An Implementation of Smoothed Particle Hydrodynamics

Jesse Sprinkle
Texas A&M University Computer Science
College Station, USA
jrs7943@tamu.edu

***Abstract*—This document was written to provide insight into my implementation of Smoothed Particle Hydrodynamics. The implementation was done as a final project for the course CSCE 489 Computer Animation, Fall 2021.**

I. Introduction

The goal for this project was to produce a real-time fluid animation. The chosen fluid simulaiton model was Smoothed Particle Hydrodynamics (SPH). In this report, the research, design, parameters, and implementation of this model will be described.

II. Smoothed Particle Hydrodynamics

SPH is a fluid simulation model developed in the 70s by Monaghan and Gingold, original paper here [1]. The general idea is this: fluids are modeled by the Navier-Stokes Equations (NSE) over a continuous field. We discretize this field, then calculate each parameter at each time step. In this method, we use the incompressible varient of the NSE:

$$\frac{du}{dt} = -\frac{\nabla p}{\rho} + v\nabla^2 u + g \tag{1}$$

Where p is the pressure, ρ is the density, v is viscocity, u is the velocity, and g is external forces such as gravity.

A. Derivation of Discretization

To derive this discretization, we start with the following fact:

$$A(z) = \int_R A(x)\delta(x-z)dx \tag{2}$$

If you wish to know more about this equation, read into the Dirac Delta function. This equation is essentially states that any point can be calculated by adding every point over the field and multipying it by zero unless it is the point you are trying to calculate the value of, just in continuous terms.

We can't discretize this integral, as we don't know how to discretize the Dirac Delta function. So, we approximate it with a weighting function W, called a kernel. We assume $A(x)$ is continuous, and as a result nearby points should be similar to $A(z)$ which makes replacing with a kernel reasonable.

$$A(z) = \int_R A(x)W(|x-z|, h)dx \tag{3}$$

The kernel takes in a distance to z and a smoothing length h, which is there to give the function a reference of distance. For example, in many cases if $|x-z| > h$, then $W(|x-z|, h) = 0$.

We can now approximate this integral with a summation to discretize. However, we cannot neglect the volume term in moving from a continuous space to a discrete space:

$$A(z) = \sum_i V_i A(x_i)W(|x_i - z|, h) \tag{4}$$

It is from this equation we can approximate every term of the NSE at a particular point, for example:

$$\rho(z) = \sum_i V_i \rho(x_i)W(|x_i - z|, h) = \sum_i m_i W(|x_i - z|, h) \tag{5}$$

where m is mass.

It is important to know that modifications are often made depending on the situation, so not every SPH implementation uses the same equations. We use the following equations:

$$-\frac{\nabla p(x_i)}{\rho(x_i)} = \sum_k m_k\left(\frac{p(x_i)}{\rho(x_i)^2} + \frac{p(x_k)}{\rho(x_k)^2}\right)\nabla W(|x_i - x_k|, h) \tag{6}$$

$$\nabla^2 u = \sum_k m_k \nabla^2 W(|x_i - x_k|, h)(u(x_k) - u(x_i)) \tag{7}$$

The Laplacian term in Eq. 7 is actually the scalar that would be multiplied by a vector to get the Laplacian. Most of the equations used in the final implementation are located here [2] [3]

We can now create particles that contain the data, such as position, density, mass, of the discrete space.

III. Other Equations and Parameters

At the start of the simulation, it is expected the position, velocity, and mass of the particles to be known. From this we can use Eq. 5 to calculate density. There are multiple choices in pressure calculations. There are two common choices:

$$p_i = k\left(\frac{\rho_i}{\rho_0} - 1\right) \tag{8}$$

$$p_i = k\left(\left(\frac{\rho_i}{\rho_0}\right)^7 - 1\right) \tag{9}$$

Where k is a stiffness constant to be tuned as wished and ρ_0 is a reference density (for example $1000kg/m^3$ for water). There exist many more complex equations for pressure. We use Eq. 8 in our implementation. Often, the reference density is decided upon, then the mass is calculated as such:

$$m = \rho_0 h^3 \tag{10}$$

Another choice is the kernel. There exist many kernels, however I used the one in [2]. The smoothing length h is situationally based on the kernel and simulation. For example, if the particles are intended to be small, h should be smaller. In general, it is a parameter you tune based on model scale and optimization.

The viscocity constant v is another parameter to tune based on the fluid you would like. In general v will be much higher than real world viscocities [3]. The external acceleration constant g is also to be tuned as wished. If your model is to-world scaled for example, (0, -9.8, 0) could simulate gravity.

IV. Simulating Walls

It is often nice to have the fluids in a container. To create a one, you can simply put invisible immovable particles with a custom initial density to better reflect particles, or sometimes have multiple layers of particles. More generally, you can bind fixed particles to any surface you want particles to collide with.

V. Optimization

The runtime to calculate values for each particle in a standard search of all the particles is $O(n^2)$. This cannot be used to simulate thousands of particles in real time. There are dozens of search algorithms out there, but I will only explain the method I used.

To find the nearby particles, a grid with cells of size h was used. At each time step, each particle sorted itself into the grid based on its position. To search for nearby particles, each particle would only check the cell and surrounding cells for particles. This vastly improved simulation speed.

To increase speed further, we can utilize multithreading. From the equations above, it is easy to see how many worker threads running at a time can calculate all the independent values.

To increase render speed, as thousands of particles are rendered per frame, we can use instancing.

VI. Implementation

The initial parameters are set as such: $h = 0.1$, $v = 0.01$, $k = 2.0$, $g = (0, -0.23, 0)$, $dt = 0.05$. For wall particles $d_0 = 2500$ while for regular particles $d_0 = 1000$. In my scene, the viscosity lowers to 0.0004 after 6 seconds, as the initial viscosity is there to get the fluid stable.

A. Scene Initialization

To create the simulation, I created a grid of particles (with small random offsets) for the fluid and created a larger box around it without a top. The wall particles and fluid particles are put into two vectors so we only have to iterate over the fluid particles.

Creating worker threads for each iteration is too slow, so we create 80 workers threads for calculating density and 80 worker threads for calculating accelerations and store them. These threads are given a range of particles the calculate their values over and wait for the step function to tell them to calculate their section.

B. Stepping

At the start of each time step, we update the grid with the positions of the particles. Then, we activate the density worker threads which calculate the density and pressure of each particle. When all are completed, the acceleration worker threads wake up and calculate the acceleration each particle. The particle velocities get updated, followed by the particle positions (we use implicit euler which is the standard method).

VII. Results

While the model can handle tens of thousands of particles, my model can only handle 3000 fluid particles to run in near real time. However, for larger particle counts we do get a decent simulation as can be seen in Fig. 1. It is clear more optimization needs to be done to simulation things on a larger scale.

VIII. Revisions

In the future, I plan to further optimize my model. A few things I plan on doing is switching vectors of particles of data to particles backed by arrays of data to better utilize cacheing. I plan on implementing a faster particle search algorithm, perhaps with some cacheing and hashing, or only updating the particle grid positions every other time step.

One optimization I haven't seen anything on is a replacement for particle based walls. Though I'm not sure the performance increase, I may attempt to replace the wall particles with planes and bounce the particles off them.

A final change that I would like to do is skin the fluid. Currently the model is just a bunch of particles, so implementing a nice fluid look could really improve the looks.

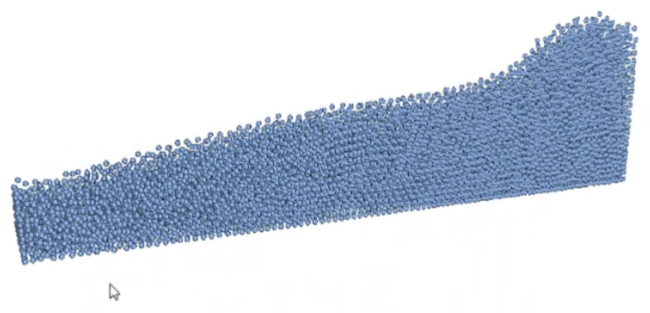

Fig. 1. !3,000 Particles Swaying in a Container

IX. Conclusion

I learned much from this project, from fluids to optimizations. While not the most grand or impressive of projects, I genuinely felt challenged by the research and problem solving I had to do to get my model in the state it is.

References

[1] R. A. Gingold, J. J. Monaghan, Smoothed particle hydrodynamics: theory and application to non-spherical stars, Monthly Notices of the Royal Astronomical Society, Volume 181, Issue 3, December 1977, Pages 375–389

[2] D. House, J. Keyser, Foundations of Physically Based Modelling and Animation, CRC Press, 2017, Pages 283-292

[3] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, M. Teschner, SPH Fluids in Computer Graphics, EUROGRAPHICS, 2014