Implementing Particle-based Viscoelastic Fluid Simulation

Ao Tang
University of Toronto
Toronto, Canada
aoao.tang@mail.utoronto.ca

ABSTRACT

This report covers the implementation of a fluid simulation using a particle-based method. The method has been introduced by Simon Clavet et al. [2] in SIGGRAPH 2005. In this method, Incompressibility and particle anti-clustering are enforced with a double density relaxation procedure. This procedure updates particle positions according to two opposing pressure terms. My implementation follows object-orientation architecture to achieve high code reusability, scalability, and efficiency. I provide both 2D and 3D visualization to prove the correctness of the algorithm.

KEYWORDS

computer graphics, particle-based fluid simulation

1 INTRODUCTION

Fluids are the most common material we can find in the world. However, the behaviour of fluids is complex, and researchers have been making efforts to understanding and representing the dynamic behaviour of fluids using mathematical models. Clavet et al. introduce a novel particle-based model to simulate fluids with viscosity and elasticity. This report aims to briefly explain the algorithm of the paper, and it documents one implementation of 2D particle-based fluid simulation using Java and graphical library Processing [4]. The implementation can be easily extended to 3D and integrated with Blender for 3D visualization.

2 BACKGROUND

There are two main categories of fluid simulation methods in terms of the viewpoint: the Lagrangian methods and the Eulerian methods. In the Lagrangian methods, fluids are treated as particle systems, where each point in the fluid is labelled as a separate particle with its position and velocity. Lagrangian methods are intuitive and understandable compared to Eulerian methods, and the conservation of mass of fluids is easy to achieve [1]. On the other hand, in the Eulerian representation, the behaviour of fluid is observed from fixed points in space. The benefits are it is easier to numerically approximate spatial derivatives on a fixed Eulerian mesh.

3 ALGORITHM OVERVIEW

In the particle system, each particle can be described by its velocity and position. There are two types of particles: fluid and rigid body. The algorithm calculates the forces act on each particle by its nearby particles and update its velocity and position. Therefore, the algorithm is frame-based: it uses the current frame information to calculate the next frame. The algorithm extends on the Smoothed Particle Hydrodynamics (SPH) approach [5], and the procedure to calculate the next frame information can be broken down into eight components:

(1) Apply Gravity
(2) Apply Viscosity
(3) Update Particle Position
(4) Adjust Springs
(5) Apply Spring Displacements
(6) Apply Double Density Relaxation
(7) Resolve Collisions
(8) Update Particle Velocity

Clavet's paper proposes a novel procedure to simulate incompressibility and particle anti-clustering of fluid, which is the (6) component. It computes two different particle densities, and it uses these two densities to calculate the force act on each particle. This approach results in smooth liquid surfaces.

In addition, this paper aims to simulate a viscoelastic fluid, and this behaviour is achieved by (2), (4), and (5) components. Essentially, the elasticity of the fluid is obtained by inserting springs between particles (introduced in (4)), and plasticity comes from modifying the rest lengths of springs (in (4)). Viscosity is simulated by exchanging radial impulses between nearby particles.

The fluid system can be integrated with the rigid-body system as well. The contact between fluid-body and body-body is resolved in the (7) component. Just like the fluid particle, rigid-body can be regarded as particles as well, and it has its own velocity and position. If any fluid particle happens to be inside a rigid-body, the algorithm would calculate the collision impulse and apply it to fluid and rigid-body. Body-body interaction can be handled in the same way.

The other components of the procedure are very intuitive and straightforward. (1) modifies the velocity by gravity acceleration. (3) uses particle's velocity to update the particle's position. (8) uses the particle's position to calculate its velocity.

4 IMPLEMENTATION

4.1 System Diagram

The 2D version is implemented using Java. As shown in Figure 1, the system consists of seven classes. The visualization of the simulation is displayed on the GUI interface through the Processing library. The GUI interface provides the following features:

- Display simulation results: Play/Pause the animation
- Tweak customize parameters: Gravity, viscosity, plasticity, elasticity
- Interact with the fluid system: Add/remove fluid/rigid bodies

These features allow the user to fully control the simulator to obtain interesting fluid behaviour.

Physics-based Animation, Dec 2020, University of Toronto

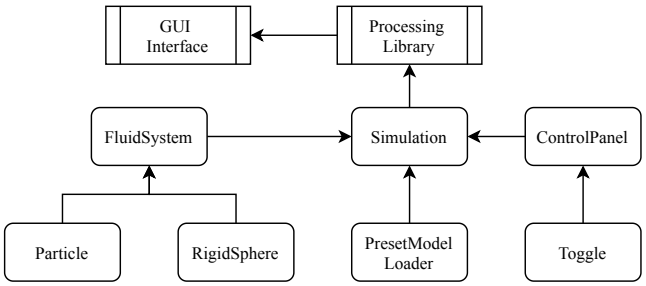

Figure 1: System Diagram of Fluid Simulation

`Simulation` controls the cycle of the animation. In the setup stage, it uses `PresetModelLoader` to load the pre-defined fluid-shape and rigid-bodies to the `FluidSystem`. During each frame, it invokes `FluidSystem` *simulationStep()* method to update the particle system information. It then draws the particles on the GUI. The components in the `ControlPannel` is drawn on the GUI as well.

`ControlPannel` defines text boxes, buttons, and `Pannel`. Text boxes show information about the fluid system such as the frame rate of the animation. Buttons provide functionalities such as allowing users to add/remove particles to the fluid system. `Panel` can adjust the parameters of the fluid system such as the gravity acceleration.

`FluidSystem` consists of two components: a collection of `Particle` and a collection of `RigidSphere`. Each `Particle` records the velocity and the position of a fluid particle. `RigidSphere` is the rigid-body with circle-shape in 2D.

4.2 Simulation Step

The *SimulationStep()* method follows the exact order defined in Section 3.

4.2.1 Apply Gravity. The gravity is applied to the velocity using the following formula:

$$\mathbf{v} = \mathbf{v} + \Delta t \mathbf{g} \tag{1}$$

4.2.2 Apply Viscosity. Viscosity is applied as radial pairwise impulses between neighboring particles. Impulses are then used to update the velocities of particles, and it can be calculated using the following formula inspired by SPH techniques [3]:

$$\mathbf{I} = \Delta t(1 - \frac{|\mathbf{r}_{ij}|}{h})(\sigma u + \beta u^2)\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \tag{2}$$

4.2.3 Update Particle Position. The position is updated using the particle's velocity:

$$\mathbf{x} = \mathbf{x} + \Delta t \mathbf{v} \tag{3}$$

4.2.4 Adjust Springs. The rest length of the springs between neighbouring particles is changeable to simulate plasticity. Whenever the actual deformation is larger than a threshold, the rest length of the springs will be changed using the following formula: where the threshold is proportional to the rest length of the spring as γL

$$\Delta L = \Delta t \alpha sign(r - L) max(0, |r - L| - \gamma L) \tag{4}$$

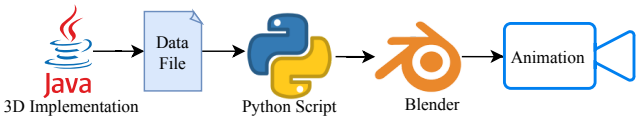

Figure 2: Procedure of Generating 3D Animation

4.2.5 Apply Spring Displacements. Unlike normal springs, the springs between neighbouring particles only affect positions of the particles but not their velocity:

$$\Delta \mathbf{x} = \Delta t^2 k_{spring}(1 - \frac{|\mathbf{r}_{ij}|}{h})(L_{ij} - |\mathbf{r}_{ij}|)\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \tag{5}$$

4.2.6 Apply Double Density Relaxation. The density for any particle is approximated by summing weighted contributions from its neighbours. The pressure is proportional to the difference between the current density ρ and a rest density ρ_0. The density relaxation displacement between particle pairs can be calculated using P using a linear kernel function. In addition to pressure, another terminology called near-pressure is used to constrain the corresponding force between particle pairs exclusively repulsive, and the result of double density relaxation is a coherent fluid with a smooth surface.

$$\begin{aligned} \rho &= \sum_{neighbors} (1 - \frac{|\mathbf{r}_{ij}|}{h})^2 \\ P &= k(\rho - \rho_0) \qquad\qquad P_{near} = k_{near} P \\ \mathbf{D} &= \Delta t^2 (P(1 - \frac{|\mathbf{r}_{ij}|}{h}) + P_{near}(1 - |\mathbf{r}_{ij}|/h)^2)\frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|} \end{aligned} \tag{6}$$

4.2.7 Resolve Collisions. The collision is simulated by calculating the impulse during the collision. The velocity difference between two colliding particles $\bar{\mathbf{v}}$ is orthogonal decomposed into $\bar{\mathbf{v}}_{normal}$ and $\bar{\mathbf{v}}_{tagent}$, and the impulse is calculated using the following formula with a controable parameter μ:

$$\mathbf{I} = m(\bar{\mathbf{v}}_{normal} - \mu \bar{\mathbf{v}}_{tagent}) \tag{7}$$

Collisions can be categorized into four types: fluid-with-fixed-body, fluid-with-movable-body, movable-body-with-non-movable-body, movable-body-with-movable-body. The impulse is always applied to movable bodies and fluid, but not non-movable-body. The walls are regarded as non-movable-bodies. The implementation does not support the last type of collision.

4.2.8 Update Particle Velocity. The velocity is updated based on the position difference:

$$\mathbf{v} = (\mathbf{x}_{new} - \mathbf{x}_{old})/\Delta t \tag{8}$$

4.3 3D Implementation

The processing library does not support the 3D GUI. Therefore, I use Blender for rendering the animation. I leverage the 2D-version to use 3D vectors for velocity and position, and write the position of each particle for each frame into a file. I use python integration to load the data into Blender and then render the animation. The procedure is shown in Figure 2.

Model	Description	# Particles	# Spheres
0	Water Dam	1008	0
1	Water with Movable Sphere	1008	1
2	Water with Fixed Spheres	1116	3
3	Water with Text Wall	1008	146
4	Seven Water Cubes	738	1
5	Fountain	N/A	0

Table 1: List of 2D Simulation Models

5 RESULTS AND PERFORMANCE

My 2D implementation was tested using six different models. Each model has approximately one thousand fluid particles, and some models have either movable or non-movable rigid bodies as well. The performance is quite satisfactory, as the frame rate is around 50-60 at the beginning of the simulation. However, the frame rate would drop as the simulation goes on due to the fact that fluid becomes "tighter" and there are more forces to compute, which slows down every cycle.

3D implementation is visually appealing. It is tested using five models, and the generated animation looks realistic given the fact that each model only consists of approximately one thousand particles. However, the python integration which imports data to Blender is very slow, and the importing time grows significantly when increasing the number of particles.

REFERENCES

[1] Robert Bridson. 2008. *Fluid Simulation.* A. K. Peters, Ltd., USA.
[2] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. 2005. Particle-Based Viscoelastic Fluid Simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Los Angeles, California) *(SCA '05)*. Association for Computing Machinery, New York, NY, USA, 219–228. https://doi.org/10.1145/1073368.1073400
[3] Mathieu Desbrun and Marie-Paule Gascuel. 1996. Smoothed Particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96*, Ronan Boulic and Gerard Hégron (Eds.). Springer Vienna, Vienna, 61–76.
[4] Ben Fry and Casey Reas. [n.d.]. https://processing.org/
[5] L. B. Lucy. 1977. A numerical approach to the testing of the fission hypothesis. 82 (Dec. 1977), 1013–1024. https://doi.org/10.1086/112164