Physically-Based Simulation
Final Project Presentation
Waterwheel

Group 12

Ge Cao, Xiao Wu, Mingyang Song
Introduction:

Implemented Features:

- Coupling with Rigid Bodies
- Iterative SESPH
- Particles Visualization (OVITO)
- Surface Reconstruction (splashsurf)
- Particles importing & exporting
- Rendering (Blender)
- Dambreak Scenario
- Waterwheel Scenario

Advanced Features:

- Multithreaded program
- Rendering with GPU
viscosity = 0.02 without waterwheel
SPH Method:

- Coupling with Rigid Bodies
  Boundary Handling:
  Several Layers with Uniform Boundary Samples
- Incompressibility
  Iterative SESPH
SPH Pipeline:

Navier-Stokes equation: \( \rho \frac{Dv}{Dt} = -\nabla p + \mu \nabla^2 v + f_{ext} \)

Algorithm (basic pipeline):

- Update \( v_i \) by non-pressure force: \( v_i^* = v_i + \Delta t \left( \frac{\mu}{m_i} \sum_j \frac{m_j}{\rho_j} \frac{v_{ij}}{||r_{ij}||} \right) + \frac{1}{m_i} F_{ext} \)

- Determine pressure force \( F_i^p \) using state equation: \( p_i = k(\rho_i - \rho_0) \)

\[ F_i^p = \sum_j m_j \left( \frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla_i W_{ij} \]

- Update \( v_i \) by solving: \( v_i(t + \Delta t) = v_i^* - \frac{\Delta t}{m_i} F_i^p \)

- Update \( x_i \) by solving: \( x_i(t + \Delta t) = x_i(t) + \Delta t v_i(t + \Delta t) \)
Boundary Handling:

To compute $F_i^p$, boundaries are sampled with particles:

$$\rho_i = m_i \sum_{if} W_{iif} + m_i \sum_{ib} W_{iib}$$

Boundary neighbors contribute to the density

$$p_i = k \left( \frac{\rho_i}{\rho_0} - 1 \right)$$

Pressure at boundary samples: Mirroring

$$a_i^p = -m_i \sum_{if} \left( \frac{p_i}{\rho_i^2} + \frac{p_{if}}{\rho_{if}^2} \right) \nabla W_{iif} - m_i \sum_{ib} \left( \frac{p_i}{\rho_i^2} + \frac{p_{ib}}{\rho_{ib}^2} \right) \nabla W_{iib}$$

Mirroring of pressure and density from fluid to boundary

$$\Rightarrow a_i^p = -m_i \sum_{if} \left( \frac{p_i}{\rho_i^2} + \frac{p_{if}}{\rho_{if}^2} \right) \nabla W_{iif} - m_i \sum_{ib} \left( \frac{p_i}{\rho_i^2} + \frac{p_i}{\rho_i^2} \right) \nabla W_{iib}$$

Contributions from fluid neighbors

Contributions from boundary neighbors
Incompressibility: Iterative SESPH

for all particle i do
    find neighbors j
for all particle i do
    \( a_i^{\text{nonp}} = v \nabla^2 v_i + g \); \( v_i^* = v_i(t) + \Delta t a_i^{\text{nonp}} \)

repeat
    for all particle i do
        \( \rho_i^* = \sum_j m_j W_{ij} + \Delta t \sum_j m_j (v_i^* - v_j^*) \nabla W_{ij} \)
        \( p_i = k \left( \frac{\rho_i^*}{\rho_0} - 1 \right) \)
    for all particle i do
        \( v_i^* = v_i^* - \Delta t \left( \frac{1}{\rho_i^*} \nabla p_i \right) \)
until \( \rho_i^* - \rho_0 < \eta \) (or iteration > max iteration)
for all particle i do
    \( v_i(t + \Delta t) = v_i^* \); \( x_i(t + \Delta t) = x_i(t) + \Delta t v_i(t + \Delta t) \)
Liquid with different viscosity

No viscosity can introduce a huge instability

\[ \text{viscosity} = 0 \quad \text{viscosity} = 0.002 \quad \text{viscosity} = 0.02 \]
Rendering: rendered as particles

visualized in OVITO

rendered as spheres in Blender
Rendering: rendered with surface reconstruction

Surface reconstruction: splashsurf
Rendering: Blender

- reconstructed surface
- smaller kernel size
- higher surface threshold

- sharp drops & uneven surface
- reduce drops number
Performance:

- Multi-threading computing
- Using GPU-based ray-tracing engine

For each frame, it takes 4 seconds to compute particles, 50 seconds to render image using ray-tracing
Thanks for your attention!

Any questions?