Implementation and Optimization of Particle Effects based on Unreal Engine 4

Juan Zhang
Beijing Information Technology College No.5 Fangyuan West Road Chaoyang District Beijing, China
Email: zhangjuan@bitc.edu.cn

Abstract. The wide application of virtual reality and augmented reality makes the engine of developing virtual reality develop towards more realistic rendering, more convenient operation and better performance. In the realization of virtual reality scene, the three-dimensional particle effects play an excellent performance, and enhance the immersion of the experiencer. However, the production and optimization of three-dimensional particle effects have always been a complicated task for developers. Based on the visual blueprint script of Unreal Engine 4 and the modular particle editor, the author elaborates the method of making particle effects in detail, calculates the cost of particle system in real time by using the console of Unreal Engine 4, and then analyzes the reasons that affect the performance of particle effects according to the data. Three methods, GPU launcher, material instruction reduction and LOD application, are used to optimize the performance of particle effects, and data are used to show the results of optimization.

1. Introduction
Particle system is a collection of a large number of particles with the same law of performance as a whole, but with different characteristics at random. [1] In 1983, William T. Reeves put forward the concept of particle system for the first time in his paper entitled "Particle Systems—a Technique for Modeling a Class of Fuzzy Objects", clearly explained the key parameters and meaning of new particles, and applied particle system to realize the production of special effects such as flame and fireworks. [2] Since then, particle system has been widely used in 2D and 3D computer graphics processing to simulate various fuzzy scenes and natural phenomena. With the development of virtual reality technology, particle system also blooms unparalleled color in the virtual engine, which plays a huge role in creating real scenes.

Unreal Engine 4, developed by Epic Games, is one of the most widely licensed virtual reality game engines in the world. Unreal Engine 4 has been favored by more and more users for its advantages of visual blueprint script system, high-quality picture effect and convenient operation. [3] Therefore, based on the development platform of Unreal Engine 4, this paper studies the implementation and optimization of particle system in the engine.

2. Making Snow Particle Effect Based on Unreal Engine 4
The particle system of Unreal Engine 4 is based on modular management. Each module represents a specific aspect of particle behavior and provides specific attribute parameters, such as color, generation position, movement behavior, scaling behavior, etc. You can add modules as needed, and further define the parameters of particle behavior described by the module. [4] The particle effects editor provides real-time feedback, making it easy for users to set parameters. Cascading the different attribute modules to form particle effects can make the production process faster and easier even for...
very complex effects. In the process of cascade module editing particle effects, the system only needs to create the most basic attribute and behavior module to describe the particle effects, without the participation of unnecessary attribute variables, so the system will not generate additional calculation data. At the same time, these modules can be created, deleted and copied according to user requirements.

2.1. Editing Particle Material
The material of particles determines the appearance of particle effects. In the material editor of the Unreal Engine 4, using nodes and connection methods as shown in Figure 1, we can create a material with a white base color and a dashed edge.

![Particle material](image)

**Figure 1.** Particle material

2.2. Creating Particle Emitter
When particle effect is applied in a level, it needs to be referenced by particle emitters. Different emitters define the effect used by different particles in the scene. First, you need to create a particle system asset in the content browser of Unreal Engine 4. This case uses the basic emitter sprite emitters, which is also the most widely used emitter. The emitter is always facing the camera, and it can be used to make smoke, fire and other effects.

2.3. Editing Particle System Modules and Parameters
In the particle emitter, add the basic module attributes that define the particle effect. The modules and related parameter settings used in this particle effect are shown in Table 1.
Table 1. Modules and parameters of snow particle effect

| Module name  | Particle attribute | Parameter value |
|--------------|--------------------|-----------------|
| Required     | Material           | M_white         |
| Spawn        | Rate               | 5000            |
| Lifetime     | Lifetime           | Min 30; Max 300 |
| Initial Size | Start Size         | Max 20,0,20,0; Min 2,2,0 |
| Initial Velocity | Start Velocity | Max 0,0,-300; Min 0,0,-100 |
| Initial Color | Start Color        | R 1.0; G 1.0; B 1.0 |
| Initial Location | Start Location   | Max 2000,2000,2000; Min 0,0,0 |
| Actor Collision | Collision Response | Freeze         |

In the above modules, the required module and spawn module are permanent modules, which are the parameters necessary for particle effects. Other modules can be deleted or added as needed. The parameter settings under the module are set in the detail panel according to the requirements. In the preview window, you can see the effect of parameter settings in real time. After editing the particle system, add it to the level and apply it.

3. Optimizing Particle Effects Based on Unreal Engine 4

As everyone knows, particle effect takes up memory, and the amount of data calculation is very large. When running a scene, it is very easy to cause a jam. Take the snow effect as an example. The computer is configured as Core i7 CPU @ 3.6GHz, memory 8G, graphics card GTX 750ti. When the snow particle effect is loaded, it will still appear the phenomenon of jamming.

3.1. Analyzing the Cost of Particle System

In Unreal Engine 4, you can call commands to view and analyze the performance overhead of computer. In the running scene state, call out the console and enter "stat unit" to detect and display the time spent on the game thread, rendering thread and GPU of the current frame.

![Figure 2. Cost of particle system](image)

Frame time is the time needed to generate a frame. Because the logical thread (Game) and the rendering thread (Draw) need to synchronize at the end of a frame, the time needed to generate a frame is often similar to that of one of the threads. GPU time is the time required by the graphics card to render the current scene. Because GPU time is synchronized with the current frame, GPU time is basically the same as frame time. If the frame time is close to the logical thread time, the bottleneck appears in the logical thread; on the contrary, if the frame time is close to the rendering thread time, the bottleneck appears in the rendering thread. If both times are not close, the bottleneck appears on the graphics card. Through this data, we can preliminarily determine the cause of bottleneck. As shown in Figure 2, frame time is close to GPU time, so the operation of the scene will be affected by the performance of the graphics card. This is related to the snow particle effect used in the scene.

Enter the "stat particles" command in the console to estimate and display the time spent by particles on different threads, as shown in Figure 3. The particle operation time, emitter emission time, particle collision time and particle update time marked in red in the figure are relatively long, which are the reasons why particle effects affect performance.
3.2. Analysis of the Reasons for the High Cost of Particle Effect

When you add a material to the surface of a sprite, that material has a cost in instruction count. For translucent objects, this instruction cost adds up as translucency stacks up, creating overdraw. The more layers of translucency, the higher the cost. In its simplest form, overdraw can be described as:

\[ \text{Overdraw} = \text{Pixel shader cost} = \text{number of layers} \times \text{average number of pixels affected by a layer} \times \text{average number of instructions for a layer} \]

In the above snow particle effect, each emitter uses about 5000 * 121 * 900 = 544.5 million instructions. In order to achieve the effect, the scene uses a total of 20 transmitters, processing about 1089 million instructions per second, which is undoubtedly a huge overhead.

Tick Time represents the amount of time spent updating particle systems in the scene. Tick Time is directly influenced by the number of active emitter actors in the scene. The more active emitters in the scene, the higher Tick Time will be.

3.3. Optimization of Particle Effect

The fundamental principles of particle effect optimization are: reducing the number and range of particle systems used in the scene; reducing the number of particle emitters; reducing the number of particles emitted by emitters; reducing the life time of particles in the scene; reducing the calculation of behaviors such as collisions that can consume a lot. In Unreal Engine 4, you can optimize the performance of particle effects by the following methods.

3.3.1. Use GPU emitter to reduce emissivity. Using CPU to generate tens of thousands of particles can lead to performance degradation. In scenes with a large number of particles, GPU emitters can be used to simulate and render hundreds of thousands of particles more easily and efficiently.

Modify the snowy particle emitter as "GPU sprite". Since the applicable modules of GPU emitter and CPU emitter are not the same, it is necessary to modify individual functional modules. The particle effect based on GPU emitter is added to the scene. It is found that one GPU particle emitter can achieve the effect of 20 CPU particle emitters, which not only reduces the number of particle emitters, but also greatly reduces the performance overhead of particle system, as shown in Figure 4. Using GPU particle emitter instead of CPU particle emitter is a very effective way to optimize the performance of particle effects.

3.3.2. Reduce the number of material instructions. Different materials of particles need different instructions. The more complex the materials are, the more instructions they need. In the material
editor of Unreal Engine 4, press the "Alt + 8" shortcut key, and you can see the cost in the opened shader complexity mode. The snowflake material needs about 2000 instructions. After adjusting the white snowflake material with self illumination and translucency attributes to a white material without special effects, the performance of particle effect has also been significantly improved, as shown in Figure 5. Therefore, it is a feasible method to optimize the performance of particle effects to use materials with low instruction number as much as possible.

Figure 5. Time spent by simple material particles

3.3.3. Create level of detail (LOD) to reduce costs at different view distances. The optimization of particle effect also needs to consider the distance between particles and players and the actual effect of particle effect. For example, if the player is near a campfire, he can see the embers and sparks of the fire in the smoke. However, if the observation point is hundreds of meters away, and these embers and sparks are actually invisible, then the calculation of the embers and sparks is unnecessary. This can be controlled by level of detail. The LODs system can customize the distance range, and the particle system will automatically simplify after a certain distance is exceeded. Each distance range corresponds to a LOD. The simplified intervention is to disable some modules or use lower values, or even disable some transmitters.

In this case, the level of detail is set for snow particle effect based on CPU transmitter. When the player is at the lowest LOD level in a long distance, the particle system reduces the number of emissions to one tenth of the original number, and cancels the collision setting. The consumption during running the scene is shown in Figure 6. The performance of particle effect in rendering time and other performance has also been significantly improved.

Figure 6. Time spent by particles using LOD

4. Conclusion
Based on the modular parameter management and real-time effect display, the particle editing system of Unreal Engine 4 makes it more convenient to make particle effects in virtual reality engine. 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. Using GPU particle emitter, reducing material over stacking and using level of detail optimization can effectively reduce the cost of particle effects. 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.
5. References

[1] Yu Lichun J 2015 Research on the Application of Movie Particle System Based on OpenGL. Journal of Langfang Normal University Vol.15, No.4, August 2015:35-38

[2] WILLIAM T. REEVES J 1983 Particle Systems-A Technique for Modeling A Class of Fuzzy Objects. ACM Transactions on Graphics, Vol.2, No.2, April 1983: 91-108.

[3] Yao Liang 2018 Technical Basis of Virtual Engine (UE4) Beijing: Electronic Industry Press, September 2018

[4] https://docs.unrealengine.com/en-US/Engine/Rendering/ParticleSystems