文章编号:1671 - 6833(2016)03 - 0006 - 05

基于双缓冲队列的海量地形数据并行处理方法

陈小潘[1],渠涧涛[1,2],赵亚萌[2],王 鹏[1,2],陈玉林[1]

(1. 河南大学 计算机与信息工程学院,河南 开封 475004; 2. 中国科学院 遥感与数字地球研究所,北京 100012)

摘 要: 海量地形数据处理过程中,硬件的性能优势不能得到充分利用,已成为制约大规模地形绘制速度的瓶颈. 针对这一问题,分析了影响硬件处理能力发挥的关键因素,采用双缓冲队列的设计思路,提出了一种支持海量地形数据的并行处理方法,即将大规模地形绘制分为数据处理和渲染绘制两个独立的过程,分别进行并行处理:设立两个缓冲队列,将数据的读、写操作区分开来;充分考虑了瓦片加载的优先级,并据此进行任务分配. 实验表明,该方法有效地提升了大规模地形绘制的整体速度.

关键词: 海量地形数据;双缓冲队列;并行化;大规模地形绘制;瓦片加载

中图分类号: TP391 **文献标志码:** A doi:10.13705/j.issn.1671 - 6833.2016.03.002

0 引言

大规模地形绘制在战场模拟、虚拟现实以及地理信息系统中都有着广泛的应用[1-2]. 然而,用于生成地形瓦片模型的高程数据以及纹理数据的数据量大,无法一次性全部存入内存. 通常的解决方法是将数据按照金字塔模型的标准进行切分,分块调入内存[3-5],使得一个三维场景由相当数量的地形瓦片模型构成. 在硬件处理能力不足的情况下,尽可能地简化地形瓦片模型的复杂度,是实现大规模地形绘制的一个重要手段[6]. 然而,随着计算机图形处理硬件性能的不断提高,如何有效地调度及处理大量的瓦片数据,使得计算机的处理能力能够得到充分利用,已经成为提升大规模地形绘制速度的一个关键.

近几十年来,国内外学者在大规模地形实时绘制方面进行了大量的研究,一个主流研究方向是基于层次细节模型(LOD, Levels of Detail)[7-10]. 该方法的主要思路是采用不同的标准筛选出每个无效的三角形并剔除,以减少待渲染三角形的数量. 但近几年,随着 CPU(Central Processing Unit)和 GPU(Graphics Processing Unit)等硬件的处理能力不断提升,通过一定的技术手段,充分发挥硬件的处理能力,已经成为提升大规模地形绘制速度的一个新的方向[11-13].

基于以上研究现状,笔者提出了一种基于双缓冲队列的海量地形数据并行处理方法:在依据瓦片加载优先级进行任务分配后,将地形数据处理与地形瓦片模型的渲染绘制分隔开来,分别进行并行处理,以充分发挥硬件的性能;同时,设计两个缓冲队列交替作为数据的读队列和写队列,减少了线程间同步产生的开销,并利用 WPF(Windows Presentation Foundation)技术完成任务.

1 问题分析与整体架构设计

大规模地形绘制中海量地形数据的处理,通常由多个地形瓦片数据的处理组合而成[14]. 采用单缓冲队列的单线程处理流程效率较低,可采用单缓冲队列的并行处理机制进行处理,如图 1 所示.

数据处理和渲染绘制分别进行处理的设计模式区分了 CPU 与 GPU 的职责,并行处理方式也使得两者的性能在整个海量地形数据处理的过程中都得到了充分的利用. 但单一的缓冲区在面对多线程的读写操作时,会出现线程同步的问题,造成时间上的浪费. 同时,在加载瓦片时,并未考虑瓦片加载的优先级,也降低了用户体验. 因此,笔者设计了一种基于双缓冲队列的海量地形数据处理方法,该方法具有以下特点.

收稿日期:2015 - 09 - 26;**修订日期:**2015 - 11 - 19

基金项目: 国家'973'计划资助项目(Y070072070);国家国防科技工业局高分重大专项项目(Y4D0100038;Y4D00100GF)

作者简介: 陈小潘(1982—),男,河南偃师人,河南大学讲师,博士研究生,主要从事空间分析优化、图像处理与模式识别研究,E-mail:xpchen@henu.edu.cn.

(1)根据瓦片中心点与视线的距离,设立瓦片加载的优先级. 当距离越近,则优先级越高,在单个线程中优先级越高的瓦片就越早被加载.

(2)将瓦片的绘制过程分为数据处理和渲染绘制两个独立的过程,并采用多线程并发的策略进行加速.

(3)采用双缓冲队列的设计思路,减少了线程间同步产生的开销.

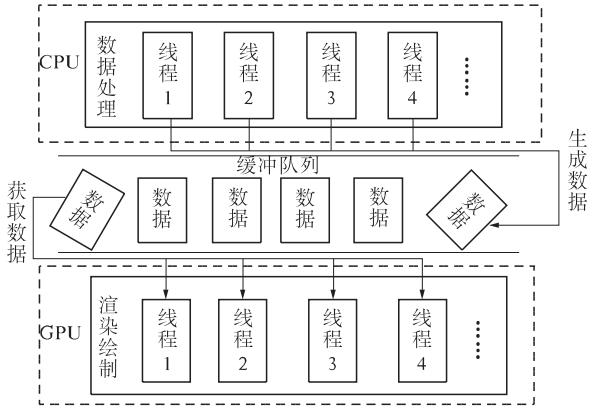

图 1　单缓冲队列的并行处理机制

Fig. 1　Parallel processing mechanism of single buffer queue

基于双缓冲队列的海量地形数据并行处理架构,主要由瓦片数据选取及优先级处理、双缓冲队列、数据处理以及渲染绘制 4 个主要模块构成,其整体架构设计如图 2 所示.

其中,瓦片数据选取及优先级处理模块主要根据当前的视域范围选取待加载瓦片数据;同时,根据视点与瓦片之间的距离,计算所有待加载瓦片数据的优先级,生成瓦片加载优先级队列,并将生成的瓦片加载优先级队列传递给数据处理模块.

缓冲队列 1 和缓冲队列 2 的主要功能是提供固定大小的数据缓冲空间,分别负责读出和写入,使读操作和写操作分别针对不同的缓冲区进行,从而减少了线程间同步的开销. 当读队列为空,或者写队列满时,两个队列会进行交换,保证读写操作能够不间断进行.

数据处理模块能够根据瓦片的绘制优先级为各个线程分配数据处理任务,保证每个线程中优先级较高的瓦片数据能够优先被处理. 数据处理任务主要有:根据地形瓦片模型网格的大小对高程数据进行插值拉伸;计算网格顶点的索引值;以及确定每个顶点的纹理映射等. 数据处理完毕后,直接存入写队列中;在写队列满时,能够控制队列交换.

渲染绘制模块能建立多个线程,且并行地从读队列中读出数据,利用读出的数据最终生成地形瓦片模型,加载到场景中. 在读队列为空时,能够控制队列进行交换.

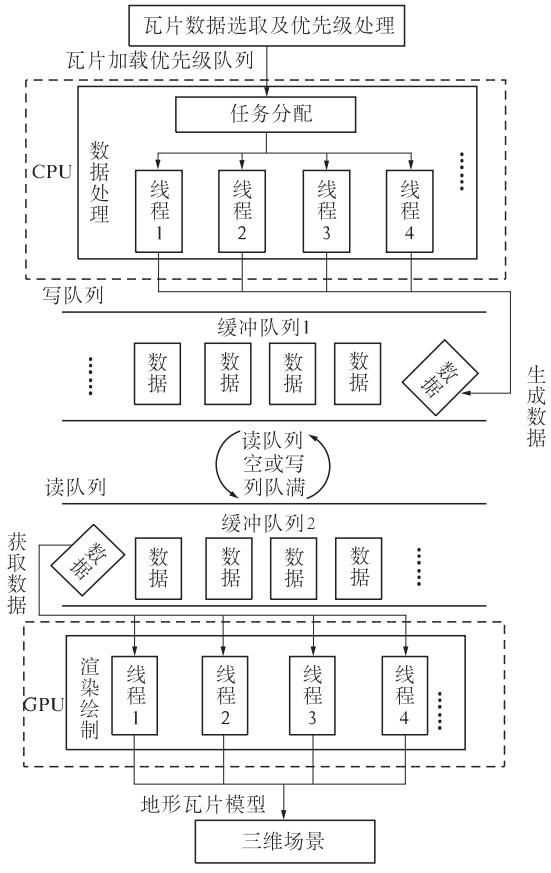

图 2　双缓冲队列的并行处理机制整体架构图

Fig. 2　Parallel processing mechanism of double queue

2　算法实施

笔者采用金字塔模型对海量地形数据进行组织和存储,其基本思想是自底向上生成金字塔,根据需求直接取其中某一级作为操作对象以提高整体效率. 在实现海量地形数据处理之前,场景中所有待处理的高程数据以及纹理数据,均已经按照金字塔瓦片模型的设计思路,按 Google Map 标准切分成瓦片,切分后的瓦片数据按照以下形式存储在本地,如:“D:\Root\4\3\2. abc”,该路径所代表的含义是:“磁盘根目录\数据根目录\层级\行号\列号. 后缀名”.

步骤 1:计算待加载瓦片行列号的范围. 首先,根据屏幕的可视范围,建立同屏幕视域等大的边界盒(Bounding Box,BBOX). 在 Google Map 标准中,可以通过公式(1)~(3)获取任意经纬度所

在瓦片的行列号:

$$n = 2^{\text{zoom}}; \tag{1}$$

$$x\text{tile} = \frac{(\text{lon} + \pi)}{2\pi} \times n; \tag{2}$$

$$y\text{tile} = n - (1 - (\log \frac{\tan(\text{lat}) + \sec(\text{lat})}{\pi}))/2 \times n. \tag{3}$$

式中:n 是当前层级列方向或行方向上瓦片总数;zoom 是层数;$x$tile 和 $y$tile 分别表示列号和行号;lon 是瓦片左上角的经度;lat 是瓦片左上角的纬度.

然后,将边界盒的最小经度、纬度以及最大经度、纬度分别代入公式(1)~(3),就能够确定整个可视范围内待加载瓦片行列号的范围.

步骤2:计算瓦片的加载优先级.确定单张瓦片加载优先级的主要依据是:瓦片中心点与视点的距离,距离越小,绘制的优先级就越高.依据步骤1中的公式可得到以下公式:

$$f(x\text{tile}) = \text{lon} = \frac{x\text{tile}}{n} \times 2\pi - \pi; \tag{4}$$

$$g(y\text{tile}) = \text{lat} = \text{arc}\tan(\sinh(\pi \times (1 - 2 \times \frac{n - y\text{tile}}{n}))). \tag{5}$$

假设瓦片的行号为 x,列号为 y,则瓦片的中心点坐标经纬度 CenterLon、CenterLat 分别可由以下公式求出:

$$\text{CenterLat} = \frac{g(y) + g(y+1)}{2}; \tag{6}$$

$$\text{CenterLon} = \frac{f(x) + f(x+1)}{2}. \tag{7}$$

求出瓦片的中心点经纬度坐标后,再转化为三维空间坐标;根据三维空间坐标系两点间距离的运算公式,求出瓦片中心点与视点的距离,然后将瓦片按距离大小从小到大排序,便得出了一个瓦片加载优先级队列.

步骤3:按照瓦片加载的优先级进行任务分配.将所有瓦片的数据处理任务,按照瓦片加载优先级分配给多个线程并行处理.任务分配时,将所有待处理任务,按照瓦片加载优先级从大到小依次分配给各个线程进行处理.假设负责数据处理的线程总数为4,具体分配方法如图3所示.

从步骤2得出的瓦片加载优先级队列中,按优先级从高到低得到待处理瓦片,依次分配给各个线程.设 Tile_k 为瓦片加载优先级队列中瓦片,则第 m 个线程中待处理瓦片队列为:$\{\text{Tile}_{4k+m}\}$ ($k=0,1,2,\cdots$),这就保证了每个线程中优先级较高的瓦片能够优先得到处理.

步骤4:执行数据处理.任务分配完成后,各个线程开始执行数据处理任务.数据处理任务除了进行高程数据的插值拉伸、生成顶点、计算顶点索引和纹理映射之外,还需要读取写队列的状态,控制两个缓冲队列进行交换.数据在单个线程上的处理过程如下.

(1)从待处理瓦片队列中获取原始数据,进行拉伸、插值等处理.

(2)判断当前写队列是否已满,如果队列已满,执行(3);如果队列未满,执行(4).

(3)通过原子操作更改读写队列状态标识符,控制读写队列交换,返回(2).

(4)将处理完毕的数据写入写队列中,并判断待处理瓦片队列的状态,如果待处理瓦片队列为空,结束;否则,返回(1).

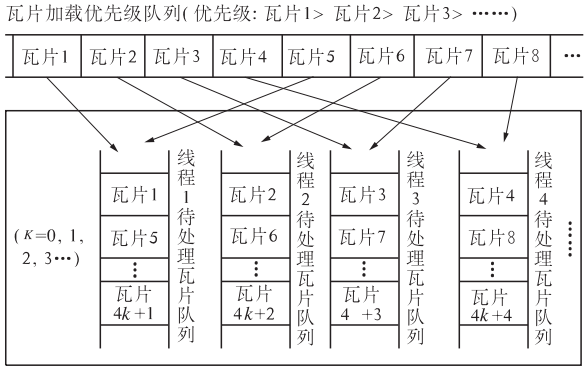

图3 基于优先级的任务分配

Fig. 3 Task allocation based on priority

步骤5:地形绘制.数据处理完毕后,渲染绘制模块利用处理完毕的数据生成地形瓦片模型,并将生成地形瓦片模型绘制到主界面中,这个过程同样采用并行的方式进行,但由于主界面所在的主线程不能被其他线程调用,所以,在三维场景中加载地形瓦片模型必须采用异步的方式.笔者利用 WPF 中 Dispatcher 对象进行异步调用.地形在单个线程上的绘制过程如下:

(1)判断当前读队列是否为空,如果读队列为空,则执行(2);如果不为空,则执行(3).

(2)通过原子操作更改读写队列状态标识符,控制读写队列进行交换,返回(1).

(3)获取处理完毕的数据,生成地形瓦片模型,并加载到场景中,将已加载瓦片数目加1.

(4)当已加载的瓦片数目与待加载的瓦片总数目相同时,完成地形绘制;否则,结束当前任务.

3　实验分析

在 Visual Studio 2012 的开发环境下利用 WPF 实现目标,并在硬件配置为 Intel(R) Core(TM) i7-2600 3.40GHz(8 CPUs),4G RAM,GeForce GT 420 的计算机上进行测试. 实验数据为北京昌平地区的 SRTM(Shuttle Radar Topography Mission)数据和 SAR 影像数据,其中,SRTM 数据的分辨率为 90 m,单个数据的像素尺寸为 1 024 ×1 024. 纹理图像采用相同地区的 SAR 影像数据制作,图像尺寸为 19 132 ×15 314,原始大小为 279 MB. 在进行实验之前,两种数据均已按照 Google Map 的标准进行了瓦片金字塔切分,单张瓦片的大小为 256 ×256.

3.1　确定最佳读、写线程数

为了确定最佳读、写线程数,以生成由 100 块地形瓦片模型组成的三维地形场景为目标,设置缓冲队列长度为 100 进行实验,每个平均值均由 15 次测量得出. 具体实验步骤如下:

步骤 1:确定数据处理的最佳线程数. 在不同的线程数下进行数据处理测试,记录每个线程数目下处理时间的平均值,令渲染绘制的线程数目为 1,测试结果如表 1 所示. 从表 1 可以看出,在线程数目为 8 时数据处理的平均时间达到最小. 为了得到最佳的处理速度,实验时采用 8 个线程对数据进行处理为最佳.

表 1　不同写线程下数据处理的平均时间

Tab.1　Average time of data processing under different write threads

线程数	1	2	3	4	5
平均时间/ms	537.3	527.3	478.6	475.6	447.1
线程数	6	7	8	9	10
平均时间/ms	449.7	446.6	433.8	455.3	459.5

步骤 2:确定渲染绘制的最佳线程数. 指定数据处理的线程数目为 8,记录在不同渲染绘制线程数目下处理时间的平均值,通过对比,最终确定最佳的渲染绘制线程数,实验结果如表 2 所示.

表 2　不同读线程数目下渲染绘制的平均时间

Tab.2　Average time of rendering under different read threads

线程数	1	2	3	4	5
平均时间/ms	445.8	432.3	451.0	457.6	497.3

由表 2 可以看出:当渲染绘制线程数目超过 2 个后,处理时间开始增加,因此,将渲染绘制线程的数目设置为 2.

步骤 3:对比实验. 按照以上两个步骤得到的最佳线程数目,建立双缓冲队列方法的实验场景,并与单线程和多线程单缓冲区两种方法进行对比实验,3 个实验场景如表 3 所示.

场景 1、2、3 分别代表:单线程、多线程单缓冲区以及多线程双缓冲区为对每个场景进行了 15 次测试,获得各个实验场景下生成三维地形所需的平均时间:场景 1 为 3 374.8 s,场景 2 为 790.5 s,场景 3 为 432.3 s.

表 3　对比实验场景

Tab.3　Scene of contrast experiment

参数	场景 1	场景 2	场景 3
数据处理线程数	1	8	8
渲染绘制线程数	1	2	2
缓冲队列容量	100	100	100
加载瓦片总数	100	100	100

可以得出:在大规模地形绘制中,采用双缓冲队列方法的处理速度,约是单线程方法的 7.8 倍,是多线程单缓冲队列的 1.83 倍.

3.2　对比分析

为了验证本文方法在处理大规模地形数据时的效率,选择与文献[7,10]中提出的两种传统 LOD 方法以及 Mipmap 方法进行对比. 其中,文献[7]提出了一种基于四叉树孤立分割和屏幕误差的地形 LOD 算法;文献[10]提出了一种 3D_DP 和 Quad_TIN 支持下的大规模地形自适应 LOD 算法,实验结果如表 4 所示.

表 4　实验结果对比

Tab.4　Comparison of experimental result

加载地形瓦片块数	所采用方法	绘制三角形个数	单帧绘制时间/ms
129	文献[7]方法	84 611	2 751.0
	本文方法	580 500	597.2
100	Mipmap 方法	450 000	3 162.0
	本文方法	450 000	432.3
44	文献[10]方法	34 602	3 483.0
	本文方法	198 000	197.5

从表 4 可以看出,在一次性加载大量地形瓦片时,本文方法在绘制时间上要优于 Mipmap 方法、文献[7]方法和文献[10]方法,这说明本文方法能够在保证地形模型细节的前提下,显著提升海量地形数据的处理速度. 在实际应用中,当对三维瓦片模型的展示细节要求较高、或需要频繁重新构建三维场景时,本文方法更有优势.

4 结束语

针对在大规模地形绘制过程中硬件的性能优势得不到充分利用的状况,提出了一种基于双缓冲队列的海量地形数据并行处理方法.该方法将瓦片的绘制过程分解为数据处理以及渲染绘制两个过程,将读、写操作分隔开来,减少了CPU与GPU的空闲时间.在对读、写两个过程进行并行处理的同时,设计了两个缓冲队列,分别负责暂存用于读入和写出的数据,这不仅能够充分发挥多核处理器的性能,而且能够最大限度地减少线程间同步所浪费的时间;在数据处理时也考虑了瓦片加载的优先级,从而根据优先级进行任务分配,提升了用户体验.实验表明,该方法能够显著提升海量地形数据处理速度,为海量地形数据处理提供了新的思路.后续研究将考虑把本文的方法与传统LOD的方法相结合,以达到更好效果.

参考文献:

[1] 姚凡凡,梁强,许仁杰,等. 基于Vega Prime的三维虚拟战场大地形动态生成研究[J]. 系统仿真学报,2012,24(9):1900-1904.

[2] 闫佳,闫枫. 基于XNA的地理信息系统的研究与实现[J]. 测绘与空间地理信息,2014,37(11):152-154.

[3] 龚桂荣,杜莹,欧阳峰. 地形金字塔中地物模型的空间划分及层级索引研究[J]. 测绘科学技术学报,2014,31(2):198-202.

[4] 李建勋,郭莲丽,李杨,等. 面向瓦片金字塔的层深确定与投影变换方法[J]. 计算机应用,2014,34(9):2683-2686.

[5] 应申,靳凤攒,李霖,等. 基于ArcGIS Engine的矢量数据分层分块技术研究[J]. 测绘地理信息,2014,39(6):50-53.

[6] 胡鹏昱,王晓军. 基于Geometry Clipmap算法的大规模地形可视化研究[J]. 计算机应用与软件,2012,29(7):87-90.

[7] 张俊峰,姚志宏. 基于四叉树孤立分割和屏幕误差的地形LOD算法[J]. 西南交通大学学报,2013,48(4):666-671.

[8] HEIDRICH W, SLUSALLEK P, SEIDEL H P. Real-time generation of continuous levels of detail for height fields[C]//Proc Winter School of Computer Graphics'98. Plzen: Science press,1998: 315-322.

[9] TURNER B. Real-time dynamic level of detail terrain rendering with roam (2000-04-03) [2015-02-02] [EB/OL]. http://www.gamasutra.com/view/feature/131596/realtime_dynamic_level_of_detail_.php

[10] 张俊峰,孙大鹏,许德合. 3D_DP和Quad_TIN支持下的大规模地形自适应LOD算法[J]. 地理与地理信息科学,2014,30(3):29-32.

[11] 时钢. 基于Mipmap的大规模地形绘制算法与仿真[J]. 计算机仿真,2015,32(2):270-274.

[12] 刘浩, 曹巍, 赵文吉, 等. 面向大规模地形的瓦片调度与实时绘制算法[J]. 计算机科学, 2013, 40(6A):120-124.

[13] 杨秀峰,靳海亮,臧文乾. 基于GPU并行处理的地形三维重建技术研究[J]. 江西科学,2014,32(1):22-25.

[14] 于卫东. 三维地理空间环境重建与绘制技术研究[D]. 郑州: 解放军信息工程大学地理空间信息学院, 2012.

Double-Buffered Queue-Based Parallel Processing Algorithm for Massive Terrain Dataset

CHEN Xiaopan[1], QU Jiantao[1,2], ZHAO Yameng[2], WANG Peng[1,2], CHEN Yulin[1]

(1. College of Computer and Information Engineering, Henan University, Kaifeng 475004, China; 2. Institute of Remote Sensing and Digital Earth, Chinese Academy of Sciences, Beijing 100012, China)

Abstract: When dealing with massive terrain data, the advantage of hardware performance can't be fully utilized. This has become a bottleneck, which restricts the speed of massive terrain tiles rendering. This paper analyzes the key factors that affect large-scale terrain rendering speed, and proposes a parallel algorithm for massive terrain data processing. The algorithm adopts double buffer queues and divides large scale terrain rendering into two parallel processing which includes data processing and rendering. The two buffer queues are responsible for data reading and writing operations in turn. The loading priority of terrain tiles is considered and tasks are allocated based on the priority. The experimental results show that this approach improves the speed of rendering massive terrain tiles greatly.

Key words: massive terrain data; double buffer queue; parallelization; large-scale terrain rendering; tile load