Lachlan Sinclair

sinclala@oregonstate.edu

6/6/2020

CS 475 Spring

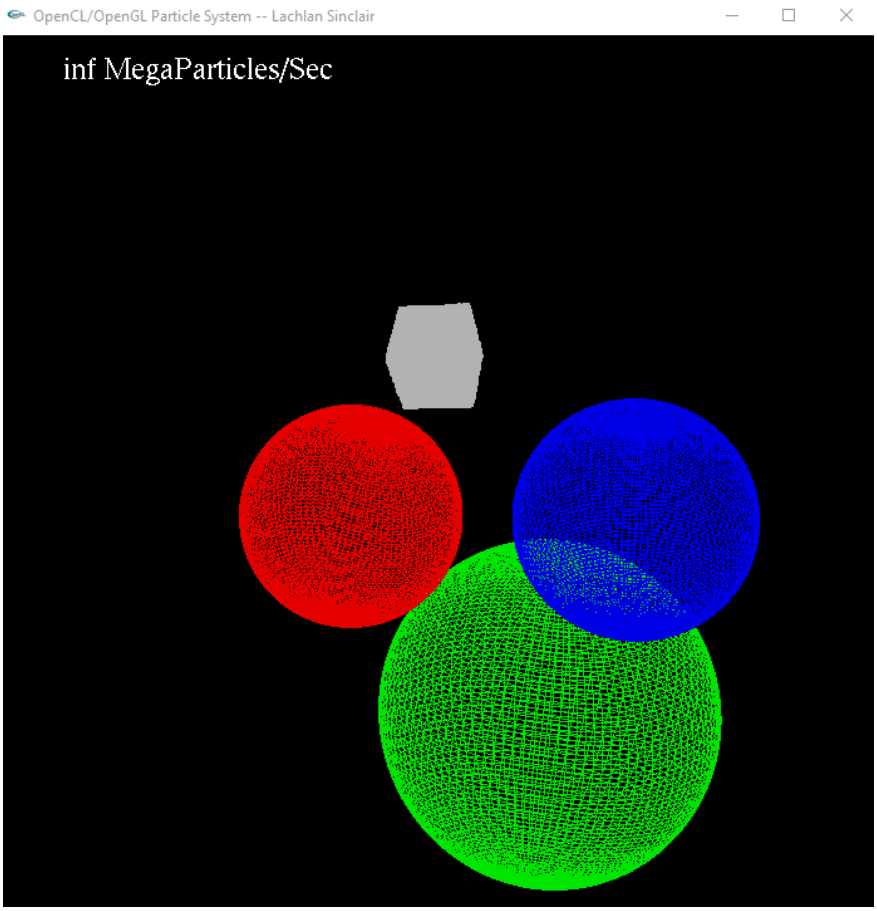
Project #7a: OpenCL/OpenGL Particle System

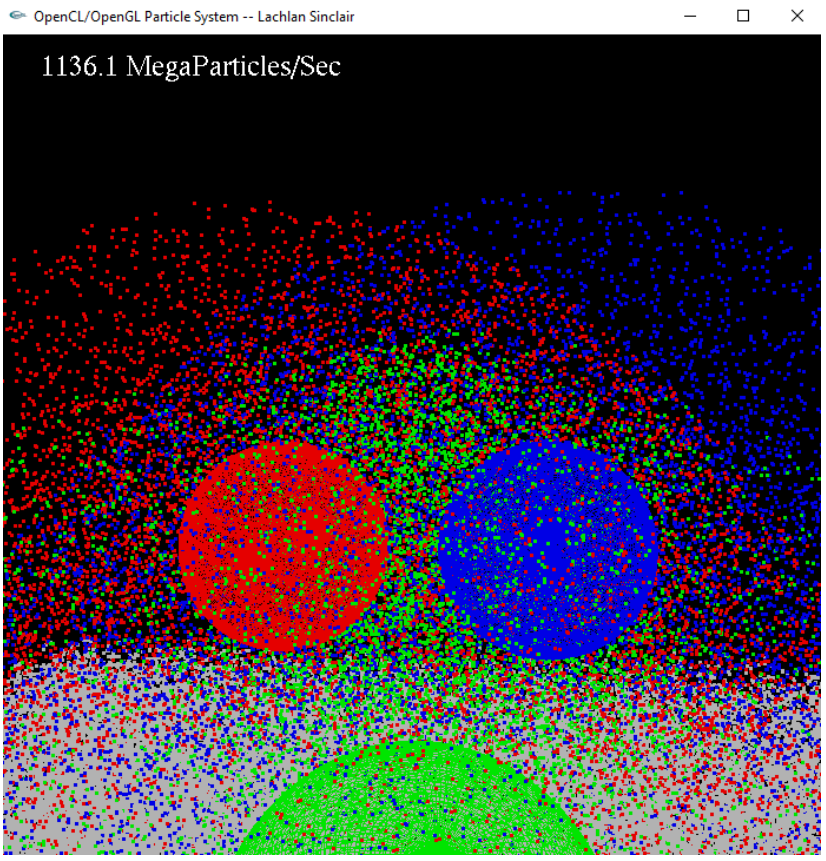
Video Link: https://media.oregonstate.edu/media/t/0_3g6xpual

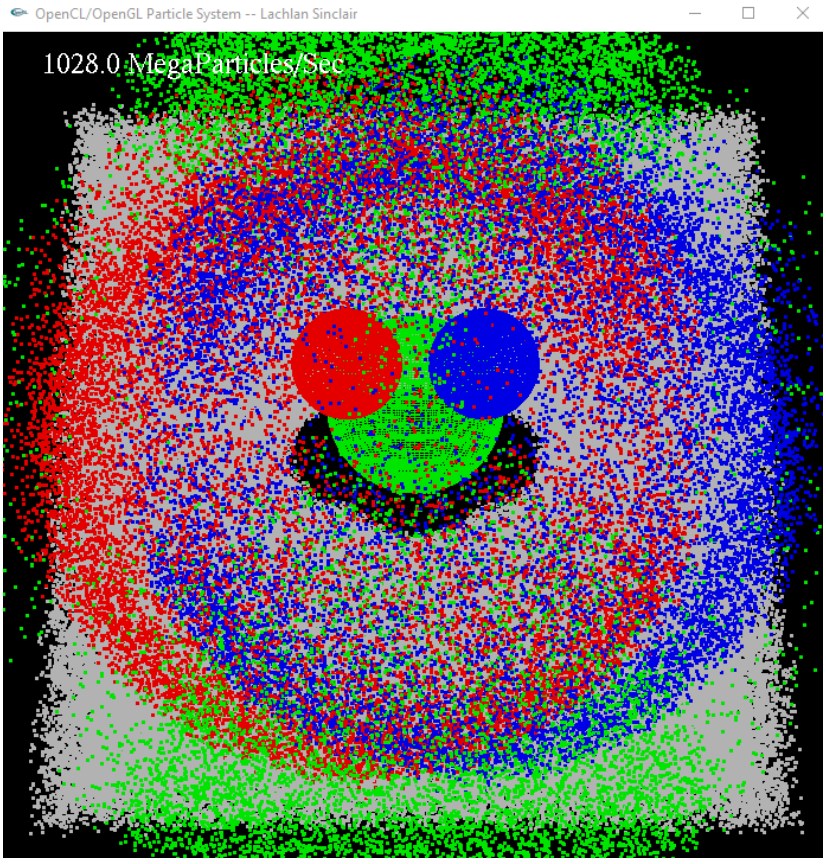
System: I ran this project on my personal computer which has a NVIDIA GeForce GTX 980 Ti graphics card. It uses OpenCL 1.2 CUDA 10.2.95. My CPU is an Intel Core i7-6700k (4 cores) and my computer runs Windows 10. I used Visual Studios 2017 community, so I had to change the projects platform toolset to v141.

Dynamic Particle Colors: I added two additional surfaces. All three surfaces have their own color (red, blue and green). I also changed the initial color of the particles to a light grey color. The particles change color to match the color of the surface they most recently bounced off. The particle will change color multiple times if it bounces off multiple surfaces.

Screen Captures:



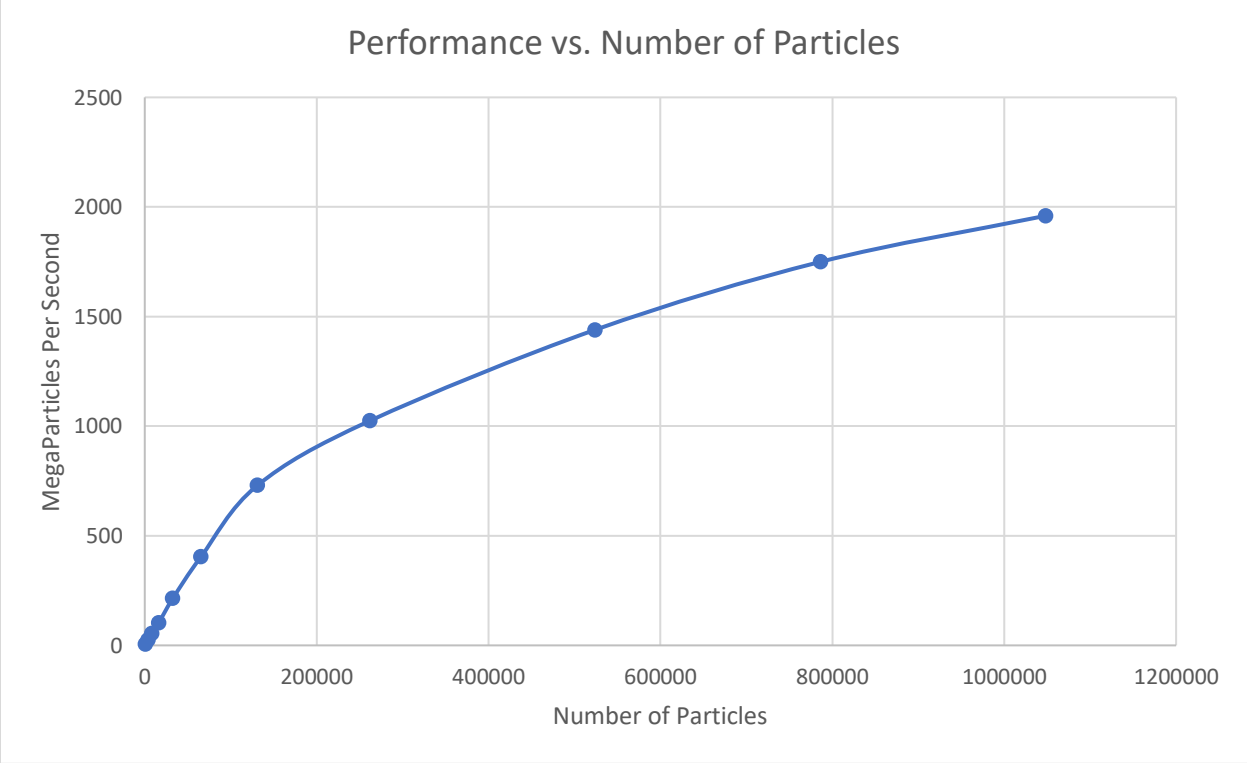

OpenCL/OpenGL Particle System -- Lachlan Sinclair
1136.1 MegaParticles/Sec


OpenCL/OpenGL Particle System -- Lachlan Sinclair
1028.0 MegaParticles/Sec

Results:

Local work group size 128.

Number of Particles	MegaParticles Per Second
1024	6.5
4096	26
8192	54
16384	104
32768	215
65536	405
131072	730
262144	1025
524288	1440
786432	1750
1048576	1960



Explanation:

All results were obtained when using a local work group size of 128. After running tests using 1024*1024 particles with different work group sizes I choose to use a size of 128, increasing the size further did not produce noticeable increases in performance. This also corresponds to the results achieved in Project 6. To obtain performance metrics I used the method Professor Bailey recommend on Piazza, which was running it a few times and "Pick a value that you think is a good representation of the average."

As the title of the project suggests, this program works by using OpenCL and OpenGL. As described in the assignment, first, OpenGL buffers are created for the particle's positions and colors. The particles velocity information is stored in a C++ array. The initial position and color values are then sent to the buffers on the GPU. Then, the position and color OpenCL buffers are created using the existing OpenGL buffers and the C++ velocity array is used to generate an OpenCL buffer. This creation of OpenCL buffers from the OpenGL buffers shows off the OpenCL/OpenGL interoperability as discussed in the week 9 lectures. After the setup is complete the animation begins, the OpenGL idle function tells OpenCL to acquire the buffers. OpenCL then runs its kernel and releases the buffers, after which OpenGL draws the particles using the changed buffers. This process then continually repeats itself.

Analysis:

When the number of particles was low the performance was poor, this is because there wasn't enough data to utilize a significant amount of the GPU's cores in parallel. The performance increases significantly as the number of particles begins to increase from 1024, this is because the performance at those low sizes is locked by the number of particles. At these sizes, each time a new particle is added the calculations are done in the same amount of time, but it is doing more calculations which increases the performance. At around 260k particles the rate at which the performance was increasing begins to slow down. At 260k particles and above I think that GPU is utilizing a significant number of its cores and is now making use of the additional warps provided by having a local work group size of 128. As the number of particles increases past 260k the performance still continually increases. As discussed in the analysis for project 6 I think this is because of Gustafson's observation about Amdahl's law, as the amount of data increases the maximum speedup achievable increases. I suspect if I were to keep increasing the number of particles the performance would continue to go up, albeit at a continually slower rate.

At a particle count of approximately 1 million the program was achieving almost 2000 megaparticles per second on a 5-year-old GPU. This shows the massive performance achievable when using GPU parallel computing in the proper manner. This also confirms the interoperability of OpenCL and OpenGL discussed in the lectures, when they are sharing a buffer the performance achieved is impressive. It was slower than performance we saw when simply using OpenCL in the previous projects, however it is still around the same order of

magnitude which is remarkable considering the amount of calculations taking place and the time required to render the graphics.