基于外存八叉树的大模型多分辨率并行构建

张亚萍 熊 华 姜晓红 石教英
(浙江大学 CAD&CG 国家重点实验室,杭州 310058)

摘 要 随着 3 维扫描、计算机辅助设计和科学仿真等技术的发展,包含上千万甚至数十亿几何图元的 3 维网格模型变得十分普遍,如何实现这些模型的交互式绘制成为日益迫切需要解决的难题。外存多分辨率技术作为提高大模型绘制性能最有效的方法之一,成为近几年计算机图形学领域的研究热点。然而大型 3 维网格模型多分辨率表示的构建通常需要很长的预处理时间,这非常不利于系统调试和下游应用。针对基于外存八叉树的大模型多分辨率表示的构建,提出了基于子树的任务分割策略和基于基准测试的动态构建任务管理机制,实现大模型多分辨率表示的并行构建和负载平衡,有效地提高了大模型多分辨率表示的构建速度。

关键词 大型网格模型 多分辨率表示 并行构建
中图法分类号:TP391 **文献标志码**:A **文章编号**:1006-8961(2010)04-650-08

Parallel Construction of Multiresolution Representation for Massive Meshes Based on External Memory Octree

ZHANG Yaping, XIONG Hua, JIANG Xiaohong, SHI Jiaoying
(*State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou* 310058)

Abstract With the advance of 3D scanning, computer-aided design and scientific simulation technologies, massive meshes containing over billions of geometric primitives are becoming commonplace. It is difficult to render these meshes interactively. Out-of-core multi-resolution technique, which is one of the most efficient approaches to improve the rendering performance, has become a research hotspot in the field of computer graphics. However, the construction process for multi-resolution representation of massive meshes is often time-consuming, which is not conducive to system debugging and downstream applications. This paper proposes task scheduling based on sub-tree and dynamic construction task management mechanism, and realizes the parallel construction of multiresolution representation for massive meshes which improves the construction speed effectively.

Keywords massive meshes, multiresolution representation, parallel construction

0 引 言

随着 3 维扫描、计算机辅助设计和科学仿真等技术的发展,包含上千万甚至数十亿几何图元的 3 维网格模型变得十分普遍,如何实现这些模型的交互式绘制成为日益迫切需要解决的难题。多分辨率技术作为提高绘制性能最有效的方法之一,通过构造原始模型的多个可变逼近表示,结合硬件资源的绘制能力和绘制误差选择最优的细节层次进行绘制,在保证绘制速度的前提下,尽可能提高绘制质量。但是对于大型 3 维网格模型,其多分辨率表示

基金项目:国家重点基础研究发展计划(973)项目(2002CB312105);国家自然科学基金重点项目(60533080)
收稿日期:2008-09-10;**改回日期**:2009-01-21

第一作者简介:张亚萍(1979—),女。浙江大学计算机科学与技术专业博士研究生。主要研究领域为计算机图形学,可视化。
E-mail:yapingzhang@cad.zju.edu.cn

的设计、构建和绘制本身也存在诸多难点。受限于内存容量,网格模型的几何数据及其辅助数据结构不能完全装载进内存,无法直接应用传统的多分辨率表示构建和绘制算法;受限于总线带宽和 CPU 处理能力,大型 3 维网格模型的多分辨率表示构建普遍耗时很长。因此,针对大型 3 维网格模型的多分辨率技术成为近几年计算机图形学领域的研究热点。

本文首先介绍了该领域的研究现状和最新进展,并针对基于外存八叉树的大模型多分辨率表示的构建,提出了基于子树的任务分割策略和基于基准测试的动态构建任务管理机制,实现大模型多分辨率表示的并行构建和负载平衡,有效地提高了大模型多分辨率表示的构建速度。

1　研究现状

要实现大型 3 维网格模型的交互式绘制,建立有效的多分辨率层次结构,进行视点相关的细节层次选择是至关重要的。根据层次结构的基本构建单位,模型的多分辨率表示主要可以分为两类:顶点层次结构表示和聚类层次结构表示。

1.1　顶点层次结构表示

最早的适用于视点相关绘制的多分辨率层次结构是由 Xia 等人构建称为合并树的顶点二叉树[1],其树节点表示原网格顶点,并将边折叠操作存储在该树中。绘制时根据光照和屏幕投影边长执行部分边折叠(粗化)和顶点分裂(细化)操作。Hoppe 提出的顶点层次结构为一个森林[2],其中每棵树的根节点为基网格的顶点,与该顶点相关的顶点分裂记录都存储在这棵二叉树中。绘制时,根据视锥范围、表面法向和屏幕空间几何误差等,遍历所有二叉树,选择执行部分边折叠和顶点分裂[3]。

FastMesh[4]和 XFastMesh[5]层次结构中的每个节点不表示原网格顶点,而是表示一个边折叠操作,这可以将多分辨率表示存储量减少一半。El-Sana 等人将边折叠操作推广为点对收缩操作,并构建视点相关的点对收缩二叉树[6]。

1.2　聚类层次结构表示

在绘制连续多分辨率表示时,通常需要跟随视点变化维护一个层次结构的活动节点列表。对于顶点层次结构表示,该列表的更新代价与模型的顶点数量成正比,是影响绘制速度的主要瓶颈。为解决该问题,研究者提出了聚类层次结构表示方法。其主要思想是将细节层次的切换单元从几个图元扩展到一批图元,从而降低更新代价,并且更能充分利用现代图形处理器的缓存机制[3]。

Luebke 等人提出了基于顶点聚类的多分辨率表示[7]。该方法利用八叉树划分场景,并建立顶点树层次结构,每个树节点包含一批顶点。若某个树节点收缩,其中所有顶点将聚类为代表顶点,而退化的三角形将被去掉[3]。

Cignoni 等人提出了基于四面体空间层次结构的多分辨率表示方法[8]。该方法采用四面体划分模型的包围体空间,每个四面体单元包含原始网格的部分简化网格。四面体内的所有三角形为细节层次的基本切换单元,并且被组织成三角形条带,以减少内存到显存的数据传输量。Yoon 等人提出了渐进网格的聚类层次结构 CHPM(clustered hierarchy of progressive meshes)[9]。该方法首先将网格图元聚类为数量均匀的子网格,然后自顶向下划分所有子网格,构建以其为叶节点的二叉树,最后从下到上构建每个树节点的渐进网格。该表示方法提供两个层次的细化,即子网格的细化和渐进网格的细化,在降低更新代价的同时又能较好地保证绘制质量[3]。

Lindstrom 提出了基于顶点聚类的连续多分辨率表示方法[10]。该方法首先采用顶点聚类方法以外存方式简化网格模型,然后在简化模型基础上构建一棵外存八叉树。该外存八叉树提供视点相关的细节层次选择,但是由于没有保留原始网格模型,因此,不能提供原始分辨率的绘制。Shaffer 等人[11]提出了一种可以访问原始网格模型的多分辨率表示方法 MRMM(a multiresolution representation for massive meshes)。该方法也是基于顶点聚类,首先构建一棵精简的内存八叉树将其节点一一映射到外存构建外存八叉树。节点中的几何数据都存储在外存中,而内存八叉树则作为构建和绘制时的索引结构。对于上述两种表示方法,每个三角形归属于使其退化的树节点中。在展开(收缩)每个节点时,将会增加(删除)与其关联的一批三角形,从而降低更新代价[3]。

2　MRMM 构建算法简介

基于顶点聚类的连续多分辨率表示方法作为目前最为高效和最易实现的一种多分辨率表示方

法,在处理如 St. Matthew (372M)这样巨大的网格模型时,单机的外存多分辨率构建时间一般也需要 10 个小时以上,这不仅不利于下游应用,也不利于系统本身的调试。因此,本文着重研究与实现 MRMM 构建算法的并行化,以缩短多分辨率表示的预处理时间。文章接下来将对 MRMM 算法作简要介绍。

MRMM 算法的核心就是外存八叉树的构建,如下分别给出了外内存八叉树节点的数据结构。

外存八叉树节点数据结构

float[3]	$\boldsymbol{v}$	节点的代理顶点
float[3]	$\boldsymbol{n}$	节点法向锥的法向量
float	$\boldsymbol{na}$	节点法向锥的半角
float	e	节点的二次误差
integer	o	节点的内存八叉树索引
integer	f_i	节点包含的三角形起始索引
short	f_n	节点包含的三角形个数
integer	c_i	第一个孩子节点的外存八叉树索引(或节点包含的顶点起始索引)
short	c_n	孩子节点的个数(或节点包含的顶点个数)
byte	l	节点的深度

内存八叉树节点数据结构

integer	o	节点的内存八叉树索引
integer	i	对应的外存八叉树节点线性索引
integer	a	顶点计数器
byte	$\boldsymbol{c}$	孩子向量,每一位代表一个可能的孩子
byte	l	节点的深度

外存八叉树的构建主要由 3 步组成:顶点扫描、三角形扫描和八叉树信息补全,如图 1 所示。

顶点扫描主要是用来确定内存八叉树中的非空节点,生成顶点记录文件。每个顶点对应一条顶点记录,其中除了包含该顶点的几何信息之外,还包括该顶点所属的八叉树叶节点索引及其在该节点中的局部索引。

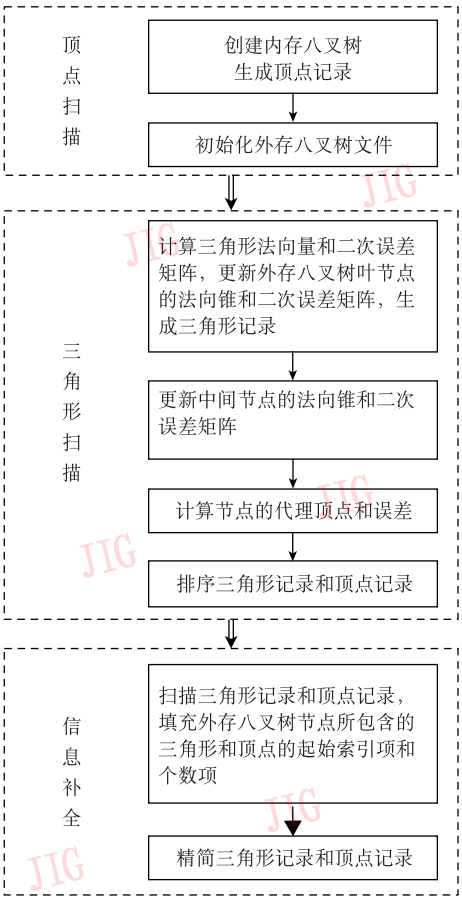

图 1 外存八叉树构建流程图

Fig. 1 External octree construction flow

三角形扫描是其中最为耗时的一步,需要解析三角形的每个顶点,计算面积加权法向量和二次误差矩阵,并将其累积到三角形的各顶点所归属的外存八叉树叶节点中,同时生成三角形记录。每个三角形记录由该三角形的八叉树分裂节点索引、其 3 个顶点所属的内存八叉树叶节点索引及其在该节点中的局部索引组成。若三角形有两个或两个以上的顶点都落入同一个八叉树节点时,称该节点为三角形的分裂节点。如果三个顶点落入的八叉树节点都不相同时,需要向上回溯比较 3 个节点的父亲节点是否相同,迭代执行该过程,直到至少有两个或两个以上顶点的祖先节点是相同的,然后以此祖先节点作为三角形的八叉树分裂节点。完成外存八叉树叶节点的更新后,接着更新中间节点的法向锥和二次误差矩阵,计算节点的代理顶点和二次误差。最后根据三角形的分裂节点索引对三角形记录进行排序,根据顶点所属的八叉树叶节点索引及其在该节点中的局部索引对顶点记录进行排

序,从而避免绘制时对外存的随机访问,提高系统的绘制性能。

构建算法的最后步骤是外存八叉树信息的补全。通过对已排序好的三角形记录文件和顶点记录文件进行扫描,填充外存八叉树节点所包含的三角形和顶点的起始索引项(f_i,c_i)和个数项(f_n,c_n)。最后进行顶点记录文件和三角形记录文件的精简,顶点记录中只保留顶点的几何坐标信息,三角形记录中只保留对应顶点的内存八叉树节点索引及其在该节点中的局部索引。

3　外存八叉树并行构建

表 1 给出了目前能够获得的几个大模型的八叉树构建详细时间消耗,包括顶点扫描时间 T_{vs},三角形扫描时间 T_{fs} 和八叉树信息补全时间 T_{of}。所有数据在 Pentium Ⅳ 2.4GHz,NVIDIA GeForce FX 5700,1G 内存的 PC 机上测得。与 Shaffer 等人的实验结果相比[11],在几乎相同的运行环境下,几个相同模型的构建时间基本上是一致的。

表 1　八叉树单机构建性能

Tab.1　Octree construction performance

数据集	顶点数目	三角形数目	节点数目	T_{vs}/s	T_{fs}/s	T_{of}/s	总时间/s
Dragon	437 645	871 414	648 045	9	14	1	24
Happy Buddha	543 652	1 087 716	759 206	8	17	1	26
Buddha	757 490	1 514 962	996 623	13	25	2	40
Asian Dragon	3 609 600	7 219 045	2 939 050	103	182	9	294
Thaistatue	4 999 996	10 000 000	3 828 602	216	356	13	585
Lucy	14 027 872	28 055 742	4 910 997	356	1 217	230	1 803

从表 1 可以看出,三角形扫描步骤占据了构建时间的大部分,这不仅是因为其中的外存八叉树节点法向锥和二次误差矩阵计算比较复杂,另一个主要的原因是该计算涉及的许多几何数据及辅助数据结构不能完全装载进内存,导致了大量的 I/O 操作。这一问题在其他两个构建步骤中也普遍存在。为了解决以上问题,本文提出了基于子树的任务分割策略和基于基准测试的动态构建任务管理机制,采用 PC 集群实现外存八叉树的并行构建。下面详细介绍文章的并行构建思想和算法,并给出相关实验结果和讨论。

3.1　基于子树的任务分割

正如前面所介绍的,节点法向锥和二次误差矩阵的计算是通过对三角形的扫描,依次由叶节点向上回溯计算得到的,也就是说各个子树节点的法向锥和二次误差矩阵的计算是相互独立的。因此,考虑采用基于子树的任务分割策略,将各个子树分布到不同的 PC 机上进行并行构建。为了保证分割后的每个子树构建任务所涉及的数据都能装载进内存,以每个八叉树节点所包含的顶点个数及其子孙节点的个数作为内存需求的衡量标准。通常情况下三角形个数大约为顶点个数的两倍,因此,根据节点所包含的顶点个数(a)可以大概估计以该节点为根的子树所包含的三角形个数,并通过访问该节点的所有子孙节点中的孩子向量($\boldsymbol{c}$)计算出该子树包含的节点个数,从而估算出该子树大概的内存需求。当节点的内存需求大于一定的阈值时,以该节点为根的子树将被进一步细分。具体的分割过程包括以下几个步骤:

1) 使用 $2^n \times 2^n \times 2^n$ 的均匀网格对模型包围盒进行划分,每个单元格对应八叉树第 $n+1$ 层中的一个叶节点。顺序扫描顶点列表,根据每个顶点的坐标计算出其所属的单元格索引,即八叉树叶节点的索引。对于一个很大的 n,为每个单元格都分配内存空间是不实际的,而且也是没有必要的。这里只需要给非空单元格所对应的叶节点分配内存空间,并采用由下至上的方式,依次构建内存八叉树中间节点和根节点。对应每个顶点生成一条顶点记录。在处理完所有的顶点后,以广度优先顺序扫描内存八叉树,建立对应的外存八叉树文件。

2) 以深度优先顺序扫描内存八叉树,根据八叉树节点所包含的顶点个数和其子孙节点的个数决定是否对以该节点为根的子树做进一步的分割。记录各个子树的根节点索引。

3）根据子树分割结果，扫描顶点记录文件，将同一子树的所有顶点记录存储在单独文件中。扫描三角形列表，若三角形的两个或两个以上的顶点属于同一子树，则该三角形属于该子树，否则该三角形跨越多个子树，标记为边界三角形。将属于同一棵子树的三角形提取到单独文件，并修改其对应的顶点索引，然后与对应的顶点记录文件一起作为子树的网格模型文件。而对于边界三角形，将其重复分配到其所跨越的多棵子树中，并将不在该子树中的三角形顶点记录附加到该子树的顶点记录列表末端。值得注意的是，在进行三角形扫描时并没有产生相应的三角形记录，这主要是因为各个构建服务器能够根据获得的顶点记录列表和三角形列表在各自的内存中创建对应的三角形记录。这不仅减轻了分割过程的计算量，同时也减少了数据的传输量。

3.2 负载平衡的并行构建

3.2.1 子树构建过程

对于划分后得到的每个子树网格模型，可以将其全部加载到内存中来计算各个节点的法向锥、二次误差矩阵和代理顶点。

由于每个顶点记录中已经存储了其所归属的八叉树叶节点索引，因此，在顺序扫描三角形列表时，各个构建服务器可以在本地生成三角形各顶点对应的内存八叉树叶节点和对应的三角形记录，并将三角形的二次误差矩阵和法向累积到各节点中。这里需要注意的是，对于那些附加到顶点记录列表末端的其他子树顶点，构建服务器不生成与其对应的内存八叉树叶节点。而且构建服务器生成的内存八叉树节点的数据结构与原有 MRMM 算法中的略有不同，其数据结构主要包括节点的内存八叉树索引、二次误差矩阵、代理顶点以及节点法向锥的法向量和半角。

当处理完所有的三角形记录后，该子树的叶节点法向锥和二次误差矩阵也更新完毕，这时可以向上回溯生成和计算各个中间节点的法向锥和二次误差矩阵，直到子树的根节点。有了各个节点的二次误差矩阵之后，就可以计算节点的代理顶点。

构建服务器在完成子树节点的法向锥和代理顶点计算后，需要对其本地的三角形记录和顶点记录进行排序，将它们一起作为该子树构建任务的返回数据。边界三角形记录和顶点记录列表末端的其他子树顶点记录不参与排序。

3.2.2 基于基准测试的动态构建任务管理机制

在并行构建过程中，影响构建整体性能的因素主要有两个：1）PC 资源管理，尤其是对于具有不同内存容量和处理器性能的 PC 集群；2）构建任务管理，只有平衡的任务分配才能充分利用整个 PC 集群处理能力加速构建过程。针对这两个问题，提出了基于基准测试的动态构建任务管理机制。

获取 PC 集群中不同机器的处理性能有利于更好地实现并行处理过程中的负载平衡。在并行构建前，使用一个基准网格测试每台 PC 的实际处理性能。整个基准测试过程包括前面所介绍的顶点扫描、三角形扫描和外存八叉树信息补全 3 步。以模型的多分辨率表示构建时间作为 PC 构建性能的衡量指标。对于静态任务分配方案，该性能指标可以直接用于任务分配，即将构建任务划分为 n 批（n 为 PC 个数），并分配到不同 PC 上，使得所有批次的总构建时间相等。但是构建任务的实际运行状况和时间一般难以预测，尤其是对于面向多任务的 PC 集群环境以及网格环境，其节点往往同时在执行多个任务。因此，为了获得更好的并行构建性能，采用动态任务分配方式，以该性能参数来控制每台 PC 的任务输入输出缓存区大小。此外，又使用一台 PC 作为主控 PC，其主要功能包括任务分割、调度、监测 PC 集群状态、任务收集和重组等。动态构建任务管理机制如图 2 所示。

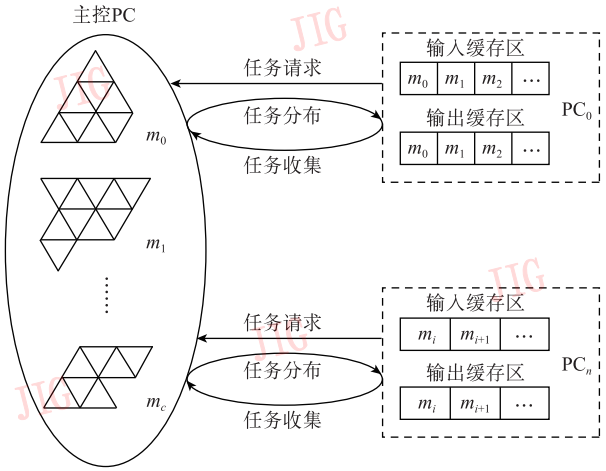

图 2 动态构建任务管理机制

Fig. 2 Dynamic construction task management mechanism

在构建过程中，每个构建服务器动态向主控 PC 申请构建任务，并将构建后的子树节点信息返回给

主控 PC。当输入缓存区的空闲率增大到一定比例时则从主控 PC 申请一批构建任务,以这种方式可以保证输入缓存区中始终保持一定的构建任务,从而避免构建服务器空闲。构建后的子树信息被插入到输出缓存区,当其空闲率减少到一定比例时则向主控 PC 发送子树信息。构建过程和数据传输过程同时交织进行,可以有效隐藏网络的数据传输延时。

3.3　八叉树重组

当主控 PC 收到构建服务器返回的数据时,首先处理该子树的节点信息。根据这些节点的内存八叉树索引找到对应的外存八叉树节点,填充相应的二次误差、法向锥和代理顶点项。当处理完所有返回的节点之后,需要将该子树的二次误差矩阵和法向锥累积到其父亲节点。当该父亲节点的所有子树都返回后,就可以计算该父亲节点的代理顶点。该过程迭代回溯,直到整棵外存八叉树的根节点。

和节点数据一起返回的还有属于该子树的三角形记录和顶点记录。为了提高记录的归并速度,各个构建服务器完成子树构建后在主控 PC 的监控下进行二路归并。随着模型规模的增大,这种二路归并的效率也就更为明显。

在处理完以上所有步骤之后,主控 PC 将顺序扫描排序好的顶点记录文件和三角形记录文件,填充外存八叉树节点包含的顶点和三角形的起始索引项和个数项。最后进行顶点记录文件和三角形记录文件的精简。

4　实验结果

4.1　并行构建性能

为了测试上述基于子树分割的外存八叉树并行构建性能,采用包含 4 个节点的 PC 集群(每台 PC 配置 Pentium Ⅳ 2.4 GHz CPU 和 1 GB 内存,所有 PC 由千兆以太网连接)作为系统的运行环境。表 2 给出了测试模型在任务分割、子树构建及八叉树重组阶段的详细时间消耗。任务分配传输时间包含在任务分割时间中,构建任务返回时间包含在八叉树重组时间中。

与文献[11]相比,他们在配置为 1.8 GHz Athlon XP2500 + 处理器,NVIDIA GeForce FX 5200 显卡和 1G 内存的单机环境下,构建节点个数为4 901 459的 Lucy 模型的外存八叉树时需要 32 min,本文的单机实现需要花费大约 30 min,而本文并行构建在 4 台 PC 上只需要花费大约 14 min 就可以完成该模型的外存八叉树构建。表 3 显示了 Lucy 模型在分割为 8 棵子树后的数据量大小以及在 4 台 PC 上的分布情况。从表中可以看出各个子树的数据量相差比较大,因此在进行任务分配时,可将构建服务器的输入缓存区的大小设为 1,以实现任务的负载平衡。随着任务规模的增加,动态任务管理机制将发挥更大的优势,文中 4.2 小节对此进行了验证。

表 2　基于子树分割的并行构建结果

Tab. 2　Parallel construction performance based on subtree partition

数据集	Buddha	Dragon	Thai Statue	Lucy
顶点数目	543 652	3 609 600	4 999 996	14 027 872
三角形数目	1 087 716	7 219 045	10 000 000	28 055 742
节点数目	759 206	2 939 050	3 828 602	4 910 997
子树数目	8	8	8	8
任务分割时间/s	10	112	228	434
子树构建时间/s	3	15	22	37
八叉树重组时间/s	4	27	41	395
总时间/s	17	154	291	866
单机构建时间	26	294	585	1 803
加速比	1.53 ∶1	1.91 ∶1	2.01 ∶1	2.08 ∶1

表 3 Lucy 模型的子树分割与分布

Tab. 3 Subtree partition and distribution of lucy model

子 树	顶点数目	三角形数目	节点数目	内存需求/M	任务分布
SubTree1	2 078 922	4 148 063	713 848	246	PC_1
SubTree2	2 323 000	4 635 941	825 163	277	PC_2
SubTree3	1 326 258	2 642 565	457 227	158	PC_3
SubTree4	1 707 773	3 404 505	598 218	204	PC_4
SubTree5	1 361 382	2 714 313	479 625	163	PC_1
SubTree6	1 291 739	2 575 435	422 231	152	PC_3
SubTree7	2 227 512	4 442 495	787 996	266	PC_2
SubTree8	1 752 741	3 492 513	626 688	210	PC_4

表 2 中任务分割的大部分时间是用来构建内存八叉树,生成顶点记录文件和初始化外存八叉树。由于这些数据是主控 PC 用于子树分割的全局数据,也是构建过程最终的数据结构,所以并没对其进行并行处理。而对于八叉树的重组,除了需要对构建服务器返回的三角形记录文件和顶点记录文件进行归并之外,还需要顺序扫描这两个文件以完成外存八叉树节点的信息填充,最后是对这两个文件进行精简。根据外存算法的理论计算模型,上述的二路归并排序过程的 I/O 复杂度为 $O(N/B\log_2 N/B)$,而顺序扫描所有数据单元的 I/O 复杂度为 $O(N/B)$,N 为所求解问题的数据单元的总体个数,B 为单个磁盘读取单元可以容纳的数据单元的个数。因此,随着模型规模的增大,任务分割和重组所需要的时间会明显增加。

4.2 任务管理性能

为了测试动态任务管理的性能,采用不同的内存需求标准来划分 Lucy 模型,以便产生足够多的子树构建任务,并与静态任务分配方案的测试结果进行比较,测试结果如图 3 所示。从中可见,随着构建任务的增加,动态与静态任务分配构建时间的差别也越来越大。这是由于静态任务分配方案在任务规模较大的情况下无法对任务的执行时间进行准确的估计,导致负载的不平衡;而动态任务分配方案可以根据实际运行情况调整负载分配,获得较好的负载平衡效果。此外,该测试结果也显示对于某个给定的网格模型,子树的划分并不是越细越好,而是应该根据对内存的需求进行适度的划分。例如 Lucy 模型,在划分为 8 棵子树时外存八叉树的总构建时间是最少的,而随着细分程度的加深,额外的通讯开销和重组开销反而导致更长的构建时间。

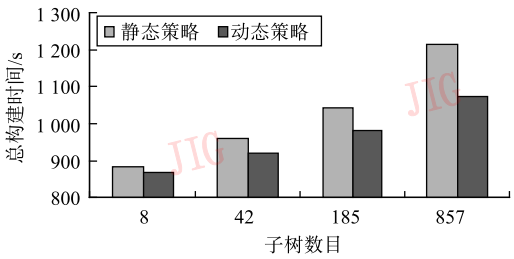

图 3 任务管理性能比较

Fig. 3 Comparision of task management performance

5 结 论

随着 3 维扫描和建模技术的不断提高,网格模型的图元规模也随之不断增长。受限于内存容量、总线带宽和 CPU 处理能力,交互式绘制这些大型 3 维网格模型具有很大挑战。外存多分辨率技术作为提高大模型绘制性能最有效的方法之一,成为近几年计算机图形学领域的研究热点。本文对该领域目前的研究进展进行了简要介绍,并针对基于外存八叉树的大模型多分辨率表示的构建,提出了基于子树的任务分割策略和基于基准测试的动态构建任务管理机制,实现大模型多分辨率表示的并行构建和负载平衡,有效地提高了大模型多分辨率表示的构建速度。

外存多分辨率构建是实现大型 3 维网格模型交互式绘制的基础。然而,传统的以顶点或者三角形作为基本单位的构建算法却不适用于这些大模型。因为这不仅会导致建立起来的多分辨率结构规模过于庞大,更重要的是这样一来更加剧了交互式绘制

时的数据调度负担。所以,加大基本构建单元的粒度势必成为大型 3 维网格模型多分辨率构建技术未来的重要发展方向之一。另外,考虑到近年来 GPU 处理能力的突飞猛进,充分利用 GPU 流水线中的并行处理单元,将更多的计算交给 GPU 来完成,尽量平衡 CPU 和 GPU 之间的负载平衡也将成为大型 3 维网格模型多分辨率构建的发展趋势。

参考文献(References)

[1] Xia J C, Varshney A. Dynamic view-dependent simplification for polygonal models[C]//Proceedings of the 7th conference on Visualization. New York, NY, USA: ACM Press, 1996: 327-334.

[2] Hoppe H. View dependent refinement of progressive meshes[C]//Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co, 1997: 189-198.

[3] Zhang Yaping, Xiong Hua, Jiang Xiao-hong, et al. Out-of-Core constructing and interactive rendering of multiresolution representations for massive meshes[J]. Journal of Computer-Aided Design & Computer Graphics, 2008, 20(9): 1126-1131. [张亚萍, 熊华, 姜晓红, 等. 大型网格模型多分辨率的外存构建与交互绘制[J]. 计算机辅助设计与图形学学报, 2008, 20(9): 1126-1131.]

[4] Pajarola R. Fastmesh: Efficient view-dependent meshing[C]// Proceedings of the 9th Pacific Conference on Computer Graphics and Applications. Los Alamitos, CA, USA: IEEE Computer Society Press, 2001: 22-30.

[5] DeCoro C, Pajarola R. Xfastmesh: Fast view-dependent meshing from external memory[C]//Proceedings of the conference on Visualization. Washington DC, USA: IEEE Computer Society, 2002: 363-370.

[6] El-Sana J, Varshney A. Generalized view-dependent simplification[J]. Computer Graphics Forum, 1999, 18(3): 83-94.

[7] Luebke D, Erikson C. View-Dependent simplification of arbitrary polygonal environments[C]// Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques. New York, NY, USA: ACM pess/Addison-Wesley publishing co, 1997: 199-208.

[8] Cignoni P, Ganovelli F, Gobbetti E, et al. Adaptive tetrapuzzles: Efficient out-of-core construction and visualization of gigantic multiresolution polygonal models[J]. ACM Transactions on Graphics, 2004, 23(3): 796-803.

[9] Yoon S E, Salomon B, Gayle R, et al. Quick-vdr: Interactive view-dependent rendering of massive models[C]// Proceedings of the IEEE Visualization. Washington DC, USA: IEEE Computer Society, 2004: 131-138.

[10] Lindstrom P. Out-of-core construction and visualization of multiresolution surfaces[C]//Proceedings of IEEE Symposium on Interactive 3D Graphics. New York, NY, USA: ACM Press, 2003: 93-102.

[11] Shaffer E, Garland M. A multiresolution representation for massive meshes[J]. IEEE Transactions on Visualization and Computer Graphics, 2005, 11(2): 139-148.