Trevor Stahl

Project 7A

6/7/19

CS475e Spring 2019

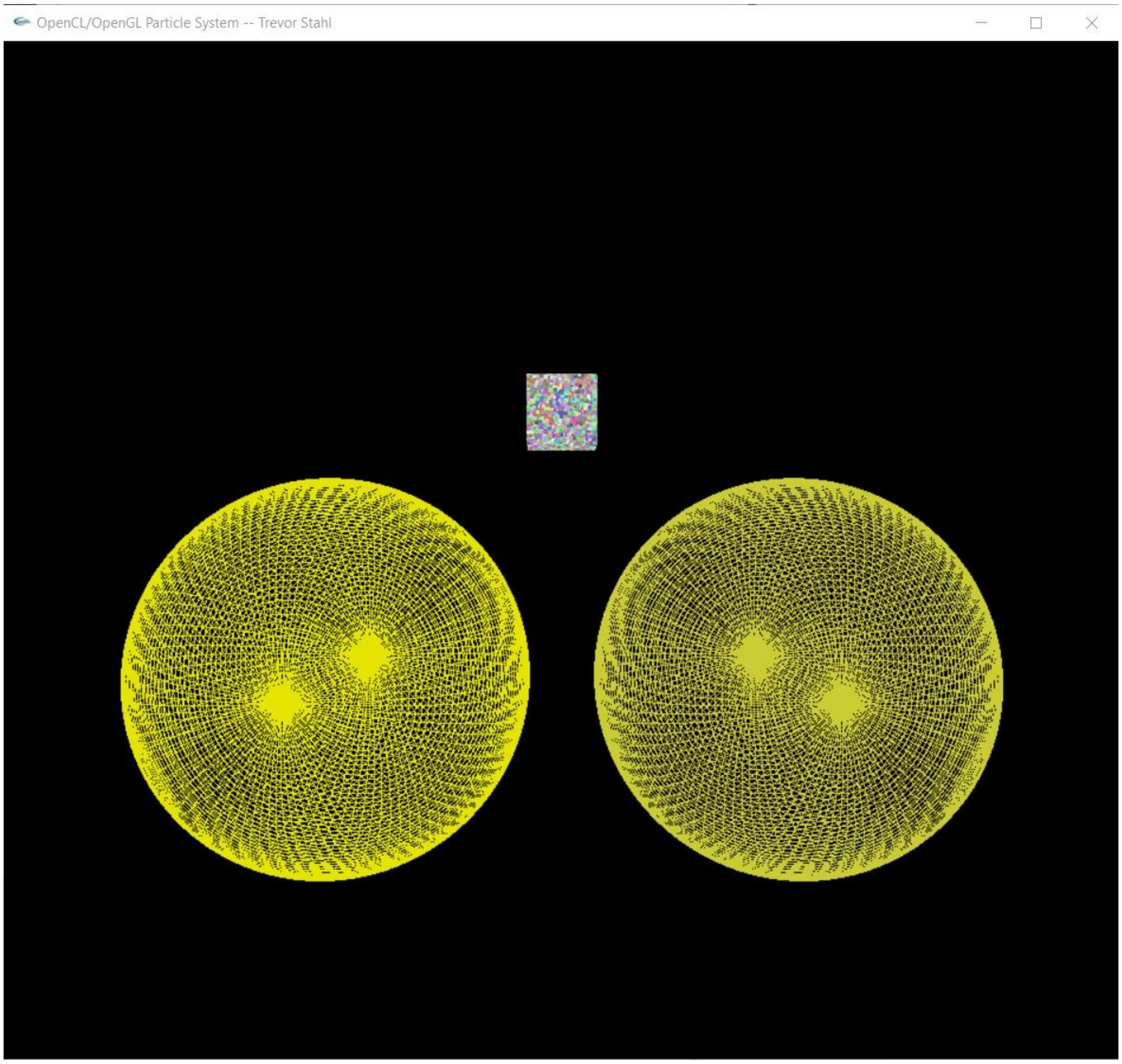
Professor Bailey

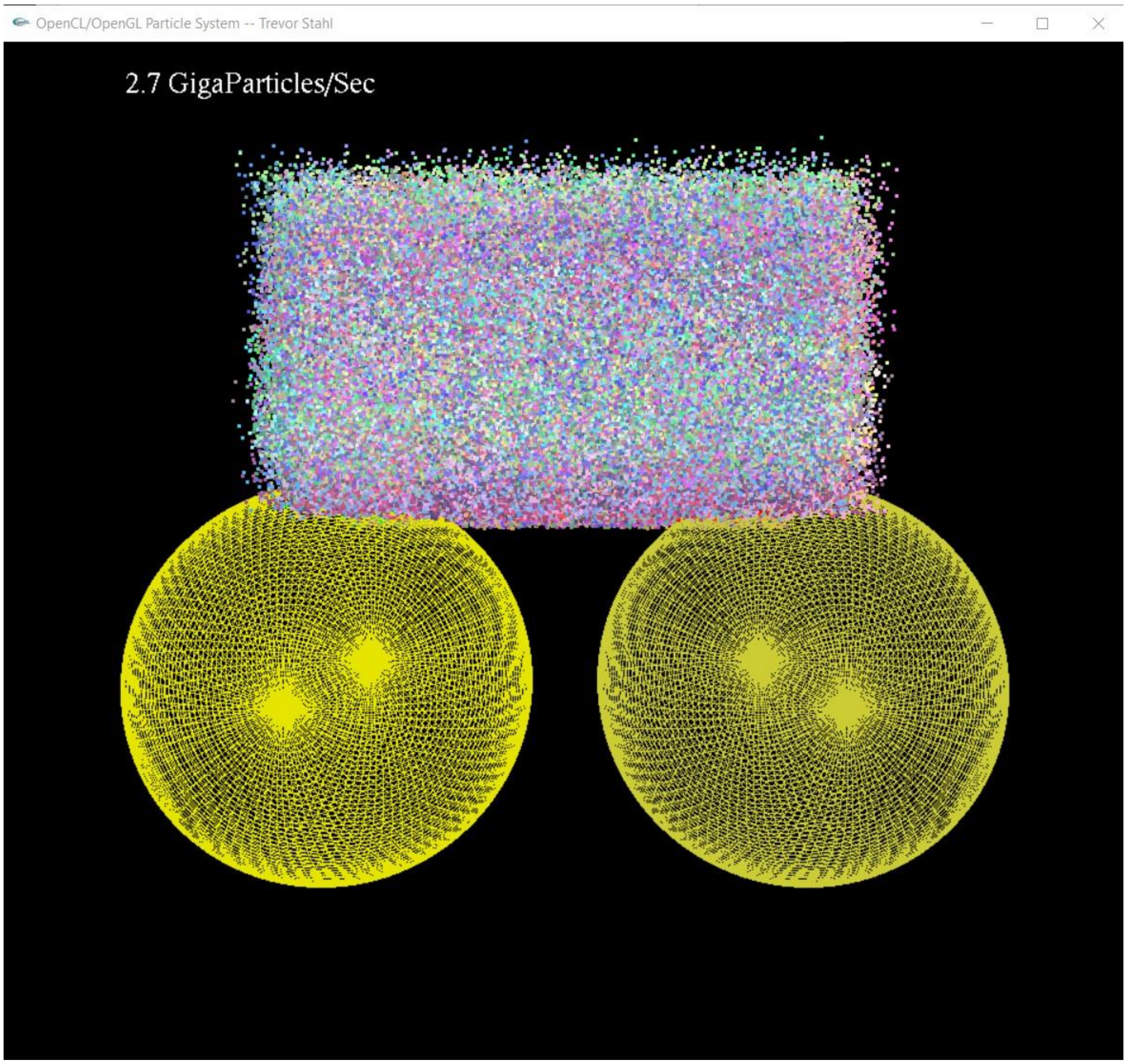
Link to video: https://media.oregonstate.edu/media/t/0_uqcig1jv

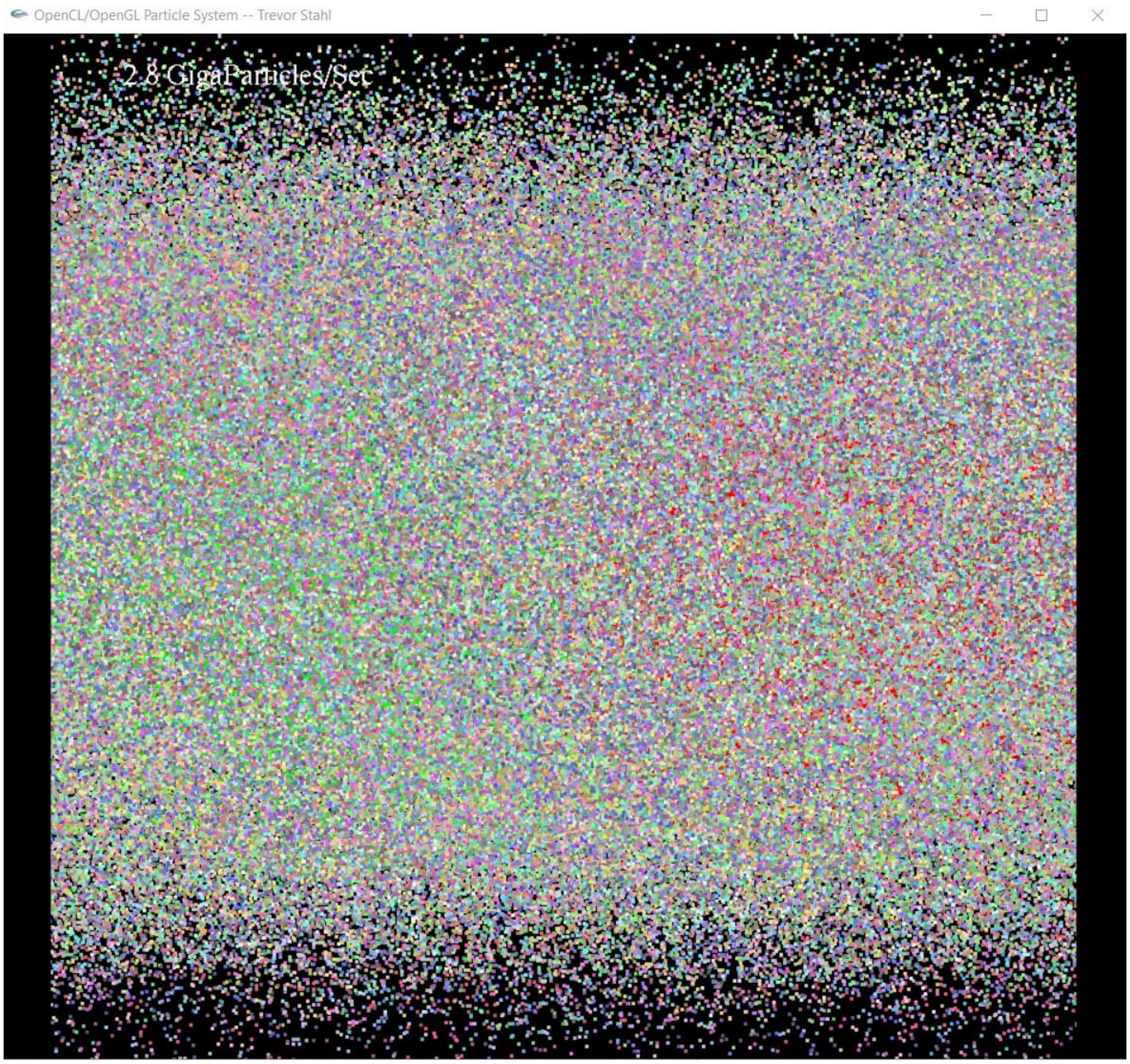
I ran my code on my personal Windows 10 machine through visual studio 2017. I have an AMD 2700x, X470 Mobo, 16 Gigabytes RAM, and a 1080ti. I had to retarget solution to run on my computer. I tried to not change line endings when prompted by visual studio. I did not change r to rb within the fopen() parameters.

The dynamic thing I had my particles do was: start out with the random colors, I had two spheres symmetric around the center of the initial mass of particles but also below them and spaced such that only some of the particles will drop down between them without hitting a sphere. When the particles hit one of the spheres they change color to red, and when they hit the other sphere they change to green. Because of the spacing and velocities, many particles bound back and forth between the two spheres changing color several times.

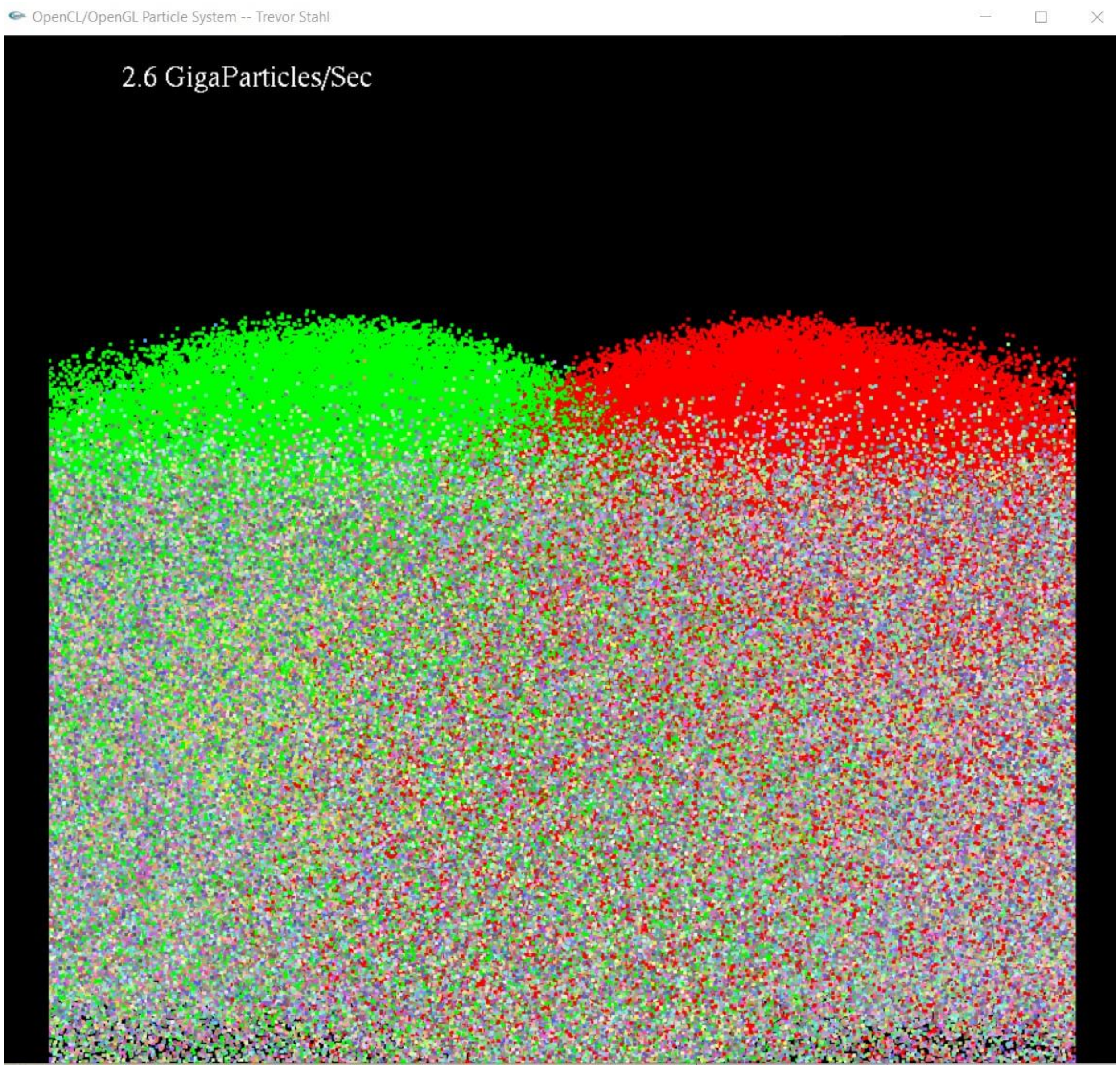
Screen captures:

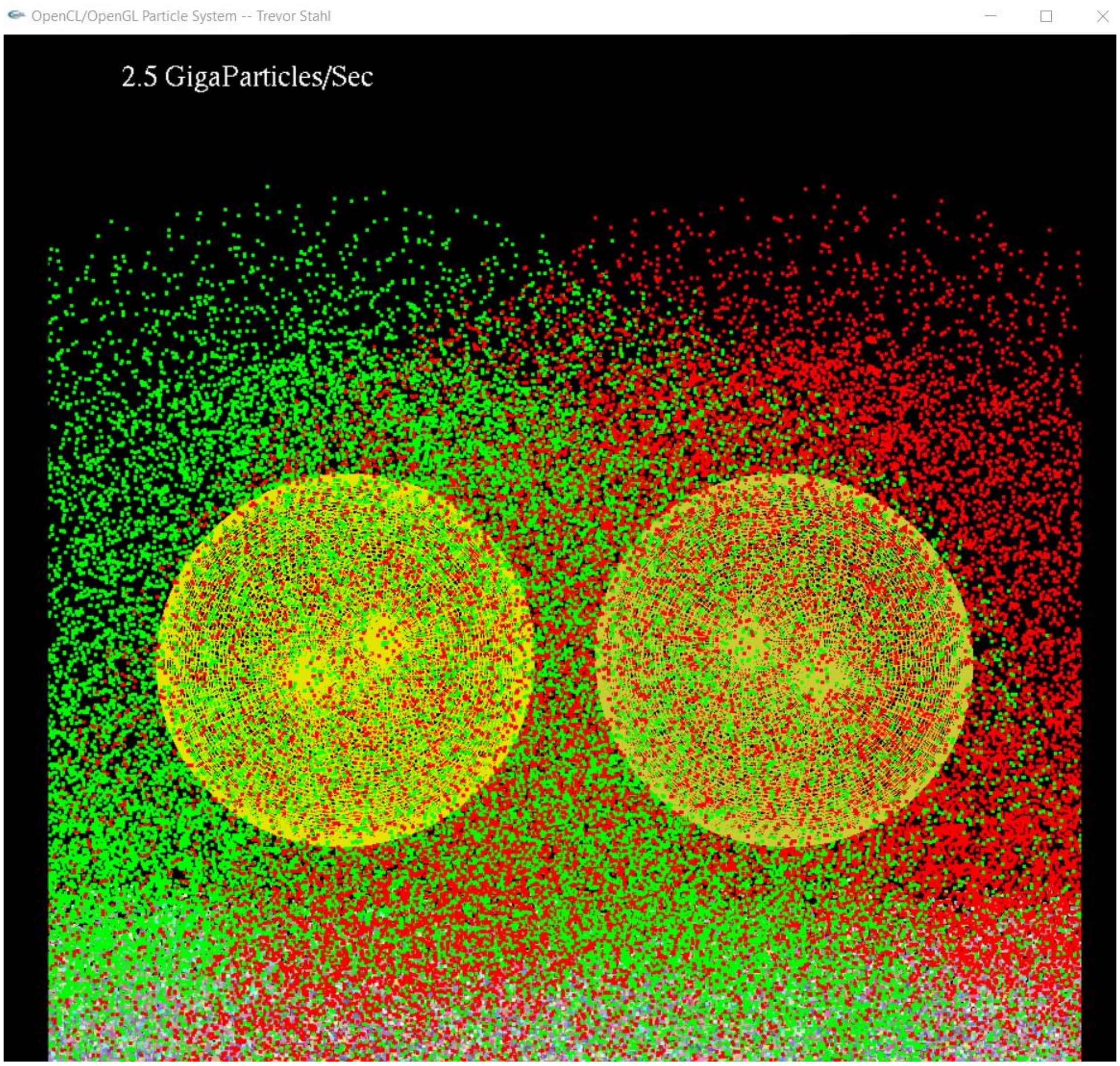


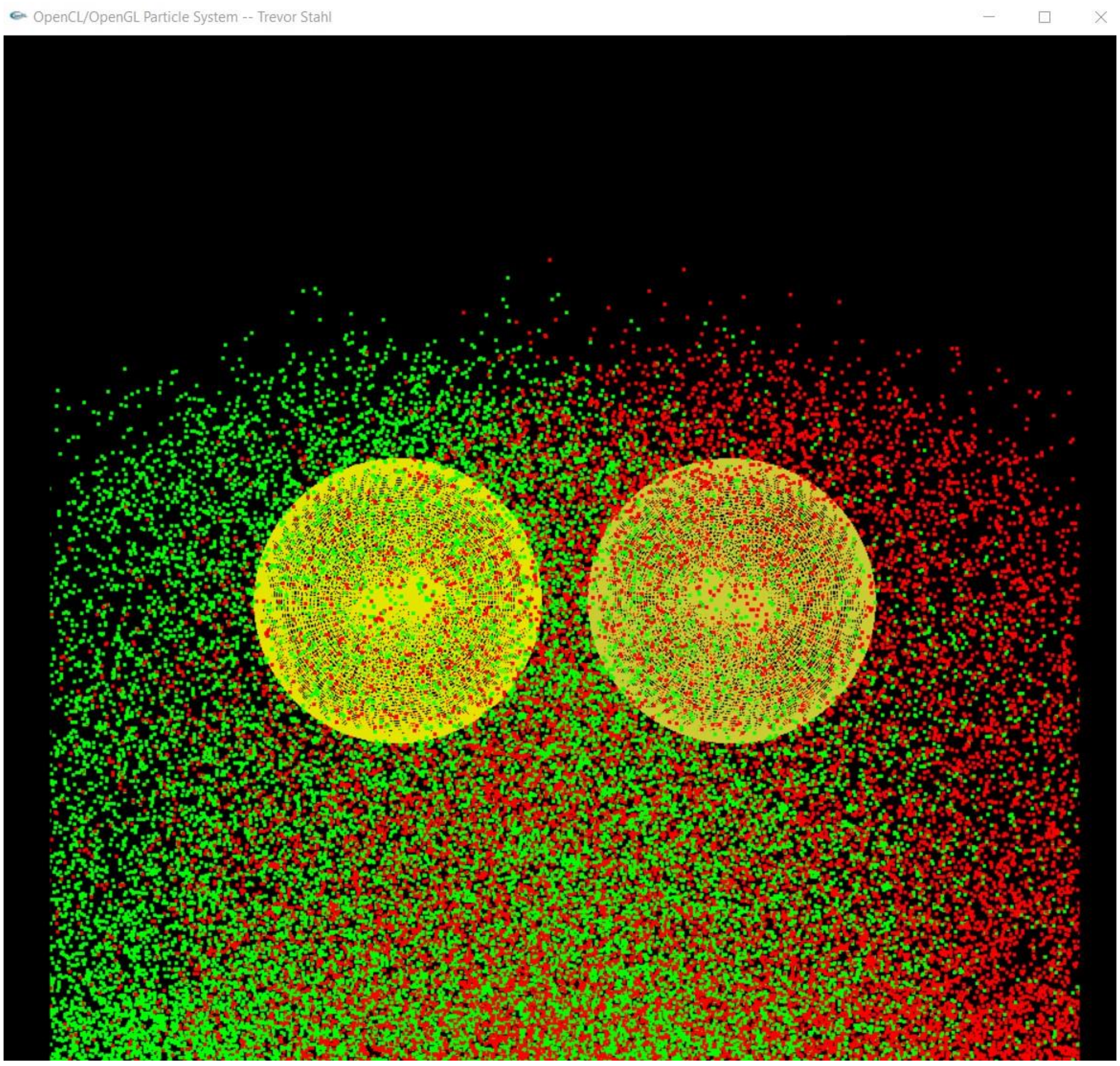

OpenCL/OpenGL Particle System -- Trevor Stahl
2.7 GigaParticles/Sec

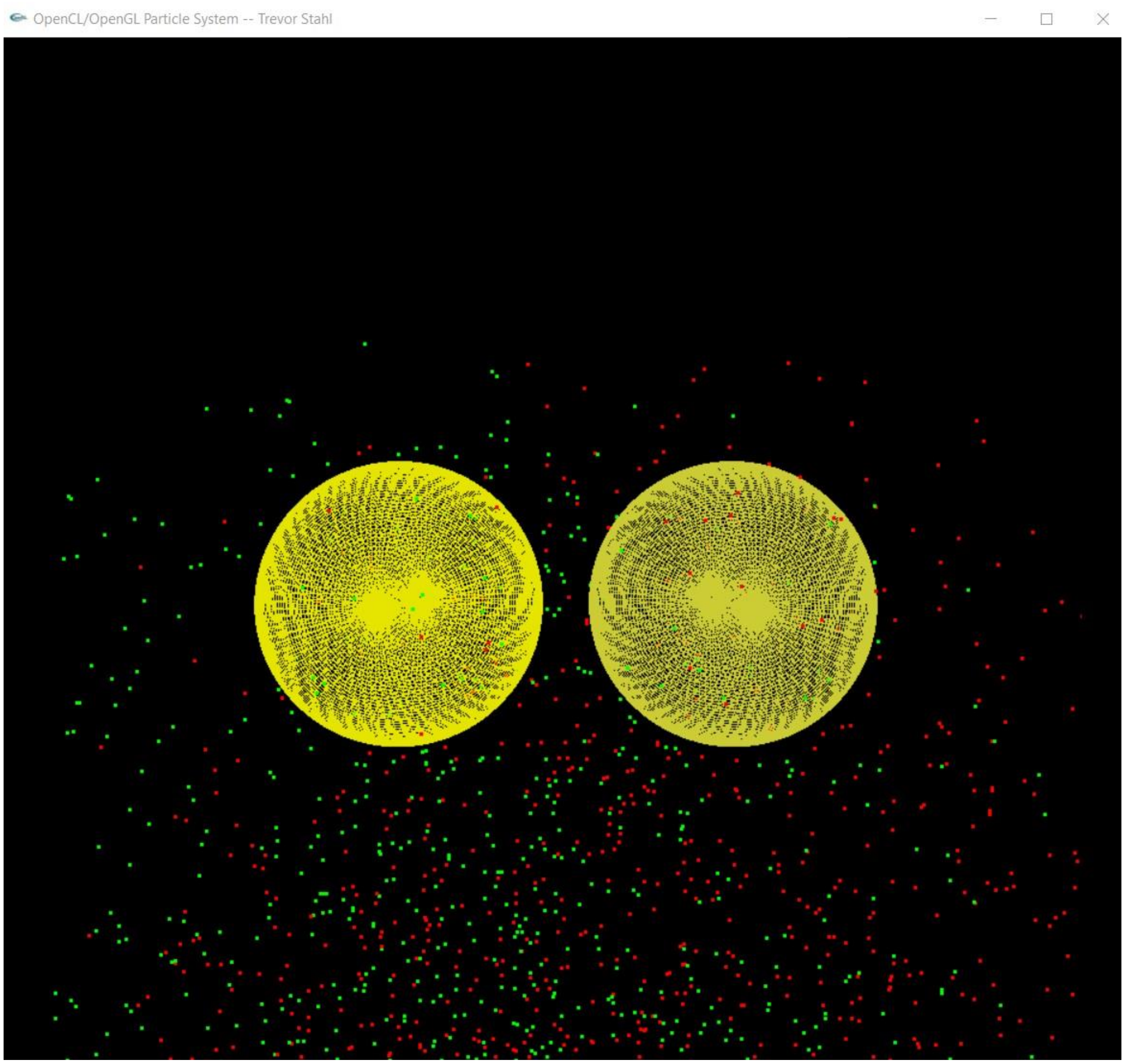

OpenCL/OpenGL Particle System -- Trevor Stahl

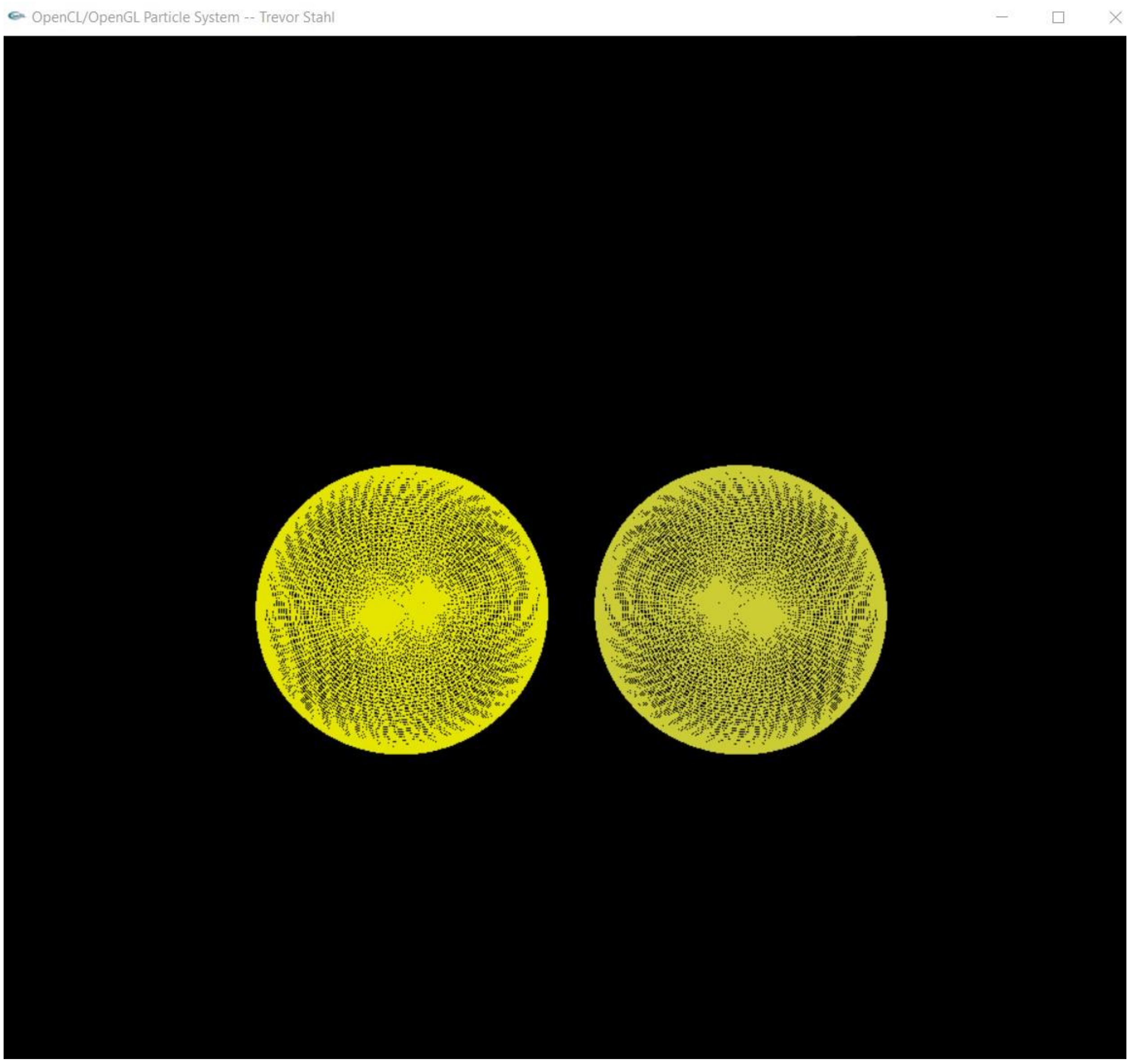
2.8 GigaParticles/Sec


OpenCL/OpenGL Particle System -- Trevor Stahl
2.6 GigaParticles/Sec


OpenCL/OpenGL Particle System -- Trevor Stahl
2.5 GigaParticles/Sec


OpenCL/OpenGL Particle System -- Trevor Stahl

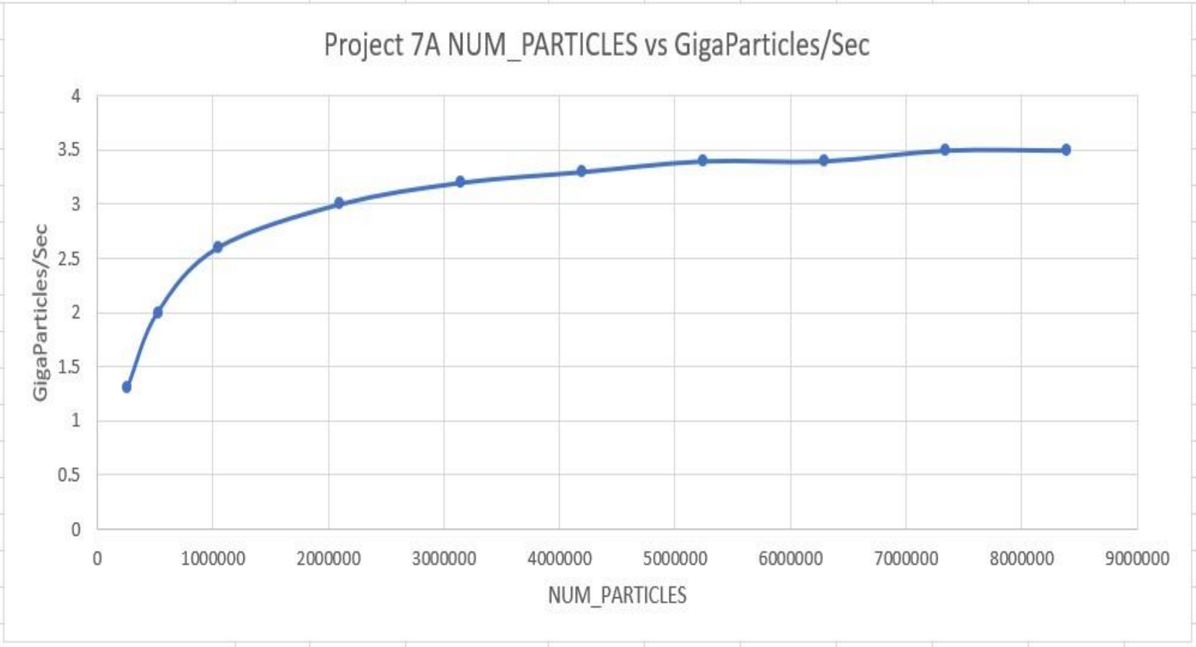

OpenCL/OpenGL Particle System -- Trevor Stahl


OpenCL/OpenGL Particle System -- Trevor Stahl

The value for DT I chose was: `const float  DT      = 0.01;`
I think this why my performance is kind of relatively low despite the power of my rig.

Table and Graph:

Total Number of Particles	262144	524288	1048576	2097152	3145728	4194304	5242880	6291456	7340032	8388608
Performance	1.3	2	2.6	3	3.2	3.3	3.4	3.4	3.5	3.5



Patterns:

The only pattern I am seeing is a sharp increase in performance as NUM_PARTICLES increases on the left half of the graph. Once NUM_PARTICLES is big enough, around 3 million, the increases in performance per increase in NUM_PARTICLES begins to slow down and flattens outs by the time we hit 5 million particles.

Why I think these patterns are:

This one pattern can be explained by the fact that there needs to be enough particles to make the overhead of OpenCL and OpenGL worth it. When the number of particles is small, the ratio between computation given to particles compared to computer power need for setting things up initially(overhead) and communication across the motherboard, etc, is larger and thus the performance is worse. Once there are enough particles, most computation is being used on particles and performance per second increases.

What this means for GPU computing:

I would suggest that this means one needs to be computing something very large with very many particles, or the case specific equivalent. Otherwise it would be faster to compute entirely only the CPU possibly, at least if the number of particles was extremely small. This applies to OpenCL but I am not sure about openGL. I have not yet gone back and studied the openGL notes and code, but I will be after the finals.