Simulating rain

Desirée Ohlsson*

Lund University
Sweden

Abstract

This report describes the implementation of a rain effect in the Sponza scene using mesh particles based on the *Simulating Particle Effects using OpenGL* article [van Oosten 2011]. The implementation of the rain is added on top of the code for the second assignment in the course *High Performance Computer Graphics* and the goal is to create a realistic rain effect that reacts to light.

1 Introduction

An interesting effect in a 3D scene to make it feel more alive is weather. The goal with this project is to add a rain effect to the Sponza scene. Real rain drops are not perfectly round and the way we perceive them is more like lines.

Since real rain make things wet, that is a thing that will be considered in this project as well. The way it is done here is by flooding the Sponza scene and have a constantly rising water level.

2 Algorithms

2.1 Particle system

The particle system consists of mesh particles. In this case the particle is a sphere that has been distorted to resemble a rain drop when it is in motion. A shader is then applied to the particle rain drop to give it the appearance of a rain drop. The transformed used on the sphere is a scaling transform:

$$\text{scale}(x, y, z) = (0.25, 6, 0.25)$$

Since the goal is not to have only one raindrop, animation was added in the implementation of the particle effect. Since the Sponza atrium has a rectangular box shape, a bounding box with approximately the same size was used to determine where it should be raindrops and where it should not be rain drops.

The rain drops where then where then spawned randomly (from a rectangular distribution) within the bounding box. The amount of particles used was 30000, which was chosen to resemble a quite heavy rainfall. The raindrops where then given a velocity of 20 in the negative y-direction to animate the rainfall. When a rain drop then hits the ground, it is immediately placed back on its starting height and re-used to create more rain without creating more particles.

To make the raindrops react to light, they are not added to the depth buffer.

2.2 Flooding effect

The rain particle system on its own is not enough to create a realistic rain effect. The ground also has to be wet after being hit with heavy rain. Since the Sponza scene looks pretty solid without any real doors, windows or drainage system, I thought it would be reasonable to assume that the scene would be flooded in case of rain.

To make the scene look flooded, a large tesselated quad was added on top of the scene. To give more life to the water, the vertexes on the quad were displaced using a sum of two sine functions and a colour was added to the water. The water does not account for how optics work in water. To make the water look transparent, OpenGLs blending functions are used.

Since the scene is flooded because of the heavy rain, the water level has to rise over time when it keeps raining. This is solved by changing the y-coordinate of the quad with time.

Since the water level is rising, the raindrops have to ”disappear” when they hit the water surface. In this project this is done by checking the depth buffer.

2.3 The rendering

The effects created in this project are all added into the same geometry buffer as the Sponza model. The effects are then all rendered in the first render pass as shown in figure 1 to make the effects react to light.

*e-mail: desiree.ohlsson@gmail.com

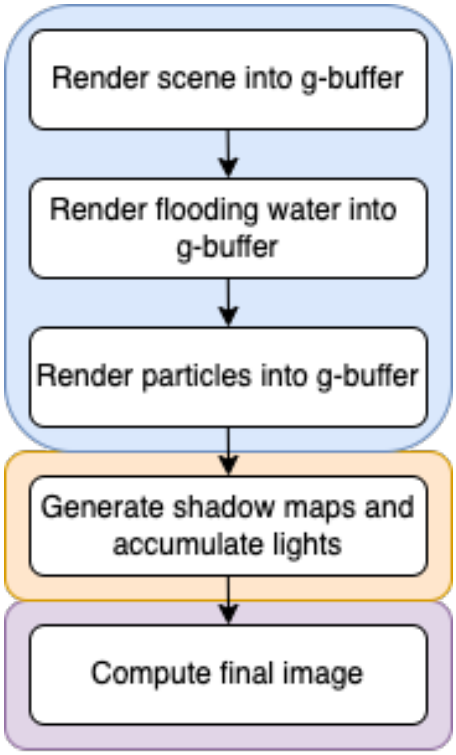

Figure 1: Order of the rendering and rendering passes.

3 Results

The rain may not look super realistic on still images like the screenshots in figure 2 and 3, but it is still resembles rain. The images still show how the rain reacts to light. When the rain is animated, which is the purpose here, the rain looks more like actual rain.

Figure 2: Raindrops and flooded Sponza scene

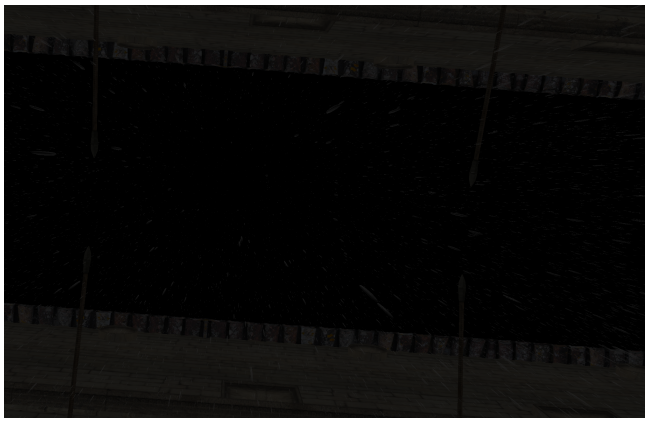

Figure 3: Raindrops as seen when looking from the ground up to the ”sky”.

3.1 Usefulness

Since the rendering of the rain is not too inefficient, this is an effect that could be useful. The effect could for example be used in a scene in a game to simulate more realistic weather.

4 Discussion

A lot of shortcuts have been taken to create the rain effect in this project, so there are a lot of ideas for improvements! The shaders used for both the rain and the rising water level are simple. The shaders could be improved to make a better model of how water and light interacts, but that could be at the cost of performance.

One thing that could be added to the water level shader to make is ripples where the rain hits the surface instead of just having the slightly wavy effect that is used now. If ripples where used instead, they could either be modelled using a lot of sine waves I believe or they could be modelled by just using a ripple bump map, which would probably be computationally more efficient.

One thing that I tried was to use different sizes of the raindrops, but it did not really look better. The rain looked less like rain when the size of the raindrops varied.

The particle effect is achieved using mesh particles which is not as efficient as for example billboard particles. But since the raindrops consists of mesh particles, it is easy to add another shape for fun. If you for example skip the part where the spherical particle is scaled and reduce the velocity of the particles, you would have something that resembles snow instead without much work.

References

VAN OOSTEN, J., 2011. Simulating particle effects using opengl. https://www.3dgep.com/simulating-particle-effects-using-opengl/.