Journal of System Simulation

Volume 27 | Issue 10 Article 32

8-7-2020

GPU-Based Fluid Collision Detection Method with Surface of Three-Dimensional Scenes

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

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

Fang Hao
Faculty of Information Science and Technology, Ningbo University, Ningbo 315211, China;

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

See next page for additional authors

GPU-Based Fluid Collision Detection Method with Surface of Three-Dimensional Scenes

Abstract

Abstract: Virtual simulation of fluid phenomena is of great value, and SPH fluid simulation-based method can truly reflect the motion of the fluid. However, as the size of the fluid increasing, the computation becomes large, especially fluid collision detection in complex scenes significantly reduces the efficiency of fluid simulation. *By using GPU to accelerate SPH physical calculation and collision detection, and use of new neighbors particles chain table building method for particles searching, can improve fluid simulation efficiency. By using voxel scene rasterization of sampling methods, the calculation of collision detection was speeded up, and the storage space was saved by compressing each voxel value into 1bit and saving it to file*. Preliminary experiment results show that the method is efficient for collision detection processing, and can reduce processing time of collision detection.

Keywords

SPH, GPU acceleration, collision detection, fluid simulation, voxelization

Authors

Liangping Liu, Liu Zhen, Fang Hao, Cuijuan Liu, and Tingting Liu

Recommended Citation

Liu Liangping, Liu Zhen, Fang Hao, Liu Cuijuan, Liu Tingting. GPU-Based Fluid Collision Detection Method with Surface of Three-Dimensional Scenes[J]. Journal of System Simulation, 2015, 27(10): 2439-2445.

第 27 卷第 10 期 系统仿真学报© Vol. 27 No. 10
2015 年 10 月 Journal of System Simulation Oct., 2015

基于 GPU 的三维场景表面流体碰撞检测方法研究

刘良平，刘箴，方昊，刘翠娟，刘婷婷
（宁波大学信息科学与工程学院，宁波 315211）

摘要：流体现象的虚拟仿真有重要的应用价值，基于 SPH(Smoothed- Particle Hydrodynamics)的流体仿真方法能够真实地反映流体的运动规律。在流体规模增大时流体仿真计算量很大，特别是复杂场景中的流体碰撞检测计算显著降低流体仿真效率。*将 SPH 物理计算以及碰撞检测利用 GPU 进行并行加速，并在 GPU 显存中使用新的邻居粒子链表构建方法加快邻居粒子的查找，可以提高流体仿真效率；利用对体素化场景进行栅格化采样方法来加速碰撞检测计算，并将每个体素值压缩成 1bit 并保存在文件中以节省存储空间，*初步的实验结果表明，该方法能够用于碰撞检测处理，且能够减少碰撞检测处理时间。

关键词：SPH；GPU 加速；碰撞检测；流体仿真；体素化
中图分类号：TP391.9 文献标识码：A 文章编号：1004-731X (2015) 10-2439-08

GPU-Based Fluid Collision Detection Method with Surface of Three-Dimensional Scenes

Liu Liangping, Liu Zhen, Fang Hao, Liu Cuijuan, Liu Tingting
(Faculty of Information Science and Technology, Ningbo University, Ningbo 315211, China)

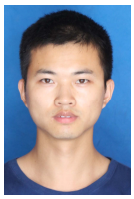
Abstract: Virtual simulation of fluid phenomena is of great value, and SPH fluid simulation-based method can truly reflect the motion of the fluid. However, as the size of the fluid increasing, the computation becomes large, especially fluid collision detection in complex scenes significantly reduces the efficiency of fluid simulation. *By using GPU to accelerate SPH physical calculation and collision detection, and use of new neighbors particles chain table building method for particles searching, can improve fluid simulation efficiency. By using voxel scene rasterization of sampling methods, the calculation of collision detection was speeded up, and the storage space was saved by compressing each voxel value into 1bit and saving it to file.* Preliminary experiment results show that the method is efficient for collision detection processing, and can reduce processing time of collision detection.

Keywords: SPH; GPU acceleration; collision detection; fluid simulation; voxelization

引言

随着虚拟现实技术应用需求的不断发展，人们对仿真视觉效果的要求越来越高。其中流体仿真是虚拟现实领域研究的难点与热点之一，特别是复杂场景中的流固碰撞问题是流体动画中的一个难点。

流体的模拟有多种可行的方法，为了真实的模拟流体地运动及流固碰撞效果，基于物理的流体仿真是一种较好的方法，该方法中，流体能够遵循流体力学 N-S 方程(Navier-Stokes Equations)进行运动，从而使模拟的流体符合真实流体的运动规律。本文采用的光滑粒子流体动力学方法(Smoothed-Particle Hydrodynamics，SPH)正是一种基于物理的方法，该方法不需要划分细致的网格，并且能够模



收稿日期：2015-06-14 修回日期：2015-07-26；
基金项目：国家自然科学基金(61373068)；
作者简介：刘良平(1991-)，男，江西赣州，硕士生，研究方向为虚拟现实；刘箴(1965-)，男，辽宁铁岭，博士，研究员，研究方向为虚拟现实；方昊(1991-)，男，安徽舒城，硕士生，研究方向为虚拟现实。

拟更多的流体细节，因此很适合模拟类似流固碰撞这种流体的运动。但是由于基于物理的流体仿真方法在流体规模增大时计算量很大，因此在使用该方法的情况下，流体仿真的实时性与真实感就将成为一对矛盾。而且随着场景复杂性的增加，流体与场景碰撞交互所消耗的计算资源也将大大增加。因此，为了实时而真实的模拟流体在复杂三维场景中的运动，本文提出一种改进的场景体素化碰撞检测算法，并利用 OpenGL Computer Shader 将 SPH 的物理计算以及碰撞检测在 GPU 中加速实现，从而达到较好的流体仿真效果。

1 相关工作

基于物理的流体模拟方法是通过实现 N-S 方程来控制流体的运动的，根据对 N-S 方程的不同解释，基于物理的模拟方法又分为基于网格的欧拉方法和基于粒子的拉格朗日方法[1]。欧拉方法将流体模拟空间划分为网格，将流体的速度、压力、密度等参数离散到各个网格点上，从而根据网格点上属性随时间的变化模拟流体的运动；拉格朗日法利用带有速度、压力、密度等属性的粒子模拟流体，再根据各个粒子的运动整合得到整个流体的运动，而且由于其能够很好的模拟流体细节，越来越成为研究者关注的热点。

SPH 方法是基于粒子的拉格朗日法中最有前景且应用广泛的模拟方法。1995 年，Stam 等[2]最早将 SPH 引入流体仿真领域用于仿真气体和火焰；而 Desbrun 等[3]则将 SPH 引入图形学领域模拟了可变形物体；随后，Müller 等[4]将 SPH 用于模拟流体的交互应用。

在复杂场景中的流体运动需要考虑较为复杂的碰撞检测等问题。传统的碰撞检测方法主要利用包围盒来进行碰撞检测，这虽然能够加快碰撞检测的计算，但其精确度较低，不能满足复杂表面的精确碰撞问题。Schechter 等[5]利用虚粒子法来表示固体与流体的边界，能够较好的处理流固边界问题，但是虚粒子的构建比较复杂。Shao 等[6]通过结合固体采样边界粒子以及动量守恒保持位置-速度修正的固流耦合的方法，能够得到流体与刚体的碰撞耦合效果。Boyles 等[7]首先利用体素化方法进行碰撞检测计算。He 等[8]提出一个两步碰撞检测法，首先获得场景体素化结果，再通过判断粒子包围盒与体素是否相交来判断碰撞检测结果，但其没有给出关联场景与其体素空间变换的方法。为了能在复杂场景中快速而精确的进行碰撞检测，本文提出了一种利用对体素化场景进行栅格化采样的方法来加速碰撞检测的计算。实验证明，该方法能够在静态复杂场景中进行快速而精确的碰撞检测。

随着图形硬件的快速发展，为了能够实时的模拟流体运动，越来越多的研究者开始将 SPH 方法实现于 GPU 中。Harada 等[9]利用 RGBA 四通道像素来存储每个网格的状态，但这限制了每个网格能够存储的粒子数目，从而在粒子数目很大时会影响计算效率。Chen 等[10]提出了一个非均匀采样 SPH 框架，并通过改进的三维空间网格划分算法和基数排序算法将该框架全部实现于 GPU 中，提高了算法的性能。Wen 等[11]将 SPH 实现于 GPU 中，并直接将计算结果缓存传输到着色器进行渲染，避免了 CPU 与 GPU 之间的数据传输。Xiao 等[12]发现将 SPH 完全实现于 GPU 中将导致 CPU 的空闲，因此将网格更新实现于 CPU 中，并同 GPU 中的渲染并行执行，但这将导致 CPU 与 GPU 之间的数据传输，造成性能的损失。本文将 SPH 及碰撞检测算法全部实现于 GPU 中，并利用新的方法构建了粒子邻居链表来提高 GPU 中邻居粒子搜索效率，从而能够提高流体仿真速度。

2 SPH 物理框架

SPH 是基于粒子的拉格朗日方法，其通过将流体离散成一颗颗的粒子，并在每颗粒子上存储密度、压强、速度等属性来模拟流体的运动。粒子之间的影响是通过光滑核来施展的。粒子属性累加公式为：

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

其中: m_j 和 ρ_j 分别是周围粒子的质量和密度; A 是要累加的某种属性; $\boldsymbol{x}_j$ 是粒子 j 的位置， h 是光滑核半径。函数 W 就是光滑核函数。

要求解粒子在下一时间步后的位置，我们需要知道粒子的加速度。根据 N-S 方程在 SPH 上的应用，可得到方程：

$$\boldsymbol{a}=\boldsymbol{g}-\frac{\boldsymbol{F}_i^{pressure}}{\rho}+\frac{\boldsymbol{F}_i^{viscosity}}{\rho} \quad (2)$$

其中右边三项分别代表加速度的外力项、压力项和粘度项，粒子的加速度主要由这三个部分组成。

下面给出 SPH 算法具体步骤：

根据公式(1)，以密度 ρ 代替属性 A ，可得到粒子密度的计算公式：

$$\rho(\boldsymbol{x})=\sum_j m_j W(|\boldsymbol{x}-\boldsymbol{x}_j|,h) \quad (3)$$

每颗粒子的压强可使用理想气体的压强方程来计算：

$$p=k(\rho-\rho_0) \quad (4)$$

其中 k 表示不可压缩性常量，而 ρ_0 是流体静态密度。压力项计算公式如公式(5)所示，其中 ∇ 表示求梯度。

$$\boldsymbol{F}_i^{\text{pressure}}=-\sum_{j\neq i}m_j\left(\frac{p_i+p_j}{2\rho_j}\right)\nabla W(|\boldsymbol{x}_i-\boldsymbol{x}_j|,h) \quad (5)$$

粘性力是由粒子之间的速度差引起的，速度不同的粒子之间会有剪切力的作用，而且粘性力项可以用来维持流体系统的稳定性。粘度项的计算公式如下：

$$\boldsymbol{F}_i^{\text{viscosity}}=\mu\sum_j m_j\frac{\boldsymbol{u}_j-\boldsymbol{u}_i}{\rho_j}\nabla^2 W(|\boldsymbol{x}_i-\boldsymbol{x}_j|,h) \quad (6)$$

其中 $\boldsymbol{u}_j$ 和 $\boldsymbol{u}_i$ 分别表示粒子 j 和 i 在上一帧时的速度， ∇^2 为拉普拉辛算子。

3 SPH 算法的 GPU 实现

SPH 算法的实现需要大量的物理计算，而其将流体模拟离散为粒子的思想又特别适合进行并行化计算。因此利用 GPU 对 SPH 方法加速是一种很自然的想法。

本文 SPH 物理计算和碰撞检测全部采用 GPU 进行计算，并且利用 OpenGL 计算着色器与其它着色器共享 GPU 缓存的优点，直接将计算结果传到着色器中渲染而省去了与 CPU 进行数据交互而带来的性能损失。

本文的 GPU 实现流程如图 1 所示。首先在 CPU 中初始化粒子数量及粒子位置，并创建所需要的 GPU 缓存对象，将 CPU 中的数据传入 GPU 缓存对象中进行计算。本文中使用的 GPU 缓存对象共有 7 个，其中速度缓存和粒子位置缓存分别有两个，这是为了方便保存上一帧粒子位置和速度数据以及当前帧粒子位置和速度数据，用于之后进行碰撞检测计算。一个网格缓存对象用于存储每个网格中包含的某个粒子的索引，该索引可用于在粒子位置缓存中检索粒子。一个加速度缓存对象存储 SPH 计算中得到的粒子加速度，用于之后的碰撞检测计算中。最后一个缓存是三维边界纹理，存储场景体素化结果，用于碰撞检测时采样计算是否发生碰撞。

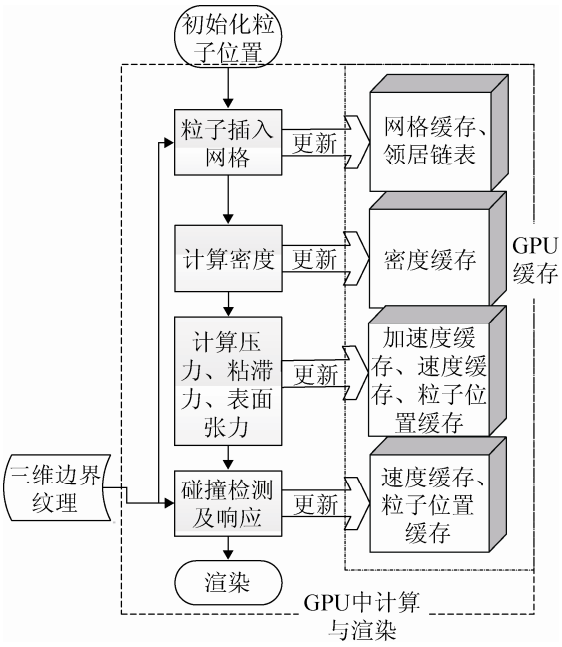


图 1 本文流体仿真 GPU 实现流程

在 GPU 计算中，为了减少对显存的占用以及

减少缓存的数量，本文利用速度缓存中的 w 分量来保存粒子的密度值。

在 SPH 实现中，搜索粒子所在网格及其附近的网格即可生成每颗粒子的邻居粒子表。这在 CPU 中很容易实现，只要在粒子结构体中维持一个数组存储邻居粒子的索引即可。然而，由于 GPU 中数据结构的灵活性不如 CPU，因此对于 GPU 中的每个粒子都加上一个邻居粒子数组是不现实的。为此，以往的研究者们采用 hash 结构将粒子与网格关联起来，并且在每一帧对 hash 结果排序，则在同一个网格中的粒子排序后将是相连的，从而可以获得网格中的粒子列表。

本文采用另一种新的方法存储网格中粒子。将每个网格中存储的数据作为粒子链表的头指针，一旦有粒子插入到网格中，则将新粒子的索引存储到该网格中，而将网格中的数据存储到粒子位置数据的 w 分量中，这样在粒子全部插入后，每个网格相当于维持了一个粒子的链表。当然，需要说明的是，在插入粒子的时候，数据是需要互斥操作的，因此本文采用原子操作来保证粒子插入的正确执行。粒子邻居链表的构建如图 2 所示。

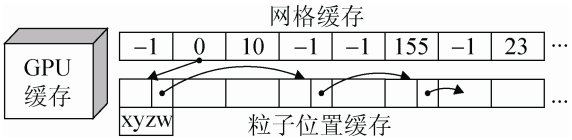


图 2 粒子邻居链表构建图

4 碰撞检测

复杂场景中含有各种复杂的三维表面与边界，因此在进行直接精确的碰撞检测计算的时候，每颗粒子都需要遍历对象三角片元以判断粒子是否和曲面产生碰撞，而这将消耗大量的计算资源，从而使仿真很难同时达到实时与精确的效果。为此，本文改进一种利用三维场景体素化来简化碰撞检测的方法。该方法首先利用复杂曲面模型的深度图得到一个包含模型边界信息的三维纹理，即体素化；然后将三维纹理载入 GPU 中，在需要进行碰撞检测的时候对该三维纹理进行栅格化采样，从而在较短的时间内获得碰撞检测的结果。

4.1 计算三维边界纹理

对于一个三维空间中的对象，可将其抽象为一个外接六面体包围盒，将六面体的各个方向按照相同的距离 d 分别等分为 n_x,n_y,n_z 等份，则该对象的边界可由一个 $n_x \times n_y \times n_z$ 的三维纹理表示出来。纹理中每个 voxel 存储的值为：0(若该 voxel 非对象边界)，1(若该 voxel 为对象边界)。

得到一个六面体包围盒后，对六面体的每一面，采用 OpenGL 编程接口的正视投影渲染对象从而得到该面的深度图。则对于从 xy 面，朝向 z 轴正向方向渲染得到的深度图来说，当 $\text{depth}_{i,j}=0$ (i,j 分别表示三维纹理在 x,y 轴上的索引，$\text{depth}_{i,j}=0$ 表示深度值)时，相应的三维纹理上的那列都不是对象边界，因此该列的纹理值都设为 0.而对于 $0<\text{depth}_{i,j}<1$，可根据公式

$$\text{voxel}_{i,j,k}=\begin{cases}1, 0<\text{depth}_{i,j}<1,\ k=\text{depth}_{i,j}\times n_z\\ 0, \text{depth}_{i,j}=0,\ k=0,1,2,...,n_z-1\end{cases} \quad (7)$$

填充三维纹理图中的相应元素值。对于其它各面的深度图，也可通过相应公式转换为三维纹理图的相应元素值，这里不再赘述。图 3 显示了如何在二维空间上获得边界纹理，在三维空间中的体素化原理也是一样的，只是更加复杂。

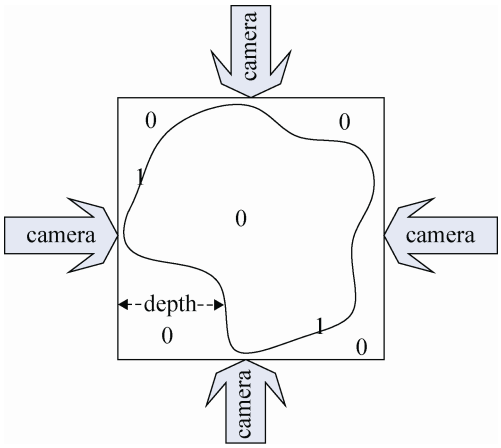


图 3 二维空间中的边界纹理图生成

对于更为复杂的甚至含有内部空洞的对象，还需要通过减小六面体包围盒继续扫描对象的边界，直至三维边界纹理的每一个纹理元素都被填充为 0 或 1，从而得到完整的对象三维边界纹理。对于有三维复杂表面的场景，其三维边界纹理计算达不到实时的效果，因此，对于静态场景，本文通过将首次计算得到的三维边界纹理存储于文件中，方便以后用到该场景时直接使用。

4.2 三维边界纹理的装载

得到三维边界纹理之后，需要将其装入 GPU 缓存中。三维纹理的存储是特别消耗空间的，如果每个纹理元素都以 float 类型存储，那么将大大限制得到的纹理的分辨率的大小，而由于纹理中存储的数据只是 0 或者 1，因此，可将纹理元素存储到 char 类型甚至压缩到每位存储一个元素，这样可以降低三维纹理所占用的存储空间。本文采用 unsigned char 类型来存储三维边界纹理数据，并且为了按位存储纹理数据，将 z 轴方向的索引进行压缩，即用 $ceil(n_z/8.0)$ (ceil 表示向上取整)来表示 z 轴方向的纹理个数，这样用一个字节就可以存储 8 个 voxel 值，从而所需要的纹理存储空间变为 $n_x \times n_y \times \text{ceil}(n_z/8.0)$ 个字节。

4.3 碰撞检测及处理

4.3.1 粒子空间变换

为了利用三维空间边界纹理对粒子进行碰撞检测，首先需要将粒子变换到该三维纹理空间。三维纹理空间是以原点为起始点，沿着 xyz 轴正向的空间。根据以下公式，可将粒子变换到三维纹理空间：

$$M = \text{scale}(\frac{x_{\text{resolution}}}{x_{\text{size}}}, \frac{y_{\text{resolution}}}{y_{\text{size}}}, \frac{z_{\text{resolution}}}{z_{\text{size}}}) \quad (8)$$

其中 $x_{\text{resolution}}$， $y_{\text{resolution}}$， $z_{\text{resolution}}$ 分别表示三维纹理在 xyz 方向上的分辨率， x_{size}， y_{size}， z_{size} 表示三维纹理对应的对象的大小。此公式能够按照对象到该对象三维纹理的缩放比例对粒子进行缩放，从而将粒子变换到三维边界纹理空间。

以上只是三维纹理空间中的变换，为了使视觉空间的效果与三维纹理空间的碰撞检测保持一致，需要在视觉空间中对对象进行变换了之后将相应的变换应用到粒子在三维纹理空间中的变换中去。不过由于在视觉空间中是对对象的变换，而在三维纹理空间中是对粒子的变换，因此这两个变换正好是相反的。图 4 给出了二维条件下进行平移变换时视觉空间中的对象变换以及纹理空间中的粒子变换的对应关系。开始时对象与其体素化纹理都在 A 点，粒子在 a 点，当视觉空间中对象移动到 B 点时，粒子在对象的左边，因此在三维纹理空间中需要将粒子由 a 点反向的平移到 b 点，这样才能与视觉空间中的结果保持一致。

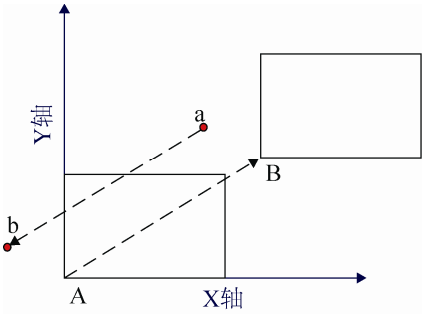
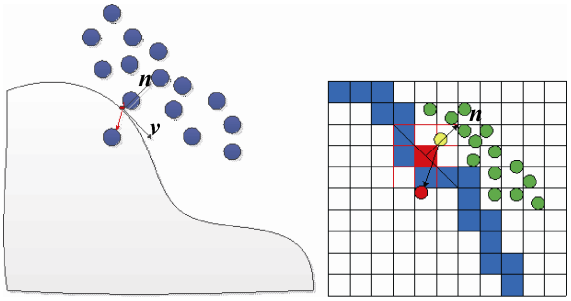


图 4 二维条件下视觉空间中对象的平移与纹理空间中粒子的平移的对应关系

4.3.2 碰撞检测计算

粒子变换到三维边界纹理空间后，就可以利用粒子位置在三维纹理中进行采样来判断是否发生碰撞了。但是由于对象内部的纹理值与外部一样都是 0，当粒子穿透边界到达内部时，直接利用粒子位置值进行纹理采样也会被判定为未发生碰撞。为了解决这个问题，本文同时存储了上一帧粒子位置值和本帧粒子位置值，从而可以利用上一帧粒子位置值与本帧计算得到的粒子位置值的连线是否经过对象边界来判断是否发生碰撞。计算时，从上一帧粒子位置点开始，向着本帧粒子位置点每次移动一个位置并对三维纹理采样，直到发生碰撞或者移动到本帧粒子位置点为止。这与图形管线中的光栅

化类似，因此叫做栅格化采样。二维条件下的碰撞检测情况如图 5 所示。



(a) 粒子与对象发生碰撞 (b) 碰撞点附近纹理边界图

图 5 碰撞检测判断

图 5(a)中粒子由对象边界外经过一个时间步长移动到对象内部，与对象边界发生碰撞，碰撞点的法线为 $\boldsymbol{n}$，由流体运动边界条件 $\boldsymbol{u}\cdot\boldsymbol{n}=0$ 可得碰撞点法线方向速度为 0，碰撞后速度与法线垂直。图 5(b)为粒子碰撞点局部边界纹理图，蓝色小块表示边界，粒子由黄色点运动到红色点经过了两个蓝色小块，粒子将在第一个小块处与对象发生碰撞。

粒子发生碰撞后，需要计算碰撞点的法线 $\boldsymbol{n}$ 用于更新碰撞后的速度与加速度值。对于 $\boldsymbol{n}$ 的计算，本文采用边界纹理梯度值来进行计算。边界纹理梯度值的计算可根据公式(1)先计算出颜色域值，再求梯度，即：

$$\boldsymbol{n}=\nabla \boldsymbol{T}=\sum\nolimits_j \text{voxel}_j \nabla W(|\boldsymbol{x}-\boldsymbol{x}_j|,h) \tag{9}$$

其中：$\nabla \boldsymbol{T}$ 表示纹理梯度值；$\boldsymbol{x}$ 表示纹理位置；voxel_j 表示纹理值。可对于图 5(b)中碰撞点法线的计算，只需计算图中红色方块表示的纹理图的梯度即可。

有了 $\boldsymbol{n}$ 之后，可以计算碰撞后的速度和位置：

$$\boldsymbol{v}_{\text{collision}}=\boldsymbol{v}_{\text{pre}}-(\boldsymbol{v}_{\text{pre}}\cdot\frac{\boldsymbol{n}}{|\boldsymbol{n}|})\frac{\boldsymbol{n}}{|\boldsymbol{n}|} \tag{10}$$

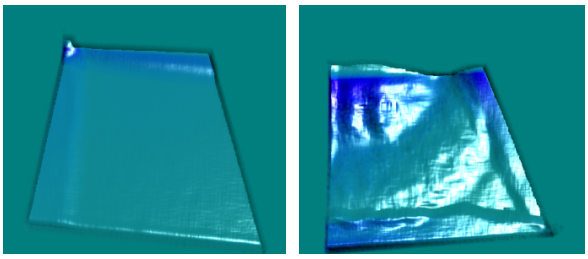
$$\boldsymbol{p}_{\text{collision}}=\boldsymbol{p}_{\text{pre}}+\boldsymbol{v}_{\text{collision}}\times\Delta t \tag{11}$$

其中 $\boldsymbol{v}_{pre}$ 和 $\boldsymbol{p}_{pre}$ 分别是上一帧的速度和位置，$\boldsymbol{v}_{collision}$ 和 $\boldsymbol{p}_{collision}$ 是碰撞后的速度和位置，Δt 表示时间步长。

5 实验及结果分析

本文实验环境为一台搭载 win8.1 64 位操作系统的 PC 机，该机配置为 Inter Core(TM) i7-4790 (3.6GHz) CPU，12GB 内存、NVIDIA GeForce GTX745 显卡。

如图 6 所示，为本文进行 GPU 加速模拟 SPH 结果。在本文 GPU 加速效果实验中，本文模拟了粒子数分别为 1，2，3，4，5，6，7，8 乘以 16384 的流体运动效果，并与 GPU+CPU 协同加速[12]结果进行了比较。从图 7 中可以看出，本文的 GPU 加速效果同等条件下要优于 GPU+CPU 协同加速效果。这是由于本文使用的邻居粒子链表减少了邻居粒子搜索时间以及没有了 GPU 与 CPU 间数据传输损耗的缘故。



(a) 粒子数 4×16 384 (b) 粒子数 8×16 384

图 6 GPU 加速流体仿真

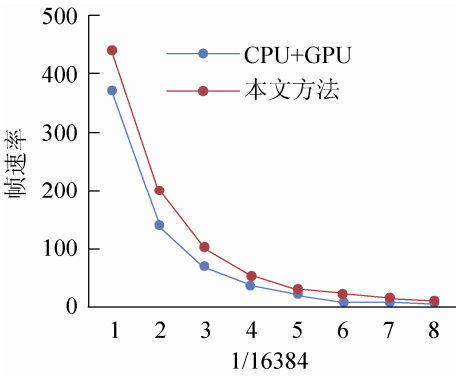
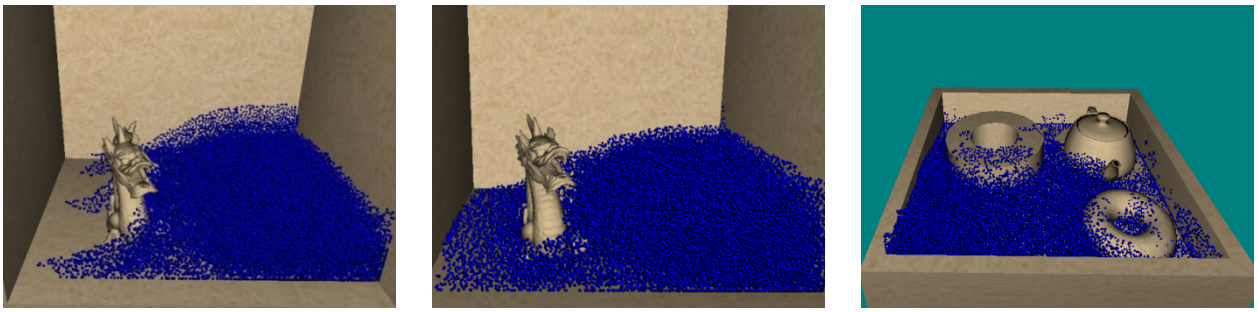


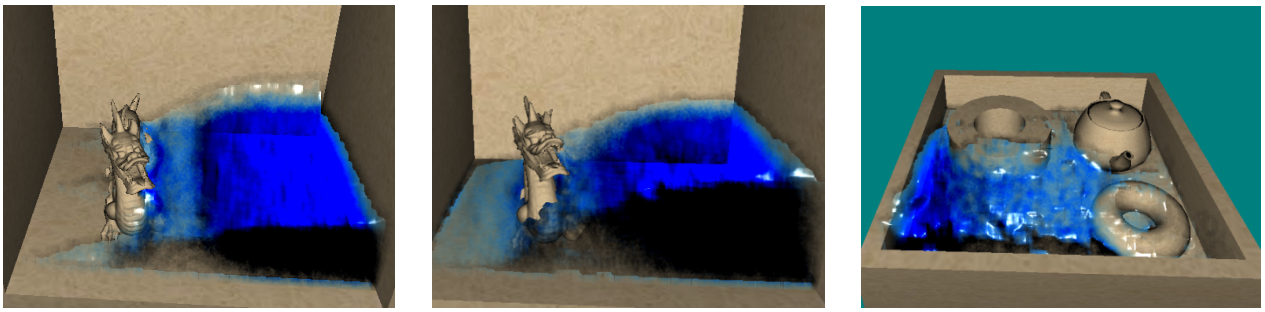
图 7 本文方法与 CPU+GPU 方法仿真效率对比

图 8 模拟了流体在场景中发生碰撞的效果。场景中包含一个长方体盒子和拥有复杂表面的斯坦福龙。在粒子数目为 4×16384 的时候，仿真速率能够达到 25 帧，实现了实时的仿真效果。图 8(b)(e)

中没有施加碰撞检测，因此流体可以直接穿透斯坦福龙。而图 8(a)(d)是加上了本文的碰撞检测的效果图，从图中可以看到流体在遇到斯坦福龙的时候，明显产生了绕开的效果，即检测到了流体与斯坦福龙的碰撞。图 8(c)(f)是流体与几个复杂表面物体的交互效果，可以看出流体与物体碰撞后飞溅起来。该实验效果验证了本文方法能够较好的用于碰撞检测的计算。



(a) 粒子与斯坦福龙发生碰撞(点精灵渲染) (b) 未施加碰撞检测(点精灵渲染) (c) 与复杂场景碰撞(点精灵渲染)



(d) 粒子与斯坦福龙发生碰撞(屏幕空间渲染) (e)未施加碰撞检测(屏幕空间渲染) (f)与复杂场景碰撞(屏幕空间渲染)

图 8 流体与复杂场景的碰撞检测模拟

6 结论

本文提出了一种复杂场景三维表面流体碰撞检测算法，能够通过栅格化采样体素化的场景纹理来快速而精确的进行粒子与三维表面的碰撞检测；并提出一种在 GPU 显存中构建粒子邻居链表的方法，以加快 GPU 中流体的仿真速度。本文将 SPH 物理计算、碰撞检测以及渲染全部在 GPU 中进行，不必与 CPU 交换数据，提升了流体的仿真效率。

由于复杂场景体素化的速度还不能达到实时，因此本文的碰撞检测方法只对静态复杂场景或者较简单的场景能够实现实时计算。我们未来的工作包括提高场景体素化速度、优化场景交界处的法线计算方法等。

参考文献:

[1] 方贵盛，潘志庚. 水虚拟仿真技术研究进展[J]. 系统仿真学报, 2013, 25(09): 1981-1989.

[2] 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.

[3] Desbrun M, Gascuel M P. Smoothed particles: A new paradigm for animating highly deformable bodies [M]. France: Computer Animation and Simulation'96, Springer Vienna, 1996: 61-76.

[4] Müller 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.

（下转第 2452 页）