基于多重网格法的实时流体模拟

周世哲[1]　满家巨[2]

(1，2：湖南师范大学，数学与计算机科学学院，410081)

摘　要：本文研究实时流体模拟问题，在 GPU(Graphic Processor Unit，简称 GPU)上实现了多重网格法，并用它改进了二维的实时流体模拟，更充分地利用 GPU 的并行计算能力。我们使用四层网格，依靠渲染到纹理的计算方式，帧缓存扩展的纹理管理方法，提高了图形硬件的利用率。通过实验对比，在同样的帧数下我们的方法能提高GPU 实时流体模拟的精度。尤其在较大规模的问题上与同等精度的基于一般迭代方法的GPU 实时流体模拟相比，我们的方法在速度上有成倍的提高。

关键词：GPU；实时流体模拟；多重网格法；帧缓存扩展

Realtime Fluid Simulation Based on Multigrid Method

Zhou Shizhe[1]　Man Jiaju[2]

(1 College of Mathematics and Computer Science, Hunan Normal University, 410081)

Abstract: In this paper we research realtime fluid simulation,we implement multigrid method on GPU and improve the accuracy of 2D realtime fluid simulation by using the multigrid method. We use 4 layers of grids in our GPU based multigrid method, and by using framebuffer extension and the way of rendering to textures, we manage to put all the textures needed in graphic hardware memory to increases the rate of utilization of GPU. The test results show that our method can improve the accuracy of GPU based 2D realtime fluid simulation with the same fps. Especially in lager scale situation our method speeds up the simulation by times compared with other GPU based realtime fluid simulation using normal iteration method with equivalent accuracy.

Keywords：GPU(graphic processing unit); realtime fluid simulation; multigrid method; framebuffer extension

1　引言

计算机图形学研究者在对复杂物理现象的模拟上已经迈进了一大步。这其中对水，云，烟等流动现象的模拟是其中的重要内容。随着这几年 GPU 在性能和可编程性上的巨大提升，如何利用 GPU 来进行实时可交互的流体模拟已成为计算机图形学中的一个热点问题。

在计算流体力学(Computational Fluid Dynamics，简称 CFD)CFD 中为了力求精确地描述流动现象，必须求解复杂的数学物理方程，其中最基本的是纳维-斯托克斯方程组(Navier-Stokes Equation，简称 NSE)。目前在 CFD 中许多数值方法[1-4]都要求使用极小的时间步长，这导致其求解相当耗时，速度无法达到实时模拟的要求。1999 年，Stam[5]引入所谓不完全拉格朗日算法或称半拉格朗日算法。该方法允许采用较大的时间步长，同时保持良好的稳定性，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。到了近几年，在图形处理器上进行的实时流体模拟(Realtime Fluid Simulation，简称 RFS)取得了许多成果。

在 RFS 中需要对大型稀疏线性方程组进行求解，目前基本上是使用 Jacobi 迭代法，该方法结构简单，且迭代时没有额外的数据通信，因此很适合在 GPU 上运行。但如果希望提高 RFS 的模拟精度和细节丰富程度，并仅仅通过提高 Jacobi 迭代次数的方法，将使动画的帧数急剧下降。有一种被广泛接受的解大型稀疏线性方程组的加速方法是多重网格法，Goodnight 等人[12]在 GPU 上应用多重网格算法

通讯作者：满家巨 manjiaju@hotmail.com

求解边界值问题; Bolz 等人在[13]中实现了基于像素编程的非结构网格的共轭梯度法和规则网格上的两层网格的多重网格法，但是由于网格数过少并不能体现出多重网格方法的优势。我们的工作是在 GPU 上实现了多重网格法(Multigrid-Method，简称 MGM)，并将它引入了 RFS，在较大规模和较高模拟精度的情形下我们的方法能成倍的提高速度，并且可以方便的采用更多层的网格来加强效果。

2 相关工作

Stam和Fiume等人在[6]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，采用对流扩散方程进行计算，而随机分量则采用Kolmogorov频谱来模拟小尺度行为，这样能让模拟的效果产生混乱无序感，更加逼近自然界中真实的风现象; Fedkiw在[7]中引入了漩涡约束因子(vorticity confinement)以弥补数值耗散，该方法的基本原理是着眼于流畅中的旋度场，人为恢复由高旋度区域指向低旋度区域的向心力，来产生局部区域中的漩涡现象，从而更逼真地模拟烟雾，并且在[8]中对火焰等燃烧现象进行了模拟; Harris在[9]中引入热力学参量参与流体方程的求解，加入了对浮力和温度的变化，很好的模拟了云的生成和飘动。最新的成果有: Irving等在[17]中使用二维高度场与三维NSE方程相结合的方法有效地模拟了大尺度的流体与物体交互时的动态; Treuille等[18]在GPU上采用局部模型缩减法很大的提高了流体模拟在于用户交互时的速度; Klinger等[19]则另辟蹊径，使用动态生成的三角形网格剖分模拟区域，提高了模拟的场景复杂度，但由于是非正交的不规则网格目前难以在GPU上实现。

随着近几年GPU在运算性能上和可编程度上的飞速提高，以上提到的研究成果中有相当一部分被从CPU上移植倒GPU上来实现。从2004年开始，利用GPU进行通用计算的研究(General Purpose on GPU，简称GPGPU)[16]得到开展和普及，一些基于GPGPU的线性代数和偏微分方程的数值解法被实现，这些方法被证明比当下的CPU程序要快许多倍。其中比较著名的包括: Krüger在[10]中利用GPU实现了求解大型稀疏线性方程组的高斯-赛德耳迭代(简称G-S迭代)法和共轭梯度法; RUMPF和STRZODKA在[11]中利用GPU进行基于有限元方法的图像处理; Goodnight等人[13]在GPU上应用多重网格算法求解边界值问题; Bolz等人在[12]中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。正是这些数学方面的研究，使得RFS等这一类的物理动画(Physicall-Based Animation)的开发达到新高度。

3 基于 GPU 流体模拟基本方法

流体模拟的核心是对NSE方程的不断求解。在每个时间步长中，都要按NSE方程的解对速度场u和密度场 ρ 就进行更新。Stam[5]中提出了对NSE动量方程(1)进行分解，然后逐个求解的方案。

$$\begin{cases} \dfrac{\partial \boldsymbol{u}}{\partial t} = -(\boldsymbol{u}\cdot\nabla)\boldsymbol{u} - \nabla p + \nu\Delta\boldsymbol{u} + \boldsymbol{F} \\ \nabla\cdot\boldsymbol{u} = 0 \end{cases} \tag{1}$$

NSE动量方程可以被分解为，外力作用方程，输运方程，黏度扩散方程，压强方程，和连续性方程。每个方程都以上个方程所求得的结果作为定解条件，依次完成后即算作对NSE动量方程的一次完整地求解，得到下一个时间步长的新速度场u。详细的内容可参见[5][14]。

以u代入密度的对流方程(2)，可求得下一个时间步长的新密度场 ρ 。

$$\frac{\partial \rho}{\partial t} = -(\boldsymbol{u}\cdot\nabla)\rho \tag{2}$$

这样流场内一个时间步长后的两大物理元素速度和密度就都得到了更新。最后的步骤是可视化密度场。我们通过密度场实时生成法线贴图，并配合颜色偏移来产生光照效果。重复上述过程，即可得到一段时间内连续的流体动画。

在求解 NSE 动量方程的黏度扩散方程和压强方程这两步中都涉及对大型稀疏线性方程组进行求解。Harris 在[14]和柳有权等人[15 所作的模拟程序，都是采用 Jacobi 迭代法来求解压强和黏度扩散方程。正如以上两篇文献中都提到的，如果只是为了初步满足视觉要求，在流场规模为 300*300 网格分辨率的情形下，30-50 次左右的 Jacobi 迭代即可胜任。但是我们发现，如果提高网格分辨率，并对模拟的精度提出更高的要求，则 Jacobi 方法的效率较差。

4 多重网格法及其硬件加速

多重网格法（MultiGridMethod，简称MGM）是一种在工程数值计算领域广泛使用的偏微分方程数值解法，在文献[3]中提到MGM具有收敛速度不受网格步长影响的特点。其效率明显优于其它如Jacobi和G-S等固定网格步长迭代法。

对一维问题 $\dfrac{\partial^2 u}{\partial x^2}+f=0$

在步长为h的均匀网格上中心差分离散格式为：

$$u_{i-1}-2u_i+u_{i+1}=-h^2 f_i$$

其G-S迭带格式可写成

$$u_{i-1}^{n}-2u_i^{n}+u_{i+1}^{n-1}=-h^2 f_i$$

对于误差 $r_i^n=u_i-u_i^n$，显然 r_i^n 和 u_i^n 具有同样的迭代格式 $r_{i-1}^{n}-2r_i^{n}+r_{i+1}^{n-1}=0$

若对误差作傅立叶展开，将其第k重谐波分量记为 $r_i^{n}(k)=A^n e^{\sqrt{-1}ikh}$，$A^n=\left|r_i^n\right|$ 为该分量的振幅，可得 $\dfrac{A^n}{A^{n-1}}=\lambda=\dfrac{e^{2\sqrt{-1}ikh}}{2e^{\sqrt{-1}ikh}-1}$，$\left|\lambda\right|=\dfrac{1}{\sqrt{5-4\cos kh}}$。

$\left|\lambda\right|$ 代表该谐波振幅的衰减速度，$\left|\lambda\right|$ 越小，该分量衰减越慢。当kh在区间(0，π)内增大时，$\left|\lambda\right|$ 变小，而kh越大，表明该分量的频率越高。这说明，在固定步长h的情况下，误差 r_i 中的高频分量在迭代过程中衰减得较快，而低频分量收敛得慢，这也是Jacobi和G-S等固定网格步长迭代法的收敛速度慢的原因。针对这个缺陷，可将迭代过程作改进。在以初始步长h的网格上迭代若干次后，误差中的后面的高频部分已经基本衰减掉了，剩下位于前部从k=1开始的低频分量，这时可将步长h放大，kh随之增大，使这些低频分量获得较高的衰减速度，然后进一将步长增大，更低频的误差分量也得以加速衰减。

MGM 算法包含四大步骤：松弛迭代，计算网格残差，残差限定和误差延拓。为了便于网格点上数据的传递与插值，我们采用 4 层标准粗化系列网格[3]，即每层网格是均匀剖分，相邻两层网格步长比为 2。下标"l"表示对应的网格层，越细的网格层 l 越大。

松弛迭代是在最细网格上迭代初始解使之达到相应阶数的平滑。对线性方程组(3)

$$\boldsymbol{Lu}=\boldsymbol{f} \qquad (3)$$

可简单采用 Jacobi 迭代完成，也可使用加权的 Jacobi 迭代[13]；得该层的不精确解 u_l；计算网格残差是将(3)迭代数次后得到的不精确解 u_l 回带式(4)求得该层网格残差 r_l $\boldsymbol{r}_l=\boldsymbol{f}-\boldsymbol{L}\boldsymbol{u}_l$ (4)

残差限定是将细网格的残差向下层粗网格传递，在我们使用的标准化网格上 restrict 限定算子为简单的平凡单射

$$\boldsymbol{r}_{l-1}=\text{restrict}(\boldsymbol{r}_l)$$

$$\text{restrict}:\ I_{l-1}(x,y)=I_l(2x-1,2y-1) \qquad (5)$$

误差延拓是将本层产生的解误差 e_l 传递到上层较粗网格，为了满足所解方程(3)的二阶光滑性要求，我们使用双线性插值，将粗网格中的一点延拓到上层细网格中相应位置以及相邻的上，右和右上的四个点 $\boldsymbol{e}_{l+1}=\text{extend}(\boldsymbol{e}_l)$

extend:

$I_{l+1}(2x+1,2y+1)=I_l(x,y)$ --- 相应

$I_{l+1}(2x+1,2y+2)=\frac{1}{2}\left(I_l(x,y)+I_l(x,y+1)\right)$--- 上

$I_{l+1}(2x+2,2y+1)=\frac{1}{2}\left(I_l(x,y)+I_l(x+1,y)\right)$--- 右

$$I_{l+1}(2x+2,2y+2)=\frac{1}{4}\begin{pmatrix}I_l(x,y)+I_l(x+1,y+1)+\\ I_l(x,y+1)+I_l(x+1,y)\end{pmatrix}$$

---右上 (6)

式(4)和(5)中的 $I_l(x,y)$ 表示 l 层网格上位于第 x 列，第 y 行的数据。网格最左下角的数据是 $I_l(1,1)$。

我们使用 MGM 算法的具体流程以图 1 来表示。图中"o"表示网格残差 r_l，"□"表示解误差 e_l，"\"表示限定，"/"表示延拓，"⌒"表示松弛迭代，"⌣"表示求网格残差。如图，算法大体上可分为先下降和后上升两阶段：下降阶段即将最细层上的残差限定到底(最粗层 layer1)；上升阶段则是将最粗层精确求出的误差当成上层的校正量，将校正量插值到顶(最细层 layer4)； 最后在最细层校正后得新的近似解。重复这样一个过程可得到满足精度要求的解。这是一种叫做多层 V 循环的 MGM 方法[3]。

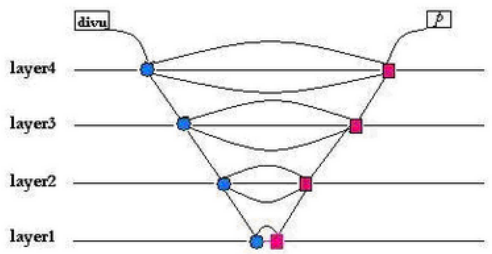

图1 MGM算法流程示意图

(图中的div$\boldsymbol{u}$表示在RFS求压强的步骤中MGM的输入为速度的散度，而 p表示输出为压强。)

为了完成在GPU上的流处理计算，我们把计算过程需要的网格数据分别保存在纹理中。MGM运行过程所需的全都是正交网格数据，这也是它便于在GPU上编程的优势之一。统计所需纹理到每一层上，为保证算法的连续执行，必须在每一层上保存该层的误差 e_l 和残差 r_l，如果每个网格数据用一个纹理来保存，则需要的纹理数 N_t 是网格层数 N_l 的两倍即 $N_t = 2* N_l$。

如前述，算法可分为下降和上升两阶段。在下降阶段，渲染区域逐层缩小，而在上升阶段则相反。被渲染的纹理依次为下降阶段的误差纹理 E_4, E_3, E_2, E_1，和上升阶段的网格残差纹理 R_1, R_2, R_3, R_4，为了最大限度的发挥 GPU 的运算能力，我们采用渲染到纹理(Render To Texture，简称RTT)的计算方式，将程序运行所需的所有数据资源和计算场所移植到 GPU 和与其配套的帧缓存(Frame Buffers，简称 FB)上来，以避免缓慢的显存-主存间的数据交换。我们使用最近流行的为 Opengl 开发的 FBO Extension 存储管理方案。

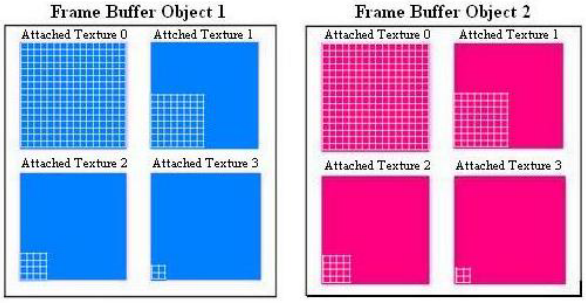

图2　MGM计算纹理布局:

(蓝色表示存放网格残差的纹理 R_l，红色表示存放误差的纹理 E_l，被网格覆盖的区域为渲染区。四个 R_l 都附着在FB1上，而四个 E_l 都附着在FB2上。)

Goodnight 等人在[12]中使用的纹理储存管理方法是使用 OpenGL Pixel Buffer 中的 FRONT 和 BACK 帧缓存切换法，来满足算法在每一层上计算误差和残差时所需的渲染区，这样做需要不断地在存放各网格层数据的 Buffer 间来回切换。与[12]不同的是我们将下降阶段依次生成的四个网格残差纹理作为一组附着在同一个 FB 上，再将上升阶段依次生成的四个误差纹理作为一组附着在另一个 FB 上，如图 2。这样做不但不需要与系统主存数据交换还减少了运行时在不同的 FB 间切换的次数。实验证明这样做的确可以获得约 5%的性能提升。我们使用 4 层网格，相比[13]的两层网格更能发挥 MGM 的优势，同时还能刚好将一个 FB 上的 4 个纹理附着区用完，更大的发挥图形硬件的效能。

我们用CG语言编写MGM算法的核程序。在MGM的限定程序中，我们使用CG内建的投影取样函数texRECTproj(tex ，half3(coords，scale))，这样在每个像素上我们都能节省一次二元数组的除法，能使限定步骤的GPU计算量减少。

在延拓程序上，我们使用了多模板缓存方法，对被渲染区内不同位置上的纹素分别使用不同的像素程序。如图3，灰色区域为模板所遮盖的区域，GPU将不会在这些区域执行像素程序；我们首先运行第一个像素程序，配合“井”字形的模板，使绿色O的值由下层粗网格传递上来(图3.a)；然后运行第二个像素程序，配合隔列的“||” 形模板，使蓝色三角形的值等于左右两侧网格值和的一半(图3.b)；最后再运行第三个像素程序，配合隔行的“=” 形模板，使红色方框的值等于上下相邻网格值和的一半(图3.c)。经过这三个流程后，便能完成满足式(6)的延拓效果。所需模板缓存事先以纹理形式保存。该方法能避免在像素程序中出现动态分支操作，因为对目前的GPU来说，在每个像素上都进行if、while等动态分支操作的效率是极其低下的。

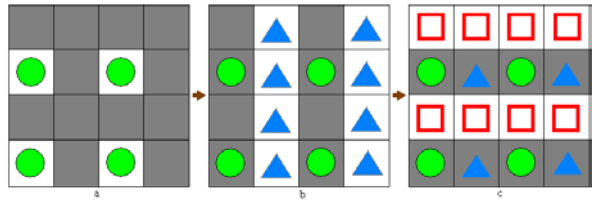

图3　多模板缓存示意图

5　GPU 实时流体模拟的实现

我们将模拟程序分为更新速度场，更新密度场和可视化密度场三大部分来依次完成。流场内的初始速度场和初始密度场我们分别制成纹理保存。在更新速度场部分，对每个时间步长，我们首先按照[5]中稳定流体的半拉格朗日方法计算输运方程，然后用我们的4层MGM计算黏度扩散方程和压强方程，在解压强方程的步骤中我们以速度的散度场divu作为 layer4的 r_l，将所求的压强 p 作为layer4的 e_l (图1)；对以下各层，我们都以全零网格作为下降阶段的

迭代初始预估值e_l。当最终回到顶层layer4时，即可得到结果e_4，也就是我们想要的压强 p 。而在黏度扩散方程中的MGM算法的输入和输出都是速度场。最后用压强修正速度场以满足连续性方程。在更新密度场部分，我们以更新后的速度场来计算密度的输运方程式(2)。最后将更新后的密度场进行显示。

在数据的管理方面，以往采用pBuffer的方式[14][15]需要将纹理数据写回系统主存。与此不同的是我们采用RTT的方式在GPU上执行这一系列的运算，每一个步骤的结果都以纹理保存并作为下一个程序的输入，所有的计算输入输出包括中间结果都以纹理的形式存储，并附着在FB上，整个过程不需要显存与系统主存间的数据交换。

在边界条件的处理上，我们采取的方法和[5]中叙述的基本一致。对于压强方程(2)，其边界条件离散到我们的网格上，就是流域边界线两侧网格压强值相等，这样就能达成在边界线上压强导数处处为零的边界条件；而对于黏度扩散方程，我们使速度在边界线两侧互为相反数，以满足所谓的不可穿越条件[5]。我们区别对待在 MGM 算法执行过程中生成的网格残差与误差的边界条件。对于误差e_l，因为他们是上层较细网格解的修正量，最终是解压强 p 的一部分，所以它们与 p 满足同样的边界条件。即对每一个误差纹理E_l，其边界纹元与相邻的纹元相同；而对于网格残差r_l，它们不存在边界限制，所以残差纹理R_l的边界纹元为全零。

6 实验结果和分析

我们实验使用的显卡是 Geforce6800GS，内建有 12 条像素渲染管线。核心频率是 350MHZ。为了保证模拟动画一定的流畅度，必须采用额定迭代次数的方法。在目前的 GPU 实时流体模拟程序[14][15]中，基本上都采取了这样的方式。[5]和[14]的研究发现一般迭代次数在 30-50 次，就可以很好的满足视觉要求。但是这样做会损失流体模拟的精度，如果提高迭代次数，则模拟可以有更准确的细节和更真实的表现。

通过采用[12]介绍的方法，我们使用GF6800GS提供的硬件遮挡查询(Occlusion Query)，设定迭代前后两次的误差阈值，不额定迭代次数，对模拟在第一时间步长内达到预定精度收敛所需要迭代次数进行了记录。结果是，如果要达到精度为10^{-4}的稳态，Jacobi迭代需要901次，动画帧数为1.5fps。而要达到精度为10^{-6}的稳态，Jacobi迭代需要4527次，动画的帧数仅为0.3fps左右。采用四层网格MGM程序，能将上述这两种精度的动画帧数提高到4.8fps和2.5fps。我们列出901次Jacobi迭代(图4-A)和相应的MGM方法的模拟动画截图(图4-B)。

对于同等大规模和较低的精度要求的情形，我们设定了适当的迭代次数，使得Jacobi程序和MGM程序能获得相同的帧数，见图5。不难发现MGM程序模拟的流体动画在形态上更加接近图4(A)中达到10^{-4}稳态的Jacobi迭代901次的模拟动画，细节也更加丰富。而与之对比的Jacobi程序虽然拥有相同的帧数，但是在模拟到第212帧时，与图4的第212帧相比其形态已完全走样，这说明Jacobi方法在较大规模的RFS问题上如果迭代次数较少，模拟的精度是很差的。由于我们采用的存储管理方式，我们这里的Jacobi程序与[14]在同样规模和同样的迭代次数下的运行帧数相比高3到5fps。本实验说明仅用两层网格也能让RFS在保持速度的前提下获得较高的模拟精度，但是效果提升度不如4层网格明显。

实验证明：在 GPU 上进行高精度的 RFS，特别是在流场的离散规模较高的情况下，MGM 能成倍的提高速度；而且在具有同等速度的只需满足视觉要求的 RFS，使用 MGM 也能让模拟更准确，细节更加丰富，更加接近工程要求的模拟精度。同时我们的方法没有改变 RFS 的可交互性，也没有破坏基于 GPU 的 RFS 的模块化结构，因而可以很容易地移植到 3D 情形。与[13]采用的两层网格相比，我们的方法的网格层数更多；并且由于我们采用的编程模式可以充分的利用图形硬件的显存容量，使得进一步提高 RFS 的离散规模和网格层数变得简便易行。

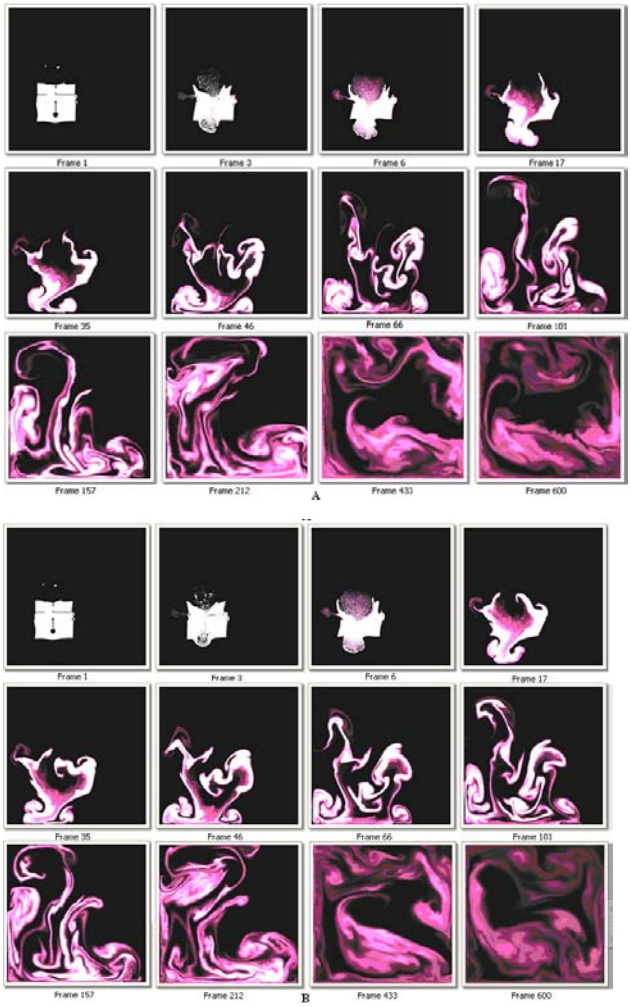

图 4　不可压缩二维方腔流的模拟

(流场离散规模为 514*514。初始的密度场表示为 Frame1 中的一块正方型淡红色染料，在受到自上而下和自右向左的两个外力冲击后，产生流体运动的形态。A 部分 12 幅图片采用 901 次 Jacobi 迭代生成。动画帧数为 1.5fps；B 部分 12 幅使用 4 层 V 循环 MGM 法，在 Layer4 到 Layer1 上的 Jacobi 迭代次数分别为 128,64,32 和 75。动画的帧数为 4.8fps。我们截取了两种模拟在第相同帧时的图片。)

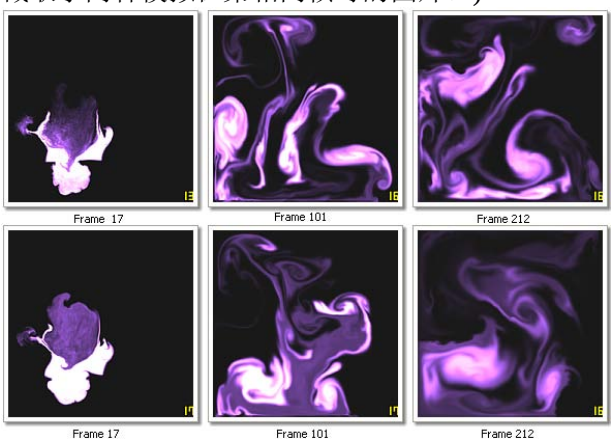

图5　有相同帧数的使用两层网格 MGM 程序和使用 Jacobi 迭代程序的模拟效果的对比图

(初设条件与图 4 的相同，只是将染料换成了紫色。上三幅采用两层网格 MGM 法，细网格上迭代 24 次，粗网格上迭代 35 次；下三幅采用 60 次 Jacobi 迭代。二者的帧数都为 17fps 左右。)

参考文献

[1] W. F. 休斯等. 流体动力学[M]. 科学出版社, 2002.

[2] 吴子牛.计算流体力学基本原理[M].科学出版社,2001.

[3] 李炜. 粘性流体混合有限分析解法[M]. 科学出版社, 2000.

[4] 李德元等. 二维非定常流体力学数值方法[M]. 科学出版社,1998.

[5] Stam J. Stable fluids. In: Rockwood A，ed[C] // Proc. of the SIGGRAPH'99. New York: ACM Press，1999 : 121−128.

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

[7] Fedkiw R, Stam J. Visual simulation of smoke[C] // Proc. of the SIGGRAPH. New York: ACM Press，2001 : 15−22.

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

[9] Harris M, Coombe G, Scheuermann T. Simulation of cloud dynamics on graphics hardware[C] // Proc. of the Graphics Hardware. Aire-la-Ville: Eurographics Association, 2003 : 92−101.

[10] Krüger J，Westermann R. Linear algebra operators for GPU implementation of numerical algorithms. ACM Trans. on Graphics[C] // Proc. of the SIGGRAPH,2003: 908−916.

[11] Martin R, Robert S. Using Graphics Cards for Quantized FEM Computations[C] // Proceedings VIIP'01，2001.

[12] Nolan G, Gregory Lewin，David Luebke. A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware[R]. University of Virginia. Technical Report CS-2003-03.

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

[14] Mark J. Harris. Fast Fluid Dynamics Simulation on the GPU[C] // Randima Fernando. GPU Gem. University of North Carolina at Chapel Hill : Mark Harris， 2004 :

chapter 38.

[15] Liu Youquan. LIU Xue-Hui, WU En-Hua. Real-Time 3D Fluid Simulation on GPU with Complex Obstacles.Journal of Software, 2006, 17(3): 568-576.
(柳有权，刘学慧， 吴恩华. 基于 GPU 带有复杂边界的三维实时流体模拟[J]。软件学报, 2006, 17(3): 568-576。)

[16] GPGPU website. http://www. gpgpu. org

[17] Irving G，Guendelman E，Losasso F. Efficient simulation of large bodies of water by coupling two and three dimensional techniques[C] // ACM Transactions on Graphics 25(3) , SIGGRAPH 2006. 2006.

[18] Treuille A ，Lewis A ，Popovi´c Z. Model Reduction for Real-time Fluids[C] // ACM Transactions on Graphics 25(3) , SIGGRAPH 2006 . 2006.

[19] Klingner B, Feldman B, Chentanez N. Fluid Animation with Dynamic Meshes[C] // ACM Transactions on Graphics 25(3) , SIGGRAPH 2006. 2006.

作者简介：周世哲，男，1984，硕士研究生，目前主要从事计算机图形学和虚拟现实动画的研究。
满家巨，男， 博士, 硕士生导师，目前主要从事计算机图形学和图像处理的研究和教学等。

基金项目：受国家自然科学基金(10571053) 和湖南省教育厅青年基金项目(04B037)资助。