Fluid Simulation with SPH method

SUN ZHENHUAN and YANG CHENGHUA

1 INTRODUCTION

In our final project, the goal is to simulate fluid, which is coupling with solid object (one-way coupling or two-way coupling).

As for our team, we want to achieve this goal though SPH method. We find some physical formula of SPH on some blog, then we implement the code of SPH according to these formula. After that, we implement boundary condition, which is equivalent to one-way coupling between rigid body and fluid. Finally, we find code which can rendering particles for SPH method.

2 IMPLEMENTATION DETAILS

First, we need to figure out physical formula of SPH method. The key idea of SPH method is, using finite amount of particles and some smooth kernel functions to represent state of any points in fluid. In SPH method, we have this formula:

$$A(\overrightarrow{r}) = \sum_i A_i \frac{m_i}{\rho_i} W(\overrightarrow{r} - \overrightarrow{r_i}, h)$$

where A is a physical quantity (such as density or pressure), $\overrightarrow{r}$ is an arbitrary position in fluid, A_i is value of physical quantity in particle i, m_i is mass of particle i, ρ_i is density of particle i, and W is smooth kernel function.

For particles, there have three type of force to drive them moving: External force (in this case, gravity is the only external force), pressure and viscous force.

First, we need to calculate density of each particles. We assume every particles have same mass. We use Poly6 function here, which is $W(\overrightarrow{r}, h) = \begin{cases} K_{poly6}(h^2 - |\overrightarrow{r}|)^3 & |\overrightarrow{r}| < h \\ 0 & |\overrightarrow{r}| \geq h \end{cases}$. To make sure $\int_{\overrightarrow{r}} W = 1$, $K_{poly6} = \frac{315}{64\pi h^9}$. With this smooth kernel function, we get formula of density is:

$$\rho(\overrightarrow{r_i}) = m \frac{315}{64\pi h^9} \sum_j (h^2 - |\overrightarrow{r_j} - \overrightarrow{r_i}|)^3$$

For pressure, according to ideal gas law, we have

$$p = K(\rho - \rho_0)$$

where ρ_0 is density of fluid in rest state. We use spiky function to interpolate pressure. Spiky function have this formula:$W(\overrightarrow{r}, h) = \begin{cases} K_{spiky}(h - |\overrightarrow{r}|)^3 & |\overrightarrow{r}| < h \\ 0 & |\overrightarrow{r}| \geq h \end{cases}$, where $K_{spiky} = \frac{15}{\pi h^6}$.

With these physical quantity, we finally can derive euqation of acceleration of particles:

$$a(\overrightarrow{r_i}) = \overrightarrow{g} + m\frac{45}{\pi h^6} \sum_j (\frac{p_i + p_j}{2\rho_i \rho_j}(h-r)^2 \frac{\overrightarrow{r_i} - \overrightarrow{r_j}}{r}) + m\mu \frac{45}{\pi h^6} \sum_j (\frac{\overrightarrow{u_i} - \overrightarrow{u_j}}{\rho_i \rho_j}(h-r))$$

where $r = |\overrightarrow{r_i} - \overrightarrow{r_i}|$, u is velocity of each particles.

After having accereration of each particles, we can update its position. In this part, we just simply update velocity by $v_{n+1} = v_n + a_n \Delta t$ and $r_{n+1} = r_n + v_n \Delta t$. Finally, what we need to do is just draw particles on the screen.

As for boundary condition, we just treat particles as rigid body, so when a particle hit the wall, we just let the particle bounce, which is, flip the direction of component of particle's velocity which is parallel to normal vector of the plane.

Again, we use Unity engine to implement our code and do rendering.

We create a function named "calculate_force", this function is used to calculate equation about acceleration. To calculate acceleration, we first need to calculate density and pressure of each particles. After that, we implement formula about acceleration. We create a function named "State_Update", this function is used to update velocity and position accroding to acceleration. Finally, we create another function called "draw_particles", this function is used to draw particles on the screen.

However, in the process we talked above, there is still one problem that haven't be solve yet: we we calculate force of each particles, we need to find neighbor particles, but, how to find it? This is a problem that need to be consider seriously, because performance of this process determined the performance of all the simulation. This problem is actually a neighbor-search problem. We implement a neighbor-search algorithm by ourself, and here is the detail:

Our method is based on spatial hashing. First, build the spatial grid and select the particle into the grid according to their positions. Then, given an index of a particle p, it's able to know the grid it belongs to and thus the 26 (since $3^3 - 1 = 26$) adjacent grid in the space. Finally, search the 27 grid and select those particle whose distance to p is less than the effect radius into a relationship list, these particles are the neighbors.

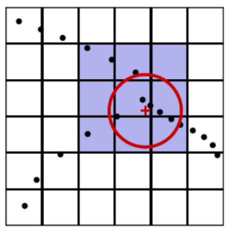

Fig. 1. Use uniform grid to solve neighbor-search problem (in 2-D)

Also, we can optimize our neighbor-search algorithm: 1. the neighbor algorithm won't change the grid and particle info themselves, thus the algorithm for each particle can be done parallelly.

2. The neighbor relation is binary, thus it's possible to only use $\frac{n^2-n}{2}$ times instead of n^2 times search. The more specific step to optimal is when discovering a neighbor relationship of p and another particle after p(particle in ascending order of index), update the relation lists of both. And pass when another particle is before p.

3. Yet Optimal2 is incompatible with Optimal1 in multithreading optimal, as Optimal2 allow a particle p to update other particles' relationship lists. When multiple particles do the search algorithm at the same time, different threads may collide when writing to the same list at the same time.

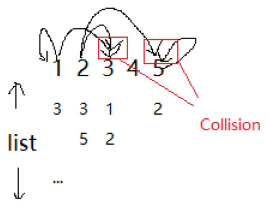

Fig. 2. sketch map of optimal 3

4. We use the "Parallel.For" function of native $C\#$ to realize multithread, and using the interlock to solve these problem results in a slower time than serial method. Therefore we abandoned Optimal2 in the end.

3 RESULTS

When we only draw particles as a sphere, and set the sphere big enough, the effect looks like this:

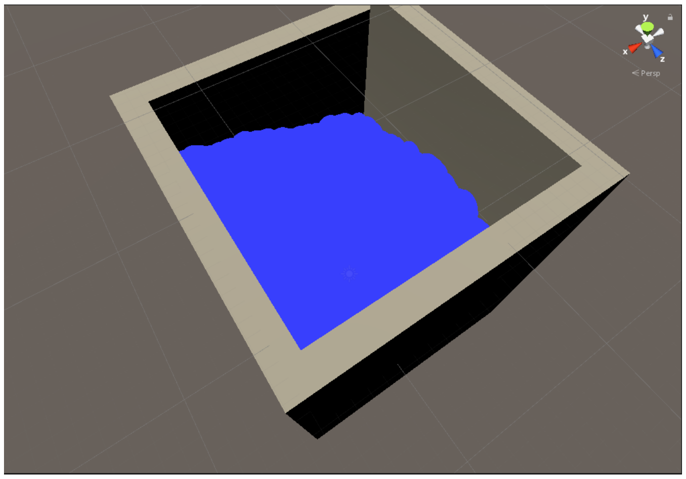

Fig. 3. The result of simulation

In this picture, we simulate a cup of water in a square cup, and we swing the cup in horizontal, we can observe that the surface of liquid is aslant, which is consist to our common sense.

After we finish the code, we find a project that contains method to rendering SPH method. The result looks like this:

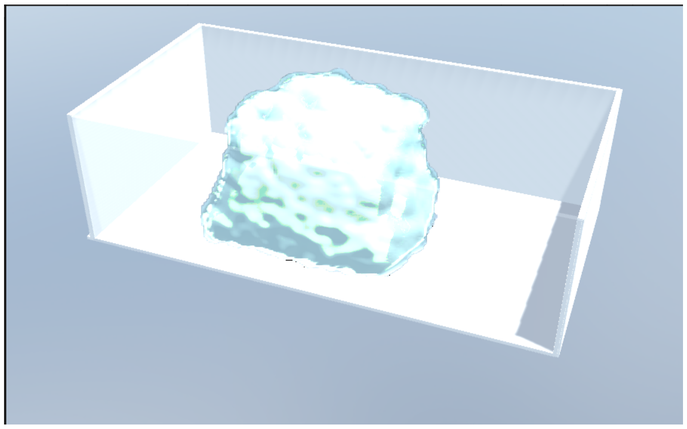

Fig. 4. The result of simulation

In these two result, I set number of particles be $15^3 = 3375$, coefficient of viscosity $\mu = 10^{-4}$, radius of smooth kernel $h = 1.8$, $K = 1000$ and $\rho_0 = 1000$ (K is for formula $p = K(\rho - \rho_0)$). For figure3, our simulation can achieve 30FPS, and for figure4, we achieve nearly 40 FPS.