同济大学
本科毕业设计

不可压流体的模拟、渲染和应用研究

1452286　朱可仁
指导老师：赵君峤

内 容

- 课题背景和工作概述
- 流体的并行化模拟
- 流体的并行化表面重建和渲染
- 演示

流体的范畴

液体[1]

烟尘[2]

冰淇凌[3]

物理真实、大计算量

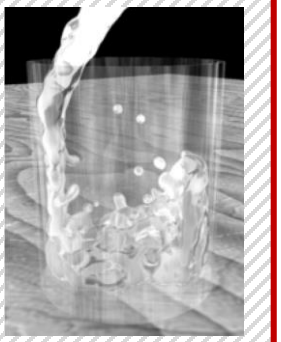
离线算法

[1] Realflow. http://docklandsmedia.com/course/realflow-training/

[2] A. Chern, et al., “Schrödinger’s smoke,” *ACM Trans. Graph.*, vol. 35, no. 4, pp. 1–13, Jul. 2016.

[3] C. Jiang, et al., “The affine particle-in-cell method,” *ACM Trans. Graph.*, vol. 34, no. 4, p. 51:1-51:10, Jul. 2015.

本文的研究



液体[4]

爆炸[5]

烟尘[6]

视觉真实、小计算量

实时算法

[4] M. Müller, el at., “Particle-Based Fluid Simulation for Interactive Applications,” *Proc. 2003 ACM SIGGRAPH*, no. 5

[5] D. Q. Nguyen, el at., “Physically based modeling and animation of fire,” *ACM Trans. Graph.*, vol. 21, no. 3, 2002.

[6] R. Fedkiw, el at., “Visual simulation of smoke,” *Proc. 2001 SIGGRAPH*, pp. 15–22.

背景和现状

- 模拟算法不符合物理规律（如：不满足不可压性质$\nabla \cdot \boldsymbol{v} = \mathbf{0}$）
- 模拟算法没有充分挖掘现代硬件的性能，实时性不好
- 渲染算法重建液体表面代价高昂
- 渲染算法表面着色模型不够真实

背景和现状

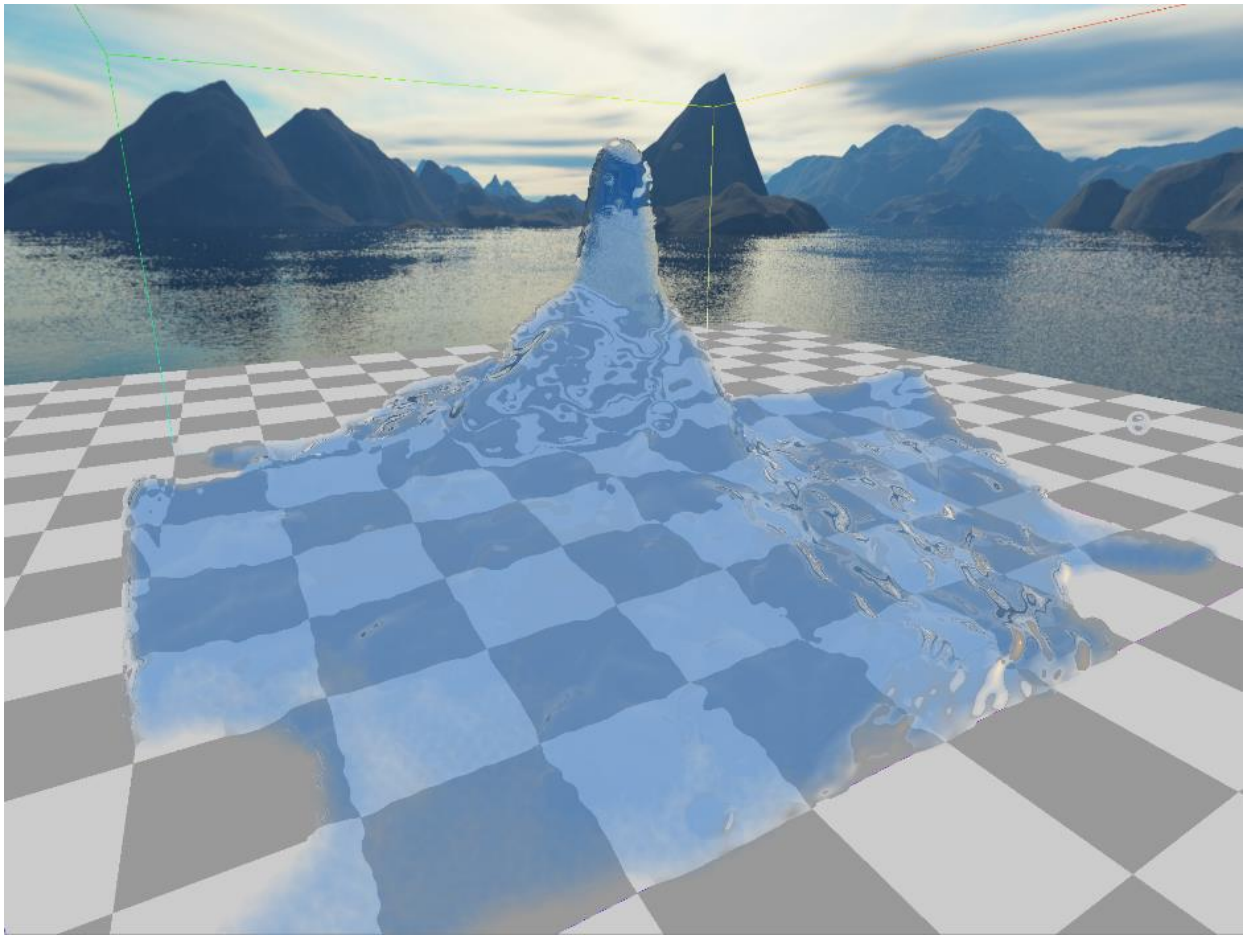
- 模拟算法不符合物理规律（如：不满足不可压性质$\nabla \cdot \boldsymbol{v} = \mathbf{0}$）
- 模拟算法没有充分挖掘现代硬件性能，实时性不好
- 渲染算法重建液体表面代价高昂
- 渲染算法表面着色模型不够真实

本文的贡献：具有真实感的实时流体模拟和渲染

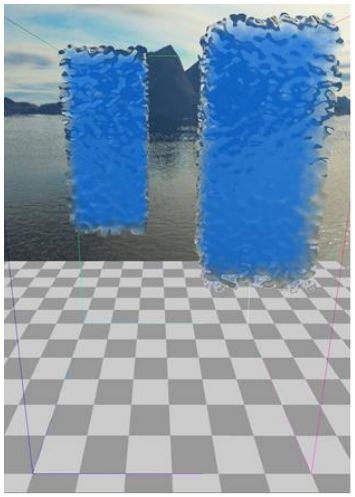
- 使用CUDA给出了基于位置的流体（Position Based Fluids）[7]的并行化实现
- 使用OpenGL给出了液体的屏幕空间渲染（Screen-space Rendering）[8]的并行化实现
- 提出使用双边滤波平滑液体表面的深度纹理
- 提出具有真实感的液体表面着色模型

[7] M. Macklin, el at., “Position based fluids,” *ACM Trans. Graph.*, vol. 32, no. 4, p. 1, Jul. 2013.
[8] W. J. van der Laan, el at., “Screen space fluid rendering with curvature flow,” I3D ’09, 2009, p. 91.

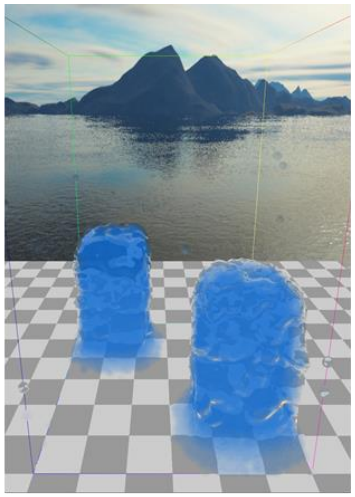
本文的结果



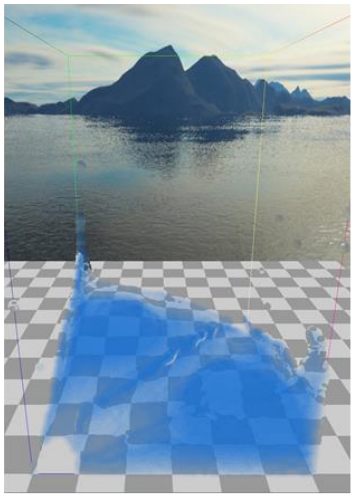
本文的结果



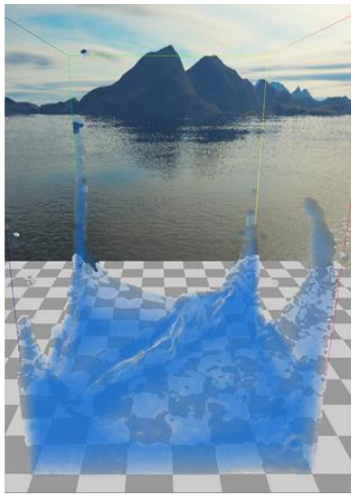
Frame 0



Frame 80



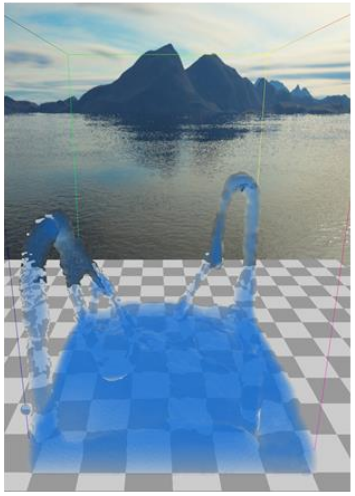
Frame 101



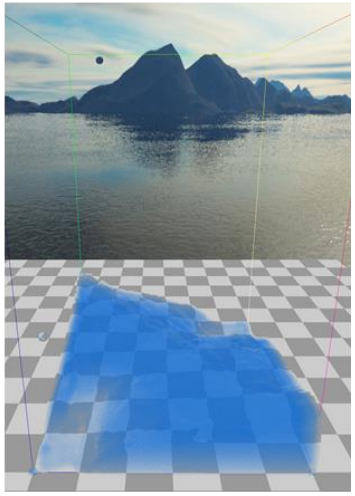
Frame 120

Frame 147

Frame 195



Frame 265



Frame 335

内 容

- 课题背景和工作概述
- 流体的并行化模拟
- 流体的并行化表面重建和渲染
- 演示

NS方程

Navier-Stokes方程组描述了流体运动的规律

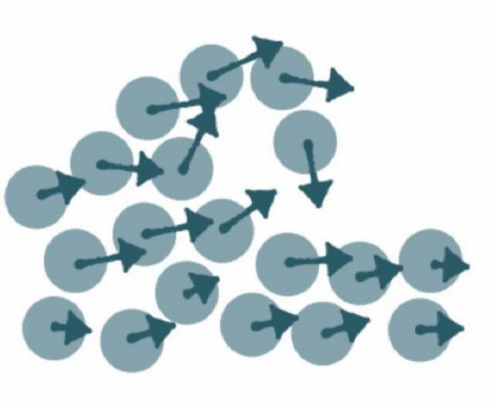
不可压条件 $\nabla \cdot \boldsymbol{v} = 0$

动量方程 $\rho \frac{D\boldsymbol{v_i}}{Dt} = \rho \boldsymbol{g} - \nabla p + \mu \nabla^2 \boldsymbol{v}$

粒子加速度　　压强力

传统方法

- 不考虑不可压性
- 使用经验公式计算压强
  （例如：粒子间距离）



Lagrangian

拉式离散
（粒子）

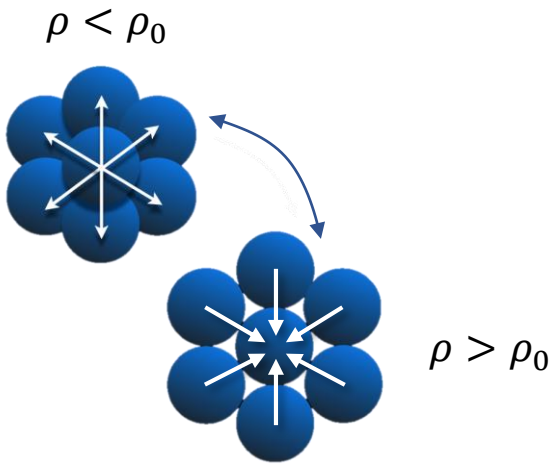
NS方程

Navier-Stokes方程组描述了流体运动的规律

不可压条件 $\nabla \cdot \boldsymbol{v} = 0$

动量方程 $\rho \frac{D\boldsymbol{v_i}}{Dt} = \rho \boldsymbol{g} - \nabla p + \mu \nabla^2 \boldsymbol{v}$

粒子加速度　　压强力



基于位置的流体[7]

- 不可压性等价于密度平衡条件

$$\nabla \cdot \boldsymbol{v} = 0 \Leftrightarrow \rho = \rho_0$$

[7] M. Macklin, el at., “Position based fluids,” *ACM Trans. Graph.*, vol. 32, no. 4, p. 1, Jul. 2013.

基于位置的流体

Navier-Stokes方程组描述了流体运动的规律

基于位置的流体[7]

- 不可压性等价于密度平衡条件

$$\nabla \cdot \boldsymbol{v} = 0 \Leftrightarrow \rho = \rho_0$$

- 密度平衡方程

$$C_i(\boldsymbol{p}_i, \cdots, \boldsymbol{p}_j) = \frac{\rho_i}{\rho_0} - 1 \neq 0$$

- 雅可比迭代

1. 对每个约束

$$\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{\Delta p}_{C_i}$$

2. 牛顿法，位置沿约束梯度下降

$$\boldsymbol{\Delta p} = \boldsymbol{\lambda} \nabla C$$

雅可比迭代特点：

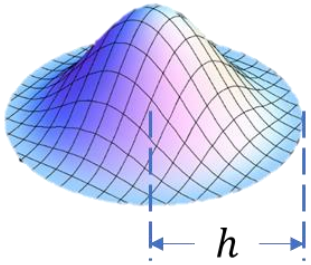
1. 一轮迭代仅依赖上轮迭代结果
2. 比高斯-赛德尔法慢2倍以上[10]
3. **易于并行！**

[7] M. Macklin, el at., “Position based fluids,” *ACM Trans. Graph.*, vol. 32, no. 4, p. 1, Jul. 2013.
[10] 同济大学计算数学教研室, *现代数值计算（第2版）*, Second Edi. Beijing: 人民邮电出版社, 2014.

基于位置的流体

Navier-Stokes方程组描述了流体运动的规律

核函数



核半径

基于位置的流体[7]

- 不可压性等价于密度平衡条件

$$\nabla \cdot \boldsymbol{v} = 0 \Leftrightarrow \rho = \rho_0$$

- 密度平衡方程

$$C_i(\boldsymbol{p}_i, \cdots, \boldsymbol{p}_j) = \frac{\rho_i}{\rho_0} - 1 \neq 0$$

- 雅可比迭代

1. 对每个约束

$$\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{\Delta p}_{C_i}$$

2. 牛顿法，位置沿约束梯度下降

$$\boldsymbol{\Delta p} = \boldsymbol{\lambda} \nabla C$$

步长计算公式

$$\lambda_i = -\frac{C_i(\boldsymbol{p}_1, \cdots . \boldsymbol{p}_n)}{\sum_{j \in \delta(\boldsymbol{p}_i)} \left\| \nabla_{\boldsymbol{p}_j} C_i \right\| + \epsilon}$$

位置更新公式

$$\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \frac{1}{\rho_0} \sum_{j \in \delta(\boldsymbol{p}_i)} (\lambda_i + \lambda_j) \nabla W(\boldsymbol{p}_i - \boldsymbol{p}_j)$$

邻居查找

[7] M. Macklin, el at., “Position based fluids,” *ACM Trans. Graph.*, vol. 32, no. 4, p. 1, Jul. 2013.

算法流程

CUDA核函数

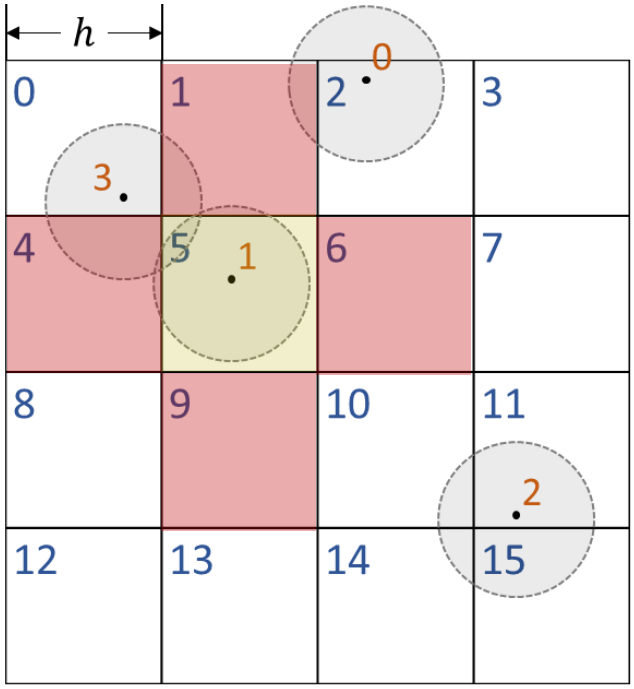
1. 预测新速度与位置
   1. $\boldsymbol{v}' \leftarrow \boldsymbol{v} + \Delta t \cdot \boldsymbol{g}$
   2. $\boldsymbol{p}' \leftarrow \boldsymbol{p} + \Delta t \cdot \boldsymbol{v}'$

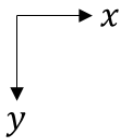
   ……advectKernel
2. 邻居查找预处理 ……computeGridRange
3. 密度修正迭代循环
   1. 计算位置修正系数λ_i ……computeLambda
   2. 计算$\Delta \boldsymbol{p}_i$
   3. 更新$\boldsymbol{p}_i' \leftarrow \boldsymbol{p}_i' + \Delta \boldsymbol{p}_i$ ……computePos
   4. 边界处理
4. 更新速度$\boldsymbol{v} \leftarrow \frac{\boldsymbol{p}^* - \boldsymbol{p}}{\Delta t}$ ……updateVelocity
5. 更新位置$\boldsymbol{p} \leftarrow \boldsymbol{p}^*$ ……updatePositon
6. 速度的粘着力修正

注：核函数以粒子作为并行级别

哈希网格邻居查找算法

将空间分割为边长为h的网格
查找粒子邻居时，查找周边网格内的粒子





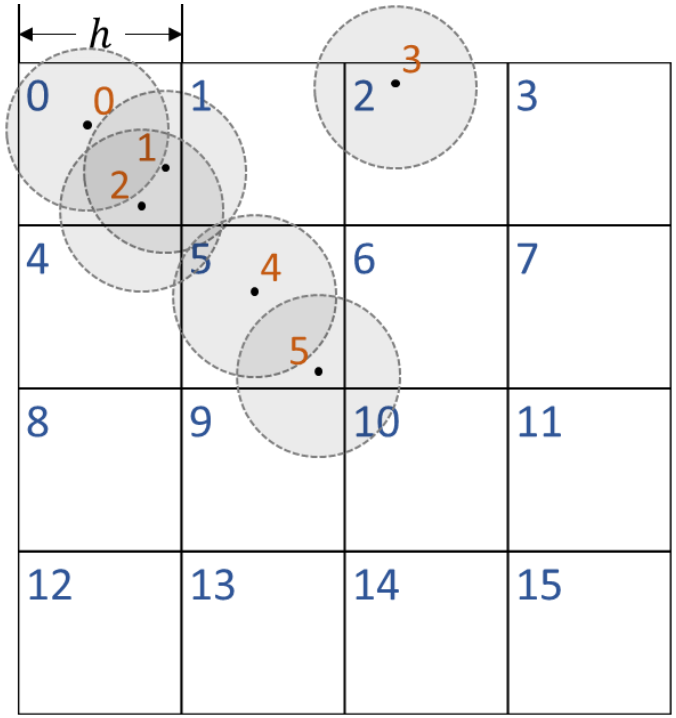

```
hash(x,y)=
  floor(x/h) +
  floor(y/h) * dim.x
```

Grid ID	Particle ID
0	3
2	0
5	1
11	2

哈希函数$hash(\boldsymbol{p})$
将空间位置映射到一个网格编号上

哈希网格邻居查找算法

将一个网格内的粒子重新连续编号
方法：将粒子按照网格编号排序，从头开始编号
查找网格内粒子：记录网格内首个、最后一个粒子的编号



New particle ID	Grid ID	gridStart	gridEnd
0		gridStart[0] = 0	
1	0	-	
2		-	gridEnd[0]=2
3	2	gridStart[2]=3	gridEnd[2]=3
4	5	gridStart[5]=4	
5		-	gridEnd[5]=5
…		…	

CUDA核函数
computeGridRange

串行：$O(n)$
并行：$O(1)$

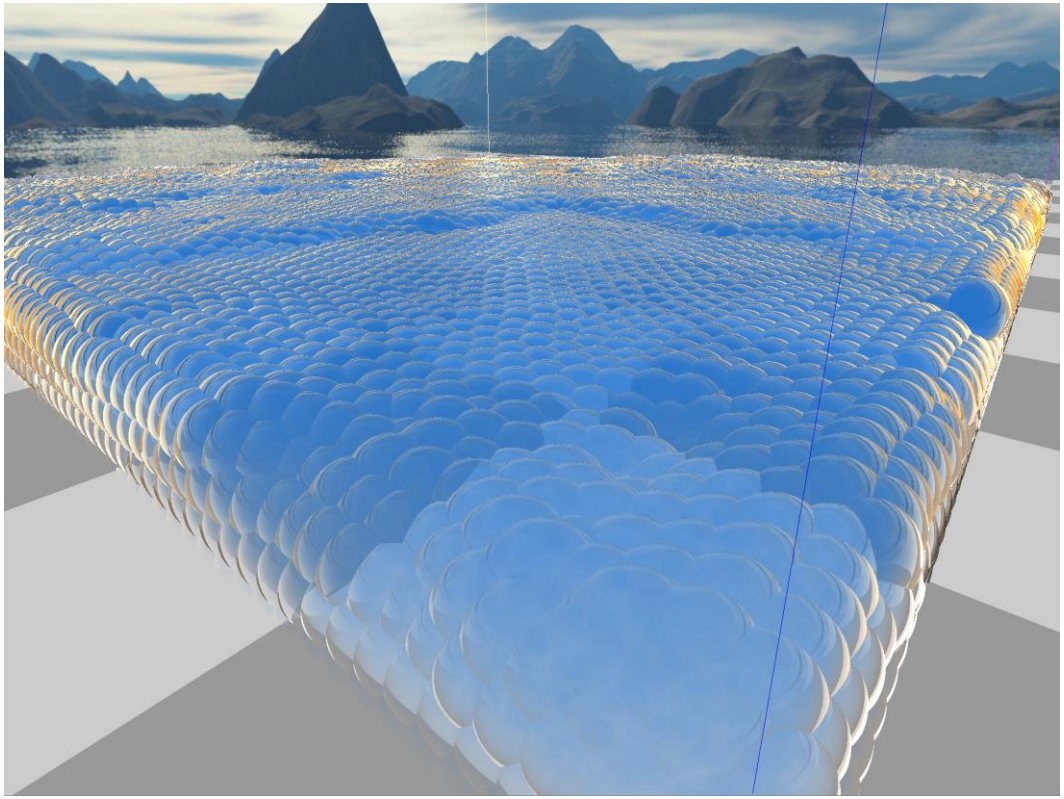
内 容

- 课题背景和工作概述
- 流体的并行化模拟
- 流体的并行化表面重建和渲染
- 演示

拉式液体的渲染

目标：渲染平滑的液体表面，减少“粒子感”

无平滑



拉式液体的渲染

目标：渲染平滑的液体表面，减少“粒子感”

传统方法：移动立方体（Marching Cubes）法[9]

- 从粒子位置生成密度分布
- 生成多边形等值曲面（Isosurface）
- 能够实现GPU上并行，但计算和编程复杂度较高

[9] William E. Lorensen, el at., ‘Marching Cubes: A high resolution 3D surface construction algorithm’. In: Computer Graphics’, Vol. 21, Nr. 4, July 1987

液体的屏幕空间渲染

目标：渲染平滑的液体表面，减少“粒子感”

屏幕空间渲染（Screen-space rendering）法[8]

- 将粒子深度、厚度信息渲染到屏幕空间纹理上
- 对深度纹理进行平滑操作
- 从深度纹理还原液体表面法向量
- 对液体表面进行着色

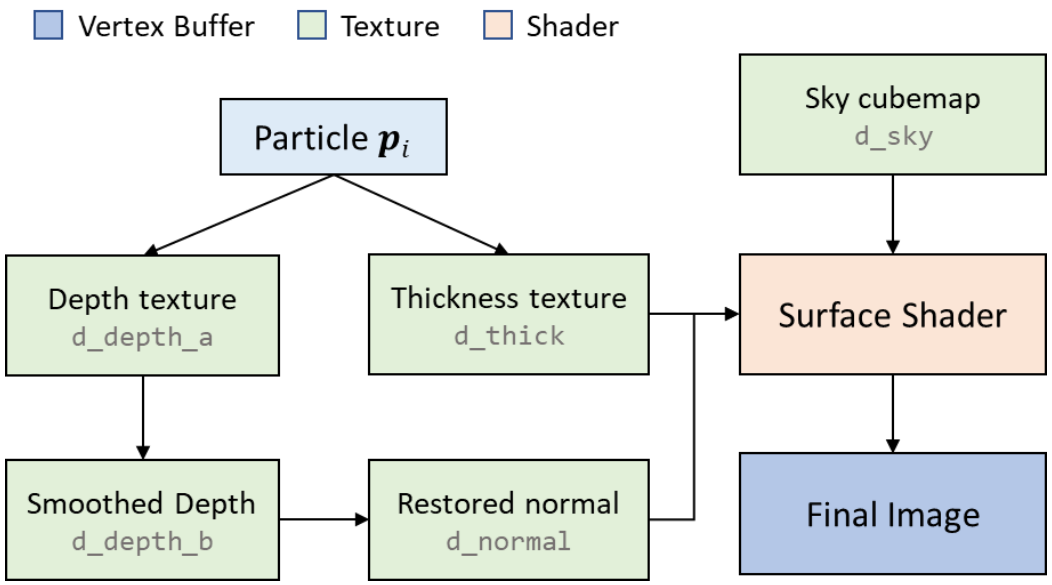
传统方法：移动立方体（Marching Cubes）法[9]

- 从粒子位置生成密度分布
- 生成多边形等值曲面（Isosurface）
- 能够实现GPU上并行，但计算和编程复杂度较高

[8] W. J. van der Laan, el at., “Screen space fluid rendering with curvature flow,” I3D ’09, 2009, p. 91.
[9] William E. Lorensen, el at., ‘Marching Cubes: A high resolution 3D surface construction algorithm’. In: Computer Graphics’, Vol. 21, Nr. 4, July 1987

液体的屏幕空间渲染

算法流程



注：深度平滑与法向量重建过程包含大量计算，使用片元着色器完成。
也可使用CUDA或OpenGL compute shader完成，但增加了不必要的复杂性。

深度纹理

深度纹理

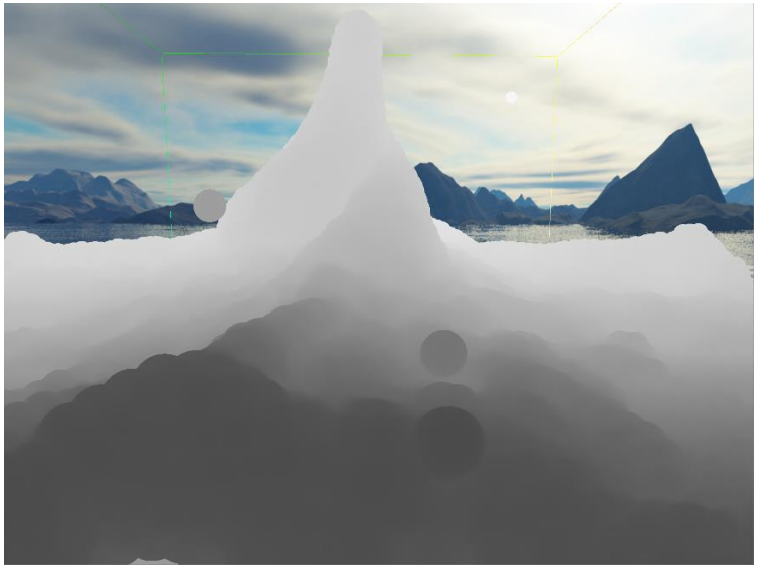
意义：每个像素堆叠的粒子个数

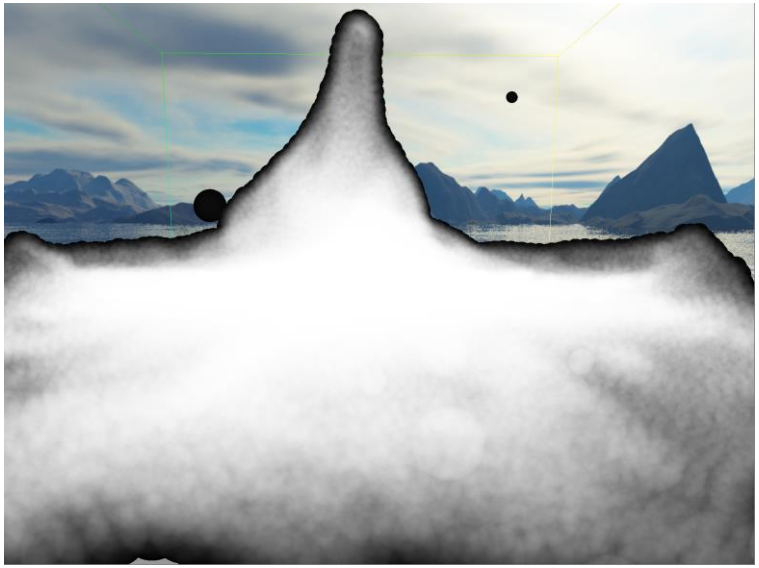
- 注1：开启深度测试
- 注2：粒子大小在世界坐标内不变，
  渲染大小gl_PointSize随距离摄像机远近调整。

厚度纹理

输出：每个像素在摄像机空间的深度（z坐标）

- 注1：关闭深度测试，开启纹理加法混合
- 注2：粒子大小在世界坐标内不变，
  渲染大小gl_PointSize随距离摄像机远近调整。





表面法向量重建

着色器 restore_normal

输入：深度纹理

输出：液体表面法向量

目标：法向量纹理（GL_RGB32F类型）

方法：

1. 深度纹理上任意一点(s,t)对应液体表面上一点$\boldsymbol{P}(s,t)=(x,y,z)$
2. z可从深度纹理中取出，(x,y)可利用投影变换的逆变换从(s,t)得到
3. 根据微分几何，表面法向量等于表面位置在两方向上偏导的叉积

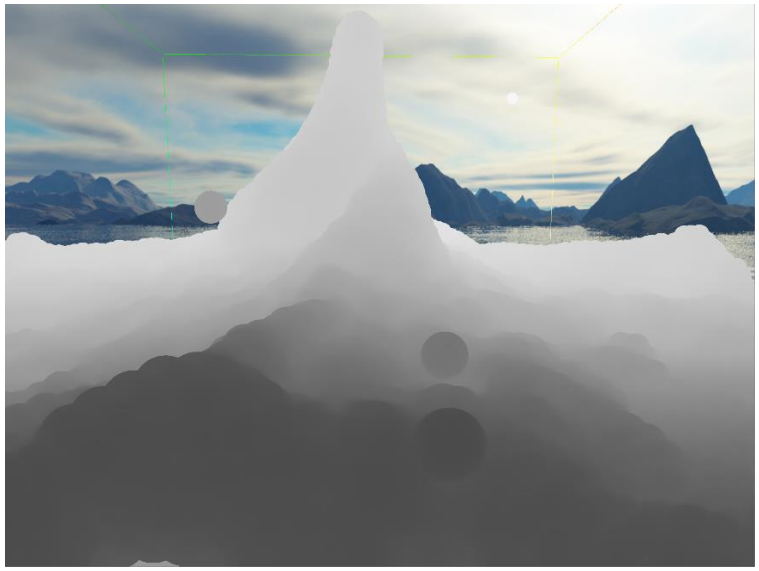
$$\boldsymbol{n}(s,t)=\frac{\partial \boldsymbol{P}}{\partial s}\times\frac{\partial \boldsymbol{P}}{\partial t}$$

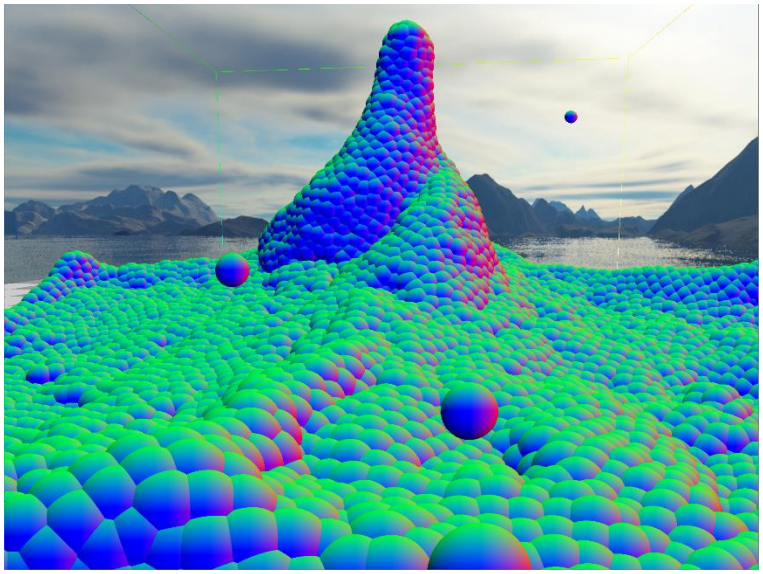
4. 利用一阶差分近似求偏导，Δs是一个像素对应的纹理坐标差值

$$\frac{\partial \boldsymbol{P}}{\partial s}\approx\frac{\boldsymbol{P}(s+\Delta s,t)}{\Delta s}$$

表面法向量重建

重建结果

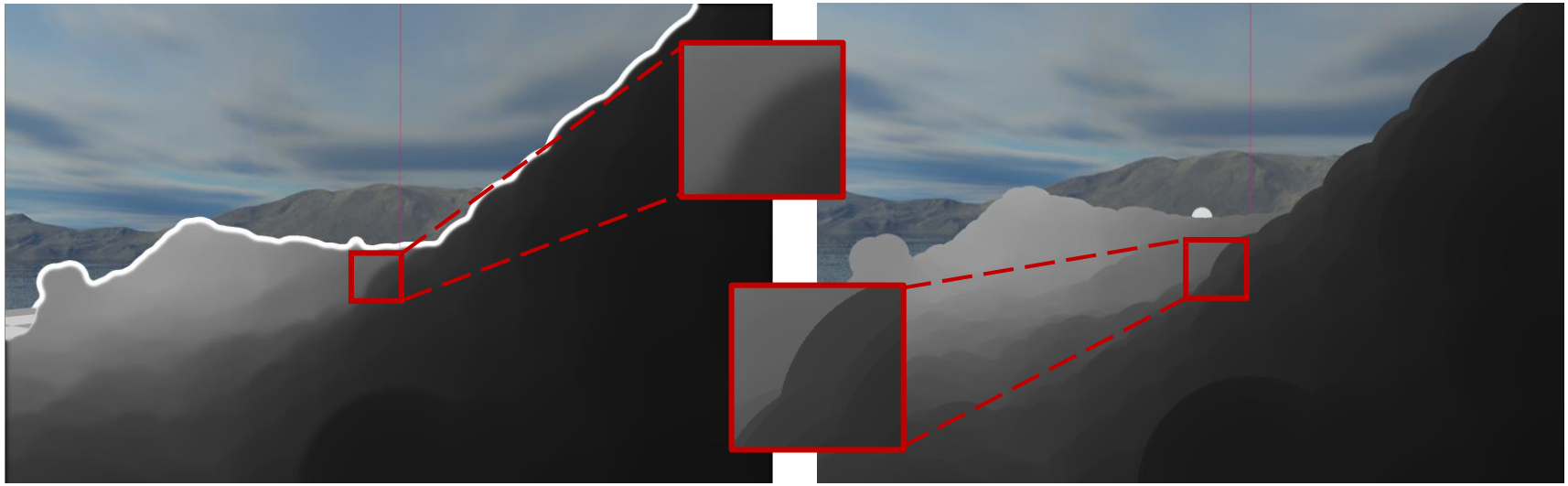




深度纹理平滑

为了表现平滑的液体表面效果，需要对深度纹理进行平滑
平滑操作：将邻近的像素加权平均

高斯模糊：错误的抹去了不连续的液体边缘！（应保留不连续的深度）



深度纹理平滑

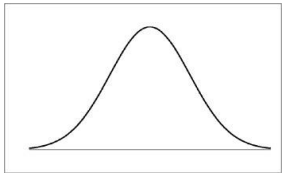
为了表现平滑的液体表面效果，需要对深度纹理进行平滑
平滑操作：将邻近的像素加权平均

双边滤波：同时考虑**位置差权重**和**深度差权重**

滤波后深度 $$I^{\text{filter}}(x) = \frac{1}{W_p}\sum_{y\in\Omega} \underbrace{I(y)}_{\text{深度}} \cdot \underbrace{f_r(\|I(y)-I(x)\|)}_{\text{深度差权重}} \cdot \underbrace{g_s(\|x-y\|)}_{\text{位置差权重}}$$

归一化因子 $$W_p = \sum_{y\in\Omega} f_r(\|I(y)-I(x)\|)g_s(\|x-y\|)$$

滤波核函数 $$f_r(d) = e^{-\sigma_r d}, \quad g_s(d) = e^{-\sigma_s d}$$



深度差越大，距离越远，权重越小

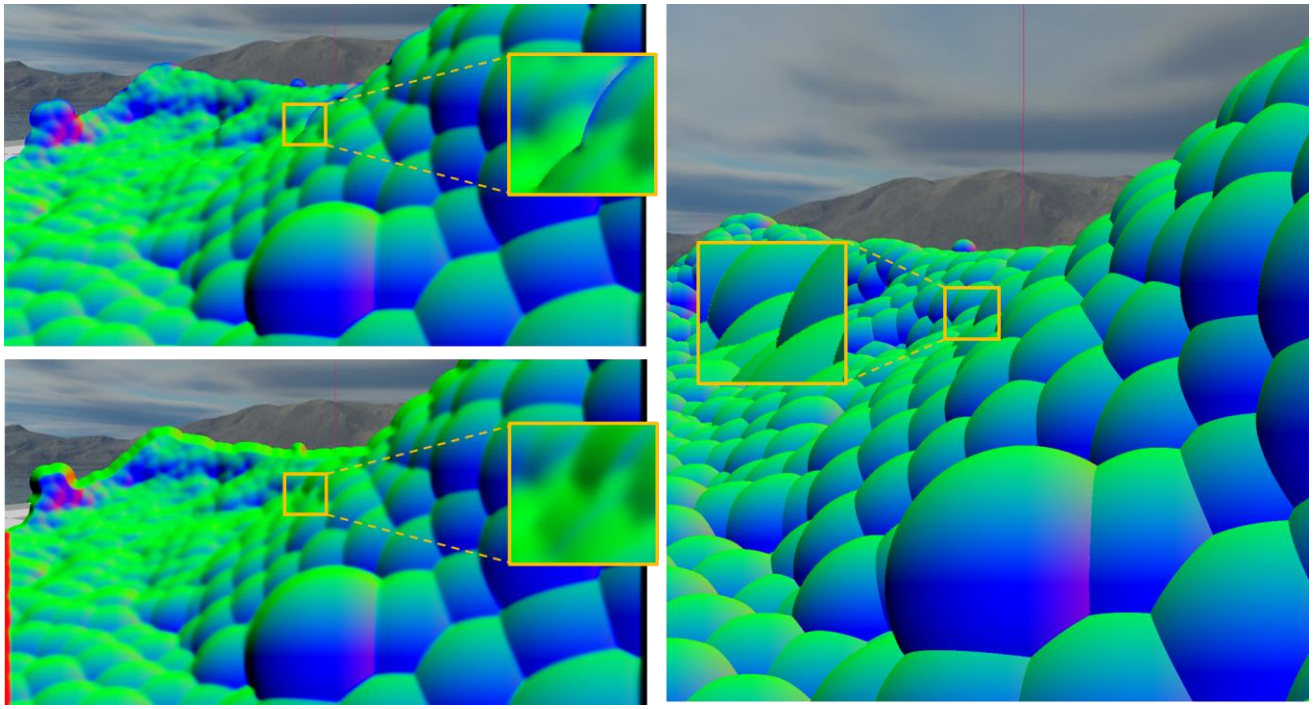
深度纹理平滑

平滑方法对比

原法线图

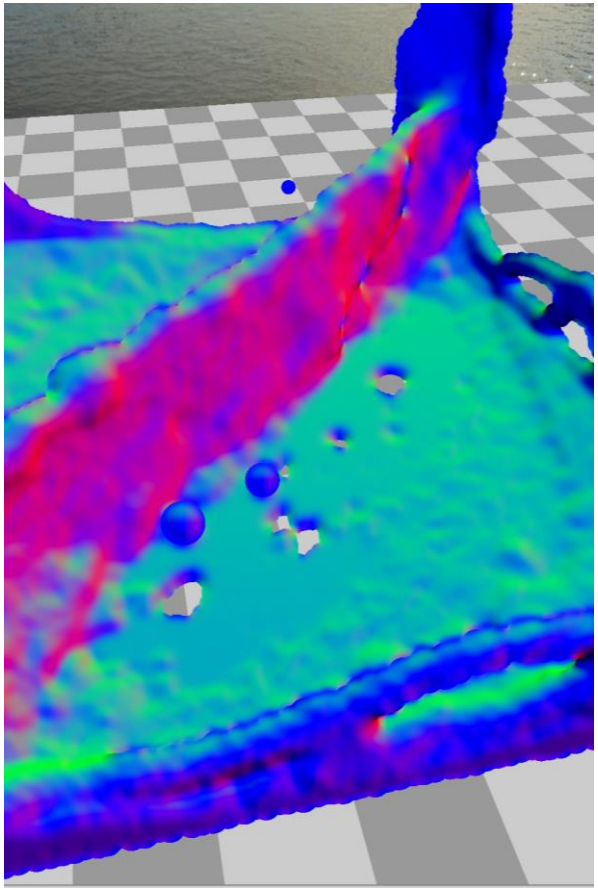
双边滤波

高斯模糊

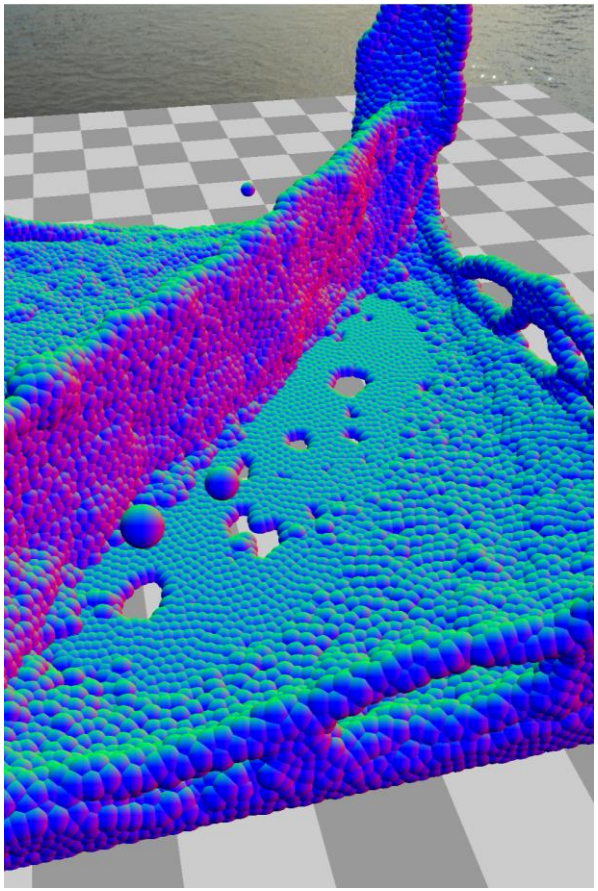


深度纹理平滑

平滑后



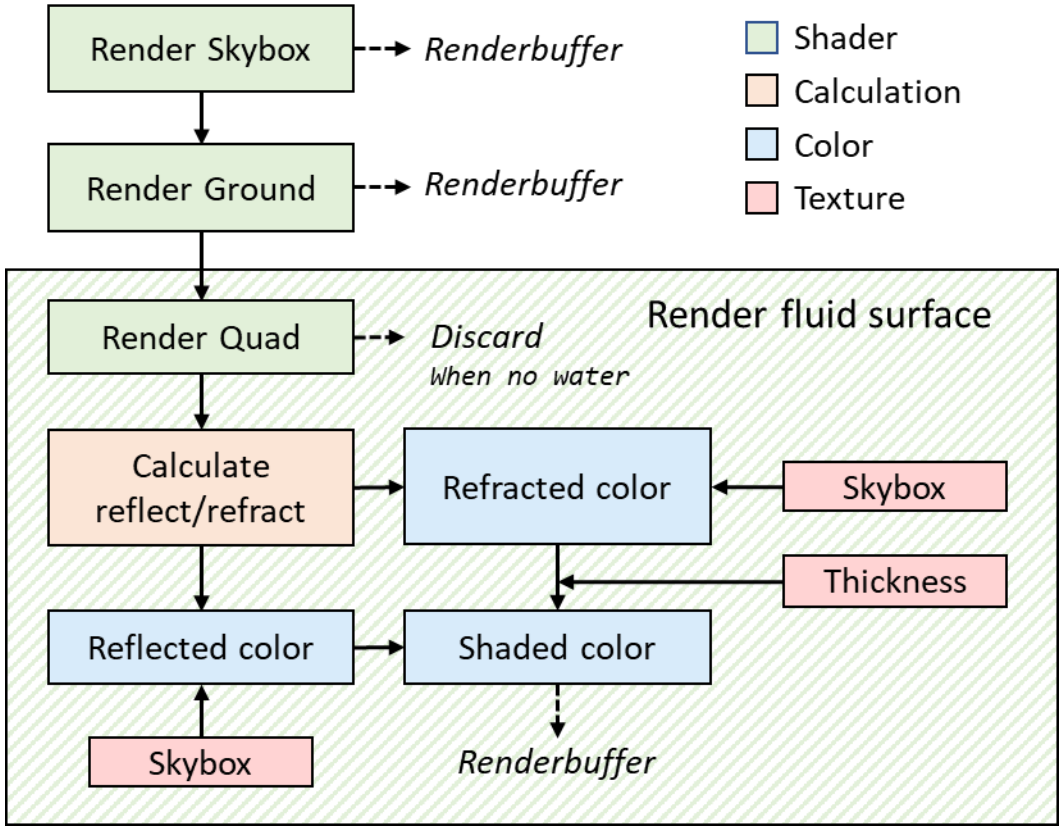
平滑前



表面着色

得到平滑的液体表面法向量后，对其进行着色

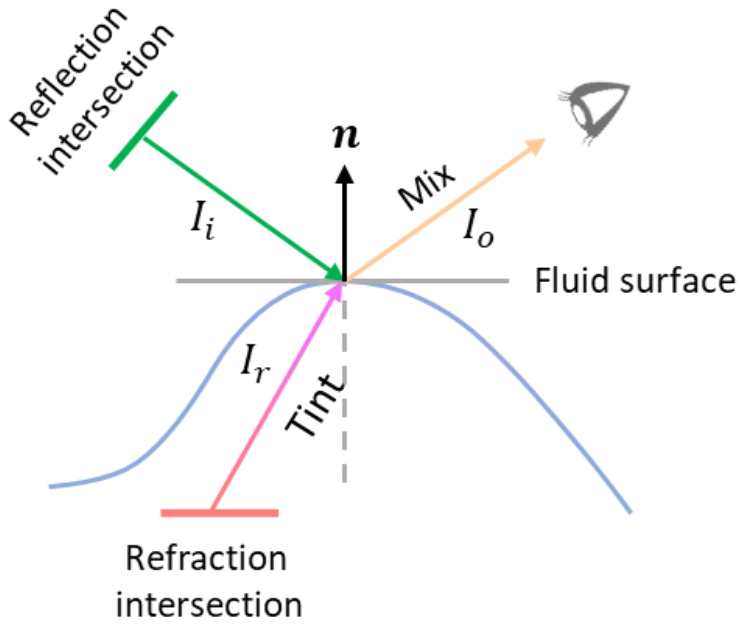
整个场景的渲染步骤如下所示



液体表面着色

液体表面的着色由折射光+反射光两部分混合而成

折射/反射的比例由菲涅尔方程给出

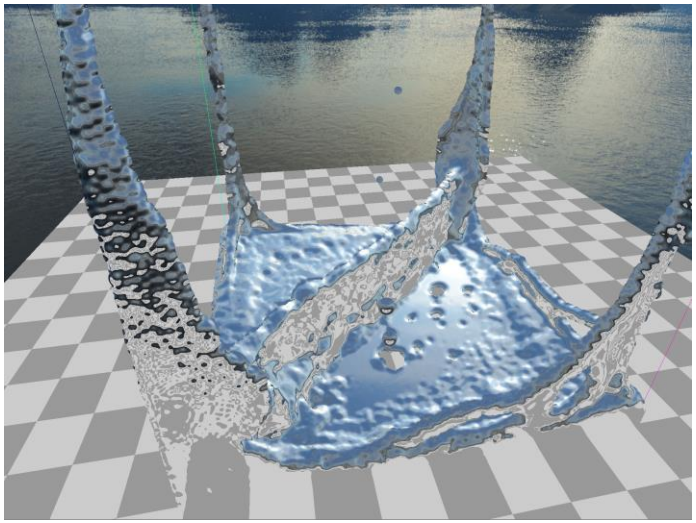


液体表面着色

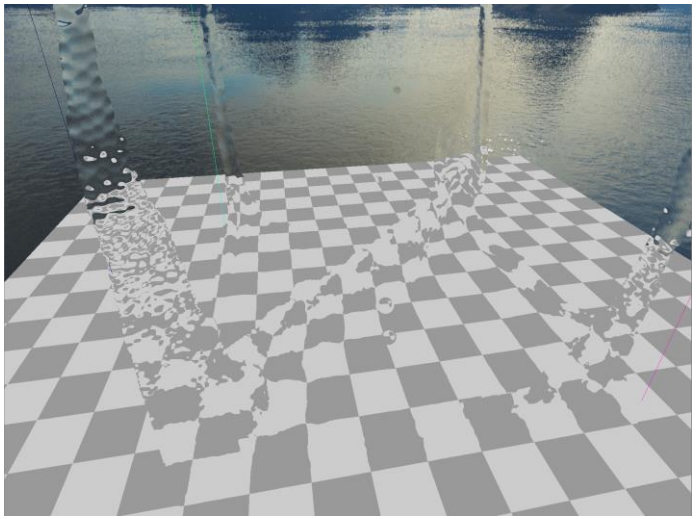
液体表面的着色由折射光+反射光两部分混合而成
折射/反射的比例由菲涅尔方程给出

光线追踪得到反射和光线的颜色

反射颜色



折射颜色



液体表面着色

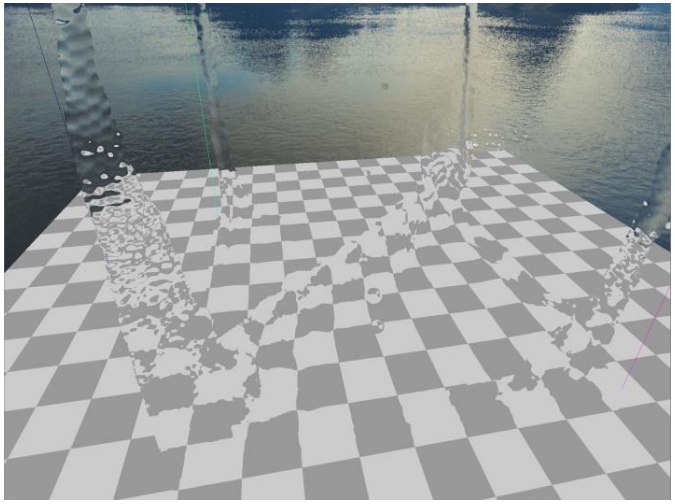
液体表面的着色由折射光+反射光两部分混合而成
折射/反射的比例由菲涅尔方程给出

考虑厚度：Beer–Lambert定律
液体透光率随着厚度增加指数减小

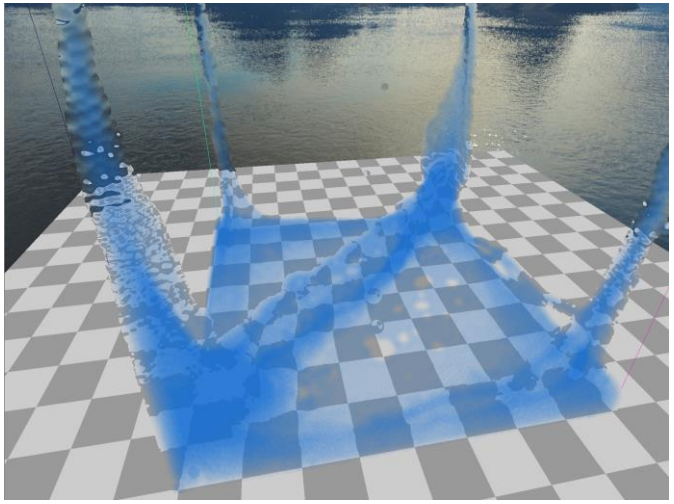
透光率 $A = \max\left(e^{-0.5\,T}, 0.2\right)$
修正后折射颜色 $I_r' = AI_r + (1 - A)I_f$

T厚度
I_f液体固有颜色

折射颜色



厚度染色



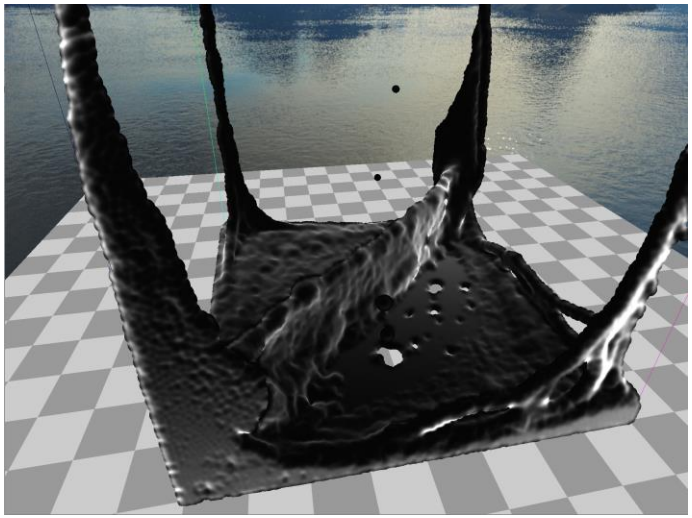
液体表面着色

液体表面的着色由折射光+反射光两部分混合而成
折射/反射的比例由菲涅尔方程给出

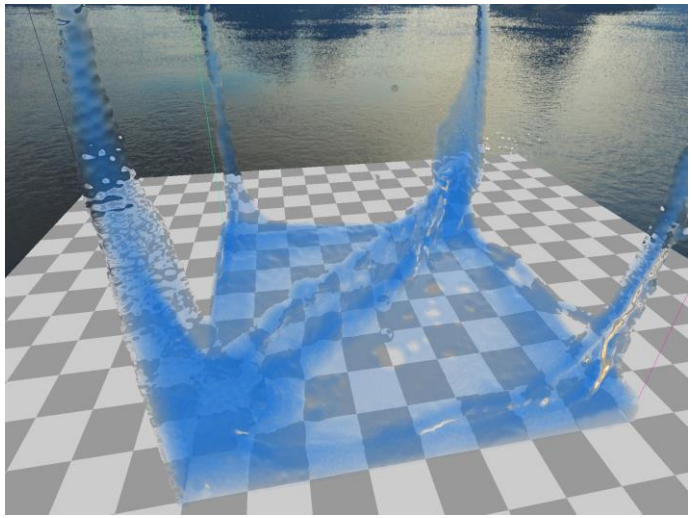
为了计算简便，使用菲涅尔定律的近似，Schlick公式

反射率 $R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$

反射率



最终着色



内 容

- 课题背景和工作概述
- 流体的并行化模拟
- 流体的并行化表面重建和渲染
- 演示

谢 谢

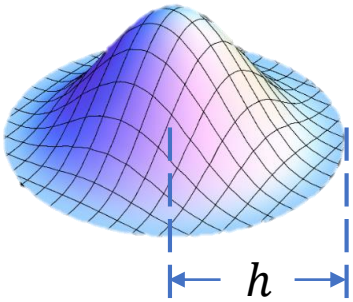
附加内容

平滑粒子动力学

平滑粒子动力学（Smoothed Particle Hydrodynamics, SPH）

将流体粒子表示为空间中一个径向对称的分布。即粒子的“势力范围”。

核函数



核半径

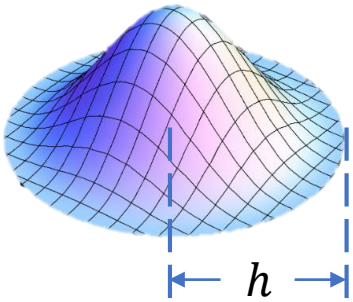
$$W_{poly6}(\boldsymbol{r},h)=\frac{315}{64\pi h^9}\begin{cases}\left(h^2-r^2\right)^3, & 0\le r\le h\\ 0, & otherwise\end{cases}$$

平滑粒子动力学

平滑粒子动力学（Smoothed Particle Hydrodynamics, SPH）

将流体粒子表示为空间中一个径向对称的分布。即粒子的“势力范围”。

核函数



核半径

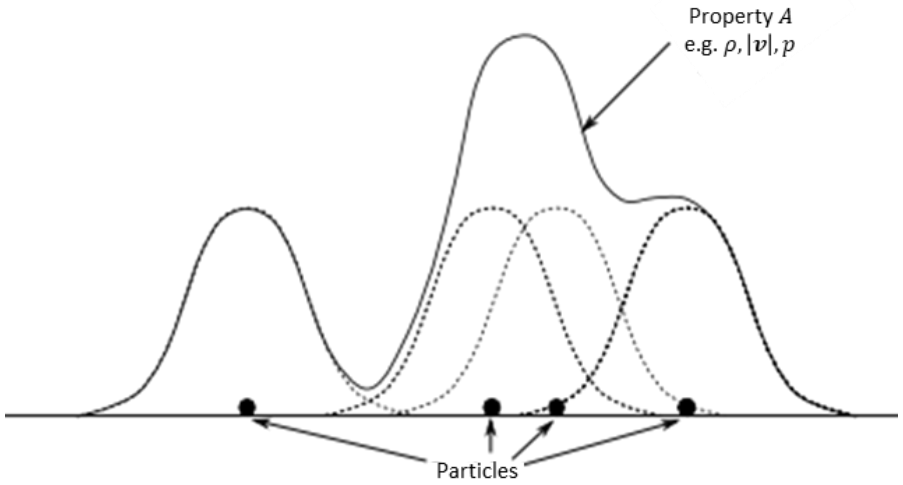
$$W_{poly6}(\boldsymbol{r},h)=\frac{315}{64\pi h^9}\begin{cases}\left(h^2-r^2\right)^3, & 0\le r\le h\\ 0, & otherwise\end{cases}$$

平滑粒子动力学

在粒子上记录的物理量

可估计空间中任意一点的物理量

可求物理量的梯度*



虚线：粒子上的物理量；实线：空间中的物理量

液体密度的估计

$$\rho(\boldsymbol{r}_i) = \sum_{i=1}^{N} m_i W(\boldsymbol{r}_{ij})$$

液体速度的估计

$$\boldsymbol{v}(\boldsymbol{r}_i) = \sum_{j=1}^{N} \boldsymbol{v}_j W(\boldsymbol{r}_{ij})$$

密度的梯度

$$\nabla\rho(\boldsymbol{r}_i) = \sum_{i=1}^{N} m_i \nabla W(\boldsymbol{r}_{ij})$$

*求梯度时，Poly6核函数存在梯度消失的问题，
应使用Spiky核函数

Navier-Stokes方程组

Navier-Stokes方程组描述了流体运动的规律

不可压条件 $\nabla \cdot \boldsymbol{v} = 0$

动量方程

重力

$$\rho \frac{D\boldsymbol{v_i}}{Dt} = \rho \boldsymbol{g} - \nabla p + \mu \nabla^2 \boldsymbol{v}$$

粒子加速度　压强力　粘性力

传统模拟方法无不可压条件
使用一个经验公式计算压强力

基于位置的流体

Macklin等人[7]注意到不可压条件的等效条件是密度平衡条件

$$\nabla \cdot \boldsymbol{v} = 0 \Leftrightarrow \rho = \rho_0$$

对每个粒子建立密度平衡约束

$$C_i(\boldsymbol{p}_i, \cdots, \boldsymbol{p}_j) = \frac{\rho_i}{\rho_0} - 1 = 0$$

牛顿法更新粒子位置，令$\Delta \boldsymbol{p} = \boldsymbol{\lambda} \nabla \mathrm{C}$延$C_i$的梯度下降

$$\begin{aligned} C(\boldsymbol{p} + \Delta \boldsymbol{p}) &\approx C(\boldsymbol{p}) + \nabla C^{\mathrm{T}} \Delta \boldsymbol{p} \\ &= C(p) + \nabla C^{\mathrm{T}} \nabla C \boldsymbol{\lambda} = 0 \end{aligned}$$

[7] M. Macklin, el at., “Position based fluids,” *ACM Trans. Graph.*, vol. 32, no. 4, p. 1, Jul. 2013.

基于位置的流体

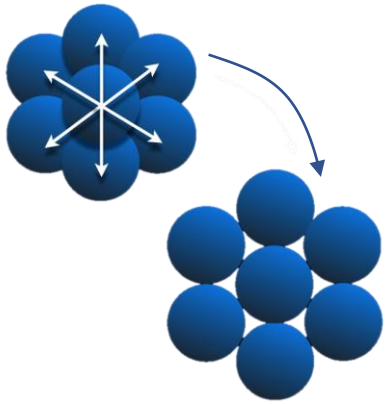
更新公式

$$\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \frac{1}{\rho_0}\sum_{j\in\delta(\boldsymbol{p}_i)}(\lambda_i+\lambda_j)\nabla W(\boldsymbol{p}_i-\boldsymbol{p}_j)$$

$$\lambda_i = -\frac{C_i(\boldsymbol{p}_1,\cdots.\boldsymbol{p}_n)}{\sum_{j\in\delta(\boldsymbol{p}_i)}\left\|\nabla_{\boldsymbol{p}_j}C_i\right\| + \epsilon}$$

稳定性修正

邻居查找

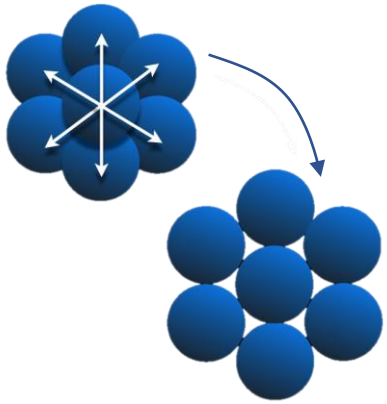


基于位置的流体

更新公式

$$\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \frac{1}{\rho_0}\sum_{j\in\delta(\boldsymbol{p}_i)}(\lambda_i+\lambda_j)\nabla W(\boldsymbol{p}_i-\boldsymbol{p}_j)$$

$$\lambda_i = -\frac{C_i(\boldsymbol{p}_1,\cdots.\boldsymbol{p}_n)}{\sum_{j\in\delta(\boldsymbol{p}_i)}\left\|\nabla_{\boldsymbol{p}_j}C_i\right\|+\epsilon}$$



更新方法

雅可比迭代：独立对每个约束进行求解

并行实现：使用两个CUDA核函数，首先计算λ_i，然后更新$\boldsymbol{p}_i$。

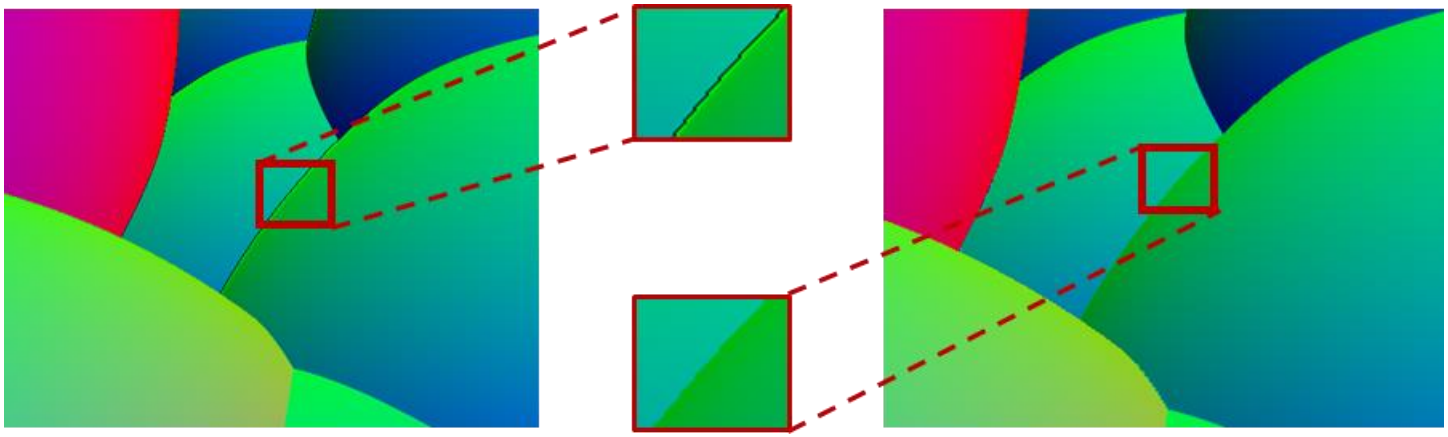
表面法向量重建

4. 利用一阶差分近似求偏导。不连续处失效！

$$\frac{\partial \boldsymbol{P}}{\partial s} \approx \frac{\boldsymbol{P}(s+\Delta s, t)}{\Delta s}$$

5. 选择连续一侧的偏导

$$\frac{\partial \boldsymbol{P}}{\partial s} \approx \min\left(\frac{\boldsymbol{P}(s+\Delta s, t)}{\Delta s}, \frac{\boldsymbol{P}(s-\Delta s, t)}{\Delta s}\right)$$



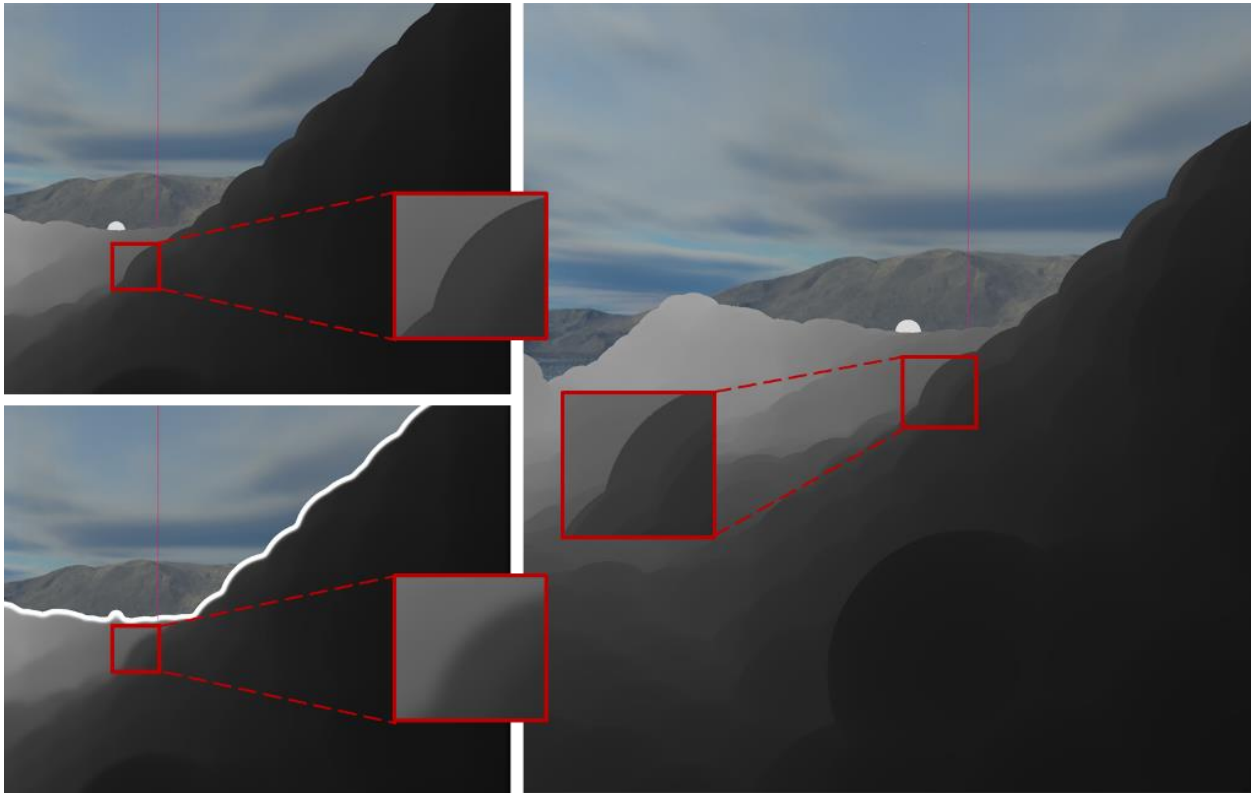
深度纹理平滑

平滑结果

原深度图

双边滤波

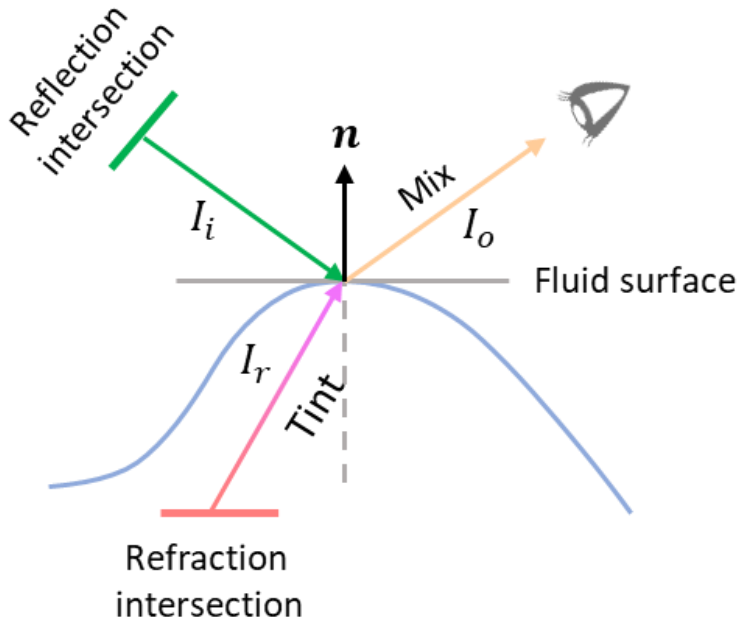
高斯模糊



液体表面着色

液体表面的着色由折射光+反射光两部分混合而成

折射/反射的比例由菲涅尔方程给出



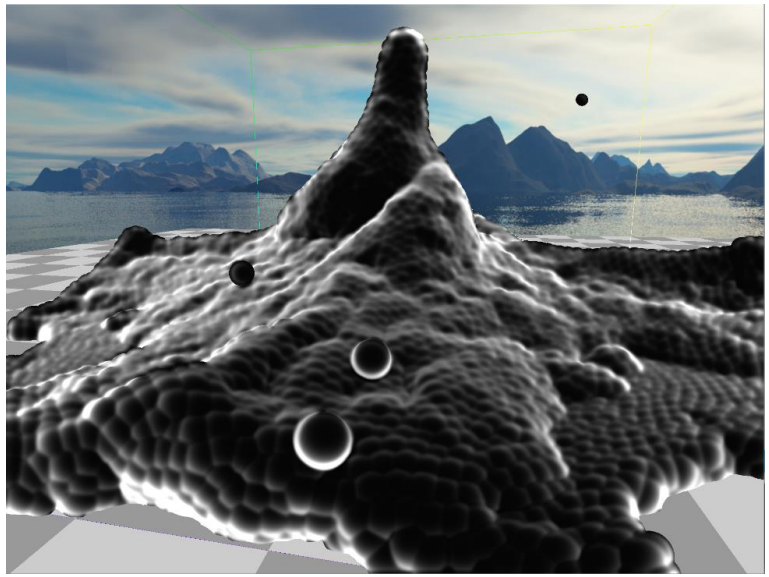
液体表面着色

液体表面的着色由折射光+反射光两部分混合而成
折射/反射的比例由菲涅尔方程给出

为了计算简便，使用菲涅尔定律的近似，Schlick公式

反射率 $R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$

$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$



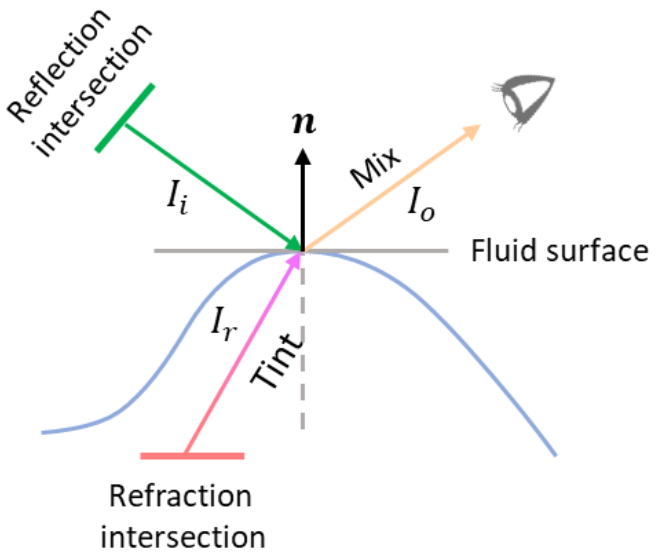
液体表面着色

液体表面的着色由折射光+反射光两部分混合而成

折射/反射的比例由菲涅尔方程给出

反射光线 $\hat{\boldsymbol{I}}_i = -\hat{\boldsymbol{I}}_o + 2\boldsymbol{n}$

使用光线追踪求解反射光线的颜色
忽略中间经过的液体
最终光线来源：天空盒 / 棋盘格地面



液体表面着色

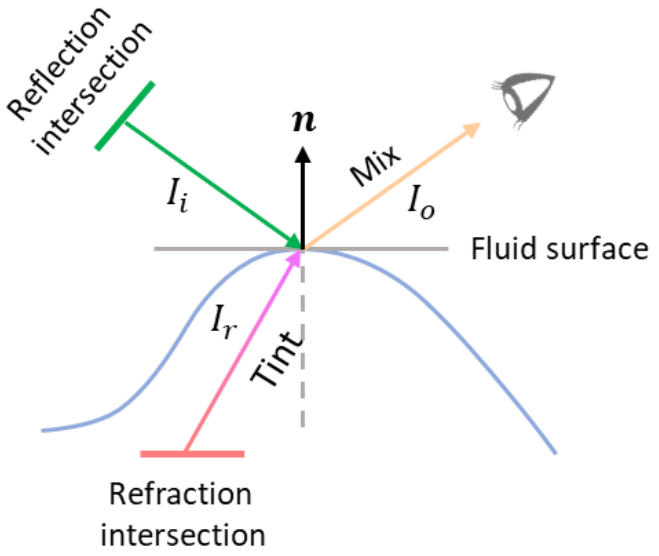
液体表面的着色由折射光+反射光两部分混合而成

折射/反射的比例由菲涅尔方程给出

折射光线 $\hat{\boldsymbol{l}}_r = -\hat{\boldsymbol{l}}_o - c \cdot \boldsymbol{n}$

HACK： 出射光线的反向，向法向做微小偏移

使用光线追踪求解折射光线的颜色
忽略中间经过的液体
最终光线来源：天空盒 / 棋盘格地面



液体表面着色

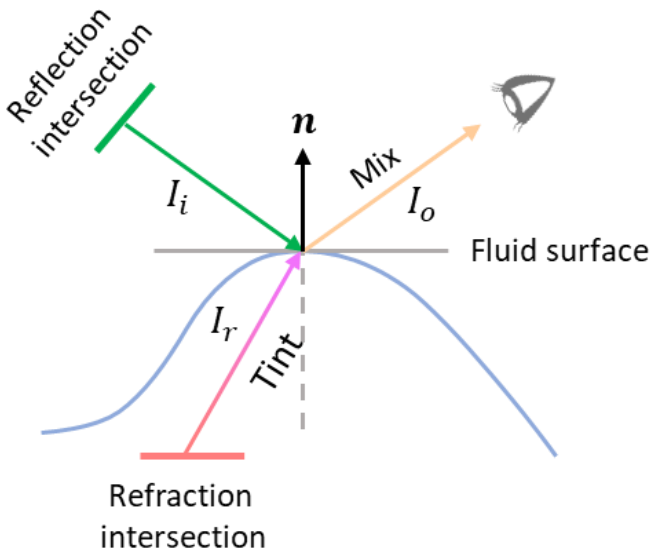
液体表面的着色由折射光+反射光两部分混合而成
折射/反射的比例由菲涅尔方程给出

考虑厚度：Beer–Lambert定律
液体透光率随着厚度增加指数减小

透光率 $A = \max\left(e^{-0.5\,T}, 0.2\right)$

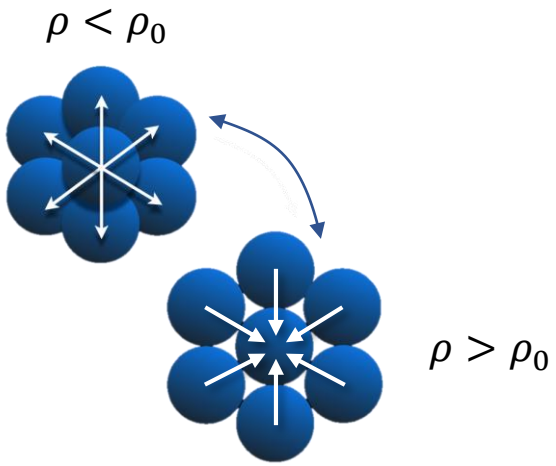
修正后折射颜色 $I_r' = AI_r + (1-A)I_f$

T厚度，I_f液体固有颜色



基于位置的流体

Navier-Stokes方程组描述了流体运动的规律



基于位置的流体[7]

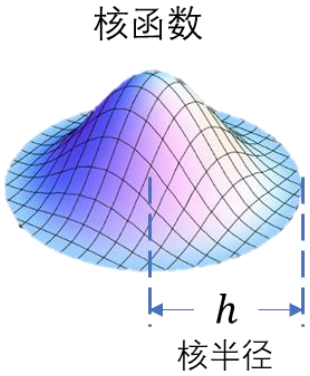
- 不可压性等价于密度平衡条件

$$\nabla \cdot \boldsymbol{v} = 0 \Leftrightarrow \rho = \rho_0$$

- 密度平衡方程

$$C_i = \frac{\rho_i}{\rho_0} - 1 = 0$$

- 密度计算：周围粒子的质量贡献



$$m_i = \int m_i W dV = 1$$

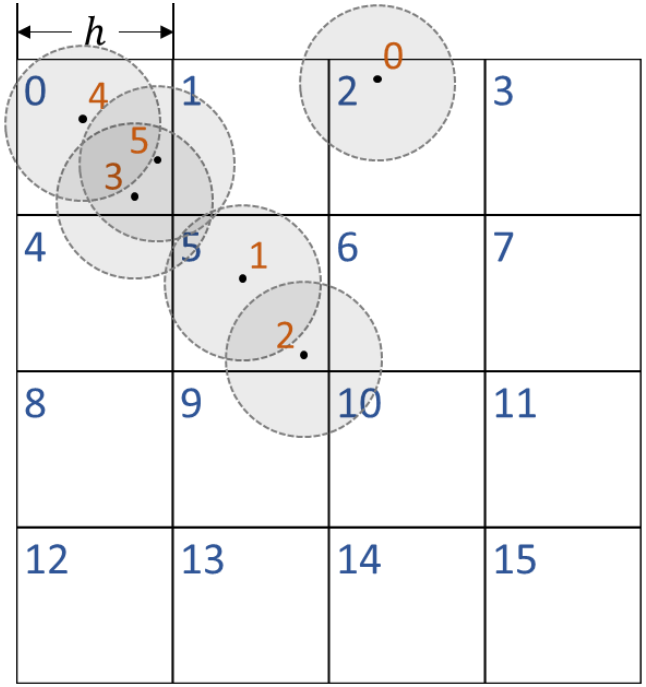
[7] M. Macklin, el at., “Position based fluids,” *ACM Trans. Graph.*, vol. 32, no. 4, p. 1, Jul. 2013.

哈希网格邻居查找算法

将一个网格内的粒子重新连续编号

方法：将粒子按照网格编号排序，从头开始编号

⇒ 一个网格内粒子编号连续



New particle ID	Old Particle ID	Grid ID
0	4	
1	5	0
2	3	
3	0	2
4	1	5
5	2	
…	…	