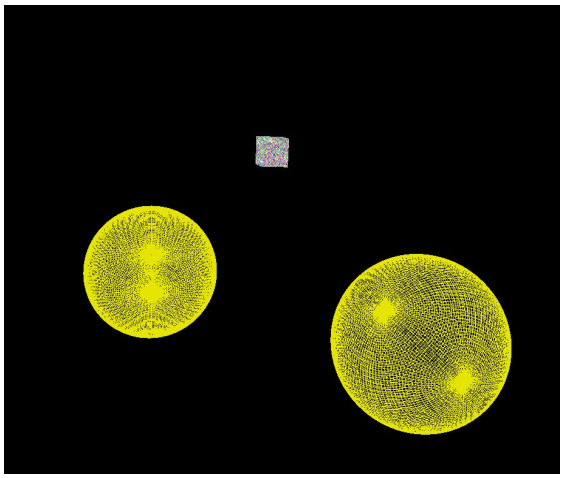
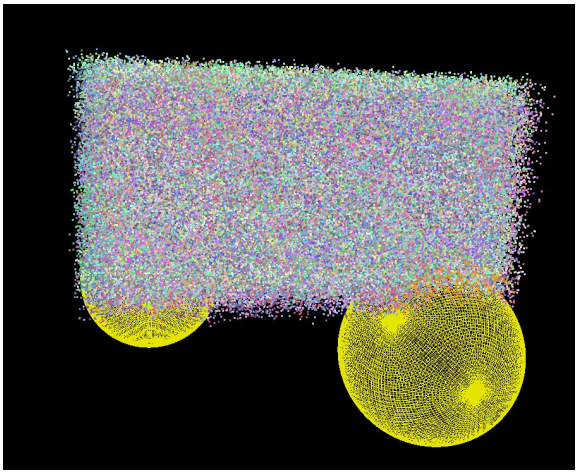
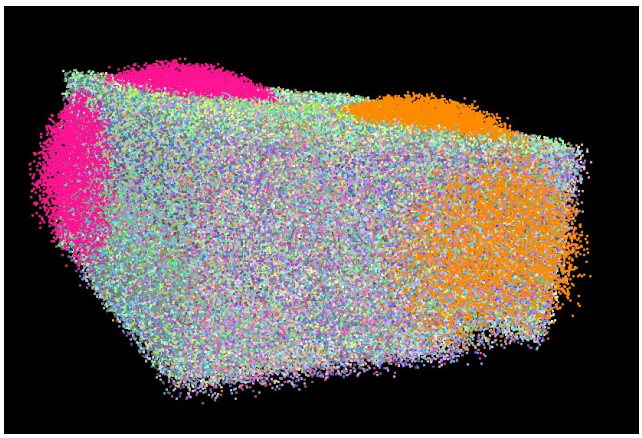
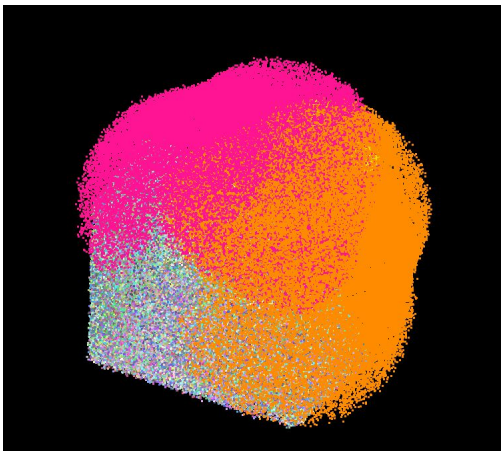
Linge Ge - CS475
Project 7A - OpenCL/OpenGL Particle System

Overview: Project 7A runs a particle system where particles follow gravity and rain down from a certain point given by a set velocity. There are two spheres on the way for some particles, and these particles bounce off. This system is meant to capture how well the GPU through OpenCL and OpenGL renders the particles.

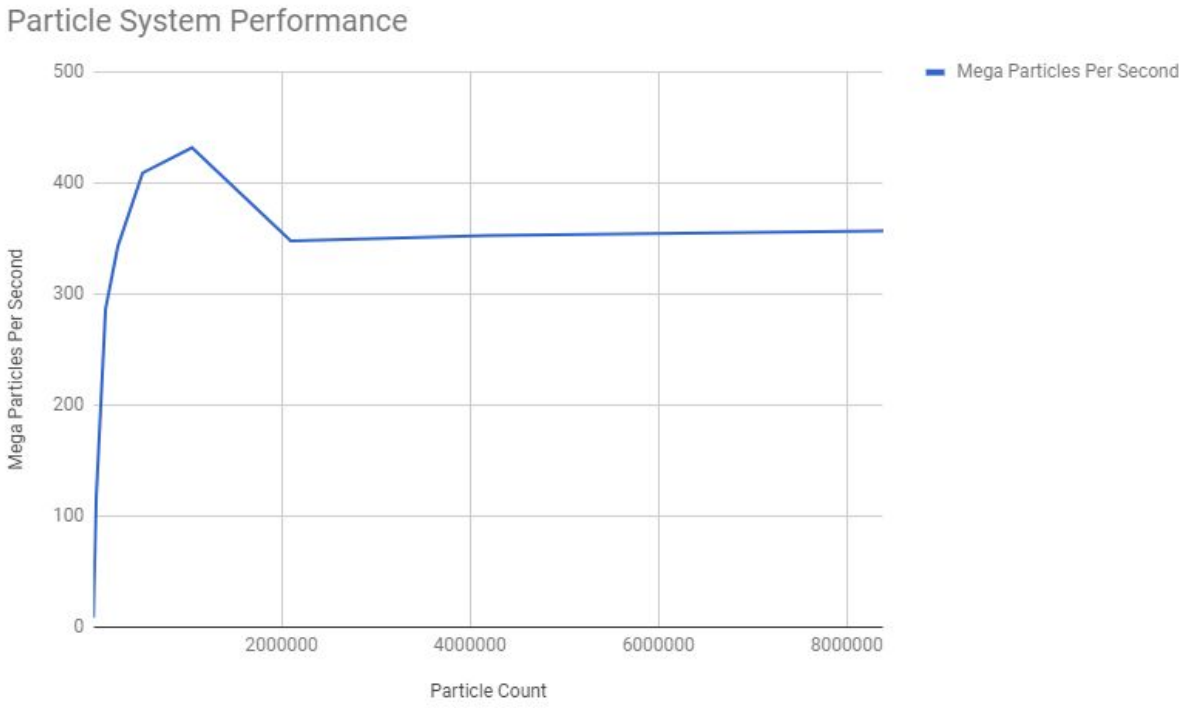
1. The code ran on:
    - Windows 8 - 64bit
    - Intel i7-3615QM - 2.30GHz - 4 cores
    - NVIDIA GeForce GT 650M - 384 CUDA cores, up to 900MHz

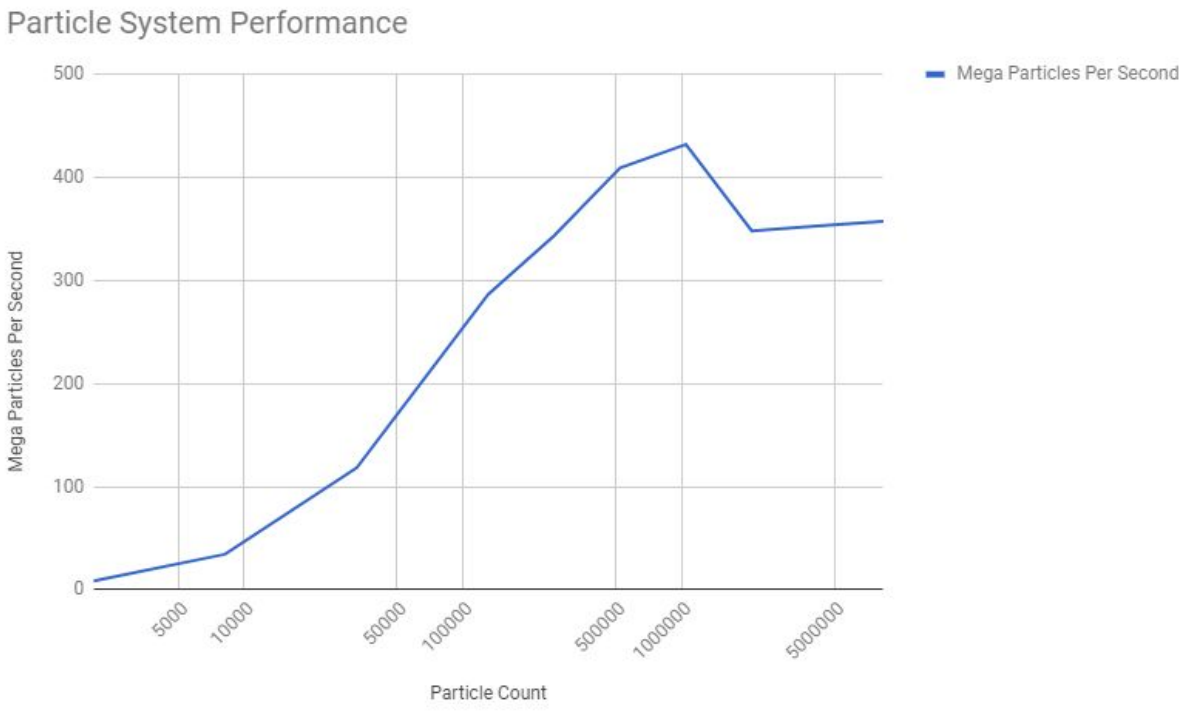
2. The particles will change to a specific color if they bounce on any of the two spheres. One sphere will change the particle color to orange, while the other sphere changes the color to pink.

3. 




4.

Particle Count	Mega Particles Per Second
2048	8.5
8192	34.4
32768	118.2
131072	286.8
262144	343.5
524288	409.4
1048576	432.1
2097152	348.2
4194304	352.8
8388608	357.3





5. The performance increases steadily until particle count reaches around 1M. Afterwards, performance degrades to around 350 MP/s in the 2M to 8M range.

6. The performance increases steadily from the beginning up to a particle count of around 1M because as we increase the number of particle count, the parallel work relative to the singular overhead increases, hence overhead costs of organizing the GPU work and having the CPU setup and send over the data become more and more minimal to the overall performance. The performance takes a minor hit past 1M and from 2M-8M, it remains consistently the same. I noticed that around these particle count values, there's also lag in the rendering the moving particles. This indicate that at >2M, the GPU is working at full capacity, and actually starting to overwork. This is detrimental to performance, as the lag shows that there are additional overhead that are being done with the CUDA kernel calls in order to still render effectively and consistently. At 1M, the performance is optimal because the GPU is working near or at full capacity, but the streaming processing is not compromised by additional tasks that must be done and rendered.

7. GPU should be used when there are tons of simple, straightforward data processing to be done. In this case, it was utilized in OpenCL to do simple calculations for particle position given a velocity and rendering that particle. Across many many particles. The higher number of cores and threads allow GPUs to easily parallelize tasks. However, a GPU should not be used when there is irregular or complicated flow control involved, as

it lacks the ability to do branch prediction or out-of-order logic. In this program, there were some simple if-then statements that dictated particle color. However, anything more complicated (particles affecting each other given conditions, other considerations, etc.) should not be handled by GPU, and such things should probably be handled using CPU parallelism (ex: interactions in SimCity, etc.). Essentially, a GPU should be used to do grunt work processing for large streaming data, that's worth the cost of a good amount of initial setup (setting up buffers, loading data over from CPU, etc.) and overhead.