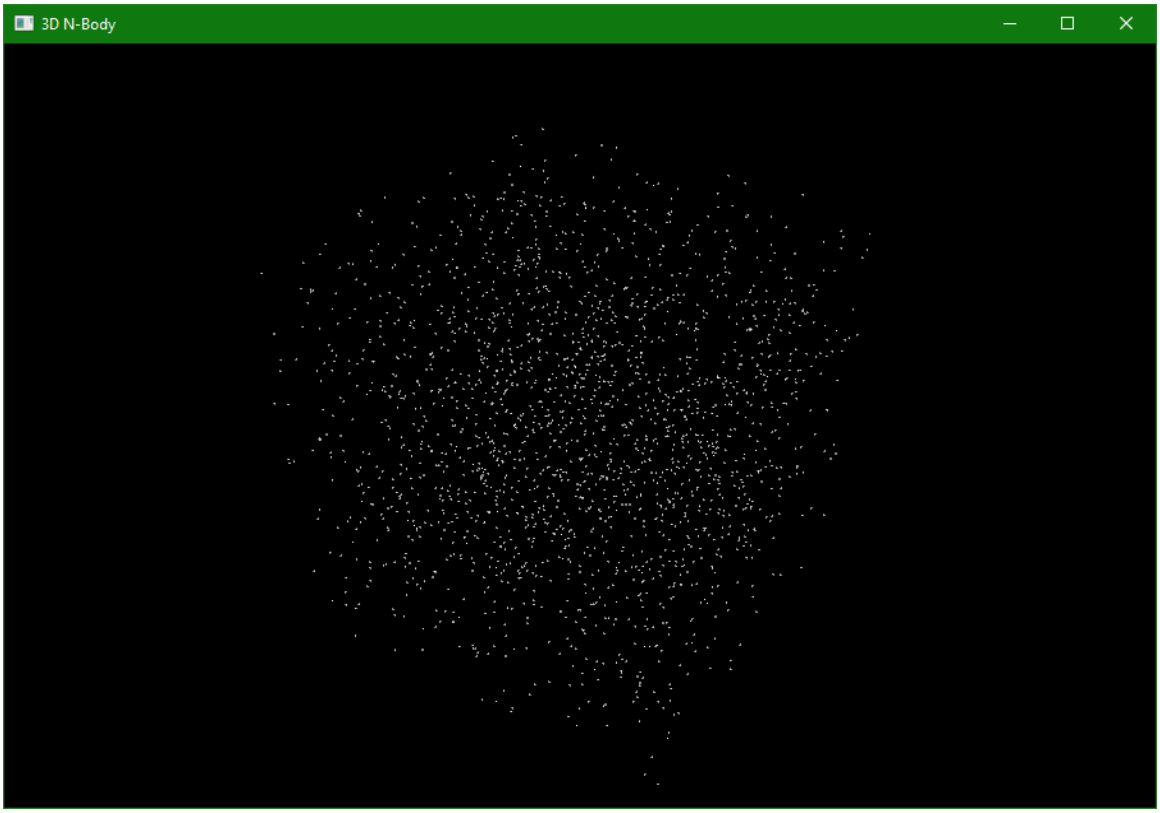
Logan Terry

CS6610 Final Report

Progress Report

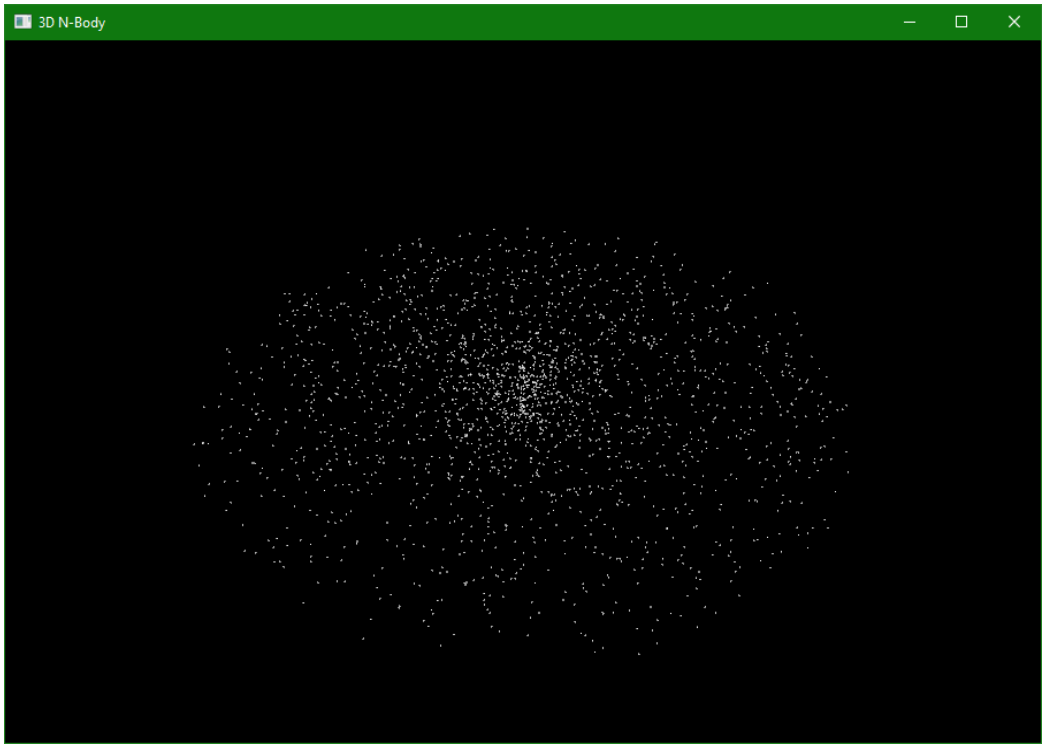
While my previous assignments have quite a bit of good helper functions, they also have a bunch of leftover code from adapting from assignment to assignment. Because of that I'm going to be starting with an empty main file and just copy over the helper methods I need.

Because we have to have pictures in this report, and because we recently used a geometry shader, I thought it would be easiest to start with those parts of the project. I started by randomly generating points, and rendering the particles as I specified in the proposal. I started by copying over the needed helper methods. After that I started by randomly generating positions inside a 20,20,20 cube for the particles, and rendering them as points.

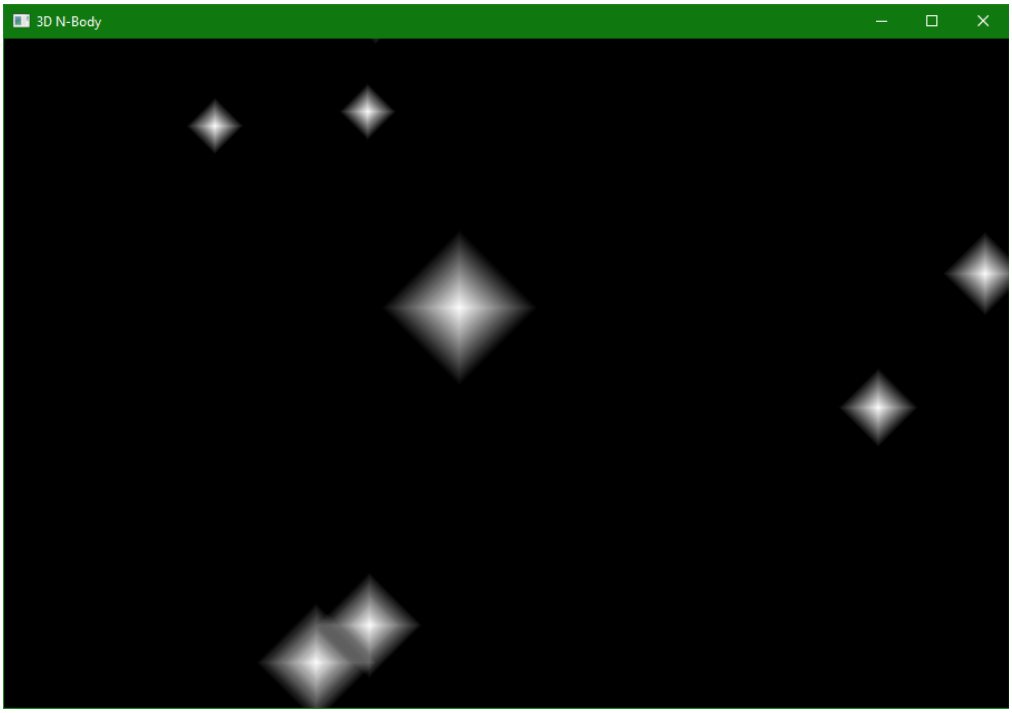


For fun I made the generation of points in a cube a function, and also set up one for a cylinder. I think that one will make for a more fun simulation. I may try a toroid later.

Logan Terry

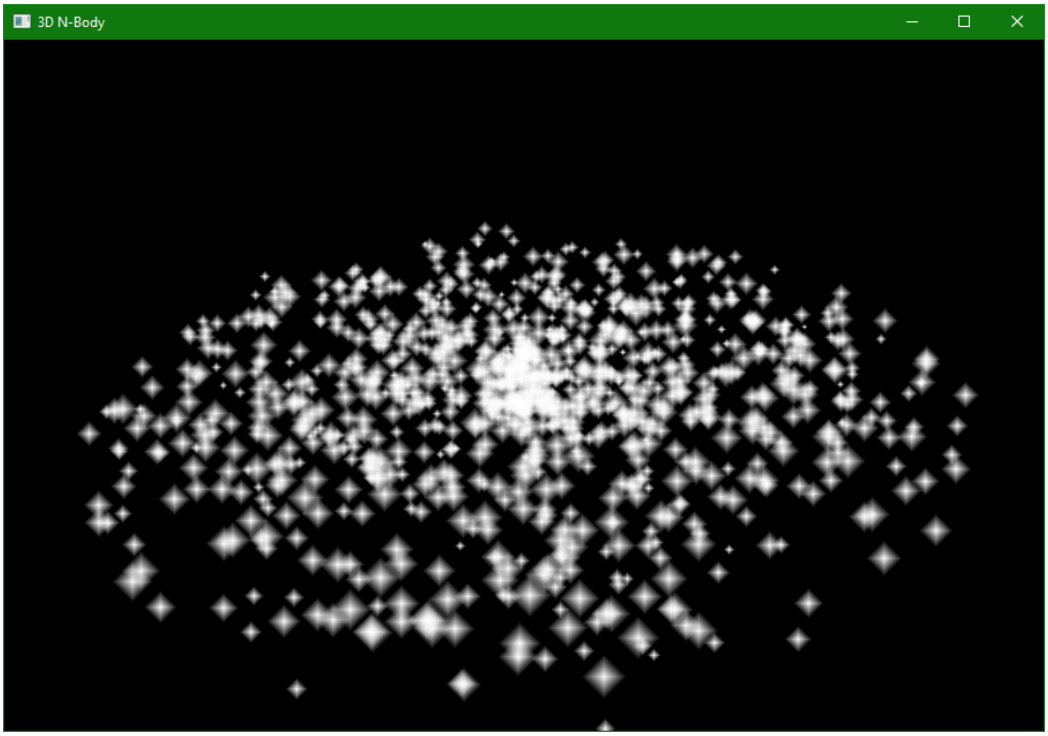


I then got the geometry shader set up such that I was able to create a 4 pointed star with transparency that would always face the camera. The centerpoint has a 1 alpha value while the points have 0 alpha.

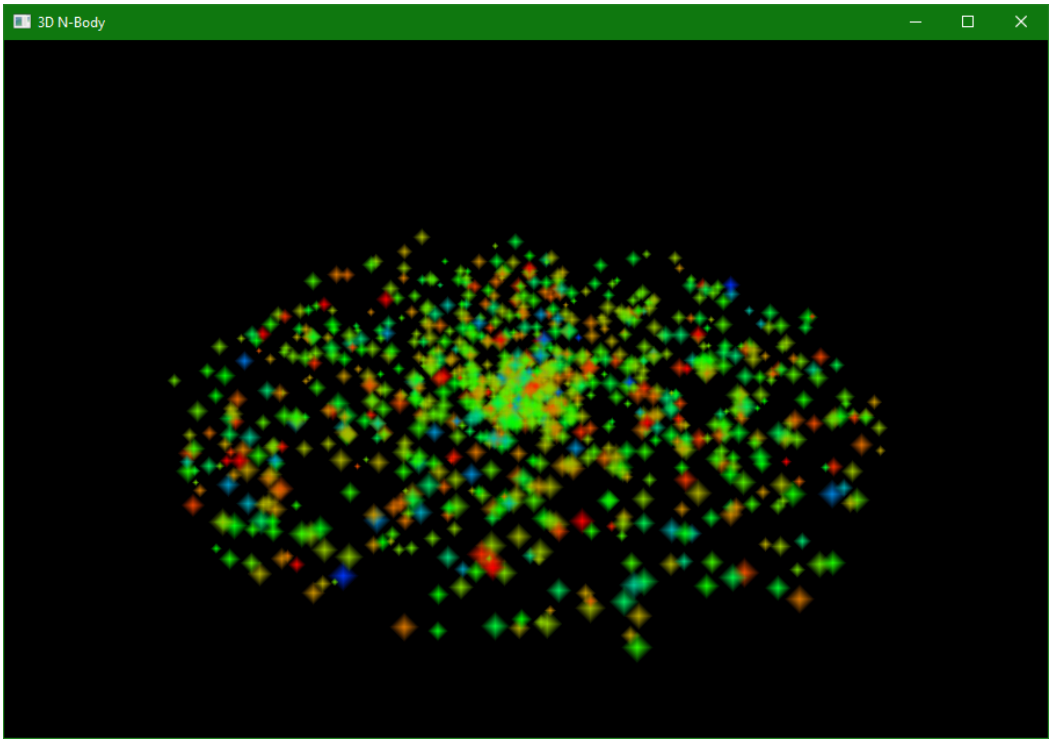


From there I thought It would be fun to set up different masses, and compute the size based on that.

Logan Terry

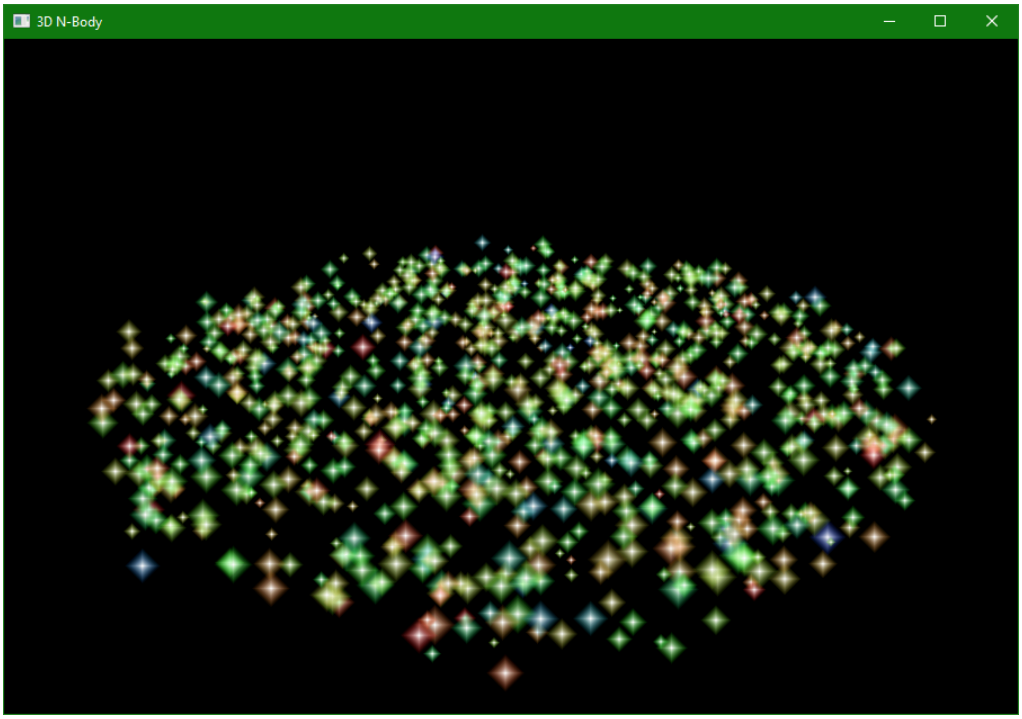


Then I set their color based on velocity. Beyond a certain velocity the color is clamped to red. Almost no velocity is blue. For now I'm setting their velocity to some random value, this will later be evaluated in the compute shader.

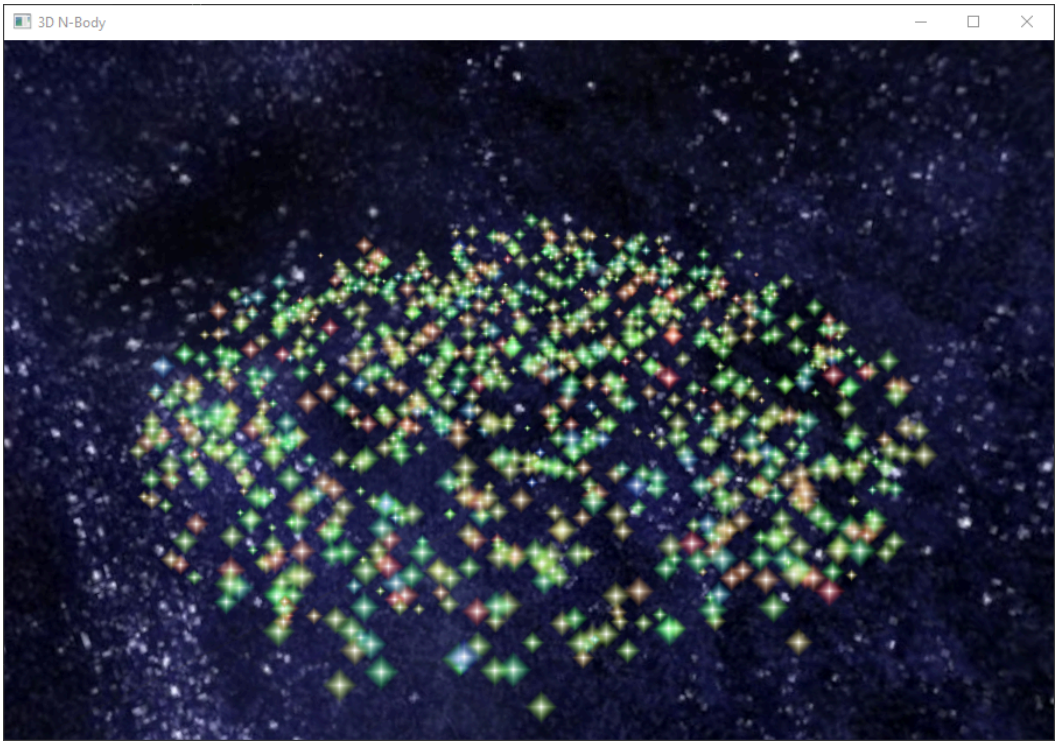


Logan Terry

At this point I didn’t like how the points were quite dense at the center of the disk, so I changed how that was computed so that it’s evenly distributed. I also set the centers of the stars to be white with the edges colored, as I think that looks better.



From here I needed to add the cubemap. Most of the code was copied from assignment 6 for this. I believe I found a good cube map that still contrasts with the particles. The only downside is it’s not a great resolution, but it’s also ~8MB so I’m fine with it.



Logan Terry

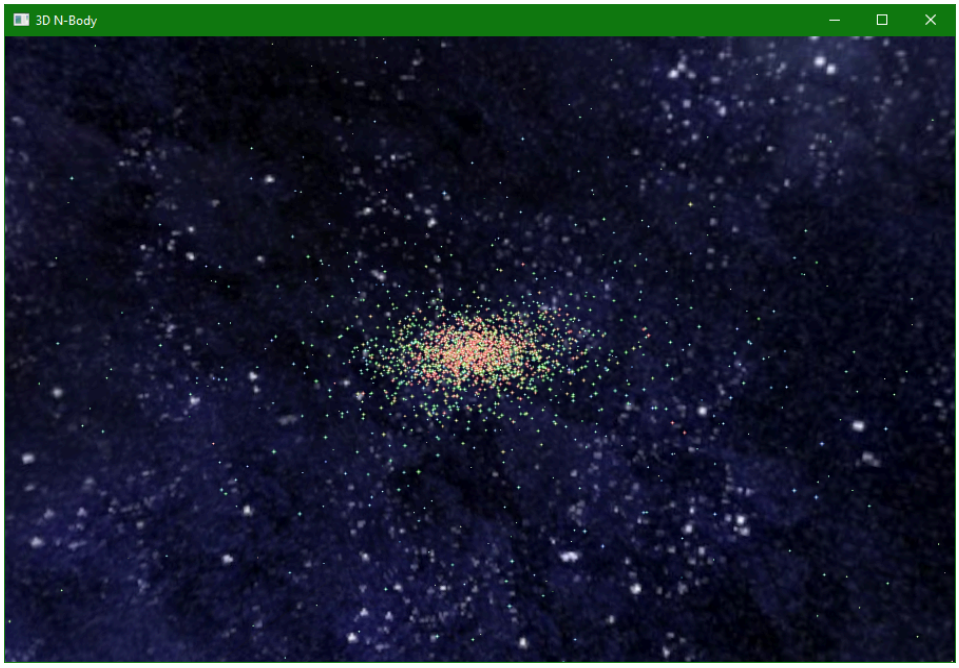
Final Report

I started by writing a basic compute shader that just adds to the x position of the particles to test that it works. As I was doing this, I realized that I needed to change from using a list of positions, velocities, and mass to a list of particle structs. Turning it into structs wasn't difficult, but getting the SSBO into the vertex shader was a tremendous headache, as I didn't know that the padding of the struct in c++ doesn't match the expected padding in the shader. From what I've seen, it looks like glsl expects the struct to be a multiple of 16 bytes in order to line up.

I also spent a significant amount of time trying to figure out why my compute shader didn't appear to affect my ssbo's. It turns out I was compiling it as a vertex shader instead of a compute shader. I am only human after all.

After I had that working, I made it so that the compute shader is only used if the simulation isn't paused (pause and unpause with spacebar), and swap the buffer pointers every update step so positions are constantly updating.

Then it was just a matter of setting up the gravity code in the compute shader. This confused me for a bit, as computing the radius squared was easiest to do with dot, but I was assigning the result to a double which seemed to cause some sort of shader crash, but not a crash to the overall program. Beyond that it was just tweaking variables to get them where I like them. I also set it up to reset positions and velocities with "r".



The controls are the same as previous assignments where. Left mouse drag rotates around the starfield. Right click and drag zooms in/out. Space starts and stops the simulation, R resets to the initial state. It was built on Windows with vsCode and MinGW. If you use vsCode and MinGW, I've provided a launch script for you. If not, you will likely need to replace the libraries in lib and in the root directory (GLFW, GLEW) before compiling.