31 个位就足够了,留出 4 个字节可以用来保存其他的附加信息(主要用于资源的动态管理),64 位程序可能需要使用更多的节点,因此也需要比 31 位更多的内存空间.



4 结束语

虽然本文专注于讨论剔除算法,但 ABT 树在其他很多领域也是非常有用的,包括 AI 和 3D 音效管理。当前的硬件技术发展,要求我们去研究对不同的内存层访问数据时所需要的系统开销。CPU 的速度会继续大幅提升,但是内存的发展却跟不上这个脚步。计算机的性能将会变得越来越依赖于内存访问,而不是单纯的 CPU 原始性能。实时仿真需要大量的数据集,而内存瓶颈会成为它的限制,CPU 发展所带来好的影响也会因此而弱化。但是,忽略缓存的算法具有很好的适应性,它为缓存感知算法提供了一个很好的替代方案,且二者的性能不相上下。

参考文献:

- [1] Frigo, Leiserson, Ramachandran Prokop. Cache-Oblivious Algorithms[C]//Proc 40th Annual Symposium on Foundations of Computer Science. Washington, DC, USA: IEEE Computer Society, 1999: 285—297.
- [2] Patterson J, Hennessy F. Computer organization and design second edition The Hardware/Software Interface[M]. [S. L.]Morgan Kaufmann, 1997.

- [3] Ding C. Data layout optimizations[R]. Rochester, NY: computer organization Lecture, 2004.
- [4] van Emde Boas. Preserving order in a forest in less than logarithmic time And linear space. [J]. Inf Process, 1977, 6(3): 80—82.
- [5] Szecsi. An Effective Implementation of the K-D Tree Graphics Programming Methods[M]. [S. L.]: Charles River Media, 2003: 315—326.
- [6] Uatthias Wloka Batch, Batch, Batch: What Does It Really Mean? [M]. San Jose, CA: GDC, 2003.
- [7] Sedgewick. Algorithms in C[M]. [S. L.]: Addison Wesley, 1990.
- [8] Brodal. R Cache Oblivious Searching and Sorting[D]. Copenhagen, Denmark: Seminar, ITUniversity of Copenhagen, 2003.
- [9] Gomez, Loura. Compressed axis-aligned bounding box trees[M]. [S. L.]: Charles River Media, 2001.

作者简介:

杨明 男, (1972—), 讲师。研究方向作战仿真与虚拟现实。

刘宏义 男, (1972—), 副教授。研究方向作战仿真与虚拟现实。

(上接第 134 页)

网络试验平台可以成功改变传感器网络的拓扑结构,而且命令响应的延迟时间不大,说明了这个平台是可重构的。在实际的车联网中,车载终端和路侧节点都是大量的,而且车载终端是移动的,如何明确你要发送信息的目的车载终端是一个需要解决的问题,也是作者需要进一步研究的问题。

参考文献:

- [1] 王建强, 吴辰文, 李晓军. 车联网架构与关键技术研究[J]. 微计算机信息, 2011, 27(4): 156—158.
- [2] 刘敏钰, 吴泳, 伍卫国. 无线传感网络(WSN)研究[J]. 微电子学与计算机, 2005, 22(7): 58—61.
- [3] Handziski V, Köpke A, Willig A, et al. Twist: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks[C]// Proceedings of the 2nd International Workshop on Multi-hop Ad Hoc Networks: from Theory to Reality. New York, NY, USA: ACM, 2006: 63—70.

- [4] EBOX—3300MX 产品介绍[EB/OL]. [2013—03—26]. <http://www.compactpc.com.tw/product.aspx?act=detail&id=25>.
- [5] 张保华, 李士宁, 腾文星, 等. 基于无线传感器网络的温室测控系统研究设计[J]. 微电子学与计算机, 2008, 25(5): 154—157.
- [6] 何苏勤, 张彦芬. 基于 RFID 技术及 GSM/GPS 的车台系统[J]. 微电子学与计算机, 2004, 21(10): 74—76.

作者简介:

段宗涛 男, (1977—), 博士后, 副教授。研究方向多媒体智能信息处理、交通信息综合处理与应用、多核/众核并行计算技术及其应用。

王栋 男, (1989—), 硕士研究生, 研究方向为物联网及智能交通。

康军 男, (1975—), 博士, 讲师。研究方向为网络环境下复杂系统控制理论与工程研究。