SPH Demo

Nalin Mahajan, Vineeth Bandi

May 9, 2022

1 Introduction

In this project, we implemented the basic smoothed-particle hydrodynamics (SPH) loops by discretizing the naive-stokes to create a simulation that modeled a fluid. The goals of this project were to first and foremost gain a basic grasp on the fundamentals behind SPH. Then we sought to create a believable demo using what we learned as well as improving general performance so it can run successfully on our laptops. We believe that we have been successful in doing so providing a demo of a dam break that allows the user to tweak several simulation parameters and interact using the mouse, providing in our perspective a realistic simulation of a fluid in a box. The source code and demo are located here: https://github.com/nalin29/sph_final

2 Theory

In class we discussed discretizing a fluid by creating a grid by which we can track the velocity at each corner and compute the pressure and other required values for each grid cell and then integrate that across the fluid by bi-linearly interpolating these values. This is an Eulerian approach to fluid simulation which seeks to identify fixed values and then observe these locations and update the velocity field accordingly. Instead, SPH is a Lagrangian approach. This approach involves discretizing and representing the fluid as a group of particles. These particles will represent a position to sample and contain their own field values. This forms the basis for the approximation of the fluid.

The general process of SPH works as follows; we have a set of field values associated to each particle, such as mass, and some functions we wish to approximate such as calculating density. We can then use a kernel convolution which serves as a weighting function and apply the function at neighboring particles. These samples are integrated over by taking a weighted sum using the kernel. We can then use these samples along with the numerical approximation of the required differential operators on each function to provide a particle discretized view of the fluid. For instance, we can consider some function A that can be calculated over the fluid and each particle contains some field m_i and ρ_i representing mass and density which is discretized using some kernel W. We can form this equation to discretize the function over each particle, i at position x_i.

$$(A * W)(x_i) \approx \sum_j A_j \frac{m_j}{\rho_j} W(x_i - x_j, h)$$

This simple approximation forms the basic intuition behind SPH. From this approximation we can try and approximate the discrete differential operators directly. For instance, we can

approximate ∇A_i by directly applying the gradient operation to the above approximation. To yield that $\nabla A_i \approx \sum_j A_j \frac{m_j}{\rho_j} \nabla W(x_i - x_j, h)$. Similarly, we can apply this direct approximation to the other differential operators.

$$\nabla A_i \approx \sum_j A_j \frac{m_j}{\rho_j} \nabla W(x_i - x_j, h)$$

$$\nabla \cdot A_i \approx \sum_j A_j \frac{m_j}{\rho_j} \cdot \nabla W(x_i - x_j, h)$$

$$\nabla \times A_i \approx -\sum_j A_j \frac{m_j}{\rho_j} \nabla W(x_i - x_j, h)$$

$$\nabla^2 A_i \approx \sum_j A_j \frac{m_j}{\rho_j} \nabla^2 W(x_i - x_j, h)$$

These direct approximations however can have high error and in practice may be unstable. Therefore we can augment these equations to recover 0th-order and 1st-order error and improve stability. We can recover 0th-order by subtracting by the 0th-order error term to yield.

$$\nabla A_i \approx \sum_j \frac{m_j}{\rho_j} (A_j - A_i) \nabla_i W_{ij}$$

Other approximations such as the symmetric formula: $\nabla A_i \approx \rho_i \sum_j \frac{m_j}{\rho_j} (\frac{A_i}{\rho_i^2} + \frac{A_j}{\rho_j^2}) \nabla_i W_{ij}$. Have different properties such as conserving momentum. Or a better approximation for the Laplace operator: $\nabla^2 A_i \approx -\sum_j A_{ij} \frac{m_j}{\rho_j} \frac{2||\nabla W(x_i - x_j, h)||}{||r_{ij}||}$.

We can then use these approximations along with an incompressibility condition which states that the density of the fluid remains fixed represented by $\frac{D\rho}{Dt} \approx 0$ to solve for the naive-stokes equation we saw in class namely:

$$\rho \frac{Dv}{Dt} = -\nabla p + \mu \nabla^2 v + F_{ext}$$

Using the algorithim below to update the velocity field and position of each particle:

Algorithm 1: Simple SPH Algorithim

for *all particle i do* **do**
- Reconstruct the density ρ_i at x_i
- $\rho_i = \sum_j m_j W_{ij}$
- Compute the pressure using gas equation
- $p_i = k(\rho_i - \rho_0)$

for *all particle i do* **do**
- Compute the $F^{\text{pressure}} = -\frac{1}{\rho}\nabla p$
- $F_i^p = \sum_j m_j(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2})\nabla W_{ij}$
- Compute the $F_i^{\text{viscosity}} = m_i \mu \nabla^2 v_i$
- $F_i^v = \mu \sum_j \frac{m_j}{\rho_j} v_{ij} \frac{2|||\nabla W_{ij}||}{||r_{ij}||}$

Perform time integration

for *all particle i do* **do**
- $v_i^{t+1} = v_i^t + \frac{\Delta t}{m_i}(F_i^v + F_i^p + F_i^{\text{ext}})$
- $x_i^{t+1} = x_i + \Delta t v_i^{t+1}$

3 Demo Implementation

The demo simulates a dam break in 2d. This is represented by suspending a block of particles separated by a constant length suspending above the floor. When the simulation starts the force of gravity causes the fluid to fall onto the floor providing a good understanding and visual of the fluid forming a wave as it expands and breaks on the edges of the box.

To render the dam break we utilize polyscope and re-tooled the rendering code for the previous fluid simulation to allow its usage to visualize our demo. Polyscope, unfortunately, ties the simulation to the framerate, so we target a stable 60 fps for a good simulation. We are able to reach that on more powerful machines even with a lot of particles. On our laptops, however, we are restricted to a lower frame-rate. Although this doesn't hurt the presentation of the demo, it does slow down the physics.

The particles and their representative values such as position, force, density, and pressure are represented using Eigen Vectors and Matrices. The particle positions are initialized in the top half of the grid, each seperated in height and width by a half of the sampling radius for the kernel. The user may specify in the polyscope GUI the total number of particles for the simulation.

For the simulation, we operate a tight loop of first performing time integration using Velocity Verlet. Velocity Verlet was chosen since it is simpler than implicit integration which is not strictly necessary to have a good effect. Velocity Verlet is also more numerically stable than Explicit Euler which was important as the simulation with many particles can easily spiral out of control.

Then we regenerate the density at each particle. This is done by looping over all particles and then all particles again. We skip over particles whose distance is larger than the radius of the neighborhood that we defined as a user augment-able constant. We originally, wanted to follow the above algorithm using a basic kernel such Gaussian or Cubic-splice. However, we found that this could easily blow up the simulation or cause there to be a distinct lack of pressure. Therefore, we decided to create a much simpler weighting function. We first compute $1 - (||r_{ij}||/h)$, then create a quadratic and cubic term to create two densities: one that takes a broader sample and another that weights closer particles more. These densities

are summed up and stored in the requisite eigen vectors.

To compute pressure we again loop over every particle and utilize the gas equation with a user augmentable gas constant k. We use the gas equation to provide a pressure estimate using the broad estimate of density. We additionally compute a pressure component using the nearby density to help keep particles together.

We then compute the force of pressure using a simplified equation of the SPH algorithm. We utilize the more basic direct approximation for this. We incorporate the nearby pressure estimate by adding it to the force pressure by the square of the kernel. The force of pressure is then added to the force Eigen vector for that particle along the direction of r_{ji}.

We then calculate the force viscosity. Viscosity creates extra adhesion and to some regard a surface tension to the fluid making it more believable. We utilize the difference optimization for the direct formula to calculate the viscosity. this is a simple addition and multiplication of scalar values and a precomputed constant laplacian for the viscosity kernel.

We also included some external forces. Firstly, the most important is gravity which simply provides a downwards force on each particle in the fluid which we apply during time integration. Additionally, we include a mouse force that allows the user to interact with the fluid in the simulation. This force is given by an attractor to the mouse. When the mouse is clicked within some radius of the mouse the particles experience a force inversely relative to its distance from the mouse location in the direction of the mouses position. This attracts the particles allowing the user to make waves by dragging the mouse. Additionally, we utilize a simpler velocity based impulse to keep particles within the box, which serves us well.

To help optimize the simulation we utilized OpenMP which allows us to perform some of the for loops in parallel with just annotations in the code. This provided a noticeable performance benefit, although even with 1000 particles we could comfortably run at 60fps on our desktop computers.

4 Issues and Future Implementation

Some of the various issues and difficulties we found with the simulation had some to do with our lack of knowledge with SPH and in general the difficulty of tuning parameters just right. As mentioned, we forgoed utilizing one of the standard kernels because we were getting inconsistent results, the simulation would often blow up or produce fluids whose pressure levels were too low to produce a nice visual result. In the future we would like to try spending more time tuning for these kernels and extending the simulation to 3d. Additionally, as mentioned even without the special kernels tuning parameters such as the size of the neighborhood, the gas constant and viscosity took a lot of time and small variations can cause the simulation to break. Although small enough variations can produce the results we seek while also causing a meaningful visual change in the simulation such as increasing viscosity.