Particle Effect Engine

- Programming 3 Project -

Nikola Kovačević
89191015@student.upr.si

May 13th, 2022

Abstract

In this report, a particle effect engine is described. A particle is a fully independent construct, it is able to compute its position and draw itself. Every particle has a directional vector and speed. Particles have a limited lifespan (time to live). The time to live is used to increase the transparency of the particle when drawn. When the particle's time to live has expired, the particle is removed. All particles start at the position of the emitter. The emitter emits new particles constantly, and particles emit vertically.

1. Introduction

A simple particle system is composed of a single emitter node, in our case this is a lane located at the center of the application, and many short-lived nodes called particles. The emitter is the one that defines where and at what rate particles are created. The particles themselves determine how they will move and when they should be removed from the scene. The visual impact is determined by how they are emitted (e.g. burst emitting or constant emitting), as well as the appearance of a given particle.

This report includes the following sections:

- **Approach:** In this section JavaFX framework and its use is explained.
- **Implementation:** In this section the code is shown and described.
- **Results:** In this section the results are represented with the pictures of the application and console logs..
- **Benchmarks:** In this section we discuss the test cases that we're given to us to test and determine the efficiency of our program.
- **Conclusion:** In this section we discuss everything we learned about the project and the test we have run in the program.

2. Approach

Particle systems are used in many video games and other animations. A particle system is a general term for any visual effect that uses a large number of small, simple, animated nodes to create a larger, complex effect. This is all implemented using the JavaFX framework. JavaFX is an open source, next generation client application platform for desktop, mobile and embedded systems built on Java. First of all we need to figure out where to put the emitter and choose how we want to emit our particles. In our case the emitter is a horizontal line located in the middle of the application screen, and the particles are emitted vertically from the emitter. We will talk about the implementation of this more thoroughly in the next chapter. Secondly, each particle needs to be able to have location, velocity, acceleration, width and height. Every single one of them, depending on the number we put into our application, needs to be able to draw itself on screen. We also need to care about the life of each particle and that can be easily seen by the color of the particle at any given moment. Lastly, we have two separate nodes that extend the Sprite abstract class, the AttractNode and RepelNode. As the names suggest, one is used to attract our particles and the second one repels them.

3. Implementation

As we mentioned previously, the application was made using Java and JavaFX framework, per the problem instructions. Here, we will briefly show the code and discuss the implementation of the following parts of the application:

- Emitter:

```
double x = ((Settings.SCENE_WIDTH / 2) +
            (random.nextDouble() * Settings.EMITTER_WIDTH)) -
            (Settings.EMITTER_WIDTH >> 1);
double y = Settings.EMITTER_LOCATION_Y;
```

This sets up the particles to be emitted from the horizontal line in the center of the application. The emitter constantly emits the particles.

- Particles:

```
// dimensions
double width = Settings.PARTICLE_WIDTH;
double height = Settings.PARTICLE_HEIGHT;
// create motion data
Vector2D location = new Vector2D(x, y);
double vx = random.nextGaussian() * 0.3;
double vy = random.nextGaussian() * 0.3 - 1.0;
Vector2D velocity = new Vector2D(vx, vy);
Vector2D acceleration = new Vector2D(0, 0);
// create sprite and add to layer
Particle sprite = new Particle(location, velocity, acceleration, width, height);
// register sprite
allParticles.add(sprite);
```

Particles implements the Sprite abstract class which extends the Region class. Every particle has a location, velocity, acceleration, width and height, as well as time to live (TTL) and a color that changes depending on TTL.
Particles are added into an ArrayList for ease of iteration over them when we are adding them on the scene.

- Attract & Repel:

```
// attract
double force = Settings.ATTRACTOR_STRENGTH / (distance * distance);
// repel
double force = -1 * Settings.ATTRACTOR_STRENGTH / (distance * distance);
```

First we calculate the direction of force, which means we need to subtract the location of either attraction or repel node, depending on which one we are using at a given moment, and then we need to get that distance, normalize the vector, and clamp using some reasonable range. At the end we use the given code to make a vector out of direction and magnitude.

3.1. Run modes

3.1.1. Sequential

Adding of particles and their force was done with enhanced for loops, and the use of built in forEach() function for ArrayList. The same method is used for the acceleration and location of particles, as well as for their removal from the scene.

Here's an example of how it looks like in the code:

```
allParticles.forEach(sprite -> sprite.applyForce(Settings.FORCE_GRAVITY));
for (AttractNode attractor : allAttractors) { // apply attract
   allParticles.forEach(sprite -> {
                Vector2D force = attractor.attract(sprite);
               sprite.applyForce(force);
        });
}
```

3.1.2. Parallel

Using Threads and parallel streams, this was the most efficient way to implement parallelism into the JavaFX framework and the results and benchmarks reflect on that as it will be seen in the two sections.

In total there were 5 Threads running and additionally all the force (attract and repel) was added to particles via parallel streams, as well as application of acceleration and calculation of velocity and location of each particle.

Here's an example of how it looks like in the code:

```
// apply attract
Thread thread2 = new Thread(() -> Platform.runLater(() -> {
// apply gravity
   allParticles.stream().parallel().forEach(sprite ->
sprite.applyForce(Settings.FORCE_GRAVITY));

   for (AttractNode attractor : allAttractors) {
         allParticles.stream().parallel().forEach(sprite -> {
               Vector2D force = attractor.attract(sprite);
               sprite.applyForce(force);
         });
         }
}));
thread2.start();
```

4. Result

The final application version looks like this:

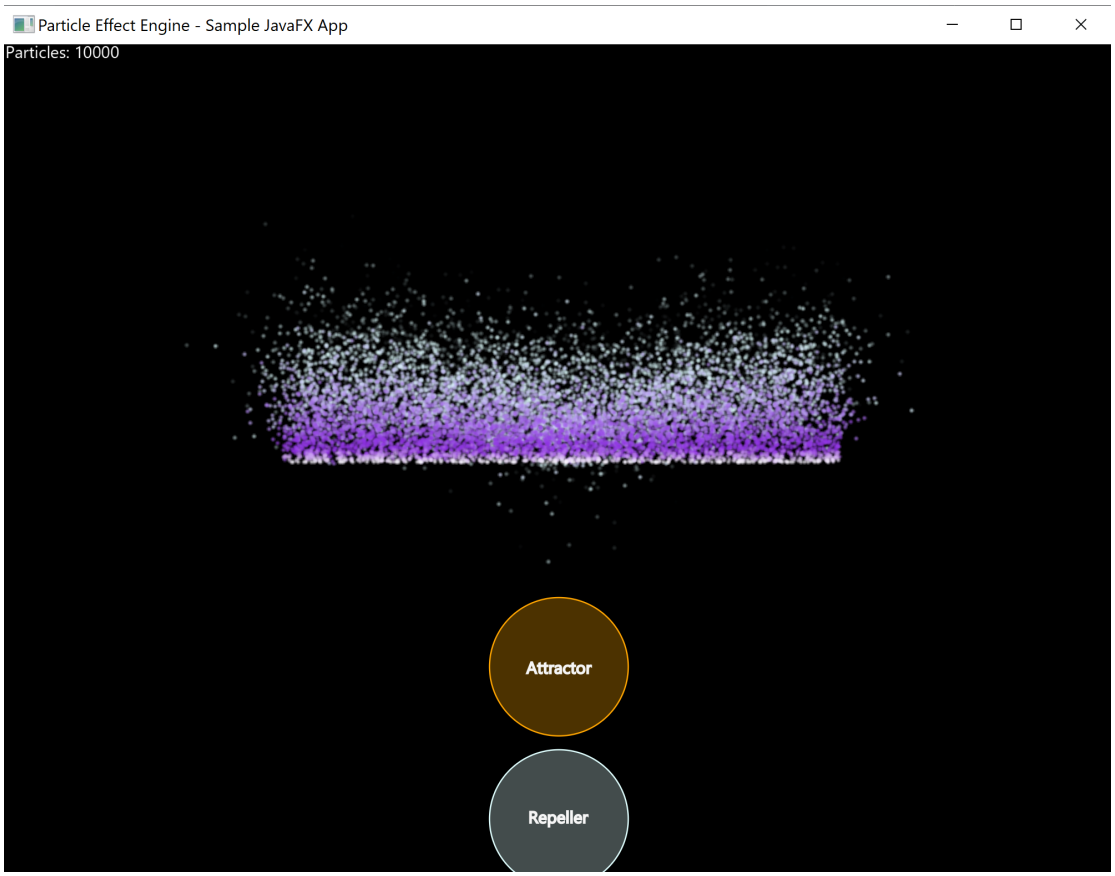

Figure 1: The Application Screen

It consists of the particle emitter, and two nodes: Attraction and Repeller. In the upper left corner we have the frames per second (FPS) counter, and its values are stored into an ArrayList, where at the end of the program execution we calculate the average FPS of the application. This calculation is important for the next section, the benchmark section.

5. Benchmark

In this section we show the results of the tests we conducted using graphs and tables. The first test starts at 100 particles and we increase it by a 100 more with each iteration, until we get to 10000 particles. We ran the program for exactly 10 seconds with each iteration and stored the average FPS in the excel sheet.
The second test is all about testing the limit of the performance of the application. We start from 1000 particles and we keep increasing by 1000 until we get a significant drop in the average FPS. We concluded the testing at 31 iterations.
All tests were performed on a machine with 32GB of RAM and 6-core (12 threads) Intel(R) Core(TM) i7-8750H CPU.

5.1. Test with up to 10000 particles

We'll show the data of the last 20 iterations of the program and the graph that consists of data of all 100 iterations.

5.1.1. Sequential

Iteration	Number of Particles	Average FPS
81.	8100	58.889
82.	8200	58.667
83.	8300	58.667
84.	8400	58.667
85.	8500	58.667
86.	8600	58.778
87.	8700	58.778

88.	8800	58.667
89.	8900	58.667
90.	9000	58.889
91.	9100	58.667
92.	9200	58.556
93.	9300	58.667
94.	9400	58.667
95.	9500	58.778
96.	9600	58.778
97.	9700	58.667
98.	9800	58.667
99.	9900	58.667
100.	10000	58.667

5.1.2. Parallel

Iteration	Number of Particles	Average FPS
81.	8100	58.889
82.	8200	58.889
83.	8300	58.556
84.	8400	58.667
85.	8500	58.667
86.	8600	58.778
87.	8700	59.112

88.	8800	58.778
89.	8900	58.778
90.	9000	58.778
91.	9100	58.889
92.	9200	58.889
93.	9300	58.778
94.	9400	58.778
95.	9500	58.889
96.	9600	58.778
97.	9700	58.778
98.	9800	58.778
99.	9900	58.778
100.	10000	58.667

As we can see from the Average FPS from these two tables the difference in average frames per seconds is negligible. Here's the graph containing the data from all 100 iterations:

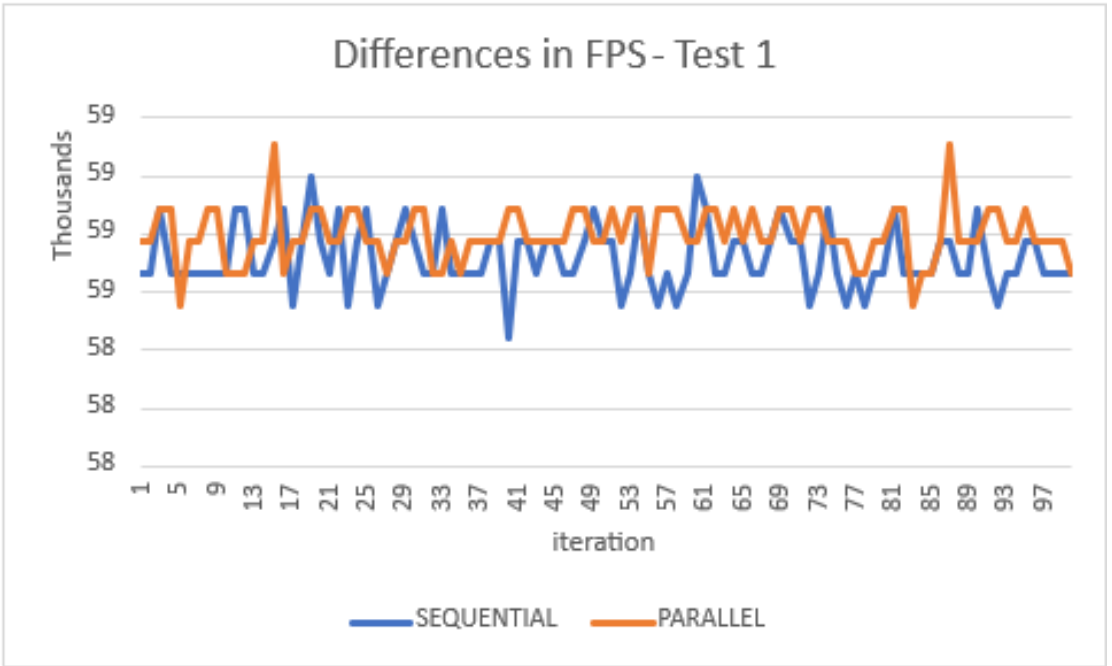

Figure 2: The Differences in Average Test 1

The average of 100 iterations:

SEQUENTIAL AVG	PARALLEL AVG
58.720	58.801

5.2. Test with over 10000 particles

5.2.1. Sequential

Iteration	Number of Particles	Average FPS
12.	21000	57.667
13.	22000	54.112
14.	23000	50.334
15.	24000	47.445
16.	25000	46.889
17.	26000	44.667
18.	27000	42.778
19.	28000	42.445
20.	29000	42.223
21.	30000	41.778
22.	31000	40.334
23.	32000	39.112
24.	33000	39.334
25.	34000	37.778
26.	35000	37.112

27.	36000	36.334
28.	37000	35.112
29.	38000	34.445
30.	39000	33.667
31.	40000	33.556

5.2.2. Parallel

Iteration	Number of Particles	Average FPS
12.	21000	58.667
13.	22000	58.556
14.	23000	58.556
15.	24000	58.667
16.	25000	58.445
17.	26000	58.445
18.	27000	58.667
19.	28000	58.445
20.	29000	58.556
21.	30000	58.556
22.	31000	58.334
23.	32000	58.445
24.	33000	58.445
25.	34000	58.445
26.	35000	58.445

27.	36000	57.778
28.	37000	57.778
29.	38000	57.889
30.	39000	57.445
31.	40000	57.778

In the second test we get a completely different story. Sequential performance starts to drop off heavily when we hit around ~24000 particles, while parallel performance has negligible difference constantly throughout the test.

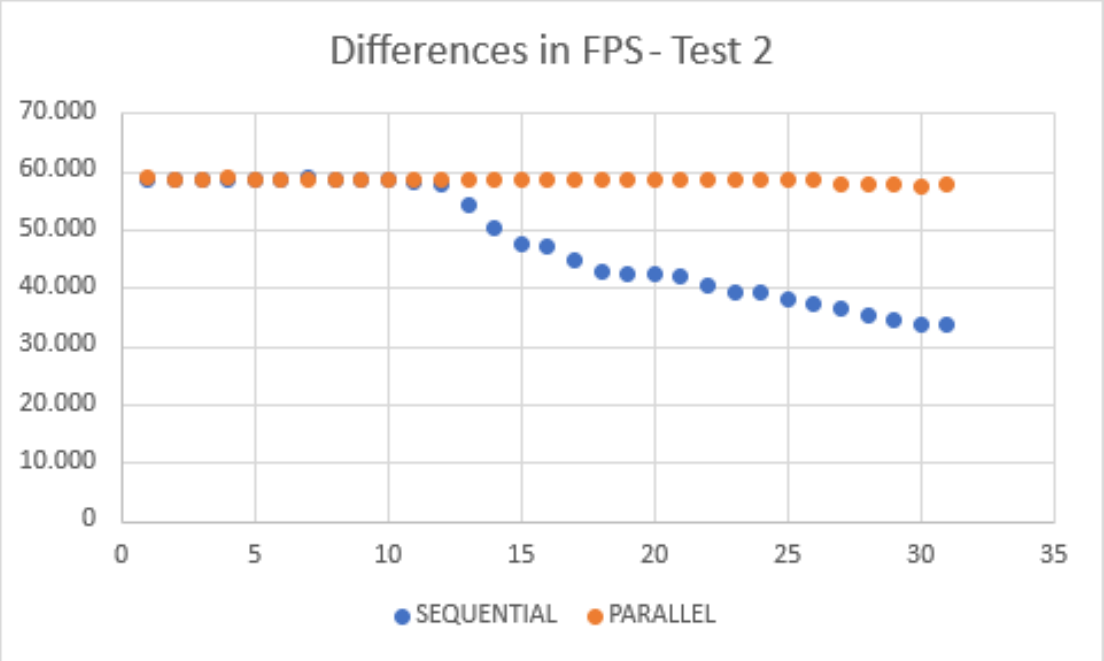

Figure 3: The Differences in Average Test 2

The average of 31 iterations:

SEQUENTIAL AVG	PARALLEL AVG
47.782	58.441

6. Conclusion

We can conclude that this implementation has no issues running efficiently in both modes while we are working with a relatively small amount of particles. This shifts when we cross the border of 10000 particles and start increasing by 1000 each time we run the program. The sequential mode has significant drop off in performance, while parallel mode doesn’t experience any major drop off and runs fluidly.