Ice Ice Baby, a Temperature Dependent Water Simulation

Alex Schedel
alexschedel@berkeley.edu

Lucy Lu
zhixinlu@berkeley.edu

Olivia Huang
ohuang@berkeley.edu

Eddy Byun
eddybyun@berkeley.edu

1 ABSTRACT

Simulations are essential to modern computer graphics. They allow for realistic-looking renders in a fully virtual space and add realism to films, video games, and other digital images. Fluid simulations in particular are essential for adding realism and often employ abstractions that enable the simulation of millions of water particles in a fast and easy way. Something that is less commonly discussed, however, is simulating fluids as they transition between states. The computational models that are optimized for fluid simulation may no longer be applicable when that water turns into ice or steam. In this project, we aim to implement a novel particle-based simulation in order to model the state transition of water to ice, in effect "freezing" our simulation in a realistic-looking way.

2 INTRODUCTION AND BACKGROUND

When simulating and rendering fluids, it is often useful to model the physical nature of the fluid into a more abstract representation, such as a fluid grid. By removing the individual particles of the fluid and instead modeling the forces that change large swaths of fluid, grids can act as an acceleration structure, enabling faster rendering for large fluid sims. However, design flexibility is also lost by removing individual particles, meaning that if it is desired to have individual particles of water behaving differently, a more expressive design is required. This problem becomes apparent when simulating state transitions of various fluids. In particular, simulating the transition from water into ice is not easily compatible with a fluid grid model as the colder the water becomes, the more it behaves like a series of particles and the less accurately a grid of forces would be able to capture its behavior. To solve this issue, we present Ice Ice Baby, a temperature-dependent water sim.

Unlike many water simulations, our simulation uses a particle-based model to simulate water. On its own, this is enough to give realistic-looking water behavior. The real reason for this particle-based design however becomes more apparent as we begin to simulate the state transition from water into ice. To simulate this slowing and bunching of water, we leverage a mass and spring-based system, as is common in physical simulations or cloth and other larger particle meshes, to slow and bunch together the water into ice. While this would be difficult to achieve with a fluid grid-based model, our particle-based simulation offers a natural jumping-off point for this extension.

This simulation is built off the architecture provided by CS284a project 4, which implements a point and spring-based cloth system. Additional equations and algorithmic inspiration for the particle-based simulation are inspired by the work of Miles Macklin and Matthias Muller [2] with additional equations inspired by Matthias Müller, David Charypar, and Markus Gross [1]

2.1 Particle Based Water Simulations

Inspired by the work of Matthias Müller, David Charypar, and Markus Gross, we begin by implementing a particle-based water simulation. To begin, we model a mass of water as a series of discrete particles. The size of these particles is adjustable, but even at a relatively large size (say a 50 x 50 x 20 grid) there are enough of them to begin approximating fluids. Each particle has a coordinate and associated mass, which will be updated as we simulate timesteps in our model, causing the simulation to appear to move. The basic implementation of particle-based simulations for physical objects like sand and larger objects like bouncing balls is relatively common and presents few novel ideas. What is novel, however, is how we are able to make the particles themselves simulate water.

2.2 Mass and Spring Systems

Mass and spring systems are often used to model movable meshes, such as cloth and hair simulations. That said, they are rarely applied to fluid simulations, as fluid sims often forego representing individual particles in favor of representing forces and currents throughout a mass of fluid. As mentioned above, we are taking the somewhat unorthodox approach of modeling a fluid through a particle simulation, the primary reason for which being that this model makes it easy for us to extend the implementation with a spring and mass-based system.

3 ICE ICE BABY

3.1 Our Approach

Our simulation consists of two main elements, which shift in importance as we change the temperature of our simulation.

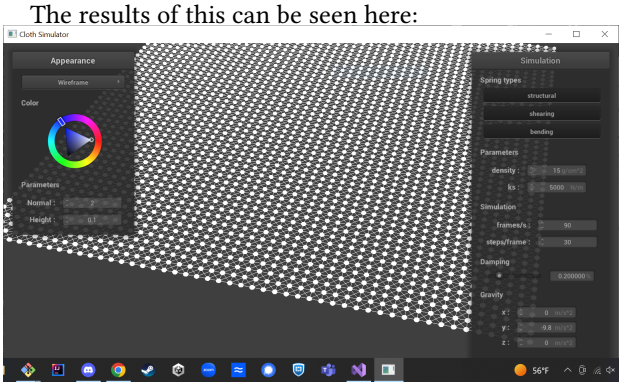
The first of these elements is our particle-based water simulation. Each particle represents a certain amount of water and will alter be rendered to appear continuous with other particles. Water is by nature incomprehensible, meaning that it is essential we constrain these particles to maintain minimum distances from each other, even if large amounts of pressure are pressed upon them. To this end, we implement the Navier-Stokes Equations as well as several gradient and kernel functions to simulate impressibility. These additions alone are enough to begin making the water simulation appear at least somewhat realistic, even without the particles yet being rendered as a continuous amount of realistic water.

The second element is our mass and spring system. The system works by drawing springs between individual particles which exert forces on those particles. Our simulation aims to begin with all particles fully disconnected, meaning there are no springs in the model. As we simulate a temperature decrease and an eventual transition into ice, we problematically add springs and make them increasingly rigid with the ultimate aim of eventually stopping movement altogether once the simulation reaches a desired temperature.

3.1.1 Code Organization. The project itself uses the CS284a cloth simulator as a jumping-off point for implementation.

The first change we make to this implementation is to add logic to render our particles. The original code makes use of 'PointMass'es and is not designed to render them, instead opting to render only the springs between those 'PointMass'es instead. We begin by providing the cloth class with a function, based on the sphere rendering function in sphere.cpp, to render point masses as spheres, thus allowing us to observe our simulation in real-time and assess if it is working properly or not in a purely visual manner.

The results of this can be seen here:



Note that we have transformed the cloth to consist of dozens of spheres. The springs are left in this image to demonstrate what the relationship between the particles might look like when they are fully frozen and thus fully held in place by the springs.

For the purposes of this checkpoint, we have so far only implemented our water simulation, later we will experiment with exactly what parameters we need in our spring simulation, but for now it is sufficient to note that we will be adding springs into the system with a pdf based on the current temperature. The lower the temperature, the more likely we are to generate a new spring at a given time step. The rest lengths and stiffness of these springs will further be decreased as the temperature drops until there is no longer any movement. At this stage, we will have ice.

3.1.2 Kernels and Gradients. In order to realistically simulate fluid dynamics and the movement of water, we used a position correction calculated by Macklin and Muller [2] that allowed for the particles to have a clumping effect and maintain a consistent density. We use the poly6 kernel in order to calculate the density of the particle and smooth out the position in relation to its neighbors, which is defined by the following equation:

$$W(\mathbf{r}, h) = \frac{315}{\pi * h^9} \left\{ \begin{array}{ll} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & otherwise \end{array} \right\} \tag{1}$$

We first calculate the constraint functions and scaling factor for each particle, defined as the following:

$$C_i(p_1 ... p_n) = \frac{\rho_i}{\rho_0} - 1 \tag{2}$$

where

$$\rho_i = \sum_j m_j W(p_i - p_j, h) \tag{3}$$

We iterate over each neighbor j, calculated by storing the neighboring particles in a position-based hash table, called a spatial map. The mass m_j is defined as a uniform mass over the entire mass of the liquid, ρ_0 is the rest density of water approximated as 1, and $h = 1.23$. We then calculate the gradient of the constraint function with respect to the particle p_k as

$$\nabla p_k C_i = \frac{1}{\rho_0} \left\{ \begin{array}{ll} \sum_j \nabla_{p_k} W(p_i - p_j, h) & ifk = i(itself) \\ -\nabla_{p_k} W(p_i - p_j, h) & ifk = j(neighbor) \end{array} \right\} \tag{4}$$

With these two values, we can finally calculate the scaling factor λ_i with offset $\epsilon = 0.2$

$$\lambda_i = -\frac{C_i(p_1 ... p_n)}{\sum_k |\nabla_{p_k} C_i|^2 + \epsilon} \tag{5}$$

We then finally calculated the position correction Δp_i as

$$\Delta p_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(p_i - p_j, h) \tag{6}$$

where the gradient of our kernel was the gradient of the spiky kernel, defined by

$$W(\mathbf{r}, h) = \frac{15}{\pi * h^6} \left\{ \begin{array}{ll} (h-r)^3 & 0 \leq r \leq h \\ 0 & otherwise \end{array} \right\} \quad (7)$$

This Δ_p was applied to the position vector of each particle after calculating the new position in the timestep of our simulation.

3.1.3 Texturing. Not implemented as of the checkpoint. Toward the end of the project, we plan on making the water look fully connected by texturing each particle with a Gaussian blob.

3.1.4 Rendering. Not implemented as of the checkpoint. The simulation is too expensive to run in real-time, so we will look into rendering it asynchronously and producing a gif of the simulation as our final product.

4 RESULTS

Not implemented as of the checkpoint.

4.0.1 Current Testing. To test the particle-based fluid simulation, we want to drop the particles into a container and observe how they interact with each other and the container walls. We created a new collision object: container. This consists of 5 planes to form a rectangular box with an opening at the top. Then, to use the object, we create an input JSON file containing the container and the particles. The particles are placed vertically and dropped into the container when P is pressed.

While testing with the Project 4 base code, the particles properly dropped. However, when we added the new algorithm from the research paper, the particles disappeared when P was pressed. While debugging we found that the gradient of the kernel is incorrectly calculated. Therefore, when the correction vector is added to account for incompressibility, the position is set to an incorrect place. We found that we failed to do a condition check when calculating the gradient of the kernel. However, the current system blows up, so we are in the process of debugging it.

5 FUTURE WORK

Not implemented as of the checkpoint.

REFERENCES

[1] Matthias Müller, David Charypar and Markus Gross. 2003. Particle-Based Fluid Simulation for Interactive Applications. https://matthias-research.github.io/pages/publications/sca03.pdf. [Online; accessed 5-April-2023].

[2] Miles Macklin and Matthias Muller. 2013. Position Based Fluids. https://mmacklin.com/pbf_sig_preprint.pdf. [Online; accessed 5-April-2023].