·信号与信息处理·

复杂网格模型仿射变换的GPU加速计算

钟 庆,华顺刚

(大连理工大学机械工程学院CAD与网络技术研究所,辽宁 大连 116024)

摘 要:为提高计算速度,复杂网格模型操作处理中的仿射变换计算被移植到具有可编程能力的GPU上实现。在并行计算中,每个线程计算一个顶点的k邻域权重值和坐标变换,多个线程同时执行。经过对线程结构安排、设备存储器分配等的优化,充分发挥GPU并行运算性能。实验结果表明,GPU加速计算对大规模网格顶点仿射变换的处理得到了较好的加速效果。

关键词:图形处理器;并行计算;三维网格变形;仿射变换

中图分类号:TP391.72 **文献标识码**:A **文章编号**:1673-1255(2011)01-0059-04

GPU Accelerated Computation for Affine Transformation of Complex Mesh Model

ZHONG Qing, HUA Shun-gang

(*Institute of CAD and Network Technology, School of Mechanical Engineering, Dalian University of Technology, Dalian* 116024, *China*)

Abstract: The affine transformation of complex mesh model is implemented on the programmable GPU in order to speed up the calculation process. Based on the characteristics of parallel computing, each thread calculates the weight of the *k*-nearest neighbor and the transformed coordinate for one node, as well as multithreads run at the same time. By arranging the thread hierarchy and distributing the device memory reasonably, the parallel operation capability of GPU is brought into full play. And the experimental result shows GPU executes the affine transformation of massive mesh model with a significantly accelerated effect.

Key words: graphic processing unit; parallel computing; 3D mesh deformation; affine transformation

在设计及制造领域中,经常需要对复杂模型进行局部或整体的变形操作,以形成不同的产品外观造型,以及满足产品结构性要求等。但是在处理较复杂的网格模型时,因其曲面形状不规则,顶点数据量大,操作处理中所涉及的顶点仿射变换计算时间长,难以满足实时变形渲染的要求。

近年来,图形处理器(graphic processing unit, GPU)的并行处理能力不断提升,给大规模数据快速计算提供了一条新思路。统一计算设备架构(compute unified device architecture, CUDA)[1]是新兴的GPU运算平台,采用单指令多线程(SIMT)的执行模式,同一个核心函数(kernel)被多个线程同时执行,缩短计算时间,达到计算加速的目的。由于CUDA具有不需要借助图形学API(Direct3D、OpenGL)函数,只需在C语言的基础上结合并行思想和GPU架构方面的知识就可以直接开发的优点,在图形图像处理[2]、音视频编解码[3]以及机械产品设计过程中涉及到的大型计算和实时显示中均有应用,计算速度可以提高两个数量级。

基于CUDA平台,实现复杂网格模型仿射变换的

收稿日期:2010-12-14

基金项目:国家部委科研基金项目(40401010105)

作者简介:钟庆(1986-),女(满),辽宁大连人,硕士研究生,主要研究方向为计算机图形处理.

加速计算。首先计算网格模型中每个顶点的k邻域点权值，然后根据变换矩阵执行顶点坐标变换。在并行计算分析的基础上，优化了线程结构和内存分配等，最大化发挥GPU并行处理能力。经过与CPU的计算时间测试对比，验证了并行计算的加速性。

1 网格模型仿射变换算法描述

复杂三维网格模型变形设计、操作处理中，需要对顶点进行仿射变换计算。网格模型仿射变换是一个将模型顶点根据各自的变换矩阵进行实时位置更新的过程，它不仅可以对模型进行整体的平移、旋转或缩放[4]，同时应能对复杂网格模型的任意曲面做局部变形。不失一般性，顶点的仿射变换由两部分组成：旋转矩阵$\boldsymbol{R}=\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$和平移矩阵$\boldsymbol{T}=[t_1, t_2, t_3]$。为了使变换效果连贯、平滑，每个顶点变换处理时不仅要考虑自身的变换矩阵，也要考虑其余所有顶点对其的影响。三维网格模型顶点集合P中，当前处理顶点p_i可根据模型所有顶点的变换矩阵进行仿射变换得到新位置$\widetilde{p}_i$，计算公式描述如下

$$\widetilde{p}_i=\sum_{j\in v}\left[\boldsymbol{R}_j(p_i-p_j)+\boldsymbol{T}_j+p_j\right] \tag{1}$$

其中，v是模型顶点数；$\boldsymbol{R}_j$和$\boldsymbol{T}_j$分别是顶点p_j的旋转矩阵和平移矩阵。$\boldsymbol{R}_j$和$\boldsymbol{T}_j$的求解需考虑以下3方面因素：

(1)用户指定某顶点为固定点或移动点时，这些顶点应固定不动(固定点)，或跟随鼠标移动到相应位置(移动点)；

(2)指定点(固定点或移动点)之间的顶点应作相应的移动，以保证曲面变形平滑；

(3)指定点(固定点或移动点)之间的顶点位置在随着移动点变化时，应尽量保持局部形状细节，所以应约束其仿射变换的$\boldsymbol{R}_j$为纯旋转变换。

综合上述3项因素，可建立一个非线性目标函数，通过优化迭代来求解顶点的最优旋转和平移矩阵。这里只重点介绍顶点仿射变换计算的GPU加速，有关$\boldsymbol{R}_j$、$\boldsymbol{T}_j$求解具体算法参见文献[5]。

由于复杂网格模型顶点众多，每一个顶点的变换过程都遍历其他所有顶点，导致计算过程复杂、耗时。通过对计算速度和变换效果的权衡，选择采用k邻域法，即只选取离该顶点最近的k个顶点的加权和进行计算，则式(1)改写为

$$\widetilde{p}_i=\sum_{j\in k}w_j\left[R_j(p_i-p_j)+\boldsymbol{T}_j+p_j\right] \tag{2}$$

其中，w_j是第j个邻域点p_j的权值，由下式确定

$$w_j=(1-\left\|p_i-p_j\right\|/d_{\max})^2 \tag{3}$$

其中，$d_{\max}$是顶点p_i与离它最近顶点排序中第$(k+1)$个点的欧式距离。

使用k邻域法既能在一定程度上降低计算复杂度，又能保证变换效果的连贯性。并且该仿射变换，只需要输入模型的顶点坐标即可进行，因此对任意以顶点信息给出的三维形体均适用。

2 网格模型仿射变换的并行架构设计

通过对复杂网格模型仿射变换计算的分析可知，在串行代码中，程序每次执行一个(或者两个)顶点的仿射变换，由于顶点数量大，因此循环次数多，计算速度慢。然而对于一个顶点，它的k邻域点权重值计算和坐标变换都只利用顶点的原始坐标和预处理中得到的旋转矩阵数据，并不依赖于其他顶点仿射变换后的坐标值，也就是说，顶点间的仿射变换过程相互独立。于是利用GPU并行模式，安排每一个线程单独处理一个顶点的变换过程，多个线程同时进行，这样计算时间减少，达到加速的目的。具体的算法流程如图1所示。

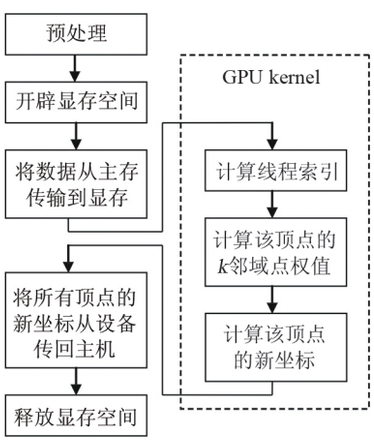


图1 复杂网格模型仿射变换的并行算法流程

针对上述并行计算，采取以下方式优化：

(1)在数据传输方面，主机与设备之间的数据传输带宽远小于显存带宽[6]，因此数据传输在整个计算

过程中占用大量时间,为尽量避免这种时间耗费,采用整体传输方法,即在启动kernel之前,一次性将所要用到的数据(顶点坐标、索引和变换矩阵以及顶点k邻域索引值)传入显存,保证主机与设备通信之间进行尽量多的计算,减少往复通信。最后只将用于结果显示的模型顶点坐标传回主存,其余数据均不传输。

(2)GPU上实际的算术逻辑单元(ALU)数量有限,但是却能维护大量的工作线程,这种模式用于保证有足够的线程随时填补主存访问和其他操作的延时,使所有计算单元满载,获得更高的运算效率[7]。对于模型中的每一个顶点,都需要根据k个邻域点的变换矩阵进行计算,所以采用每一个线程对应处理一个顶点的方式,每个块中采用最多的线程数512(计算能力1.3),块数根据处理的顶点数具体定义为(顶点数/512+1)。由于顶点数不一定恰好是512的整数倍,所以限定只计算线程地址小于顶点数的部分,避免访问越界。

(3)设备存储器有不同的模型,因文中的数据是需要频繁访问的只读参数,并且不是顺序读取,所以采用常量存储器来存储。常量存储器中的数据位于设备存储器,虽然没有共享存储器读取速度快,但是由于每个多处理器有一个常量缓存以加速常量存储器空间的读操作,因此速度比全局存储器快。文中的具体线程结构安排及存储器使用模式如图2所示。

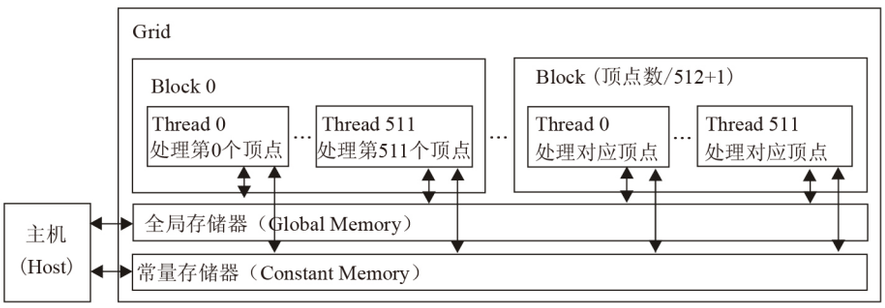
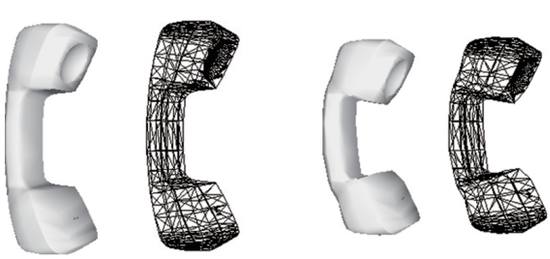


图2　线程安排及存储器使用模式

3　实验与分析

文中的实验平台为:CPU采用Intel Core2 E7400,该处理器主频为2.8 GHz,内存为2 GB,GPU采用NVIDIA GTX260,时钟频率为1.24 GHz,显存为896 MB,CUDA采用2.3版本。在CUDA上实现了不规则网格模型的快速仿射变换。

图3是电话听筒变形的实例,图3中当话柄变短、弯曲弧度变大时,听筒两端顺应变化趋势实现光滑、自然的变形。



(a)原始模型及网格表示　　(b)变形模型及网格表示

图3　电话听筒模型仿射变换效果

为验证GPU并行计算的加速性能,现以3个具体复杂网格模型为例,分别进行了计算时间测量结果的对比实验,数据如表1。

表1　网格模型仿射变换的CPU计算时间与GPU计算时间对比

不规则网格模型	顶点数	k值	CPU时间/ms	GPU时间/ms	加速比
1	7 207	1	1.024	0.266	4.526
		3	2.859	0.328	8.716
		7	6.078	0.547	11.112
2	1 2306	1	2.141	0.385	5.561
		3	4.766	0.483	9.867
		7	10.688	0.899	11.889
3	2 1887	1	4.782	0.578	8.273
		3	11.391	0.812	14.028
		7	19.016	1.562	12.174

以上GPU的计算包括数据传输和kernel计算两部分。由于GPU计算不稳定,每次计算时间差异较大,因此采取计算1 000次取平均值的方式来保证测量的准确性。

通过表1可以看出,对于单一模型来说,GPU的计算时间明显短于CPU,加速比在k邻域取值3的

模型3中达到了14.028。由模型间对比得出：相同k值下，由于模型3的顶点数目是模型1顶点数目的3倍，GPU加速效果更好；随着k邻域值的增大，求解权值矩阵的计算复杂度增高，GPU与CPU的计算加速比也随之增长，体现了GPU对大规模数据快速并行运算的优势。一般来说，GPU的计算高速性必须在1 000个以上的线程同时执行时才能发挥，计算量越大，加速效果越明显。表1中模型2为图3电话听筒模型的数据。

GPU对大规模数据计算时才有明显的加速优势，当计算规模较小时，其优势没得到充分发挥。一是因为数据量小时CPU本身的计算速度也很快；二是在GPU上消耗的大部分时间被主机与设备间的数据通信过程所占用。图4以模型1为例，将数据传输时间和kernel计算时间进行了对比测试。

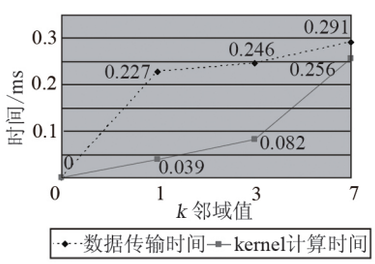


图4 模型1的kernel计算时间与数据传输时间对比

由图4可以看出，数据传输时间明显高于kernel计算时间，在k值取1的模型仿射变换中，数据传输部分的时间达到了kernel计算时间的5.8倍。随着数据量的增加，虽然数据传输时间与kernel计算时间的比例有所下降，但是数据传输部分仍然占用了大量的整体计算时间。因此，利用GPU对大规模数据进行并行计算，能更好地体现计算加速的优势。

4 结 论

针对复杂网格模型仿射变换的特点，利用GPU强大的并行计算能力，实现了该算法的明显加速。对于机械产品设计中涉及到的大规模数据计算部分，在数据相关性不大的前提下，可以转移到GPU上完成，能够在一定数量级上提高计算速度，节约计算成本，得到更快更好的处理效果。下一步工作是在复杂网格模型变形中，利用GPU并行计算求解变换矩阵优化迭代，以整体加速三维模型的变换操作，实现三维复杂模型的实时渲染绘制。

参考文献

[1] 张舒, 褚艳利. GPU高性能运算之CUDA[M]. 北京：中国水利水电出版社, 2009.

[2] Danny Crookes, Kevin Boyle, Paul Miller, et al. GPU Implementation of the Affine Transform for 3D Image Registration[C]//13th International Machine Vision and Image Processing Conference, 2009.

[3] NVIDIA. CUDA Zone, CUDA In Action. http://www.nvidia.com/cuda_in_action.

[4] 覃方涛, 房斌. CUDA并行技术与数字图像几何变换[J]. 计算机系统应用, 2010, 19(10):168–172.

[5] Sumner R W, Schmid J, Pauly M. Embedded deformation for shape manipulation[J]. ACM Transactions on Graphics 2007, 26(3), 80.

[6] NVIDIA. NVIDIA CUDA Compute Unified Device Architecture Programming Guide. <http://www.nvidia.com/cuda>.

[7] 肖汉. 利用GPU计算的双线性插值并行算法[J]. 小型微型计算机系统，2010，32(11): 2241–2245.

(上接第58页)

大学出版社，2005.

[3] 姜涌，曹杰，杜亚玲，等. 基于形态学梯度矢量的图像边缘提取算法[J]. 南京航空航天大学学报，2005，37(6)：771–775.

[4] 陶洪久，柳键，田金文. 基于小波变换和数学形态学的遥感图像边缘检测[J]. 红外与激光工程，2002，31(2)：154–157.

[5] 陈凡武. 彩色图像边缘检测的新方法——广义模糊算子法[J]. 中国科学(A辑)，1995，25(2)：219–220.

[6] Pietro Malik. Scale-Space and Edge Defection Using Anisotropic Diffusion[J]. IEEE，1990，12(7)：98–100.

[7] Chars Minghua，Lee David. Residual analysis for Feature DeCectipn[J]. IEEE Trans on Patt and Mmach Intelligence，1991，13(1)：30–35.

[8] 饶海涛，翁桂荣. 基于数学形态学的图像边缘检测[J]. 苏州大学学报(自然科学版)，2004，20(2)：42–45.

[9] 刘直芳，游志胜，曹刚，等. 基于多尺度彩色形态矢量算子的边缘检测[J]. 中国图像图形学报，2002，7(9)(A版)：888–893.

[10] 崔屹. 图像处理与分析——数学形态学方法与运用[M]. 北京：科学出版社，2000.

[11] 冯桂，桂预风，林宗坚. 灰度边缘检测中的形态学方法[J]. 理论研究，2000，5(4)：12–14.