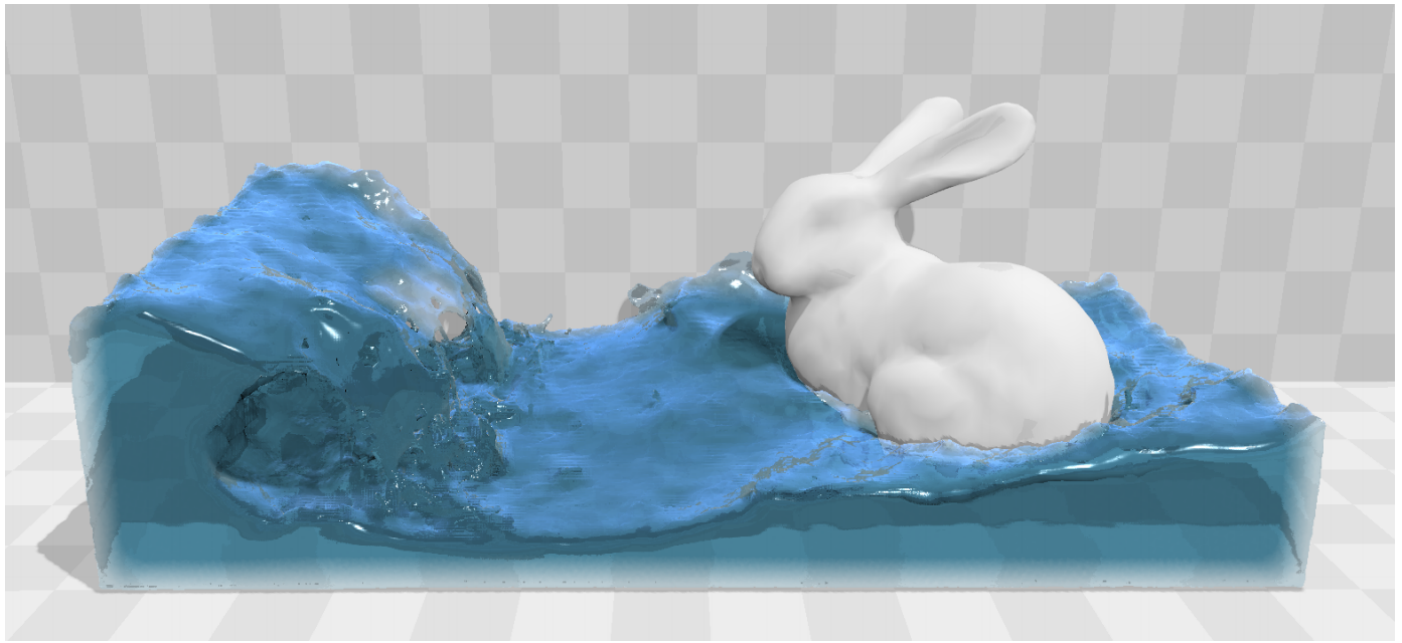
[Note] Position Based Fluids

2020-09-02 | 2021-02-03 | Computer Graphics , Physics Simulation , Fluid Simulation |
26
8.8k | 22 分钟



> 原论文： M.Macklin and M. Müller. Position Based Fluids. ACM Transactions on Graphics, 2013.

1. Introduction

对于互动的环境来说，流体模拟的稳定性非常重要。Smoothed Particle Hydrodynamics (SPH) 是一个在互动环境中被广泛使用的Particle-Based 方法，然而他却有个致命的缺点：当粒子周围的相邻粒子过少时会很难维持住这个算法的稳定性，尤其以在流体表面、或是在边缘的流体粒子更常发生。将每个Time step 调到足够小，或是增加足够的粒子数量即使可避免掉这项问题，却会大幅度增加计算成本。

Position Based Dynamics (PBD) 在游戏开发或是影片制作中都是很受欢迎的一套物理模拟方式，作者选择使用PBD 正是因为其具有Unconditionally Stable 以及稳定性等性质。这篇Paper 将介绍如何使用PBD Framework 来模拟Incompressible flow，并克服上述在Free Surfaces 发生Particle Deficiency 的问题。

2. Related Work

- Muller [2003] 等人在Particle-Based Fluid Simulation for Intereactive Applicatoins中提出使用Smoothed Particle Hydrodynamics (SPH) 模拟具有Viscosity 跟Surface Tension 的流体
- 为了维持住Incompressibility，Weakly Compressible SPH (WCSPH) [2007] 与标准SPH 所使用的Stiff Equation 会大大限制住Time step 的长度。
- Predictive-Corrective Incompressible SPH [2009] 利用Iterative Jacobi-style 方法，藉由不断迭代、累积流体压力并逐步施力的方式，来确保流体能够在较长的Time step 上也能够稳定存在，而不需要额外设置Stiffness value 且可以分散掉不断矫正相邻粒子密度的计算成本。

- Bodin [2012] 将Incompressibility 组成一个Velocity Constraints 的线性方程，并使用Gauss-Seidel Iteration 解出线性方程来确保流体密度的一致。相反的，Position-Based 跟PCISPH 则是使用类似Jacobi Iteraion 的方法，解出非线性方程，并不断重新估计误差与梯度。
- Hybrid Method，像是Fluid Implicit-Particle (FLIP) 则是使用Grid 来解出流体压力并将流体速度扩散到粒子上模拟。Zhu 与Bridson [2005] 结合PIC/FLIP 方法。Raveendran [2011] 使用Coarse grid 方法混合SPH 趋近divergence free velocity field。
- Clavet [2005] 使用Position Based 方法模拟Viscoelastic Fluids。但是他们的方法只是Conditionally Stable。
- Position Based Dynamics [2007] 基于Verlet integration 提供一个在游戏中模拟物理的方法。

3. Enforcing Incompressibility

对每一个粒子使用Density Constraint 来确保流体密度不变。Constraint 是每个粒子位置与邻近粒子位置的函数，位置写作$\mathbf{p}_1, \cdots, \mathbf{p}_n$，以下是对第$i$个粒子的Constraint Function：

$$C_i\left(\mathbf{p}_1, \cdots, \mathbf{p}_n\right) = \frac{\rho_i}{\rho_0} - 1 \tag{1}$$

其中ρ_0是静止密度(Rest Density)，ρ_i是由SPH density estimator 计算出来的粒子密度:

$$\rho_i = \sum_j m_j W\left(\mathbf{p}_i - \mathbf{p}_j, h\right) \tag{2}$$

使用**Poly6**作为Density Estimator 的Kernel。并使用**Spiky Kernel**计算gradient

> **注**: 由于原论文中考虑粒子质量m_j为定值且相同，因此在后续推倒公式中皆省略不写，但是在本篇Note 中会将他归至原位。

> **注:** Poly 6 为6^{th}degree polynomial kernel 的简称，以下为其函数:
>
> $$W_{\text{poly6}}(\mathbf{r},h)=\frac{315}{64\pi h^9}\begin{cases}\left(h^2-\|\mathbf{r}\|^2\right)^3, & 0\le\|\mathbf{r}\|\le h\\ 0, & \|\mathbf{r}\|>h\end{cases}$$
>
> Poly 6 的Gradient:
>
> $$\nabla W_{\text{poly6}}(\mathbf{r},h)=-\frac{945}{32\pi h^9}\mathbf{r}\left(h^2-\|\mathbf{r}\|^2\right)^2$$
>
> Poly 6 的Laplacian:
>
> $$\nabla^2 W_{\text{poly6}}(\mathbf{r},h)=-\frac{945}{32\pi h^9}\left(h^2-\|\mathbf{r}\|^2\right)\left(3h^2-7\|\mathbf{r}\|^2\right)$$

> **注:** Spiky Kernel 函数:
>
> $$W_{\text{spiky}}(\mathbf{r},h)=\frac{15}{\pi h^6}\begin{cases}\left(h-\|\mathbf{r}\|\right)^3, & 0\le\|\mathbf{r}\|\le h\\ 0, & \|\mathbf{r}\|>h\end{cases}$$

Spiky Kernel 的Gradient:

$$\nabla W_{\text{spiky}}(\mathbf{r}, h) = -\frac{45}{\pi h^6}\frac{\mathbf{r}}{\|\mathbf{r}\|}(h - \|\mathbf{r}\|)^2,$$
$$\lim_{r\to 0^-} \nabla W_{\text{spiky}}(\mathbf{r}, h) = \frac{45}{\pi h^6},$$
$$\lim_{r\to 0^+} \nabla W_{\text{spiky}}(\mathbf{r}, h) = -\frac{45}{\pi h^6},$$

Spiky Kernel 的Laplacian:

$$\nabla^2 W_{\text{spiky}}(\mathbf{r}, h) = -\frac{90}{\pi h^6}\frac{1}{\|\mathbf{r}\|}(h - \|\mathbf{r}\|)(h - 2\|\mathbf{r}\|),$$
$$\lim_{r\to 0} \nabla^2 W_{\text{spiky}}(\mathbf{r}, h) = -\infty$$

PBD 的目标在于找出粒子位置的修正项$\Delta\mathbf{p}$来满足Constraint:

$$C(\mathbf{p} + \Delta\mathbf{p}) = 0 \tag{3}$$

使用牛顿法来求解:

$$\Delta\mathbf{p} \approx \nabla C(\mathbf{p})\lambda \tag{4}$$

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla C^T \Delta\mathbf{p} = 0 \tag{5}$$
$$\approx C(\mathbf{p}) + \nabla C^T \nabla C\lambda = 0 \tag{6}$$

> **注**:λ是一个变量。这个思路其实很简单，我们要追求每一个Time step 都要满足约束函数C，但是粒子的位置$\mathbf{p}$不一定会满足，因此我们必须求出修正项$\Delta\mathbf{p}$来修正粒子位置直到满足约束条件。而修正项可以定义为往$\nabla C(\mathbf{p})$方向乘上一个微小变量λ。根据Taylor 一阶展开式，$C(\mathbf{p}+\Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla C^T \Delta\mathbf{p}$最后再将$\Delta\mathbf{p}$带入，得到式(6)。

> **注**:∇C^T中的T是Transpose，由于∇C是向量，$\nabla C^T \Delta\mathbf{p}$表示的是向量$\nabla C$与向量$\Delta\mathbf{p}$的内积，因此$\nabla C$会需要Transpose。

根据SPH 方法，我们可以将粒子i对粒子k的约束函数C梯度定义为：

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_0} \sum_j m_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{7}$$

根据i与k的关系分为:

$$\nabla_{\mathbf{p}_k} C_i = \frac{1}{\rho_0} \begin{cases} \sum_j m_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) & \text{if } k = i \\ -m_j \nabla_{\mathbf{p}_k} W(\mathbf{p}_i - \mathbf{p}_j, h) & \text{if } k = j \end{cases} \tag{8}$$

> **注**:i是自身粒子，j是邻近粒子。由于Kernel$W$是$\mathbf{p}_i - \mathbf{p}_j$的函数，因此对于其他$k \neq i$与$k \neq j$的粒子来说其梯度皆为$0$，只有$k = i$与$k = j$会满足条件。至于对$\mathbf{p}_i$或$\mathbf{p}_j$的梯度差别只在于方向：由于对象都是座标$\mathbf{p}$，因此梯度皆为针对座标每个维度作微分，只是由于$W$是$\mathbf{p}_i - \mathbf{p}_j$的函数，如果是对$\mathbf{p}_i$取$W$梯度则根据Chain Rule得到

> $\nabla_{\mathbf{p}_i} W = W' \nabla_{\mathbf{p}_i} (\mathbf{p}_i - \mathbf{p}_j) = W'$，而对$\mathbf{p}_j$取$W$梯度则得到$\nabla_{\mathbf{p}_j} W = W' \nabla_{\mathbf{p}_j} (\mathbf{p}_i - \mathbf{p}_j) = -W'$，因此式(8) 在$k = j$的情况下加了一个负号，将方向导正。

将式(8) 代入式(6) 求得变量λ:

$$\lambda_i = -\frac{C_i(\mathbf{p}_i, \cdots, \mathbf{p}_j)}{\sum_k \left| \nabla_{\mathbf{p}_k} C_i \right|^2} \tag{9}$$

> **注:** $\left| \nabla_{\mathbf{p}_k} C_i \right|^2$是因为式(6) 中的$\nabla C^T \nabla C$

由于约束函数(1) 是非线性，当粒子逐渐分离时，Kernel 的边缘值会趋近0，导致式(9) 的梯度分母逐渐趋近0，进而造成整体模拟的不稳定性。使用Constraint Force Mixing (CFM) 则可以避免这个情况，将式(6) 改写为:

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla C^T \nabla C \lambda + \varepsilon \lambda = 0 \tag{10}$$

其中ε是使用者自订的Relaxation Parameter，λ_i变成:

$$\lambda_i = -\frac{C_i(\mathbf{p}_i, \cdots \mathbf{p}_j)}{\sum_k \left| \nabla_{\mathbf{p}_k} C_i \right|^2 + \varepsilon} \tag{11}$$

> **注:**简而言之就是在分母的部份加上一个不为0的微小常数来确保分母不会为0

最后修正项$\Delta \mathbf{p}_i$的函数变成:

$$\Delta \mathbf{p}_i = \frac{1}{\rho_0} \sum_j \big(\lambda_i + \lambda_j\big) m_j \nabla W\big(\mathbf{p}_i - \mathbf{p}_j, h\big) \tag{12}$$

注:

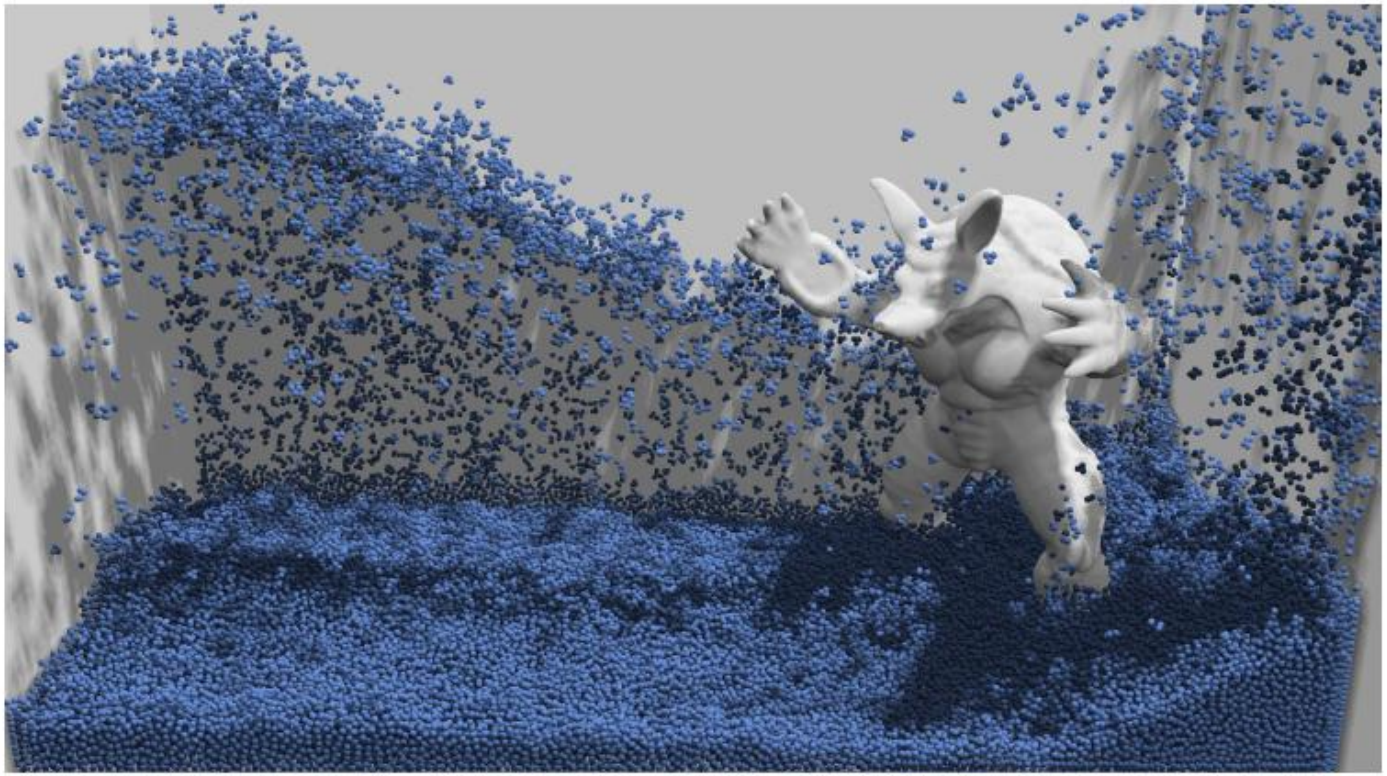
$$\begin{aligned}
\Delta \mathbf{p}_i &= \lambda_i \nabla_{\mathbf{p}_i} C_i + \sum_j \lambda_j \nabla_{\mathbf{p}_j} C_i && (9)\\
&= \frac{1}{\rho_0} \sum_j \lambda_i m_j \nabla_{\mathbf{p}_i} W\big(\mathbf{p}_i - \mathbf{p}_j, h\big) + \Big(- \frac{1}{\rho_0} \sum_j \lambda_j m_j \nabla_{\mathbf{p}_j} W\big(\mathbf{p}_i - \mathbf{p}_j, h\big)\Big) && (10)\\
&= \frac{1}{\rho_0} \sum_j \lambda_i m_j \nabla_{\mathbf{p}_i} W\big(\mathbf{p}_i - \mathbf{p}_j, h\big) + \frac{1}{\rho_0} \sum_j \lambda_j m_j \nabla_{\mathbf{p}_i} W\big(\mathbf{p}_i - \mathbf{p}_j, h\big) && (11)\\
&= \frac{1}{\rho_0} \sum_j \big(\lambda_i + \lambda_j\big) m_j \nabla_{\mathbf{p}_i} W\big(\mathbf{p}_i - \mathbf{p}_j, h\big) && (12)
\end{aligned}$$

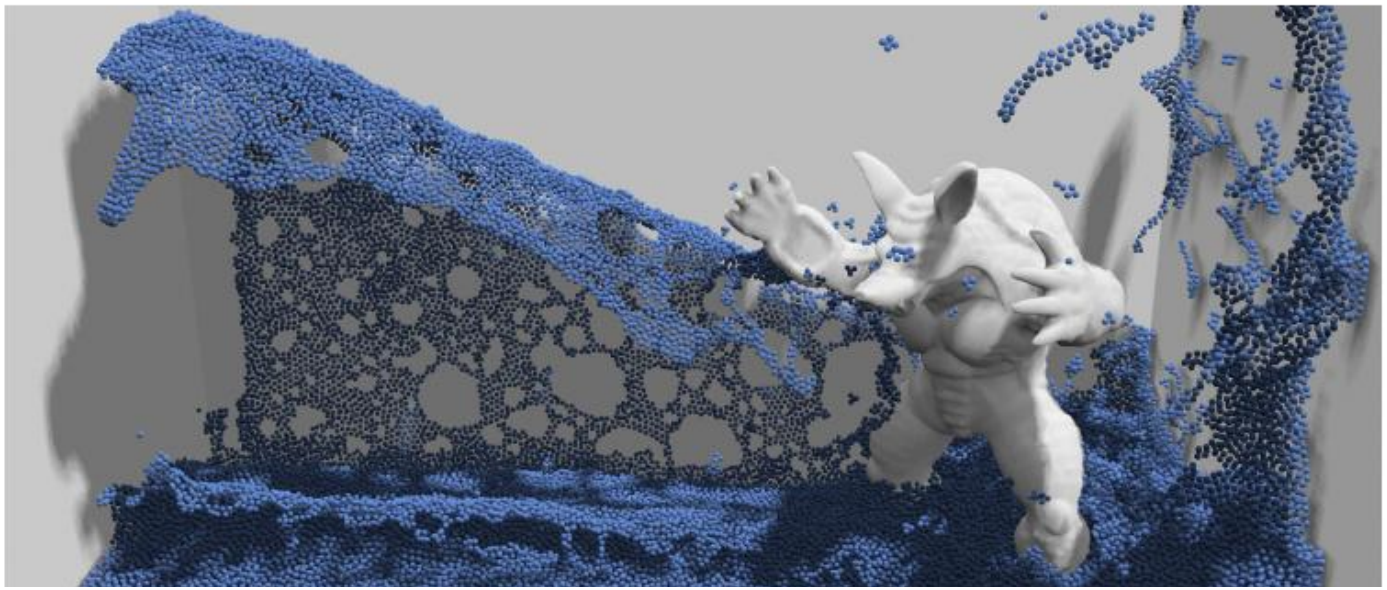
其中第2 行到第3 行是因为:

$$\because \nabla_{\mathbf{p}_j} W\big(\mathbf{p}_i - \mathbf{p}_j, h\big) = -\nabla_{\mathbf{p}_i} W\big(\mathbf{p}_i - \mathbf{p}_j, h\big)$$

4. Tensile Instability

SPH 中常见的问题就是因邻近粒子数量不足无法达到Rest Density，而产生负压力所导致的粒子的聚集效应(粒子间的排斥力变成吸力)。





上图为聚集效应产生的不自然现象/ 下图为施加人工压力项的结果

一种解决方案是对压力做Clamping 让压力不为负值，但是这会让粒子的内聚力衰弱。

> **注:**
>
> ```
> clamp(x) {
>     return x>0 ? x:0;
> }
> ```

其他解决方案:

- Clavet [2005] 使用second near pressure term
- Alduan and Otaduy [2011] 使用discrete element (DEM) forces
- Schechter and Bridson [2012] 使用ghost particles

此篇论文使用Monaghan [2000] 的方法，加上一个人工压力项:

$$s_{corr} = -k\left(\frac{W(\mathbf{p}_i - \mathbf{p}_j, h)}{W(\Delta\mathbf{q}, h)}\right)^n \tag{13}$$

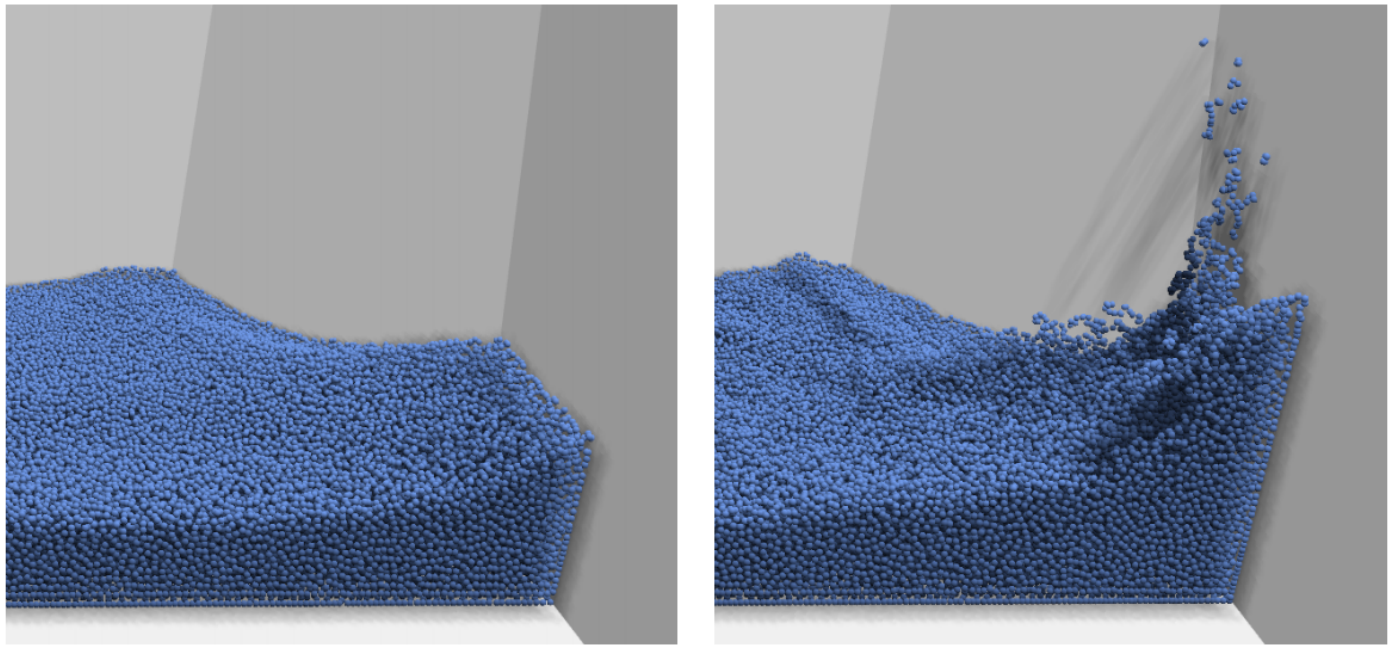
其中$\Delta\mathbf{q}$是固定的距离，k是微小的正常数。通常取$|\Delta\mathbf{q}| = 0.1h, \cdots, 0.3h$，$k = 0.1$，$n = 4$。将此项纳入修正项:

$$\Delta\mathbf{p}_i = \frac{1}{\rho_0}\sum_j \big(\lambda_i + \lambda_j + s_{corr}\big)m_j\nabla W\big(\mathbf{p}_i - \mathbf{p}_j, h\big) \tag{14}$$

这个丑陋的项会保持粒子略低于Rest Density，最终产生出类似Surface Tension 的效果。

5. Vorticity Confinement and Viscosity

PBD 方法通常会产生额外的Damping 造成涡流快速消散。Fedkiw [2001] 引进Vorticity Confinement 方法来克服模拟烟雾时的消散问题(Numerical Dissipation)，后来由Lentine [2011] 当作Energy Conserving 引入流体模拟。Hong [2008] 展示如何将Vorticity Confinement 导入Hybrid 的模拟方法。



左边没有加Vorticity Confinement / 右边有加Vorticity Confinement

此篇文章使用Vorticity Confinement 来补充流失的能量。此方法需要先计算粒子位置的Vorticity，Monaghan [1992]:

$$\omega_i = \nabla \times \mathbf{v} = \sum_j m_j \mathbf{v}_{ij} \times \nabla \mathbf{p}_j W\left(\mathbf{p}_i - \mathbf{p}_j, h\right) \tag{15}$$

$$\mathbf{f}_i^{\text{vorticity}} = \varepsilon\left(\mathbf{N} \times \omega_i\right) \tag{16}$$

其中$\mathbf{N} = \frac{\eta}{|\eta|}$，$\eta = \nabla|\omega|_i$，$\mathbf{v}_{ij} = \mathbf{v}_j - \mathbf{v}_i$。

注:这边有点小Tricky，原论文里面也没有写清楚，首先这边目标是加速粒子的涡流速度，因此他先算出旋度向量ω_i，也就是遵守右手定则的旋转轴心方向(大姆指指的方向)。接着用ω_i的Gradient L2-norm 算出往旋转轴心方向的向量η，算出单位向量$\mathbf{N}$，最后再用外积算出旋转方向并乘上ε算出Vorticity forces

注:

$$\eta = \nabla|\omega| = \nabla\left(\sum_{k=1}^{n}\omega_k^2\right)^{\frac{1}{2}} \tag{13}$$

$$= \sum_{j=1}^{n}\frac{\partial}{\partial\omega_j}\left(\sum_{k=1}^{n}\omega_k^2\right)^{\frac{1}{2}}\hat{\omega}_j \tag{14}$$

$$= \frac{1}{2}\sum_{j=1}^{n}\frac{2\omega_j}{\left(\sum_{k=1}^{n}\omega_k^2\right)^{\frac{1}{2}}}\hat{\omega}_j \tag{15}$$

$$= \sum_{j=1}^{n}\frac{\omega_j}{|\omega|}\hat{\omega}_j \tag{16}$$

此外，此篇论文也使用XSPH viscosity 来模拟流体的黏滞力:

$$\mathbf{v}_i^{new} = \mathbf{v}_i + c\sum_j m_j\mathbf{v}_{ij}\cdot W\left(\mathbf{p}_i - \mathbf{p}_j, h\right) \tag{17}$$

其中，参数c通常设为0.1。

6. Algorithm

Algorithm 1 Simulation Loop

for all particles i **do**
  apply forces $\mathbf{v}_i \Leftarrow \mathbf{v}_i + \Delta t \mathbf{f}_{ext}(\mathbf{x}_i)$
  predict position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i + \Delta t \mathbf{v}_i$
end for
for all particles i **do**
  find neighboring particles $N_i(\mathbf{x}_i^*)$
end for
while $iter < solverIterations$ **do**
  for all particles i **do**
    calculate λ_i
  end for
  for all particles i **do**
    calculate $\Delta \mathbf{p}_i$
    perform collision detection and response
  end for
  for all particles i **do**
    update position $\mathbf{x}_i^* \Leftarrow \mathbf{x}_i^* + \Delta \mathbf{p}_i$
  end for
end while
for all particles i **do**
  update velocity $\mathbf{v}_i \Leftarrow \frac{1}{\Delta t}\left(\mathbf{x}_i^* - \mathbf{x}_i\right)$
  apply vorticity confinement and XSPH viscosity
  update position $\mathbf{x}_i \Leftarrow \mathbf{x}_i^*$
end for

7. Rendering

使用GPU based ellipsoid splatting technique 详情见 液体渲染：一种屏幕空间方法

相关文章

- [Note] AuTO: Scaling Deep Reinforcement Learnign for Datacenter-Scale Automatic Traffic Optimization
- [Note] Quantum Computation and Quantum Information - Chapter 3: Introduction to compuer science
- [Note] Quantum Computation and Quantum Information - Midterm exam
- [Note] Quantum Computation and Quantum Information - Final exam
- [Note] Quantum Computation and Quantum Information - Chapter 4: Quantum Circuits

Note Paper Note Computer Graphics Physics Simulation Fluid Simulation

‹ [Note] 神的语言Metaprogramming: one_of

[Note] AuTO: Scaling Deep Reinforcement Learnign for Datacenter-Scale Automatic Traffic Optimization ›

0笔评论

访客

写点什么

支援Markdown 语法

使用GitHub 登入

预览

成为首个留言的人吧!