ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.12, December 2008, pp.3237−3248
DOI: 10.3724/SP.J.1001.2008.03237

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

海量医学数据处理框架及快速体绘制算法*

薛 健[1], 田 捷[1,2+], 戴亚康[1,2], 陈 健[1,2]

[1](中国科学院 研究生院 计算与通信工程学院,北京 100049)

[2](中国科学院 自动化研究所 复杂系统与智能科学重点实验室 医学图像处理研究组,北京 100190)

Processing Framework and the Fast Volume Rendering Algorithms for Out- of-Core Medical Data

XUE Jian[1], TIAN Jie[1,2+], DAI Ya-Kang[1,2], CHEN Jian[1,2]

[1](College of Computing & Communication Engineering, Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

[2](Medical Image Processing Group, Key Laboratory of Complex Systems and Intelligence Science, Institute of Automation, The Chinese Academy of Sciences, Beijing 100190, China)

+ Corresponding author: E-mail: tian@ieee.org

Xue J, Tian J, Dai YK, Chen J. Processing framework and the fast volume rendering algorithms for out-of-core medical data. *Journal of Software*, 2008,19(12):3237−3248. http://www.jos.org.cn/1000-9825/19/3237.htm

Abstract: This paper designs and implements an algorithm framework for the out-of-core medical data processing and analyzing and integrates it into MITK (medical imaging toolkit), an algorithm toolkit for medical image processing and analyzing accomplished by the group. With the help of this, a processing platform for the out-of-core medical data is set up and fast out-of-core volume rendering algorithms based on volume ray casting and 3D texture are studied in this paper. A semi-adaptive partitioning method is proposed to divide original data sets into sub-blocks and get a better partitioning result without influencing the partitioning speed. Furthermore, the graphics hardware is also used to accelerate the rendering process. The experimental results indicate that the new framework and algorithms are effective and efficient for the processing and visualization of the out-of-core medical data sets.

Key words: out-of-core data processing; medical image visualization; volume rendering; algorithm toolkit

摘 要: 设计并实现了一套针对海量数据的处理和分析算法框架,并将其融入实验室早先开发完成的医学影像算法研发平台 MITK(medical imaging toolkit)中,真正建立起一个海量医学影像数据的处理平台,并在此基础上研究了针对海量数据的基于光线投射和三维纹理的快速体绘制算法,提出了一种半自适应分块的方法对原始数据进行分

* Supported by the National Natural Science Foundation of China under Grant Nos.60532050, 30600151, 30500131, 30672690 (国家自然科学基金); the National Basic Research Program of China under Grant No.2006CB705700 (国家重点基础研究发展计划(973)); the National High-Tech Research and Development Plan of China under Grant No.2006AA04Z216 (国家高技术研究发展计划(863)); the Chinese Academy of Sciences Hundred Talents Program (中国科学院百人计划); the Chinese Academy of Sciences Scientific Research Equipment Develop Program under Grant Nos.YZ0642, YZ200766 (中国科学院科研装备研制项目); the Beijing Natural Science Fund of China under Grant Nos.4051002, 4071003 (北京市自然科学基金)

Received 2007-05-16; Accepted 2007-11-20

块,在不对分块速度产生太大影响的基础上得到了更好的分块结果,同时使用图形硬件来进一步加速整个算法的绘制流程.实验结果表明了该平台和算法对于海量医学数据处理和可视化的有效性.

关键词: 海量数据处理;医学影像可视化;体绘制;算法研发平台

中图法分类号: TP391 文献标识码: A

1 背景介绍

在医学影像处理与分析领域,大多数传统算法是在计算机内存足够的假设下设计和实现的,即需要预先将数据全部导入内存再进行处理.对于需要对三维数据进行全局处理的可视化算法来说,当实际内存不够大时,就存在无法运行或者运行效率过低的问题;而对于图像处理类算法(如分割、配准等)而言似乎并没有什么不妥,单张图像的大小毕竟还是非常有限的.而实际上,这类算法也在向着直接处理三维数据的方向发展,这就使得内存容量的限制成为传统算法在实际应用中的一个隐患.

随着近几年来医学影像设备的不断更新换代,采集图像的分辨率越来越高,相应的数据量也越来越大,其中尤为突出的例子是自从美国的可视人体(visible human)项目实施以来,虚拟人体采集的图像数据量已经达到43GB,而国内研究机构于 2002 年、2003 年完成的中国男性和女性数字化可视人体的数据量已分别达到90.65GB 和 131.04GB.国际上一般将这样大规模以至于无法完全加载到内存的数据称为 Out-of-Core 数据,将处理这类数据的算法称为 Out-of-Core 算法,本文也称其为“海量”数据和“海量数据处理”算法.计算机硬件的发展还远没有赶上数据量的飞速增长,上面所提到的隐患也越来越凸显出来,而目前主流的算法开发包(如VTK,ITK)并没有建立相应的算法框架,在底层提供对海量数据的支持,这就使得在可视化领域,一些经典算法(如基于光线投射的体绘制算法)由于需要对原始体数据作随机访问而无法直接应用于海量数据,即使是普通的图像处理算法,也不得不关注底层数据存储的细节,或多或少地作一些改动才能适应海量数据处理的需求,而这些工作无疑分散了研究人员的精力,对算法研究是不利的,同时也使得这些算法开发包在面对海量数据时失去了作为算法研发支撑平台所应起到的作用.

针对这一情况,本文设计并实现了一套面向对象的针对海量数据的处理和分析算法框架,并将其融入实验室开发完成的医学影像算法研发平台 MITK 中,真正建立起一个海量医学影像数据的处理平台,并在此基础上研究了针对海量数据的基于光线投射和三维纹理的快速体绘制算法.

2 相关工作

目前在医学影像处理与分析领域,对于算法研发平台的研究正越来越受到重视.在国际上,很多科研机构正致力于算法平台的研发和完善,比如久副盛名的可视化算法开发包 VTK(visualization toolkit)[1],近年完成并还在不断发展的分割和配准算法开发包 ITK[2]等.但是这些主流研发平台并没有在底层算法框架和数据结构上提供对海量数据的支持.以 VTK 为例,虽然它所提供的某些算法已经具备处理海量数据的能力(由具体算法的特点所决定),但这种支持只是停留在个别具体算法上,并不具有普遍性和通用性.而 ITK 由于还是专门针对图像处理类算法的开发包,在框架结构上并没有特别提供对海量数据的支持,虽然通过对其相关模板类进行一定程度的扩展可以实现海量数据支持,但是这种框架结构上的扩展对普通算法研究者或使用者来说存在很大困难.

在本文工作之前,本研究组也设计和实现了一套专门面向医学影像领域的算法开发包 MITK(medical imaging toolkit)[3],其设计初衷就是为了给医学影像领域的研究者提供一套风格统一、接口一致、可复用,包括可视化、分割、配准功能的集成化的医学影像开发包,弥补 VTK 和 ITK 的一些缺憾,并引入一些新的特性,使得 MITK 能够成为除了 VTK+ITK 以外的另外一个选择.但是,MITK 最初的版本也没有在框架设计上考虑对海量数据处理的支持,而本文的一项重要工作就是设计和实现了一套针对海量数据的处理和分析算法框架,并将其融入 MITK,使得 MITK 真正从底层框架结构上支持对海量数据的处理.

就算法层面来说,一般图像处理类算法(如去噪、分割、配准等)支持海量数据处理并不存在实质性的难度,

因为这类算法在处理图像时只需要把当前所处理的图像装入内存(最多再加上整个图像序列前后几幅),而不需要将整个图像序列都装入内存.而针对医学影像的可视化算法则不然,很多传统可视化算法特别是体绘制算法需要对原始数据进行随机的存取访问,这就需要将整个数据集事先装入内存再进行处理,所以海量医学影像处理的难点在可视化算法上.

在体绘制算法中,经典的光线投射方法(volume ray casting method)以其能够产生高质量的绘制结果而闻名,自从最初的光线投射体绘制算法[4,5]被提出以来,出现了许多改进的算法,但其中的多数不适合处理海量数据,因为对原始数据的随机存取必将产生频繁的内外存数据交换,从而导致算法效率的低下.而图形硬件的快速发展给快速体绘制算法的出现带来了新的可能性[6].1994 年,Carbral 在文献[7]中首先提出利用图形硬件提供的三维纹理映射功能来加速体绘制.此后也出现了许多基于该方法的改进算法[8−10],但是与前面的原因类似,这些算法更受到显卡纹理缓存大小的限制,也无法直接应用于海量数据.

针对海量数据的科学可视化算法,文献[11]给出了一个比较详尽的综述.其中,Farias 和 Silva 在文献[12]中提出了多种用于大规模无结构网格的直接体绘制技术,可以在有限内存的机器上运用,其中包括将 ZSWEEP 算法[13]扩展到 Out-of-Core 领域.Chiang 等人[14]则提出了适用于分布式存储并行机的并行 Out-of-Core 体绘制算法.Nuber 等人[15]也提出了一种基于点的 Out-of-Core 体绘制算法.针对传统的光线投射算法,Novins 等人[16]提出了一种改进方法,将整个切片序列分为一个个 Slab,每个 Slab 由相邻两张切片组成,按光线投射方向依次导入所有 Slab,在当前 Slab 中推进在该 Slab 范围内的光线,最终得到光线投射的绘制结果,这样,每一时刻就只需在内存中保留一个 Slab(两张切片)即可,从而能够在内存容量比较小的情况下处理很大的数据集.此外,还有一些研究人员采用对原始体数据进行分块的方式来处理较大的数据集.例如,文献[17]针对显卡纹理内存容量的限制提出了一种自适应分块的方法,使得分块后的每块数据能够单独装入纹理内存进行基于三维纹理的加速绘制,同时由正交 BSP 树(orthogonal BSP tree)来保证所有子纹理块按从后往前的顺序绘制,这种方法可以在纹理内存有限的情况下绘制比较大的数据集,并且由于采用自适应的分块方法,能够最大限度地将背景与非背景部分分离,而背景块往往不需要绘制,从而能够大大加速整个绘制过程,但由于其仍然需要在内存中对整个数据集进行分块操作,故尚不能直接用于海量数据的体绘制.文献[18]提出了一种针对数字人切片数据的硬件加速体绘制方法,同样是用三维纹理来加速绘制,不过该方法使用等尺寸分块的策略对原始数据进行分块,因而不可能像文献[17]那样将背景与非背景部分尽可能地分离.同时,在等尺寸分块的策略下,如何选择合适的分块尺寸也是一个难以解决的问题,如果尺寸选得太大,将保留过多的背景区域,而如果尺寸选得太小,则会产生大量的小数据块,影响内、外存数据交换的效率.另外,文献[19]也使用等尺寸分块的方式进行处理,但其在绘制时采用了一些合并策略将小块合并成大块再装入纹理缓存进行绘制,这在某种程度上提高了绘制效率,不过在分块上同样存在文献[18]中的问题,在处理海量数据时效率不高.

3 海量医学影像数据处理算法框架的设计和实现

3.1 计算框架的设计

与 MITK 以前的版本一致,计算框架的设计仍然采用了一个简单的数据流模型[3].在数据流模型中,数据处理是核心,数据和算法分立于不同的抽象模块中,如图 1 所示.数据和算法都由对象来表示,数据被抽象为 Data 类,处理数据的算法被抽象为 Filter 类,每一个 Filter 接受一组输入数据,经过处理产生一组输出数据,这样的一组 Filter 连接成一个流水线,构成统一的计算框架.为了能够以统一的方式处理海量数据,每个 Data 对象都可以带一个磁盘缓冲,而 Data 类则封装了内外存数据交换的操作细节,对外提供了统一的访问接口,从而使上层的算法设计者无须考虑算法所处理的数据是否为海量数据,保持了框架的简洁和一致.

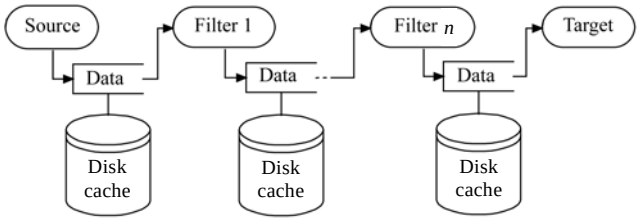


Fig.1 Computational framework for out-of-core data processing

图 1 海量数据处理计算框架

3.2 海量数据的存取

为了达到对医学影像体数据的透明访问,我们为 in-core(内存可容纳)数据和 out-of-core(海量)数据设计了一套统一的访问接口.对 in-core 数据,所有处理均在内存中实现,因此访问接口的实现是简单而直观的.而对 out-of-core 数据,则需要谨慎处理内、外存之间的数据交换,既要对操作细节进行很好的封装,又要达到快速访问的目的.

为了使整个框架结构便于维护和扩展,我们并没有独立地在每个抽象数据类型中分别实现对海量数据的访问支持,而是采用了 Bridge 设计模式,将其中具有共性的负责内存缓冲区维护和内、外存数据交换的部分抽取出来成为一个独立的模板类 OoCStorage,以其存储的数据元素类型为模板参数,这样可以适应对各种类型数据的存储.而涉及海量数据的类则委托 OoCStorage 来实现对数据的存储管理功能.这样不仅降低了耦合,还有利于扩展新的存储管理模块.图 2 所示即为海量三角面片数据类 OoCTriangleMesh 实例.

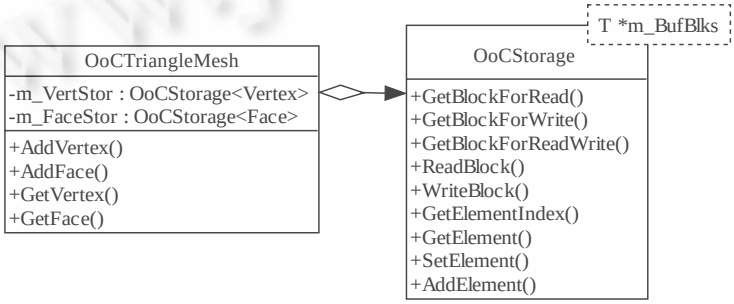


Fig.2 Storage management of the out-of-core data

图 2 海量数据的存储管理

在 OoCStorage 的实现中,内存开辟了一块缓冲区,如图 3 所示,以块(block)为单位存储当前正在处理的数据,同时维护了 4 个列表来管理整个缓冲区,它们分别是:

- Block list:一个线性列表,存储分块后原始数据的每一子数据块所用内存缓冲区的首地址,用于对子数据块的快速访问(可能为空,表示该子数据块尚未导入内存缓冲区);
- Empty buffer block list:一个环形队列,保存当前空闲缓冲区块的指针,这些空闲块可用于缓冲新的子数据块;
- Occupied buffer block list:一个优先队列,维护当前被占用的缓冲区块,同时决定当缓冲区满的时候,哪一个缓冲区块的数据最优先被兑换回磁盘;
- Buffer block nodes list:一个线性列表,存储每个缓冲区块的相关参数和状态信息.

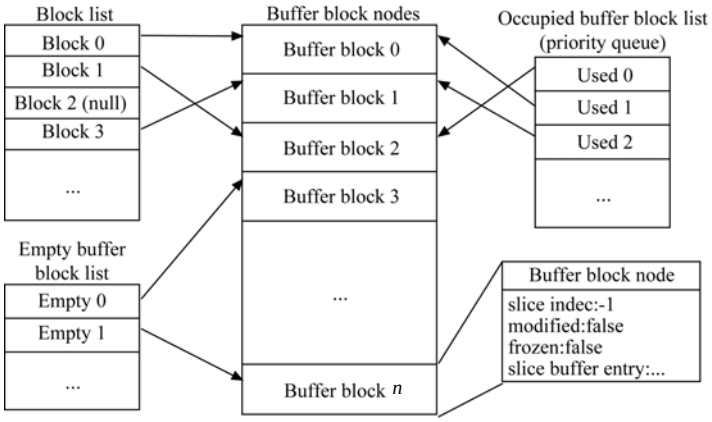


Fig.3 Structure of the OoCStorage memory buffer

图 3 OoCStorage 内存缓冲区结构

OoCStorage 提供了一组接口用于直接存取子数据块(如图 2 所示).在其实现中,获取一个子数据块的基本操作按如下步骤进行:

步骤 1. 检查 Block list,如果该 Block 内存首地址非空,则转步骤 5;

步骤 2. 从 Empty buffer block list 取得一个空闲缓冲区块,若成功则转步骤 4;

步骤 3. 取得 Occupied buffer block list 的头元素,其所指向缓冲区块中的数据若有改动,则写回磁盘,将该缓冲区块归还入 Empty buffer block list,然后转步骤 2;

步骤 4. 将所需 Block 数据从磁盘导入所找到的空闲缓冲区块;

步骤 5. 更新 Occupied buffer block list 和相关缓冲区块的状态信息;

步骤 6. 返回存储 Block 数据的内存首地址或将 Block 数据复制入指定地址的内存区.

为了尽量减少磁盘访问,步骤 3 使用了 LRU(least recently used)规则来确定最先被兑换出去的缓存切片;同时,缓冲数据块只有当需要被兑换出内存并且被修改过时才能真正被写回磁盘(类似于 copy-on-write 策略),从而进一步减少了磁盘访问量.如果要访问某一特定元素,可以使用 OoCStorage 提供的 GetElement()接口,它将元素索引拆为两部分:一部分是子数据块索引,用该索引按上述步骤可以得到该元素所在的子数据块;另一部分是块内索引,用于在得到的子数据块内找到所需元素.

上述实现细节被很好地封装在 OoCStorage 中,具体算法只需要访问顶层数据类型(如 Volume 或 Mesh)提供的统一接口来存取数据,而无须知道它所处理的数据是 in-core 还是 out-of-core,从而真正做到了算法与数据的分离.

3.3 可视化算法框架的设计

有了针对海量数据的基本计算框架及数据类型的抽象和封装之后,就可以很方便地对原有的各类算法框架进行升级,使其支持海量数据处理.在此基础上,由于算法所处理的数据访问接口是统一的,很多普通的 in-core 算法可以不加改变地移植过来成为支持海量数据的 out-of-core 算法,这里包括普通的图像处理类算法、表面重建算法(如 marching cubes 算法)、普通的表面绘制算法等.限于篇幅,这里不再一一细述各类算法框架的改造,而只对与体绘制相关的算法框架设计作一介绍.

与体绘制算法相关的是可视化算法框架.基于第 3.2 节的描述,我们升级了 MITK 原有的可视化算法框架,提供了对海量数据的支持,如图 4 所示,鉴于本文主要讨论体绘制算法,图中忽略了表面绘制部分.

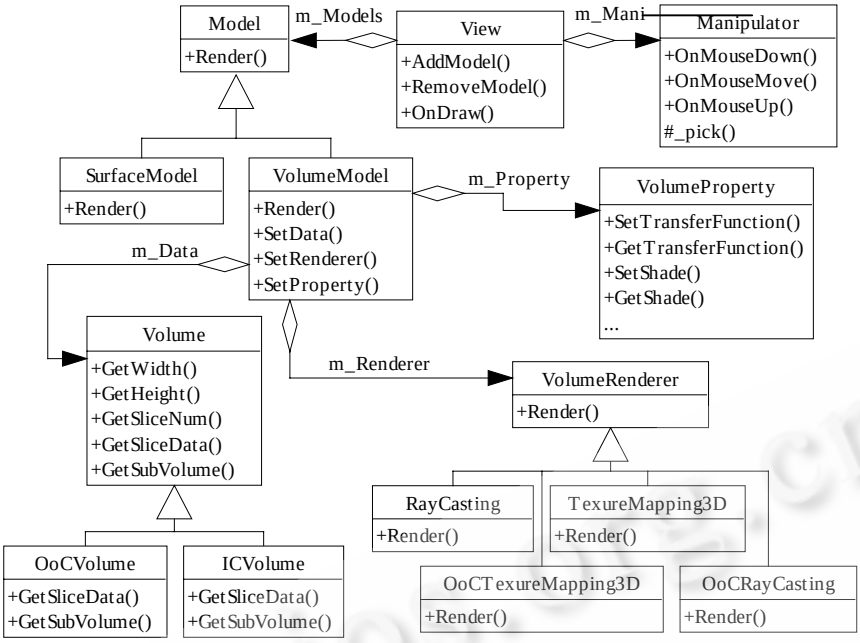


Fig.4 Framework of the out-of-core visualization algorithm

图 4 海量数据可视化算法框架

View 与具体操作系统相关联并提供一个绘制环境,它维护了一个 Model 列表,其中的每个 Model 负责将其所关联的数据通过特定算法绘制出来.而对于体绘制算法,VolumeModel 封装了绘制对象和具体绘制算法.绘制对象 Volume 提供对体数据的透明访问,绘制算法 VolumeRenderer 则负责加载具体的算法,其子类如 OoCRayCasting 和 OoCTexureMapping3D 等则提供了对海量数据的体绘制支持.这样就将绘制算法与数据进行了有效的分离,同时,我们也能将对海量数据处理的支持有效地融入这个框架中而不改变整个框架的对外接口.基于该框架,可以很方便地添加和测试新的可视化算法而无须太多的重复劳动.

4 海量医学影像数据快速体绘制算法

基于上述海量医学影像数据处理算法框架,我们分别对现有的光线投射体绘制算法和基于纹理的体绘制算法作了改进,使其能够实现海量医学影像数据的快速体绘制,下面分别加以介绍.

4.1 基于光线投射的海量数据体绘制算法

传统的光线投射体绘制算法是从最终投影图像平面的每个像素发出一条光线穿过体数据场,累积其所通过的体素对最终投影点像素的贡献.基于第 3 节所描述的算法框架,该传统算法可以不加改变地就可用于海量数据,但是绘制速度极低,因为传统方法在投射光线时需要对原始体数据场作随机访问,而在海量数据情况下,大部分数据是存储在磁盘上的,随机访问必将增加内存和磁盘的数据交换,将大量时间浪费在速度较慢的磁盘访问操作上,从而严重影响了绘制效率.文献[16]的方法将按光线计算的顺序改为按切片计算,避免了投射光线时对原始数据场的随机访问,从而可以在不将所有切片导入内存的情况下进行高效的光线投射体绘制.我们在第 3 节所描述的体绘制算法框架基础上,按照与文献[16]类似的思路实现了一个用于海量数据的光线投射体绘制算法,所不同的是,将 Slab 的大小从两张切片扩展到多张切片(切片数由内存缓冲的大小决定),而在处理透视投影下光线投射方向与切片导入顺序不一致的问题时,只是将光线分为正向和逆向光线,在需要时按不同方向导入所有切片两遍,分别投射不同类型的光线,而没有像文献[16]那样对切片作导入顺序的分类.另外,我们也没

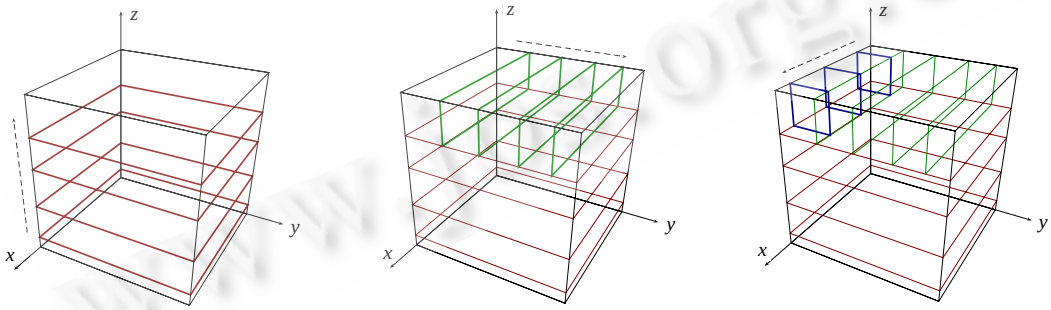
有采用文献[16]中提出的自适应超采样(adaptive supersampling)的方法对原始数据作更细致的采样.尽管如此,我们也得到了不错的绘制结果,同时,绘制速度要比将传统算法直接用于海量数据快得多.

上述算法实际上还有改进的余地,在实际的海量数据集中,往往存在很大一部分体素值一致的区域,这一部分区域在经过传递函数映射后很可能成为完全透明的区域[20],对最终的绘制结果没有任何贡献,从而在绘制时可以略过,而上述算法并没有考虑这种特殊区域的存在,有很大一部分绘制时间浪费在从外存导入这部分不需要绘制的数据上.

针对这一问题,如前所述,已有研究人员采用分块的方式处理大规模的数据,但是现有的算法存在一些问题,能直接用于海量数据的体绘制.例如,文献[17]提出的自适应分块方法,首先由于其使用盒区域增长(box growing)的方式生成最初的分块,而区域增长过程中的跨切片操作将带来相同切片的重复导入,并且切片的导入也是随机的,这使得预处理成为一个极其低效的过程;其次,最初的分块由于存在相互遮盖的区域而无法用平面分离相邻的子块,因此在建立用于最终绘制的 BSP 树时需要对初步的分块再进行分割和合并处理,这无疑进一步增加了预处理的复杂程度;第三,完全自适应的分块难免产生一些比较小的子块,而过多的对小数据块的读写操作必然会影响内外存数据交换的效率.上述这些因素极大地限制了这种分块方法在海量数据处理中的应用.而文献[18]所使用的等尺寸分块策略虽然可用于海量数据绘制,但由于不能像文献[17]那样将背景与非背景部分尽可能地分离,在一定程度上也影响了最终的绘制效率,同时在处理海量数据时也很难找到一个合适的分块尺寸.

鉴于上述考虑,我们提出并实现了一种新的基于光线投射的海量数据直接体绘制算法.该算法对原始体数据进行一种半自适应的分块,其关键是在自适应分块的基础上加一些限制条件,避免产生相互遮盖的子数据块以及过小的子数据块,以适应对海量数据的处理.该方法总体上说还是采取长方体分块,附加限制包括两类:一是子数据块的产生过程:不像完全自适应的方法由一块种子区域逐渐生长出来,而是采用在 x,y,z 三个坐标轴方向以逐级递进的方式产生子数据块;二是子数据块的大小:为了不产生过小的数据块,同时使最后的子数据块能够装入显卡纹理缓存以便对子数据块的绘制进行硬件加速,对子数据块长方体 3 个方向的长度各设定了最小值和最大值限制.

该方法的具体步骤是:首先在 z 坐标轴方向对原始数据进行分层处理,分割出体素值一致的区域,如图 5(a)所示.因为在算法框架的底层,原始数据在磁盘上是按照切片组织存储的,先对 z 方向进行分割较好地利用了这一点,在预处理过程中减少了切片数据的重复读取.接下来,对上一步骤产生的 z 方向上的每一个板状分块在 x 和 y 方向继续分割,这时,x,y 的地位是对等的,可以任选一个方向作为下一级分割的方向,仍然是将体素值一致的区域分割出来,以 y 方向为例,如图 5(b)所示(为清晰起见,图中只显示了一个板块的分割,其他板块类似).最后即是在上一步产生的每个条状分块中在剩下的一个方向上作最后的分割,以 x 方向为例如图 5(c)所示.



(a) Partition along z axis (a) 沿 z 方向分割 (b) Partition along y axis (b) 沿 y 方向分割 (c) Partition along x axis (c) 沿 x 方向分割

Fig.5 Demonstration of semi-adaptive partitioning

图 5 半自适应分块步骤示意图

上述算法将分块限制在 3 个坐标轴方向上,逐级递进,而在每个坐标轴方向上又采用自适应的分块方式,在

一定程度上提高了分块精度,能够比等尺寸分块更好地分离出可能的背景区域,同时也避免了完全自适应分块给海量数据处理带来的效率影响,在通常情况下,只需遍历所有切片两次即可完成分块,比完全自适应的分块大大缩短了预处理时间,同时也不会产生过小的分块而影响后续步骤的绘制效率.

此外,在分块边界处需要特别处理,在相邻分块边界保留一定的重复区域,以保证在边界处也能得到正确的绘制结果.这里引入了两种类型的重复区域:一种是边界(margin)区域,该类重复区域并不纳入子数据块绘制范围,主要是为了保证边界处能够得到正确的插值结果;另一种是交迭(overlap)区域,该类重复区域在绘制范围之内,对于光线投射来说一般只保留一个像素的交迭区域,从而避免在边界处投射光线出现断裂现象.

分块后为了得到正确的绘制结果,必须按照从前往后的顺序导入每个子数据块,在子数据块内进行光线投射计算.要达到这一目的,建立子数据块空间的 BSP 树(binary space partitioning tree)是最直接而有效的方式,通过中根遍历 BSP 树并根据视点与分割平面的位置关系即可得到正确的子数据块绘制顺序.而基于上述分块方法,BSP 树的建立是非常自然的,也是采用逐级递进的方式,先是在 z 方向进行分割,建立 z 方向板状分块的 BSP 树,这时,每个叶节点即表示 z 方向的一个子块区域;然后从每个叶节点开始在 y 或 x 方向进行分割,建立条状分块的 BSP 树,最终的叶节点表示 y 或 x 方向的一个条状子块区域;最后在上一步得到的每个叶节点对剩下的一个方向进行分割,最终建立完整的 BSP 树.而在每一级的空间分割中均采用折半分割的方式,保证分割平面两边的子数据块各占一半(最多相差一个),这样就使得最终建立起来的 BSP 树能够尽可能地保持平衡,不影响遍历的效率.建树过程实际上是与分块同步进行的,分块完成,相应的 BSP 树也就建立起来了.

在绘制时按照从前往后的顺序遍历 BSP 树,在当前导入的子数据块中推进光线,并且忽略经传递函数映射为完全透明的数据块,从而可以提高绘制效率.同时,由于采用分块策略并限制每个子数据块的大小,使得每块数据可以单独被装入容量比内存更有限的显卡显存,因此,本算法进一步利用目前主流显卡的可编程特性在 GPU 上实现了每个子数据块内的光线投射算法,从而使绘制效率得到进一步的提高.

4.2 基于三维纹理的海量数据体绘制算法

基于纹理的体绘制算法由于充分利用了显卡的插值计算能力,因此绘制速度一般要快于基于光线投射的算法,而显卡纹理缓存大小是有限的,并且比内存容量要小得多,所以若要利用三维纹理硬件加速对海量数据的体绘制,仍然必须对原始数据进行分块.这里,我们将利用主流显卡提供的三维纹理功能来加速海量数据的直接体绘制.整个算法分为 3 个步骤:数据预处理;计算当前视线方向并确定子块绘制顺序;分块绘制.

在第 1 步预处理中,首先是对原始数据进行分块,采用与第 4.1 节相同的半自适应分块方法对原始数据进行分块,各块数据存入磁盘备用;其次是对体绘制传递函数进行处理,生成快速查找表,以便后面在绘制子块时能够快速生成最终用于绘制的纹理块.如果开启了光照效果,则需要计算原始数据的梯度场,用于后面的光照计算,这里参考 VTK 中的相关算法,将梯度向量编码为一个 16bit 的整型数,用于索引对一个八面体进行递归细分产生的向量表,向量表中的向量可以近似代替原始梯度向量,这样就有效地减少了梯度向量所需的存储空间和光照效果的复杂运算.即使如此,梯度场的数据量与原始体数据相当,仍然是海量数据,需要用一个 OoCVolume 来存储它,同时采用与原始数据一样的方式对其进行分块,将分块后的梯度数据附加在相应子体数据块的后面,以备光照计算使用.在第 1 步的最后是根据传递函数对子块进行分类,剔除完全透明的子块,从而提高绘制效率.

在第 2 步中,首先根据当前视点位置计算视线方向,按照视线方向从后向前遍历子数据块的 BSP 树即可得到子数据块的正确绘制顺序.如果开启了光照效果,则在这一步中还需要根据当前的视线方向更新光照计算表,该表将第 1 步所得 16bit 编码的梯度向量映射到经光照计算后的颜色值,由于已对向量作了近似处理,该光照计算表的大小是有限的,可以直接放在内存中以便快速访问.

在第 3 步中,按照第 2 步所确定的子块绘制顺序遍历所有子块,对每个子块:首先应用第 1 步生成的传递函数查找表快速生成最终绘制所用的纹理块(而如果开启了光照效果,则还需经过光照计算表的映射,合成最终的纹理颜色值),并将此纹理块导入到显卡的纹理缓存中;然后,用垂直于视线的平面从后向前切割纹理块长方体,生成纹理多边形.长方体边与平面求交可以得到多边形顶点;最后,仍然按从后向前的顺序绘制所有生成的纹理

多边形,得到子块的绘制结果.在生成纹理多边形时,为了避免复杂的点集凸壳计算,我们用另一种方法来快速地生成这些多边形.当平面与长方体相交时,其 8 个顶点共有两种状态:在平面正向空间内及平面上(代入平面方程求得的值大于等于 0)或者在平面负向空间内(代入平面方程求得的值小于 0).所以平面与长方体相交最多有 2^8=256 种情况,实际上其中有一部分是不可能出现的,但是为了简化查找表结构以加快查找,仍采用 256 个元素的列表记录这些情况,8 个顶点的状态码排在一起恰好作为索引,而每个列表元素记录长方体与平面相交的边的序号,其顺序就是多边形顶点顺序.一个平面最多同时与长方体的 6 条边相交,所以每个列表元素固定为 7 个元素的整数数组,其中第 1 个元素为多边形顶点个数,后面依次排列顶点所在长方形边的序号.这样,生成多边形时,只需计算 8 个顶点的状态码,得到索引,查表得到多边形顶点序列,依次计算各顶点坐标即可,避免了先求顶点坐标再生成多边形所需的复杂运算,加快了绘制流程.

5 实验结果

我们在 MITK 中实现了第 3 节所描述的海量医学影像数据处理算法框架,并在此基础上实现了第 4 节所描述的基于光线投射和三维纹理的海量数据体绘制算法.为了对比测试算法性能,我们同时还实现了 in-core 的光线投射体绘制算法及其直接基于海量数据处理算法框架的 out-of-core 版本.

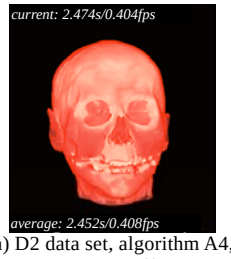
为了便于对比实验,我们采用了同一数据集的不同尺寸版本.原始数据来源于 www.psychology.nottingham.ac.uk,其尺寸为 208×256×225×8bit(D1,11.43MB),我们将其重采样至 624×768×675×8bit(D2,308MB)和 1040×1280×1125×8bit(D3,1.39GB),作为 3 个级别的测试数据集,以测试比较算法在各种规模数据集上的性能.测试环境为:Pentium 4 2.8GHz 处理器,1GB 物理内存,GeForce 6800 显卡,Windows 2000 操作系统.经测试,各种算法性能对比见表 1,其中 Speed 一栏给出的是绘制一帧所需的时间,Memory 一栏给出的是内存缓冲区大小.表 1 内的各项解释为:A1:基于光线投射的 in-core 算法;A2:A1 的 out-of-core 版本;A3:类似文献[16]的算法;A4:基于半自适应分块和 GPU 加速的光线投射算法;A5:基于等尺寸分块和三维纹理加速的算法;A6:基于半自适应分块和三维纹理加速的算法.当打开光照时,内存占用量包括每个体素的 16 位编码法向量.

Table 1 Comparison of the algorithm test results

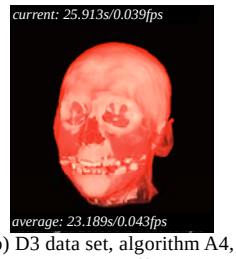
表 1 算法测试结果对比

		D1(11.43MB)		D2(308MB)		D3(1.39GB)	
		Speed (s)	Memory	Speed (s)	Memory	Speed	Memory
Shading off	A1	2.19	11.43MB	6.95	308MB	N/A	N/A
	A2	95.7	1.63MB	741	14.6MB	≥1h	40.6MB
	A3	2.35	1.63MB	7.3	14.6MB	43.3s	40.6MB
	A4	0.15	2.32MB	2.45	17.3MB	23.2s	34.0MB
	A5	0.143	160KB	2.52	160KB	25.5s	1.25MB
	A6	0.118	2.32MB	2.04	17.3MB	9.09s	17.3MB
Shading on	A1	2.5	34.28MB	12.7	925MB	N/A	N/A
	A2	276	4.88MB	2447	43.9MB	≥1h	121.9MB
	A3	2.91	4.88MB	46.6	43.9MB	211s	121.9MB
	A4	1.32	4.66MB	25.9	34.5MB	131s	68.1MB
	A5	1.10	224KB	19.1	224KB	168s	1.75MB
	A6	0.922	3.49MB	18.2	25.9MB	243s	25.9MB

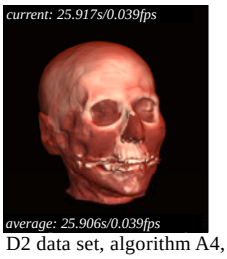
本文提出算法的部分绘制结果如图 6 所示.同时,为了测试本文算法在真实数据集上的处理能力,我们采用了第一军医大学所完成的中国虚拟人体 CT 扫描数据集,其尺寸为 512×512×1714×16bit(857MB),部分绘制结果如图 7 和图 8 所示.其中,图 7(a)为算法 A4 在关闭光照的情况下得到的结果,绘制时间为 4.46s,缓冲数据占用内存大小为 17.26MB;图 7(b)为算法 A4 在开启光照且只保留骨骼部分的情况下得到的结果,绘制时间为 25.7s,缓冲数据占用内存大小为 34.51MB;图 8(a)为算法 A6 在关闭光照情况下的绘制结果,绘制时间为 5.50s,缓冲数据占用内存大小为 7.06MB;图 8(b)为算法 A6 在开启光照且只保留骨骼部分情况下的绘制结果,绘制时间为 14.6s,缓冲数据占用内存大小为 9.41MB.



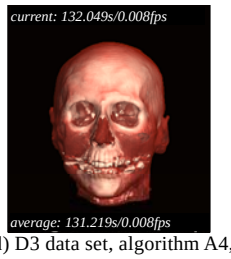
(a) D2 data set, algorithm A4, shading off
(a) D2 数据集,算法 A4, 关闭光照



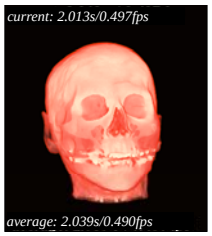
(b) D3 data set, algorithm A4, shading off
(b) D3 数据集,算法 A4, 关闭光照



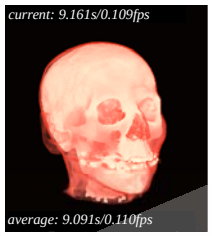
(c) D2 data set, algorithm A4, shading on
(c) D2 数据集,算法 A4, 开启光照



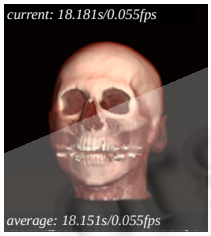
(d) D3 data set, algorithm A4, shading on
(d) D3 数据集,算法 A4, 开启光照



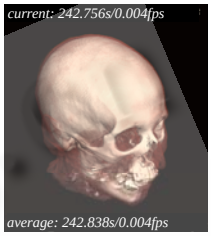
(e) D2 data set, algorithm A6, shading off
(e) D2 数据集,算法 A6, 关闭光照



(f) D3 data set, algorithm A6, shading off
(f) D3 数据集,算法 A6, 关闭光照



(g) D2 data set, algorithm A6, shading on
(g) D2 数据集,算法 A6, 打开光照



(h) D3 data set, algorithm A6, shading on
(h) D3 数据集,算法 A6, 打开光照

Fig.6 Some rendering results of our algorithms on the CT head data sets

图 6 本文所提出算法在 CT 头部数据集上的一些绘制结果

(a) Shading off
(a) 关闭光照

(b) Shading on
(b) 开启光照

Fig.7 Some rendering results of algorithm A4 on the Chinese visible human CT data set

图 7 算法 A4 在中国虚拟人体 CT 扫描数据集上的绘制结果

(a) Shading off
(a) 关闭光照

(b) Shading on
(b) 开启光照

Fig.8 Some rendering results of algorithm A6 on the Chinese visible human CT data set

图 8 算法 A6 在中国虚拟人体 CT 数据集上的绘制结果

由比较结果可以看出,在参与对比的算法中,本文提出的基于半自适应分块及 GPU 加速的海量数据光线投射体绘制算法(算法 A4)能够处理 in-core 算法不能处理的海量数据,绘制速度比 in-core 算法直接的 out-of-core 版本要提高两个数量级以上,同时比单纯按切片投射光线的算法 A3 平均快 1 倍以上,但由于显卡使用 16bit 浮点运算,精度不及算法 A3,绘制效果要略逊一筹;而算法 A3 由于计算过程完全与 in-core 算法等价,因此绘制效果相对较好,但速度却不及算法 A4,同时要得到较高的绘制速度,缓冲数据所占用的内存也要大于算法 A4.

使用三维纹理加速的算法由于没有复杂的光线投射计算,故在关闭光照时绘制速度一般要优于基于光线投射的算法,但与 GPU 加速的光线投射算法相比差别不大;而在开启光照时由于读取 16bit 编码法向量的开销,

速度没有在关闭光照的情况下那样大幅度的提高.而基于半自适应分块和三维纹理加速的算法在一般情况下是绘制速度较快的,且能够得到比较好的绘制效果.

另外,为了测试半自适应分块方法的有效性,我们在中国虚拟人体 CT 扫描数据集上比较了等尺寸分块和半自适应分块两种分块策略的性能,其结果见表 2,其中,半自适应分块在每个坐标轴方向上的尺寸有最小值和最大值约束,因此是一个闭区间.

Table 2 Comparison of the different partitioning methods

表 2 不同分块方法的对比

Partitioning method	Block size	Non-Background blocks	Read voxels
Partitioning with uniform block size	48×48×48	2 058	227 598 336
	64×64×64	894	234 356 736
	128×128×128	155	325 058 560
Semi-Adaptive partitioning	[16,48]×[16,48]×[16,48]	5 206	193 297 442
	[16,64]×[16,64]×[16,64]	1 100	211 624 837
	[16,250]×[16,32]×[16,64]	748	219 153 002

从对比结果可以看出在不同尺寸下,半自适应分块后需读取的体素(非背景区域)总量始终要小于等尺寸分块,确实能够比等尺寸分块更准确地分离出背景区域,而分割后的子数据块个数通常要比等尺寸分块多,但是通过调节尺寸参数很容易取得非背景子数据块个数和读取体素总量之间较好的平衡.

6 总结及展望

本文针对医学影像处理与分析领域中海量(out-of-core)数据处理的需求,设计并实现了一套面向对象的针对海量数据处理和分析的算法框架,并将其融入实验室开发的医学影像算法研发平台 MITK 中,完成了面向海量医学影像数据处理与分析的 MITK 2.0 并在 www.mitk.net 上发布,真正建立起一个海量医学影像数据的处理平台,并在此基础上研究和实现了针对海量数据的基于光线投射和三维纹理的快速体绘制算法.实验结果表明,该算法框架有效地对算法和数据进行了分离,屏蔽了底层实现细节,能够很好地运用于海量医学影像数据处理算法的研究和实现;同时,本文在此基础上改进的体绘制算法能够有效地对海量数据进行处理,与已有算法相比,在绘制速度和内存消耗上都有一定程度的改善.

尽管如此,目前海量数据的体绘制算法在大数据量(超过 10G)的情况下仍然无法达到实时交互的要求,基于硬件加速的算法其绘制效果也尚待进一步改善.因此下一步所要进行的工作,首先是继续发展和完善海量数据处理的算法框架,提供更高效的海量数据存取操作;其次是继续针对难度较大的海量数据可视化算法进行研究,进一步提高绘制速度和改善绘制效果.

References:

[1] Schroeder WJ, Martin K, Schroeder BL. The Visualization Toolkit: An Object Oriented Approach to 3D Graphics. 3rd ed. New York: Kitware, Inc., 2003.

[2] Ibáñez L, Schroeder W, Ng L, Cates J. The Insight Software Consortium. The ITK Software Guide. New York: Kitware, Inc., 2003.

[3] Zhao MC, Tian J, Zhu X, Xue J, Cheng ZL, Zhao H. The design and implementation of a C++ toolkit for integrated medical image processing and analyzing. In: Jr. Galloway RL, ed. Proc. of the SPIE, Vol.5367. San Diego: Int'l Society for Optical Engineering, 2004. 39−47.

[4] Levoy M. Display of surfaces from volume data. IEEE Computer Graphics and Applications, 1988,8(3):29−37.

[5] Drebin RA, Carpenter L, Hanrahan P. Volume rendering. In: Beach RJ, ed. Proc. of the 15th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press, 1988. 65−74.

[6] Akeley K. Reality engine graphics. In: Whitton MC, ed. Proc. of the 20th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press, 1993. 109−116.

[7] Cabral B, Cam N, Foran J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: Kaufman A, Krueger W, eds. Proc. of the 1994 Symp. on Volume Visualization. New York: ACM Press, 1994. 91−98.

[8] Wilson O, Van Gelder A, Wilhelms J. Direct volume rendering via 3d textures. Technical Report, Santa Cruz: University of California at Santa Cruz, 1994.

[9] Van Gelder A, Kim K. Direct volume rendering with shading via three-dimensional textures. In: Crawfis R, Hansen C, eds. Proc. of the 1996 Symp. on Volume Visualization. Piscataway: IEEE Press, 1996. 23−30.

[10] Westermann R, Ertl T. Efficiently using graphics hardware in volume rendering applications. In: Cunningham S, Bransford W, Cohen MF, eds. Proc. of the 25th Annual Conf. on Computer Graphics and Interactive Techniques. New York: ACM Press, 1998. 169−177.

[11] Silva CT, Chiang YJ, Sana JE, Lindstrom P. Out-of-Core algorithms for scientific visualization and computer graphics. Course Notes for Tutorial 4 of the IEEE Visualization 2002. 2002. http://cis.poly.edu/chiang/Vis02-tutorial4.pdf

[12] Farias R, Silva CT. Out-of-Core rendering of large, unstructured grids. IEEE Computer Graphics and Applications, 2001,21(4): 42−50.

[13] Farias R, Mitchell JSB, Silva CT. Zsweep: An efficient and exact projection algorithm for unstructured volume rendering. In: Lorensen B, Crawfis R, Cohen-Or D, eds. Proc. of the 2000 IEEE Symp. on Volume Visualization. New York: ACM Press, 2000. 91−99.

[14] Chiang YJ, Farias R, Silva CT, Wei B. A unified infrastructure for parallel out-of-core isosurface extraction and volume rendering of unstructured grids. In: Breen D, Heirich A, Koning A, eds. Proc. of the IEEE Parallel Visualization and Graphics Symp. Piscataway: IEEE Press, 2001. 59−151.

[15] Nuber C, Bruckschen RW, Hamann B, Joy KI. Interactive visualization of very large medical datasets using point-based rendering. In: Jr Galloway RL, ed. Proc. of the SPIE Medical Imaging 2003, Vol.5029. San Diego: International Society for Optical Engineering, 2003. 27−36.

[16] Novins KL, Sillion FX, Greenberg DP. An efficient method for volume rendering using perspective projection. Computer Graphics, 1990,24(5):95−102.

[17] Li W, Mueller K, Kaufman A. Empty space skipping and occlusion clipping for texture-based volume rendering. In: Turk G, van Wijk JJ, Moorhead II RJ, eds. Proc. of the 14th IEEE Visualization 2003. Washington: IEEE Computer Society, 2003. 42.

[18] Wang SR, Jia FC, Sun XP, Liu LY, Li H. Hardware-Accelerated volume rendering based on digital human slice images. Journal of Computer-Aided Design & Computer Graphics, 2005,17(9):1997−2002 (in Chinese with English abstract).

[19] Tong X, Wang WP, Tsang WW, Tang ZS. Efficiently rendering large volume data using texture mapping hardware. Journal of Tsinghua University, 2000,40(1):72−75.

[20] Kniss J, Kindlmann G, Hansen C. Multidimensional transfer functions for interactive volume rendering. IEEE Trans. on Visualization and Computer Graphics, 2002,8(3):270−285.

附中文参考文献:

[18] 王少荣,贾富仓,孙晓鹏,刘丽艳,李华.数字人切片数据的硬件加速体绘制.计算机辅助设计与图形学学报,2005,17(9):1997−2002.

薛健(1979－),男,江苏宜兴人,博士,主要研究领域为三维可视化.

戴亚康(1982－),男,博士生,主要研究领域为三维可视化.

田捷(1960－),男,博士,教授,博士生导师,主要研究领域为医学影像处理,生物特征识别.

陈健(1981－),男,博士生,主要研究领域为医学图像分割与配准.