Journal of System Simulation

Volume 28 | Issue 10 Article 25

8-13-2020

Real Time Fluid Simulation Framework Based on DirectCompute

Liu Lu
1. Faculty of Electrical Engineering and Computer Science of Ningbo University, Ningbo 315211, China;;

Liu Zhen
1. Faculty of Electrical Engineering and Computer Science of Ningbo University, Ningbo 315211, China;;

Gaoqi He
2. Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China;;

Chen Tian
3. Shanghai Dianji University, Shanghai 200000, China;

See next page for additional authors

Real Time Fluid Simulation Framework Based on DirectCompute

Abstract

Abstract: *A real time fluid simulation framework based on DirectCompute acceleration was proposed* to solve problems of low simulation efficiency that caused by the increasing fluid particle size and long processing time that caused by the complex solid-liquid boundary. *In the simulation, a grid local collision detection method was adopted to speed up the search for the solid-liquid boundary. Fluid calculation, collision detection and surface reconstruction were computed parallel in the GPU with DirectCompute technology to accelerate the SPH fluid simulation. It could optimize the storage structure of fluid simulation, and save the storage space.* Experimental results show that the framework can effectively improve the fluid simulation speed with more complex static scene, and can show the details of the fluid spray.

Keywords

fluid simulation, smoothed particle hydrodynamics, collision detection, GPU acceleration

Authors

Liu Lu, Liu Zhen, Gaoqi He, Chen Tian, Tingting Liu, and Cuijuan Liu

Recommended Citation

Liu Lu, Liu Zhen, He Gaoqi, Chen Tian, Liu Tingting, Liu Cuijuan. Real Time Fluid Simulation Framework Based on DirectCompute[J]. Journal of System Simulation, 2016, 28(10): 2467-2475.

第 28 卷第 10 期 系统仿真学报© Vol. 28 No. 10
2016 年 10 月 Journal of System Simulation Oct., 2016

一种基于 DirectCompute 加速的实时流体仿真框架

刘璐[1]，刘箴[1]，何高奇[2]，陈田[3]，刘婷婷[1]，刘翠娟[1]
(1. 宁波大学信息科学与工程学院 宁波 315211；
2. 华东理工大学计算机科学与工程系 上海 200237；3. 上海电机学院 上海 200000)

摘要：针对流体仿真中流体粒子规模增大导致的仿真效率较低和较复杂固液边界处理耗时较长的问题，*提出了一种基于 DirectCompute 加速的实时流体仿真框架，仿真中采用一种基于网格的局部碰撞检测方法来加快搜索固液边界，使用 DirectCompute 技术将流体计算、碰撞检测和流体表面重构部分放入 GPU(Graphics Processing Unit)中并行计算来加速 SPH(Smoothed Particle Hydrodynamics)流体仿真，并利用其特性优化了 SPH 流体仿真的存储结构，节省了存储空间。*实验结果证明，该框架有效提高了带有较复杂静态场景的流体仿真速度，并能展现出流体水花细节。

关键词：流体仿真；光滑流体动力学方法；碰撞检测；GPU 加速
中图分类号：TP391.9 文献标识码：A 文章编号：1004-731X (2016) 10-2467-09

Real Time Fluid Simulation Framework Based on DirectCompute

Liu Lu[1], *Liu Zhen*[1], *He Gaoqi*[2], *Chen Tian*[3], *Liu Tingting*[1], *Liu Cuijuan*[1]
(1. Faculty of Electrical Engineering and Computer Science of Ningbo University, Ningbo 315211, China; 2. Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China; 3. Shanghai Dianji University, Shanghai 200000, China)

Abstract: *A real time fluid simulation framework based on DirectCompute acceleration was proposed* to solve problems of low simulation efficiency that caused by the increasing fluid particle size and long processing time that caused by the complex solid-liquid boundary. *In the simulation, a grid local collision detection method was adopted to speed up the search for the solid-liquid boundary. Fluid calculation, collision detection and surface reconstruction were computed parallel in the GPU with DirectCompute technology to accelerate the SPH fluid simulation. It could optimize the storage structure of fluid simulation, and save the storage space.* Experimental results show that the framework can effectively improve the fluid simulation speed with more complex static scene, and can show the details of the fluid spray.

Keywords: fluid simulation; smoothed particle hydrodynamics; collision detection; GPU acceleration

引言

基于物理的流体仿真作为虚拟现实技术中的重要部分，一直是相关领域中的研究热点。实时性高且真实感强的流体模拟是增强虚拟现实系统生动性的重要内容，也是未来流体仿真技术研究的重点[1]。SPH 方法作为基于物理的流体计算中的经典方法，近年来得到了广泛的关注，但是 SPH 流体与环境的交互需要耗费大量的时空开销，是其难以推广应用的瓶颈。随着计算机图形硬件的不断更新换代，基于 GPU 的通用计算能力显示出了它的优越性。DirectCompute 作为 DirectX 中专门用于通用计算的 API(Application Programming Interface)，

收稿日期：2016-05-31 修回日期：2016-07-09；
基金项目：国家自然科学基金(61373068)，宁波市科技计划项目(2015A610128，2015C50053，2015D10011，2016D10016)，高等学校博士学科点专项科研基金(20133305110004)；
作者简介：刘璐(1991-)，女，重庆，硕士，研究方向为虚拟现实。

使 DirectX 成为了一个通用计算平台，这套由微软主导开发的标准使其在 Windows 平台上具有更大的计算优势。目前，利用 DirectCompute 技术完成全 GPU 流体仿真的研究并不多见，本文用 DirectCompute 完成流体仿真计算、碰撞检测和表面重构算法，充分发挥了 GPU 的并行计算优势。对于复杂固体边界的碰撞检测，除了 GPU 并行计算能力可以利用外，往往通过八叉树和层次包围盒等空间分割方法进行局部的粒子搜索，但其数据结构往往较为复杂，其构造开销巨大。本文提出了一种基于网格的局部碰撞检测方法，用基于链表的邻居粒子网格将场景模型顶点划分，通过 AABB 包围盒进行粗碰撞检测，再根据流体位置进行精细碰撞检测，该方法能快速搜索到可能与流体发生碰撞的局部边界，有效提高了碰撞检测的速度。此外，流体的表面重建方法由于需要构造大规模的标量场，往往需要大量时空消耗，传统 CPU 上的表面重构方法无法达到实时，本文用 DirectCompute 实现表面重构算法，节省了部分存储空间，并实现了实时的流体表面重建。

1 相关工作

基于物理的流体模拟方法主要为欧拉法和拉格朗日法[2]。欧拉法基于网格的特点使其更适合于大规模流体模拟，拉格朗日法将连续的流体离散为粒子系统，容易捕捉流体细节，且更容易适应实时性要求。本文采用的光滑流体动力学(SPH)方法是基于拉格朗日观点的方法。

Müller[3]提出了利用光滑流体动力学(SPH)方法模拟流体，采用不同的光滑核函数分别对流体粒子的密度、压强、加速度等属性进行计算，文献中还未考虑流体模拟效率问题。随着图形硬件的不断发展，GPU 成为提升流体模拟速度的重要手段。文献[4] 中利用 CUDA(Compute Unified Device Architecture)的并行特性，将基于粒子的流体模拟移植到 GPU 中，文中模拟空间被划分为规则网格，使用基数排序实现了邻居粒子搜索，提高了仿真效率。文献[5]在文献[4]的基础上，在 GPU 上实现了 SPH 三维流体。文献[6]采用多个 GPU，改进粒子邻域搜索方法，修正了单个 GPU 粒子不能正确找到边界的问题。文献[7]用 Z-order 排序方法改进了邻域搜索，提高了 GPU 的命中率。上述计算方法均需要额外开辟空间存放粒子索引并进行索引的重新排序，造成一定的空间消耗，本文借鉴文献[8]中基于链表的邻居粒子搜索方法进行粒子搜索，在仿真速率不变的情况下节省存储空间。

流固之间的碰撞检测要求流体粒子能检测到周围的固体边界，防止流体进入固体内部，Monaghan[9]在固体边界上固定一组虚粒子，对流体粒子施加强排斥力，防止流体进入边界。文献[10]改进了 Monaghan 的方法，将虚粒子固定在固体边界表面和内部，并保证密度和压力与实体粒子相同，能进一步防止流体进入边界。文献[11]通过对边界粒子质量进行差值估计，避免了流体穿透边界。文献[12]通过定义场景语义，高效处理了较大规模的流体场景。上述方法需要对模型边界进行复杂的预处理，本文从流体粒子与三角面片的碰撞检测角度出发，借鉴基于空间划分的思想，根据流体粒子位置动态地构造包围盒，减少了搜索模型三角面片的范围，提高了碰撞检测的效率。

流体表面重构中最经典的方法之一为 marching cube 方法，文献[13]最早提出该方法用于模型的表面重构，Müller 提出了颜色场的方法提取流体表面，但提取的表面凹凸不平。文献[14]采用基于距离场的方法构造标量场，使流体表面更加光滑。文献[15-16]均改进了标量场的计算方法，进一步提高流体表面的光滑程度。对于流体渲染方面，文献[17]通过为不同类物质建立吸收散射模型，模拟了浑浊水体的效果。本文借鉴文献[14]中构造标量场的方法，将标量场构造和表面重构过程放入 GPU 中计算，采用可伸缩缓冲区，节省了存储空间，渲染上采用菲涅尔定律和环境贴图完成流体渲染。

2 SPH 流体仿真思路

本文利用 DirectX11 的统一渲染框架实现可实时的流体仿真效果，流体仿真的框架如图 1 所示。

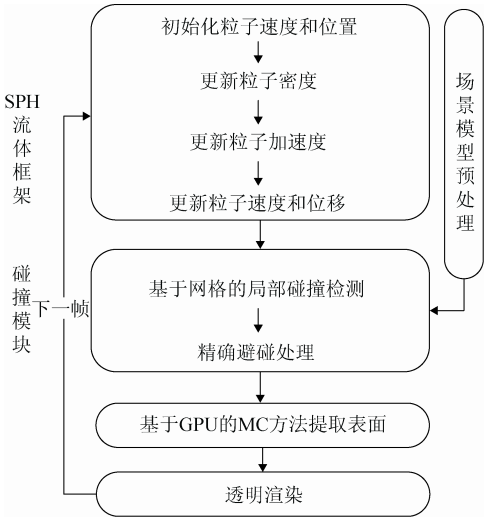


图 1 流体仿真框架

在流体仿真阶段，由于 SPH 算法的特点适合于并行计算，利用 DirectCompute 并行实现流体数值计算，本文借鉴文献[8]中基于链表的邻域粒子搜索方法，进一步优化了存储结构。在碰撞检测阶段，本文将场景模型所在空间划分为均匀网格，将场景模型中三角面片的顶点采用链表结构将同一网格中的顶点粒子串成链表，存储于 GPU 中。碰撞检测时，可根据粒子位置动态形成 AABB 包围盒，利用基于网格的方法检索包围盒中的模型顶点和三角面片，再进行碰撞点计算和避碰处理。本文用 DirectCompute 实现全 GPU 的表面重建算法，利用其新特性节省了数据的存储空间,流体的透明渲染采用菲涅尔公式和环境贴图实现。通过上述框架，可实现具有较复杂静态场景下的实时流体仿真，且仿真速度得以有效提升。

3 SPH 流体仿真的实现方法

SPH 方法不同于基于网格的欧拉法，它是从拉格朗日观点出发的一种无网格方法。3.1 节对 SPH 物理原理作简单介绍，3.2 节对 DirectCompute 进行了简介，3.3 节详细描述利用 DirectCompute 技术可节省存储空间。

3.1 SPH 算法的物理原理

SPH 算法是一种快速的流体模拟算法，它通过对连续的流体介质采样，用承载了物理属性的相互作用的质点组对流体进行描述。在流体仿真空间中，任意一点 $\boldsymbol{x}$ 处的物理属性 $A(\boldsymbol{x})$为半径 h 范围内同种属性值的加权平均，其计算公式为：

$$A_i(\boldsymbol{x}) = \sum_j A_j \frac{m_j}{\rho_j} W(\boldsymbol{x} - \boldsymbol{x}_j, h) \tag{1}$$

式中：$A_i(\boldsymbol{x})$为第 i 个采样点处的物理属性；h 为光滑核半径；W 为光滑核函数；$\boldsymbol{x}_j$ 为核半径范围内的采样点；ρ_j 为第 j 个采样点处的密度。SPH 方法中利用公式(1)求解流体的密度、压强和粘度。本文对流体密度、压强、粘度计算采用的核函数均与文献[3]中的核函数相同。将采用核近似函数表示的压力项、粘度项以及外力项带入纳维-斯托克斯方程，结合牛顿第二定律，可求解流体的加速度(2)，

$$\boldsymbol{a_i} = \boldsymbol{g} - \frac{\nabla \boldsymbol{p}}{\rho_i} + \frac{\mu \nabla^2 \boldsymbol{v}}{\rho_i}. \tag{2}$$

式中：$\boldsymbol{a_i}$ 表示粒子 i 的加速度；ρ_i 为粒子 i 的密度；$\boldsymbol{g}$ 为重力加速度；$\boldsymbol{p},\boldsymbol{\mu},\boldsymbol{v}$ 分别表示流体的压强、粘度系数和速度。将流体压力项和粘度项的光滑核函数带入公式(2)，得到最终加速度公式(3)，

$$\begin{aligned}\boldsymbol{a_i} = \boldsymbol{g} + m\frac{45}{\pi h^6}\sum_j \left(\frac{\boldsymbol{p_i} + \boldsymbol{p_j}}{2\rho_i\rho_j}(h-r)^2 \frac{\boldsymbol{r_i} - \boldsymbol{r_j}}{r}\right) + \\ m\mu\frac{45}{\pi h^6}\sum_j \frac{\boldsymbol{u_j} - \boldsymbol{u_i}}{\rho_i\rho_j}(h-r).\end{aligned} \tag{3}$$

式中：$\boldsymbol{p_i}$ 为粒子 i 受到的压强；$\boldsymbol{r_i}$ 为粒子 i 的位置；r 表示两个流体粒子之间的距离；$\boldsymbol{u_i}$ 为粒子 i 的速度；其余符号说明同公式(1)，(2)。根据加速度以及流体粒子的初始速度和位置，可以不断更新粒子速度和位移。

3.2 DirectCompute 特性简介

DirectCompute 作为微软开发的一款用于 GPU 通用计算的 API，其在并行计算上的性能与 CUDA

和 OpenCL 可分庭抗礼，CUDA 作为一款成熟的并行计算架构，却以半开放的方式供用户使用，OpenCL 虽然灵活强大，编程难度却相对较高。与其它通用 GPU 通用计算框架相比，DirectCompute 具有如下优势：

1) DirectCompute 作为 DirectX 的一部分，与 DirectX 中其它渲染着色器有着一定的区别和联系。它不属于渲染着色器的某一级，因此可以将流体仿真中可并行的复杂计算随时利用 DirectCompute 进行计算，同时又可以与 DirectX 中其它图形资源进行有效互操作。本文将可与渲染着色器分离的可并行计算部分用 DirectCompute 计算，并用 GPU 缓存存储数据，以便与其它渲染管线进行互操作。

2) DirectCompute 不同其它渲染着色器的并行处理方式，它不需要将数据和线程完成一对一的映射，一个线程可处理多个数据，在应用程序中可根据处理数据的大小直接分配线程数。

3) DirectCompute 允许无序访问 GPU 缓存，因此可利用其时序性特点来更新数据，避免申请额外缓存空间用于数据交换。

3.3 用 DirectCompute 实现 SPH 算法

由于采用 SPH 算法时，流体粒子物理属性的计算之间并不存在相关性，利用 GPU 并行计算可以快速模拟流体运动。本文在进行流体计算时，仅在 CPU 中进行流体粒子和位置初始化，其余计算均放入 GPU 中进行，省去了与 CPU 的数据交换，且利用 DirectCompute 的无序访问特性节省了存储空间。

以往的 SPH 算法流程中，需要构建 hash 表来存储流体粒子索引，将其与粒子所属的网格对应起来，在每一帧中都需要根据 hash 表对流体粒子进行排序，这种方式需要开辟额外的空间存储 hash 表和网格的索引值。

本文借鉴文献[8]的思想，采用邻居粒子链表来存储流体粒子。文献[8]中算法的基本流程为，首先将流体模拟空间划分为边长为光滑核半径的规则网格，申请与规则网格的个数相等的粒子个数缓存和粒子头结点缓存；然后申请存储流体粒子位置和密度的缓存空间，为了节约缓存空间，文献中利用粒子位置缓存的 w 分量来存储与当前粒子相邻的下一个粒子的索引，利用速度缓存的 w 分量存储粒子密度；最后在 GPU 中利用互斥操作来实现邻居粒子链表的并行化构建。这种方式中，流体模拟需要开辟 7 个不同的缓存对象，其中粒子速度缓存和位置缓存分别需要两个，用于交换和更新粒子的速度和位置信息。

与这种方法不同的是，本文在流体模拟阶段仅需要 5 个缓存对象，其中粒子速度缓存和粒子位置缓存分别只需要一个，这是由于 DirectCompute 提供了无序访问资源(UAV)，UAV 的特性是允许多个线程无序地读写缓存资源，只要更新之前的数据在后续处理中不再被利用，用好 UAV 的时序性，就可以避免开辟多块缓存来进行数据交换，仅用单块缓存便能实现数据的更新。SPH 流体算法中，针对流体的更新恰好可以利用这种特性，流体在每一次更新时，前一帧的数据不需要在后期利用，实现了存储空间的有效缩减。

本文与前述两种方法的存储空间大小进行简单对比，假定模拟空间大小相等，由于邻居粒子链表法将 hash 搜索法中的粒子密度存储于速度缓存的 w 分量，可节省密度的所占的空间。本文将邻居粒子链表存储空间进一步缩减，经过粗略分析，由图 2 可以看出三种方法的空间大小对比，其中蓝色部分为三种方法至少需要的空间大小，图中横坐标中数字表示一个单位长度，比如 8 表示 8 个单位长度。

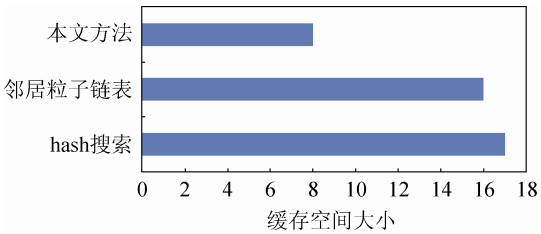


图 2 缓存空间大小对比图

4 碰撞检测

固液耦合在流体模拟中占有很大的时间开销，

目前存在多种用于处理复杂固液边界的算法。其中，基于虚粒子的流固耦合方法需要复杂的预处理过程；基于三角面片的碰撞检测方法中，传统的方法需要遍历所有流体粒子和三角面片，利用层次包围盒、八叉树等空间数据结构的算法需要在预处理阶段生成复杂的空间树形结构，且具有嵌套递归的特性，构造的开销比较大。本文提出一种新的碰撞检测方法，在预处理阶段将模型按照邻居粒子链表方法重新存储顶点，碰撞检测时，首先构造 AABB 包围盒进行粗略碰撞检测，然后利用流体粒子前后两帧的位置动态地构造局部的空间包围盒，根据邻居粒子链表搜索包围盒中的固体顶点及其所属的三角面片，最后进行碰撞点计算和避碰计算。该方法预处理过程相对简单，且碰撞检测均在 GPU 中实现并行化，能有效地加速碰撞检测过程。

4.1 模型的预处理

假定一个场景模型共有 N_{face} 个三角面片，那么在缓存中场景模型包含的顶点数为 3. N_{face}。在预处理阶段，将模型的顶点当成粒子，构建邻居粒子链表。为了构建邻居粒子链表，需要找到模型所处的空间包围盒，该包围盒通过模型包围盒按照一定比例缩放获得。将包围盒空间分成均匀的空间网格，网格的分辨率可根据碰撞检测的精确度进行调整，本文中网格的分辨率为 $h/2$，图 3 用二维模型说明包围盒和邻居粒子链表的建立过程。

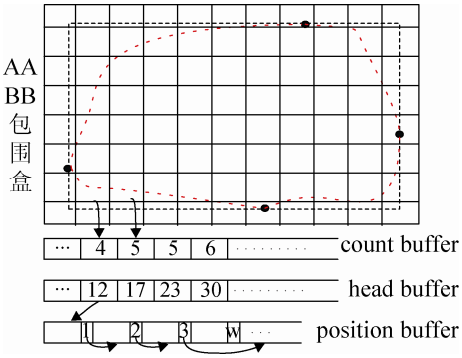


图 3 模型预处理示意图

图中虚线黑色长方形框为模型包围盒，实线黑色长方形框为构建网格的 AABB 包围盒，将包围盒划分为如黑色网格所示的规则网格，countbuffer 中记录着每个网格的顶点粒子个数，headbuffer 中记录每个网格中模型粒子链表的头结点索引，positionbuffer 中记录所有顶点粒子的位置和同一网格中下一个顶点的索引。经过预处理之后，能很快地根据顶点粒子的位置搜索到其相邻网格中所包含的顶点粒子，从而找到包含该顶点的三角形面片。

4.2 碰撞检测计算

本文中碰撞检测算法分为两个阶段：粗略碰撞检测阶段和精细碰撞检测阶段。

粗略碰撞检测阶段中，在流体模拟空间范围内构造一个略大于固体模型的 AABB 包围盒，即碰撞包围盒。计算流体粒子的位移时，如果检测到流体粒子当前帧的位置进入碰撞包围盒内，则该流体粒子就进入到精细碰撞检测阶段，否则，不做进一步检测。碰撞包围盒构造方式如下，

$$\begin{cases} \boldsymbol{V}_{\text{AABB\_min}} = \boldsymbol{V}_{\text{model\_min}} - d \cdot \boldsymbol{D}_{\text{distance}}. \\ \boldsymbol{V}_{\text{AABB\_max}} = \boldsymbol{V}_{\text{model\_max}} + d \cdot \boldsymbol{D}_{\text{distance}}. \end{cases} \tag{4}$$

其中 $\boldsymbol{V}_{\text{AABB\_min}}$ 和 $\boldsymbol{V}_{\text{AABB\_max}}$ 分别为碰撞包围盒的最小顶点和最大顶点，$\boldsymbol{V}_{\text{model\_min}}$ 和 $\boldsymbol{V}_{\text{model\_max}}$ 分别为固体模型的最小顶点和最大顶点，d 为一个缩放参数，用于调整碰撞包围盒大小，$\boldsymbol{D}_{\text{distance}}$ 为固体模型最大顶点与最小顶点之间的距离向量，即

$$\boldsymbol{D}_{\text{distance}} = \boldsymbol{V}_{\text{model\_max}} - \boldsymbol{V}_{\text{model\_min}}. \tag{5}$$

粗略碰撞检测仅需判断流体粒子位置 $\boldsymbol{P}(\boldsymbol{x}_i)$是否在 AABB 包围盒内即可。

如果 $\boldsymbol{P}(\boldsymbol{x}_i)$在碰撞包围盒内，则进入精细碰撞阶段。精细碰撞检测方法如下，首先计算 $\boldsymbol{P}(\boldsymbol{x}_i)$所在网格单元的三维索引，再预计算下一帧该流体粒子的位置 $\boldsymbol{P}_{\text{next}}(\boldsymbol{x}_i)$，计算公式为，

$$\boldsymbol{P}_{\text{next}}(\boldsymbol{x}_i) = \boldsymbol{P}(\boldsymbol{x}_i) + \boldsymbol{v}_i t + \frac{1}{2}\boldsymbol{a}_i t^2. \tag{6}$$

其中 $\boldsymbol{v}_i$ 为当前帧流体粒子的速度；$\boldsymbol{a}_i$ 为粒子加速度，t 为时间步长；同样计算 $\boldsymbol{P}_{\text{next}}(\boldsymbol{x}_i)$所在网格单元的三维索引。那么，以 $\boldsymbol{P}(\boldsymbol{x}_i)$和 $\boldsymbol{P}_{\text{next}}(\boldsymbol{x}_i)$为最小顶点或最大顶点可构造一个局部包围盒，图 4 用一个二维示意图说明构造方法。

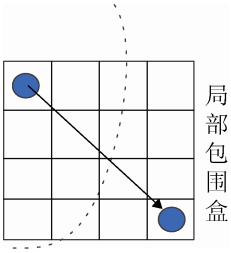


图 4 局部包围盒构造过程

蓝色圆点表示流体粒子位置 $\boldsymbol{P}(\boldsymbol{x}_i)$和 $\boldsymbol{P}_{\text{next}}(\boldsymbol{x}_i)$，箭头从前一帧指向后一帧，它们构造的包围盒如黑色方框所示，黑色网格即包含模型粒子的网格,将网格线性化之后，通过访问 countbuffer 判断两帧粒子之间是否有模型顶点，如果有顶点，便依次访问这些顶点，找出包含该顶点的三角形面片，进入后续碰撞检测处理。由模型顶点粒子在缓存中的存储方式可知，三个相邻顶点可构成一个三角面片，如果知道当前顶点在缓存的下标为 $I(\boldsymbol{x}_i)$，则寻找当前顶点所属的三角面片和其它两个顶点，从而判断是否发生碰撞的算法为，

输入. $I(\boldsymbol{x}_i)$.

输出. 包含该顶点的三角面片的三个顶点.

Step 1. 求 $I(\boldsymbol{x}_i)/3$ 的余数.

Step 2. 若余数为 0，取出与其相邻的后两个位置的顶点；若余数为 1，取出与其相邻的前后两个位置的顶点；若余数为 2，取出与其相邻的前两个位置的顶点。

Step 3.根据求得的顶点构成三角面片，求碰撞点。

借鉴 Möller 的算法[18]，若发生碰撞，三角面片上碰撞点 $\boldsymbol{P}$ 可由公式(8)表示，

$$\boldsymbol{P} = \boldsymbol{P}(\boldsymbol{x}_i) + t\boldsymbol{d}. \tag{7}$$

$\boldsymbol{d}$ 为从 $\boldsymbol{P}(\boldsymbol{x}_i)$到 $P_{\text{next}}(\boldsymbol{x}_i)$的方向向量，$0 \leqslant t \leqslant 1$。同时碰撞点 $\boldsymbol{P}$ 可由三角形的三个顶点线性表示，

$$\boldsymbol{P} = a\boldsymbol{A} + b\boldsymbol{B} + c\boldsymbol{C}. \tag{8}$$

A,B,C 分别为*三角形*的三个顶点且 $0 \leqslant a \leqslant 1$，$0 \leqslant b \leqslant 1$，$0 \leqslant c \leqslant 1$。联合公式(7)，(8)，若判断出 $0 < t < 1$，$0 \leqslant a \leqslant 1, 0 \leqslant b \leqslant 1, 0 \leqslant c \leqslant 1$，则表示该粒子会与三角面片发生碰撞, 否则不发生碰撞。避碰处理时，根据公式(9)可得出碰撞后的粒子位置 $\boldsymbol{P}'_{\text{next}}(\boldsymbol{x}_i)$ 为，

$$\boldsymbol{P}'_{\text{next}}(\boldsymbol{x}_i) = \boldsymbol{P} + (\boldsymbol{P}_{\text{next}}(\boldsymbol{x}_i) - \boldsymbol{P} - 2 \cdot \boldsymbol{n} \cdot \boldsymbol{n} \cdot (\boldsymbol{P}_{\text{next}}(\boldsymbol{x}_i) - \boldsymbol{P})). \tag{9}$$

式中：$\boldsymbol{n}$ 为相应三角面片的法线向量。根据公式(10)可计算碰撞后的速度 $\boldsymbol{v}'_{\text{next}}(\boldsymbol{x}_i)$，

$$\boldsymbol{v}'_{\text{next}}(\boldsymbol{x}_i) = \sigma(\boldsymbol{v}_{\text{next}} - 2 \cdot \boldsymbol{n} \cdot \boldsymbol{n} \cdot \boldsymbol{v}_{\text{next}}). \tag{10}$$

式中：$\boldsymbol{v}_{\text{next}}$ 为未发生碰撞时粒子的速度；σ 为碰撞的速度衰减系数。

5 流体表面重建

本文采用 marching cube 方法在 GPU 中完成流体表面的重建，为了在相同 marching cube 网格精度下达到更细腻的水花效果，本文采用文献[14]的基于距离场的方法构造三维标量场，并利用 DirectX11 的可伸缩缓冲区存储由 marching cubes 算法产生的流体表面三角面片，通过预估可伸缩缓冲区的大小，减少 GPU 缓存开销。

基于距离场构建三维标量场的方法如下，构造隐式函数 $\varphi(\boldsymbol{x})$，

$$\phi(\boldsymbol{x}) = \left|\boldsymbol{x} - \bar{\boldsymbol{x}}\right| - \bar{r}. \tag{11}$$

$$\bar{\boldsymbol{x}} = \sum_i w_i \boldsymbol{x}_j. \tag{12}$$

$$\bar{r} = \sum_i w_i r_j. \tag{13}$$

$$w_i = \frac{k(\left|\boldsymbol{x} - \boldsymbol{x}_i\right| / \mathrm{R})}{\sum_j k(\left|\boldsymbol{x} - \boldsymbol{x}_j\right| / \mathrm{R})}. \tag{14}$$

$$k(s) = \max(0, (1 - s^2)^3). \tag{15}$$

式中：$\boldsymbol{x}$ 为一个流体粒子采样点；$\bar{\boldsymbol{x}}$ 为以 R 为半径范围内采样点位置的加权平均；w_i 为第 i 个采样点的权重；$\bar{r}$ 为以 R 为半径范围内采样点处粒子半径的加权平均；r_j 为第 j 个流体粒子的半径；$k(s)$为以 s 为自变量的适应性核函数；由于本系统中所有流体粒子半径均为 r，因此取 $\bar{r} = r$.将公式(15)带入(14)中，可求得权重 w_i。

本系统的 marching cube 网格分辨率为 $h/3$，每个网格点处存储着隐式函数值 $\varphi(\boldsymbol{x})$和流体法向量。由于在 GPU 中为每个网格点分配了一个线程，使得 $\varphi(\boldsymbol{x})$及流体表面法线的计算得以快速进行，本文

采用中心差分法计算每个网格点处的法线，计算方法如下，

$$\boldsymbol{n}_i = \frac{\phi_{i+1} - \phi_{i-1}}{2\Delta l}. \tag{16}$$

式中：$\boldsymbol{n}_i$ 为第 i 个网格点的法线；ϕ_{i+1} 为第 i+1 个网格点的隐式函数值；Δl 为相邻两个网格点之间的距离。由于空间网格中最外层网格点不能进行中心差分操作，我们简单地将其赋值为零向量。

为了节省存储空间，本系统采用了 DirectX11 中的可伸缩缓冲区，该缓冲区的工作原理与堆栈类似。经过合理设计，将缓冲区的伸与缩在同一存储空间中得到了支持，每个线程中产生的新的三角形按照其时间先后放入缓冲区的末尾，在下一帧时，利用缩减操作，将缓冲区数据清空，以便再一次从缓冲区的第一位开始存放三角形。与以往的方法中采用多个缓存空间用于交换存储三角片面，或者预先分配高于三角面片数量上限的缓存空间相比，由于采用单缓存，且对三角面片数量经过了预估，有效地节约了空间资源。

可伸缩缓冲区存取三角形的过程如图 5 所示，在同一帧当中，三角形先存入缓冲区，将其传入像素着色器绘制完之后，清空缓冲区，进入下一帧的循环再次将新产生的三角面片存入缓冲区。图中标号 1，2 为同一缓冲区的循环过程。

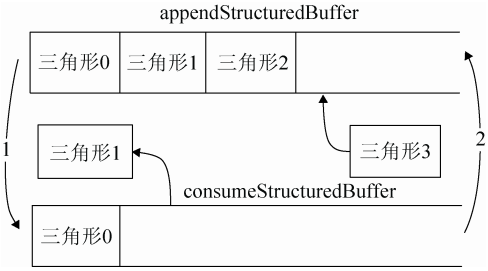


图 5 可伸缩缓冲区存取数据示意图

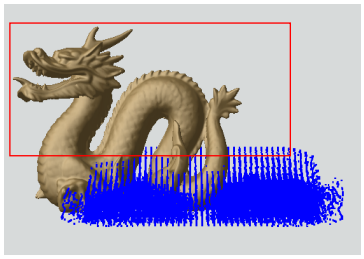
6 流体渲染

流体表面渲染效果由菲涅尔定律和环境贴图共同产生。根据斯涅耳定律和菲涅尔定律求得光线反射系数和折射系数，再结合环境贴图中采样颜色值和透明度渲染，实现了简单的水体渲染效果。

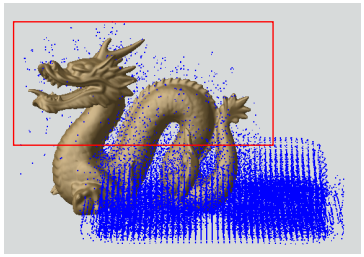
7 实验与结果分析

本文的实验平台为 64 位操作系统的 PC 机，配置为 Inter(R) Xeon(R) E5-2620(2.1GHz)CPU, 16GB 内存，NVIDIA Quadro K2000 显卡。本文从三个方面进行了对比。

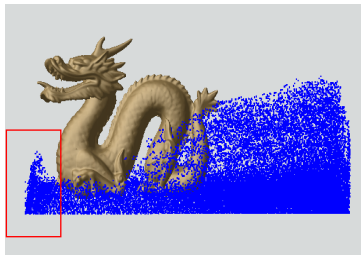
(1)场景与流体粒子产生交互：图 6 为本文基于网格的局部碰撞方法的实验结果截图。



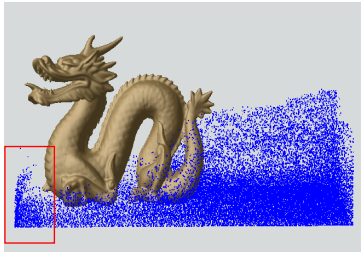
(a) 流体在上方未发生碰撞



(b) 流体在上方发生碰撞



(c) 流体从侧面未发生碰撞



(.)#流体从侧面发生碰撞

图 6 流体与场景的碰撞检测模拟

从图中可见，本文的碰撞检测方法能有效检测到与固体模型发生碰撞的粒子，为了与文献[8]进行对比，实验场景为一条斯坦福龙，流体模拟空间是长宽高分别为 0.7 m，0.4 m，1.0 m 的长方体模拟区域。图中用红色方框标注的区域是可以观察到碰撞发生的区域，其中图 6(a)和(b)为一组流体从斯坦福龙上方掉落时，可观察到龙的头顶有碰撞发生后粒子散落现象。图 6(c)和(d)为流体从斯坦福龙侧面流经龙身，红色区域在经过相同的时间，粒子数明显少于未发生碰撞时的粒子数。

(2)对比分析：为了证明本文的碰撞检测算法效率，在未进行流体渲染情况下，与文献[8]中基于体素化的碰撞检测方法和基于 GPU 的全局碰撞检测进行了速度对比，对比效果如图 7 所示。本文分别选取了流体粒子数为 10 k ~ 64 k 等 6 种规模，通过对比，本文的碰撞检测效率高于另外两种方法，比如在 32 k 流体粒子规模下，本文方法一帧需要消耗 23.69 ms，文献[8]需要 49.95 ms，GPU 全局碰撞方法需要 1 000 ms。由于本系统帧速最高只统计到 1 000 ms/fps，低于此速度均视为 1 000 ms/fps。

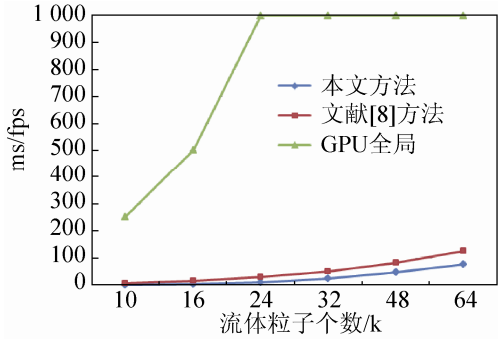
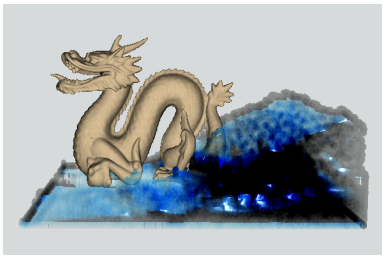


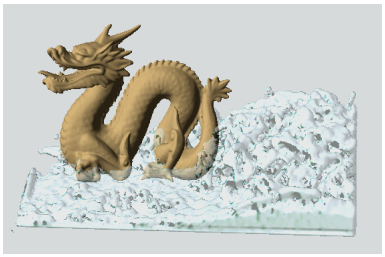
图 7 碰撞检测帧速对比图

文献[8]中由于需要将粒子位置映射到纹理空间，根据前后两帧粒子位置连线每次移动一个位置并采样，直到发生碰撞或者移动到后一帧位置点，降低了碰撞检测速率。本文碰撞检测方法获得加速的原因在于，第一，预处理阶段为模型顶点构建了邻居粒子链表，countbuffer 中已经记录了每个空间网格中顶点个数，访问 countbuffer 可知局部包围盒内是否含有边界顶点；第二，局部包围盒是根据流体粒子前后两帧之间的位置构建的，包围盒相对较小，因此对固体边界的搜索范围也较小；第三，包围盒的构建及碰撞检测过程均在 GPU 中并行实现。

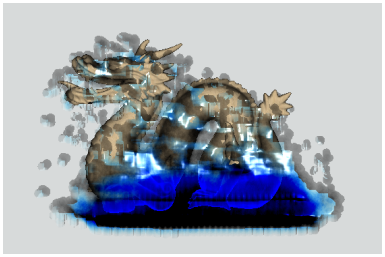
(3)渲染效果对比：由于碰撞检测效率的提升，本系统采用了更能体现流体细节的 marching cube 算法，在流体细节的渲染效果上与文献[8]的对比结果如图 8 所示。



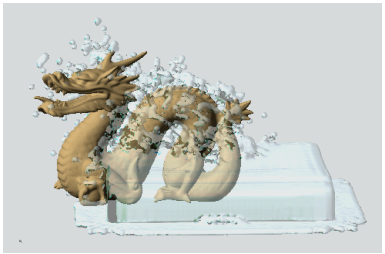
(a) 文献[8]流体从侧面流出



(b) 本文流体从侧面流出



(c) 文献[8]流体从顶部下落



(d) 本文流体从顶部下落

图 8 流体渲染效果对比

图 8(a)和(b)为流体从侧面流经斯坦福龙，图 8(c)和 8(d)为流体从顶部流向龙身，从图中可以看出在都能满足实时性的情况下，本文仿真可展示更细腻的流体细节。

8 结论

本文实现了一个基于 DirectCompute 加速的带静态场景的流体仿真框架，利用 DirectCompute 的无序访问特性优化了流体仿真存储结构，将用于数据交换的两块 GPU 缓存减少至一块 GPU 缓存，节省了用于交换流体粒子位置和速度的缓存空间。本文提出了一种基于网格的局部碰撞检测算法，该算法在不需要复杂的空间数据结构和递归层次下，能动态地构造尽可能小的包围盒，减少了对模型顶点的搜索次数，有效提高碰撞检测的速度。

本流体仿真框架中，针对的只是静态的表面模型，对于动态的模型及弹性模型未作考虑，未来工作将进一步研究动态场景中的流固耦合及弹性材质物体与流体的耦合现象。

参考文献:

[1] 柳有权, 王章野, 朱鉴, 等. 基于物理的流体动画加速技术的研究进展 [J]. 计算机辅助设计与图形学学报, 2013, 25(3): 312-321.

[2] 谭捷, 杨旭波. 基于物理的流体动画综述 [J]. 中国科学(F 辑:信息科学), 2009, 39(5): 499-514.

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

[4] Green S. Particle simulation using cuda [CP/OL]. NVIDIA Whitepaper Packaged with CUDA ToolKit, Nvidia Coporation, 2010. http://developer.nvidia.com/cuda-downloads.

[5] Øystein E Krog, Elster A C. Fast GPU-based fluid simulations using SPH [C]// Applied Parallel and Scientific Computing-International Conference, Iceland: Berlin Heidelberg, 2010: 98-109.

[6] Junior J R S, Joselli M, Zamith M, et al. An architecture for real time fluid simulation using multiple GPUs [C]// SBC-Proceedings of SBGames. Brazil: SBGames, 2012: 93-100.

[7] 周煜坤, 陈清华, 余潇. 基于 CUDA 的大规模流体实时模拟 [J]. 计算机应用与软件, 2015, 32(1): 143-147.

[8] 刘良平, 刘箴, 方昊, 等. 基于 GPU 的三维场景表面流体碰撞检测方法研究 [J]. 系统仿真学报, 2015, 27(10): 2439-2445,2452.

[9] Monaghan J J. Simulating Free Surface Flows with SPH [J]. Journal of Computational Physics (S0021-9991), 1994, 110(2): 399-406.

[10] Liu G R, Liu M B, Li S. Smoothed Particle Hydrodynamics-A Mesh-Free Particle Method [J]. Computational Mechanics (S0178-7675), 2004, 33(6): 491-491.

[11] 刘旭, 班晓娟, 杨鸣远, 等. 一种 SPH 流体仿真边界校正方法 [J]. 图学学报, 2015, 36(3): 462-467.

[12] 张桂娟, 朱登明, 邱显杰, 等. 一种面向流体仿真的场景处理方法 [J]. 计算机辅助设计与图形学学报, 2010, 22(8): 1360-1365.

[13] Lorensen W E, Cline H E. Marching cubes: a high resolution 3D surface construction algorithm [J]. SIGGRAPH Comput Graph (S0097-8930), 1987, 21(4): 163-169.

[14] Zhu Y, Bridson R. Animating sand as fluid [J]. Acm Transactions on Graphics (S0730-0301), 2005, 24(3): 965-972.

[15] Solenthaler B, Schläfli J, Pajarola R. A unified particle model for fluid-solid interactions. [J]. Computer Animation & Virtual Worlds (S1546-4261), 2007, 18(1): 69-82.

[16] Yu J, Turk G. Reconstructing surfaces of particle-based fluids using anisotropic kernels [J]. Proc. of the Acm Siggraph/Eurographics Symp. on Comp. Anim. Eurographics Association (S0730-0301), 2010, 32(1): 217-225.

[17] 刘世光, 王泽, 彭群生. 基于物理的浑浊水体的光照模拟 [J]. 计算机辅助设计与图形学学报, 2011, 23(1): 40-45.

[18] Möller T, Trumbore B. Fast, minimum storage ray/triangle intersection [C]// ACM SIGGRAPH 2005 Courses. USA: ACM, 2005: 7.