CS7GV3 Final Assignment: Atmospheric Scattering

Timon Sommer

Student ID: 24334440
Video: `https://www.youtube.com/watch?v=x72tCHFUa7M`

Abstract

In this report, a method for rendering atmospheric scattering in real-time is investigated and parts of it are implemented using OpenGL and C++. Instead of using a pre-computed look-up table (LUT) for properties of the atmosphere, the method relies on a ray-marching approach that determines the scattering of light in the atmosphere in real-time using simplifying assumptions about atmospheric transmittance. We eventually find that the implementation is capable of producing visually plausible results at interactive frame rates, while higher sampling resolutions only slightly improve fidelity yet significantly decrease performance. Additionally, the chosen approach currently ignores transparency of the atmosphere towards objects in space and remains limited to views directed from space towards the atmosphere, but can be extended to support views from the ground towards the sky.

1 Background

1.1 Natural Atmospheric Scattering

Atmospheric scattering is a phenomenon that occurs when light from space such as sun- or moonlight interacts with particles in the atmosphere, creating visual effects such as the sky appearing blue during the day and red when approaching dawn or dusk. Depending on the kind of particle involved, there are two main scattering components as described in [1]:

- Rayleigh scattering: This type of scattering interacts with smaller particles such as air molecules. Since its scattering behaviour depends on the wavelength of the light, it is responsible for the varying colour of the sky at different times of the day (see Figure 1).
- Mie scattering: Here, light rays are scattered by bigger particles in the atmosphere such as aerosols. Scattering of this kind affects all wavelengths equally, which causes the sky to appear white or hazy when the sun approaches the horizon.

Figure 1: Photograph of the atmosphere of the earth taken from space.
Source: `https://earthobservatory.nasa.gov/images/76534/hovering-on-the-horizon`

1.2 Precomputed Atmospheric Scattering

As light rays travel through the atmosphere from the source to the viewer, light is scattered into and out of the rays along their trajectories, modulating the final intensity of all wavelengths. Both the light resulting from in- and out-scattering can be described using integrals over the ray.

Due to the complexity of these processes, the approach presented [1] uses a pre-computed LUT to store the scattering properties of a given atmosphere. Most importantly, this LUT contains the transmittance of the atmosphere at a given height, where the values are calculated using a numerical integration of the scattering equations. This way, the integral approximation does not need to be sampled in real-time, which would otherwise be computationally expensive.

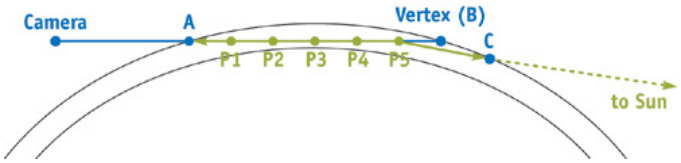

Figure 2: Discrete ray sampling as described in [2].Source: [2]

The article [2] implemented in this project does approximate the scattering integrals but still makes simplifying assumptions to allow evaluating both the out- and in-scattering terms at interactive frame rates. Based on the formulation of the scattering equations in [3], the author of [2]

derives the equation for the out-scattering component as

$$t(P_xP_y, \lambda) = 4\pi \times K(\lambda) \times \int_{P_x}^{P_y} \exp\left(\frac{-h}{H_0}\right) ds, \tag{1}$$

where $K(\lambda)$ is the scattering coefficient dependent on the wavelength λ, H_0 is the altitude of the highest density in the atmosphere, and h is the height of the discrete sample point above the planet surface. We can then nest this equation inside another integral over all sample points along the ray to yield the in-scattering term as

$$I_v(\lambda) = I_s(\lambda) \times K(\lambda) \times F(\theta, g) \times \int_{P_a}^{P_b} \left(\exp\left(\frac{-h}{H_0}\right) \times \exp\left(-t(P_xP_c, \lambda) - t(P_xP_a, \lambda)\right)\right) ds, \tag{2}$$

where $I_s(\lambda)$ is the intensity of the light source, and $F(\theta, g)$ is the phase function dependent on the angle between the incident light and camera rays at each sample point (see points $P_1 - P_5$ in Figure 2 for an example):

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

Since both K and the scattering symmetry factor g and thus also F are different for Rayleigh and Mie scattering, contributions of both components need to be calculated separately and are subsequently added to yield the final result.

The main simplification made in [2] is to approximate the height-related integral in equations (2) and (1) by fitting it to an exponential curve:

$$\int_{P_a}^{P_b} \exp\left(\frac{-h}{H_0}\right) ds \approx \exp(-4h). \tag{4}$$

However, since this approximation is based on normalising the height h, it needs to be multiplied by a transformation function that maps the values to the actual atmosphere height range. This polynomial transformation function is defined dependent on both H_0 as well as the relative distance between the surface and the outer atmosphere and thus would have to be recomputed every time any of these parameters change.

2 Implementation

Given that the approach presented in [2] requires a new transformation function to be computed whenever the atmosphere height or H_0 for either Rayleigh or Mie changes, the implementation in this report foregoes this step. Inspired by the approach in [4], the exponential optical depth term is instead approximated by evaluating the left side of equation (4) at discrete sample points along the ray trajectory. This decision was made given that said article was published in 2005, and the capabilities of modern GPUs have improved significantly since then.

Still, the overall raymarching approach employed in [2] is retained with the modification that it is performed in a fragment shader instead of a vertex shader. While computationally more expensive, this leads to a higher resolution of the rendered atmosphere, especially because we assign the shader to a sphere mesh instead of a full screen quad. This is done to keep depth testing intact without having to write manually to the depth buffer, enabling us to integrate the atmosphere shader with standard rasterisation-based shaders assigned to other objects such as the moon or the sun (see Figure 3).

Figure 3: Stills from the shader implementation demonstrating atmospheres being partly occluded by the moon (left) and occluding the sun disk in the distance (right).

The first step is thus to cast a ray from the camera position P_c to the world position of the current fragment, where the latter is calculated in the vertex shader by ignoring the view and projection matrices. Using an implicitly defined sphere that matches the size of the sphere mesh in the scene, we then determine the distance P_a at which the ray intersects with this sphere, while no intersection results in the current fragment being discarded. After repeating this intersection test for the surface sphere yielding P_b, the shader then proceeds with evaluating the in-scattering equation (2) using the Riemann sum method for approximating the outer integral. For each sampling step P_x and both components (Rayleigh and Mie) separately, we determine the optical depth at the current altitude first, since it is used in both the out- and in-scattering terms. In accordance with equation (2), the out-scattering is calculated using the Riemann approximation of the integral in equation (1) for the rays P_xP_c and P_xP_a. This determines the amount of light that is scattered out of the ray towards the camera, which is then multiplied by the phase function F and the scattering coefficient K to yield the in-scattering influence on the light intensity I_s emitted by sun, which is assumed to be the only light source in the scene. It is important to note that (especially Rayleigh) scattering affects different wavelengths of light with varying intensity, which is why all scattering terms are calculated using vectors of three components corresponding to the RGB channels. Analogously, the coefficients K are also expressed in this vector format, where the values are determined based on the wavelength peaks of the RGB channels as specified in the manuscript and source code of [2].

Concluding the approach in [2], the final fragment colour is then given as a combination of the in-scattering result I_v and the scattered surface colour I_e:

$$I_f(\lambda) = I_v(\lambda) + I_e(\lambda) \cdot \exp\left(-t(P_a P_b, \lambda)\right). \tag{5}$$

For this project, I_e is assigned a constant colour value that can be adjusted in the demo application, yet this can easily be replaced by common reflectance models such as Blinn-Phong combined with texture sampling to create a more realistic surface appearance.

3 Results

3.1 Visual Output

To visualise the output of the implemented vertex and fragment scattering shaders, a demo application (see Figure 4) for MacOS was developed using C++ that allows various atmosphere, camera, and light source parameters to be adjusted in real-time.

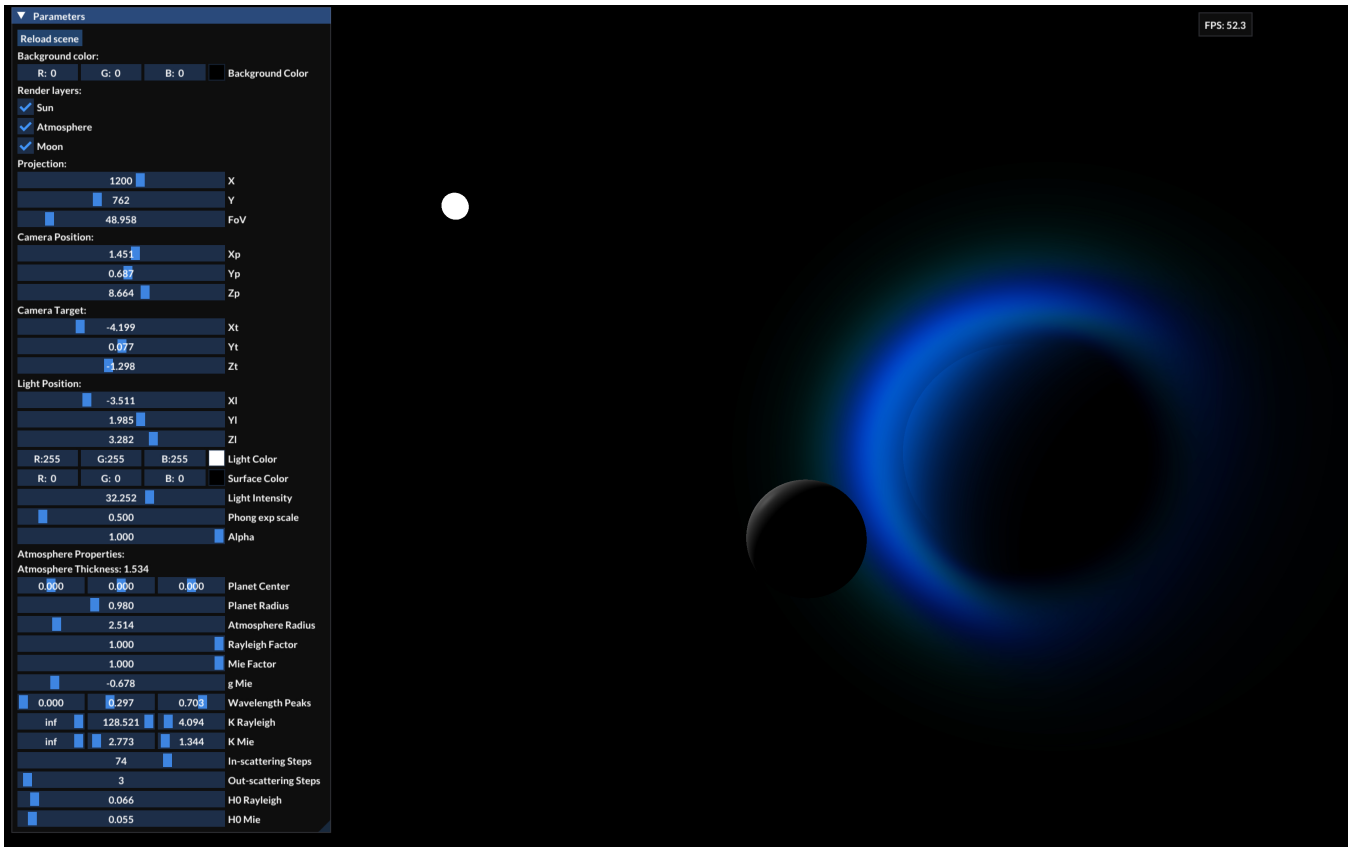

Figure 4: Screenshot of the demo application.

As illustrated in Figure 5, the shader is capable of producing visually plausible results that clearly show the sky colour gradient from blue to red as the sun approaches the horizon. We also

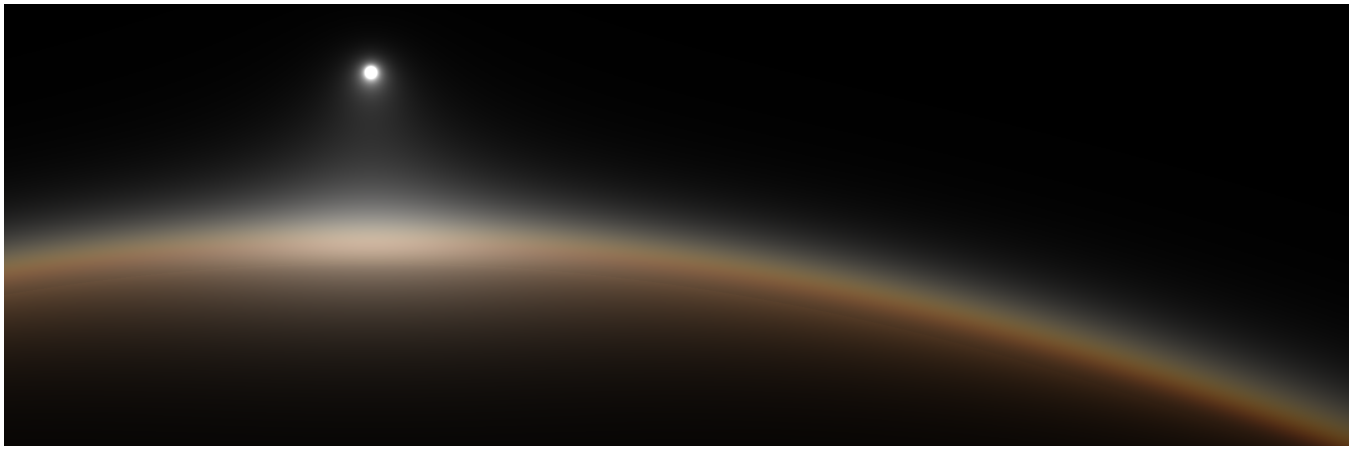

Figure 5: Twilight scenario produced by the shader in the.

Figure 6: Scattering components in isolation.

observe a glow effect created by the Mie scattering around the sun disk which is mostly white in colour since Mie scattering does not depend on the wavelength of the light. The effect of both scattering components can further be isolated by setting the factor of the other component to zero, which is illustrated in Figure 6.

3.2 Performance

Since the fragment shader contains an outer in-scattering loop that iterates over the number of samples along the ray and a nested out-scattering loop, it is important to measure its real-time performance. Rendered on the Apple M2 Pro chip with an integrated GPU at a pixel resolution of 1920×1080 at different step counts, we achieve the framerate values shown in Figure 7, where the camera is moved closer the atmosphere so that the latter covers a circular area measuring approximately 800 pixels in diameter. This is done because we observe that with decreasing distance between the camera and the atmosphere, the framerate declines significantly due to the higher number of fragments that now need to be rendered. Since the implementation discards any fragments that are not inside the atmosphere, we can avoid further unnecessary calculations.

As the number of sample points is reduced, the resolution of the atmosphere gradient decreases as well, creating more clearly defined bands of colour which can be seen in Figure 8. The change in the out-scattering step count is less noticeable, where changes in visual fidelity are most noticeable for values < 5. Yet, the results presented in Figure 7 indicate a strong influence of this parameter

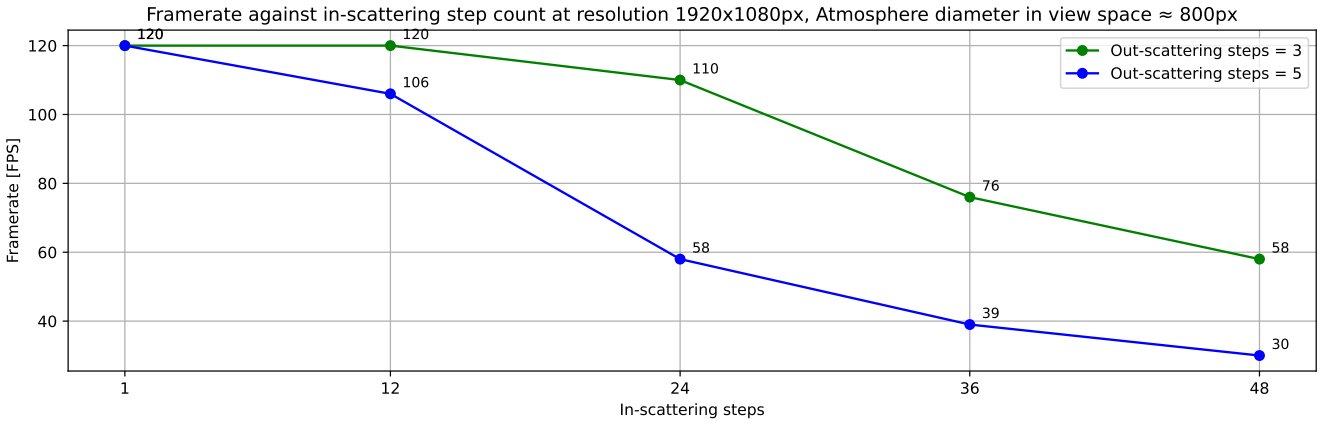

Figure 7: Achieved shader framerates at different step counts.

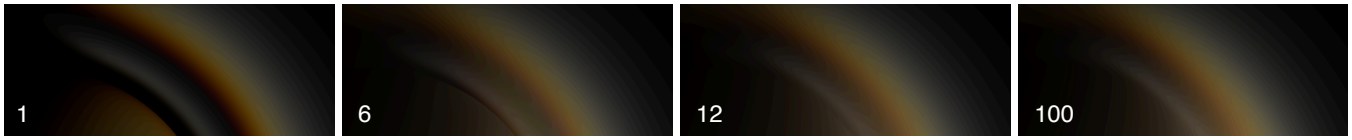

Figure 8: Visual atmosphere resolution at different in-scattering step counts.

on the framerate, which is expected since the out-scattering step count value effectively acts as a multiplier for the number of iterations in the outer loop.

4 Conclusion and Future Work

The findings of the previous section show that the implemented shader creates believable visualisations of real planetary atmospheres at high framerates. While the in-scattering is more sensitive to the number of sample points, small step counts for the out-scattering term still yield believable and smooth results. For the in-scattering step count, values in the vicinity of 12 were shown to produce renderings of plausible quality that avoid major banding artifacts.

Compared to the work done in [2] and [1], the implementation in this report still shows some limitations. Firstly, the shader in its current form only supports viewing the atmosphere from space. For additional perspectives, such as the planetary surface, supplementary scattering logic and modifications to the ray marching approach would be required to produce plausible results. Secondly, as can be seen in Figure 3, the atmosphere is currently not transparent when it occludes other objects in the scene. For this to work, the alpha value would have to be included in the scattering equations, which requires further considerations. Since this implementation does not explicitly rely on precomputed values, future work could also investigate the effect that determining the optical depth using equation (4) would have on both visual quality and real-time performance.

References

[1] E. Bruneton and F. Neyret, "Precomputed atmospheric scattering," *Computer Graphics Forum*, vol. 27, no. 4, pp. 1079–1086, Jun. 2008, ISSN: 0167-7055, 1467-8659. DOI: 10.1111/j.1467-8659.2008.01245.x. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2008.01245.x (visited on 02/23/2025).

[2] S. O'Neil. "Chapter 16. accurate atmospheric scattering," NVIDIA Developer. (), [Online]. Available: https://developer.nvidia.com/gpugems/gpugems2/part-ii-shading-lighting-and-shadows/chapter-16-accurate-atmospheric-scattering (visited on 04/06/2025).

[3] T. Nishita, T. Sirai, K. Tadamura, and E. Nakamae, "Display of the earth taking into account atmospheric scattering," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, Anaheim CA: ACM, Sep. 1993, pp. 175–182, ISBN: 978-0-89791-601-1. DOI: 10.1145/166117.166140. [Online]. Available: https://dl.acm.org/doi/10.1145/166117.166140 (visited on 04/06/2025).

[4] gltracy. "Shadertoy." (), [Online]. Available: https://www.shadertoy.com/view/lslXDr (visited on 04/06/2025).