基于 GPU 的 HASM 动态模拟与实时渲染方法

闫长青[1,2],岳天祥[1]

(1. 中国科学院地理科学与资源研究所,资源与环境信息系统国家重点实验室,北京 100101;
2. 山东科技大学信息工程系,泰安 271019)

摘要: 基于微分几何曲面论的高精度曲面模拟(high accuracy surface modeling, HASM)需要大量的复杂密集计算,在 CPU 上模拟极为耗时,使得在现有的硬件条件下,实时动态模拟曲面并实时可视化极具挑战性。论文提出了 GPU 加速的 HASM 方法,充分利用现代显示适配器(graphic processor unit, GPU)技术,运用 GPU 最新发展起来的并行计算能力,使用并行化的预处理共轭梯度方法解算曲面,完成曲面模拟,并同时利用 GPU 的高速缓存架构,对渲染操作进行充分优化,以实现高效实时可视化。HASM 需要的有限差分离散和高速解算操作,均充分利用现代 GPU 架构,所具有的多处理器和众多的流处理器所产生的强大并行计算能力,可视化也用 GPU 高速缓存技术和三角条带方法进行充分优化。数值实验和实际项目区高程模拟实验均表明,在 GPU 为 NVIDIA quadro 2000 和 CPU 为 DualCore Intel Core 2 Duo E8400 的硬件配置下,GPU 并行化的曲面模拟方法比普通方法速度提高了约 10 倍,使得动态模拟与可视化算法可以达到交互式的帧速及实时可视化的要求。

关键词: HASM;GPU 并行;实时可视化;CUDA
DOI: 10.3724/SP.J.1047.2012.00149

1 引言

曲面建模是通过采样点集合、线集合或面域数据集合模拟曲面的过程,其最终目的是形成一个网格对象,其中每个网格单元赋以一个相应的估计值[1]。自 1950 年以来,研究人员发展了各种曲面建模方法,其中较为经典的方法包括不规则格网的线性插值方法、样条函数插值方法、克里金方法和反距离加权方法。但是由于缺乏完备的理论基础,实际应用于地理信息系统中的地理信息曲面模拟时,导致产生严重的误差问题使结果不可用,失去了模拟意义。为了从根本上解决曲面模拟的误差问题,以微分几何曲面论[2]为基础,我们发展了高精度曲面建模(HASM)方法[1]。该方法与其他经典方法相比,HASM 具有较高的精度,但由于该方法需要求解大规模线性方程组,导致计算速度较慢,使得在三维空间中的动态模拟和实时可视化无法满足实时要求。而现代 GPU 技术的发展为这一极具挑战性的难题提供了解决方案。

近年来,在三维仿真游戏及真实感图形学市场的推动下,可编程的 GPU 逐渐从以图形处理为主演化为高度并行、有较强计算能力和较高带宽、多核多线程的处理器,并可用于通用计算任务。而传统意义上的 GPU 主要用于图形光栅化,完成从矢量图形到光栅图形的转换。尽管随着计算能力的不断增强,可以执行部分通用计算任务,但由于需要对 GPU 编程控制,而针对传统的图形操作流程管线和专用硬件编程学习较困难,极大地制约了其在通用计算方面的应用。而 2006 年 11 月英伟达公司发布了针对 GPU 编程的开发工具包——统一计算设备架构(compute unified device architecture, CUDA)以来,改变了这一状况,使得 GPU 的编程方式发生了根本性的变化,用户可以用多种简单灵活的编程方式,对 GPU 编程以利用其通用计算能力,如 CUDA C 或 C++、OpenCL、DirectCompute 和 CUDA Fortrain 等。而 GPU 技术的不断发展使得现代 GPU 已经具有强大的计算能力、新型统一架构、灵活的可编程性和不断提升的性

收稿日期: 2011-10-20; 修回日期: 2012-02-20.
基金项目: 国家杰出青年科学基金项目(40825003);国家高技术研究发展计划" 863 计划" 项目(2011AA120306);国家自然科学基金项目(41023010)。
作者简介: 闫长青(1978-),男,博士研究生,讲师,主要研究方向为资源环境模型与系统模拟。Email: yancq@lreis.ac.cn

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

能,使得GPU通用计算编程在科研领域备受欢迎。研究人员已使用GPU开展了大量通用计算的工作[3-5],研究结果表明通过GPU对计算密集型的任务进行并行化处理可极大地提高计算性能。

GPU通用计算的诸多领域均得到开展。科学计算方面有大量的研究工作使用了CUDA进行性能提升[6-10],研究结果均表明,使用GPU能获得明显的加速。有关三维物体实时自由变形,在GPU上使用CUDA加速的研究[11-12]也已开展。另外,使用CUDA在GPU上对流体模拟和渲染优化进行了大量研究工作[4,5,8,13-16],其研究结果表明,流体的模拟和可视化可完全在GPU上完成。所有工作都表明,运用现代GPU并行技术,在大量涉及复杂计算的领域都能使算法性能有明显的改善,能取得明显的加速效果。近年来中国学者也进行了此类研究[17-19],但针对使用CUDA进行曲面建模并实时渲染的研究工作仍然较少。

由于曲面建模需要大量复杂的密集计算,耗时较大,本研究尝试引入GPU的通用计算技术进行改进。利用GPU提供的通用计算能力,使用CUDA对高精度曲面建模方法并行化并在GPU上执行,并利用现代GPU最新高效渲染技术尤其是高性能存储技术对渲染性能进行优化,实现高效渲染,最终实现动态模拟曲面,实时可视化模拟结果。

在本文算法中,使用共轭梯度(CG)和预处理共轭梯度(PCG)方法,对高精度曲面建模所产生的大规模线性方程组[1]迭代求解,并与高效的渲染方法相结合实现高效可视化。我们使用CUDA对PCG方法进行了并行化,设置了模拟的最大迭代次数。每次迭代完成后,所得结果被发送至高效渲染进程。通过使用OpenGL-CUDA互操作[20],模拟和渲染过程紧密结合,取得了最佳性能。OpenGL-CUDA互操作通过顶点缓冲对象(vertex buffer object, VBO)实现,CUDA和OpenGL互斥的实现[21]对VBO的访问,并发的完成模拟和实时可视化。VBO是GPU的高性能存储器,使用它可以极大地提高访问速度,减少通信及访问延迟,从而大大改善渲染性能。另外,还使用了纹理缓存对象(texture buffer object, TBO)存储动态产生的纹理以进一步提高渲染效率。加速算法实现曲面模拟和可视化全部在GPU上执行,避免了CPU/GPU转移数据所带来的系统负荷,在很大程度上解决了通信的瓶颈问题。由于运用了现代GPU技术,我们的方法可以快速产生模拟结果,与高效的图形渲染管线相结合,可以产生交互式高质量的渲染效果。

本章内容组织如下:第二部分对GPU并行计算和并行化的HASM方法进行了简要介绍,并详细介绍了使用OpenGL-CUDA互操作实现曲面模拟与GPU优化的可视化方法并发执行和算法的实现。第三部分详细介绍了实验结果及对结果的分析。最后一部分给出了结论。

2 GPU并行的HASM和实时渲染方法

本文提出了GPU加速的HASM动态模拟和实时可视化方法。运用GPU通用计算能力对并行化的HASM方法进行加速模拟,模拟运用PCG方法迭代进行。每次迭代完成即由GPU通过OPENGL进行可视化,动态模拟和可视化通过GPU高速缓存实现交互。可视化方法运用GPU高速缓存并经过充分优化,保证了较高效率。随着模拟进行,若模拟精度达到要求,则只执行高性能可视化算法。因为模拟结果在精度控制下不断发生渐进变化,称之为动态模拟。

2.1 模型与方法

随着GPU技术的不断进步,GPU被越来越多的应用于通用计算任务。CUDA是一个可用于对GPU并行编程实现实现通用计算任务的并行计算架构,具有一个并行编程模型和一套指令集合,它可以提供对图形管线高效透明的访问。CUDA的硬件模型有一个可扩展的多线程的多流处理器阵列,根据计算能力的不同,可以包含8,32或48个流处理器,并且有四种集成在显示芯片上的存储器,包括寄存器、共享存储器、只读常量存储器和只读纹理存储器。CUDA的层次存储架构中还有全局存储器又称设备存储器,存在于GPU的显存之中,该存储器不存在高速缓存,有较高延迟[20]。可以通过同时运行大量的线程减少或隐藏延迟。

另外,CUDA通过定义和使用内核函数(kernel)对C语言进行了扩展(内核函数经调用在GPU上执行),从而提供了可以用C语言进行编程开发的环境,使得用户可以容易地开发CUDA程序。一个CUDA程序由在CPU上执行的主机程序和

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

由主机程序启动的一套内核函数组成,内核函数在称之为网格的单元上执行,每个网格由一个或多个块组成,每个块含有相同的数量的线程。内核函数的线程成组执行,每一组称为一个包(wrap)。Wrap 是一个执行单位,同一个 wrap 中的线程可以在流处理器(SPs)上并发执行,而同一流多处理器(SM)上的 SPs 执行相同的指令。在一个 kernel 中,寄存器通常用来存储被频繁访问线程的私有变量,共享存储器被分配至每一线程块(block),而全局存储器可以供所有的线程访问[21]。

为了最大程度地提高算法的执行效率,充分利用 GPU 的通用计算能力,必须统筹安排各类存储器的使用以便最大限度地增大系统吞吐量减少延迟。

为了对数值计算有较好的支持,CUDA 环境中还包含了两个高级数学库 CUBLAS[22] 和 CUSPARSE[23]。二者均用 CUDA 实现,前者是 BLAS 数学库的并行实现,后者是稀疏矩阵操作的并行实现。我们可以方便地使用这两个数学库实现数值计算的 GPU 并行编程。

高精度曲面建模[1,24]基于微分几何曲面论[2]提出[24]。根据文献[1],HASM 模拟需要先对 HASM 主方程组[1]:

$$\begin{cases} f_{xx} = \Gamma_{11}^1 \cdot f_x + \Gamma_{11}^2 \cdot f_y + L \cdot (E \cdot G - F^2)^{-1/2} \\ f_{yy} = \Gamma_{22}^1 \cdot f_x + \Gamma_{22}^2 \cdot f_y + N \cdot (E \cdot G - F^2)^{-1/2} \end{cases} \tag{1}$$

进行差分离散[25]得到:

$$\begin{cases} (f_{1+i,j}^{(n+1)} - 2f_{i,j}^{(n+1)} + f_{i-1,j}^{(n+1)}) \cdot h^{-2} = \\ (2h)^{-1} \cdot ((\Gamma_{11}^1)_{i,j}^{(n)} \cdot (f_{i+1,j}^{(n)} - f_{i-1,j}^{(n)}) + (\Gamma_{11}^2)_{i,j}^{n} \cdot \\ \quad (f_{i,j+1}^{(n)} - f_{i,j-1}^{(n)})) + L_{i,j}^{(n)} / \sqrt{E_{i,j}^n + G_{i,j}^n - 1} \\ (f_{i,j+1}^{(n+1)} - 2f_{i,j}^{(n+1)} + f_{i,j-1}^{(n+1)}) \cdot h^{-2} = \\ (2h)^{-1} \cdot ((\Gamma_{22}^1)_{i,j}^{(n)} \cdot (f_{i+1,j}^{(n)} - f_{i-1,j}^{(n)}) + (\Gamma_{22}^2)_{i,j}^{n} \cdot \\ \quad (f_{i,j+1}^{(n)} - f_{i,j-1}^{(n)})) + \\ N_{i,j}^{(n)} / \sqrt{E_{i,j}^n + G_{i,j}^n - 1} \end{cases} \tag{2}$$

其中,f 表示曲面表达式 $z = f(x,y)$ E,F,G,L,M,N 表达形式如下:

$$E = 1 + f_x^2; F = f_x \cdot f_y; G = 1 + f_y^2;$$

$$L = f_{xx} / \sqrt{1 + f_x^2 + f_y^2};$$

$$M = f_{xy} / \sqrt{1 + f_x^2 + f_y^2}; N = f_{yy} / \sqrt{1 + f_x^2 + f_y^2}$$

f_x 和 f_y 分别为:$f(x,y)$ 对 x 和 y 的一阶偏导;f_{xx} 和 f_{yy} 为 $f(x,y)$ 分别对 x 和 y 的二阶偏导;f_{xy} 为混合二阶偏导。$\Gamma_{11}^1, \Gamma_{11}^2, \Gamma_{22}^1, \Gamma_{22}^2$ 如下:

$$\Gamma_{11}^1 = 1/2 \cdot (G \cdot E_x - 2F \cdot F_x + F \cdot E_y) \cdot (E \cdot G - F^2)^{-1}$$

$$\Gamma_{11}^2 = 1/2 \cdot (2E \cdot F_x - E \cdot F_y + F \cdot E_x) \cdot (E \cdot G - F^2)^{-1}$$

$$\Gamma_{22}^1 = 1/2 \cdot (2G \cdot F_y - G \cdot G_x + F \cdot G_y) \cdot (E \cdot G - F^2)^{-1}$$

$$\Gamma_{11}^1 = 1/2 \cdot (E \cdot G_y - 2F \cdot F_y + F \cdot G_x) \cdot (E \cdot G - F^2)^{-1}$$

其中,$E_x, F_x, G_x, E_y, F_y, G_y$ 分别为 E,F,G 的一阶偏导。(2)式中相应符号表示(1)中对应符号在网格点(x_i, y_j) 第 n 次迭代得到的结果[26];h 表示模拟步长;假定 $M+2$ 为沿 x 或 y 方向的网格数目,$n >= 0$;$f_{0,j}^{(0)}, f_{i,0}^0, f_{M+1,j}^0, f_{i,M}^0$ 边界条件。

假设$\{\overline{f}_{i,j}\}$是 $z = f(x,y)$ 在计算区域中第 k 个采样点$\{(x_i, y_j)\}$处的采样值,在此网格点处模拟值应等于或者接近采样值。把此约束加到方程组(2)并改写成矩阵形式[1,24],我们可以得到带约束的最小二乘估计如下:

$$\begin{cases} \min \| \begin{bmatrix} A \\ B \end{bmatrix} \cdot X^{(n+1)} - \begin{bmatrix} D^{(n)} \\ E^{(n)} \end{bmatrix} \| \\ s.t. \qquad S \cdot X^{(n+1)} = K \end{cases} \tag{3}$$

其中,$S(k,(i-1) \cdot M + j) = 1, K(k) = \overline{f}_{i,j}$

$X^{n+1} = (f_{1,1}^{(n+1)}, \cdots, f_{1,M}^{(n+1)}, f_{2,1}^{(n+1)}, \cdots, f_{2,M}^{(n+1)}, \cdots, f_{M,1}^{(n+1)}, \cdots, f_{M,M}^{(n+1)})$;$A$ 和 B 分别表示方程组(2)的两个系数矩阵;D^n 和 E^n 分别为两矩阵方程的右端项。

对足够大的 λ,表达式(6)可转换为无约束的最小二乘估计:

$$\begin{bmatrix} A^T & B^T & \lambda \cdot S^T \end{bmatrix} \begin{bmatrix} A \\ B \\ \lambda \cdot S \end{bmatrix} X^{(n+1)} = \begin{bmatrix} A^T & B^T & \lambda \cdot S^T \end{bmatrix} \begin{bmatrix} D^{(n)} \\ E^{(n)} \\ \lambda \cdot K \end{bmatrix} \tag{4}$$

令 $F = \begin{bmatrix} A^T & B^T & \lambda \cdot S^T \end{bmatrix} \begin{bmatrix} A \\ B \\ \lambda \cdot S \end{bmatrix}$,

$T^{(n)} = \begin{bmatrix} A^T & B^T & \lambda \cdot S^T \end{bmatrix} \begin{bmatrix} D^{(n)} \\ E^{(n)} \\ \lambda \cdot K \end{bmatrix}$,表达式(4)可表示为:

$$FX^{(n+1)} = T^{(n)} \tag{5}$$

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

2.2 并行化的高精度曲面建模

共轭梯度方法(CG)是求解对称正定的大规模方程组的最著名方法之一,进行合适的预处理(即乘以一个预处理矩阵(preconditioner)),构造相应的预处理共轭梯度(PCG)方法,可以大大提高方法的性能。预处理共轭梯度方法的效率和健壮性已在广泛的应用中得到证明。高精度曲面建模的方程组(5)的系数矩阵是对称正定的,用 PCG 方法易于求解。若并行化 PCG,求解时会获得较高的效率[7]。

对于方程组(5),假定右端项 $T^{(n)}$ 已知,用 PCG 求解的算法可描述如下:

```
R0:=Tn−FXn;
M:= (diag (F))−1;
Z0:=MR0;
P0:=Z0;
k:=0;
While (k<=maxIteraNum &&){/ ‖ rk ‖ ‖ r0 ‖ −1>ε){
qk=Fpk;
αk=zkTrk/pkTqk;
xk+1=xk+αkpk;
rk+1=rk+αkqk;
zk+1=Mrk+1;
βk=zk+1Trk+1/zkTrk;
pk+1=rk+1+βkpk;
k:=k+1;
}
}
```

图 1 用 PCG 求解 HASM 算法

Fig. 1 The algorithm for HASM using PCG

从图 1 看出,预处理矩阵 M 在每次 while 循环即每次迭代都要应用于线性方程组,算法中对于预处理矩阵 M 的选取应使 M 对应线性方程组的求解代价小且易于并行化实现,M 选取通常是考虑其对 PCG 求解的收敛加速和 M 所对应的线性系统计算代价之间的一个折衷。预处理矩阵的选取有多种方式,其中,Jacobi 预处理是最简单的一种,它采用系数矩阵的对角线矩阵作为预处理矩阵,可转换为向量乘处理,易于 GPU 并行化,因而在我们的算法中,选取 $M=(diag(F))^{-1}$,$diag(F)$ 表示提取矩阵 F 的对角矩阵。

分析图 1 算法,算法单次迭代需执行两次内积操作,三次向量加减操作,一次稀疏矩阵与向量的乘积操作,一次预处理矩阵的逆与向量的乘积操作。其中,稀疏矩阵与向量的乘积操作最为耗时,其次为向量的内积操作。而 GPU 高性能是通过同时启动大量的线程共同完成该任务,这些大量的线程以线程块的形式组织,这些线程块组成一个或多个线程网格。其组织方式为单指令多数据,即多个线程对多个数据执行相同的操作。为了达到加速的效果,算法需要对前述操作进行并行化,划分成相对独立的任务,以映射到每个线程执行。对于向量的内积操作,可用如图 2 归约方式完成:

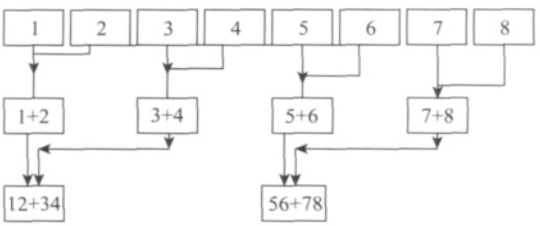

图 2 归约方法

Fig. 2 The reduction

而对于稀疏矩阵和向量的乘积操作,因为 HASM 左系数矩阵为对称正定非零元值有规律的稀疏矩阵,可采用一行稀疏矩阵对应一个线程的方式并行化方式处理,伪代码描述如图 3。

```
__global__void
sparseMV_kernel(int n, int * colIndsICP, int * rowPtrsICP,
float
* valsICP, const float * x, float * Ax)
{
int row = blockDim.x * blockIdx.x + threadIdx.x ;
if(row < n ){
int j;
Ax[i] = 0;
for(j = rowPtrsICP[row]; j <rowPtrsICP[row+1]; ++j)
    Ax[i] += valsICP[j]* x[colIndsICP[j]];
}
```

图 3 稀疏矩阵向量乘积并行算法

Fig. 3 Parallel algorithm of sparse matrix-vector products

图 3 算法采用了 CSR 存储稀疏矩阵,因为 HASM 的稀疏矩阵每行非零元最多 9 个且有规律分布,也可以不存储该矩阵,只存储对角线元素进行优化以节省存储空间,但对算法的速度未有明显的提高,本文为了与 CUDA 算法库结合使用,采用了 CSR 存储格式存储,未进行存储空间的进一步优化。

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

整个算法包含离散差分求矩阵 T,PCG 求解线性方程系统两大部分。运算前需将数据上载至 GPU,运算完后再复制回主机内存,整个并行化算法描述如图 4。

```
dim3 dimBlock(TILE_WIDTH, TILE_WIDTH);
dim3 dimGrid(Width/TILE_WIDTH, Length/TILE_WIDTH);
cudaMemcpy(d_OriginDem, d_B, sizeof(float) * (Width-2) *
(Length-2), cudaMemcpyHostToDevice);
kernel_T<<<dimGrid, dimBlock>>>(d_T, d_OriginDem, ,
Width, Length, cellsize, cellsize);
gpuTSample_Deal<<<1, elenumMat>>>(d_scatter_p, d_T,
Width-2);
kernel_PCG<<<dimGrid, dimBlock>>> (d_T , Length,
colIndsICP, rowPtrsICP, valsICP, d_x)
cudaMemcpy(x, d_x, sizeof(float) * (Width-2) * (Length-2),
cudaMemcpyDeviceToHost);
```

图 4　并行化的 HASM 算法

Fig. 4　Parallel algorithm of HASM

程序中我们首先设置包含线程块的规模和线程块中包含线程的规模,然后,上载数据至 GPU,调用 kernel_T 进行有限差分并行求算 T,再调用 kernel_PCG 求算线性系统,PCG 中对向量内积、稀疏矩阵向量积、向量的加减等用大量线程并行求算。由于我们的算法的计算效率取决于向量内积、矩阵向量运算和预处理线性系统的求解,故对这些运算的并行化处理是运用 GPU 实现并行化的关键。而 CUDA 也已在 CUBLAS[22] 中实现了向量内积操作。同样的,稀疏矩阵与向量的相关操作也可以用 CUSPARSE[23]。至于添加预处理的线性系统求解,由于采用了最简单的对角形式的预处理矩阵,容易用 CUDA 的 kernel 编程实现并行化。这样,我们可以用 CUBLA、CUSPARSE 和 CUDA kernel 函数 CUDA 灵活地实现对整个算法的最优并行化。

在运用图 1 中的算法求解方程组(5)时,必须先求出其右端项 $T^{(n)}$,我们通过对 X^n 进行有限差分对其求解。初值 X^0 用根据采样点用克里金插值的结果赋值。$T^{(n)}$ 的求算是完全并行的,易用 CUDA 并行化。结合图 1 的并行化,即可完成整个 HASM 算法在 GPU 上如图 4 所示的并行化。

2.3　高精度曲面建模的实时渲染

传统的渲染方法通常由 CPU 提取层次细节后,发送至 GPU 用 OpenGL 完成渲染。这种方法需要 CPU 和 GPU 大量通信,而二者的通信及 CPU 提取细节的低效性和 GPU 渲染的高效性往往会导致 GPU 花费大量的时间等待 CPU 完成层次细节提取并发送至 GPU 显存,是制约实时渲染性能的瓶颈。本算法中,由于高精度曲面建模过程是在 GPU 显存内进行,避免了 CPU 和 GPU 的通讯,因而采取了一种不同的渲染方法,完全在 GPU 上实施优化,完成渲染,提高了性能。

2.3.1　OpenGL 接口

OpenGL (Open graphics library)定义了一种跨平台的语言,是一种可以编写 2D 和 3D 计算机图形学程序的应用程序编程接口。该接口包括 250 不同的函数调用,程序员可以对其调用实现用简单图元绘制三维复杂场景的操作[27]。它可以隐藏程序员对硬件的操作,实现高效的图像图形处理。在本文研究中,使用了 OpenGL 3.0 的扩展版本,需要支持 OpenGL 3.0 或以上版本的 GPU。

2.3.2　高效渲染方法

渲染性能的提高主要取决于减沙渲染过程中 CPU 和 GPU 通讯时间和渲染操作调用的次数,使得渲染操作对 GPU 显存的访问尽可能高效。相比于以往 CPU 和 GPU 相结合的方法,我们利用 GPU 高速缓存和对渲染对象的建立三角条带索引提高渲染操作的执行效率。

我们使用 OpenGL 和 CUDA 的互操作实现 GPU 上的高精度曲面建模动态模拟结果的渲染。实现过程中的第一步是初始化 OpenGL 并加载根据采样点插值得到的曲面数据至顶点缓冲对象(VBO),插值方法采用克里金插值。VBO 概念上来说是一组字节数组,可以处理上载的顶点属性如顶点位置、法向、颜色和纹理坐标等[11],是集成在 GPU 芯片上的一种高效存储器,可以在很大程度上提高渲染速度,它从。初始化后运行我们的并行化的 HASM 算法执行模拟操作并进行渲染绘制,模拟和渲染在同一个渲染循环里边执行。每次迭代、模拟系根据上次迭代结果求算,存储在 VBO 中的结果使用 OpenGL 渲染。

为最大化渲染效率,算法中使用了 4 个 OpenGL 缓冲区。其中一个存储顶点位置,第二个存储动态纹理,第三个存储顶点索引(IBO),最后一个纹理坐标缓冲(TBO)存储纹理坐标。存储纹理的缓冲区(pixel buffer object, PBO),存储动态纹理,该纹理由 CUDA 动态产生,可以每一帧都发生改变。

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

动态产生的纹理，用 OpenGL 命令 glTexSubImage2D()转移至 PBO 进行纹理应用。使用 PBO，OpenGL 可以在 PBO 和纹理对象之间执行异步 DMA 传输，极大地提高纹理传输性能，而使用缓冲对象存储纹理坐标同样能提高绘制效率。另外，我们使用顶点数组进一步提高绘制几何图元时的性能，而使用顶点索引缓冲区能进一步提高对顶点数组访问的效率。

整个算法过程使用 OpenGL 和 CUDA 合作完成，二者公用缓冲区，以各自的命名方式进行访问。曲面模拟和纹理产生使用 CUDA 产生，CUDA 使用 VBOs 前，必须进行一次注册，使之成为 CUDA 资源。为访问 VBO 数据的具体位置，还需取得相应存储位置的指针。这通过建立 VBO 资源到 CUDA 地址空间的映射实现，映射操作的结果即返回相应数据的位置指针，可用于 CUDA kernel 函数。

模拟结果的显示通常涉及到对共享顶点的大量冗余的访问，会影响渲染效率。使用顶点数组可以有效地减少这些冗余访问，从而减少图元绘制的调用操作，提高渲染的性能。使用顶点数组，需先根据图元的绘制规则建立顶点的索引。本算法用三角网显示模拟曲面，使用三角条带表示方法可显著地提高渲染效率，减少基本图元绘制命令的调用频率，需建立整个顶点数组的三角条带索引。最后，调用 OpenGL 命令 glDrawElements () 使用 VBO，PBO 和 IBO 中的数据进行曲面绘制。索引建立描述如下：

```
// indexVarray 表示顶点存放位置，ndx 为索引数组
indexVarray=(GLuint *) malloc(mesh_width * 2 * (mesh_
height-1) * sizeof(GLuint));
for (int j=0; j<mesh_height-1; j++) {
  for (int i=0; i<mesh_width; i++) {
      GLuint *ndx = indexVarray + j*2*mesh_width + i
*2;
        ndx[0] = j*mesh_width + i;
        ndx[1] = (j+1)*mesh_width + i; }}
```

图 5 建立索引

Fig. 5 Index construction

渲染算法总结如下：

首先，对 OpenGL 和 CUDA 进行初始化。其次，建立 VBO，PBO，TBO 和 IBO 四个高性能缓冲区，上载初始插值结果至缓冲区作为初始化结果。再次，使用并行化的 HASM 算法进行模拟并并行产生动态纹理。最后，渲染纹理化的模拟结果。整个渲染方法优化立足于完全用 GPU 实现及绘图操作对 GPU 存储器的高效访问，因篇幅所限，未与其他渲染方法做比较。

在算法执行过程中，如果模拟结果达到我们的精度要求，算法仅执行可视化部分渲染模拟结果而不再执行算法模拟过程直至退出程序，算法过程可描述如图 6。

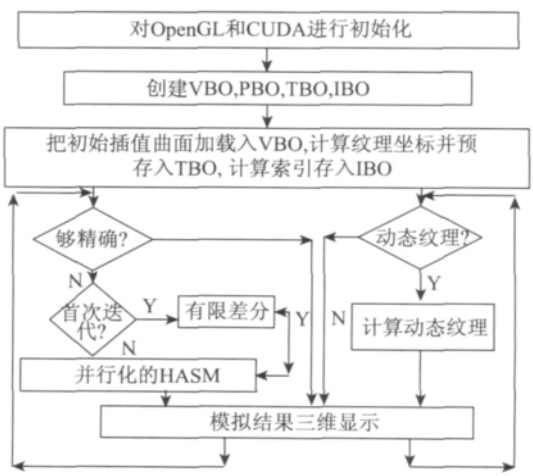

图 6 渲染过程

Fig. 6 The rendering process

2.4 算法的实现

我们使用 CUDA kernel、CUBLAS 和 CUSPARSE 实现了我们的并行化的 HASM 算法。渲染可视化过程采用 OpenGL 和 CUDA 互操作实现。另外，我们充分利用了现代 GPU 技术提高性能，整个算法完全基于 GPU 实现。算法在 NVIDIA quadro 2000 GPU 上执行，主机是 Dual-Core Intel Core 2 Duo E8400，操作系统为 windows xp，CUDA 4.0 和 OpenGL 4.0 版本。

在我们的算法中，系数矩阵采用了稀疏矩阵的存储模式，具体存储形式为 CSR (compressed sparse row)[23] 格式，以便使用 CUSPARSE 库编程。设置了最大迭代次数和误差阈值，还设置了整个算法的执行次数。

在绘制渲染结果之前，需要调用 CUDA 命令 cudaGraphicsUnmapResources()保持 CUDA 和其它使用相同缓冲区的图形应用程序部分同步[28]。

3 实验结果与分析

为比较 CPU 实现的 HASM 方法(HASM—

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

CPU)和并行化的 HASM 方法(HASM－GPU)的性能,我们进行了一系列的实验对算法进行了测试。第一个实验选取某地区气温数据的一部分,数据规模较小为 10×11,目的是测试在小规模数据下算法的运行情况。第二个实验选取高斯合成曲面做模拟对象,数据规模为 201×201,目的是检测算法对数学曲面的模拟情况。第三、第四和第五个实验选取了实际项目区中的三个部分规模分别为 970×835、1024×1125 和 1118×1130 数字高程曲面,以检测对于实际应用实验算法的执行情况。所有的实验均采用克里金方法根据采样点插值得到的结果作为初始值。

在两种方法中,我们选取相同的初始值 x_k 开始,并设置相同的退出误差阈值 $\|r_k\|\|r_0\|^{-1}<10^{-8}$,并设置最大迭代次数为 5000。测试结果如下:

表 1　HASM 的 CPU 方法与 并行化方法耗时对比

Tab. 1　HASM -CPU vs HASM-GPU: total time

网格规模	CPU 方法执行时间(s)	GPU 方法执行时间 (s)
10×11	0.518339	0.011270
201×201	10.390260	1.547023
970×835	465.156982	44.635750
1024×1125	662.741577	61.189999
1118×1130	721.392245	62.284002

表 2　HASM 的 CPU 方法与 并行化方法每次迭代耗时及比值对照

Tab. 2　HASM－CPU vs HASM-GPU: time per iteration

网格规模	CPU 方法执行时间(s)	GPU 方法执行时间(s)	二者耗费时间比
10×11	0.007004581	0.000154384	45.37128828
201×201	0.004166103	0.000633507	6.576258183
970×835	0.093031396	0.00892715	10.42117545
1024×1125	0.132548315	0.012238	10.83088066
1118×1130	0.144278	0.0124568	11.5823

在表 1、2 中,我们比较了 HASM-GPU 和 HASM-CPU 的运行情况。表中可看出,HASM-GPU 运行速度大约快 10 倍。当处理大规模数据时,当执行最大迭代次数时,即在迭代次数内,误差条件未满足,未自动退出的情况下,可以获得大约 10 倍的加速。从表 2 可见,HASM-CPU 速度太慢,每次迭代耗费太长时间,以致无法达到交互帧速。并且,对于合适规模的数据,执行并行化的 HASM 算法,进行实时渲染是可能的。

我们对表 1 中所有的实验执行了动态模拟和实时绘制渲染算法,并测量了最小帧速记录如下(表 3)。

表 3　HASM-GPU: 每帧耗时与最小帧速(即每秒的帧数)

Tab. 3　HASM-GPU: time per iteration vs minimum frame rates per second

网格规模	GPU 方法执行时间 (s)	帧速
10×11	0.000154384	79.9
201×201	0.000633507	79.8
970×835	0.00892715	39.6
1024×1125	0.012238	9.9
1118×1130	0.0124568	9.9

表 3 中,GPU 方法执行时间指 GPU 方法完成一次迭代模拟过程得到结果需要的时间,帧速指每秒可渲染多少副图像。我们的方法实时计算模拟结果并完成结果渲染,因而在相同的时间内交替完成了模拟和计算两个过程。从表 3 可见,如果数据规模非常小,由于其较低的迭代时间耗费,其最小帧速可达近 80 帧每秒。然而,随着数据规模的增加,迭代模拟时间增加,致使帧速减少。因而,如果处理大规模数据,为达到理想的帧速,需要把数据分成合适大小的块。另外,如果模拟过程完成,算法仅需执行渲染部分,帧速则会增加。

综上所述,对于一定规模的数据,并行化的 HASM 模拟算法可以达到 10 倍以上的加速比。因为 GPU 加速比的大小与 GPU 硬件的计算能力、算法的可并行程度数据的规模均有关系,如果采用专业的计算能力更强的 GPU 可以获得更高的加速比,如果处理大规模数据,数据的组织加载及处理的统筹安排也可以提高加速比。而对于整个模拟与可视化算法,要想达到理想的帧速,并且在可视化的过程中完成模拟计算,需要保证计算区域在一定合适的规模内,既要保证较高的加速比,又要保证合适的计算量以便有较多的时间供 GPU 做可视化操作提高帧速。

4　讨论

本文提出的曲面实时动态模拟并实时渲染可视化方法,克服了以往可视化需要曲面事先模拟耗费大量时间的问题。该方法的特点可达到 10 倍的

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

加速，在我们的高效渲染过程中模拟，可获得交互式的帧速。算法完全在利用现代 GPU 技术，避免了传统可视化方法 CPU 和 GPU 通信的瓶颈，取得了高效率。对于大规模数据，我们可以分成大小合适的块处理可达到交互式的帧速。对于可视化部分，我们还可以实施层次细节表示及基于视图体的裁剪操作进一步减少冗余处理规模，提高效率，这是我们未来工作的重点。

参考文献：

[1] Yue T X. Surface modeling: High accuracy and high speed methods[M]. Boca Raton, FL: CRC Press, 2011.

[2] Toponogov V A. Differential geometry of curves and surfaces: a concise guide[M]. New York: Birkhauser, 2006.

[3] Stone J E, Hardy D J, *et al*. GPU-accelerated molecular modeling coming of age[J]. Journal of Molecular Graphics and Modelling, 2010, 29(2): 116-125.

[4] Goswami P, Schlegel P, *et al*. Interactive SPH simulation and rendering on the GPU[C]. // Proceedings of the 2010 ACM SIGGRAPH/Eurographics symposium on Computer animation, 2010, In SCA '10: Eurographics Association.

[5] Kalyanapu A J, Shankar S, *et al*. Assessment of GPU computational enhancement to a 2D flood model[J]. Environmental Modelling & Software, 2011, 26(8): 1009-1016.

[6] Hawick K A and Playne D P. Numerical simulation of the complex Ginzburg-Landau equation on GPUs with CUDA[C]. // IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN), Innsbruck, Austria: ACTA Press, 2011.

[7] Helfenstein R and Koko J. Parallel preconditioned conjugate gradient algorithm on GPU[J]. Journal of Computational and Applied Mathematics, In Press, Corrected Proof, 2011.

[8] Lee H and Han S. Solving the shallow water equations using 2D SPH particles for interactive applications[J]. The Visual Computer, 2010, 26(6): 865-872.

[9] Ji X, Cheng T and Wang Q. CUDA-based solver for large-scale groundwater flow simulation[J]. Engineering with Computers, 2011, 28(1): 13-19.

[10] Manavski S A. CUDA compatible GPU as an efficient hardware accelerator for AES cryptography[C]. // IEEE Signal Processing and Communications(ICSPC), Dubai: IEEE. 2007.

[11] Jung Y, Graf H, *et al*. Mesh deformations in X3D via CUDA with freeform deformation lattices[M]. // Virtual and Mixed Reality-Systems and Applications, R. Shumaker, Editor Springer Berlin / Heidelberg. 2011, 343-351.

[12] Wei W and Huang Y. Real-time flame rendering with GPU and CUDA[J]. International Journal of Information Technology and Computer Science, 2011, 3(1): 40-46.

[13] Nilsson M. Constraint fluids on GPU (2009)[D]. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.160.2134&rep=rep1&type=pdf.

[14] Singh B, Pardyjak E R, *et al*. Accelerating urban fast response Lagrangian dispersion simulations using inexpensive graphics processor parallelism[J]. Environmental Modelling & Software, 2011, 26(6): 739-750.

[15] Neal J C, Fewtrell T J, *et al*. A comparison of three parallelisation methods for 2D flood inundation models [J]. Environmental Modelling & Software, 2010, 25(4): 398-411.

[16] Kalivarapu V and Winer E. A multi-fidelity software framework for interactive modeling of advective and diffusive contaminant transport in groundwater[J]. Environmental Modelling & Software, 2008, 23(12): 1370-1383.

[17] 刘二永，汪云甲. 基于 CUDA 的 IDW 并行算法及其实验分析[J]. 地球信息科学学报，2011，13(5)：707-710.

[18] 王强，檀结庆，胡敏. 基于有理样条的图像缩放算法[J]. 计算机辅助设计与图形学学，2007，19(10)：1348-1351.

[19] 桂叶晨，冯前进，刘磊，陈武凡，基于 CUDA 的双三次 B 样条缩放方法[J]. 计算机工程与应用，2009，(01)：183-185，194.

[20] NVIDIA. NVIDIA CUDA C Programming Guide[M]. Santa Clara, CA, USA: NVIDIA Corporation, v4 ed 2011.

[21] Kirk D B and Hwu W W. Programming massively parallel processors: A Hands-on approach[M]. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010.

[22] NVIDIA. CUBLAS Library[M]. Santa Clara, California, USA: NVIDIA Corporation, 2011.

[23] NVIDIA. CUSPARSE Library[M]. Santa Clara, Cal-

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

ifornia,USA: NVIDIA Corporation, 2011.

[24] Yue T X, Du Z P, *et al*. A new method of surface modeling and its application to DEM construction[J]. Geomorphology, 2007, 91(1-2): 161-172.

[25] Quarteroni A, Sacco R and Saleri F. Numerical mathematics[M]. New York: Springer, 2000.

[26] Yue T X, Song D J, *et al*. High-accuracy surface modelling and its application to DEM generation[J]. International Journal of Remote Sensing, 2010, 31(8): 2205-2226.

[27] Shreiner D. OpenGL programming guide: the official guide to learning OpenGL, versions 3.0 and 3.1[M]. Boston, MA, USA: Addison-Wesley Professional, 2009.

[28] Sanders J and Kandrot E. CUDA by example: an introduction to general-purpose GPU programming[M]. Ann Arbor, Michigan: Addison-Wesley Professional, 2010.

A Novel Method for Dynamic Modelling and Real-time Rendering Based on GPU

YAN Changqing[1,2] and YUE Tianxiang[1]

(1. *State Key Laboratory of Resources and Environment Information System, Institute of Geographic Sciences and Natural Resources Research, CAS, Beijing* 100101 *China*;

2. *Department of Information Engineering, Shandong University of Science and Technology, Taian* 271019, *China*)

Abstract: High accuracy surface modeling (HASM) is a method for building surface with high accuracy in terms of sampled points, which is based on the fundamental theorem of surfaces and gives a solution to error problem in geographic information system (GIS). Previous studies show this method can model surface with much higher accuracy than other classical methods widely used in GIS, while its speed is limited because of its huge computational cost. In order to accelerate its computational speed, a novel parallel and interactive method of HASM for real-time rendering on GPU using compute unified development architecture (CUDA) is presented, which allows for efficient and high quality visualization. The computational task of HASM is parallelized and run simultaneously on many cores of modern GPUs which have multiprocessors and many stream processors, which can improve the performance significantly. Coupled with an efficient rendering method, dynamic surface simulation and real-time rendering is done concurrently on GPU. Preconditioned conjugate gradients methods are used to solve the huge linear systems arising from HASM. Fully harnessing the processing power of modern GPUs with a highly parallel architecture and multiprocessors and many stream processors, we can simulate the surface dynamically and post it to rendering pipeline simultaneously. Making use of state-of-the-art GPU techniques such as vertex buffer object, texture buffer, the rendering can be carried out with a very high efficiency. A few experiments were carried out including some digital elevation model constructions and the tests results showed that our method can construct surface dynamically and visualize it at very high frame rates.

Key words: high accuracy surface modelling (HASM); GPU parallel; real-time visualization; CUDA

© 1994-2012 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net