ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.17, No.3, March 2006, pp.568−576
DOI: 10.1360/jos170568

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

基于 GPU 带有复杂边界的三维实时流体模拟*

柳有权[1,3+]，刘学慧[1]，吴恩华[2]

[1](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

[2](澳门大学 科学技术学院 电脑与资讯科学系,澳门)

[3](中国科学院 研究生院,北京 100049)

Real-Time 3D Fluid Simulation on GPU with Complex Obstacles

LIU You-Quan[1,3+], LIU Xue-Hui[1], WU En-Hua[2]

[1](Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macao, China)

[3](Graduate School, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: Phn: +86-10-62522028, Fax: +86-10-62563894, E-mail: lyq@ios.ac.cn, http://lcs.ios.ac.cn/~lyq/

Liu YQ, Liu XH, Wu EH. Real-Time 3D fluid simulation on GPU with complex obstacles. *Journal of Software*, 2006,17(3):568−576. http://www.jos.org.cn/1000-9825/17/568.htm

Abstract: This paper, solves the 3D fluid dynamics problem in a complex environment by taking advantage of the parallelism and programmability of GPU (graphics processing unit). In difference from other methods, innovation is made in two aspects. Firstly, more general boundary conditions could be processed on GPU in the method. By the method, the boundary from a 3D scene with capped solid clipping is generated, making the computation run on GPU despite of the complexity of the whole geometry scene. Then by grouping the voxels into different types according to their positions relative to the obstacles and locating the voxel that determines the value of the current voxel, the values on the boundaries are modified according to the boundary conditions. Secondly, more compact structure in data packing is designed at the fragment processing level to enhance parallelism and reduce execution passes. The scalar variables including density and temperature are packed into four channels of texels to accelerate the computation of 3D Navier-Stokes Equations. The test results show the efficiency of the method, and as a result, it is feasible to run middle-scale problems of 3D fluid dynamics in an interactive speed for more general environment with complex geometry on PC platform.

Key words: graphics hardware; GPU(graphics processing unit); programmability; Navier-Stokes equation; 3D fluid simulation; real-time

摘 要: 在 GPU(graphics processing unit)上求解了复杂场景中的三维流动问题,充分利用了 GPU 并行能力以加速计算.与前人的方法不同,该方法对于边界条件的处理更为通用.首先,通过在图像空间生成实心的剖切截面构

* Supported by the National Natural Science Foundation of China under Grant Nos.60223005, 60033010 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312102 (国家重点基础研究发展规划(973)); the Research Grant of University of Macau (澳门大学资助项目)

Received 2004-09-02; Accepted 2005-02-25

成整个障碍物信息图,算法使得流体计算与整个几何场景的复杂度无关,通过对各体素进行分类并结合边界条件,根据障碍物形成修正因子来修改对应的值;另外,采用更为紧凑的数据格式,以充分利用硬件的并行性.通过将所有标量的运算压缩到纹元的 4 个颜色通道并结合平铺三维纹理,减少了三维流场计算所需要的绘制次数.实验结果显示出算法的有效性和高效率.该算法可以实时计算并显示一个采用中等规模离散的复杂场景.

关键词: 图形硬件;GPU;可编程性;纳维-斯托克斯方程组;三维流动模拟;实时

中图法分类号: TP391 文献标识码: A

随着图形硬件的快速发展,如何利用硬件的可编程性和并行性来实现一些复杂计算,已成为当今的热点研究问题之一.尤其是在像素程序开始支持 32 位浮点运算和高级绘制语言不断普及之后,图形处理器(graphics processing unit,简称 GPU)逐步成为计算的主流[1],越来越多的研究者开始采用 GPU 来求解一些一般意义上的问题[2,3].

实时流体模拟是计算机图形学中的另一个热点问题.为了真实地描述流动现象,许多研究者不得不求解复杂的运动方程,如纳维-斯托克斯方程组(Navier-Stokes equations,简称 NSEs).然而,NSEs 的求解相当耗时,对时间步长极为敏感,易导致结果发散.直到几年前,Stam[4]在计算机图形学领域中引入半拉格朗日算法,才使得实时流体计算成为可能.该方法允许采用较大的时间步长,同时保持良好的稳定性.虽然这种方法的计算精度不能满足工程上的应用,但能捕获流体运动的基本特征,满足很好的视觉效果.所以,计算机图形学界逐渐开始广泛应用该方法来模拟流动现象[5−9].为了更进一步加速求解,人们开始利用 GPU 的并行性和可编程性来求解偏微分方程(partial differential equations,简称 PDEs)[10−13].但是,由于目前 GPU 上缺乏像 CPU 那样的灵活性,编程也不如在 CPU 上那样容易,所以大多数研究者只是集中在二维问题域.而且,对于边界条件的处理过于简单,以至于很难满足实际问题的需要.

本文的贡献在于:在 GPU 上采用半拉格朗日方法来求解三维 NSEs,且保证 GPU 能够处理任意复杂的边界条件,整个计算与场景的几何复杂度无关.对于中等规模的问题,整个模拟和显示能够实时进行,充分利用了 GPU 的并行性以加速计算.

为了能够处理任意障碍物形成的复杂边界,我们结合模板缓冲对三维场景进行剪切操作,形成一系列实心的剖切截面,以构成整个流动边界.我们将整个三维计算域离散成一组切片,并将这组切片平铺到一个二维空间,所以这组切面也被组织成一幅二维纹理.这样,通过将障碍物信息转化到图像空间,使得整个流体计算与场景几何复杂度无关.我们根据障碍物的位置信息将体素划分为不同的类型,并利用像素程序生成偏移坐标,然后在偏移坐标的基础上,根据边界条件形成边界修改因子,从而使得该方法能够处理任意的内部边界条件,比文献[11]的方法更为通用.另外,为了充分利用像素处理的并行性,我们将相似的变量组织在一起,压缩到纹元的 4 个颜色通道,以减少绘制的次数,达到性能提升的目的.而在绘制方面,由于流体为半透明的物质,所以本文采用光线投射的思路,直接在 GPU 上利用像素程序进行积分运算计算出光的衰减,然后通过基于 GPU 的颜色融合以获得着色效果.

1 相关工作

为了模拟烟雾中的湍流现象,文献[14]采用分解的方法将湍流风场分成确定分量和随机分量两部分.其中,确定分量用来模拟风场的大尺度行为,采用几种常见的风的形态进行叠加,而随机分量则采用 Kolmogorov 频谱来模拟小尺度行为.为了更精确地描述流体的运动,Foster 等人[15]采用直接数值模拟来求解流体方程,用以模拟水面形态以及漂浮物体的运动.但由于这种方法采用显式格式,要求时间步长很小,否则整个计算不收敛.为了在保证计算稳定的情况下能够获取较大的时间步长,Stam[4]采用半拉格朗日法来求解 NSEs.但半拉格朗日方法的数值耗散很严重,因此,在文献[6]中引入一个漩涡约束因子(vorticity confinement)以弥补耗散,从而得到很好的烟雾视觉效果.文献[5,7]采用该方法来求解 level-set 方程以模拟复杂流体表面.在模拟火焰的过程中,文献[9]对于气态燃料和气态生成物分别采用了类似的处理方法.同样,文献[8]中为了模拟大尺度烟的效果,在二维平面

上采用半拉格朗日法来求解 NSEs,三维上尺度上则只采用 Kolmogorov 频谱来增加细节.

随着 GPU 的发展,特别是可编程性和实时绘制语言的普及,很多人利用 GPU 提供的并行性,将它作为流处理器来做一些通用计算方面的工作,甚至用来求解有限差分方程组.2001 年,Rumpf 等人[16]利用图形硬件提供的多纹理操作和图像子集函数来求解传热和各向异性扩散有限单元方程,以实现图像处理的功能.2002 年,文献[17]采用 LBM(lattice Boltzmann method)来模拟流动效果.通过将粒子包合成纹理,将 Boltzmann 方程组映射到光栅化和帧缓冲操作上,采用 Register Combiner 实现整个计算.Harris 等人[18]通过 Register Combiner 结合 Texture Shader 来求解 CML(coupled map lattice)问题,从而实现交互的对流扩散模拟.

从 2003 年开始,Krüger 等人[13]利用像素编程,在基本代数运算的基础上实现了共轭梯度法和高斯-赛德尔迭代法,从而更进一步地求解 2D 波动方程和不可压 NSEs;Bolz 等人[10]则实现了基于像素编程的稀疏非结构化矩阵的共轭梯度法和多重网格法,以求解流体运动方程;Goodnight 等人[12]在 GPU 上应用多重网格算法求解边界值问题;Harris 等人[11]则基于像素程序对云彩的运动方程进行求解来得到动画效果;文献[19]给出了一个采用 Cg 实现的流体简单效果的模拟;Batty 等人[20]在 GPU 采用共轭梯度法来模拟流体运动.

2 基于 GPU 的三维流动模拟

为了模拟流动现象,计算流体力学领域(CFD)的众多研究者已经做了大量的工作.但在计算机图形学领域,目前主要是针对不可压的无粘低速流动现象模拟,以满足电影特效或者游戏特效的需要,如烟、火、云彩、流水等[5−9,11].这里也只是针对不可压的无粘低速流动现象而言.

2.1 流体运动方程

整个流体运动方程主要由两部分构成:一是连续性方程;另一个是动量方程.

$$\nabla \cdot \vec{u} = 0 \tag{1}$$

$$\partial \vec{u} / \partial t = -(\vec{u} \cdot \nabla)\vec{u} + \nu \nabla^2 \vec{u} - \nabla p + \vec{f} \tag{2}$$

这里,$\vec{u}$ 为速度矢量,ν 为动力粘性系数,$\vec{f}$ 为外部施加的作用力,p 为压强.

为了表现流动效果,引入密度ρ和温度 T 两个标量.本文不仅考虑其自身的扩散效果,也考虑计算速度场所带来的对流效应.

$$\partial \rho / \partial t = -(\vec{u} \cdot \nabla)\rho + k_\rho \nabla^2 \rho + S_\rho \tag{3}$$

$$\partial T / \partial t = -(\vec{u} \cdot \nabla)T + k_T \nabla^2 T + S_T \tag{4}$$

为了能快速模拟,我们采用半拉格朗日算法求解上述方程.更多关于半拉格朗日算法的细节可以参考文献[4].整个计算与我们在二维问题上的计算方法[28]类似.

由于温度和密度会导致速度发生变化,所以将其影响以浮力的形式[6,8,9,28]传递给速度场.

$$\vec{f}_{buoy} = -\alpha\rho\vec{y} + \beta(T - T_{amb})\vec{y} \tag{5}$$

这里,$\vec{y}$ 为向上的矢量方向,即(0,1,0),T_{amb} 为周围环境的温度,α和β分别用来控制密度和温度的影响程度.

为了密度半拉格朗日方法的数值耗散,我们引入漩涡约束因子来增加流动的细节,以外力的形式传递给速度场 $\vec{f}_{conf} = \varepsilon h(\vec{n} \times \vec{\omega})$.这里,$\vec{\omega} = \nabla \times \vec{u}$, $\vec{n} = \nabla |\vec{\omega}| / |\nabla |\vec{\omega}||$,ε用来控制加入到流场中细小尺度细节的量,h 为空间步长,以确保加入的漩涡约束是合理的.

2.2 计算纹理准备

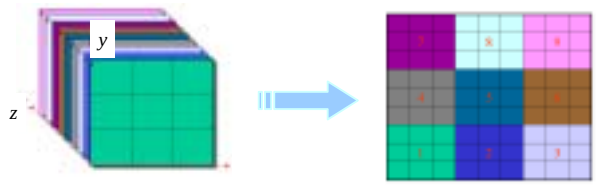
为了加速偏微分方程组的求解,我们将整个计算映射到 GPU 上来完成.文献[20]将拉普拉斯算子重新组织成 7 幅纹理.与其不同的是,我们直接将整个计算域组织成纹理[11],利用像素程序在 pbuffer 中计算整个 NSEs.

在离散整个三维计算域的时候,首先确定出需要计算的整个场景的包围盒.如图 1(a)所示,沿某根坐标轴对整个包围盒切片操作,每个切片为一个二维数组,即二维纹理,这样,每个体素就代表一个节点.为了计算方便,我们采用两套网格来表示主要变量,包括速度、密度和温度.这样,通过时刻 t 的状态量,计算出时刻 t+1 的状态量,

然后交换.由于在压强泊松方程和扩散方程的计算过程中需要迭代计算,而在 GPU 上,像素程序对纹理不能同时进行读写操作,所以迭代方法受到很大的限制,目前有雅可比迭代法[11]、共轭梯度法[10,13,20]以及红黑高斯-赛德尔法[11,12].在本文中只要求满足视觉效果,所以只需设定固定的迭代次数,而没有像文献[12]那样迭代直至收敛.如果需要,我们可以类似地采用硬件支持的 occlusion query 特征来进行收敛条件的判断.共轭梯度法需要进行矢量削减[10,13]的运算,需将数据从GPU返回给主内存,所以虽然该法整体收敛快,但单次迭代的效率不如雅可比迭代法;而红黑高斯-赛德尔法也需要两遍绘制才能完成一次迭代.故此,我们仍然选择雅可比迭代法.该方法不仅更简单,而且单次迭代的效率较高[3].

尽管我们可以像以前的方法[22]那样将某个标量和速度变量(3 个分量)压缩到纹元的 RGBA 这 4 个颜色通道以减少绘制的次数,但由于目前在我们使用的硬件上,像素程序尚只支持一个输出,所以当要求的变量多出 4 个时,不可避免地需要较多的绘制操作来完成整个变量的求解.但即使这样,我们仍可将密度和温度变量的求解压缩到一起,以减少绘制的次数.这样,我们首先计算整个速度场,然后再计算标量,最后利用密度场和温度场用来绘制流动的效果[6].

由于整个计算域是三维的,而 GPU 对于三维纹理的读写操作很慢,如果采用一系列二维切片的方式,则对应一个切片就需要一遍绘制过程,这样,需要多遍绘制才能完成整个计算域的一次迭代计算.为了避免性能的损失,尽可能地减少状态量的改变和绘制句柄的反复切换,我们采用与 Harris 等人[11]类似的方法,即将整个三维的计算域平铺到一个二维域上,如图 1(b)所示.但与文献[11]方法不同的是,我们预计算一幅二维索引纹理包含该切片的前后两个切片的位置,而不是一个简单的一维查找表纹理,这样,避免在像素程序中重新对每一个像素点进行定位计算,提高了整个模拟的运算效率.



(a) 3D texture (b) Flat 3D texture

(a) 三维纹理 (b) 平铺二维纹理

Fig.1 Flat 3D texture

图 1 平铺三维纹理

2.2.1 边界生成

对于实际问题而言,任意边界条件的处理至关重要.边界条件的不同使得各个问题不同,也才能体现出丰富多彩的流动效果.对于三维问题域来说,尽管我们可以像文献[23]那样在 CPU 上很容易地实现边界条件的处理,但那样每迭代一次,我们就需要将数据从 GPU 通过 AGP 通道读回到主内存进行边界处理,而这个过程很耗时,而且不得不在 CPU 上进行边界的判断以确定每个体素是否被障碍物占据,这无疑也十分耗时.由于主要计算是在 GPU 上进行的,所以希望边界条件的处理同样在 GPU 上完成.但目前 GPU 远不如 CPU 的操作灵活,为此我们提出一种新的方法,将所有的计算包括边界处理都放在 GPU 上完成,保证了整个计算的效率.

目前,我们使用的 GPU 上像素程序尚不支持分支和循环操作.所以,我们预先在图像空间生成边界障碍物图像,用这个障碍物图像来生成每个体素的偏移纹理;然后,根据边界条件生成主要变量(如速度、压强等)的修正因子.这里,采用两个像素程序就可以完成整个初始化的过程,这样就在 GPU 上完成了边界条件的处理,避免了 GPU 和 CPU 之间的频繁通信,加速了整个计算过程.图 2 给出了整个初始化的过程,在第 3.3 节中将详细展开.

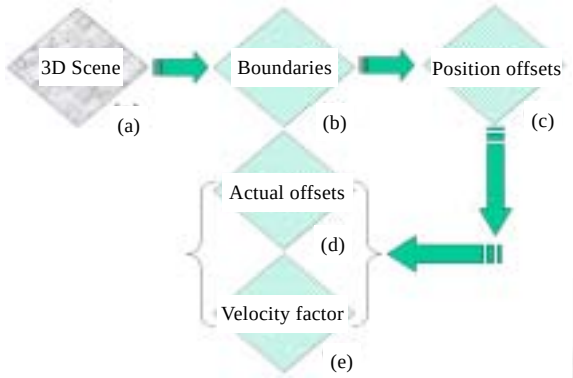


Fig.2 The whole initialization procedure

图 2 初始化过程

文献[24]采用深度剥离的方式来体素化整个场景,以生成 LBM 的动态边界.与此方法不同,我们采用直接体素化的方式,就是将整个场景不断剖切,然后正交投影,在图像空间形成障碍物图.但由于整个场景由几何面片构成,单纯的剖切操作会出现一些空洞的区域,如图 3(b)所示.为了形成实心的剖切截面,我们结合模板缓冲进行剪切,以得到正确的障碍物信息图.由于已将三维计算域转换到二维空间,所以在形成边界图的时候,我们同样将这些边界图组织成一幅纹理集,如图 3(c)所示.可以参见文献[25]了解形成实心剖切截面的具体实现过程.

(a) The solid model
(a) 原始几何场景

(b) The texture chart with just clipping
(b) 单纯剪切操作得到的障碍物纹理图

(c) The capping clipped texture chart
(c) 实心剖切截面纹理图

Fig.3 Boundary generation with 16 slices along *z*-axis

图 3 边界生成,沿 *z* 轴作 16 次切片

2.3 边界条件处理

在得到边界纹理以后,我们就可以根据边界条件对边界上的变量进行修正.任意复杂边界条件的处理对于实际问题的求解至关重要.这个问题在 CPU 上很容易解决,因为 CPU 提供了灵活的控制机制,但 GPU 为了保持高度的并行性而缺乏灵活的控制,所以不得不另辟蹊径.这里提供的方法可以处理任意复杂的边界,并且可以处理用户的任意交互输入的障碍物边界.为了能够适应一般的边界,文献[12]利用模版缓冲将边界值作为状态量模拟的一个扩充.与其不同的是,我们借鉴 Foster 等人对边界的处理方法[15]对边界进行分类,从而较文献[12]更为灵活,与 Harris 的方法[11]有些类似.但 Harris 的方法只能处理四周边界,我们的方法则可以处理任意复杂的内部边界.

通常情况下,边界条件可以分为 3 类,即 Dirichlet 边界、Neumann 边界和混合型边界,但都可以统一到一个方程:

$$a\phi + b\partial\phi/\partial\vec{n} = c \tag{6}$$

这里,ϕ 指的是速度、密度、温度和压强等,a,b 和 c 分别代表一定的系数.首先,根据障碍物纹理图将体素划分为障碍物和流体两类(如图 3(c)所示),用 0 或 1 表示.对于边界条件方程(6)的作用,我们采用一阶精度离散化,这样,边界上节点的值就由其周围的某一个节点来决定.由于方程(6)对法线方向求偏导数,为了在 GPU 上求解方便,

我们将任意法线方向简化为 28 个方向,所以采用式(7)编码的方法只需对每个节点周围的 7 个节点进行考察就可以求出该节点的类型,从而得出相对方位关系,如图 4 所示.这样,边界条件就转化为$\phi_{boundary}=d\phi_{offset}+e$,$d$ 和 e 由方程(6)的具体离散情况来决定,即表示为修正因子纹理.

$$Node'sType(i,j,k)=Obstacle(i,j,k)\times 64+Obstacle(i+1,j,k)+Obstacle(i-1,j,k)\times 2+Obstacle(i,j+1,k)\times 8 \\ +Obstacle(i,j-1,k)\times 4+Obstacle(i,j,k+1)\times 16+Obstacle(i,j,k-1)\times 32 \tag{7}$$

这里,$Obstacle(i,j)$即 1(被流体占据的节点)或者 0(被障碍物占据的节点).由于计算在图像空间进行,所以一旦得到障碍物信息,整个计算就与场景的几何复杂度无关了.

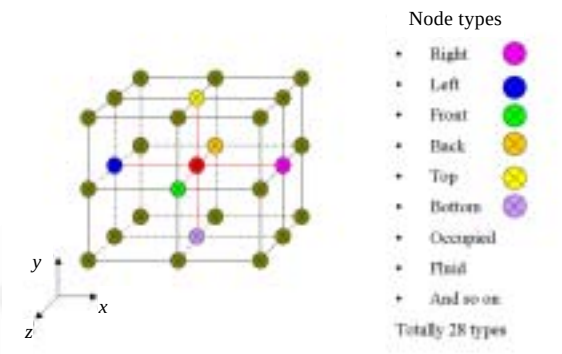


Fig.4 Relative orientation of voxels

图 4 各个体素的相对方位关系

表 1 给出了整个编码运算的过程与结果.这样,我们将编码的 64 个结果组织成 28 个方向,存储到位置偏移纹理中(如图 2(c)所示).然后,根据该偏移纹理,结合索引纹理,生成在二维平铺纹理空间的真实偏移(如图 2(d)所示).同时,根据边界条件计算出速度修正因子纹理(如图 2(e)所示).该过程只需两个像素程序即可完成.

Table 1 Code results

表 **1** 编码运算的过程与结果

Relative orientation	Binary coding	3D offset	Actual offset	Velocity modification factor
Right	000001	(1,0,0)	(1,0)	(0,−1,−1)
Left	000010	(−1,0,0)	(−1,0)	(0,−1,−1)
Bottom	000100	(0,−1,0)	(0,−1)	(−1,0,−1)
Top	001000	(0,1,0)	(0,1)	(−1,0,−1)
Front	010000	(0,0,1)	$(T_{offset-x},T_{offset-y})$	(−1,−1,0)
Back	100000	(0,0,−1)	$(T_{offset-z},T_{offset-w})$	(−1,−1,0)
Bottom-Right	000101	(1,−1,0)	(1,−1)	(0,0,−1)
Top-Right	001001	(1,1,0)	(1,1)	(0,0,−1)
…	…	…	…	…

由于边界条件的处理涉及到速度、压强、密度和温度等变量,我们将为这些变量分别根据系数 d 和 e 形成修正因子.在目前的系统中,我们对压强采用 Neumann 边界条件,即 $\partial p/\partial \vec{n}=0$.这样,边界上的值等于其相邻节点上的值,所以压强的修正因子一直是 1.对于静态的障碍物,我们认为垂直于其表面的节点速度对应分量等于 0;而对于非滑移边界,由于障碍物对流体有拖曳力的作用,将边界上的切向速度赋为附近流体节点上对应分量的负数;对于滑移边界,边界上的切向速度则直接赋为附近流体节点上对应分量.这样,根据边界类型既可动态地计算偏移位置,也可以预先计算好,对于当前处于边界上的值,根据偏移位置取出对应的值,结合 d 和 e 进行滑移或者非滑移处理后赋予它.对于流体本身和障碍物内部,其纹理偏移均为(0,0,0),即该边界处理的像素程序不改变其节点上的数值.在本文中,我们对速度变量只考虑了非滑移边界条件,通过速度修正因子纹理来体现(如图 2(e)所示).这样,我们通过两个不同的像素程序分别处理压强和速度边界条件.这些在二维域更容易理解,可以参阅我们以前的工作[22].对于标量(密度和温度)的边界处理将更为简单,我们直接指定边界上的值等于障碍物的

对应值.

显然,这种思路同样可以处理周期性边界条件.不过,需要修改一端的偏移位置指向整个计算域的另一端.这种方法还可以进一步推广到动态边界,如模拟运动物体对流体造成的影响,只不过我们需要不断更新坐标偏移纹理和修正因子纹理.

通过以上方法,结合平铺三维纹理的方式,整个计算化为二维问题.整个流体计算完全在 GPU 中实现,模拟过程高效.在目前的系统中,我们尚只求解了静态场景,但该方法本身对于动态场景仍是适用的.

3 结果与讨论

本文所采用的实验平台为 Intel Pentium 2.8GHz,主内存为 2G,显卡采用的是 GeForce FX5950 Ultra,显卡内存为 256M,显卡的核心频率为 375MHz,驱动的版本为 54.16,操作系统为 Windows 2000.整个实现基于 OpenGL.这里给出了一些实验结果,这些结果显示了本文所提出的方法的有效性和高效性.在图形硬件上,我们不仅完成了整个基于物理的流体方程求解,同时也将计算的结果实时绘制出来.

图 5 给出了气流在迷宫中流动的场景,图 6 给出了气流在一个城市中流动的情形,充分体现了本文算法处理复杂边界的能力,从而可以用来实时模拟诸如有害气体泄漏、火灾等事故.更多的演示动画可以访问 http://lcs.ios.ac.cn/~lyq/demos/gpgpu/gpgpu.htm.

Fig.5 Flowing in a maze

图 5 气流在迷宫中流动的场景

Fig.6 Flowing in a city

图 6 城市中气体流动情形

目前,我们只是在 pbuffer 中使用复制到纹理这种思路,所以在性能上还有提升的空间.如果采用绘制到纹理,可以得到更好的性能.但这时存在的一个问题是,整个计算需要多个 pbuffer 句柄的不断切换,这样使得其整体性能并不比采用一个 pbuffer 然后使用复制到纹理这种思路高效.为了避免这个问题,就需要使用 pbuffer 的多个表面[12,21]作为绘制对象,而这个问题目前各个硬件支持的情况不太一样.

在我们的实验中,整个花费的时间由 4 部分构成.在静态场景中,边界生成和边界条件的准备只需要做 1 次.这样在实时模拟过程中,我们只需要求解流体方程,然后绘制出整个效果,其中流体方程的求解占用了整个时间耗费的绝大部分.无疑,模拟耗费的时间与绘制的遍数成一个简单线性正比的关系,如果我们设定固定的迭代次数为 N,并考虑扩散对流、浮力、漩涡约束等所有因素,则总绘制遍数等于 $11+6N$.当然,迭代次数越高,整个计算越精确.图 5 中计算域的分辨率为 64×34×16,N=6,绘制尺寸为 512×512,模拟效率大概为 21.4fps.通过实验可以看到,在计算流体运动方程中,GPU 相对于 CPU 而言性能要高出很多.虽然 GPU 在编程上比较复杂,而且不如 CPU 那样灵活,但它提供的可编程性使得人们能够根据方程本身的特点来设计算法,以充分利用 GPU 的并行特性来加速,从而有效地实现快速求解.

4 结论与讨论

本文利用目前新的图形硬件求解流体的运动方程,从而达到实时模拟的效果,将方程的求解与绘制整合在一起,减小了 CPU 处理的压力,让 CPU 解放出来做其他事情.在求解过程中,为了更进一步加速整个计算过程,

我们将多个标量压缩在一起,以减少绘制的次数.另外,对于复杂边界的处理更加通用化,通过将整个几何场景转化到图像空间,使得算法与几何场景的复杂度无关.在图像空间修正边界值,使得整个算法更加实用化,能够模拟出各种复杂的流动.可以看出,图形硬件对于整个计算的加速是很明显的.

但目前利用GPU来求解流体运动方程仍存在很多限制:首先,该方法不太适合像文献[8]那样的大规模问题求解,内存有限,而且存在最大纹理尺寸的限制,使得所需求解的问题分辨率不能太高,不过这个问题可以通过纹理压缩或者多遍的方法得到缓解;而灵活性的缺乏以及指令集的局限性使得 GPU 编程的复杂度增加,而且难以实现一些运算,如位操作等.另外,由于整个计算基于像素,所以瓶颈亦在于此.通常情况下,高效的计算应该是寻求在 CPU,AGP 通道、顶点级以及像素级这几者之间达到一个很好的均衡.文献[21]为我们提供了一个很好的思路,即利用 GPU 与 CPU 之间的消息机制来维持整个计算的负载均衡.更多关于 GPU 的讨论可以参考文献[3,26].

目前,整个算法完全基于汇编来实现,而高级实时绘制语言,如 Cg 应该有助于编写更高效的代码.所以在将来的工作中,我们准备将现有的工作转到 Cg 上去.另外,由于半拉格朗日方法的精度不够高,使其在计算流体力学领域的应用很受限制,而多重网格法[12]从某种程度上可以提高整个计算的精度,从而满足工程应用的要求.需要进一步完成的工作还包括烟雾或者火焰本身的高度真实感绘制等.

随着新的硬件特性的不断出现,如 nVidia 和 ATI 公司的 Frame Buffer Object(FBO)和 Render Buffer Object(RBO)[27],它们将完全代替 pbuffer,从而将更方便我们利用GPU并行处理的能力.基于此,我们还将设计一些更为精巧的算法来加速 PDEs 的计算并增强绘制效果.

References:

[1] Macedonia M. The GPU enters computing's mainstream. IEEE Computer Society, 2003,36(10):106−108.

[2] GPGPU Website. http://www.gpgpu.org

[3] Wu EH, Liu YQ. General purpose computation on GPU. Journal of Computer-Aided Design & Computer Graphics, 2004,16(5): 601−612 (in Chinese with English abstract).

[4] Stam J. Stable fluids. In: Proc. of the SIGGRAPH. New York: ACM Press, 1999. 121−128.

[5] Enright D, Marschner S, Fedkiw R. Animation and rendering of complex water surfaces. ACM Trans. on Graphics (Proc. of the SIGGRAPH), 2002,21(3):736−744.

[6] Fedkiw R, Stam J, Jensen HW. Visual simulation of smoke. In: Proc. of the SIGGRAPH. New York: ACM Press, 2001. 15−22.

[7] Foster N, Fedkiw R. Practical animation of liquids. In: Proc. of the SIGGRAPH. New York: ACM Press, 2001. 23−30.

[8] Rasmussen N, Nguyen DQ, Geiger W, Fedkiw R. Smoke simulation for large scale phenomena. ACM Trans. on Graphics (Proc. of the SIGGRAPH), 2003,22(3):703−707.

[9] Nguyen DQ, Fedkiw R, Jensen HW. Physically based modeling and animation of fire. ACM Trans. on Graphics (Proc. of the SIGGRAPH), 2002,21(3):721−728.

[10] Bolz J, Farmer I, Grinspun E, Schröoder P. Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. ACM Trans. on Graphics (Proc. of the SIGGRAPH), 2003,22(3):917−924.

[11] Harris MJ, Coombe G, Scheuermann T, Lastra A. Simulation of cloud dynamics on graphics hardware. In: Proc. of the Graphics Hardware. Aire-la-Ville: Eurographics Association, 2003. 92−101.

[12] Goodnight N, Woolley C, Luebke D, Humphreys G. A multigrid solver for boundary value problems using programmable graphics hardware. In: Proc. of the Graphics Hardware. Aire-la-Ville: Eurographics Association, 2003. 102−111.

[13] Krüger J, Westermann R. Linear algebra operators for GPU implementation of numerical algorithms. ACM Trans. on Graphics (Proc. of the SIGGRAPH), 2003,22(3):908−916.

[14] Stam J, Fiume E. Turbulent wind fields for gaseous phenomena. In: Proc. of the SIGGRAPH. New York: ACM Press, 1993. 369−376.

[15] Foster N, Metaxas D. Realistic animation of liquids. Graphical Models and Image Processing, 1996,58(5):471−483.

[16] Rumpf M, Strzodka R. Using graphics cards for quantized FEM computations. In: Proc. of the VIIP. 2001. 98−107.

[17] Li W, Wei XM, Kaufman A. Implementing lattice Boltzmann computation on graphics hardware. The Visual Computer, 2003, 19:444−456.

[18] Harris MJ, Coombe G, Scheuermann T, Lastra A. Physically-Based visual simulation on graphics hardware. In: Proc. of the Graphics Hardware. Aire-la-Ville: Eurographics Association, 2002. 109−118.

[19] Harris MJ. Flo: A real time fluid float simulator written in Cg. 2003. http://www.markmark.net/gdc2003/

[20] Batty C, Wiebe M, Houston B. High performance production-quality fluid simulation via NVIDIA's QuadroFX. 2003. http://film.nvidia.com/docs/CP/4449/frantic_GPUAccelerationofFluids.pdf

[21] Lefohn AE, Kniss JM, Hansen CD, Whitaker RT. Interactive deformation and visualization of level set surfaces using graphics hardware. In: IEEE Visualization. IEEE Computer Society, 2003. 75−82.

[22] Wu EH, Liu YQ, Liu XH. An Improved study of real-time fluid simulation on GPU. Journal of Computer Animation and Virtual World (CASA 2004), 2004,15:139−146.

[23] Yoshida S, Nishita T. Modeling of smoke flow taking obstacles into account. In: Proc. of the Pacific Graphics. IEEE Computer Society, 2000. 135−145.

[24] Li W, Fan Z, Wei XM, Kaufman A. GPU-Based flow simulation with complex boundaries. Technical Report, 031105, Computer Science Department, SUNY at Stony Brook, 2003.

[25] Westermann R, Ertl T. Efficiently using graphics hardware in volume rendering applications. In: Proc. of the SIGGRAPH. New York: ACM Press, 1998. 169−177.

[26] Wu EH. State of the art and future challenge on general purpose computation on GPU. Journal of Software, 2004,15(10): 1493−1504 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1493.htm

[27] Nvidia Website. http://www.nvidia.com/

附中文参考文献:

[3] 吴恩华,柳有权.基于图形处理器(GPU)的通用计算.计算机辅助设计与图形学学报,2004,16(5):601−612.

[26] 吴恩华.图形处理器用于通用计算的技术现状及其挑战.软件学报,2004,15(10):1493−1504. http://www.jos.org.cn/1000-9825/15/1493.htm

柳有权(1976–),男,湖北秭归人,博士生,主要研究领域为计算机图形学,计算机动画.

刘学慧(1968–),女,博士,副研究员,CCF高级会员,主要研究领域为计算机图形学,虚拟现实.

吴恩华(1947–),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为计算机图形学,可视化,虚拟现实.