Fluid Simulation

Xie He, Xiaoting Bi

December 11, 2015

1. Introduction

This is the report of our team project of Fluid Simulation.

We use a particle-based model to simulate the motion of fluid, such as water, based on the Navier-Stocks equations from fluid mechanics. The core algorithm is based on [1], which utilizes Smoothed Particles Hydrodynamics (SPH) [2], and Navier-Stocks equations to calculate the acceleration of each particle. After getting the acceleration of each particle, we use the Velocity-Verlet algorithm to update the velocity and the position of each particle.

2. Acceleration Calculation

2.1. SPH Method

To model continuous fluid with discrete particles, we need a method to reconstruct continuous fields from field quantities that are only defined at the locations of particles. SPH is exactly such a method. For any physical quantity A, the value of A at a specific location $\mathbf{r}$ can be found by

$$A_S(\mathbf{r}) = \sum_j A_j \frac{m_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h)$$

where j means the jth neighboring particle, $\mathbf{r}_j$ is the location of that particle, m_j, ρ_j and A_j are the mass, density, and the physical quantity A stored at that particle. The function $W(\mathbf{r}, h)$ is the smoothing kernel function with radius h, which gives each particle within radius h a weight before adding that particle's contribution of quantity A.

2.2. Density and Pressure

Using the SPH method, we can calculate the value of density at each particle, by

$$\rho_S(\mathbf{r}) = \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h)$$

And then we can find the pressure at each particle by the relationship between density and pressure.

In [1] the authors suggests using $p = k\rho$ or $p = k(\rho - \rho_0)$ to find the pressure from the density. They say that $p = k\rho$ is the ideal gas state equation and k is a gas constant. But we have doubts here because ideal gas state equation is for ideal gas, not for liquids like water. We think $p = k(\rho - \rho_0)$ is the right equation and the value of k should be derived based on the bulk modulus of water. However, after doing actual simulation, we found that neither k derived from bulk modulus of water or the ideal gas law is small enough to prevent the particles to get super large accelerations and quickly bounce everywhere in our cube container. We don't know why and it may be because the number of particles in our simulation is too small. We use a self-defined smaller value in our simulation instead.

2.3. Forces

Based on [1], we mainly consider three kinds of forces – force of pressure, viscosity and surface tension, using SPH method. Other than those three forces, the force of gravity is applied by directly modifying the acceleration, and the force given by the boundary of our cube container is applied by directly modifying the velocity and position of particles.

The force density of pressure gradient can be found by a slightly modified version of SPH. This equation resolves the problem of asymmetry of the resulting force of the original SPH. It is

$$\mathbf{f}_i^{\text{pressure}} = -\sum_j \frac{p_i + p_j}{2} \cdot \frac{m_j}{\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h)$$

The force of viscosity can be calculated in similar to the force of pressure. The equation is also modified to preserve the symmetry of force. It is

$$\mathbf{f}_i^{\text{viscosity}} = \mu \sum_j (\mathbf{v}_j - \mathbf{v}_i) \frac{m_j}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h)$$

The force of surface tension is calculated in a complex way in [1]. It uses an additional field quantity called color field, which is defined as 1 in all particle locations. It calculates the normal vector and the curvature using the color field at each surface particle, and then gets the force of surface tension whose direction is determined by surface normal and magnitude is determined by surface curvature.

However, this method produces strange results in our simulation. The particles self-organize into clusters with certain size and structure, which is not the way surface tension should behave. We use a simpler method, which models surface tension by its cause – attractions from neighboring particles. We calculate the force density of surface tension by

$$\mathbf{f}_i^{\text{surface}} = a \sum_j (\mathbf{r}_j - \mathbf{r}_i) \frac{m_j}{\rho_j} W(\mathbf{r}_i - \mathbf{r}_j, h)$$

After these force densities are calculated, we add them up to get the acceleration of the particle, with gravitational acceleration applied:

$$\mathbf{a}_i = \frac{\mathbf{f}_i}{\rho_i} + \mathbf{g}$$

3. Updating Velocity and Position

We use the Velocity-Verlet algorithm described in professor's lecture to update the velocity and position. Our algorithm is like this:

```
For each particle
        Half-tick the clock to update the velocity
        Tick the clock to update the position
For each particle
        Update the density at the location of that particle
        Update the pressure based on the density
For each particle
        Calculate the acceleration of that particle
        Half-tick the clock to update the velocity
```

4. Grid of Cells and Implementation

We simulate the fluid in a cube of 1m × 1m × 1m. We divide the cube into 10 × 10 × 10 cells. We also add an additional cell for each dimension to correctly handle particles on the boundaries, so it's actually 11 × 11 × 11 cells. The grid of cells is used to eliminate the need to refer to faraway particles when doing SPH interpolation.

We use a linked list for each cell to store particles, which is implemented using C pointers.

We also uses OpenMP to speed up the simulation.

The code is in "Code/".

5. Other work we have done

5.1. Using dynamic arrays instead of linked lists

We also tried using dynamic arrays instead of linked lists to store particles. We use a simple implementation that reallocates the dynamic arrays every time the grid information is updated. We thought that the improvement of data locality using arrays instead of linked lists might make access of particle data faster, and thus improve the overall performance. However, we found that the performance of the dynamic-array version is not as good as the linked-list version.

The code is in "Code - other versions/Array version/".

5.2. Using CUDA

We also tried transplanting our code to GPU using CUDA, hoping the mass parallelization of GPU may significantly speedup the simulation. However, we found that the CUDA version is not as fast as the CPU version. We don't know why and it may be because our data structure is not suitable for mass parallel access.

The code is in "Code - other versions/CUDA version/".

5.3. Surface Tracking and Rendering

In order to visualize the fluid we simulate, we need to generate the surface of the fluid based on the positions of the particles. We use the color field to generate the surface. The color field is a field quantity that is defined as 1 at particle locations. And the isosurface of a certain value between 0 and 1 of the color field can be seen as the surface of the fluid.

Since computer graphics convention uses a triangle mesh to represent a surface. We build a triangle mesh to represent the fluid surface using the color field, and we use the Marching-Cubes algorithm [3] to do that. The Marching-Cubes algorithm firstly divides the space into small cubes. For each cube, it tests the field values at each vertex of the cube, determine what kind of triangle configuration should be in this cube, and then move the triangle vertices along the cube edges to get the best triangles in this cube.

After getting the triangles, we put them into OpenGL using its build-in shader to render the surface out.

The result of rendering is not very beautiful, because the number of particles and the number of small cubes for Marching-Cubes algorithm are too small, and also because we didn't write our own shader specifically for the shading of the water surface. And also, since we need to evaluate the value of the color field at each vertex of a small cube, the program is much slower than the original version that do not do surface rendering.

The code is in "Code - other versions/Surface rendering/".

6. Reference

[1] M. Müller, D. Charypar, M. Gross. Particle-Based Fluid Simulation for Interactive Applications. Eurographics/SIGGRAPH Symposium on Computer Animation, 2003.

[2] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. The Astronomical Journal, 82:1013–1024, 1977.

[3] P. Bourke. Polygonising a scalar field. http://paulbourke.net/geometry/polygonise/, 1994.