Particle-Based Fluid Simulation

CSC417: Physics-Based Animation – Final Project

Eric Koehli
University of Toronto
eric.koehli@mail.utoronto.ca

ABSTRACT

In this project, I use the smoothed-particle hydrodynamics (SPH) method described by [Müller et al. 2003] to simulate water using particles.

KEYWORDS

Fluid simulation, Smoothed-particle hydrodynamics, SPH, Particle-based fluid simulation

ACM Reference Format:
Eric Koehli. 2021. Particle-Based Fluid Simulation: CSC417: Physics-Based Animation – Final Project. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Fluids in the computer graphics world can represent anything from wind, water, fire, to snow. We're all familiar with everyday fluids such as water, but rarely think about the computational complexity required to simulate them. Two of the most common approaches to fluid simulations are the Eulerian grid-based approach and the particle-based Lagrangian approach, which is the method I take in my implementation. Each method has their place in graphics frameworks and each come with their own trade-offs. Some of the attractive properties of SPH methods include mass-conservation, Lagrangian discretization, and computational simplicity [Macklin and Müller 2013]. Many real-time applications such as video games that desire a high degree of fidelity, while also maintaining performance requirements, choose SPH-based methods for these reasons.

2 RELATED WORK

Computational Fluid Dynamics has a long history that started before the advent of computers: In 1822 Claude Navier and in 1845 George Stokes formulated the famous Navier-Stokes equations which describe the dynamics of fluids [Müller et al. 2003]. All the work I describe here as part of my project has been developed by [Müller et al. 2003] and I refer the reader to their paper for more a more comprehensive understanding of their work.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

3 BACKGROUND

In this section, I will describe the core mathematical formulations used in a smoothed-particle hydrodynamics approach.

3.1 Smoothed Particle Hydrodynamics

SPH-based methods discretize the fluid using 'particles'. I would like to emphasize this point because I found this to be confusing while I was learning. It might be natural to think of a particle as representing a fluid molecule, say water. However, a more accurate analogy would be to think of each particle as representing a small *volume* of water:

$$V_i = \frac{m_i}{\rho_i}$$

where m_i is the mass of the particle and is usually defined as a constant for all particles in the system, and ρ_i is the density.

The central concept of an SPH system is that it's an interpolation method for the particle system. Particles store attributes such as density, pressure, mass, velocity, etc. When we want to evaluate an attribute, we can simply take a weighted average of particle values within a specific radius of the particle of interest:

$$A(\mathbf{r}_i) = \sum_j \frac{m_i}{\rho_i} A_j W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{1}$$

In this case, A is any scalar quantity that is interpolated at location $\mathbf{r}_i$ by a weighted sum of the contribution from all other particles j.

The function W is called the smoothing kernel and has a radius of influence h. This means that the smoothing kernel has finite support, which is good news for efficiency: we don't need to iterate over all other particles j, only those in which $r = ||\mathbf{r}_i - \mathbf{r}_j|| < h$.

Since the density ρ varies for each particle, this scalar quantity needs to be computed at each time step. Substituting the density ρ into equation (1) we get

$$\rho(\mathbf{r}_i) = \sum_j m_i W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{2}$$

One of the features about using SPH is that the derivatives of these field quantities only affect the smoothing kernel W. For instance, if we take the gradient of a scalar quantity A, we get

$$\nabla A(\mathbf{r}_i) = \sum_j \frac{m_i}{\rho_i} A_j \nabla W(\mathbf{r}_i - \mathbf{r}_j, h)$$

and similarly if we take the Laplacian ∇^2.

3.2 Equations of Motion

When modelling fluids with particles, it's no surprise that we're going to see the famous *incompressible* Navier-Stokes equation:

$$\rho \left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v} \right) = -\nabla p + \rho \mathbf{g} + \mu \nabla^2 \mathbf{v} \tag{3}$$

For brevity, and for the purpose of my implementation, there are three force density fields on the right hand side of this equation that we need to determine: the pressure force $-\nabla p$, the external forces $\rho \mathbf{g}$, and the viscosity force $\mu \nabla^2 \mathbf{v}$. We will see how to compute each of these in turn.

3.3 Pressure

After the density has been computed from equation 2, we can calculate the pressure by using the ideal gas state equation

$$p = k\rho$$

where k is a gas constant that depends on the temperature. From the original article, [Müller et al. 2003] actually use a modified version

$$p = k(\rho - \rho_0) \tag{4}$$

where ρ_0 is the rest density. The overall effect of this modification is to increase the simulation's numerical stability. Some interesting observations I observed is when the density ρ_i is much larger than the rest density ρ_0, this results in a much larger pressure and consequently a larger repulsion forces between the particles, and vice versa when the density is lower than the rest density.

As noted in Section 3.1, when we apply a gradient to a scalar field such as pressure p and substitute into equation 1, we obtain the incredible result:

$$\mathbf{f}_i^{\text{pressure}} = -\nabla p(\mathbf{r}_i) = -\sum_j \frac{m_j}{\rho_j} p_j \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{5}$$

But this is not entirely accurate because the equation above violates Newton's 3rd law that every action (force) has an equal and opposite reaction. This is due to the fact that the above equation is not symmetric between particles i and j. However, there is a simple and elegant fix to this problem by taking the mean of the pressure values and making the substitution

$$p_j = \frac{p_i + p_j}{2}$$

which gives us the symmetric force due to pressure. One way to think of the pressure force is that it aims to restore the rest state of the fluid where $\rho_i = \rho_0$. Lastly, the pressure force acts along the vector between the particles, which can be seen when you take the gradient of the smoothing kernel.

3.4 Viscosity

Once again, when we apply equation 1 to the viscosity term $\mu \nabla^2 \mathbf{v}$ and take the Laplacian, we end up with asymmetric forces, which again, is elegantly symmetrized by taking the velocity differences:

$$\mathbf{f}_i^{\text{viscosity}} = \mu \nabla^2 \mathbf{v}(\mathbf{r}_i) = \sum_j \frac{m_j}{\rho_j} (\mathbf{v}_j - \mathbf{v}_i) \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{6}$$

Some great intuition of the viscosity force was stated by the authors: "... if you look at the neighbors of particle i from i's own moving frame of reference, then particle i is accelerated in the direction of the relative speed of its environment."

3.5 Smoothing Kernels

I don't have much room left, so please refer to the original paper by [Müller et al. 2003] for details of the smoothing kernels I used in my implementation.

4 IMPLEMENTATION

My implementation follows the same structural layout as the details given in the paper [Müller et al. 2003] as well as that presented here. The core of my simulation loop is as follows:

Algorithm 1: Simulation Loop

1 `// Compute the density and pressure for each particle`
2 **foreach** *particle i* **do**
3 Find all neighboring particles $\mathcal{N}_i(\mathbf{r}_i)$
4 Use equation 2 to compute density ρ_i
5 Use equation 4 to compute pressure p_i
6 `// Compute the forces for each particle`
7 **foreach** *particle i* **do**
8 Use equation 5 to compute $\mathbf{f}_i^{\text{pressure}}$
9 Use equation 6 to compute $\mathbf{f}_i^{\text{viscosity}}$
10 Sum all the forces on each particle: $\mathbf{f}_i = \mathbf{f}_i^{\text{pressure}} + \mathbf{f}_i^{\text{viscosity}} + \text{gravity}$
11 `// Sympletic Euler Integration + handle wall collisions`
12 **foreach** *particle i* **do**
13 $\mathbf{u}_i^{t+1} = \mathbf{u}_i^t + \Delta t * \frac{\mathbf{f}_i}{\rho_i}$ `// update velocity`
14 $\mathbf{r}_i^{t+1} = \mathbf{r}_i^t + \Delta t * \mathbf{u}_i^{t+1}$ `// update position`
15
16 **if** *new position is outside container* **then**
17 Reflect the velocity component that's perpendicular to the object's surface and push the object inside.

5 RESULTS

As you can see from the algorithm above, it's a fairly straightforward application of the theory built up by [Müller et al. 2003]. There are a couple key points of the algorithm worth discussing: The core of the algorithm is somewhat set in stone, in the sense that you can't compute forces before computing the densities and pressures and you can't try to shortcut the computation by computing the forces within the same loop as density (I tried!). However, you are entirely free to come up with your own data structure for finding neighboring particles on line 3 of the algorithm. In the original paper, [Müller et al. 2003] discuss a common method to reduce the computation complexity by using a grid divided into squares (or cubes in 3D) of size h (which is the kernel's radius of support). This cuts the computational cost significantly because you now only need to check particles in it's own grid square plus the surrounding 7 neighboring grid squares. This is the approach I took in my implementation, however, I did not really see much improvement over a more naive approach. I believe the reason for this is because I could not simulate enough particles on my PC for the grid data structure to actually be worth maintaining. Another benefit to using a Grid-based data structure to maintain the particles is that it naturally allows for parallelism [Braley and Sandu 2009].

On my computer, I was able to simulate ~ 100 particles using both a naive approach to finding neighboring particles and using

a grid data structure. Interesting, I was able to simulate around ~ 400 particles using MATLAB's `rangesearch` function.

6 CHALLENGES

This is not part of my submission since it's over two pages, but there are some additional things I would like to mention. There were too many challenges in this project to mention them all, but some of which I feel the need to elaborate.

6.1 Why I chose Python

Finding some way to render my simulation was a difficult task for me because I wanted to focus on the core aspects of the problem and not worry so much about learning all the details of OpenGL or something similar, and Python seemed to be the language that could offered me this level simplicity. My original plan was to use Python only to develop a quick 'Proof of Concept' to see that my implementation could work and was working before porting it over to C++. Little did I realize how much time I would spend just trying to get this 'proof of concept' to work that I never got the chance to re-implement this code in C++. However, after the course is finished this is a small project I would like to complete.

6.2 Finding tuning parameters

Finding some set of parameters for the simulation such that it would actually look like a fluid was a challenging task. Part of this problem is because it's so dependent on the size of the window frame, computer processing speed, time step, and much more. There were some great resources I found to help with this process beyond trial-and-error testing:
Stanford Graphics Project
SPH Survival Kit

7 REFERENCES

I would like to add a couple additional references that really helped me with this project and that were not added to the other "References" section because I didn't specifically cite them in this technical brief:
[Bridson 2008] Chapter 15 on SPH
Cornell Spatial Binning Grid Data Structure - Bindel
CMU Animation Lecture Slides on SPH - Scoros

REFERENCES

C. Braley and A. Sandu. 2009. Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Based Methods.

Robert Bridson. 2008. *Fluid Simulation.* A. K. Peters, Ltd., USA.

Miles Macklin and Matthias Müller. 2013. Position Based Fluids. *ACM Trans. Graph.* 32, 4, Article 104 (July 2013), 12 pages. https://doi.org/10.1145/2461912.2461984

Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-Based Fluid Simulation for Interactive Applications. In *Symposium on Computer Animation*, D. Breen and M. Lin (Eds.). The Eurographics Association. https://doi.org/10.2312/SCA03/154-159