USTC-CG/2024 课程作业 实验报告

实验 9	**SPH Fluid**
马天开	PB21000030 (ID: 08)
Due: 2024.05.13	Submitted: 2024.06.21

原理概述 Theory

WCSPH (弱可压缩的 SPH 流体仿真方法)

Navier-Stokes 方程：

$$ \begin{aligned} \rho \frac{D \mathbf{v}}{D t} & =\rho g-\nabla p+\mu \nabla^2 \mathbf{v} \ \nabla \cdot \mathbf{v} & =0 \end{aligned} $$

使用核函数做离散化：

1. 忽略压力，更新 $\mathbf{v}$

   $$ \rho \frac{D \mathbf{v}}{D t} =\rho g+\mu \nabla^2 \mathbf{v} $$

2. 考虑压强，计算压力，更新 $\mathbf{v}$

   $$ \rho \frac{D \mathbf{v}}{D t} = -\nabla p $$

3. 根据最终的速度，更新粒子位置

其中：

$$ \rho_i = \sum_{j+i} \left(\frac{m_j}{\rho_j} \right) m_j W(\mathbf{x}_ i - \mathbf{x}_ j, h) = \sum_{j+i} m_j W_{ij} $$

$$ \nabla \cdot \mathbf{v}_ i = \sum_ j \frac{m_j}{\rho_ j} (\mathbf{v}_ j - \mathbf{v}_ i) \cdot \nabla W_{ij} $$

$$ \nabla^2 \mathbf{v}_ i=2(d+2) \sum_j \frac{m_j}{\rho_j} \frac{\mathbf{v}_ {i j} \cdot \mathbf{x}_ {i j}}{\left|\mathbf{x}_ {i j}\right|^2+0.01 h^2} \nabla W_{i, j} $$

$$ p_i =k_1 \left(\left(\frac{\rho_i}{\rho_0} \right)^{k_2} -1 \right) $$

$$ \nabla p*i = \rho_i \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W*{ij} $$

IISPH (隐式不可压缩的 SPH 流体仿真方法)

对每个元素进行迭代：

$$ \begin{aligned} \mathbf{p}_ i^{(k+1)} &=\mathbf{p} _ i^{(k)} + \frac{\omega}{a _ {i i}}\left(b_i- \sum_{j} a* {i j} \mathbf{p}* j^{(k)}\right), \ &= \mathbf{p}_ i^{(k)} + \frac{\omega}{a_{i i}}\left(b_i- (\mathbf{A}\mathbf{p}^{(k)})_i \right), \quad i=1,2, \ldots, n \end{aligned} $$

其中

$$ a*{i i}=- \sum_j m_j\left(\mathbf{d}* {i i}-\mathbf{d} _ {j i}\right) \nabla W_{ij} $$

$$ \mathbf{d}_ {ii} := \sum _ j \frac{m*j}{\rho_i^2} \nabla W*{ij} \ $$

$$ \mathbf{d}_ {ji} := \frac{m_i}{\rho_i^2} \nabla W_{ji} \ $$

$$ \mathbf{b}_i = \frac{\rho_0-\rho^*}{(\Delta t)^2} $$

$$ (\mathbf{A}\mathbf{p}^{(k)})*i \approx \sum_j m_j (\mathbf{a}^p_i - \mathbf{a}^p_j) \cdot \nabla W*{ij} $$

功能实现 Features Implemented

WSPCH

```
void WCSPH::compute_density() {
#pragma omp parallel for
  {
    for (auto &p : ps_.particles()) {
      p->density_ = 0.0;
      double w_zero = W_zero(ps_.h());
      p->density_ += ps_.mass() * w_zero;
      for (auto &q : p->neighbors()) {
        double w_ij = W(p->x() - q->x(), ps_.h());
        p->density_ += ps_.mass() * w_ij;
      }
      double rho_0 = ps_.density0();
      p->pressure_ = stiffness_ * (pow(p->density_ / rho_0, exponent_) -
1.0);
    }
  }
}
```

IISPH

```
void IISPH::predict_advection() {
#pragma omp parallel for
  {

    for (auto &p : ps_.particles()) {
      p->density_ = 0.0;
      double w_zero = W_zero(ps_.h());
      p->density_ += ps_.mass() * w_zero;
      dii_[p->idx_] = Vector3d::Zero();
      for (auto &q : p->neighbors()) {
        double w_ij = W(p->x() - q->x(), ps_.h());
        p->density_ += ps_.mass() * w_ij;

        Vector3d grad = grad_W(p->x() - q->x(), ps_.h());
```

```
        dii_[p->idx_] += ps_.mass() * grad;
      }
      dii_[p->idx_] = dii_[p->idx_] / pow(p->density_, 2);
    }

    compute_non_pressure_acceleration();
    for (auto &p : ps_.particles()) {
      p->vel_ += dt_ * p->acceleration_;
    }

    for (auto &p : ps_.particles()) {
      aii_(p->idx_) = 0.0;
      for (auto &q : p->neighbors()) {
        Vector3d grad = grad_W(p->x() - q->x(), ps_.h());
        Vector3d d_ji = ps_.mass() / pow(p->density_, 2) * (-grad);
        aii_(p->idx_) += -ps_.mass() * (dii_[p->idx_] - d_ji).dot(grad);

        p->density_ += -dt_ * ps_.mass() * (q->vel_ - p->vel_).dot(grad);
      }
      p->pressure_ = 0.5 * last_pressure_(p->idx_);
    }
  }
}

double IISPH::pressure_solve_iteration() {
#pragma omp parallel for
  {
    double rho_0 = ps_.density0();
    compute_pressure_gradient_acceleration();
    for (auto &p : ps_.particles()) {
      double b_i = (rho_0 - p->density_) / pow(dt_, 2);
      Api_[p->idx_] = 0.0;
      for (auto &q : p->neighbors()) {
        Vector3d grad = grad_W(p->x() - q->x(), ps_.h());
        Api_[p->idx_] +=
            ps_.mass() *
            (p->acceleration_pressure_ - q-
>acceleration_pressure_).dot(grad);
      }
      if (aii_[p->idx_] != 0.0)
        p->pressure_ += omega_ / aii_[p->idx_] * (b_i - Api_[p->idx_]);
      else
        p->pressure_ = 0.0;
      p->pressure() = std::clamp(p->pressure(), 0.0, 8e4);
    }

    double average_density_error = 0.0;
    for (auto &p : ps_.particles()) {
      average_density_error +=
          p->density_ - rho_0 + pow(dt_, 2) * Api_[p->idx_];
    }
    average_density_error = average_density_error /
ps_.particles().size();
    average_density_error = fabs(average_density_error);
```

```
        return average_density_error;
    }
}
```