Fireworks CUDA Renderer

Jiyu Hu (jiyuh)
Zhaohong Lyu (zhaohonl)

URL
https://v1siuol.github.io/fireworks

Summary
Our goal is to create an efficient and realistic fireworks renderer using CUDA on NVIDIA GPUs. Through experimentation, we will explore different approaches, including sequential versus parallel implementations, and various rendering techniques. We will conduct a thorough analysis of our results to determine the most effective methods for achieving our objective.

Background
Computer graphics have been growing rapidly in the last decades. We can now enjoy dynamic rendering by powerful GPU devices in real time. Past projects such as fire simulation and wave simulation all achieved great success thanks to the high-performance parallel processing routines. We want to explore more possibilities with parallel programming models in achieving realistic rendering effect with a firework simulation.

In order to achieve a realistic effect for our fireworks project, we plan to incorporate a coloring component that includes shading and lighting effects. The amount of computation is too heavy for sequential process even on the latest CPU devices. Therefore, we reckon this component may benefit from parallel processing techniques.

The rendering process for our project is computationally intensive, as we need to compute numerous particle movements as well as lighting of each of the particles. We intend to utilize CUDA programming framework to fully utilize the capabilities of Nvidia GPUs and achieve real-time rendering of firework scenes.

An animation function is designed to be responsible for updating particle positions and velocities. We plan to implement parallel techniques to optimize the performance of this function.

The Challenge
It is challenging to develop a feasible rendering technique that achieves our goal of realistic effect. This might involve complex shading and lighting algorithms, which further introduce heavy computations and rendering dependency.

Another major challenge is tackling the parallelism due to the constraint of rendering order dependency. Depending on how we approach parallelism, workload may be difficult to characterize and predict due to the nature of fireworks simulation. Given the potential workload imbalance, it is challenging to develop an efficient rendering technique as well.

Each frame is highly likely to depend on multiple particles to compute the final coloring. Therefore, to lower the communication/synchronization overhead between the CUDA threads is very important. Also due to the nature of illumination and shading, each pixel might require very different amount of computation for the final rendering, resulting in heavy load imbalances. And due to the dynamic nature of the fireworks, it is hard to precompute the work assignment beforehand.

Another problem is to fully utilize the bus bandwidth as we want to display high-resolution frames in real time. We assume some pipelining and buffering is required to achieve the real time effect.

Resources
We will be working on GHC machines that contain NVIDIA GeForce RTX 2080 GPUs. We will most likely start with our Assignment 2 CUDA Lab implementation. We may consider other rendering approaches, but we at least have some insights on the rendering workflow and infrastructure.

Goals and Deliverables
Plan to achieve:

- Sequential version of the code that renders realistic fireworks as baseline
- Highly parallel implementation that achieves the same rendering effect with significant speed up
- Detailed analysis of the speedups and bottlenecks of the parallel program
- Showcase of firework simulation result during poster session

Hope to Achieve:

- Real-time rendering of the firework simulation with high frame rates (60 - 120 fps)
- Implement reflection effect of the fireworks on a lake

Deliverables:

- Final report with detailed explanations of the optimization techniques used, speedup due to different techniques and bottlenecks
- Codebase of sequential and parallel implementation.

Platform Choice

We have Nvidia GeForce RTX 2080 graphics available on the gates machine that we can test our implementation. Therefore, CUDA is necessary for us to program on the GPU. Also, after some research, we found out that the OpenGL framework can render the computed frames in real time. While this is our current selection, we may not use OpenGL if we find out that there are other easier methods to render the generated frames in real time.

Schedule

4/3 - 4/7
Research for feasible approaches
4/10 - 4/14
Implement sequential implementation
4/17 - 4/21
Draft on parallel implementation
4/24 - 4/28
Optimize on parallel implementation
5/1 - 5/5
Add documentation and benchmark