COS 426 Final Project: Simulating Fireworks

Bhaamati Borkhetaria and Chege Gitau (bhaamati@, dgitau@)

1 Introduction

1.1 Goal

What did we try to do? Who would benefit?

We wanted to render interactive fireworks. The goal was to be able to click on different parts of the screen and having fireworks originate near the mouse click. We wanted the fireworks to be aesthetically pleasing and render as prettily as possible. Ideally, the interactive fireworks would benefit someone who wanted to integrate firework displays into a scene while also allowing user interactivity.

1.2 Previous Work

What related work have other people done?

One of the main papers we read for the assignment was 'Rendering fireworks displays' [Lok+92]. This paper discusses one approach to rendering fireworks which involves storing alive and dead particles in doubly-linked lists in order to represent fireworks. The paper also discusses how to use a particle engine in order to use the same particle objects to render different types of fireworks.

When do previous approaches fail/succeed?

One of the things we read about a lot is performance when it comes to fireworks - especially multiple fireworks on the screen. Many articles we read elaborated on how costly it is to store multiple sets of particles and do operations on all of them.

1.3 Approach

What approach did we try?

We tried to use the approach to particle systems that last year's COS 426's animation assignment took [Sta]. We preserved the basic framework and the data structures which did something similar to what we wanted: they store a set number of particles that get recycled and sent through the particle engine. This was the crucial optimizations highlighted by [Lok+92].

Under what circumstances do we think it should work well?

It should work with a reasonable number of fireworks like around ten to twelve. More than than number is a lot of particles to store. The fireworks do automatically keep cycling so the display does look really nice with around eight to ten fireworks.

Why do we think it should work well under those circumstances?

It should work well with around ten to twelve fireworks because it stores enough fireworks to look nice while also not taxing the GPU.

2 Methodology

What pieces had to be implemented to execute our approach?

2.1 Firework Particle System

Were there several possible implementations? What are the advantages and disadvantages of each?

- Two linked lists (alive and dead) as advocated for by [Lok+92]. Although moving particles would have been $O(1)$ time, efficiently sorting the linked lists for rendering optimizations is not as straightforward.
- Lists of particles and related attributes. Although removing particles in the middle of the list can take linear time, we gain by utilizing the boilerplate already provided by [Sta] and buffer optimizations provided by Three.JS [Mrd]. The slow deletion time for lists is avoided by storing another list that denotes whether a particle (or attribute) at a given index is active in the scene or not.

Which implementation(s) did we do? Why?

We used a particle system stored in lists that cycle around to emit particles for the current firework. We chose this because it was elegant to work with and made it very simple to map out the fireworks.

What did we implement?

We implemented fireworks that are colorful and interactive. Users can click any point in the scene and generate really pretty fireworks. We made the decision to have a trail that originates at the origin and then "explodes" into a firework that the click point. We could have chosen to have the fireworks terminate after one iteration but the screen looked too empty. We chose to create multiple emitters to launch fireworks on a loop.

Figure 1 and figure 2 show the two types of fireworks that we implemented.

What didn't we implement? Why not?

We did not add sound to the fireworks because we had trouble syncing up the audio to the trajectory of the fireworks. This was especially so when we had multiple fireworks going off at different times in the scene.

3 Results

How did we measure success?

We measured success by how aesthetically pleasing the fireworks were. We tried out multiple possibilities for what we wanted the fireworks to look like and we chose what looked nice.

What experiments did we execute?

We ran through the system many times and created a different number of fireworks to see if the GPU could handle it or if it caused any strange behaviors. The fireworks mostly rendered without a glitch.

What do our results indicate?

The results indicate that the system can handle many fireworks at a time while still allowing users to interact with the screen to create multiple fireworks.

4 Discussion

Overall, is the approach we took promising?

The fireworks so indeed render and look nice to the approach we took is pretty promising. Our approach is quite modular. We can keep adding new types of fireworks without affecting the existing ones.

What different approach or variant of this approach is better?

The performance of the fireworks would be better if we had decided to cap the emitters at a certain number but currently we allow any number of recurring fireworks.

What follow-up work should be done next?

To follow-up, more types of fireworks would be created with different types of emitters. [Fir] has some pretty examples of patterns that would be worthwhile to be made.

To further enhance realism, we could treat the particles as sources of light and add a height map to our scene background. This would make the fireworks lighting up have believable lighting effects on the surrounding.

What did we learn by doing this project?

Doing this project, we learned the fireworks have a lot of different components that need to be manipulated especially when you take user interaction into account. We also learned that sometimes your plans change based on how the aesthetics look. For instance, we started by inducing a firework at the very moment that a user clicks but it did not look all that nice so we added a component of the firework which originates in the middle of the scene and goes to the point at which the user has clicked and then detonates.

We also learned that a non-trivial amount of time goes into understanding an existing code-base. We had to reason through out the boilerplate provided in [Sta] to complete this project. In previous class assignments, we focused on the incomplete snippets, but this time, we modified almost every source file in the non-library folder.

5 Conclusion

Rendering fireworks was an enjoyable project to implement. There was a lot we could do with the fireworks ranging from changing the colors to the directions of the firework particles. We were able to make a lot of aesthetic choices and decide in real time what we wanted. We also added a background scene for the fireworks.

6 Appendix

Figure 1: Example scene with peony fireworks

References

[Lok+92] Teng-See Loke et al. "Rendering fireworks displays." In: *IEEE Computer Graphics and Applications* 3 (1992), pp. 33–43.

[Fir] Keystone Fireworks. *Types of Fireworks Effects*. URL: https://keystonefireworks.com/news/types-of-fireworks-effects/.

[Mrd] Mr.doob. *ThreeJS: Buffer Geometry*. URL: https://threejs.org/docs/index.html#api/en/core/BufferGeometry.

[Sta] COS 426 Staff. *Particle Systems in Animated Scenes*. URL: https://www.cs.princeton.edu/courses/archive/spring18/cos426/assign/A5/.

Figure 2: Example scene with 'rising tail' fireworks