Parallel Smoothed Particle Hydrodynamics Simulation
15418 Final Project Milestone Report

Haoying Zhang (haoyingz), Qilin Sun (qilins)

December 3, 2023

https://andrew.cmu.edu/user/qilins/15418_project/index.html

1 Updated Timeline

Week	Plan	Assignee
Nov 20 - Nov 26	Background research	Qilin Sun, Haoying Zhang
Nov 27 - Dec 2	Initial CPU implementation Tuning and Debugging	Qilin Sun
Dec 3 - Dec 8	Optimize CPU implementation Complete GPU baseline implementation	Qilin Sun, Haoying Zhang
Dec 9 - Dec 10	Refine CPU version, finish CUDA version	Qilin Sun, Haoying Zhang
Dec 10 - Dec 13	Refine CUDA and Benchmarking	Qilin Sun, Haoying Zhang
Dec 14	Documentation and presentation preparation	Qilin Sun, Haoying Zhang

Table 1: Schedule

2 Current Progress

Currently we have implemented a fully functional sequential CPU implementation of 2D fluid simulation under the Smoothed Particle Hydrodynamics (SPH) methodology. At 400 particles, the simulation is able to run at 30 fps. Right now, the particles can begin with a rectangular tile pattern and move under pressure gradient to reach an equilibrium state where each particle show little motion (check out this video). We are currently implementing an optimization that partitions the particles into a grid of squares whose side lengths equal the smoothing radius. This way, only particles in 9 squares are needed to compute the density at a location.

This blocked optimization will be the baseline to be compared with GPU implementation. We haven't started implementing the GPU version as planned, for the reason that it took us longer to understand how the simulation algorithm works and debug the CPU version. Our goal remains the same for now as the goal is all about performance. We will know whether the speedup goal is too difficult to achieve after we finish a baseline GPU implementation and try some optimization. On the demo day, we want to show both videos of the simulation results and graphs about performance. Overall, we are not aware of any outstanding issues for now. We are on the right track to finalize the CPU version and ready to implement the GPU version.

3 Implementation Overview

Each particle has a position $\mathbf{p}$ and velocity $\mathbf{v}$. To convert the discrete point mass distribution into a continuous distribution of mass, a smoothing function, or a kernal, is used. The kernal

is defined as

$$W(\mathbf{x}, \mathbf{p}) = \begin{cases} (r - \|\mathbf{x} - \mathbf{p}\|)^2 & \text{if } \|\mathbf{x} - \mathbf{p}\| < r \\ 0 & \text{o.w.} \end{cases}$$

where r is the smoothing radius, $\mathbf{p}$ is the position of a particle, and $\mathbf{x}$ is a location we are interested. The smoothing radius r controls the region of influence of a particle; a location that is further than r apart from a particle will not be influenced by it at all. The influence of a particle decays quadratic from its location.

The density at a location is simply defined as the sum of the influence of all particles at that location, i.e.

$$\rho(\mathbf{x}) = \sum_i W(\mathbf{x}, \mathbf{p}_i)$$

where i iterates through all particles. Now we introduce pressure,

$$P(\mathbf{x}) = \alpha(\rho - \rho_{des})$$

where ρ_{des} is the desired density and α serves as a tunable gain that determines how aggressive the pressure is. Intuitively, pressure is proportional to how far the current density is from desired density. A location with high density will have a positive pressure; others will have lower or negative pressures. Note that this pressure is defined for each particle, in constrast to the pressure defined below.

To query a property A, e.g. pressure, at a location, the following approximation is used:

$$\begin{aligned} A(\mathbf{x}) &= \int A(\mathbf{p}_i) W(\mathbf{x}, \mathbf{p}_i) dV \\ &= \int \frac{dm}{\rho(\mathbf{x})} A(\mathbf{p}_i) W(\mathbf{x}, \mathbf{p}_i) \\ &\approx \sum_i \frac{1}{\rho(\mathbf{x})} A(\mathbf{p}_i) W(\mathbf{x}, \mathbf{p}_i) \end{aligned}$$

where m is taken as 1 to simplify the computation. This is reasonable as long as all particles have the identical mass.

Now, to query the pressure, $\mathcal{P}$ at a point, we simply compute

$$\mathcal{P} = \sum_i \frac{1}{\rho(\mathbf{x})} P(\mathbf{p}_i) W(\mathbf{x}, \mathbf{p}_i)$$

Now the central part of the simulation comes in. To make the particles move, there must be a force. The gradient of pressure is used to be this force, and the acceleration on a particle is

$$\mathbf{a}(\mathbf{p}) = \frac{\nabla \mathcal{P}(\mathbf{p})}{\rho(\mathbf{p})}$$

$\mathcal{P}$ differs from P in that it is the weighted sum of pressures at many points.

By the chain rule,

$$\nabla \mathcal{P}(\mathbf{p}) = \sum_i \frac{1}{\rho(\mathbf{x})} P(\mathbf{p}_i) \nabla W(\mathbf{x}, \mathbf{p}_i)$$

The rest is simple: all needed to be done is update the positions and velocities accordingly, just like what happened in Assignment 3/4.

$$\begin{aligned} \mathbf{p}_{t+1} &\leftarrow \mathbf{p}_t + \mathbf{v}_t \Delta t + \frac{1}{2} \mathbf{a}_t \Delta t^2 \\ \mathbf{v}_{t+1} &\leftarrow \mathbf{p}_t + \mathbf{a}_t \Delta t \end{aligned}$$

However, when the above numerical integration is taken, the velocities of particle grow over time, eventually leading to diverging simulation. We suspect this is due to numerical integration error and switched to a second-order Runge-Kutta method. Let $\mathbf{s}_t$ be defined as the state of a particle at time t, i.e.

$$\mathbf{s} = \begin{bmatrix} \mathbf{p} \\ \mathbf{v} \end{bmatrix}$$

Then the Runge-Kutta method is defined as:

$$\begin{aligned} k_1 &\leftarrow f(\mathbf{s}_t) \\ k_2 &\leftarrow f(\mathbf{s}_t + \frac{2}{3}k_1\Delta t) \\ \begin{bmatrix} \tilde{\mathbf{v}} \\ \tilde{\mathbf{a}} \end{bmatrix} &\leftarrow \mathbf{s}_t + (\frac{1}{4}k_1 + \frac{3}{4}k_2)\Delta t \\ \mathbf{s}_{t+1} &\leftarrow \begin{bmatrix} \mathbf{p}_t + \tilde{\mathbf{v}}\Delta t + \frac{1}{2}\tilde{\mathbf{a}}\Delta t^2 \\ \mathbf{v}_{t+1} + \tilde{\mathbf{v}}\Delta t \end{bmatrix} \end{aligned}$$

where f refers to the computation of $\mathbf{v}_t$ and $\mathbf{a}_t$. Because each particle has the property $\mathbf{v}$, it is taken for granted. The acceleration $\mathbf{a}_t$ is computed using weighted pressure and density, as discussed above. Note that $f(\mathbf{s}_t + \frac{2}{3}k_1\Delta t)$ is an expensive step, as it involves stepping the position and velocity of all particles by a time step of $\frac{2}{3}k_1\Delta t$ and recomputing all densities, pressures, and pressure gradients.

After adopting this method, the simulation converges to a point where all particles exhibit little motion. We could have chosen the famous RK4 algorithm instead, but that will take much more time and there is unsuitable for our application.