15418 Project Proposal

GPU Fluid Simulation

Logan Kojiro and Oscar A Ramirez Poulat

URL

https://oadrian.github.io/GPUFluidSimulator/

Summary

We are going to implement a fluid simulator using Smoothed Particle Hydrodynamics on a CPU and a GPU and compare the relative performance between the two. In addition to the simulation we will render the fluid using an efficient fluid surface rendering technique.

Background

There are two main approaches to fluid simulation we are looking at for this project: grid based simulation, and particle based simulation. Initially we will be implementing a smoothed-particle hydrodynamics fluid simulation. Though the grid based algorithms have a higher numerical accuracy, particle simulations are faster because they use neighboring particles to compute pressure rather than solving systems of linear equations and mass conservation is better and more intuitive because each particle has its own mass.

The general approach of the algorithm is to calculate the contribution to acceleration of each particle from 1) the pressure from its neighbors, 2) the force of gravity, and 3) any external forces. For each time step, first calculate the density at each particle with the equation:

$$\rho_i = \sum_j m_j W(r_i - r_j, h)$$

Where mj refers to the mass of the particle at position rj and W(d, h) is the smoothing kernel with core radius h. The Gaussian Kernel function is

$$W(d) = \frac{1}{\pi^{\frac{3}{2}} h^3} exp(\frac{r^2}{h^2})$$

Particles further than distance 2h are not counted as affecting the particle being calculated. This offers opportunity for parallelism because it means that parts of the fluid that are further away from each other are not dependent on each other and can be calculated concurrently. After finding the density, we find the force of the pressure with the following equation:

$$f_i^{pressure} = -\sum_j \frac{m_j}{\rho_j} \frac{(p_i - p_j)}{2} \nabla W(r - r_j, h)$$

Once we know the force on each particle from pressure, we can add it to the force of gravity, calculate the overall acceleration and update positions accordingly. With a huge number of

particles necessary for an accurate simulation, we can parallelize over nearby groups of particles. One thing we will need to be mindful of is, similar to the Barnes-Hut simulation, is the changes in position overtime will necessitate the periodic updating of the particle groups.

The Challenge

One of the biggest challenges in implementing this will be keeping track of the shifting dependencies. To start out, it is easy to group particles that will depend on each other together to run in blocks that can share memory. But as the simulation progresses these particles will disperse throughout the fluid. For a perfect simulation, each particle would depend on all other particles, but we can get a good enough approximation by only counting particles that are within a certain radius.

Ideally we want this to run in realtime, so we are time constrained. We also want to compute the most accurate simulation possible so we want to maximize the number of particles we can calculate in each minimum time step.

Another difficulty with this project is understanding the mathematical background for the particle interactions as it involves advanced vector calculus and a considerable amount of physics.

Resources

We are planning on developing this simulator on our personal Windows PCs. An interesting aspect of this project is that we will be able to compare the relative performance between CPU and GPU implementations on 3 distinct sets of hardware. We have access to the following machines:

1. Machine 1

CPU	AMD Ryzen 7 3700x 8-Core Processor @ 3.59 GHz
GPU	NVIDIA Geforce RTX 3080 (10 GB, 1.71 Ghz, 8704 CCs)
RAM	16 GB

2. Machine 2

CPU	Intel Core i7-7700HQ 4-Core Processor @ 2.8 GHz
GPU	NVIDIA Geforce GTX 1050Ti (4 GB, 1.29 Ghz, 768 CCs)
RAM	16 GB

3. Machine 3

CPU	AMD Phenom II X6 1045T 6-Core Processor @2.70GHz
GPU	NVIDIA Geforce GTX 970 (4 GB, 1.05 Ghz, 1664 CCs)
RAM	8 GB

Additionally, depending on the portability to the ghc machines we might be able to get an additional data point in the form of an RTX 2080 which would give a nice overview of the relative performance across the last 4 generations of NVIDIA GPUs.

We will be using the following papers as a guide for our implementation:

Goswami, Prashant, et. al. " Interactive SPH Simulation and Rendering on the GPU". Eurographics/ACM SIGGRAPH Symposium on Computer Animation. 2010.
link: https://www.ifi.uzh.ch/vmml/publications/interactive-sph/InteractiveSPH.pdf

Colin Braley, Adrian Sandu. "Fluid Simulation For Computer Graphics: A Tutorial in Grid Based and Particle Methods". Virginia Tech.
link: https://cg.informatik.uni-freiburg.de/intern/seminar/gridFluids_fluid-EulerParticle.pdf

We will be using CUDA and OpenGL examples from:
https://docs.nvidia.com/cuda/cuda-samples/index.html#graphics

Goals and Deliverables

The simplest goal that we would like to achieve is two implementations of our chosen algorithm, one on a CPU and one on a GPU. As a stretch goal, we would implement both a grid-based and a particle based simulation on both a GPU and CPU. As a fallback goal, we would either choose CPU or GPU, or only implement the simulation in 2D instead of 3D. For any of these we will also need to implement a way to render the fluid we're simulating. For fluid rendering we can either go for a more challenging full fluid surface rendering technique as outlined in the paper by Goswami, et. al, or we could render the individual particles using a 3D renderer framework like OpenGL. Our goal is to have full fluid surface rendering but we have the OpenGL solution as fallback. Depending on the performance, we will either demo it in real time, or show a pre-rendered video. In summary:

FORCE TO ACHIEVE (80%)

- 2D CPU sequential SPH solution
- 2D GPU SPH solution for particle interaction simulation using an algorithm such as the one described in Goswami, et. al.
- OpenGL rendering of particles

PLAN TO ACHIEVE (100%)

- 3D CPU sequential SPH solution
- 3D GPU SPH solution for particle interaction simulation using an algorithm such as the one described in Goswami, et. al.
- OpenGL rendering of particles

HOPE TO ACHIEVE (120%)

- 3D CPU parallel (threads, AVX, ISPC) SPH solution
- 3D GPU SPH solution for particle interaction simulation using an algorithm such as the one described in Goswami, et. al.
- Fluid surface rendering technique, as found in the same paper, or similar.

We hope to have a real time simulation for our demonstration. As a fallback we could have a pre rendered scene if we don't reach real time. Worst case scenario have some images of our simulation. Our demo will be inspired by the following clip around 2:50 in the following video: https://youtu.be/ureGelZPi3o?t=170

We will naturally not have the same level of realism but at a high level it's what we are striving for. We will also have graphs comparing the relative performance of the same algorithm running on a CPU and GPU on 3 different machines to illustrate how scalable our solution is.

Platform Choice

We will be using C++, CUDA and OpenGL to develop a project targeted at a Windows 10 pc running on an x86-64 processor. OpenGL will be a fallback in case we are unable to get the fluid surface rendering done in cuda then we would use OpenGL to render individual particles as opposed to the whole fluid surface. We chose CUDA because we have some exposure to it from A2 and since both of our machines have NVIDIA GPUs it was the natural choice. In general we chose a GPU as our parallel solution because SPH is a very data parallel problem that can be efficiently solved using GPUs since the bottleneck computation is calculating a particles interactions with surrounding particles and this lends itself naturally to the cuda thread model.

As mentioned above, in a perfect scenario we will be targeting all 3 GPUs listed above, but if issues arise then at the very least we will have our simulator running on one of these machines.

Schedule

Week	TODO
1 (11/04-11/11)	Initial project bring up: including through reading of SPH algorithms, environment setup across all 3 machines to ensure CUDA compatibility, design solution architecture for CPU and GPU, start CPU implementation

2 (11/11-11/18)	Finish Basic CPU implementation, setup basic rendering pipeline using OpenGL to get some visual representation, Begin GPU solution
3 (11/18-11/25)	For demo have at the very least a CPU implementation + a designed GPU solution
4 (11/25-12/02)	Finish up GPU solution using OpenGL rendering. Begin fluid surface solution
5 (12/02-12/09)	Finish fluid surface rendering
6 (12/09-12/16)	Performance testing, data gathering. Prepare the final report and presentation. Prepare demo