Journal of System Simulation

Volume 27 | Issue 4 Article 18

8-20-2020

Fluid Simulation of SPH Based on OpenCL

Miaomiao Xiao
Faculty of Information Science and Technology, Ningbo University, Ningbo 315211, China;

Liu Zhen
Faculty of Information Science and Technology, Ningbo University, Ningbo 315211, China;

Jiabin Shi
Faculty of Information Science and Technology, Ningbo University, Ningbo 315211, China;

Tingting Liu
Faculty of Information Science and Technology, Ningbo University, Ningbo 315211, China;

See next page for additional authors

Fluid Simulation of SPH Based on OpenCL

Abstract

Abstract: Fluid plays an important role in the Virtual Reality Scene. Physics-based fluid animation can enhance the reality of numerical simulation. The efficiency of fluid animation based on physical computing usually is not real-time rendering without parallel computing. Fluid simulation was proposed based on SPH using OpenCL, which could make full use of the data interoperability of OpenCL and the graphics API, and use the coordination work of CPU and GPU to increase the potential of the hardware; when physics computing, the surface particles and normals based on the Coulomb's law were solved out. The preliminary experiment shows that the method is hopeful in simulation a large scale fluid simulation.

Keywords

fluid simulation, parallel computing, OpenCL, SPH

Authors

Miaomiao Xiao, Liu Zhen, Jiabin Shi, Tingting Liu, Cuijuan Liu, and Bangquan Liu

Recommended Citation

Xiao Miaomiao, Liu Zhen, Shi Jiabin, Liu Tingting, Liu Cuijuan, Liu Bangquan. Fluid Simulation of SPH Based on OpenCL[J]. Journal of System Simulation, 2015, 27(4): 803-809.

第 27 卷第 4 期 系统仿真学报© Vol. 27 No. 4
2015 年 4 月 Journal of System Simulation Apr., 2015

基于 OpenCL 加速的 SPH 流体仿真

肖苗苗，刘箴，史佳宾，刘婷婷，刘翠娟，刘邦权
(宁波大学信息科学与工程学院，宁波 315211)

摘要：流体在虚拟现实场景中扮演重要角色，基于物理模型的流体动画可以增强数值仿真的真实感。但通常的基于物理模型的流体动画渲染，如果没有采用并行计算，不能达到实时模拟的效果。*使用异构计算平台 OpenCL 对 SPH (smoothed particle hydrodynamics)方法进行加速，充分利用了 OpenCL 与图形渲染 API 的数据互通性，并采用了 CPU 和 GPU 的协同工作的技术，能够发挥硬件潜力；同时，采用库伦定律和 SPH 方法计算表面张力和粒子法线，*初步的实验结果表明该方法有希望用于大规模流体仿真。

关键词：流体仿真；并行计算；OpenCL；SPH
中图分类号：TP391.9 文献标识码：A 文章编号：1004-731X (2015) 04-0803-07

Fluid Simulation of SPH Based on OpenCL

Xiao Miaomiao, Liu Zhen, Shi Jiabin, Liu Tingting, Liu Cuijuan, Liu Bangquan
(Faculty of Information Science and Technology, Ningbo University, Ningbo 315211, China)

Abstract: Fluid plays an important role in the Virtual Reality Scene. Physics-based fluid animation can enhance the reality of numerical simulation. The efficiency of fluid animation based on physical computing usually is not real-time rendering without parallel computing. *Fluid simulation was proposed based on SPH using OpenCL, which could make full use of the data interoperability of OpenCL and the graphics API, and use the coordination work of CPU and GPU to increase the potential of the hardware; when physics computing, the surface particles and normals based on the Coulomb's law were solved out.* The preliminary experiment shows that the method is hopeful in simulation a large scale fluid simulation.

Key words: fluid simulation; parallel computing; OpenCL; SPH

引言

基于物理的流体仿真是虚拟现实技术研究的热点及难点。基于物理的流体模型可以使仿真效果与现实中的现象相符；同时，也带来了较大的时间和空间的开销。所以在很长的一段时间中，基于物理的流体仿真都用于高精细度的离线渲染，而很难做到实时渲染。近年来，随着并行计算平台的发展，基于物理的流体动画终于可以达到实时的效果，从而运用到虚拟现实场景中去。

开放计算语言 OpenCL 是由 Khronos Groups 设计的异构计算标准。本文采用光滑粒子流体动力学(SPH)方法作为物理计算的数值方法，选用 OpenCL 作为数值计算的 API，以及 OpenGL 作为图形渲染的 API 进行流体仿真。不同于以往的研究者使用 GPU 完成所有的数值计算，本文使用 CPU 做仿真前网格初始化的预处理，以 CPU 与 GPU 协作的方式进行流体仿真。本文采用的流体仿真框架既能保证数值计算的准确性，又能保证流体仿真的实时性，并能为后续的渲染工作提供了准

收稿日期：2014-01-03 修回日期：2014-04-11；
基金项目：国家自然科学基金(61373068)；浙江省自然科学基金(LY13F020037)；宁波市科技计划项目(2013D10011,2014C50018)；高等学校博士学科点专项科研基金(No.20133305110004)；浙江省教育厅科研项目(Y201431792)；
作者简介：肖苗苗(1988-)，女，湖北潜江人，硕士生，研究方向为流体动画；刘箴(1965-)，男，辽宁省铁岭市人，博士，研究员，研究方向为计算机图形学、虚拟现实、情感计算。

确的表面粒子及其法线信息；证实了将基于 OpenCL 的异构计算应用到虚拟现实场景中的可行性。

1 相关工作

流体力学可以分为欧拉视角和拉格朗日视角，采用 SPH 数值计算方法可以从拉格朗日视角求解流体力学方程。文献[1]将这种数值计算的方法引入图形学，文献[2]使用 SPH 方法来模拟水面。文献[14-15]使用 SPH 方法模拟场景中的不可压缩流体。

利用图形渲染可编程管线来求解流体力学方程的方法很常见，文献[3]在 GPU 上实现了复杂边界三维实时流体模拟，他们利用图形渲染可编程管线来进行通用计算，将数据场的内容压缩为纹理，并在片元着色器中对纹理进行操作，这种方法受到图形渲染管线的局限，不能直接更新这一纹理，无法避免图形卡中数据的拷贝。

在异构计算框架 OpenCL 出现以前，占据通用并行计算主流的是 NVIDIA 公司为自己生产的 GPGPU 量身定做的 CUDA 框架[4]；国内外的很多流体仿真研究都使用这一框架进行数值计算，例如 2010 年，文献[5]采用了当时最新的 CUDA 框架，将 SPH 的全部模拟计算分配到 GPU 的流处理器中，充分发挥了图形卡的性能，得到了效率很高的仿真结果。之后文献[6]使用 OpenCL 框架，在 CPU 和 GPU 上实现了二维的混合尺度粒子模拟，CPU 和 GPU 共享数据内存，将大粒子之间的碰撞检测及计算和小粒子之间的碰撞检测及计算分别分配给 CPU 和 GPU，最后两者进行同步，完成渲染。

2 SPH 数值计算

SPH 方法最早由文献[7]提出。它是一种无网格方法，用有限个粒子来表示物理系统；每个位置上的物理属性等于它紧支域内粒子物理属性的加权平均。如公式(1)，位置 $\vec{x}$ 上的物理属性 $\phi(\vec{x})$ 为：

$$\varphi(\vec{x}) = \sum_j m_j \frac{\varphi_j}{\rho_j} W(\vec{x} - \vec{x}_j) \tag{1}$$

其中 W 是权值函数，即光滑核函数，它满足紧支性条件，即当 $\left|\vec{\mathbf{x}} - \vec{\mathbf{x}}'\right| > h$ 时，$W(\vec{\mathbf{x}} - \vec{\mathbf{x}}', h) = 0$，$h$ 称为光滑核半径。这样，积分的时候只需要考虑位置 $\vec{x}$ 周围的某个区域(核半径 h 以内)内的点的值。

在流体力学中，纳维-斯托克斯方程如公式(2)所示，可以采用 SPH 方法求解数值解。

$$\vec{f} = \rho \frac{D\vec{v}}{Dt} = -\nabla p + \rho\vec{g} + \mu\nabla^2\vec{v} \tag{2}$$

在本文的仿真过程中，根据 SPH 的原理，将流体用有限个质量相同的粒子来表示，构成 SPH 粒子系统。分析每个粒子的受力时，按照方程(1)，对每个粒子的领域内粒子的物理量进行求和；受力分析完成后，每个粒子都成为了独立的个体，即已知加速度，可以对单个粒子求解运动学公式，更新位置和速度。这样，本文以每个粒子作为一个 OpenCL 工作项，并行更新各个粒子的物理属性，这个过程将在第 3 节中详细讨论。

在 SPH 方法中，很重要的步骤是搜索每个粒子紧支域内的其他粒子：对于每一个粒子，要搜索出那些粒子与它的距离在其核半径 h 以内。若采用蛮力法，至少应计算 $N*(N-1)/2$ 次距离，其中 N 为粒子总数，这样，计算效率随着 N 的增加呈几何速率下降。解决这个问题的办法之一是先进行空间划分，再在小空间内进行搜索。在文献[8]中，NVIDIA 公司的 Simon Green 等人采用 CUDA 架构，将空间划分为等距的六面体单元来构造哈希表，在每个粒子所在的单元周围的 27 个单元中搜索相邻粒子。本文采用空间划分的方法，将每个六面体的长度设为光滑核半径，搜索过程中的并行也是基于 OpenCL 的。

3 OpenCL 与 OpenGL 的数据互通

早期 SPH 数值计算的并行加速多是利用图形渲染可编程管线。该方法虽然充分使用了 GPU 的并行计算能力，但创建三维纹理、渲染到纹理等过程会在图形卡内存中执行大量的复制操作，造成不必要的开销。

本文的流体仿真最终目的是应用到虚拟现实

场景中，即不仅要进行数值计算，更重要的是得到可视化效果。SPH 粒子的位置信息为渲染对象，再将这些信息包装为图元，输入着色器，使用光照模型进行表面着色，最终得到具有真实感的流体动画。如果整个过程完全在 GPU 上完成，动画每一帧的步骤如图 1 所示。

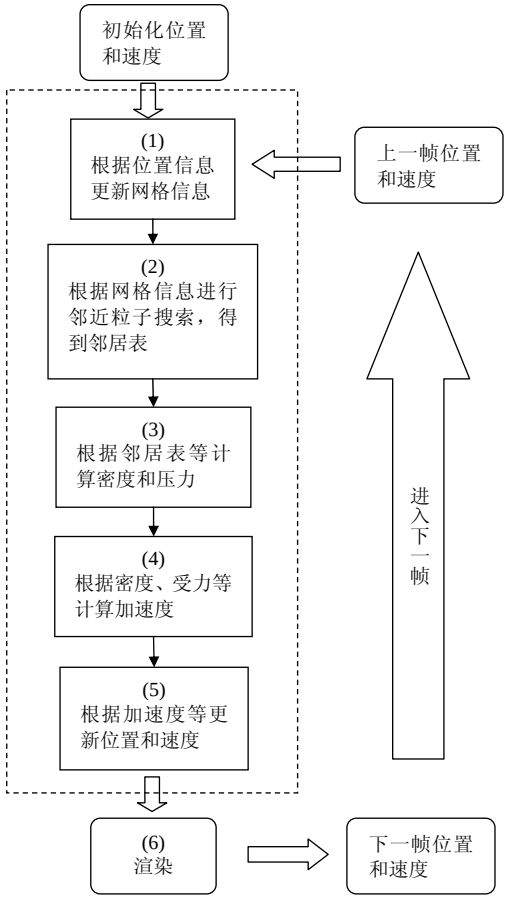


图 1 GPU 上的流体仿真步骤

虚线方框中是每一帧必须执行的仿真步骤，可以看出，它们是数据相关的，即，这些步骤不能乱序执行。

OpenCL 和 OpenGL 可以实现高效数据互通，即 OpenCL 的计算核能够和图形渲染程序共享图形卡内存。

本文由图形程序来创建通用计算中的内存对象，如粒子位置、速度等；在每个时间步中，由 OpenCL 内核程序来直接更新图形程序的渲染对象，这样可以避免主机内存提交顶点信息到设备内存、从设备内存拷贝到主机内存等复制操作。本文实验的资源分配的逻辑图如图 2 所示。

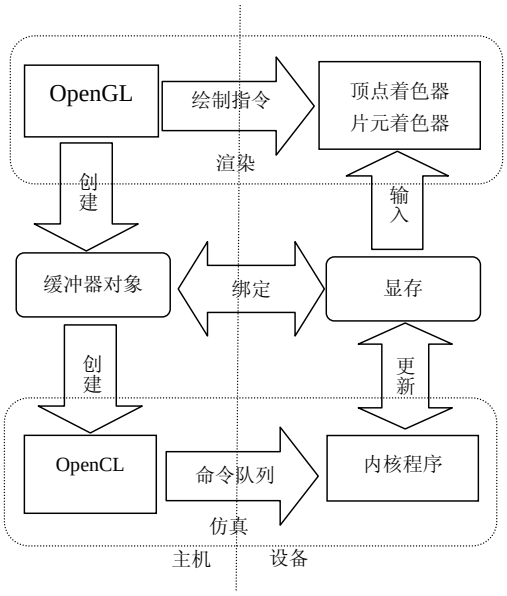


图 2 资源分配逻辑图

初始化系统时，OpenGL 为粒子的位置和速度创建顶点缓冲区对象(vertex buffer object, VBO)，并提交位置和速度的初值到图形卡顶点缓冲区，这次数据传输之后的整个仿真过程将不再有主机和设备缓冲区之间的数据传输操作。OpenCL 的速度、位置缓冲区对象直接创建于 OpenGL 顶点缓冲区，也就是它和 OpenGL 共享同一块设备内存，在每一帧读取每个粒子速度和位置信息、并直接对其进行写入，更新速度和位置。

在仿真的过程中，每个粒子作为一个 OpenCL 工作项，所有的工作项运行相同的 OpenCL 内核代码，实现并行计算。在每帧结束之前进行同步，以保证每一帧所有粒子都完成更新后再进入下一帧。

4 CPU 与 GPU 协同工作

如第 3 小节所述，图 1 中虚线方框中的各步骤必须按顺序执行，比较常见的加速方法是将这些步骤全部交给 GPU 并行处理，这样做的理由是，将所有的数据都存放在同一设备的缓冲区中，可以避免数据在不同设备中的拷贝与同步。按照图 1 所示

的流程，本文首先在 GPU 上模拟了 65 536 个水粒子在边长为 1 m 的场景中的自由落体运动，仿真的平均耗时为 36.08 ms；而各个步骤的占总耗时的比例如图 3 所示。

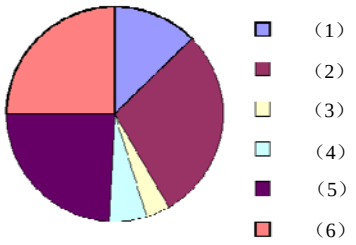


图 3　仿真中各步骤耗时饼状图(步骤编号与图 1 对应)

由图 3 可以看出，仿真中最耗时的步骤为邻近粒子搜索的过程，而渲染步骤所耗时间也有 9.01 ms 之多。渲染步骤不能和步骤(5)同时进行，必须等所有的位置信息都更新完后再执行，即在(5)之后各工作项应进行同步。

为了进一步提高效率，根据以上实验结果，本文的方法是将步骤(1)交给 CPU 来完成。如图 4 所示。

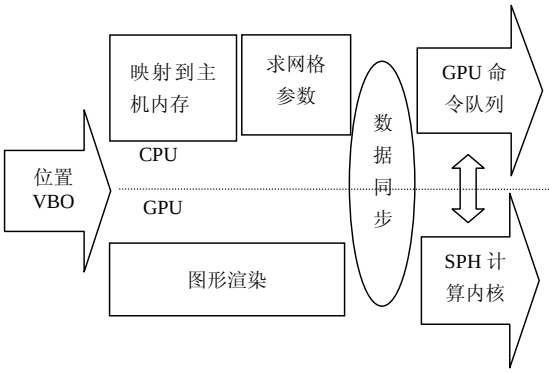


图 4　网格计算与图形渲染在 CPU 和 GPU 上并行

这样做的原因有两点：第一是完成步骤(1)只需要粒子的位置信息，而它的执行结果也是对应每个粒子，根据公式(3)得出一个网格序号，这样设备之间的数据传输与同步耗费相对较小。

$$GridID[i] = floor[(pos[i] - coord[i])/h] \quad (3)$$

公式(3)中，i=0,1,2，分别表示三维的坐标轴，GridID 表示该粒子所在的网格序号，floor()为向下取整运算，pos 指粒子这一时刻所在位置的坐标，coord 指网格划分的最小边缘的坐标值，h 为光滑核半径，也是网格每个单元的边长。

第二是因为步骤(5)更新完当前帧的位置信息后，渲染模块直接使用图形卡缓冲区中的位置信息进行渲染，渲染步骤不会改变粒子的位置。所以同时将位置信息拷贝到 CPU 设备内存，并完成步骤(1)。这两个操作是没有数据冲突的，可以实现设备并行。

步骤(1)之后，两个设备实现数据同步，并进行之后的步骤。与图 1 对比，CPU 和 GPU 协同实现的流程图如图 5 所示。

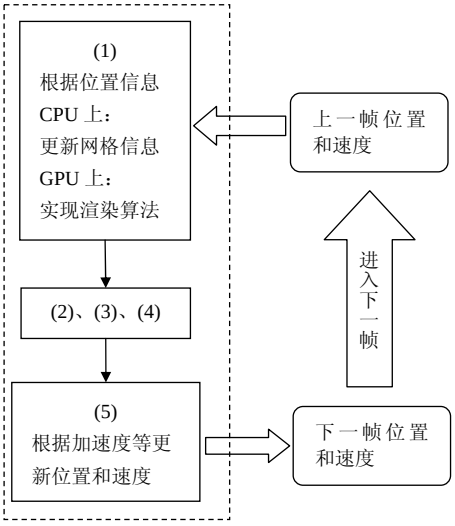


图 5　GPU 和 CPU 协同流体仿真步骤(与图 1 的对比)

5　粒子法线和表面张力

对于流体动画的渲染来说，表面粒子和法线是重要的渲染信息，首先，在物理模型中加入表面张力更符合真实的流体运动；其次，实现光照模型必须知道法线。由于仿真和渲染往往作为 2 个独立的研究部分，很多文献中，渲染仅限于已知位置的三维数据场，在此基础上再提取表面粒子或计算法线。本文提出 1 种在 SPH 数值计算的过程中同时计算出每个粒子法线的思想，由 OpenCL 计算法线，之后提交给 OpenGL 进行渲染。

文献[9]中指出，粒子系统的表面粒子及其法

线可以由计算排斥力的方法准确地找到。本文选用库伦定律来计算粒子之间的排斥力。

$$\vec{n}_i = \sum_j \frac{\rho_j}{\left|\vec{r}_i - \vec{r}_j\right|^3}(\vec{r}_i - \vec{r}_j)W_{ij}(\vec{r}_i - \vec{r}_j, h) \tag{4}$$

其中每个粒子的密度必定会在 SPH 数值计算中求出，而 W 是光滑核函数。

如果粒子 i 在流体内部，它所受周围粒子的排斥力应达到平衡，$|\vec{n}_i|$ 趋近于 0；反之，$|\vec{n}_i|$ 越大，粒子 i 越可能处于流体表面；特别地，如果粒子紧支域内没有其他粒子，它所受排斥力计算为 0，则将这个粒子视为飞溅的液滴，将其 $|\vec{n}_i|$ 的值设为 $(0.0,-m,0.0)$，m 为大于所有 $|\vec{n}_i|$ 的正数。$normalize(\vec{n}_i)$ 是粒子 i 的法线。

得到表面粒子后，本文使用公式(5)来计算表面张力：

$$F_{surface}(\vec{x}) = K\sum_j \frac{\vec{n}_j}{\rho_j} W(\vec{x} - \vec{x}_j) \tag{5}$$

其中 K 是一个常数，即表面粒子所受的表面张力与表面法线的方向相同且大小与法线的模线性相关。

这样，在 SPH 数值计算的步骤中，求出粒子密度后，只需要加上一个积分域内求和的步骤，就能得出表面张力以及渲染所需的法线，这显然比在渲染的步骤中，单独根据三维数据场求法线效率高。

本文的实验证明，由公式(4)求出的法线为正确的流体粒子法线，可以提高后续渲染过程的效率。

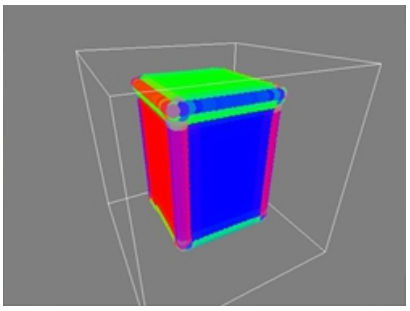
6 结论

本文的实验全部在一台 Intel Core i7-950(3.07GHz)CPU,8GB 内存，NVIDIA Quadro FX 5800 图形卡的工作站上完成。虚拟空间的尺寸为 1.0×1.0×1.0，模拟立方体水柱自由落体的场景。

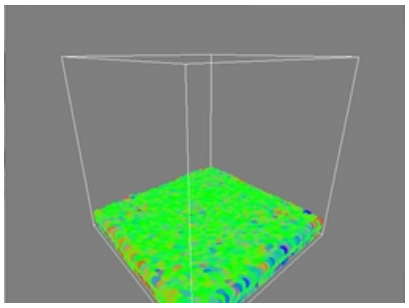
采用第 5 节中所述的方法计算法线，并将法线向量按公式(6)进行可视化的结果如图 6 所示。

$$\overrightarrow{color} = abs(normalize(|\vec{n}_i|)) \tag{6}$$

图(a)为初始时正方形水柱的法线，可见的三个面分别为红、绿、蓝色，这是因为这三个面的法线分别与可视化空间的 x 轴、y 轴和 z 轴平行；图(b)为趋于平静的水面的法线图，可以看出，表面的那层粒子法线基本为绿色，这是因为它们的法线与 y 轴平行，指向正上方。实验充分证明了第 5 节所述的法线算法正确性。



(a)



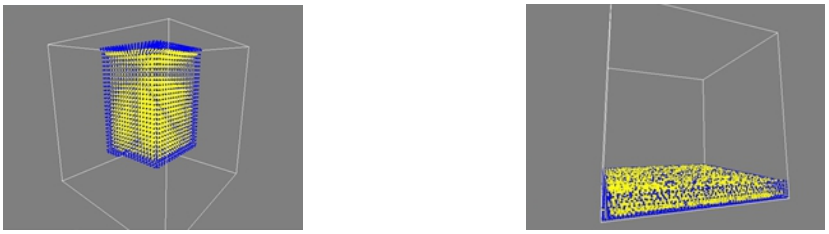
(b)

图 6 法线向量可视化

图 7 为本实验求出的表面粒子示意图，蓝色表示表面粒子，黄色表示内部粒子；(a)为初始时正方形水柱的表面粒子，(b)为趋于平静的水面的表面粒子状态；而(c)和(d)为流体冲击地面、溅起水花的瞬间，这时流体具有较复杂的表面。

之后，将粒子渲染成真实感水面时，采用的渲染方法为屏幕空间法[10]，渲染效果如图 8 所示。

在仿真效率方面，以往的研究工作已经进行了大量的 CPU 和 GPU 仿真效率对比，例如文献[13]，分别实现了基于 OpenMP 的多线程 CPU 仿真以及基于 CUDA 的 GPU 并行加速仿真，并进行了详细对比，但未考虑到数据场的渲染效率带来的影响。



(a) (b)

(c) (d)

图 7 流体表面粒子可视化(蓝色表示表面粒子)

(a) (b)

(c) (d)

图 8 屏幕空间法的渲染效果

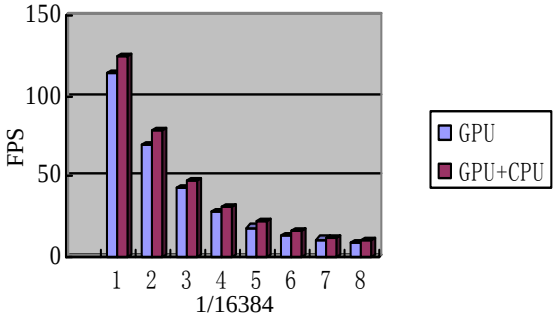


图 9 两种实现方式的平均帧速率对比

本文将第 4 节所述，GPU, CPU 协同的方法与 GPU 单独仿真的方法也进行了对比，粒子数目为 $16348 * i(i = 1, ..., 8)$ 的平均帧速率如图 9 所示。

可见，在本文的实验平台上，GPU 和 CPU 协同的方法可以进一步提高帧速率。

7 总结与展望

本文主要研究了基于 OpenCL 的 SPH 仿真，主要在物理计算上利用了 GPU 和 CPU 协同加速，获得了一定的提速效果。但是，这个结果和本文的实验的硬件环境是相关的。在不同的平台上，各设备的计算能力不同，且设备之间的数据传输效率也

不同，基于这个原因，本文首先分析了单独在 GPU 上仿真时各步骤的耗时情况，然后才决定将多少工作交给 CPU 来执行。另一方面，本文采用了新的方法，提取出了表面粒子以及法线，且提出了表面粒子和法线可以在仿真过程中计算出来，这一观点不同于以往单独渲染数据场的方法，为进一步研究如何高效和真实地绘制流体奠定了基础。

本文的不足之处是，没有在算法本身上改进 SPH 算法，例如其中计算量最大的邻居搜索的步骤，应还能在并行算法上进行优化。在流体渲染方面，本文实验中的方法使用的是已有的屏幕空间法，如何应用第 5 节中提出的法线算法和表面粒子提取算法是本文要进一步研究的工作。

本文的实验平台是多核 CPU 和单独的 GPU，它具有局限性。例如文献[11]已经实现了多 GPU 协作的仿真；而随着 APU[12]架构的出现，新一代 CPU 和 GPU 可集成在同一芯片上，它们可以共享数据缓存，比起传统架构来说，可以节省设备缓存之间的数据传输时间，这样，GPU 和 CPU 协同工作的研究显得更加重要，这也正是本文的研究意义。

参考文献:

[1] Stam J, Fiume E. Depicting fire and other gaseous phenomena using diffusion processes. Computer Graphics Proceedings, Annual Conference Series [C]// ACM SIGGRAPH, Los Angeles, USA. USA: ACM, 1995: 129-136.

[2] Muller M, Charypar D, Gross M. Particle based fluid simulation for interactive applications [C]// Proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Computer Animation, San Diego, USA. USA: ACM, 2003: 154-159.

[3] 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.

[4] Rob Farber. CUDA Application Design and Development [M]. San Francisco, USA: NVIDIA Corporation, 2011.

[5] 陈曦, 王章野, 何戡, 延诃, 彭群生. GPU 中的流体场景实时模拟算法 [J]. 计算机辅助设计与图形学学报, 2010, 22(3): 396-405.

[6] Benedict R. Gaster, Lee Howes, David R Kaeli,. Heterogeneous Computing with OpenCL [M]. 1st Edition, USA: Advanced Micro Devices, Elsevier, 2012: 197-209.

[7] Gingold R A, Monaghan J J. Smoothed Particle Hydrodynamics-theory and application to non-spherical stars [J]. Mon. Not. R. Astron. Soc (S0035-8711), 1977, 181: 375-389.

[8] Simon Green. Particle Simulation using CUDA. [R]. 2701 San Tomas Expressway, Santa Clara, CA, USA 95050: NVIDIA Corporation, July 2012.

[9] Hubert Nguyen. GPU Gems 3 [M]. USA: Addison-Wesley Professional, 2007: 93-111.

[10] Wladimir J van der Laan, Simon Green, Miguel Sainz. Screen Space Fluid Rendering with Curvature Flow [R]. Radisson Hotel Boston, USA: Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games.

[11] Jose Ricardo da S Junior, Mark Joselli. An architecture for real time fluid simulation using multiple GPUs [C]// Computing Tracking. SBC. Proceedings of SB Games 2012: 93-100.

[12] Benedict R Gaster, Lee Howes, David R Kaeli. Heterogeneous Computing with OpenCL [M]. 1st Edition, USA: Advanced Micro Devices, Elsevier, 2012: 63-64.

[13] Jose M Dominguez, Alejandro J C Crespo, Moncho Gomez-Gesteira. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method [J]. Computer Physics Communications (S0010-4655), 2013, 184(3): 617-627.

[14] Nadir Akinci, Markus Ihmsen, Gizem Akinci. Versatile Rigid-Fluid Coupling for Incompressible SPH [C]// ACM Transactions on Graphics (S0730-0301). USA: ACM, 2012, 31(4): 62: 1-62: 8.

[15] Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Implicit Incompressible SPH [C]// IEEE Transactions on Visualization and Computer Graphics (S1077-2626). USA: IEEE, July, 2013.