基于流体模拟的真实海浪和其他流体模拟器

结题报告

王兆楠 吴义凡 刘奕霖 吴沂隆

（电子科技大学，软件学院）

摘　要：围绕模拟流体的目的，制作出了基于 Monaghan SPH（smoothed particle hydrodynamics，光滑粒子流体动力学）方法的流体粒子模拟器通用模板。基于 C++面向对象编程、GLM 数学库和 OpenGL，实现了一个简单的流体粒子系统，可以根据设置的网格参数和自定义添加的材质参数，高效并发实时模拟流体的形态变化，同时实现多种流体的交互模拟。

关键词：流体模拟；流体动力学；Navier-Stokes 方程；SPH；OpenGL；流体交互

一、介绍

1. 流体模拟简介

伴随着计算机的发展，人们的生活对于计算机的这类工具也越发依赖。与此同时，伴随计算机一同增加的是工程计算、军事仿真、游戏和电影等领域对真实物理环境模拟的需求。如何提高计算机模拟仿真物理现象的真实性、准确性和效率成了科学家们关注的问题，也成了计算机图形学这一学科发展突破的方向。

流体模拟是计算机物理仿真的一个重要仿真内容，它的主要目的是实现真实高效的流体模拟。在对真实世界的模拟中，流体模拟是最重要也最常用的领域之一。从水流流动到喷射倾倒，甚至是卷动的龙卷风，计算机模拟的水、火焰、气流等的流体得到了广泛的使用。而流体动力学是一个复杂的领域，流体模拟也以计算量巨大而著称，但是一旦它出了效果就能提供巨大的产品价值以及叹为观止的视觉效果。

常见的流体模拟方法有三种：

1. 基于纹理变化的流体模拟；
2. 基于二维高度场网格的流体模拟；
3. 基于真实物理方程的流体解算；

而基于真实物理方程的流体模拟是其三个基本方法中的一种，也是现在图形学发展的一个方向，其核心也就是流体方程(Navier-Stokes 方程，也称 N-S 方程)：

$$\rho(\vec{r_i})\frac{dV}{dt} = \rho(\vec{r_i})\,\vec{g} - \nabla p(\vec{r_i}) + \mu\nabla^2\,\vec{v}\,(\vec{r_i})$$

若完全用它来进行模拟，这必然是一个不可能完成的计算量，所以伴随延伸的就是一系列用于近似模拟的算法与模型。其中最常用，也是最逼真的一类算法就是本作品实现的 SPH 方法。

2. SPH 方法简介

光滑粒子流体动力学(smoothed particle hydrodynamics, SPH)方法是流体模拟的一种常用方法，也是当前用于基于物理的流体仿真无网格法的主要方法。如何利用 SPH 方法进行更好地自由流体的表面模拟也是当今一大热点课题。

1994 年，Monaghan 首次实现了基于 SPH 的自由表面流体模拟[3]，该结果为后人的基于 SPH 方法的自由表面流体模拟的研究奠定了基础。伴随技术的发展，SPH 延伸出了多种算法模型，例如用于解决由邻域粒子缺失导致的负压力从而产生的非物理聚集现象的 Ghost SPH[4]方法，基于使用了平均曲率的表面张力模型的 Yang[5]方法等。它们的存在为各种各样的领域的流体模拟的实现贡献了不可忽视的力量。

SPH 方法中，对光滑核函数(Smoothing Kernel Function)的使用也是该方法的一个特点。

光滑核函数，也可被称为光滑函数(Smoothing Function)、光滑核(Smoothing Kernel)或核(Kernel)。不同的光滑核函数决定了函数的近似式和核近似与粒子近似的一致性和精度，定义了粒子的支持域Ω的大小。

3. 光滑核函数的主要特性

1. 归一性

$$\int_{\Omega} W(x-x',h)dx' = 1$$

确保在光滑核函数支持域上的积分是归一的。

2. 紧支域

$$W(x-x',h) = 0, |x-x'| > \kappa h$$

这个性质让 SPH 方法只对局部坐标关注，它的遍历范围将不在于全局，仅需某一范围内粒子，就可以得到当前点的目标值得近似值。

3. 点 x 的支持域内任一点非零

$$W(x-x',h) \geq 0$$

在满足紧支域这一性质下，算法的关注值将在支持域（局部）内，作为收敛条件，这个在数学上是非必要的，但为了确保物体的物理现象具有物理意义（稳定性），它非常必要。

4. 随粒子间距离 r 的增大，粒子的光滑核函数值单调递减

同样作为物理意义上的扩展考虑，距离计算粒子越近的粒子对该粒子的影响越大，即随粒子间距离 r 的增大，粒子的光滑核函数值单调递减。这条性质同样可以得到 SPH 具有紧支域这一主要性质。

5. 光滑核半径趋于零时，近似值趋近于函数值

$$\lim_{h\to 0} W(x-x',h) = \delta(x-x')$$

该性质确保了光滑核半径趋于零时，近似值趋近于函数值，即$\langle f(x)\rangle = f(x)$。即当支持域趋向于 0 时，光滑核函数就会趋近于狄克拉 δ 函数。

6. 对称性

粒子距离相同但位置不同，其对目标粒子的影响应该相同。

7. 光滑性

光滑核函数越光滑，其所求函数及其导数也将越光滑，而近似结果也将越好。因为光滑核函数对粒子的不规则分布不具有敏感性。

4. 流体模拟中的经典光滑核函数

1. W_{poly6}核函数：

Poly6 核函数是计算除压力和粘滞力外最常用的核函数，其基本形式为：

$$W_{Poly6}(r,h)=\begin{cases}K_{poly6}(h^2-r^2)^3, & 0\le r\le h\\ 0, & otherwise\end{cases}$$

Where $r=|\vec{r}|=x-x'$

$$K_{poly6}=1/\int_0^{\pi}\int_0^{2\pi}\int_0^{h}(h^2-r^2)r^2sin(\phi)drd\phi d\theta=\frac{315}{64\pi h^9}$$

梯度：$\nabla W_{poly6}(r,h)=-\frac{945}{32\pi h^9}r(h^2-r^2)^2$

二阶微分：$\nabla^2 W_{poly6}(r,h)=-\frac{945}{32\pi h^9}(h^2-r^2)(3h^2-7r^2)$

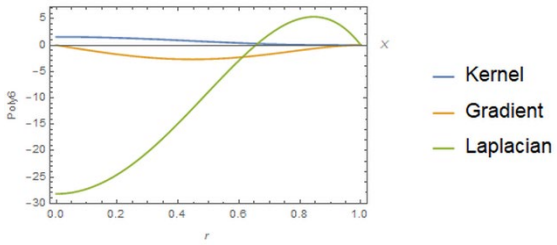


图 1 Poly6 函数

Poly6 核函数仅包含 r 的平方项，因此在计算上可以大大地节约计算量。但它的梯度当两个点无线接近，即 r 趋于 0，函数值趋近于 0：

$$\lim_{r\to 0}\nabla W_{(poly6)}(r,h)=0$$

若将这梯度函数用于计算压力，那么当两个连无限接近时，它们之间的排斥力反而接近消失，最终这会导致粒子的群聚现象。同理，若将它的二阶微分（拉普拉斯算子）用于计算粘滞力，反而起不到作为阻尼的粘滞效果。因此，为了近似模拟这两个，引入了 Spiky 核函数和 Viscosity 核函数。

2. W_{spiky}核函数：

Spiky 核函数用于计算需要核函数梯度的压力的，其基本形式为：

$$W_{Spiky}(r,h)=\begin{cases}K_{Spiky}(h-r)^3, & 0\le r\le h\\ 0, & otherwise\end{cases}$$

Where $r=|\vec{r}|=x-x'$

$$K_{Spiky} = 1/\int_0^{\pi}\int_0^{2\pi}\int_0^{h}(h-r)r^2 sin(\phi)drd\phi d\theta = \frac{15}{\pi h^6}$$

梯度：$\nabla W_{Spiky}(r,h) = -\frac{45}{\pi h^6}(h-r)^2$

二阶微分：$\nabla^2 W_{Spiky}(r,h) = -\frac{90}{\pi h^6}(h^2-r^2)(3h^2-7r^2)$

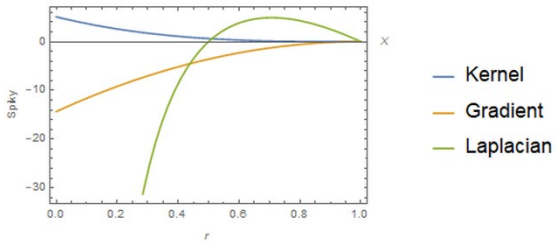


图 2 Spiky 函数

Spiky 核函数具有以下特点：

$$\lim_{r\to 0^+}\nabla W_{Spiky}(r,h) = -\frac{45}{\pi h^4}$$

$$\lim_{r\to 0^+}\nabla^2 W_{Spiky}(r,h) = -\infty$$

相比于 Poly6 核函数来说，它满足了梯度下在粒子距离无限接近时，核函数值非零。对于压力来说，它在公式中为$-\nabla p$，结合$\nabla W_{Spiky}(r,h)$后满足，随距离减少，粒子间的排斥力逐渐增大，而相互相反的特点。

3. $W_{viscosity}$核函数：

Viscosity 核函数用于计算需要核函数拉普拉斯算子下的粘滞力的，其基本形式为：

$$W_{Viscosity}(\vec{r},h) = \begin{cases} K_{Viscosity}\left(-\frac{r^3}{2h^3}+\frac{r^2}{h^2}+\frac{h}{2r}-1\right)^3, 0 \le r \le h \\ 0, \qquad otherwise \end{cases}$$

Where $r = |\vec{r}| = x - x'$

$$K_{Viscosity} = 1/\int_0^{\pi}\int_0^{2\pi}\int_0^{h}(-\frac{r^3}{2h^3}+\frac{r^2}{h^2}+\frac{h}{2r}-1)r^2 sin(\phi)drd\phi d\theta = \frac{15}{2\pi h^3}$$

梯度：$\nabla W_{Viscosity}(r,h) = -\frac{15}{2\pi h^3}r(-\frac{3r}{2h^3}+\frac{2}{h^2}-\frac{h}{2r^3})$

二阶微分：$\nabla^2 W_{Viscosity}(r,h) = \frac{45}{\pi h^6}(h-r)$

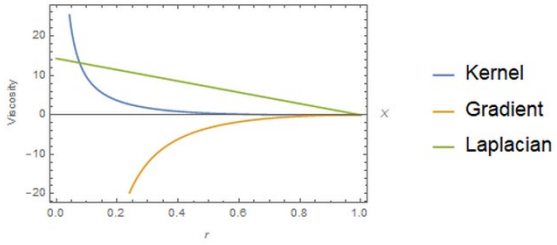


图 3 Viscosity 函数

Spiky 核函数具有以下特点：

$$\lim_{r\to 0^+} \nabla W_{\text{Viscosity}}(r,h) = -\infty$$

流体的粘滞现象主要由流体间的摩擦导致的，其宏观表现为流体的动能的减少，转换为热量流失。N-S 方程中，流体粘滞力被表达为$\mu\nabla^2 V$，因此可知，近似模拟它的结果取决于光滑核函数的拉普拉斯算子函数。而 Viscosity 可以很好的模拟粒子间由距离变化导致的糙度变化，让最后的结果更为合理。

二、 Navier-Stokes 方程推导

1. 流体方程推导

首先，根据物理基础公式，可以列出以下方程：

$$m\vec{a} = \vec{F} \tag{1}$$

$$m = \rho \times V$$

在基础 SPH 方法[1]中，一切体积 V 都以单位体积来进行表示：

$$\Rightarrow m = \rho$$

$$Substituted\ (1):\ \rho\vec{a} = \vec{F} \tag{2}$$

对于一个流体粒子来说，它受到了来自周围流体的压力和粘滞力，以及自身粒子受到的外力（如万有引力）：

$$\vec{F} = \vec{F}_{external} + \vec{F}_{pressure} + \vec{F}_{viscosity} \tag{3}$$

其中：

$\vec{\boldsymbol{F}}_{\boldsymbol{external}}$	外力，这里指重力 $\Rightarrow \vec{\boldsymbol{F}}_{\boldsymbol{external}} = \boldsymbol{\rho}\,\vec{\boldsymbol{g}}$
$\vec{\boldsymbol{F}}_{\boldsymbol{pressure}}$	流体内压力差产生的压强，数值上等于压力场的梯度，但压力的方向是与压力场的梯度方向相反 $\Rightarrow \vec{F}_{pressure} = -\nabla p$
$\vec{\boldsymbol{F}}_{\boldsymbol{viscosity}}$	流体间由于速度差产生的类似于粘滞力的作用力，与粘度（viscosity）系数μ和速度差有关 $\Rightarrow \vec{F}_{viscosity} = \mu\nabla^2\vec{v}$

$$Substituted\ (3):\ \rho\vec{a} = \rho\vec{g} - \nabla p + \mu\nabla^2\vec{v}$$

经过简单变形，我们可以得到 Navier-Stokes 方程：

$$Navier-Stokes\ Equation:\ \rho\frac{\partial\vec{V}}{\partial t} = \rho\vec{g} - \nabla p + \mu\nabla^2\vec{V}$$

简化：

$$Simplified:\ \vec{a}=\vec{g}-\frac{\nabla p}{\rho}+\frac{\mu\nabla^2\vec{v}}{\rho} \tag{4}$$

2. SPH 方法推导

细化之前得到的简化后的方程，以离散粒子的角度对其进行变形：

$$Substituted\ (4):\vec{a}(\vec{r_i})=\vec{g}-\frac{\nabla p(\vec{r_i})}{\rho(\vec{r_i})}+\frac{\mu\nabla^2\vec{v}(\vec{r_i})}{\rho(\vec{r_i})} \tag{5}$$

其中：

$\vec{r_i}$	流体中位置为r_i的点
$\rho(\vec{r_i})$	流体中位置为r_i处的密度
$p(\vec{r_i})$	流体中位置为r_i处的压力
$\vec{v}(\vec{r_i})$	流体中位置为r_i处的速度

假设：$\vec{r_i}$在光滑核半径 h 范围内有多个粒子，分别标号为$\vec{r_1}$、$\vec{r_2}$、$\vec{r_3}$…$\vec{r_n}$，则对某个属性 A 的值的累加有累加公式为：

$$A(\vec{r_i})=\sum_{j=1}^{n}A_j\frac{m_j}{\rho_j}W(\vec{r_i}-\vec{r_j},h)$$

其中：

W	光滑核函数
h	光滑核函数半径

密度推导

计算密度ρ，用ρ作为属性累加公式中的属性 A，化简离散粒子的密度表达式：

$$\rho(\vec{r_i})=\sum_{j=1}^{n}\rho_j\frac{m_j}{\rho_j}W(\vec{r_i}-\vec{r_j},h)=\sum_{j=1}^{n}m_jW(\vec{r_i}-\vec{r_j},h)$$

代入 Poly6 函数的 3D 下的表达式：

$$\rho(\vec{r_i})=\frac{315}{64\pi h^9}\sum_{j=1}^{n}m_j(h^2-|\vec{r_i}-\vec{r_j}|^2)^3$$

其中，若对于单一流体的模拟，粒子质量均为 m，可将 m 提出，简化表达式：

$$\rho(\vec{r_i})=m\frac{315}{64\pi h^9}\sum_{j=1}^{n}(h^2-|\vec{r_i}-\vec{r_j}|^2)^3 \tag{6}$$

压力推导

根据 N-S 方程中的压力项，将压力p作为累加公式中属性 A：

$$\vec{F}_i^{pressure}=-\nabla p(\vec{r_i})=-\sum_{j=1}^{n}p_j\frac{m_j}{\rho_j}\nabla W(\vec{r_i}-\vec{r_j},h)$$

由于位于不同压强区的两个粒子之间的作用力不相等，在计算中一般采用双方粒子的压强的算术平均取代单个粒子的压强：

$$\vec{F}_i^{pressure} = -\nabla p(\vec{r_i}) = -\sum_{j=1}^{n} \frac{m_j(p_i + p_j)}{2\rho_j} \nabla W(\vec{r_i} - \vec{r_j}, h)$$

对于单个粒子产生的压强p，可以利用理想气态状态方程计算：

$$p = \kappa(\rho - \rho_0)$$

其中：

ρ_0	流体的静态密度
κ	与流体相关的常数，且只和温度相关

代入 Spiky 函数的 3D 下的表达式：

$$\nabla W_{Spiky}(\vec{r}, h) = \frac{15}{\pi h^6} \nabla (h - r)^3 = -\vec{r} \frac{45}{\pi h^6 |\vec{r}|} (h - r)^2$$

代入 N-S 方程中的压力项：

$$-\frac{\nabla p(\vec{r_i})}{\rho_i} = m \frac{45}{2\rho_i \pi h^6} \sum_{i=1}^{n} [\frac{(p_i + p_j)(\vec{r_i} - \vec{r_j})}{\rho_j |\vec{r_i} - \vec{r_j}|} (h - |\vec{r_i} - \vec{r_j}|)^2] \tag{7}$$

粘度推导

根据 N-S 方程中的粘度项，将粘滞力μ作为累加公式中属性 A：

$$\vec{F}_i^{viscosity} = \mu \nabla^2 u(\vec{r_i}) = \mu \sum_{j=1}^{n} [\overrightarrow{u_j} \frac{m_j}{\rho_j} \nabla^2 W(\vec{r_i} - \vec{r_j}, h)]$$

由于位于不同压强区的两个粒子之间的作用力不相等，在计算中一般采用双方粒子的压强的算术平均取代单个粒子的压强:

$$\vec{F}_i^{viscosity} = \mu \nabla^2 u(\vec{r_i}) = \mu \sum_{j=1}^{n} [\frac{m_j(\overrightarrow{u_j} - \overrightarrow{u_i})}{2\rho_j} \nabla^2 W(\vec{r_i} - \vec{r_j}, h)]$$

代入 Visicosity 函数的 3D 下的表达式：

$$\nabla^2 W_{Viscosity}(\vec{r}, h) = \frac{15}{2\pi h^2} \nabla^2 (-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1) = \frac{45}{\pi h^6} (h - r)$$

代入 N-S 方程中的粘度项：

$$\frac{\mu \nabla^2 \vec{v}(\vec{r_i})}{\rho(\vec{r_i})} = m\mu \frac{45}{\rho_i \pi h^6} \sum_{j=1}^{n} [\frac{\overrightarrow{u_j} - \overrightarrow{u_i}}{\rho_j} (h - |\vec{r_i} - \vec{r_j}|)] \tag{8}$$

3. 最后公式

综上公式：

$$Substituted\ (5): a(\vec{r_i}) = \vec{g} + m\frac{45}{\rho_i \pi h^6}\sum_{i=1}^{n}[\frac{(p_i + p_j)(\vec{r_i} - \vec{r_j})}{2\rho_j|\vec{r_i} - \vec{r_j}|}(h - |\vec{r_i} - \vec{r_j}|)^2] + m\mu\frac{45}{\rho_i \pi h^6}\sum_{j=1}^{n}[\frac{\vec{u_j} - \vec{u_i}}{\rho_j}(h - |\vec{r_i} - \vec{r_j}|)] \tag{9}$$

对于后续版本所实现的多种流体交互，将公式修改为：

$$Substituted\ (9): a(\vec{r_i}) = \vec{g} + \frac{45}{\rho_i \pi h^6}\sum_{i=1}^{n}[m_j\frac{(p_i + p_j)(\vec{r_i} - \vec{r_j})}{2\rho_j|\vec{r_i} - \vec{r_j}|}(h - |\vec{r_i} - \vec{r_j}|)^2] + \mu\frac{45}{\rho_i \pi h^6}\sum_{j=1}^{n}[m_j\frac{\vec{u_j} - \vec{u_i}}{\rho_j}(h - |\vec{r_i} - \vec{r_j}|)] \tag{10}$$

三、 程序设计

1. 类图设计

最终目的是，仅通过创建一个流体类，通过初始化、添加粒子、更新粒子和获取粒子缓冲实现简易的函数调用，设计类图如下：

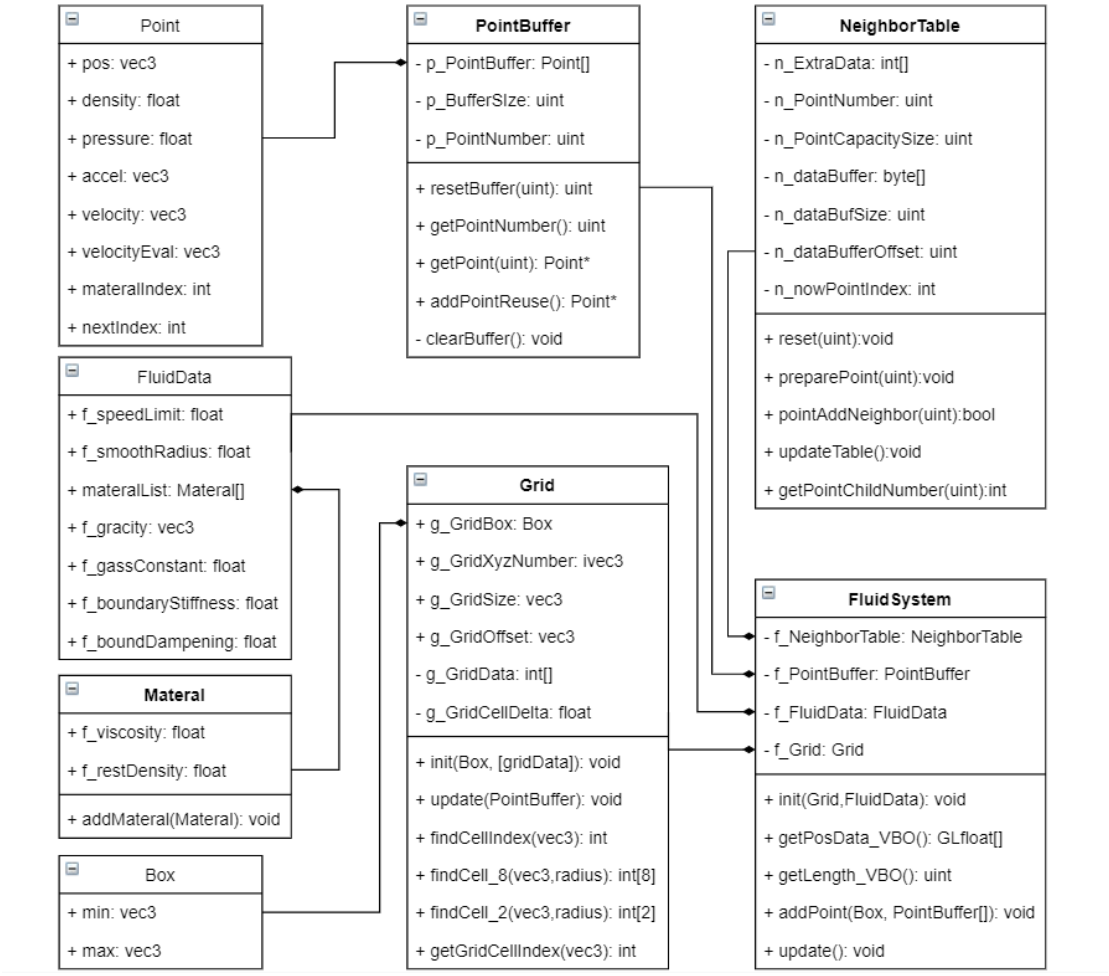


图 4 类图设计

其中：

- PointBuffer：粒子缓冲类，负责对粒子存储、索引和管理；
- Grid：网格类，负责网格的管理，用于存储粒子的哈希索引和管理空间网格的坐标；
- NeighborTable：邻接表类，用于存储上次粒子访问的在该粒子的光滑核函数半径内的粒子的索引的链表；
- Materal：材质类，用于管理默认的材质与自定义材质；

2. 主要函数的流程图设计

一、 粒子按帧更新函数

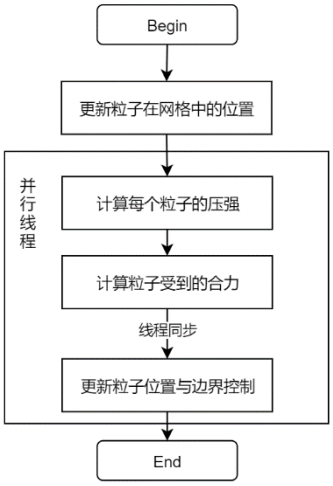


图 5 Update 函数流程图

二、 更新粒子密度函数

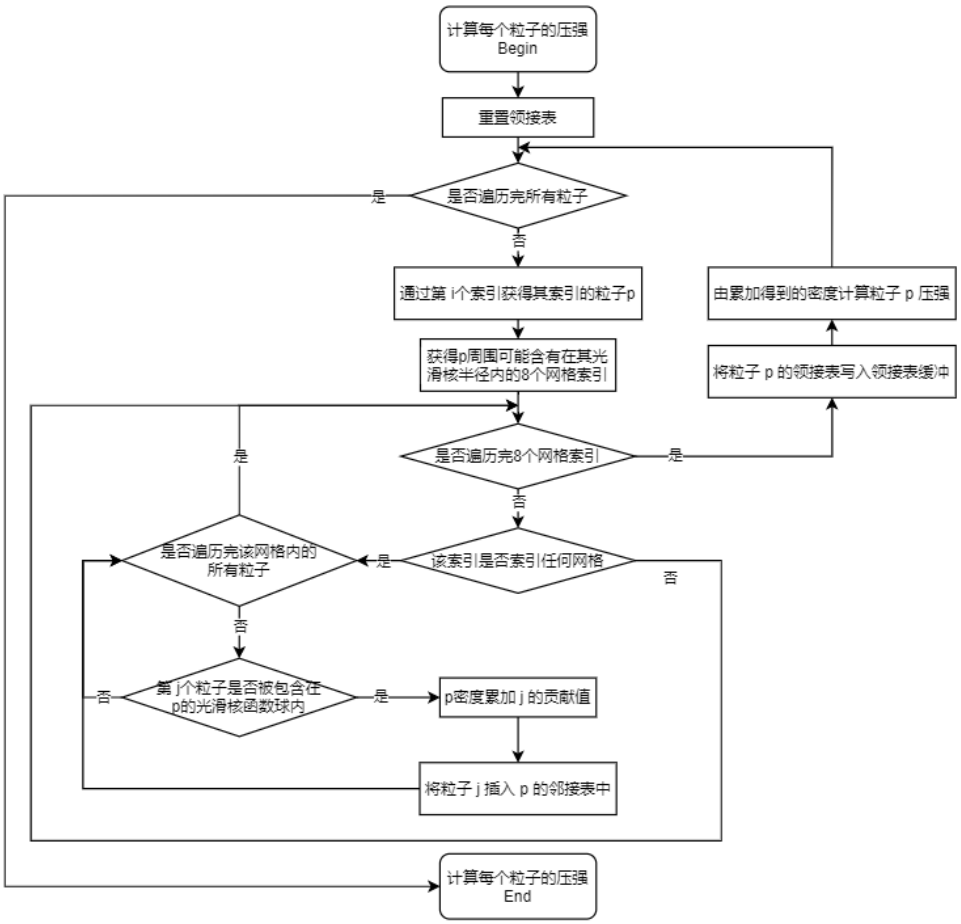


图 6 密度与邻接表维护函数流程图

三、 更新粒子受力函数

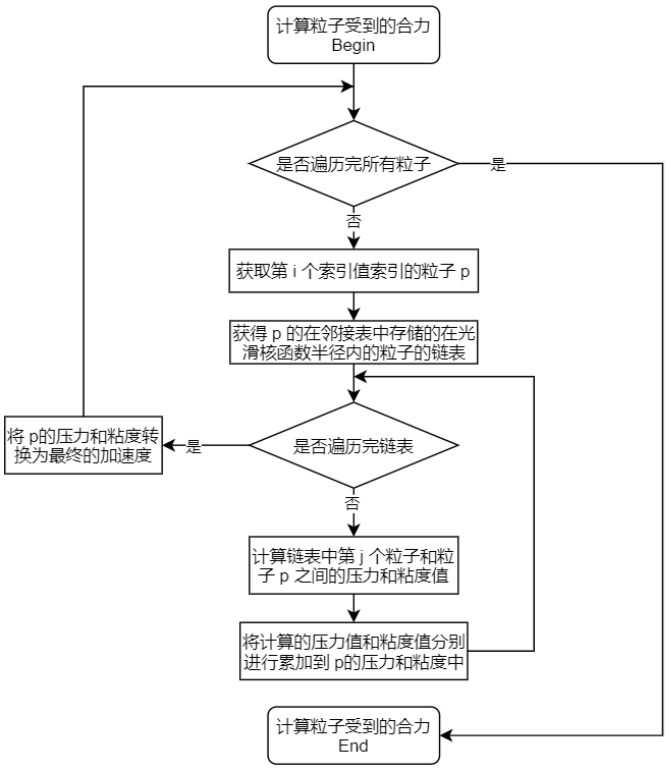


图 7 合力计算函数流程图

四、 更新粒子位移函数

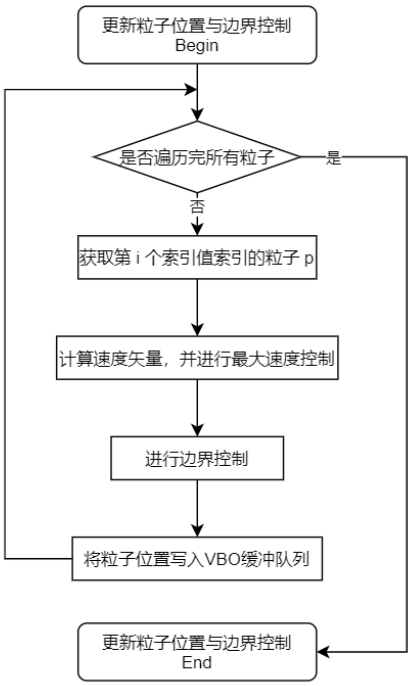


图 8 位移更新函数流程图

3. 流体类行为描述

一、 初始化流体与重置设置

流体模拟需要提供一个流体的生存区间，即流体盒，保证流体只在盒内流动。既然有了初始化，就要额外提供一个重置函数，方便内存回收。

```
void init
(
    unsigned short maxPointCounts,
    const glm::vec3& wallBox_min,
    const glm::vec3& wallBox_max,
    const glm::vec3& initFluidBox_min,
    const glm::vec3& initFluidBox_max,
    const glm::vec3& gravity
);
void reset();
```

其中：

- **maxPointCounts**：规范最大粒子数；
- **wallBox_min&max**：通过长方体的最大值坐标和最小值坐标确定一个唯一的流体盒；
- **initFluidBox_min&max**：通过长方体的最大值坐标和最小值坐标构造一个初始的流体块；
- **gravity**：确定全局重力（合外力）；

二、 构造流体块

仅通过初始化函数构造流体块是十分有局限的，这将大大降低了流体系统的自由度，因此，额外提供一组自由构造的行为接口给用户进行调用。

```
bool addPoint(glm::vec3 min, glm::vec3 max, glm::vec3 originVelocity);
bool addPoint(glm::vec3 min, glm::vec3 max, glm::vec3 originVelocity, MateralIndex materalIndex);
```

其中：

- **min&max**：通过长方体的最大值坐标和最小值坐标构造一个初始的流体块；
- **originVelocity**：初始化一个速度，可实现类似水枪射水的流体射出效果；
- **materalIndex**：（可选）默认为水流体，可自定义材质；

三、 滴答计数式构造流体块

但若只有构造流体行为接口，用户若想实现基于时间的“喷泉”流体，依然需要编写较多的判断代码。本模板提供了一个滴答计数的构造方法，写入循环中可实现基于滴答计数构造流体，并通过初速度计算滴答数，保证流体块的连续性，采用浮点设计，确保大速度小出口下的流体有喷射飞溅效果。

```
bool addPointTick(glm::vec3 min, glm::vec3 max, glm::vec3 originVelocity);
bool addPointTick(glm::vec3 min, glm::vec3 max, glm::vec3 originVelocity, MateralIndex materalIndex);
```

其中：

- **min&max**：通过长方体的最大值坐标和最小值坐标构造一个初始的流体块；

- **originVelocity**：初始化一个速度，可实现类似水枪射水的流体射出效果；
- **materalIndex**：（可选）默认为水流体，可自定义材质；

四、 添加材质

既然提供了自定义流体材质，自然需要提供添加材质的接口，方便用户调用，简化调用的复杂度，为用户体验保驾护航。

```
MateralIndex addMateral(Materal* newMateral);
```

其中：

- **newMateral**：通过 new 直接传入构建的新材质；

五、 获得绘制数据

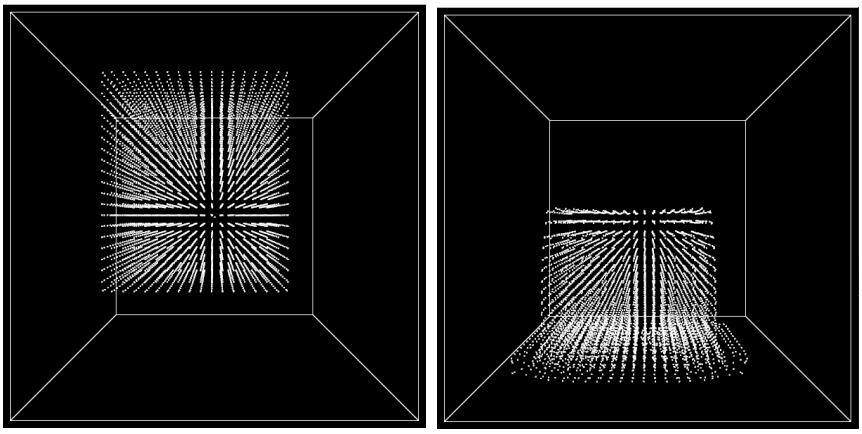
调用流体模板的用户存在自定义绘制方法和绘制场等的需求，因此要提供可供用户使用的顶点数据和相关信息，让用户可以快速得到数据。

```
//获取点的尺寸（字节）
unsigned int getPointStride()const { return sizeof(Point); }
//获取点的数量
unsigned int getPointCounts()const { return f_PointBuffer.getPointNumber(); }
//获取流体点缓存
const Point* getPointBuf()const { return f_PointBuffer.getPoint(0); }
//获得流体点的位置缓存
GLfloat *getPosData() { return &(posData[0]); }
```

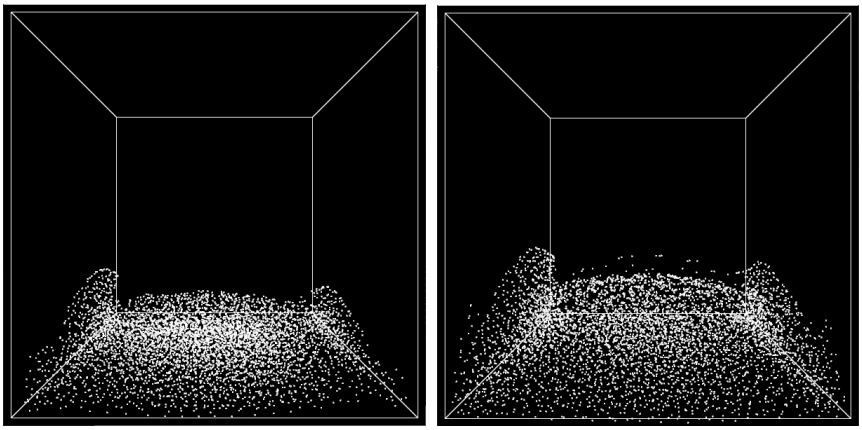
4. 程序结果

一、 构造流体自由下落

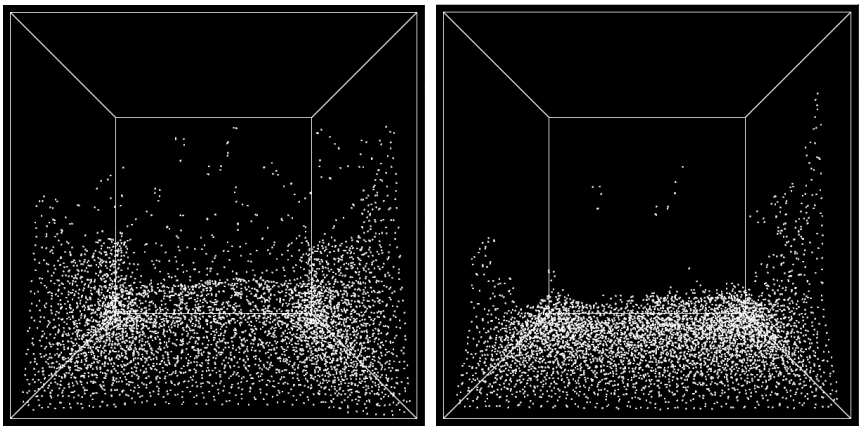
```
fluid.addPoint(glm::vec3(-15, -10, -15), glm::vec3(10, 20, 10), glm::vec3(0, 0, 0));
```



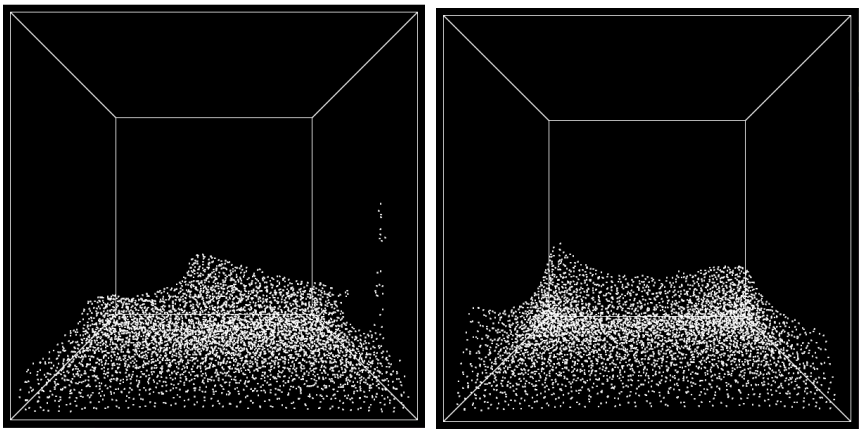
图组 1 初始化与下落中



图组 2 与地面的碰撞



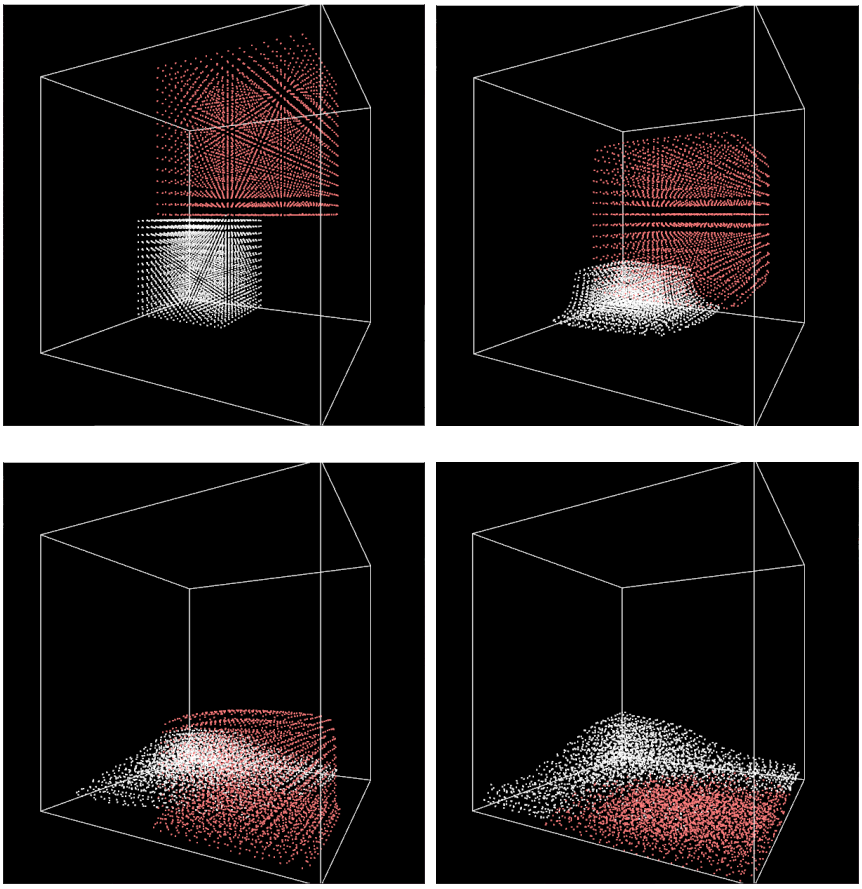
图组 3 碰撞后的飞溅



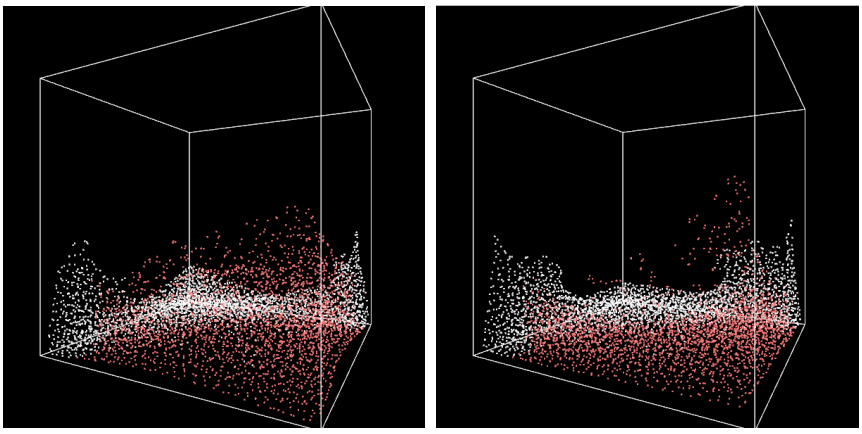
图组 4 飞溅后的波浪

二、 构造单材质双流体相互交互（第二同材质流体添加染色处理）

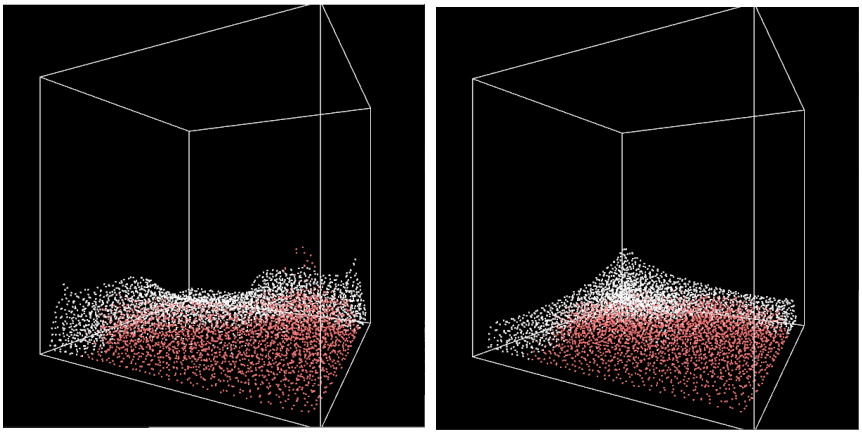
```
MateralIndex WaterRed = fluid.addMateral(
        new Materal(
                1.0f,    1000.f,              //水的参数
                glm::vec3(1, 0.5, 0.5)    //颜色
));//  红色的水
fluid.addPoint(glm::vec3(-20, -20, -20), glm::vec3(0, 0, 0), glm::vec3(0, 0, 0));
fluid.addPoint(glm::vec3(0, 0, 0), glm::vec3(20, 20, 20), glm::vec3(0, 0, 0), WaterRed);
```



图组 5 初始化与下落



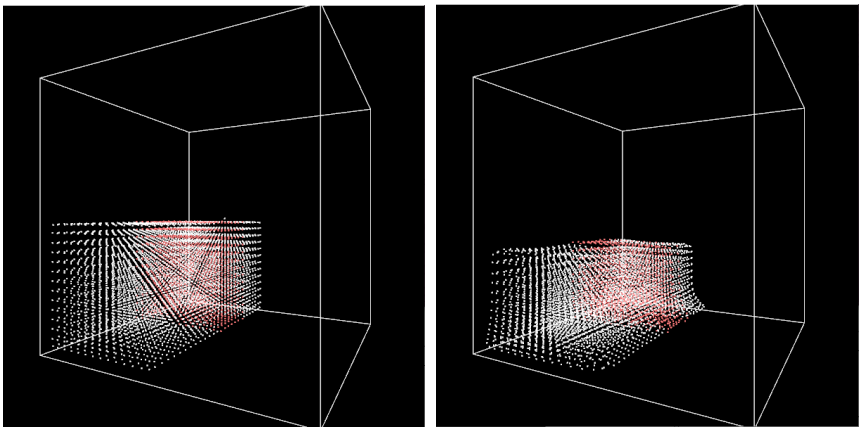
图组 6 碰撞



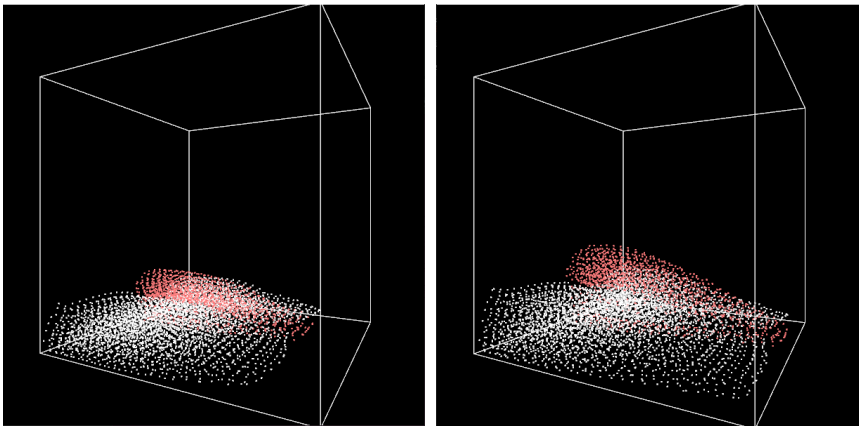
图组 7 产生卷向白色区域的波浪

三、 构造双材质双流体交互

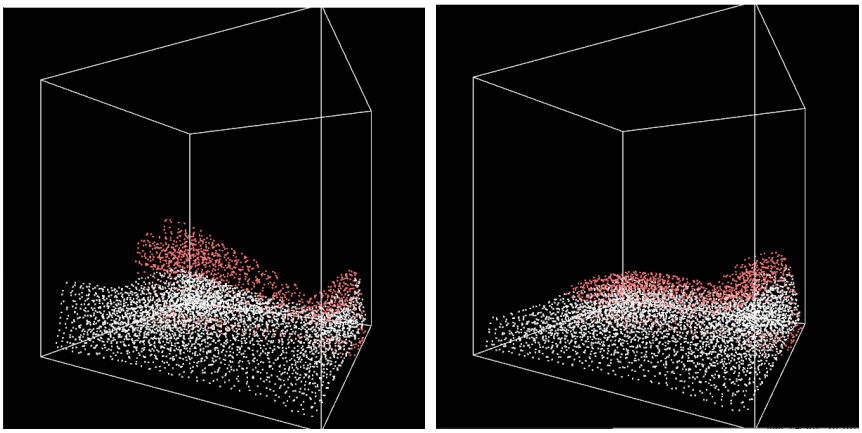
```
MateralIndex L_Water = fluid.addMateral(
    new Materal(
        1.0f,
        200.f,
        glm::vec3(1, 0.5, 0.5)
    )
);// 轻水，比水更轻，粘度相同，设为红色
fluid.addPoint(glm::vec3(-20, -20, -10), glm::vec3(0, 0, 0), glm::vec3(0, 0, 0), L_Water);
fluid.addPoint(glm::vec3(-20, -20, -20), glm::vec3(0, 0, -10), glm::vec3(0, 0, 0));
fluid.addPoint(glm::vec3(-20, -20, 0), glm::vec3(0, 0, 20), glm::vec3(0, 0, 0));
```



图组 8 初始化



图组 9 轻流体被挤到最上层

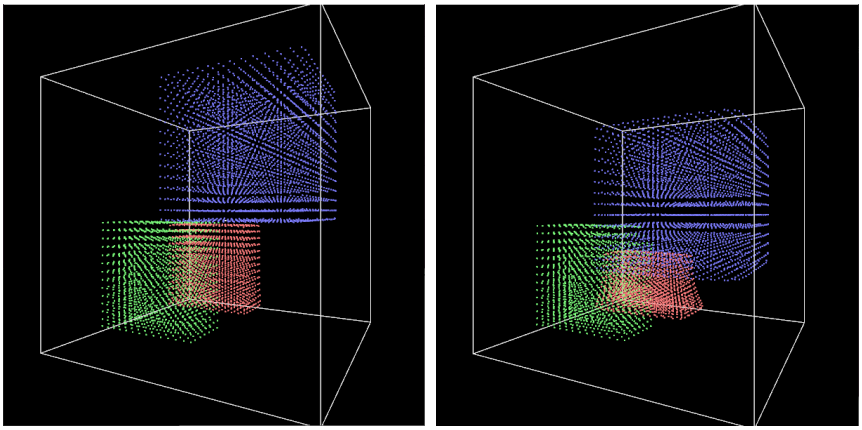


图组 10 轻流体最终漂浮在最上层

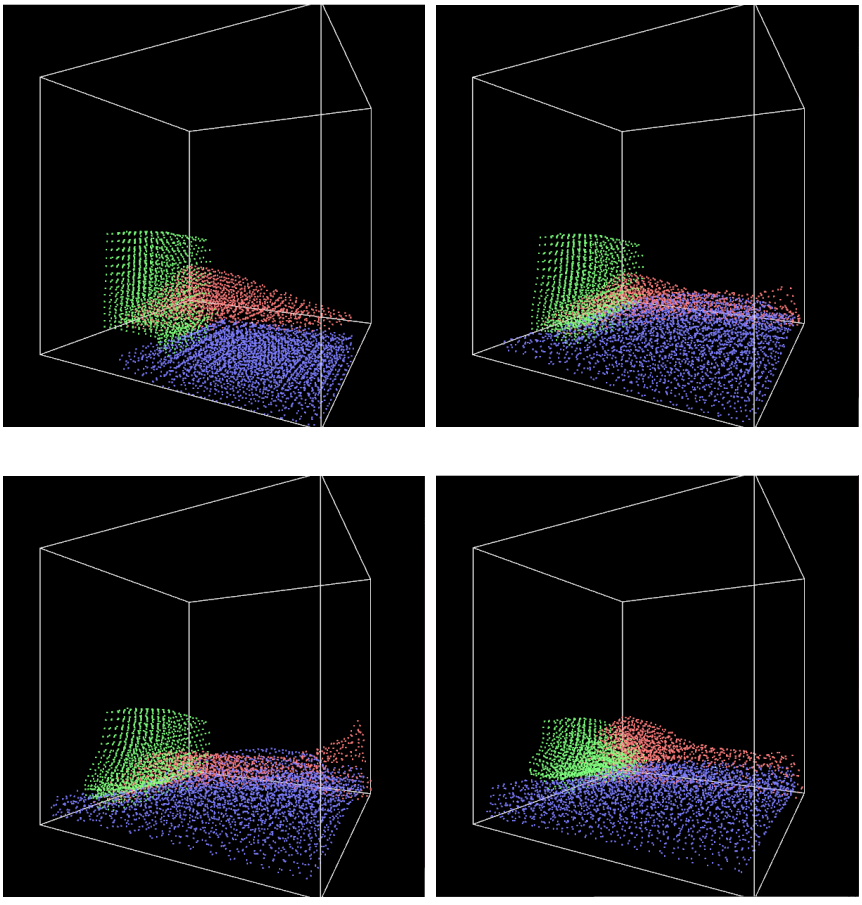
四、 **构造多材质多流体交互**

```
MateralIndex H_Oil = fluid.addMateral(
    new Materal(
        30.0f,
        5000.f,
        glm::vec3(0.5, 0.5, 1)   //蓝色
    )
);// 重油，比水更重，粘度更大，在标准大气压下，相对而言不稳定
MateralIndex L_Oil = fluid.addMateral(
    new Materal(
        30.0f,
        200.f,
        glm::vec3(0.5, 1, 0.5)   //绿色
    )
);// 轻油，比水更轻，粘度更大，在标准大气压下，相对而言更稳定
MateralIndex L_Water = fluid.addMateral(
    new Materal(
        1.0f,
```

```
            200.f,
            glm::vec3(1, 0.5, 0.5)   //红色
        )
    );// 轻水，比水更轻，粘度相同，在标准大气压下，相对而言比水更稳定
    fluid.addPoint(glm::vec3(-20, -20, 0), glm::vec3(0, 0, 10), glm::vec3(0, 0, 0), L_Oil);
    fluid.addPoint(glm::vec3(-20, -20, -20), glm::vec3(0, 0, -10), glm::vec3(0, 0, 0), L_Water);
    fluid.addPoint(glm::vec3(0, 0, 0), glm::vec3(20, 20, 20), glm::vec3(0, 0, 0), H_Oil);
```



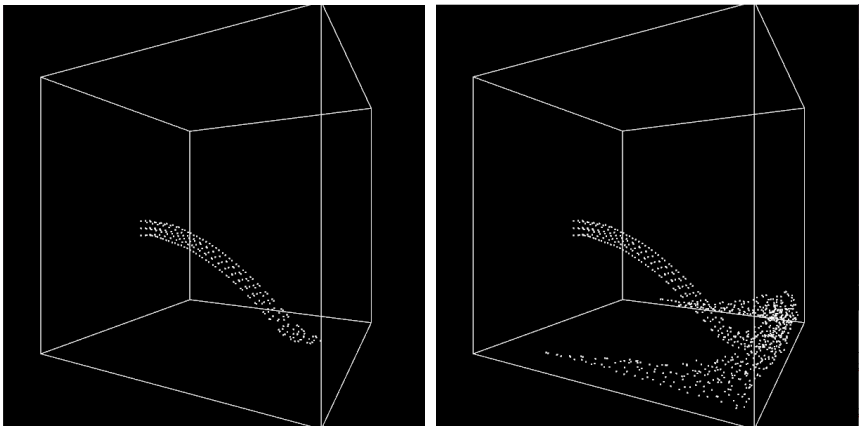
图组 11 初始化



图组 12 碰撞

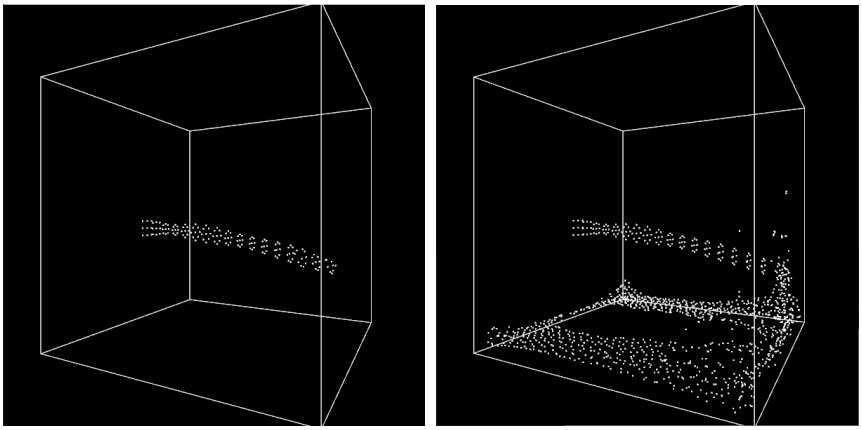
五、 **构造喷射流体**

```
while (!glfwWindowShouldClose(window))
{
    fluid.addPointTick(glm::vec3(-20, -4, -4), glm::vec3(-20, 0, 0), glm::vec3(2, 0, 0));
    …
}
```



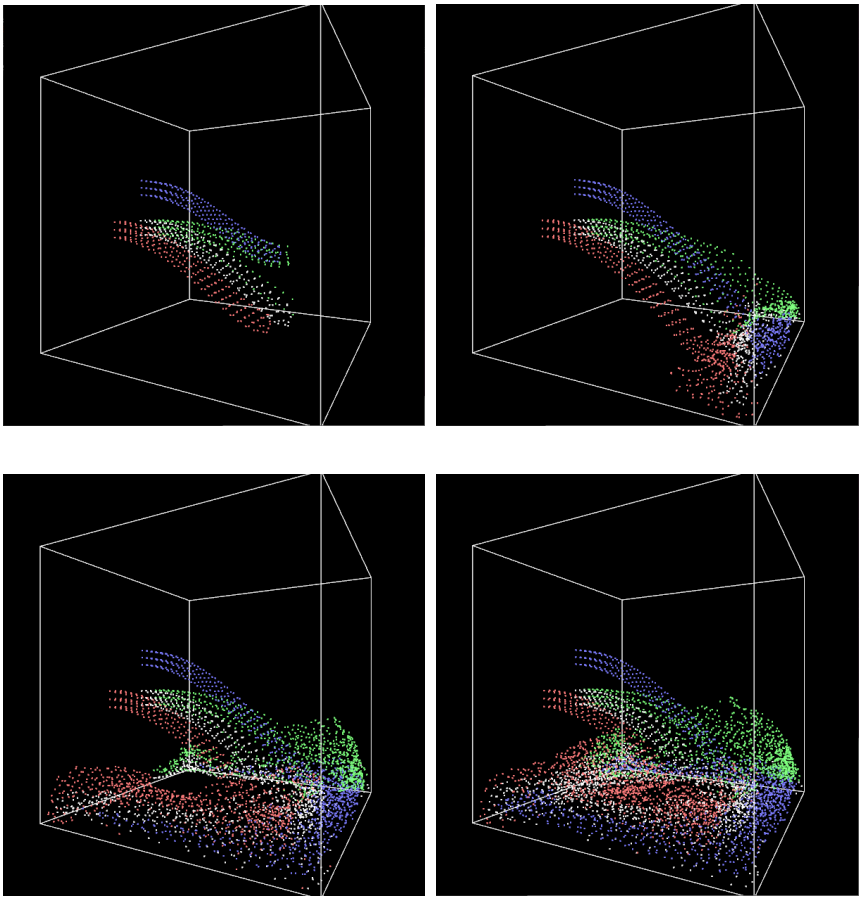
图组 13 速度为 2

```
while (!glfwWindowShouldClose(window))
{
    fluid.addPointTick(glm::vec3(-20, -4, -4), glm::vec3(-20, 0, 0), glm::vec3(3, 0, 0));
    …
}
```



图组 14 速度 3

```
while (!glfwWindowShouldClose(window)) {
    fluid.addPointTick(glm::vec3(-20, -4, 4), glm::vec3(-20, 0, 8), glm::vec3(2, 0, 0), L_Water);
    fluid.addPointTick(glm::vec3(-20, -4, -8), glm::vec3(-20, 0, -4), glm::vec3(2, 0, 0), L_Oil);
    fluid.addPointTick(glm::vec3(-20, 4, -4), glm::vec3(-20, 8, 0), glm::vec3(2, 0, 0), H_Oil);
    fluid.addPointTick(glm::vec3(-20, -4, -4), glm::vec3(-20, 0, 0), glm::vec3(2, 0, 0));
}
```



图组 15 多种流体

四、［参考文献］

[1] 程志宇,徐国庆,张岚斌,等.SPH 真实感流体交互模拟改进算法[J].武汉工程大学学报,2019,41(3):303-306.

[2] 袁志勇,徐标,廖祥云 基于 SPH 方法的快速逼真流体表面张力仿真[J].武汉大学计算机学院,中国科学院深圳先进技术研究院,2019,42(9),2062-2075.

[3] MONAGHAN J J.Simulating free surfaceflows with SPH[J]. Journal of Computationa Physicsl，1994,110(2):399-406.

[4] SCHECHTER H,BRIDSON R.Ghost SPH for animatingwater[J]. ACM Transactions on Graphics,2012,31(4):Article No.61.

[5] Yang M,Li X,Liu Y,et al.A novel surface tension formulation for SPH fluid simulation[J]. Visual Computer International Journal of Computer Graphics，2017，33(5)：1-10.