Parallel Smoothed Particle Hydrodynamics Simulation
15418 Final Project Proposal

Haoying Zhang (haoyingz), Qilin Sun (qilins)

November 13, 2023

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

1 Summary

Fluid dynamics is famous for its mathematical complexity, which is also why computer scientists fascinated about it. In this project, we will model an amount of fluid particles and allow the user to add, disturb, or remove the particles, and reshape the container. We will use CUDA to speed up the simulation.

2 Background

To reassemble realistic fluid, a large amount of particles are created, and they are smoothed to approximate the continuity of real fluids. The technique to do so is known as Smoothed Particle Hydrodynamics, where each point mass is viewed as a continuous distribution, and the property of the fluid at any location will be a weighted sum of nearby particles' properties. The further the particle is from the location, the lower its contribution is.

The motion of individual particles is driven by the pressure at its location, which points in the direction of decreasing density, and whose magnitude is proportional to the different between the current density and the desired density (defined to be the average density of all particles over the space). This is based on the intuition that the system tends to transition to a state of uniform pressure.

Because computations are always performed on a point-by-point basis, the simulation process is highly parallelizable. In addition, a source of inefficiency is the same one encountered by Assignment 3/4, which is not every particle needs to be considered for every other particle. This process can therefore be sped up. In addition, on GPU the properties of a batch of particles may be stored on the shared memory, which could also enhance performance.

3 The challenge

In a naive implementation, every particle needs to access every other particle's properties. However, particles that are outside of the domain of influence do not need to be considered. That is, methods similar to blocking would be necessary to significantly enhance the performance. But because particles move around from iteration to iteration, blocks may need to be redrawn. Due to this issue, locality may be low and execution may be divergent, so other more sophisticated methods may need to be used, such as this. Through this project, we hope to learn effective partitioning of data on a GPU.

Constraints: Frequent IO is detrimental to GPU performance, so communication of information needs to be handled wisely. In addition, because particles are not uniformly distributed, a uniform grid assignment will not give balanced loads.

4 Resources

Code will be run on an AMD Ryzen 5800 CPU and RTX 3080 GPU. We will start from this video and write our own code in C++. Other resources include this article from Nvidia discussing a way to split particles into groups and this paper introducing a way to maintain particle consistency. We will not need any special machines for this project.

5 Goals and Deliverables

Goals:

- First implement a sequential CPU version, then a CUDA version.
- Make sure that edge cases like particle collision are handled correctly.
- Compare the performance of CPU and GPU implementations.

Hope to achieve:

- If it goes well, achieve at least $4\times$ speedup from the initial version of GPU implementation.
- If it doesn't, $2\times$ speedup is enough.

Demo:

- Play a recorded animation of the simulation of at least 10k particles in a space, for both CPU and GPU implementations. Show the frame rates.

6 Platform Choice

We will first implement a C++ version on CPU, then a CUDA version on GPU. Because the simulation is computation-intensive, GPU would be a good choice, but only if the implementation is wise enough. Besides, because GPU has much higher bandwidth, it would be more suitable for such a task where the amount of particles is large.

7 Schedule

Week	Plan
Nov. 13 - 19	Background research and initial code-up
Nov. 20 - 26	Refine CPU version, finish CUDA version
Nov. 27 - Dec. 3	Refine CUDA, benchmarking
Nov. 4 - Dec. 10	Documentation and presentation preparation

Table 1: Schedule