Brendan Miller
BPM 665
CS 378H

Project Overview

sph_compute is an interactive Lagrangian fluid simulation using Smoothed Particle Hydrodynamics. The SPH calculations are done on GPU using compute shaders with spatial acceleration done on CPU.

Theory

Particle updates are calculated in the following steps: compute density and pressure at each particle, compute normals at each particle, compute forces at each particle, integrate the acceleration using verlet integration. SPH relies on the fact that a scalar field at a particle can be given by a weighted sum of its surrounding particles. To compute density at each particle, we take a sum of weighted masses at each particle within a smoothing radius *h*. The weight of each neighbor particle is given by a kernel[1] function, in this case the poly6 kernel, giving closer particles a higher weight.

$$\rho_i = \sum_j m_j poly6(\overrightarrow{r_i} - \overrightarrow{r_j}, h)$$

Pressure is computed by the (slightly modified) Tate equation by default or with the ideal gas law (by pressing *l* to toggle pressure computation type). The modified tate equation allows for more realistic, smooth flow as opposed to more rigid particle dynamics using the ideal gas law.

$$p_i = B((\frac{\rho_i}{\rho_0})^3 - 1) \text{ OR } p_i = k(\rho_i - \rho_0)$$

Computing normals is done based on a weighted gradient of a particles neighbor particles, similarly to computing density:

$$\overrightarrow{n}_i = h \sum_j \frac{m_j}{\rho_j} \nabla poly6(\overrightarrow{r_i} - \overrightarrow{r_j}, h)$$

The force acting on a given particle is $f = f_{pressure} + f_{viscosity} + f_{surface} + f_{external}$.

$$\overrightarrow{f}^i_{pressure} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla spiky(\overrightarrow{r_i} - \overrightarrow{r_j}, h)$$

$$\overrightarrow{f}^i_{viscosity} = \mu \sum_j m_j \frac{\overrightarrow{v}_i - \overrightarrow{v_j}}{\rho_j} \nabla^2 viscosity(\overrightarrow{r_i} - \overrightarrow{r_j}, h)$$

$$\overrightarrow{f}^i_{external} = \overrightarrow{g}$$

$$\overrightarrow{f}^i_{surface} = -\gamma \sum_j \frac{2\rho_0}{\rho_i + \rho_j} (m_i m_j C(|\overrightarrow{r}_i - \overrightarrow{r}_j|) \frac{\overrightarrow{r}_i - \overrightarrow{r}_j}{|\overrightarrow{r}_i - \overrightarrow{r}_j|}) + m_i(\overrightarrow{n}_i - \overrightarrow{n}_j))$$

[1] Kernel functions and C spline function are included in the appendix

We chose to use Verlet integration to integrate the velocities, specifically with the velocity Verlet algorithm. We use this method instead of Euler integration as it is stable for solving second order differential equations, such as rigid motion of individual particles.

Implementation

There is a simulation pipeline of 4 compute shaders (sph_dens_pres.glsl, sph_norm_vel.glsl, sph_force.glsl, sph_integrate.glsl) which process the fluid simulation steps. They are divided as such since each of these steps can be parallelized themselves, but the four must run sequentially. The compute shaders are given a Particle vector SSBO and a lookup table SSBO for acceleration. Each of the first three shaders loops through the 27 spatial hashes adjacent to the current particle, selecting all particles that have those hashes, and adding them into the sum if their distance from the current particle is less than h. The last shader performs verlet integration and checks for collisions with the container or with the pipe outflow (if the user enables the pipe).

Each particle is rendered as an instanced sphere. Particle color is determined based on its physical properties (dependant of user-settable coloring modes). Screen Space Ambient Occlusion was implemented with shader code from github, cited in the references section, slightly modified to suit this application. SSAO was the only additional shading added to this project, as I wanted the particle shading to encode useful information.

Performance Optimizations

To reduce adjacent particle finding from $O(N^2)$ to $O(MN)$ complexity, we give each particle a spatial hash, computed by a bit-and of the products of each position (relative to a spatial grid) component with a unique, large prime number. We then sort the list of particles based on their hash, and then create a lookup table which when given a hash, returns the first index a particle of that same hash is found in the particle list. Since our particles are sorted by hash, we iterate after that particle until we hit a particle with a different hash. When this new hash is found, the loop breaks. This idea is loosely borrowed from the paper *Interactive SPH Simulation…*, with the significant differences that I do not have access to a built in parallel GPU sort, and that my implementation uses spatial hashing instead of z-indexing.

Sorting and hash table creation are done on CPU, to make use of a well optimized std::sort function. Potential future optimizations might include creating parallel radix sort in a compute shader to avoid copying back memory from GPU to CPU. Even with this memory transfer, the GPU accelerated version significantly outperforms my CPU only version.

Using the program

Build and usage instructions are in *README.md*. Performance on the lab machines is acceptably interactive, but performance on more modern devices is smooth (60fps on a 2018 laptop with 10k particles and 1 physical simulation step per frame). As a side note, I personally recommend color mode 8, which colors based on the velocity vector.

References

P. Goswami, P. Schlegel, B. Solenthaler, R. Pajarola, *Interactive SPH Simulation and Rendering on the GPU*, https://people.inf.ethz.ch/~sobarbar/papers/Sol10b/Sol10b.pdf

Zhang Jin Quan, *Real-time SPH fluid simulation,* https://drive.google.com/file/d/0B2macuhZeO18TzQ0UmNsRUt6bUE/view

Spite, *Wagner*, https://github.com/spite/Wagner/blob/master/fragment-shaders/ssao-simple-fs.glsl

Appendix

Table of Kernel functions and C spline function

$$W_{poly6}(\vec{\boldsymbol{r}},h)=\frac{315}{64\pi h^9}\begin{cases}(h^2-|\vec{\boldsymbol{r}}|^2)^3, & 0\le|\vec{\boldsymbol{r}}|\le h\\ 0, & \text{otherwise}\end{cases}$$

$$\nabla W_{poly6}(\vec{\boldsymbol{r}},h)=\frac{945}{32\pi h^9}\begin{cases}\vec{\boldsymbol{r}}(h^2-|\vec{\boldsymbol{r}}|^2)^2, & 0\le|\vec{\boldsymbol{r}}|\le h\\ 0, & \text{otherwise}\end{cases}$$

$$\nabla W_{spiky}(\vec{\boldsymbol{r}},h)=-\frac{45}{\pi h^6}\begin{cases}\frac{\vec{\boldsymbol{r}}}{|\vec{\boldsymbol{r}}|}(h-|\vec{\boldsymbol{r}}|)^2, & 0\le|\vec{\boldsymbol{r}}|\le h\\ 0, & \text{otherwise}\end{cases}$$

$$\nabla^2 W_{viscosity}(\vec{\boldsymbol{r}},h)=\frac{45}{\pi h^6}\begin{cases}(h-|\vec{\boldsymbol{r}}|), & 0\le|\vec{\boldsymbol{r}}|\le h\\ 0, & \text{otherwise}\end{cases}$$

$$C(r)=\frac{32}{\pi h^9}\begin{cases}(h-r)^3r^3, & 2r>h\wedge r\le h\\ 2(h-r)^3r^3-\frac{h^6}{64}, & r>0\wedge 2r\le h\\ 0, & \text{otherwise}\end{cases}$$