Atmospheric Scattering In Vulkan

Michael Wang#1, Glenn Jiang#2, Yifei Miao#3, Shipeng Liu#4

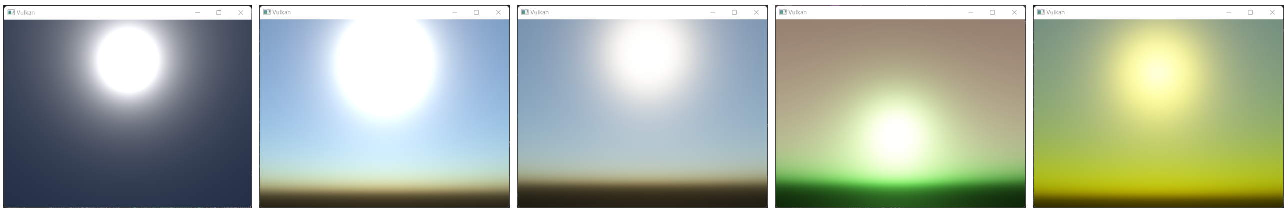
#Computer Science Department, University of Southern California, Los Angeles, CA 90007, United States

1wangmc@usc.edu

2liweijia@usc.edu

3yifeimia@usc.edu

4shipeng@usc.edu



***Abstract*— The rendering of atmospheric scattering plays a vital role in elevating the aesthetic appeal and physical realism of applications. In the mobile domain, the importance of efficient and swift atmospheric rendering cannot be overstated, and leveraging a high-efficiency graphics API undoubtedly facilitates this process. Our project has successfully executed an approach tailored for real-time atmospheric rendering, as described in [1], utilizing the Vulkan API. This accomplishment offers valuable practical insights for those looking to develop highly efficient engines with Vulkan.**

***Keywords*— Physically-Based Rendering, Atmospheric Scattering, Rendering Efficiency, Vulkan API, Procedural textures**

I. Introduction

In the field of computer graphics, simulating realistic natural environments has always been a significant and challenging task. Particularly, atmospheric rendering plays a key role in enhancing the realism and visual appeal of virtual environments. With the continuous advancement in graphics processing technology, physically-based rendering methods have become an effective means for simulating complex atmospheric phenomena.

However, in fields that prioritize system smoothness and energy consumption, such as mobile platforms, the substantial computational demands of realistic atmospheric rendering remain a key factor negatively impacting user experience.

This project is dedicated to implementing parts of the atmospheric rendering model from [1] using the Vulkan API. Vulkan is an efficient, modern graphic interface. It stands out in the realm of real-time rendering due to its high system performance and rendering efficiency. Our approach focuses on simplifying physical models and optimizing algorithms to strike a balance between maintaining rendering quality and improving operational efficiency.

Through this project, we demonstrate how to effectively enhance the computational efficiency of atmospheric rendering while maintaining visual quality. Additionally, we explore the performance of this approach, aiming to provide new perspectives and tools for researchers and developers in the field of computer graphics.

II. Related Work

The method of rendering the atmosphere from ground and space perspectives by computing atmospheric word scattering was first proposed by Nishita et al. [2] in 1992. This was one of the earliest attempts at atmospheric rendering. However, this model can only simulate atmospheric rendering from space. Moreover, some parts require constant recalculation, resulting in significant performance overhead.

Rendering originating from within the atmosphere was first addressed in [3], published in 1996. This method divided the sky into many regions and simulated the energy exchange between them. Some costly computations such as integral computations are cleverly stored in a LUT (Look-Up Table). However, this method still required a significant amount of computational time due to the limited computational power.

With the advancement of hardware, an increasing number of computations have started to shift to GPU processing. Researchers are also continuously attempting to minimize the computational load during real-time rendering, while maintaining the credibility of atmospheric rendering results, through methods such as pre-computation.

In 2008, Bruneton and others proposed in [4] the use of a precomputed four-dimensional LUT that includes the zenith angle of the line of sight, the zenith angle of the sun, the scattering angle between the sun and the line of sight, and the azimuth angle, to simulate all scenarios. The scattering results under all angles can be achieved through table lookup and minimal computation. [5] further reduced the complexity by omitting the azimuth angle ω, removing one dimension, and by discarding the computation of the Earth's shadow part in atmospheric rendering, significantly reducing the computational load. However, these methods require recalculation when atmospheric properties change, such as changes in weather or planetary environments, which can be excessively costly.

In Sébastien's 2020 work [1], a bold assumption was made by assuming that the isotropic scattering of particles in the air is uniform. This approach simplifies anisotropic scattering into a process of percentage energy attenuation, removing a significant amount of path integration. The method also reduces the dimensions of the LUT by not sampling altitude in each frame and keeping the zenith angle of the sun constant. While not entirely physically accurate, this method is computationally efficient, making real-time rendering more feasible. It has been widely applied in game engines and is very user-friendly for artists, allowing them to easily create atmospheric effects on different planets.

III. Physical Model

In the model we employed, based on [1], it primarily comprises the following components. The bending of light as it passes through the atmosphere and the influence of planetary gravity on light are disregarded.

A. *Volume Rendering Equation (VRE)*

Volume Rendering Equation (VRE) is commonly used to compute light transmission in participating media such as smoke or fog.

As described in [6], in the VRE:

$$L(P,\omega) = \int_{x=0}^{d} T(x)[\sigma_a L_e(x,\omega) + \sigma_s L_i(x,\omega)]dx + T(M)L(M,\omega) \quad (1)$$

- P, ω:The radiance observed in the direction ω at point P.
- T(x): The transmittance function from point P to x, representing the attenuation of light as it travels through the medium.
- σ_a: The absorption coefficient.
- L_e(x, ω): Emitted radiance in direction ω at point x within the medium.
- σ_s: Scattering coefficient.
- L_i(x, ω): Incident radiance in direction ω at point x within the medium.
- T(M): Transmittance function from the boundary M of the medium to point P.
- L(M, ω): Radiance at the boundary M of the medium in direction ω.

This equation takes into account the attenuation, scattering, and self-emission of light as it travels from one point to another.

B. *Rayleigh Scattering*

In the atmosphere, when the diameter of air molecules is much smaller than the wavelength of light, we can use the Rayleigh Theory [7, 8] to describe the scattering they cause.

In the Rayleigh scattering coefficient equation:

$$\sigma_s^{Rayleigh}(\lambda,h) = \frac{8\pi}{3}\left(n^2-1\right)^2 \frac{\rho(h)}{N\lambda^4} \quad (2)$$

- λ: Wavelength of light.
- h: Height in the atmosphere.
- n: Refractive index of the atmosphere.
- ρ(h): the density function of particles.
- N: Number of molecules per unit volume at sea level.

In the Rayleigh phase equation:

$$F_{Rayleigh}(\theta) = \frac{3}{16\pi}(1+\cos^2\theta) \quad (3)$$

- θ: The scattering angle.

Combine (2) and (3) by multiplying them, we get the full equation of Rayleigh scattering function.

Rayleigh scattering primarily explains the scattering of longer wavelength light, such as blue light, by gas molecules in the atmosphere, which is the main reason why the sky appears blue.

C. *Mie Scattering*

Mie scattering occurs when particle sizes are similar or larger to light wavelengths [9].

In the Mie scattering coefficient equation:

$$\sigma_s^{Mie}(\lambda,h) = \frac{8\pi}{3}\left(n^2-1\right)^2 \frac{\rho(h)}{N} \quad (4)$$

In the Mie scattering phase function:

$$F_{Mie}(\theta) = \frac{3}{8\pi}\frac{1-g^2}{2+g^2}\frac{1+\cos^2\theta}{(1-g^2-2g\cos\theta)^{3/2}} \quad (5)$$

- g: The asymmetry parameter, which ranges from -1 to 1. A value less than 0 indicates backward scattering, and a value greater than 0 indicates forward scattering. It is noticeable that when g = 0, the phase function becomes the Rayleigh phase equation.

Combine (4) and (5) by multiplying them, we get the full equation of Mie scattering function.

Unlike Rayleigh scattering, Rayleigh scattering is wavelength-independent, affecting a wide spectrum from ultraviolet to infrared. It explains light scattering by atmospheric particles like fog and smoke.

IV. Implementation Process

A. Setup

In setting up our Atmosphere Scattering project, we first established a robust technical foundation. We selected VulkanSDK v1.3.268.0 [10], a graphics and compute API, for its efficiency and control over GPU operations. Alongside this, we integrated several key libraries into our Visual Studio 2019 v16.111 [11] environment to support various functionalities: glfw v3.3.8-WIN64 [12] for image output, facilitating the rendering of our atmospheric effects; glm v0.9.9.8 [13] for advanced mathematical calculations, crucial for accurate light scattering modeling; stb for image loading, allowing us to incorporate complex textures; and tinyobjloader v2.0 [15] for loading .obj files, essential for our 3D models.

Further, we incorporated equations essential for the basic operation of Vulkan, complemented by camera code and model loading capabilities. As illustrated in Fig.1, our camera setup allows for free movement with adjustable pitch and yaw angles, a feature critical for demonstrating the dynamic nature of light scattering in different atmospheric conditions.

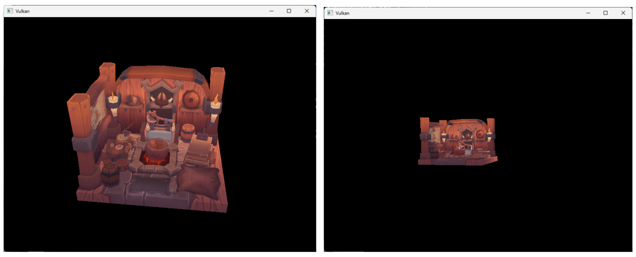


Fig. 1. Result of project setup and camera setup

B. *Sky*

In the next phase of our project, we transitioned from Vulkan's default object to a spherical model, positioning the camera at its center to effectively simulate a skybox. We then integrated the Rayleigh and Mie scattering equations, along with the constants derived from the seminal work of Nishita and Sébastien Hillaire.

At this stage, we had not yet implemented ray marching, which meant our renderings didn't fully depict the atmosphere's dynamic range. Instead, they displayed the Rayleigh and Mie scattering effects assuming a constant particle density ratio ($\rho(h) = 1.0$). This approach visualized the Scatter Coefficient, providing an intermediate view of how light scattering behaves under simplified conditions.

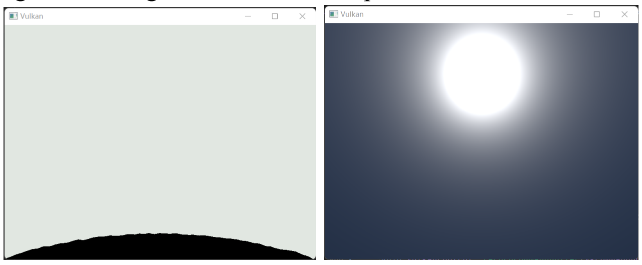


Fig. 2. Result of a raw sky box and after applied scatter coefficient

C. *Ray Marching*

After establishing the preliminary visual framework with the static scatter coefficient, we progressed to the implementation of ray marching to simulate the anisotropic density of the atmosphere, a crucial technique for rendering realistic atmospheric effects.

In our project, ray marching was employed to iteratively calculate the light paths as they travel through the atmosphere, accounting for the varying densities and compositions encountered. This approach allowed us to simulate the complex phenomena of light scattering by atmospheric particles, incorporating both Rayleigh and Mie scattering principles more dynamically.

Our implementation involved casting rays from the camera's position and marching them through the simulated atmosphere. At each step, we computed the scattering and absorption of light, integrating these values over the path of the ray. This step-by-step accumulation of light interaction provided a more accurate and visually compelling depiction of atmospheric scattering, showcasing nuances such as the gradual color shift of the sky at different angles and times of day.

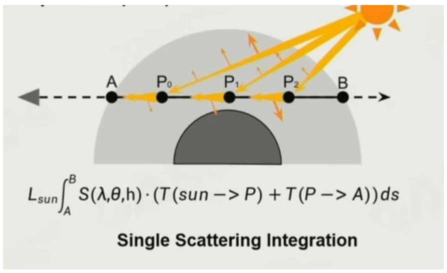

A
P_0
P_1
P_2
B
$L_{sun}\int_A^B S(\lambda,\theta,h)\cdot(T(sun -> P) + T(P -> A))ds$
Single Scattering Integration

Fig. 3. Single Scattering Integration Function [17]

Take one sample as an example; since we only need to consider the light that will be captured by the viewer, we can model scattering into two ray segments. The first segment, Viewer to Sample Point P, accounts for the path of light as it travels from the viewer's perspective to a designated point within the atmosphere. This segment is critical for determining how light is scattered and absorbed based on the viewer's location and the atmospheric conditions between the viewer and the sample point. The second segment, Sample Point P to the Sun, represents the journey of light from the sample point to the light source, which in our case is the Sun. This path is essential for computing the direct sunlight that reaches the sample point, factoring in any interactions with atmospheric particles. By calculating the light's behavior on both segments, we can accurately simulate the final color and intensity of light as perceived by the viewer, thus achieving a more realistic representation of atmospheric scattering effects.

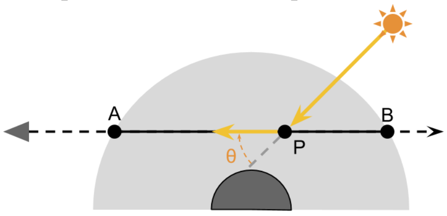


Fig. 4. Single Scattering Ray Sample [17]

With the outcomes of the ray marching process, we can apply these results to the Transmittance Attenuation function, denoted as the 'T function' in Fig 3. Then, we multiply it by the scattering function that incorporates the effects of both Rayleigh and Mie scattering. Finally, by combining these elements — the transmittance, scattering, and sun intensity — we obtain the realistic atmospheric effects as showcased in Fig 5. The result is a visually rich and accurate representation of the sky, capturing the dynamic and subtle interplay of light within the Earth's atmosphere.

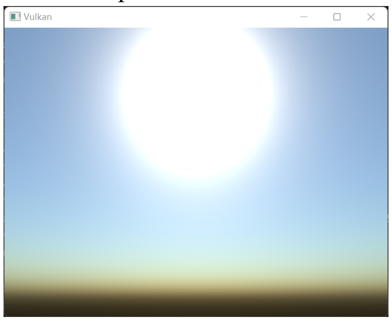


Fig. 5. Rendering Result

V. Unique or Interesting things

A. Camera Implementation

In the initial simple implementation of the Camera, we noticed that no matter how the Camera was moved, its line of sight, or the LookAt direction, remained unchanged, always facing the coordinates set initially. Therefore, we attempted to add a 'front' directional vector to the Camera to dynamically change its LookAt, aiming to achieve a camera movement similar to that in Unity. We tried to implement such a camera following the tutorials on the OpenGL official website [12], but the behavior of the camera was very peculiar. As shown in Fig. 6. When the mouse was moved left and right, the camera underwent a Roll rotation. It took us a great deal of effort to finally resolve this issue.

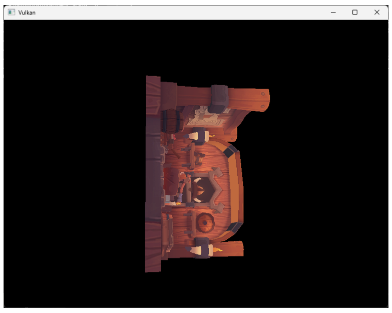


Fig. 6. Abnormally rotated model

B. Postprocess: Tonemapping

To address the issue of overexposure in our rendered images, particularly noticeable in areas with intense light such as the halo around the sun, we implemented a simple tone-mapping technique. This technique specifically targets colors with intensity values close to or exceeding 1. By clamping these values, we effectively reduce the harshness of overexposed areas, resulting in a smoother and more visually accurate depiction of high-intensity regions. The application of this tone-mapping technique significantly enhanced the overall realism and balance of our atmospheric visuals.

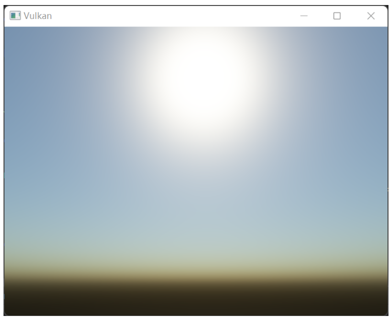


Fig. 7. Tonemapping result

C. Alien Atmosphere

By tuning the Rayleigh scattering coefficient, we were able to simulate exotic and alien atmosphere landscapes, as depicted in Fig 8. This adjustment allowed us to explore how changes in atmospheric composition could affect the scattering of light, resulting in unique and visually striking sky colors not typically seen on Earth. The variation in the Rayleigh coefficient alters the way shorter wavelengths of light are scattered, enabling us to create skies ranging from vivid purples to intense greens, demonstrating the versatility of our atmospheric scattering model.

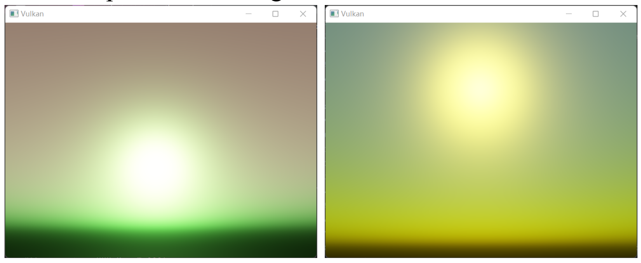


Fig. 8. Result with different Rayleigh coefficient and sun color

VI. Conclusion

In summary, our method employs the Vulkan API to partially implement the techniques outlined in [1]. We have successfully rendered skies with a high degree of physical realism. Our work contributes a novel approach to atmospheric rendering using the burgeoning Vulkan API. This project serves as a valuable guide and blueprint for developers working on mobile platforms or with performance-oriented game engines. Furthermore, it provides artists with highly customizable, efficient, and visually credible effects.

For further details, including access to our complete codebase, we invite readers to visit our GitHub repository at https://github.com/mcwang98628/Atmospheric_Scatter.

VII. Future Work

Future efforts will be directed towards a deeper exploration of the Vulkan API pipeline, designing LUT across various rendering pipelines to enhance rendering efficiency. We also aim to expand the implementation of various atmospheric rendering techniques and models through the Vulkan API, such as the impact of clouds, terrain, and oceans on atmospheric rendering. This is particularly relevant in the context of the increasing ubiquity of mobile devices and the escalating performance requirements of high-quality mobile game development.

References

[1] S. Hillaire, “A Scalable and Production Ready Sky and Atmosphere Rendering Technique,” *Computer graphics forum*, vol. 39, no. 4, pp. 13–22, 2020, doi: 10.1111/cgf.14050.

[2] 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

[3] Nishita T., Dobashi Y., Kaneda K., Yamashita H.: Display Method of the Sky Color Taking into Account Multiple Scattering, in Pacific Graphics ’96, pages 66–79, 1996

[4] E. Bruneton and F. Neyret, “Precomputed Atmospheric Scattering,” *Computer graphics forum*, vol. 27, no. 4, pp. 1079–1086, 2008, doi: 10.1111/j.1467-8659.2008.01245.x

[5] O. Elek, “Rendering Parametrizable Planetary Atmospheres with Multiple Scattering in Real-Time,” 2009.

[6] J. Fong, M. Wrenninge, C. Kulla, R. Habel, "Production Volume Rendering: SIGGRAPH 2017 Course," ACM SIGGRAPH 2017 Courses, New York, NY, USA: ACM, 2017, pp. 2:1–2:79.

[7] L. Rayleigh, "On the electromagnetic theory of light," Philos. Mag. J. Sci., vol. 12, no. 73, pp. 81–101, 1881.

[8] L. Rayleigh, "On the transmission of light through an atmosphere containing small particles in suspension, and on the origin of the blue of the sky," Philos. Mag. J. Sci., vol. 47, no. 287, pp. 375–384, 1899.

[9] Kerker, M. (1969). The scattering of light and other electromagnetic radiation. New York: Academic.

[10] LunarG, "Vulkan SDK - Home," 2023. [Online]. Available: https://vulkan.lunarg.com/sdk/home. [Accessed: 11/26/2023].

[11] Microsoft, "Visual Studio 2019 Downloads," 2023. [Online]. Available: https://my.visualstudio.com/Downloads?q=visual%20studio%202019&wt.mc_id=o~msft~vscom~older-downloads. [Accessed: 11/26/2023].

[12] GLFW, "GLFW - An OpenGL library," 2023. [Online]. Available: https://www.glfw.org/download.html. [Accessed: Date].

[13] G-Truc, "OpenGL Mathematics (GLM)," 2023. [Online]. Available: https://github.com/g-truc/glm. [Accessed: 11/26/2023].

[14] S. Barrett, "stb: single-file public domain libraries for C/C++," 2023. [Online]. Available: https://github.com/nothings/stb. [Accessed: 11/26/2023].

[15] S. Y. Kim, "tinyobjloader: Tiny but powerful single file wavefront obj loader," 2023. [Online]. Available: https://github.com/tinyobjloader/tinyobjloader. [Accessed: 11/26/2023].

[16] J. de Vries, "Camera - LearnOpenGL," 2023. [Online]. Available: https://learnopengl.com/Getting-started/Camera. [Accessed: Date].

[17] Alan Zucconi, "Volumetric atmospheric scattering," 2020. [Online]. https://www.alanzucconi.com/2017/10/10/atmospheric-scattering-1/. [Accessed: 11/26/2023].