Sky Diary

A sky scene with transformations

Zihan Yang
CMPM163 Spring 19
University of California, Santa Cruz
zyang41@ucsc.edu

Jiapei Kuang
CMPM163 Spring 19
University of California, Santa Cruz
jkuang6@ucsc.edu

Ge Lu
CMPM163 Spring 19
University of California, Santa Cruz
glu7@ucsc.edu

ABSTRACT

Our project is to present a normal day with climate change, special events such as fireworks, shooting stars by using Unity. The idea is to create different visual effects by using shaders and particle systems for the base scene (sky) in order to achieve our goal.

In general, we create a water shader for water body and use height map to create a terrain. And there is a god ray shader applied. For Daylight Scene, there are particle systems for clouds. For Night Scenes, there are particle systems for stars. Then, there are some special effects such as raining scene, firework scene, and shooting star scene.

KEYWORDS

Particle system; Trail renderer; Animation; Shader programming; C sharp script;

1 Introduction

Our group members are Jiapei, Ge and Zihan, and the distribution of the work is listed below. The main purpose of this project is to try to combine everything we learned such as shaders, particle systems and scripts together into a single scene and make it look nice.

2 EXPERIMENTAL AND COMPUTATIONAL DETAILS

2.1 Water - Jiapei

For the water body, the shader will take in a bump texture, gradient, and perlin noise. The water body's uv (waves) will be moved with the gradient picture and the color will be set by the gradient image.

Figure1: Gradient image (Light blue to Blue)

Then we use the bump texture to mix the uv coordinates to create waves. And we use the noise texture to create more natural waves color and waves movement.

2.2 God Ray - Jiapei

For the God Ray effect, I used and modified the free Crepuscular Ray effect by Super_ Shaman from the Unity Store.

The effect works by first setting a script that allow user to choose a light source and then record the position of the light source. Then, in the shader, it will take the light source position and set up for the light rays.

The light ray effect will be controlled by the custom data of decay, illumination, density, and exposure. After retrieving the sample light ray, it combined the color by calculating Decay factor, Weight, and numbers of samples together. By multiplying and returning the final combined color and exposure factor, it will create the god ray effect.

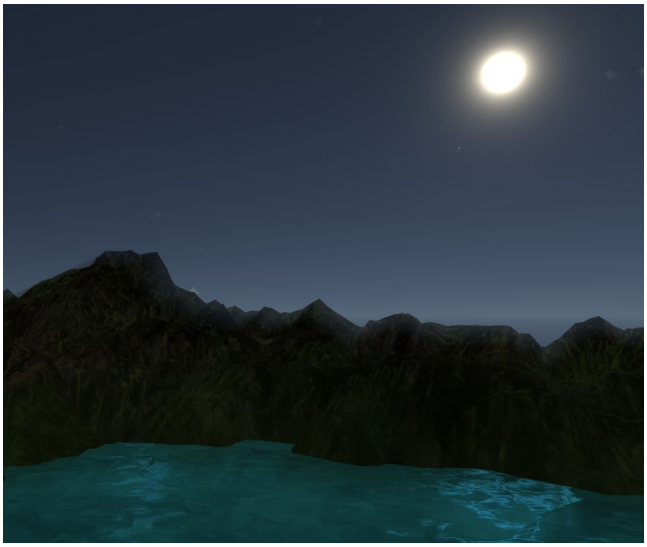

Figure1: God Ray Effect in Light Mode

2.3 Clouds - Jiapei

We use two particle systems with different clouds texture to simulate realistic clouds. By playing around with the lifetime and speed, the clouds will move towards the same direction. By aligning two different systems horizontal to each other, there will be a randomly mixed clouds scene in the sky.

Figure 1: Two clouds particle systems

2.4 Rain - Ge

In order to create realistic rain, we use multiple particle systems. From our observation, natural rain is more than just water drops falling down from the sky, as the rain is primarily composed of raindrops, mist, and raindrop splashes. With that being said, we created multiple particle systems to present each factor. Our creation is mainly based on observing natural weather and game weather. The biggest challenge in this part is the aesthetics of the materials. There are few suitable materials online so I have to draw them by myself with limited art skills.

For the raindrops, we have observed that raindrops that are closer to the camera appear larger than those that are farther. So we built two particle system with one representing larger raindrops and the other with smaller ones.

The larger one is Stretched Billboard in Render Mode, so that the raindrops fall vertically. The other one is Horizontal Billboard in Render Mode so that the tiny rain dots stretched horizontally. These horizontal little rain dots enable the

illusion that they are distant from the camera.

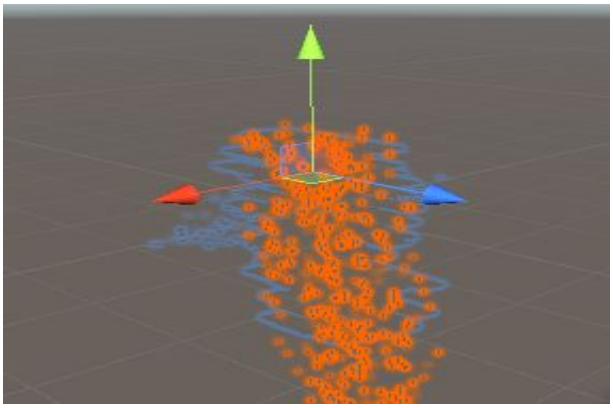

Figure 1: **Two raindrop particle systems**

Rainy days are often accompanied with mist. We use a particle system to present it.

The key to keep it realistic is that to have the material random and slow. The randomness is granted by Auto Random Seed and the slowness is accomplished by a low simulation speed.

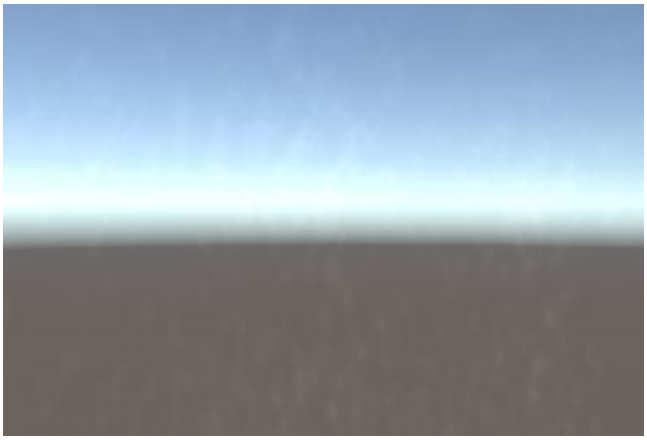

Figure 2: **Mist**

There are splashes as the raindrops hit the ground, and we use another particle system with sub Emitters to form the splashes. We keep each splash area small and random in position, emitting from ground with a very small hight.

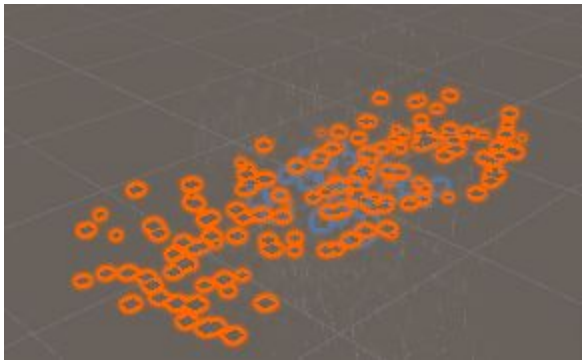

Figure 3: **Splash**

2.5 Stars - Ge

Our goal for stars is to make them twinkle and shine. When we look up into the starry night, we notice that stars blink and shine. Stars seems to be white at our first glance, but after staring at them, we had the feeling that some stars look white, some look blue and some look yellow. We also notice that stars twinkle at different frequencies and vary in sizes. Hence, we use multiple particle systems to have the stars with different materials and colors. The stars are born at random positions with sizes and colors randomly changed over its lifetime. Gradient Editor is also used to control the shiness.

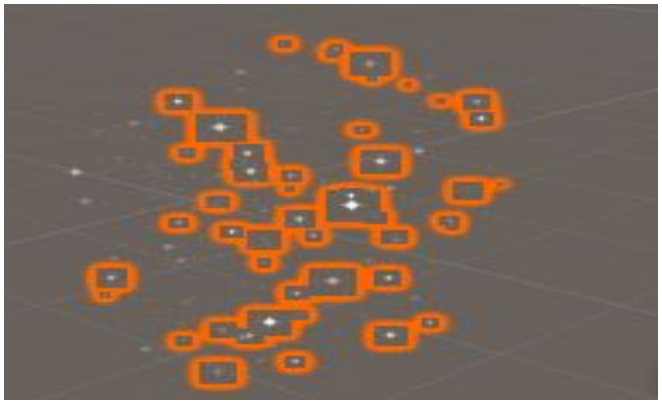

Figure 4: **Stars with random sizes and positions**

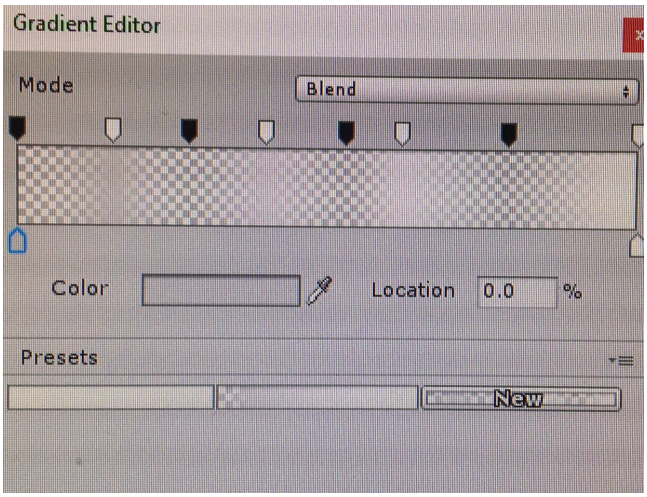

Figure 5: **Gradient Editor used to make stars twinkle**

2.6 Fireworks - Zihan

We use the multiple particle systems to make the fireworks by applying one particle system to another. The particle system is a really useful tool using a large number of very small sprites or 3D objects imported to the project to create animations and a lot of different natural phenomenons such as smoke, fire, clouds by controlling and changing the size, lifetime, color, and other parameters to those particles.

We start making the firework by creating a main particle system, which just aiming for creating some trails going out from the same point to the sky and then disappear. We use the trail inside the particle system, and create a curve on it which goes from high to low, in order to make it appear very clearly and then fading at the end.

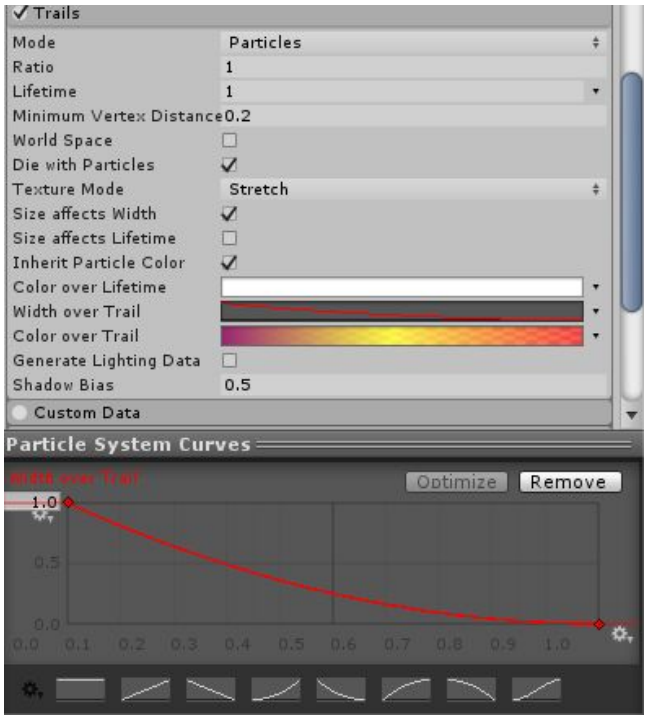

Figure 1: **The curve inside the main particle system, making firework trails**

Then we start making the sparkle effect by creating a sub emitter particle system inside the main one. We apply a sparkle texture material to it, which is just a simple sparkle PNG. Then making it move along with the main particle system by adding it to the sub emitters tab inside the main particle system, and set it to birth. Change color, size, and lifetime parameters of the sparkle sub emitter, also make each particle rotating randomly to make it look better.

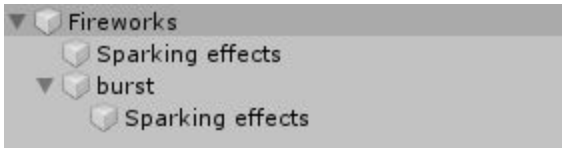

Figure 2: **How we applied multiple particle systems together**

Now we have some colorful trails coming out of a point along with some pretty sparkling effects. But another important part of fireworks is the burst when the trail goes to the sky. To do this, we duplicate the main particle system that we

just made, and add this under the main one after we made some changes to it.

For the burst particle system, inside the emission tab, we set the time to zero in order to make it explodes right after the trail disappear in the sky. Along with this, inside the main firework particle system, we apply this burst particle system to it as a sub emitter, and we set it to death. With these two changes, the burst can happen right after the firework “died”.

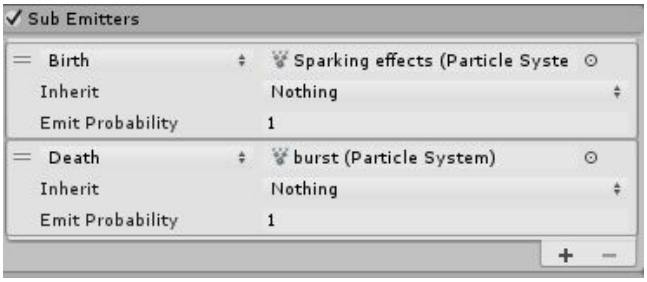

Figure 3: **The sub emitter settings inside the main particle system**

2.7 Shooting Stars - Zihan

We use the animation effect inside Unity, the trail renderer component for game objects and the particle system to create a shooting star which start from the left side of the sky and then fade out at the right side of the sky.

At first, we enable the animation inside Unity, and then apply this animation to an empty game object. Set the animation duration to 1.5 seconds long, so the shooting star will appear for 1.5 seconds; Then set the curve of the game object position from lower to higher, which can make it shooting out very fast and then turn to be slower. We click on the record button to make sure the animation remembers the trace that we made.

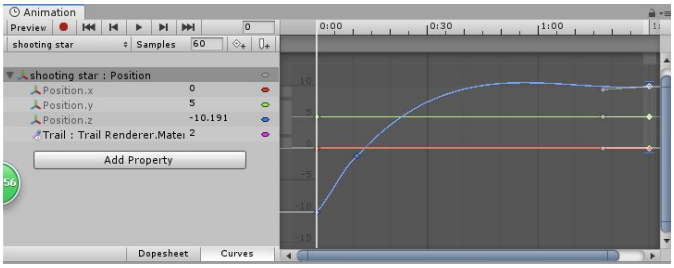

Figure 4: **The curve of the animation**

After that, we add a trail renderer to the game object. We create the trace simply by just drag the object to the right, and then apply a material onto it. We edit size, color and last time to make it look more like a real shooting star. Set the curve inside the renderer to make the shooting star goes out strongly then fading at the end.

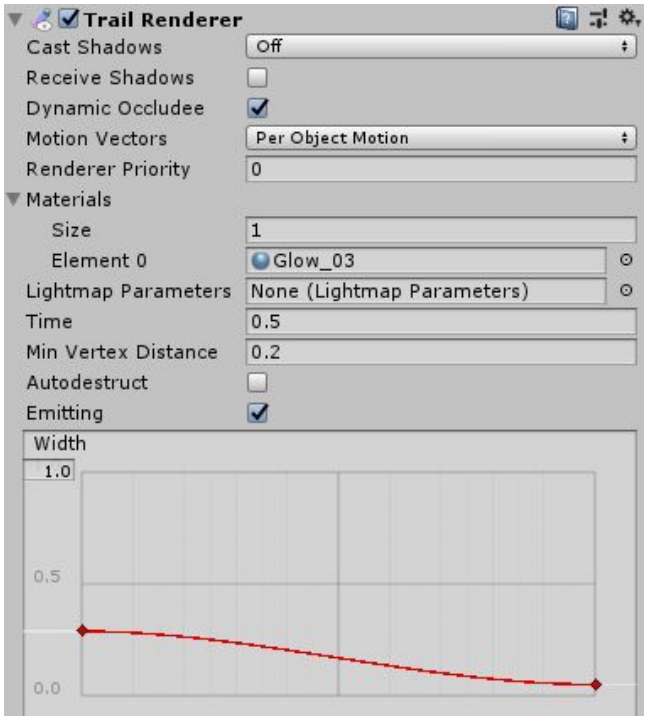

Figure 5: **Trail renderer properties of the shooting star**

Now we have a “shooting star” but it is too hard, we want it to look softer. We add some sparkling effects on to it by creating some particle systems to it. There is one thing important inside this spark particle system, which is the inherit velocity tab. We enable it and set a curve there, making sure that the sparks move along with the trail, instead of stopping in the middle or still last when the trail is gone.

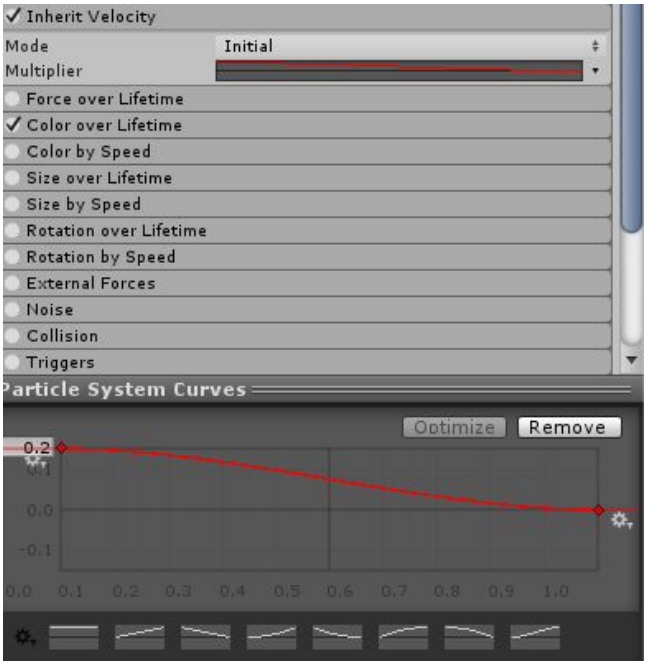

Figure 6: **The curve inside inherit velocity**

Finally, in order to make the shooting star appear, we write a script that when the user press the space key on their keyboard, the shooting star shoots. In the script, if the space key is pressed, then we set the shooting star to be active. After that, if the particle system is not alive, then we set the shooting star projectile to be inactive and we reset the status of the projectile. If the user press the space key again, the script will do the same thing again.

```
public class PlayAnimOnKeyUp : MonoBehaviour {

    public GameObject mainProjectile;
    public ParticleSystem mainParticleSystem;

    // Update is called once per frame
    void Update () {

        if (Input.GetKeyUp(KeyCode.Space))
        {
            mainProjectile.SetActive(true);
        }

        if (mainParticleSystem.IsAlive() == false)
            mainProjectile.SetActive(false);
```

Figure 7: **Code of the key-press active script**

3 CONCLUSION

We all have learned about utilizing Unity to present the visual effects like, and we also applied individual's specialty to apolish each part of our project. Every piece of individual work is completed as expected, and the project is well integrated as a whole.

4 ACKNOWLEDGMENTS

This work is for CMPM163 spring 2019 at UCSC, professor Angus Forbus.

5 REFERENCES

[1] https://www.artstation.com/artwork/qm4yD

[2] CrepuscularRays Asset from Unity Store

[3] For water body: https://blog.csdn.net/PangNanGua/article/details/85336351

[4] For water body:

https://blog.csdn.net/chrisfxs/article/details/72364924