Final Project Progress Report

Preliminary Movie

Included in the submitted zip file are two movies, each of which is a simulation of 15,000 particles interacting gravitationally. While the initial position distribution is gaussian in both simulations, the initial velocities are zero in one and rotational in the other. Each frame took approximately 1 second to render.

Effort Breakdown

Thus far, I have spent most of my time handling graphics using OpenGL, implementing the base classes, and writing code to calculate gravitational forces and update positions/velocities. I currently render each particle as a tiny sphere, which generates decent visual results that are more suggestive of a star cluster than a gas cloud. I plan on reworking this so that I instead "smear" the particles out to create a smooth density field, reflecting the gaseous nature of the particle system. Moreover, I spent a significant amount of time attempting to implement dynamic camera perspective so that the camera could orbit the particle system in 3D. However, this proved difficult given my lack of familiarity with OpenGL, and I will only return to it once I have finished all other key aspects of the project.

Aside from such rendering difficulties, I have spent most (~50%) of my time creating the Barnes-Hut tree class and writing code that calculates per-particle gravitational forces using said tree. While the brute-force gravitational force calculation was trivial, implementing the BH tree took some time, requiring pointer manipulation and recursive algorithms. Luckily, I am fairly sure that I can use the tree not only for gravitational force calculation but also to find nearest neighbors for density/pressure SPH calculations.

Updating the position and velocity of each particle was perhaps the easiest aspect of the project so far, as leapfrog integration is a straightforward technique similar to those that we have seen in class. Multiple papers have asserted that leapfrog integration is the most accurate time integration technique for SPH simulations involved self-gravity because it conserves energy.

With self-gravity and position/velocity update fully implemented aside from parameter tuning, I can already generate compelling visual results by simply varying initial parameters (i.e., position and velocity distributions). The biggest issue with the simulation at this point is the lack of viscosity and damping due to fluid interaction; the particles accelerate towards the system's center-of-mass and pass right through it, causing a spray of particles to emanate from the cloud. While this may be a problem of tuning gravitational interaction strength, I suspect it is simply due to the lack of hydrodynamical forces between nearby particles. I plan on spending most of my remaining time implementing these forces.

Revised Timeline

I have completed all the goals to date from the original timeline, with the caveat that I implemented brute-force gravity and Barnes-Hut tree gravity *before* implementing hydrodynamical forces. I proceeded in this order because gravity provided me with an easy-to-evaluate visual checkpoint, helping me ensure that time integration and other aspects of the project were implemented correctly. With that in mind, my revised timeline is below:

Week of 12/2

- Write code to find nearest neighbors of a given particle using BH tree
    - Use this to calculate density and pressure for each particle
- Include force due to pressure in leapfrog integration
    - The exact expression for the force depends on the gas equation of state:
        - Adiabatic: easiest to implement → *start with this*
        - Non-adiabatic: will likely produce more interesting/chaotic visual effects
- Many papers say that the smoothing length of each particle should be varied at each timestep so that the number of neighbors is constant:
    - Test whether the code implemented above *without* this extra step produces good visual results
    - If not, add smoothing_length to the PARTICLE class and write code to update the smoothing length such that the number of neighbors is constant

Week of 12/9

- With self-gravity and non-adiabatic hydrodynamical forces implemented, tune parameters (e.g., G, gas constants, particle mass, etc.) for optimal visual results
- Implement non-adiabatic hydrodynamical forces
    - This requires storing and updating thermal energy per particle
- Adapt code from HW2 to generate density field from particle system
- (Optional) Use thermal energy from SPH calculations to color each particle based on temperature