COMP559 Final Project Report

Particle-based Fluid Simulation

Yue Wang
260719776

ABSTRACT

This report covers the implementation of a fluid simulation using Smoothed Particle Hydrodynamics(SPH). The method used is mostly based on the paper of [8]. In the method, the fluid is simulated by particles, so that mass conservation equations and convection terms became dispensable, which greatly reduced the computation complexity. We derive force density fields from the Navier-Stocks equation, and added a new term to model surface tension effects. We have tested this implementation using different simulation cases and obtain reasonable results, all of which are shown in the results.

KEYWORDS

fluid simulation, particle-based

1 INTRODUCTION

1.1 Motivation

Water, air, and smoke, these fluids are very common things in every day life. Simulation of fluids can be very interesting, which allows us to design creative scenes that will never happen, or simulate realistic scenes that people can't tell whether it's simulation or reality. Simulation of fluids can also be very useful, which can be applied in modern movies and video games.

1.2 Related Work

Early works such as [5] and [6] defines the water in staggered MAC grids, uses marker particles and level set to define the free surfaces. [2] gives a thorough introduction on fluid simulation methods, which covered different types of fluids, and many state of art methods. Chapter 9 introduced Smoothed Particle Hydrodynamics(SPH), which is based on [4], uses particles instead of a stationary grid simplifies the simulation substantially. in contrast to Eulerian grid-based approaches, the particle-based approach makes mass conservation equations and convection terms dispensable which reduces the complexity of the simulation. Therefore, in this project, we will mainly implement the SPH method in [8].

1.3 SPH Method

In Eulerian fluid simulation, one need to compute each term in the two equations:

$$\nabla \mathbf{u} = 0 \tag{1}$$

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{g} + v\nabla^2\mathbf{u} - \mathbf{u}\nabla\mathbf{u} - \frac{1}{\rho}\nabla p \tag{2}$$

The 1 assures conservation of mass, while the Navier-Stock equation 2 conserves momentum.
However, using particles simplifies the equations substantially. Firstly, because the number of particles is constant and the mass of each particle is fixed, the mass conservation is guaranteed automatically. Secondly, in the 2, the term $\frac{\partial \mathbf{u}}{\partial t}$ on the left hand side and the term $\mathbf{u}\nabla\mathbf{u}$ on the right hand side could be combined together and form the substantial derivative $\frac{D\mathbf{u}}{Dt}$. Since the particles move with fluid, the substantial derivative of $\mathbf{u}$ is the time derivative of $\mathbf{u}$. After the $\mathbf{u}\nabla\mathbf{u}$ is eliminated, there are three force density fields left on the right hand side of 2: pressure ∇p, external forces $\mathbf{g}$ and viscosity $v\nabla^2\mathbf{u}$. These fields are updated through interpolating from neighboring particles using different smoothing kernel functions. After these fields are obtained, we carried out a Leap-Frog integration[9] to update the positions and velocities of particles.

2 TECHNICAL DETAILS

2.1 Implementation Architecture

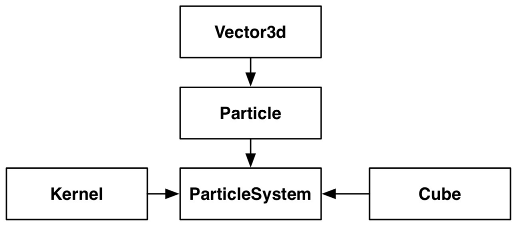

Figure 1: class diagram

We implemented the system as five parts:
Kernel is a class that encapsulates different kernel functions. In order to satisfy physical principles like symmetry of forces and conservation of momentum, the paper designed different smoothing kernel functions for pressure forces, viscosity, density, and surface tension respectively.
Particle is a single particle that stores field information like pressure, position, velocity, acceleration etc. Each particle belongs to a grid cell for computation convenience.
Cube models the container, it turns out that it doesn't necessarily to be a class. In the computation, we only need its size, because we assume that the location(center) is in the center of world space.
Particle System is the core of the simulation, it includes computing force fields of particles and update particle positions using kernel functions.

2.2 Implementation Details

2.2.1 *Particle-Grid Mapping.* The most important part of the implementation was to update position of particles, then update the grid of particles. In our implementation, the particle class holds an attribute indicating the grid cell that the particle belongs, whereas the grid was implemented as a three-dimensional array of set. Each element of the array is a set that contains all the particles fall in this grid. This mapping is done by dividing the particle location variable by the grid cell size h. After the position of particles are updated or new particles are generated, the grid will be updated by recomputing the grid index of each particle.

2.2.2 Surface Tracking. Each particle has an attribute indicating whether it is on the surface or not. The color field c_s and its normal (gradient) ∇c_s was used to identify surface particles. A particle is identified as surface particle if

$$|\mathbf{n}(\mathbf{i}_i)| \geq l \tag{3}$$

where l is a threshold parameter, the formula to compute the color field c_s and its normal (gradient) ∇c_s were given in the reference [8]. For the value of l, we will show the parameter tuning result later.

2.2.3 Smoothing Kernel Function. The reference [8] gave three different kernels for pressure, viscosity, and a general case. We implemented the kernel function as a class that handles single query for a kernel value.

2.2.4 Particle System. $ParticleSystem$ is the core of simulation. It holds all the particles as a vector, and holds the grid as an array of sets. As mentioned before, the mapping between grid and particles is done by a function in $ParticleSystem$. It is also responsible for generating scenes, for now we provided dam scene and faucet scene. In one single time step, it carries out computing all the force fields using kernel function, neighbor search function mentioned before is also included here. After the force density fields are computed, the acceleration, velocity and position is updated for each particle, the system updates the grid cell information.

2.3 Implementation Difficulties

2.3.1 Data Structure. Firstly, the smoothing kernel's computation heavily relies on the neighboring search. The data structure of particles has great influence on the efficiency of searching. The paper mentioned that they used a grid of cells of size h. Then when we query an attribute of a particle i, we only need to search iś own cell and its neighboring cells. However, in reality, if we implement every cell as a $ArrayList$ of $Particles$, interpolating among all the elements of $ArrayLists$ takes a lot of time, especially when particle density is large, even the initialization takes forever. Every time step would take more than 5 seconds to compute.
In order to speed up the computation process, we did several things: firstly we choose to implement using c++ instead of Java. We hope that using lower level data structure like vector in c++, instead of using ArrayList in Java, would improve the performance. Secondly we precomputed all the constants. For example, in the smoothing kernel formula used for density,

$$W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2) & 0 \leq r \leq h \\ 0 & otherwise \end{cases} \tag{4}$$

We precomputed the constant part $\frac{315}{64\pi h^9}$, and avoided using power function pow.
The paper also mentioned that they store copies of the particle objects in the grid cell, doubling the memory consumption and using proximity in memory to speed up computation. However, in our implementation, the computation time didn't improve much. Now we still facing the problem that when number of particles increase, the computation would slow down dramatically.

2.3.2 Explosion and Weird Behavior. During the implementation we have faced explosion of particles all the time. It turned out that not only the bugs in code could leads to explosion, improper values of parameters could also be one of the causes.
We set all terms except one to debug each term. Due to the high computation cost, we set the h to be large, used only a few particles (for example, h was set to allow at most 8 particles in each direction) to test the correctness of implementation. This is helpful to debug the explosion, however, it can also leads to very weird behavior, which was treated as a bug at first.

3 RESULTS

We used dam break and faucet scenes for testing and recording a demo. From the screen shot we could see that there are some splash

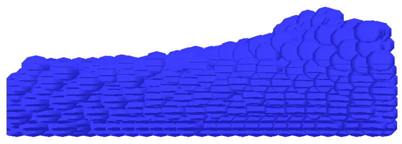

Figure 2: Dam Model Simulation

effect from the dam model, however, we weren't able to generate a scene with more splash effect, no matter how cliffy the dam is. We think it might be because of the viscosity parameter, therefore, we also tuned the viscosity parameter and observe the splash effect, we showed the result of this part in later sections.

3.1 Interaction

We added mouse interaction to simulate external forces. The system could detect the mouse drag and apply the force accordingly to all the particles. The force direction is the drag direction, the force amount is the drag distance.

3.2 Rendering

In order to achieve high-quality rendering, we used marching-cube algorithm from [1] to extract isosurfaces. At first, we tried to generate isosurface from particles that labeled as surface particles. We start by searching from all the cells that contain surface particles and from there recursively traverse the grid along the surface. However, this was proved to be a failure since this has close relation with the value of the color field threshold l, and it is very likely that the surface particles won't cover the water body entirely and evenly. Also the mesh files generated were very coarse and unrealistic. We think this could be solved by set the number of particles to be larger, however, it would be much better if there are ways to upsample the surface of the fluid. According to [3], we used the function

$$\Phi(\mathbf{x}) = (\Sigma_j (1 - r/h)^2)^{1/2} \tag{5}$$

After the mesh files were generated per time step, then we fed them into renderer Mitsuba[7]. We weren't able to deliver the high-quality rendering version because we were not able to provide Mitsuba with a valid xml description file.

Another problem remained unsolved was point splash. This case should be dealt independent from the water body. We weren't able to solve this problem. Given more time we would be able to give better results.

3.3 Parameter Tuning

3.3.1 color field threshold. The color field c_s is used to identify the surface particles. However, the threshold for equation 3 was not given in references. We marked the surface particles as red, trying to test different l and choose one that visually best in dam scene. Here are some of our results. We could see from Figure 3, smaller l will

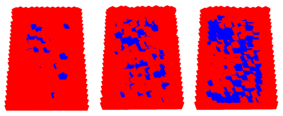

Figure 3: color field threshold l = 2, 5, 20

lead to more particles to be marked as surface particles(marked as red). However, this would cause trouble when we tried to generate isosurfaces from surface particles - more surface particles gave complex isosurfaces and consumed more time. In the end we used 5 to generate isosurfaces, therefore it doesn't matter whether we choose a bigger or smaller value of l.

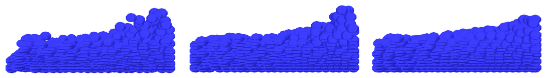

Figure 4: viscosity coefficient v = 2, 5, 10

3.3.2 color field threshold. From Figure 4, we could see that the splash effect is closely related to the viscosity coefficient. When the viscosity coefficient is small, like v = 2, we could observe more splash effect than v = 10.

4 CONCLUSION AND FUTURE WORK

We have implemented a particle-based fluid simulation. The physical model is based on Smoothed Particle Hydrodynamics, we optimized computation of smoothing kernels to speed up computation. We used color field to track and render the free surface of fluids. However, the high-quality rendering is not fully implemented, leaving several problems unsolved. In the future, we will investigate upsampling techniques to improve the mesh quality, and deliver Mitsuba description xml file automatically from our implementation.

REFERENCES

[1] Paul Bourke. 1994. Polygonising a scalar field. (1994). http://paulbourke.net/geometry/polygonise/

[2] Robert Bridson and Matthias Müller-Fischer. 2007. Fluid simulation: SIGGRAPH 2007 course notes Video files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 courses*. ACM, 1–81.

[3] 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*. ACM, 219–228.

[4] Mathieu Desbrun and Marie-Paule Gascuel. 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and SimulationâĂŹ96*. Springer, 61–76.

[5] Douglas Enright, Stephen Marschner, and Ronald Fedkiw. 2002. Animation and rendering of complex water surfaces. In *ACM Transactions on Graphics (TOG)*, Vol. 21. ACM, 736–744.

[6] Nick Foster and Ronald Fedkiw. 2001. Practical animation of liquids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 23–30.

[7] Wenzel Jakob. 2010. Mitsuba renderer. (2010). http://www.mitsuba-renderer.org.

[8] Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 154–159.

[9] Constantine Pozrikidis. 1998. *Numerical computation in science and engineering*. Vol. 6. Oxford university press New York.