Particle Effect Engine

Miloš Mamić
89201057@student.upr.si
University of Primorska
UP FAMNIT
Koper, Slovenia

ABSTRACT

In this paper we will be talking about Particle Effect Engine.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 technique produces animations that are hard to reproduce using a single vector, raster image, or other method. Examples of particle systems include fire, magic spells, sparks,moving water etc. Implementation will be done sequentially and parallel by using JavaFX in 2D space .

KEYWORDS

particles, emitter, repeller, attractor, particle force field

ACM Reference Format:
Miloš Mamić. 2023. Particle Effect Engine. In *Proceedings of (UP FAMNIT).* UP FAMNIT, Koper, Slovenia, 4 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

A particle is a completely autonomous component in the system. It is able to compute its position and make a picture of itself. Every particle has a speed and a directed vector. There is a finite lifetime that the particles have and it is used to to increase the transparency of the particle when drawn. The removal of a particle occurs when its TTL (time to live) has finished. At the location of the emitter, all particles are launched. Depending on the effect type (burst all particles at once, emit n particles every second, etc.), the emitter releases new particles. Particles collide and bounce off the window, which also acts as a bounding box. Moreover, the particles adhere to fundamental physics principles (gravity, collision, etc).

A particle system is a collection of several particles that operate according to the same law overall yet exhibit random differences in their properties. [3] Particle systems are a technique for modeling a class of fuzzy objects. In his paper titled "Particle Systems—a Technique for Modeling a Class of Fuzzy Objects," William T. Reeves introduced the idea for the first time in 1983. He then applied particle systems to realize the production of special effects like flame and fireworks. [2] Since then, different fuzzy scenario and natural phenomena have been simulated using particle systems in 2D and 3D computer graphics processing. With the advancement of virtual reality technology, the virtual engine's particle system also blossoms with unmatched color, which is essential for producing realistic scenarios.[5]

UP FAMNIT, 2023, Koper, Slovenia

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

2 IMPLEMENTATION

The next thing we want to do is discuss is how particle systems work. A particle system has three major components: particles, a particle emitter, and the engine itself[1]. A particle is a small point in your game that will be drawn with an image. A particle often has a position in the game world, as well as a velocity, an angle that it is rotated at, an angular velocity which indicates how quickly the particle is spinning, and usually a color and texture for drawing. In our particle system, we will have lots of particles, which all add up to produce an interesting effect. Also, particles almost always have some way of determining how long they should live for, so the particles can be removed as new ones are generated.A particle emitter is essentially the location that the particles are coming from. Often the particle emitter is also responsible for determining how many particles will be created at any given time.The particle system manages the state of the previous two components and is responsible for removing dead particles from the system.

2.1 Particle basics

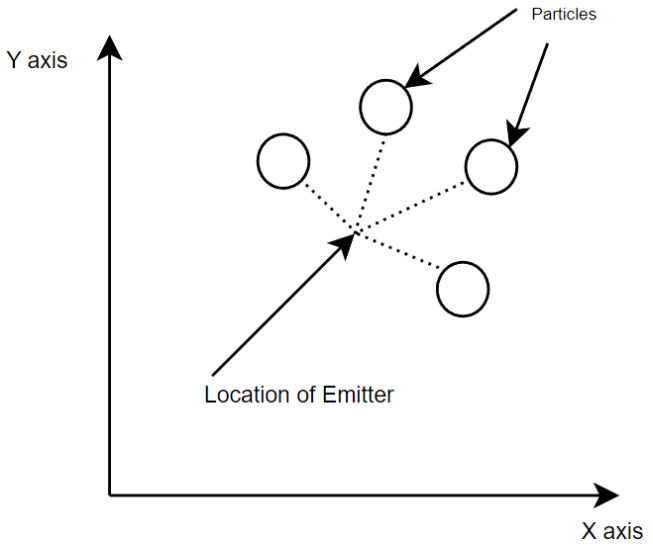

Figure 1: Particles are created at the location of the emitter(which is not visible),then travel away from the emitter.

?? The simplest particles move in a straight line and disappear after a while. but it is possible Many variations of this behavior. For example, particles can be fast at first and slow down as they arrive end of their life cycle. Particles may also become transparent or change color as they age. Creating particles that do not move in a straight line can be a powerful tool when creating effects. One way is to have the particles "fall" to the ground, like sparks from a fireworks display in the air o r the particles might move erratically, like air bubbles in water splashing to the surface. One of the most important aspects of how particles look is how they interact with each other. The first example uses only opaque particles, while the later examples show how partially transparent particles are used. Or particles combined with blending effects produce eye-catching results. [6]

```
private void addParticle() {
        // starting point where the particles are emmited
        double x = Settings.SCENE_WIDTH / 2 + random.
    nextDouble();
        double y = Settings.EMITTER_LOCATION_Y;

        // dimensions of the particles
        double width = Settings.PARTICLE_WIDTH;
        double height = Settings.PARTICLE_HEIGHT;

        //gravity
        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 the pixel and add it to the layer
        Particle pixel = new Particle(location, velocity,
     acceleration, width, height);
        allParticles.add(pixel);  }
```

Listing 1: Sample code of particles

?? Particle systems show their strengths when the particles mix together on the screen. To achieve a rudimentary mixing effect, we can adjust the transparency of the particles. This creates a more compelling visual effect as the partially transparent particles overlap in random ways to produce regions of random shape and color density. The nextGaussian() method returns random numbers with a mean of 0 and a standard deviation of 1. Remember, that this means that numbers returned by nextGaussian() will tend to "cluster" around 0 with the above shape, and that (approximately) 70 % of values will be between -1 and 1. Based on the values returned by nextGaussian(), we can scale and shift them to get other normal distributions by changing the maen (average) of the distribution, we add the required value and to change the standard deviation, we multiply the value.There is theoretically no absoluate minimum and maximum value that can occur in a normal distribution. In practice with Random.nextGaussian(), there will be some actual minimum/-maximum, but not necessarily where we want it to be. The actual minimum/maximum is determined by the fact that nextGaussian() returns a double, and by the underlying psuedo-random number generator that in most cases will have some fixed sequence length (so there'll be some actual minimum or maximum out of all the possible numbers it could generate, if you're able to sit and wait for it to generate them all).

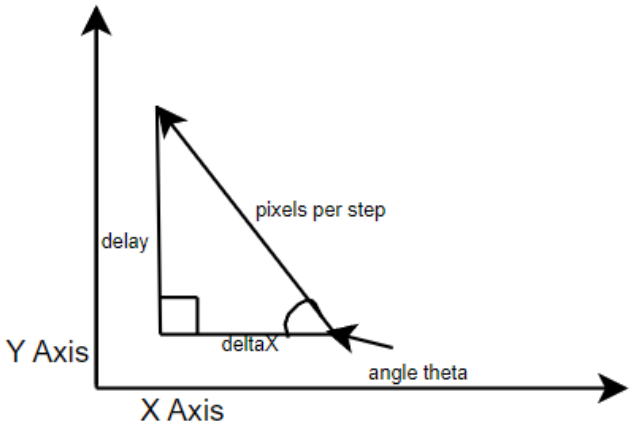

Figure 2: Calculating random direction deltaX=cos(theta) x pixels per step; deltaY=sin(theta) x pixels per step

2.2 Sequential implementation

??This transition is used to apply the list of animations on a node in sequential order (one by one). Sequential Transition is important in designing a game which animates its entities in sequential order.

In our case we use loops for adding particles and the force field is used to represent sequential transition. To remove the particles and accelerate the particles we will be utilising the function **forEach()**. These animations will be applied on the node in the sequential order.

2.3 Parallel implementation

?? Normally any java code has one stream of processing, where it is executed sequentially. Whereas by using parallel application, we can divide the code into multiple streams that are executed in parallel on separate cores and the final result is the combination of the individual outcomes. The order of execution, however, is not under our control.[4] Therefore, it is advisable to use parallel application in cases where no matter what is the order of execution, the result is unaffected and the state of one element does not affect the other as well as the source of the data also remains unaffected. Parallel Streams[7] were introduced to increase the performance of a program, but opting for parallel implementation isn't always the best choice. There are certain instances in which we need the code to be executed in a certain order and in these cases, we better use sequential way to perform our task at the cost of performance. The performance difference between the two kinds of applications is only of concern for large-scale programs or complex projects. For small-scale programs, it may not even be noticeable. In our example we will be having four threads which each is assigned with a specific task as adding new particles, reppeler force,draw all particles on canvas,removing all the particles. On Figure 3 it is shown that the parallel execution handles better then sequential, as expected.The difference is approximately 30 to 40 percent in constant time.

3 RESULTS

After the implementation of the code we have done testing between the implementation of sequential and the parallel part. After each iteration there is a increase in the number of generated particles that occur.The parallel implementation is configured to be executed using four threads.During the execution of our program,in Figure 3(??) we will have an upper bound of 300 particles per iteration that are added on the canvas which has size of 800x600.On the graphs shown each pivot represents an average of 10 iterations.

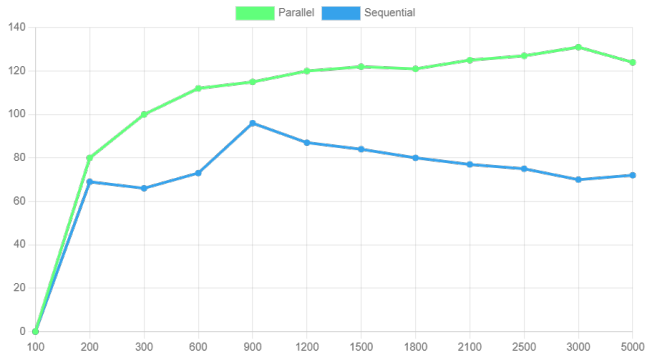

Figure 3: Comparing FPS(frames per second) and number of particles between sequential and parallel execution (up to 5000 pixels)

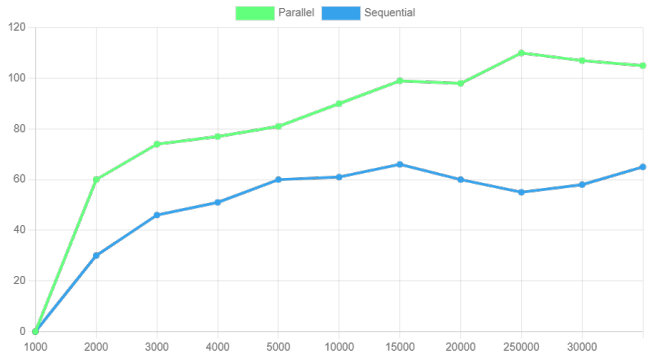

Figure 4: Comparing FPS(frames per second) and number of particles between sequential and parallel execution(up to 30000 pixels)

In the Figure 4(??) we can see how on larger scale it is sufficiently visible the difference between the executions. Here we have 1000 particles added on the same canvas per iteration. After the comparison of our experiments the final results show that there is a 30 to 40 percent difference between the two. We have to take in consideration that this test was done on a laptop with a Ryzen 5 5000 series processor and 8GB of RAM, these results can vary depending on the components used during testing This just shows on a small scale how big of an impact can multi threading have.

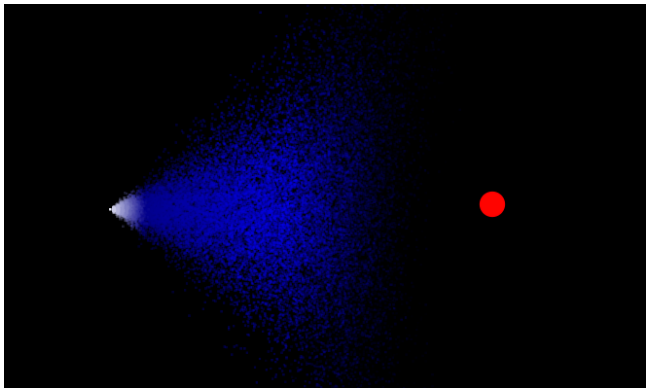

Figure 5: Implementation of JavaFxParticleEffectEngine

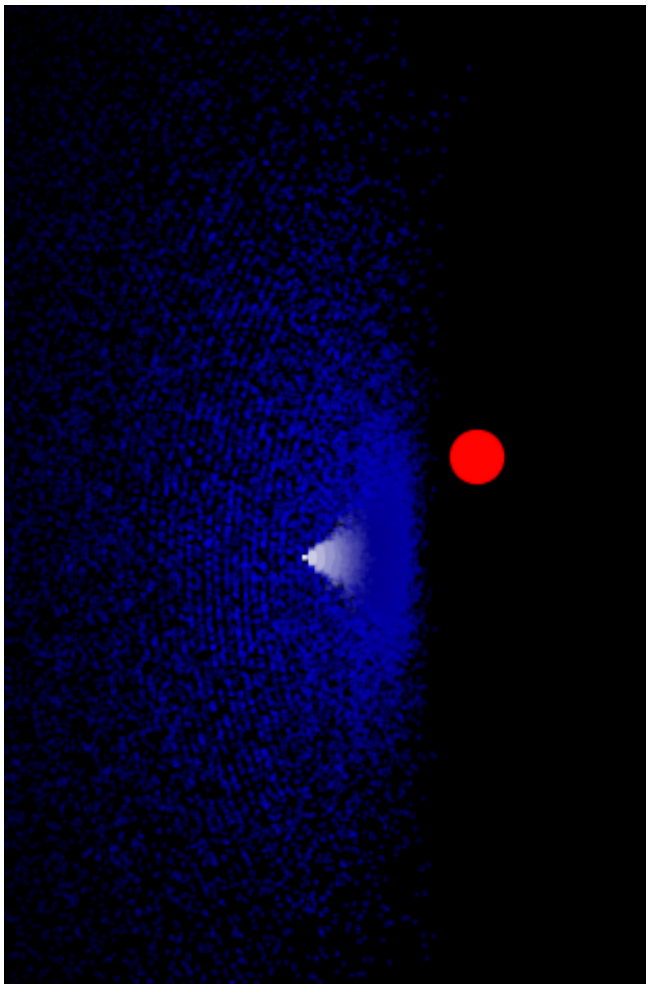

Figure 6: Reppeler node which reppels the particles that are emmited away

4 FURTHER CONSIDERATIONS

The examples in this chapter demonstrate how to create a fundamental particle system by utilizing some of JavaFX graphical capabilities. Several games or other visually appealing apps might leverage these examples. Yet, it is possible to mix the characteristics of emitters and particles in many methods to create distinctive visual effects. Below are some more characteristics that might be included in a particle system to create more alternatives. There is no need for all particles to be identical, which is the case with heterogeneous particles.Combining particles of different shape, color, or even duration can open the door to many new possibilities.Another technique to alter the appearance of an effect is to change the percentage of each particle type Another characteristic would be a moving emitter.The emitter may be moved and animated just like any other node since it is a node. A snake-like effect will result by moving an emitter that emits slowly moving or even completely stationary particles. To produce a comet-like effect, a moving emitter that produces fast-moving particles can be employed. We can also consider an animated emitter.It provides a number of attributes on the Emitter class that define the attributes of each new particle.The results produced over time would be interesting to see after the attributes on the emitter were changed.Moreover, you might gradually change the color of the particles to represent a particular application event.In the context of the game, for instance, a particle system could turn from blue to red as the player loses health.

5 SUMMARY

We talked about how to implement a particle system in JavaFX, and how the features of a particle system affect its look and its performance depending on if we use parallel or sequential execution.Not using parallel is slower for bigger data sets.The complexity of the particle material, the number of generated particle emitters, the number of particles,the distance between the particle effect and the screen, collision and other factors will affect the performance consumption of the particle effect. In practical application, the application conditions of particle effect should be analyzed, and appropriate optimization methods should be selected to achieve a balance between the effect of particle system and various performance consumption parameters.

REFERENCES

[1] Naol Getachew. *Chapter-3 JavaFX*. 2022.
[2] CYu Lichun J. *Research on the Application of Movie Particle System*. 2015.
[3] WILLIAM T. REEVES J. *Particle Systems-A Technique for Modeling A Class of Fuzzy Objects*. 1983.
[4] Byeong Man Kim In Sang Chung Byeong Man Kim Yong Rae Kwou. *The design and implementation of automata-based testing environment*. Kumoh National Institute of Technology, 2002.
[5] Yao Liang. *Technical Basis of Virtual Engine*. 2018.
[6] Foreword by Ryan Donahue Lucas L. Jordan. *JavaFX™ Special Effects: Taking Java™ RIA to the Extreme with Animation, Multimedia, and Game Elements*. apress, 2009.
[7] Jim McLaughlin. *Sequential vs Parallel Streams*. 2019.