CGRA350 FINAL PROJECT REPORT

Ocean Environment Simulation

Team members

Zhenchuan (James) Yu - 300489509 - jameszhcyu@gmail.com
Maxwell Johnson - 300539489 - max.t.johnson@gmail.com
Sean Preston - 300537970 - seanprestondev@gmail.com

School of Engineering and Computer Science
Victoria University of Wellington

Supervisor: Alex Doronin, Zohar Levi

Abstract

We implemented a real-time simulation of an ocean with atmospheric scattering and screen space reflections. The ocean simulation will use a vertex shader to simulate the appearance of natural ocean water. It will have several different variables that will allow users to adjust the properties of how the ocean moves, such as the frequency of the waves as well as their amplitude of them. Screen space reflections (SSR) will allow for accurate reflection of both dynamic and static objects, independent of scene complexity. Atmospheric scattering is my topic, and I will create the sky for the scene and make a simulation for the colour of the sky.

Group Summary

Technical Contributions	Members
Vertex shader and Fragment shader for atmospheric scattering	Zhenchuan Yu
Sphere object generation and other relevant functions	Zhenchuan Yu
GUI for atmospheric scattering and reflection modification after integration	Zhenchuan Yu
Vertex shader and Fragment shader for wave simulation	Maxwell Johnson
GUI for wave simulation	Maxwell Johnson
Plane generation and other relevant functions	Maxwell Johnson
Screen space reflections	Sean Preston

1 Introduction

Generating realistic atmospheric scattering for computer graphics has always been a complex problem, but it is essential for rendering realistic outdoor environments. Computer graphics models generally use simplified equations, and very few of them run at interactive frame rates. The scattering equations have nested integrals that are impossible to solve analytically. The equations can be approximated with some techniques, but the cost is quite expensive.

1) Related Work

One of the first papers that simplified the scattering equations was by Nishita et al. 1993 [1]. Nishita's paper assumes that the camera is always on or very close to the ground. This makes it easier to assume that the atmosphere has a constant density at all altitudes. Later, O'Neil 2004 described a similar algorithm that ran on the CPU but was too CPU-intensive. It was based on a pre-calculated 2D lookup table with four floating-point channels. Unlike Nishita's model, O'Neil's model is inefficient and cannot work on GPU. Therefore, in our project, we implemented the full equations from Nishita et al. 1993.

2) Technical problem and tasks

Since the main object is the ocean, making a realistic sky for the scenery is reasonable. Thus, my task in this project is to create the sky and make an accurate simulation of the atmospheric scattering in real-time: a) We need to understand scattering equations and implement them in the shader fully. b) Determine which term in the equation is the actual input we need and which variable needs to be set up as constant. c) Complete GUI and pass the input to the shader.

3) Contributions

In our project, I was responsible for writing the vertex and fragment shader, respectively. After that, I also wrote a built-in function to generate a sphere object for the atmosphere. I wrote the code for creating the GUI of atmospheric scattering for the input and other parameters, and finally, a `draw()` function for passing the variable to the shader. In order to make the sky interact with the ocean, we implemented Cook-Torrence shading. I was responsible for setting up the part for making the light interact with the ocean.

2 Solutions and implementation details

If a ray passes through the atmosphere from camera A to vertex B, scattering needs to be calculated. Therefore, we perform a sphere-intersection check to find where the ray intersects the outer atmosphere. For simplicity, we take three sample positions, namely P1, P2 and P3. Each point represents a point in the atmosphere at which light scatters (Figure 1). light comes into the atmosphere from the sun, scatters at that point (light ray), and is reflected toward the camera (view ray).

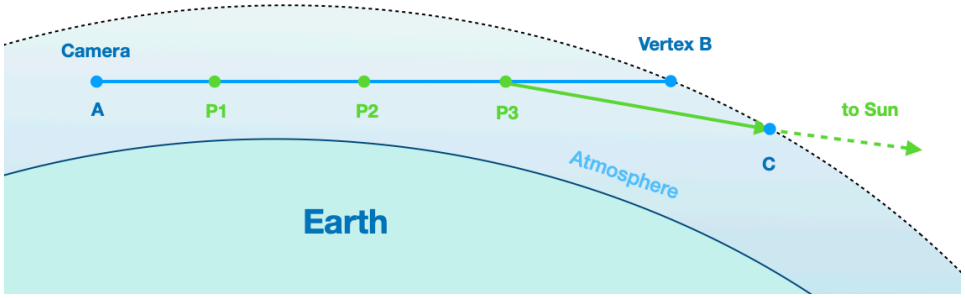

Figure 1 Demonstration of atmospheric scattering

2.1 Phase Function

$$F(\theta, g) = \frac{3 \cdot (1 - g^2)}{2 \cdot (2 + g^2)} \cdot \frac{1 + \cos^2 \theta}{1 + g^2 - 2g \cos \theta}$$

Figure 2 Phase function

For simulating the scattering, we use the phase function (*Figure 2*) to represent Rayleigh and Mie scattering. The phase function describes how much light is scattered toward the camera's direction based on the angle (the angle between green and blue rays in *Figure 1*). Rayleigh scattering can be approximated by setting g to 0, simplifying the equation. For Mie scattering, g is usually set between -0.999 and 0.999.

2.2 Scattering Equation

$$\text{result} = I_{\text{sun}} \cdot K(\lambda) \cdot F(\theta, g) \int_a^b \left\{ \exp\left(\frac{-h}{H_0}\right) \cdot \exp[-t(c, b) - t(a, c)] \right\} \, \mathrm{d}s$$

- The constant is the wavelength (or colour) of light, so $K(\lambda)$ is the scattering constant, which is dependent on the density of the atmosphere at sea level. Rayleigh and Mie scattering each have their own scattering constants.
- Segment (c, b) in this case is the light ray and (a, c) is the view ray.
- ds is the `segment length` of the ray in the pseudocode.
- The first exponential term is called 'Optical depth'. The second exponential term is attenuation combined with further optical depth calculation (*Figure 3*) on light ray and view ray respectively.

$$t(\hat{a}, \hat{b}) = \int_{\hat{a}}^{\hat{b}} \exp\left(\frac{-h}{H_0}\right) \, \mathrm{d}s$$

Figure 3 Function t (Optical depth)

Pseudocode

```
compute_scattering(rayDirection, cameraPosition) {
    getSphereIntersection(cameraPosition, rayDirection, earthRadius)
    calculate segment length on view ray
    phase_M = getPhaseMie()
    phase_R = getPhaseRayleigh()
    for each sample point on view ray
        get the position of the sample point by line interpolation
        accumulate optical depth for Rayleigh and Mie on view ray
        getSphereIntersection(samplePosition, sunDirection, atmosphereRadius)
        calculate segment length of light ray
        for each sample point on light ray
            get the position of the sample point by line interpolation
            accumulate optical depth for Rayleigh and Mie on light ray
        calculate attenuation and accumulate into final sum of optical depth for both Mie and Rayleigh
    return sunIntensity * (sumOptDepthRayleigh * rayleighConst * phase_R + sumOPtDepthMie * mieConst *
    phase_M)
}
```

In my implementation, for finding the intersection, I use `height of the camera` instead of `earthRadius`, otherwise when we move the camera up, there will be a horizontal line across the

horizon in the result. For the interacting part, since the shader of the ocean is based on Cook-Torrence model. We introduced the sun's position to the ocean's shader. The colour is the interpolation between two pre-defined colour, one for sunset and another one for the daylight.

3 Results of Atmospheric Scattering

In this project, I learned the scattering theory and successfully implemented it in the shader. Some important results are shown below:

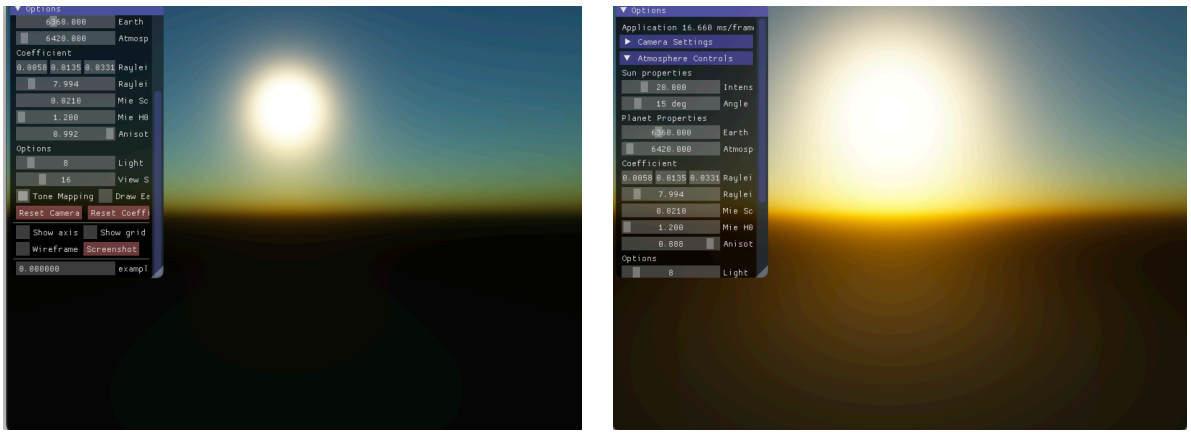

Figure 4: By changed the 'Anisotropy', the program can simulate the halo effect of the atmosphere

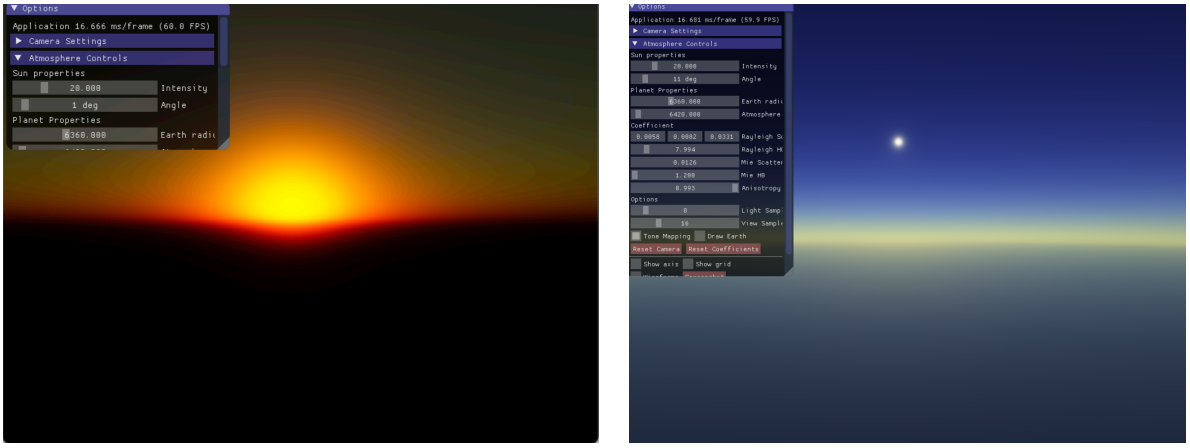

Figure 5: Simulate sunset and outer space by changing the sun's angle and camera's position

4 Discussion

Although we successfully implemented the algorithm for the scattering and the ocean in our project, the interaction between the ocean and the sky is essential (*Figure 6*). From the perspective of the team working, the challenge for us is to make the simulation work accurately (Photorealistic). My challenge is to correctly make the sky's colour interact with the ocean.

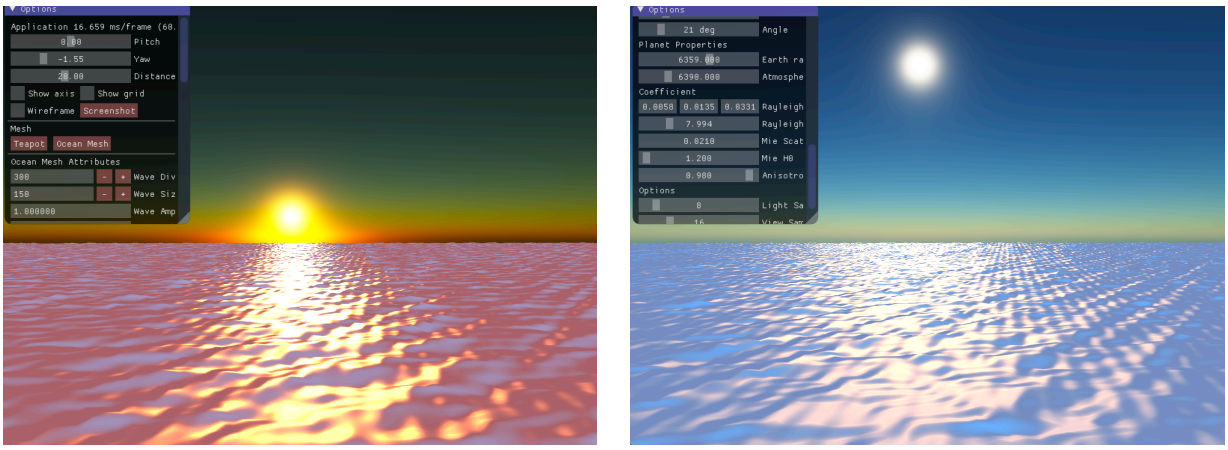

Figure 6: The sky can interact with the ocean

Our atmospheric scattering using real-world scale. Therefore, the adjust the camera's position and model view matrix is quite tricky. Our problem is that we could perfectly match the shy and ocean well. As we can observe from *Figure 6*, The colour simulation near the horizon is not very satisfactory. The reason is that the camera's position for simulating the sky does not match the camera for the ocean. In other words, to observe the sky like that, we have to stand in a higher place. However, the current camera' portion is close to the sea.

Reference

[1] Nishita T., Sirai T., Tadamura K., Nakamae E.: *Display of The Earth Taking into Account Atmospheric Scattering*, in Siggraph '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, pages 175–182, 1993

[2] https://developer.nvidia.com/gpugems/gpugems2/part-ii-shading-lighting-and-shadows/chapter-16-accurate-atmospheric-scattering

[3] https://learnopengl.com/Advanced-Lighting/HDR