SPH Fluid Simulation

Arnav Sastry, Brady Zhou

I. Project Details

In this project, we simulate fluid motion using smoothed particle hydrodynamics (SPH). This is a Lagrangian method for fluid flow, and we based this approach off of [3], [4], and [1].

Fluids differ from typical rigid body motion, as the configuration space is now an infinite dimensional vector consisting of fluid velocity and density at every single point. Using the SPH approach allows us to discretize the infinite dimensional space into a finite dimensional configuration space where the fluid is now sampled using a discrete number of particles. The heart of the physics for fluid motion is the Navier-Stokes equations, where the change in velocity is represented as

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

Subject to $\nabla u = 0$, where ρ is density, p is the pressure field, g is external forces, and v is a viscosity constant. Intuitively, the first term states that velocity changes to minimize pressure differences, and the second term is to account for the forces that smooth the velocity field. The next step is to discretize this into a set of n particles.

$$\frac{d\mathbf{u}_i}{dt} = -\frac{1}{\rho_i}\nabla p_i + v\nabla^2 u_i + g_i \tag{2}$$

Now, each particle has a couple of attributes that change as a function of time. The main ones are ρ and p. The first step in constructing an integrator for this is how to approximate the gradient over a field, given a sample of particles. The core piece behind this is the use of kernel functions, which gives the amount of weight contributed to a given attribute as a function of distance. If the value of an attribute ϕ is to be queried at a single point, the estimate is given as

$$\phi(x) = \sum_{i=1}^{n} \frac{m_i}{\rho_i}\phi_i w(||x - x_i||) \tag{3}$$

where m_i, ρ_i, x_i are the mass, density, and position of the i-th particle, and w is the kernel function. The kernel function we used is a cubic spline function with a finite support s from [2].

$$w(r) = \frac{1}{\pi s^3}\begin{cases} 1 - \frac{3}{2}(\frac{r}{2})^2 + \frac{3}{4}(\frac{r}{s})^3, & \text{if } 0 \le \frac{r}{s} \le 1. \\ \frac{1}{4}(2 - \frac{r}{2})^3, & \text{if } 1 \le \frac{r}{s} \le 2. \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Now, ∇p_i and $\nabla^2 u_i$ are estimated by simply taking first and second derivatives of the kernel function and applying the chain rule. The derivations are trivial and left as exercise to the Vouga.

II. Implementation Details

The code uses the Nanogui and LibIGL libraries for rendering and GUI controls.

Each particle keeps track of its own properties, like density, volume, and polarity. The bulk of the relevant code is in "simulation.cpp". Our code updates the positions, then the velocities using a Velocity Verlet inspired time integrator. After the velocities are updated, we also update the density of each particle.

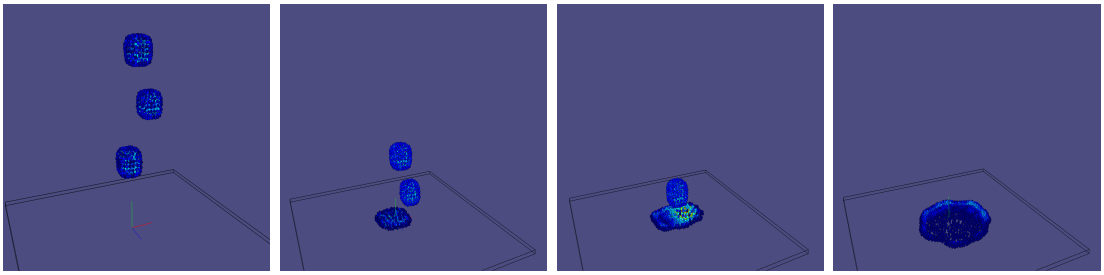
There are several modes for viewing the simulation. The fastest to run is rendering each particle as a small sphere. We color the sphere based on its density (red means higher density). However, fluids are not collections of particles, so there is also a surface mode which uses the marching cubes algorithm to approximate a surface given properties of the particles. As the constants we settled on made our fluid more like honey than water, our surface is yellow.

Since this model uses a particle based approach, adding particle-particle and particle-plane collisions are easily handled by integrating previously written penalty force methods and impulse methods.

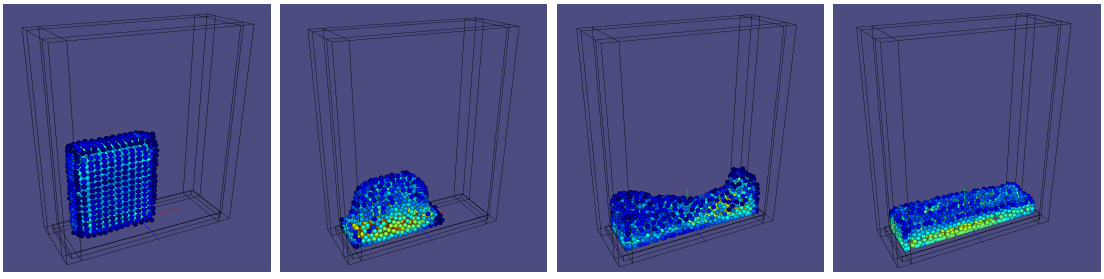
A naive implementation of this would require $O(n^2)$ work since every particle depends on attribute estimations, which is $O(n)$. Since the kernel function used only operates with a given support radius, we implement a axis-aligned bounding box tree to allow for point queries within a radius in $O(log(n))$ for the average case. This allow for "real time" renderings of 1000 particles in around 15 fps.

III. Results

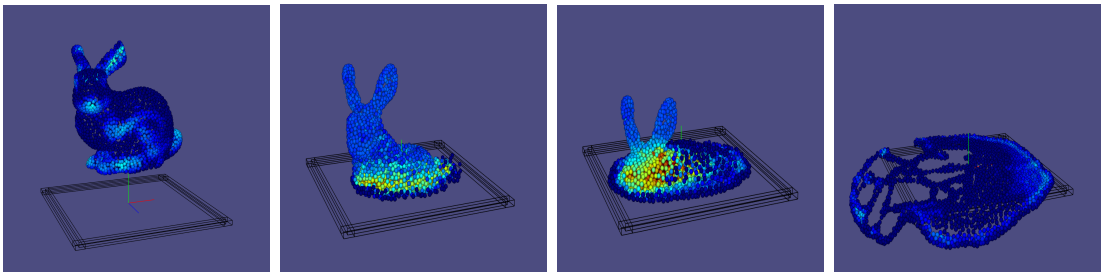
We construct a few scenes to demonstrate the correctness of our implementation. The first scene was dropping three blocks of water onto a plane.



The next scene we show is the canonical dam break.



And finally, a liquified bunny.



IV. Improvements

Due to the large amounts of particles, it is only feasible to simulate small amounts of particles in real time. Offline rendering of certain scenes (to be demoed on presentation day), 1 million particles took about one minute to render per frame.

But for our implementation, it should be possible to simulate and render about 10000 particles in real time (20 fps), however, since we were lazy and did not roll our own rendering code, the rendering portion takes 50% of the cycles. A low hanging fruit would be to run the simulation in a separate thread and double the framerate.

Another difficulty was the large amount of constants and parameters used in the simulation - things like viscosity constants, density constants, mass, kernel radius, etc. had to be meticulously hand tuned to find things that looked physically correct.

References

[1] D.H. House and J.C. Keyser. *Foundations of Physically Based Modeling and Animation*. CRC Press, 2016. ISBN: 9781482234619.

[2] Joe J Monaghan. "Smoothed particle hydrodynamics". In: *Annual review of astronomy and astrophysics* 30.1 (1992), pp. 543–574.

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

[4] Matthias Müller et al. "Particle-based fluid-fluid interaction". In: *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM. 2005, pp. 237–244.