Flicker: An Interactive Particle Simulation Game

Authored by Jayden Nyamiaka

1. Description

The code implements a GPU-acceleratable particle simulation game called "Flicker" that seamlessly integrates CUDA and OpenGL to get user-input, compute particle evolutions, and display results to the screen in real time. The behavior of particles is dependent on the user and needs to be computed and rendered every frame, introducing significant challenges for continuously passing data between the CPU and GPU. As the simulation progresses, more kinetic energy is added to the system, or from the player's perspective, the particles move faster and faster. Along with base speed acceleration, there are a few more factors that guarantee the game stays dynamic. The application currently supports 1 player and 3 different particle types.

Player:

- Controlled via WASD or Arrow Keys (simultaneously)
- Color smoothly transitions over RGB spectrum
- Can die on collision with particles. This should be on when using a game.

Particles Seeker (Particle 1):
This particle aims at the Player's position (at spawn time) and darts straight there with increasing acceleration. This particle changes colors from red to yellow depending on how fast it's traveling.

Cruiser (Particle 2):
This particle shoots in a simple (horizontal or vertical) direction but turns at seemingly random times. The turning is psuedo-random and frame-independent implemented via a custom procedure that doesn't directly use any RNG. For implementation details, reference the code (and comments). This particle also changes colors depending on the direction it's moving such that horizontally moving particles are green and vertically moving particles are violet.

Wanderer (Particle 3):
This particle wanders around by simulating Brownian motion, creating hot spots of dangerous unpredictability for the Player. Brownian motion models the random motion of particles suspended in a medium and is a mean zero, continuous process, often implemented using GPU acceleration due to its computational parallelism. This particle is also gray colored and slightly bigger than the others.

For more information, refer to the code.

2. Set Up

CUDA and OpenGL

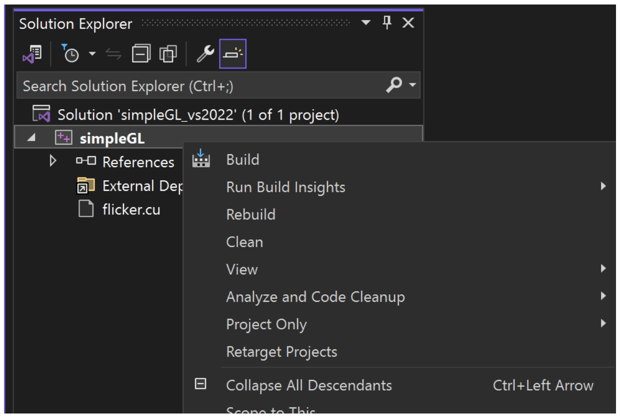
To run the code, you need CUDA set up on your local machine and built using Visual Studio 2022 (not VSCode). The code was only developed and tested for Windows OS. However, in theory, it should work for Linux and MacOS as well due to portability of the reference code. Nvidia's popular Samples repository was used as a base reference for development. Samples is supposed to be portable to any machine running any supported version of CUDA, so Linux and MacOS should be supported

If you don't have CUDA downloaded on your local machine, go to this link. It is Nvidia's official CUDA Toolkit Download which will install both the Toolkit and the Driver. Then, following these instructions will take care of the rest. Downloading the whole CUDA Toolkit will automatically download the OpenL interoperability.

WSL2 is NOT SUPPORTED because it does not support CUDA OpenGL interoperability. It is one of the only CUDA functionalities that WSL2 doesn't support. So, if you're using WSL2, you are going to need to set up CUDA on your host Windows OS to build the project.

Visual Studio 2022

Once CUDA is set up, use Visual Studio 2022 to build the project. If you don't already have Visual Studio downloaded, it is very easy to set up here. To build the project in Visual Studio 2022, go to the Flicker/Sample/Demo/simpleGL folder in the Solution Explorer, and double click on the simpleGL_vs2022.sln file. This should focus on just the simpleGL solution. From there, you can right click on simpleGL to open a drop down menu and press "Build". Finally, look at the Output console to see where the "flicker.exe" executable was built. It may be called "simpleGL.exe", but that's just some small build error. Cd into the directory, use ls to confirm the existence of "flicker.exe", and then run it via "./flicker.exe". This will run the most simple instance of the simulation. Look below for more about usage.



3. Usage

The application doubles as both a game and a particle simulation, so the arguments we opt to use for the application depend greatly on the use case.

After building, the application can be called from the command line according to the following:

```
Usage: ./flicker.exe [options]
Options:
--help           Display this information.
--gpu-accel      Accelerate the simulation using the GPU. Otherwise, use the CPU.
                 Recommended for when using as a simulation.
--can-die        Has particle collisions kill the player & stop the simulation.
                 Otherwise, the player can't die. Recommended when using as a game.
--stagger        Stagger particle starts. Otherwise, all particles start simultaneously.
                 Recommended when using as a game.
--set-n n1 n2 n3 Manually set the number of each type of particle. Each n1 n2 n3 must
                 be a non-negative integer. The default is 32 32 32.
--preset [1-11]  Run the indicated simulation preset 1-11, discarding all other options.
                 The presets are as follows:
                   1:  Easy Game
                   2:  Hard Game
                   3:  Small All Particle Simulation
                   4:  Large All Particle Simulation
                   5:  Small Seeker Particle Simulation
                   6:  Large Seeker Particle Simulation
                   7:  Small Cruiser Particle Simulation
                   8:  Large Cruiser Particle Simulation
                   9:  Small Brownian Motion Simulation (Wanderer)
                   10: Large Brownian Motion Simulation (Wanderer)
                   11: Huge Seeker Particle Simulation
```

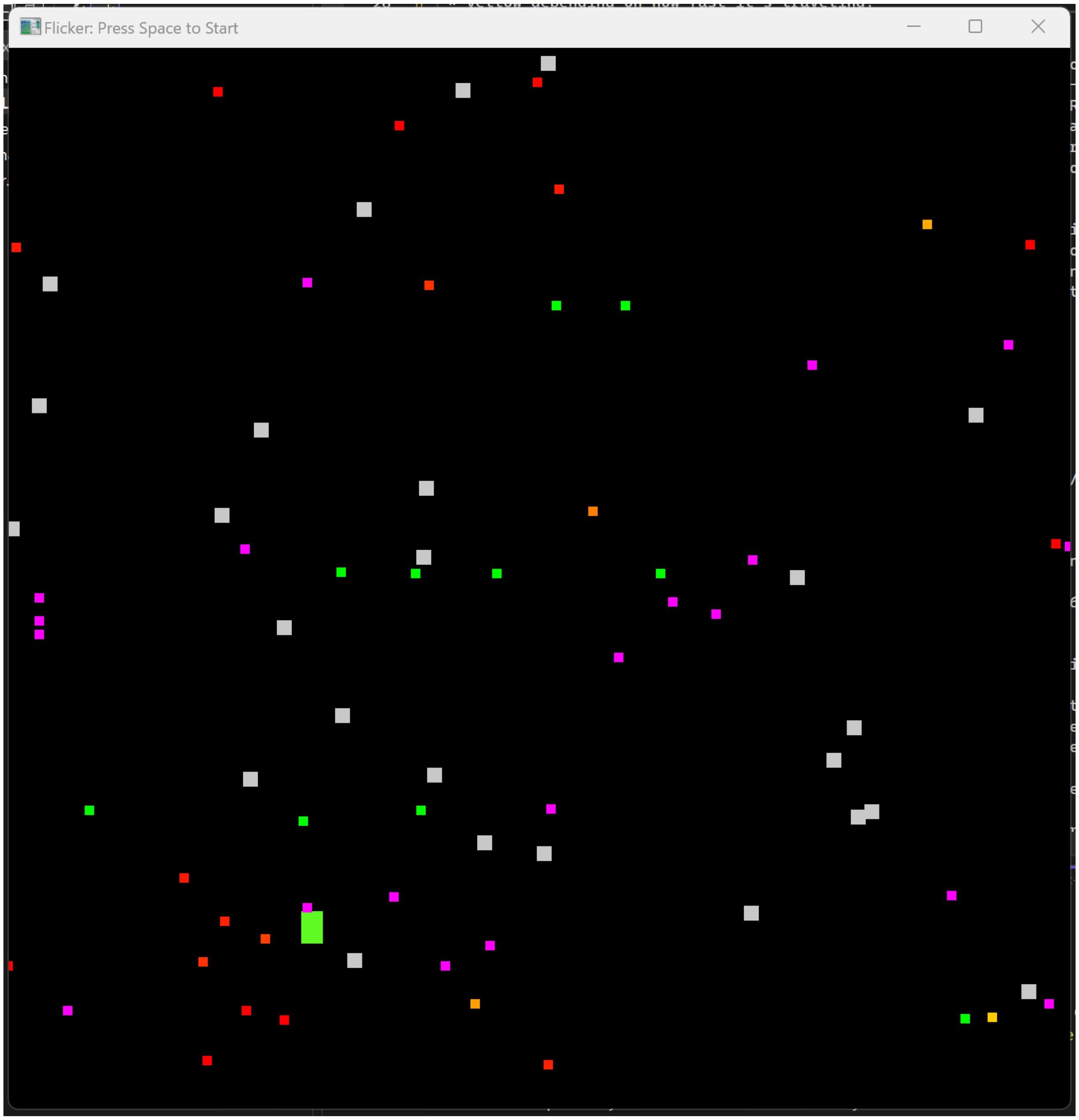
Once the application has been initialized (the screen has popped up), you can start the application by pressing space (instructed in the window title).

The presets list a few recommended calls depending on how you would like to use the application. Of course, you're encouraged to experiment with arguments and change the behavior of the application any way you like.

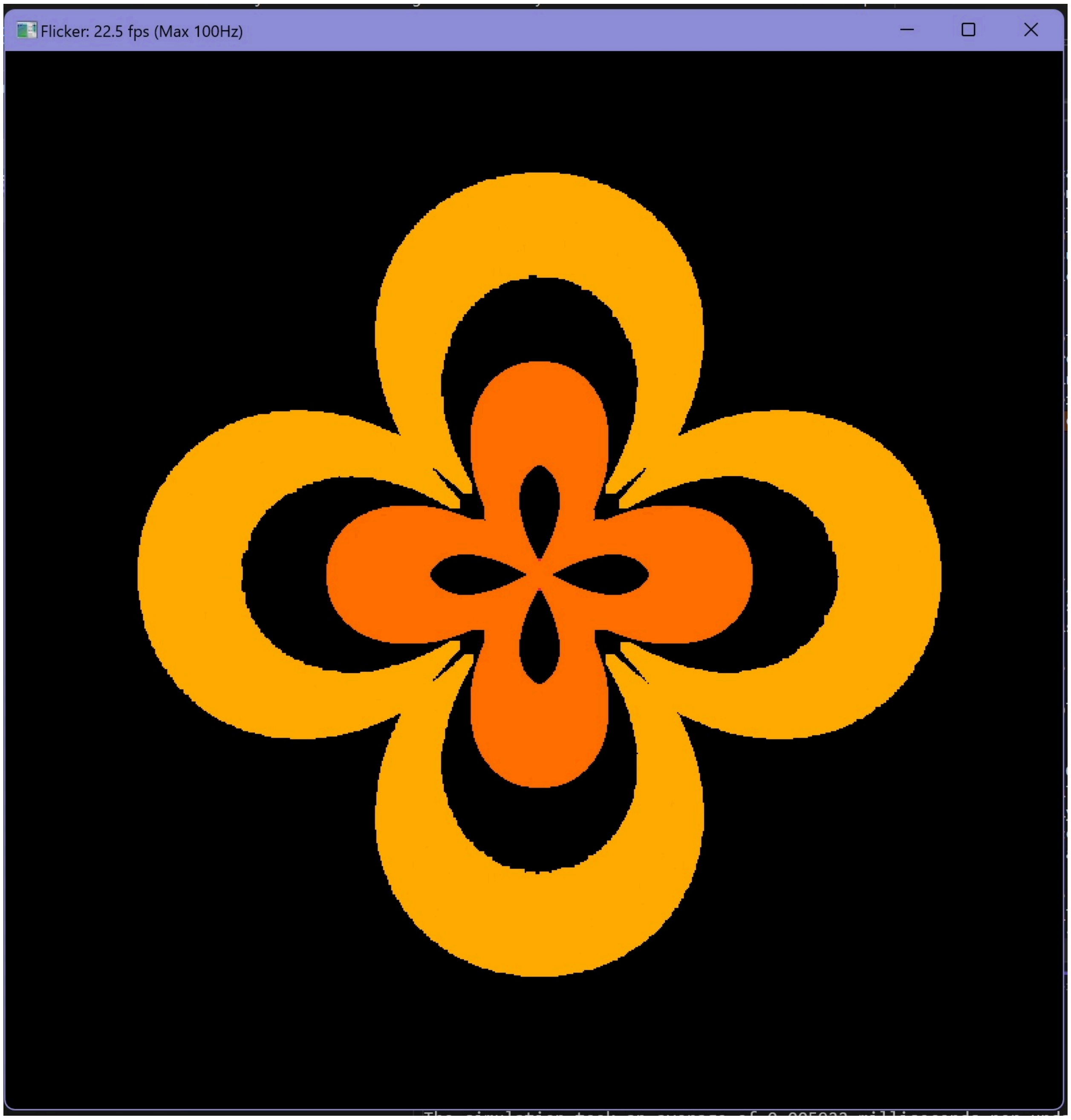
For all of the above presets, gpu_accel is toggled depending on whether the application can benefit from GPU usage. In general, larger simulations benefitted from GPU acceleration due to the large number of particles that could all be evolved in parallel whereas the game and smaller simulations had too few particles to see any significant time improvements.

4. Demo

The application works as both a game and a particle simulation at the same time. These are some pictures of the application for different presets. In addition to the pictures below, there are also videos demonstrating the application within this folder.



Hard Game
Even though this number of particles on the screen is trivial for the CPU such that GPU acceleration isn't necessary, it's entirely different for the player. It's very difficult to dodge this many particles especially once they start getting faster.



Huge Seeker Particle Simulation
The Seeker particles lock on to the player's position, independently of whether the player can die or not, so it's good to move around and see the effect that has on the distribution of the particles over time. The Huge Seeker Particle Simulation, in particular, has interesting behavior and beautiful patterns.

5. Performance Analysis

For performance analysis, we can measure the time of initializations and average frame updates for the same simulation with GPU acceleration toggled on and off. To provide a range of scenarios, the presets were used. As previously mentioned in *Usage*, larger simulations tended to benefit from GPU acceleration while for smaller scale simulations, it was mostly unnecessary.

Initializations are listed as the amount of time to initialize all particles. Average frame updates are marked as the average number of milliseconds it took to compute and render a single frame to the screen. Both measurements include both the set up and wrap up off any resources needed to do the computation. See below for figures.

Easy Game	CPU	GPU
Initialization	0.003482 seconds	0.024873 seconds
Average Frame Update	0.005718 milliseconds	0.003721 milliseconds

Small All Particle Simulation	CPU	GPU
Initialization	0.014426 seconds	0.041592 seconds
Average Frame Update	0.000236 milliseconds	0.004796 milliseconds

Large All Particle Simulation	CPU	GPU
Initialization	3.689622 seconds	0.273515 seconds
Average Frame Update	0.012038 milliseconds	0.003951 milliseconds

Huge Seeker Particle Simulation	CPU	GPU
Initialization	85.866722 seconds	3.388666 seconds
Average Frame Update	*Stopped Responding*	0.027726 milliseconds

We notice that simulations with few particles, it's not only unnecessary, but detrimental to use the GPU algorithms. With such little computational intensity, the set up and clean up required to map data to and from the CPU and GPU every frame can largely outweigh the slight time reduction off the GPU parallelism. However, with larger simulations, we see that the performance increase becomes significant as the amount of work split over different threads is worth the overhead and wrap up costs.

This demonstrates the tradeoffs we have when designing GPU algorithms and why it's important for us to consider the extent of the problem at hand before diving right into implementation. For us, we can conclude GPU acceleration is detrimental if we want to use the application for a (reasonable) game but essential if we want to run a large particle simulation.