**SWE 585: Game Programming**

**Term Project – Particle System Optimization**

**Report**

**Feridun Berk Akyol**ID: 2022719192

Date: 14.12.2024

**Introduction to Unity’s particle system:**

The Unity Particle System is a versatile and powerful tool for creating dynamic and visually appealing effects in real-time applications, such as video games and simulations. By simulating the behavior of small particles, the system enables the creation of effects like **fire, smoke, explosions, and snow**. These particles are lightweight objects that can interact with the environment, physics forces, and custom behaviors to achieve highly realistic or stylized visuals.

Unity's Particle System offers extensive customization, allowing developers to control aspects such as particle shape, color, size, velocity, and lifespan. It also includes features like emitters, which determine how and where particles are generated, and modules for applying advanced behaviors like collision and lighting.

However, the flexibility and complexity of particle systems can **impact performance**, especially when dealing with high particle counts or complex interactions. This makes optimization a critical step in ensuring the system remains efficient while delivering the desired visual fidelity. This report explores the current performance challenges of Unity’s Particle System and proposes strategies for optimizing its implementation.

**Components of the Unity’s particle System**

**1. Emitter (Main Module)**

Controls the particle system's core properties, such as duration, looping, and particle behavior over time.

Key Parameters: Start Lifetime, Start Size, Start Speed, Start Rotation, Start Color, Start Delay.

**2. Emission Module**

Manages the rate and bursts of particle emission.

Key Parameters: Rate over Time, Rate over Distance, Bursts.

**3. Shape Module**

Defines the shape and area from which particles are emitted.

Key Parameters: Shape Type (e.g., sphere, cone, box), Radius, Arc.

**4. Velocity over Lifetime Module**

Modifies particle speed and direction over time.

Key Parameters: Linear and Orbital Velocity, Radial Speed.

**5. Force over Lifetime Module**

Applies constant forces to particles during their lifespan.

Key Parameters: Linear and Random Force, Space (Local/World).

**6. Color over Lifetime Module**

Changes the color of particles over their lifetime.

Key Parameters: Gradient, Alpha Transparency.

**7. Size over Lifetime Module**

Adjusts the particle size dynamically as they age.

Key Parameters: Size Curve, Random Multipliers.

**8. Rotation over Lifetime Module**

Controls how particles rotate during their lifetime.

Key Parameters: Angular Velocity, Rotation Curve.

**9. Texture Sheet Animation Module**

Animates particles using a texture atlas or sprite sheet.

Key Parameters: Frame over Time, Cycle Count, Animation Mode.

**10. Renderer Module**

Handles the rendering of particles in the scene.

Key Parameters: Render Mode, Material, Sorting Layer, Camera Distance Fade.

**11. Collision Module**

Allows particles to interact with physical objects.

Key Parameters: Dampen, Bounce, Collision Mode (World/Planes).

**12. Sub Emitters Module**

Generates new particle systems from the original particles.

Key Parameters: Spawn on Birth/Collision/Death, Sub Emitter Templates.

**13. Noise Module**

Applies turbulence to particle movement.

Key Parameters: Strength, Frequency, Octaves, Scroll Speed.

**14. Trails Module**

Adds trails to particles for effects like smoke or light streaks.

Key Parameters: Lifetime, Color over Trail, Ribbon Mode.

**15. Lights Module**

Attaches light sources to particles for dynamic lighting effects.

Key Parameters: Ratio, Intensity, Range, Bounce Intensity.

**16. Custom Data Module**

Allows custom scripting and parameters for advanced behavior.

Key Parameters: Vector/Color Options.

By understanding these modules and their parameters, we can fine-tune particle systems to achieve the desired visual effects while balancing performance.

A screenshot of a computer

Description automatically generated

Fig 1. View of the particle system components under unity inspector.

**Creating a performance problem in Particle System**

It’s common for modern hardware to handle even exaggerated particle systems with ease, especially when using a single system. To significantly impact the performance, we will need to combine multiple high-cost settings. Here are some strategies that we will use to create a load on the system.

1. **Using Multiple Particle Systems:**
   * Adding 10–20 systems to a scene, each with high emission rates and lifetimes, and combining them to simulate large-scale effects like rain, firestorms, or explosions. We will write a pooling script to spawn multiple systems simultaneously.
2. **Overdraw Stress Test:**
   * Using a high number of particles and emission rates with **large sizes** overlapping in the camera view. This will force the GPU to render many pixels repeatedly.
3. **Enable Physics:**
   * We will add collision detection to the particle system and simulate complex particle physics to burden the CPU.
4. **Add Sub-Emitters:**
   * Using multiple cascading sub-emitters with high emission rates to exponentially increase particle counts.
5. **Texture Overhead and Animations:**
   * We will use textures with different resolutions, and we will utilize the texture sprite sheet animation module for creating animations for each particle at different frame rates, especially if the textures use alpha transparency, it will further stress the GPU memory.
6. **Dynamic Properties:**
   * Dynamically changing some particle parameters (e.g., color, size will also increase the computation overhead.

**Creating a particle pooling script:**

**A screen shot of a computer program

Description automatically generated**

Fig2. Properties and Start method of our particle pooling script. (full script is available in the project repository on Github)

This script enables us to spawn multiple particle systems on a predefined grid. Here is how it looks like in the unity inspector. By attaching this script to a spawning point we will be able to spawn multiple systems all at once to better observe the performance impacts on the system. We will use this script every time we want to spam particles to be sure that every test has the same number of particle systems at the same place.

**A screenshot of a computer

Description automatically generated**

Fg3. View of the pooling script parameters on unity inspector.

**Comparing particles with different texture resolutions and animation frame rates:**

To compare the effects of textures with different resolutions and frame rates on performance we will make sure that all the other parameters such as emission rates, collusion, max particle count etc are all same between the two systems and only difference is the texture resolution and frame rates for texture sheet animations.

Please note that the camera position is also fixed and the same for all tests to be sure that all the test conditions are the same, and we are not affected by the other objects in the scene. Since closing all the other objects is not a realistic scenario it’s expected to have lots of game objects in a single scene.

A group of white circles on a black background

Description automatically generated

A group of smoke in different shapes

Description automatically generated

Fig4. Smoke Textures 15 FPS 256x256

Fig5. Smoke Textures 45 FPS 256x256

The smoke textures with **256x256** resolutions can be seen above. I also upscaled both 45 and 15 FPS textures to **1024x1024** resolution (4x increase in resolution per particle) to test the effects of texture resolution on performance.

Fig.6 AVG FPS of different sprite sheet animations with different resolutions

Fig7. Memory Usage of the particle systems using different texture resolutions and animation frame rates

**Conclusion**

As can be seen on the charts the performance impact of the texture sheet animations is amplified proportionally to the resolution of the textures. So, in low resolutions even if we use different frame rates for the animations the performance impact is negligible and almost non-existent but as we increase the resolution of the textures (in our case 4x upscaling from 256x2516 pixels from 1024x1024 pixels). The performance impact drastically increases.

**Why is the performance impact being small in lower resolutions?**

At low resolutions, texture sampling is lightweight:

* It fits into GPU texture cache easily.
* The fill rate demands are small.

Hence, even with high FPS animations, the GPU has no significant bottlenecks processing low-res textures.

The performance impact of texture sheet animations increases with texture resolution due to three main reasons: fill rate bottlenecks, where the GPU processes more pixels (texels) per particle and struggles with overdraw; cache inefficiency, as higher-resolution textures exceed the GPU's texture cache, causing slower memory fetches; and increased memory bandwidth usage, since larger textures require more data to be read and processed. At low resolutions, these issues are negligible because textures fit into the cache easily, and the GPU can efficiently render particles without significant overhead.

Unity profiler data is included below for more detailed information.