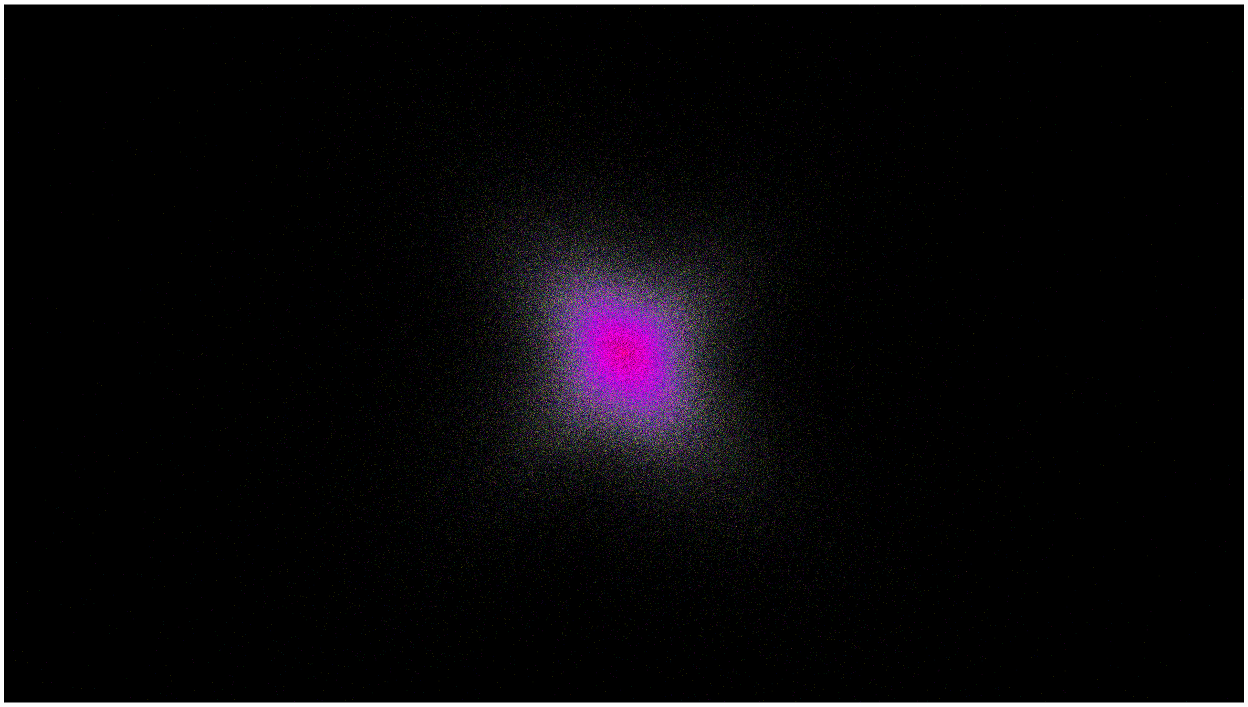
CS 475 Project 7A

Jonathan Keller (kellerjo@oregonstate.edu)



I implemented my project using Rust and Vulkan, using the vulkano API bindings. As long as you have Rust installed and you have Vulkan drivers, you can compile and run with the command `cargo run --release`.

Note that I only tested this on my laptop, a MacBook Pro with the Apple M1 Pro CPU/GPU, but it *should* work just fine on other machines. Emphasis on *should*.

Instead of creating sphere bumpers, I programmed the particles to interact with one another using a simple simulation of “pressure”. At the end of the compute shader for each frame, each particle decides what “cell” of the world it’s in and increments a counter for that cell. On the next frame, the values of those counters are used as the “pressure” for each cell, and (based on this pressure) a random vector is added to each particle’s velocity to simulate the effect of particle collisions. The resulting behavior is stochastic: on a microscopic level the particles bounce around randomly, but on a macroscopic level they tend to be pushed along decreasing pressure gradients.

The pressure between particles is counteracted by a gravitational force that pulls all particles towards the origin and a drag force that dampens high velocities, which over time leads to something of an equilibrium between gravity and pressure. But equilibriums are boring! To change it up, we play an audio track (using the minimp3 and cpan libraries), and use the audio waveform to modify the constants of the simulation.

Each frame, the pressure constant is multiplied by the average audio amplitude. So when the audio is loud the pressure force is stronger, and the cloud of particles tends to expand outward – whereas when it’s quieter, the pressure force decreases and the cloud contracts under gravity. Additionally, we perform a fast Fourier transform on the audio of each frame in order to affect each particle individually based on the frequency spectrum on the audio. Each particle has a hue assigned at startup, and different frequency bands affect different colors of particles: red particles are affected by low frequencies, and blue particles by high frequencies. As the loudness of a frequency band decreases, the particles that belong to that band increase in brightness, and the effect of drag on those particles decreases (allowing them to appear brighter and move faster).

The result is something of a music visualizer in the form of a particle cloud. You can watch the video here – though it looks really bad on Kaltura. I recommend running the program for yourself, as one million pixels randomly moving around and changing in color is not really something video compression deals with well: https://media.oregonstate.edu/media/t/1_lje5d2si

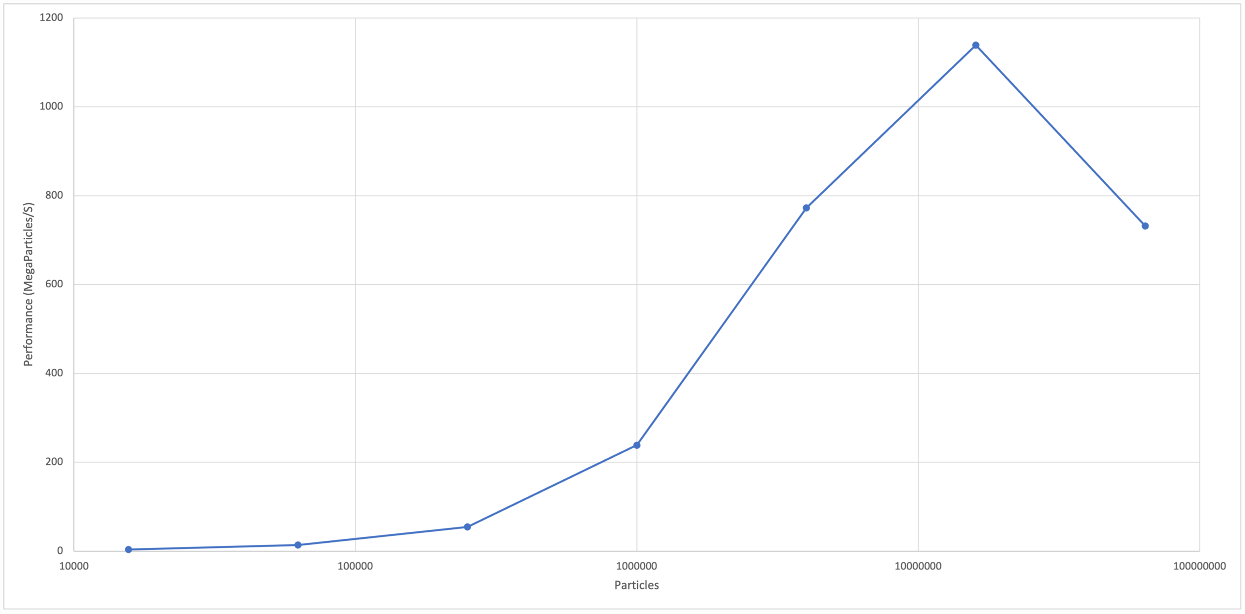
I used a local work group size of 32 elements, and a global size of 1 million particles (though I varied from 15,000 to 64 million particles when measuring performance).

The song is [*Pearl* by Anamanaguchi]. You can rename any other MP3 file to `pearl.mp3` and recompile, and it should play.

Here is my performance table:

Particles	Performance (MP/s)
15625	3.68280375
62500	13.6596261
250000	54.6329175
1000000	238.547501
4000000	772.612645
16000000	1139.02111
64000000	731.651331

And my graph:



As usual, the performance increases greatly as we increase the dataset size – at least up to a point – as the performance benefit becomes substantial enough to overcome the costs of setting up the GPU and transferring all the data. However, I was surprised to see the performance decrease above 16 million particles. I have not profiled, but I wonder if this is because very high numbers of particles leads to contention when each particle has to atomically incrementing its cell’s particle counter. If this hypothesis is correct, it indicates that heavily-contended atomic operations may not be a good use of GPU parallel computing.