Planetary Atmospheric Scattering

Jerry Lu*
luj10@rpi.edu
Rensselaer Polytechnic Institute
Troy, New York, USA

Evan Shi*
shib5@rpi.edu
Rensselaer Polytechnic Institute
Troy, New York, USA

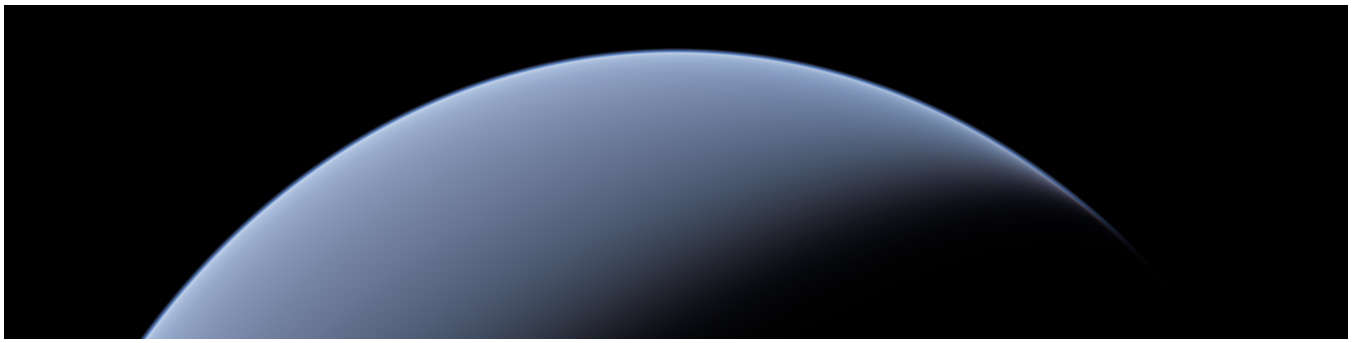

Figure 1. A view of the planet from outer space, created using our system.

Abstract

Atmospheric scattering is a complex process that occurs when sunlight interacts with particles in a planetary atmosphere, resulting in visually striking phenomena such as the blue sky and colorful sunsets. In this report, we present a comprehensive analysis of the precomputed atmospheric scattering method by Eric Bruneton and Fabrice Neyret [1], detailing our experimentation and implementation of this approach, which efficiently simulates the light scattering processes in real-time by precomputing the light transport equation, considering all possible viewpoints, view directions, and sun directions. By utilizing a series of precomputed look-up tables (LUTs) for key components such as transmittance, single scattering, multiple scattering, and ground irradiance, our implementation maintains a high level of performance and yields compelling results.

Keywords: Atmospheric scattering, precomputation, look-up table (LUT)

1 Introduction

Motivated by the interest to achieve a high degree of realism in environment simulation and inspired by the recent lecture on subsurface scattering, our group intends to investigate real-time sky and atmosphere rendering techniques that take multiple scattering into account. We aim to create high-quality atmospheric planets renderings that provide views ranging from ground-level to outer-space perspectives, thus presenting a visually compelling and immersive experience for the viewer. However, atmospheric scattering is a complex problem as it involves intricate light interactions with various particles and gases within a planet's atmosphere, and the light transport equation in a participating medium applied to the atmosphere is very difficult to solve. Therefore, many promises have been made in earlier works to render atmospheric phenomena such as sunsets, sky colors, and aerial perspectives in real-time.

Eric Bruneton and Fabrice Neyret's [1] approach aims to overcome these challenges by employing precomputed atmospheric scattering techniques. The method significantly reduces the computational demands commonly associated with atmospheric rendering using precomputed look-up tables for key components like transmittance, single scattering, multiple scattering, and ground irradiance. In this report, we explain our implementation of the approach in detail, showcasing the potential to generate visually appealing and accurate representations of planetary atmospheres while maintaining high performance. Due to time constraints, our implementation does not include the complete methodology described in the original paper; specifically, we omitted the illuminance as well as the shadows and light shafts component. Despite these modifications, our implementation still captures the essence of the original work and produces accurate and lovely renderings of the planetary atmosphere without compromising the overall results.

The next sections are organized as follows. Section 2 briefly introduces the physical model and the rendering equation used in previous works. Sections 3, 4, 5, and 6 present our implementation to precompute the transmittance, the single scattering, the second and higher orders of scattering, and the ground irradiance. Section 7 highlights miscellaneous tools employed within our rendering engine. Finally, section 8 showcases our results and discusses potential future works.

*Both authors contributed equally to this report.

2 Atmospheric models

Atmospheric scattering primarily consists of two physical phenomena: Rayleigh and Mie scattering, which are the main contributors to the colors of our sky. Rayleigh scattering is an optical phenomenon predominantly responsible for clear blue skies and vibrant orange-red sunsets. As light passes through the air, it interacts with small molecules whose wavelengths are much shorter, resulting in wavelength-sensitive scattering. In contrast, Mie scattering takes place during overcast weather and creates the distinctive Tyndall effect. It occurs when the size of airborne particles, such as those found in fog, smoke, and dust, is approximately equal to or greater than the wavelength of light. Unlike Rayleigh scattering, Mie scattering does not exhibit wavelength sensitivity and has a limited capacity to alter the color of incoming light. A more detailed overview of atmospheric scattering is shown in Figure 1. In this section, we first provide a brief overview of the physical model employed for the project, followed by an introduction to the rendering equation used to solve radiance. Additionally, we present a survey of important related works that informed and supported our project.

2.1 Physical model

Incorporating the aforementioned scatterings, we can develop a physical model consists of air molecules and aerosol particles, in a thin spherical layer of decreasing density between $R_g = 6360\ km$ and $R_t = 6420\ km$. At any given point, the proportion of light scattered θ degrees from its incident direction is determined by the product of a scattering coefficient β_s and a phase function P. The particle density influences β_s, while P delineates the angular dependency. For smaller air molecules, both β_s and P are defined by Rayleigh theory[8]:

$$\beta_R^s(h,\lambda) = \frac{8\pi^3\left(n^2-1\right)^2}{3N\lambda^4} e^{-\frac{h}{H_R}}$$
$$P_R(\mu) = \frac{3}{16\pi}\left(1+\mu^2\right)$$

where $\mu = \cos\theta$, $h = r - R_g$ is the altitude, λ is the wavelength, n is the index of refraction of air, N is the molecular density at sea level R_g, and $H_R = 8\ km$ is the thickness of the atmosphere if its density were uniform. Similarly, the Mie theory defines the scattering coefficient β_s and a phase function P for larger aerosols for a smaller height scale $H_M \approx 1.2\ km$ with an exponentially decreasing density[8]:

$$\beta_M^s(h,\lambda) = \beta_M^s(0,\lambda)\, e^{-\frac{h}{H_R}}$$
$$P_M(\mu) = \frac{3}{8\pi}\frac{\left(1-g^2\right)\left(1+\mu^2\right)}{\left(2+g^2\right)\left(1+g^2-2g\mu\right)^{\frac{3}{2}}}$$

Aerosols also absorb a fraction of the incident light, which is measured with an absorption coefficient β_M^a. Combining the absorption and scattering coefficient gives us the extinct

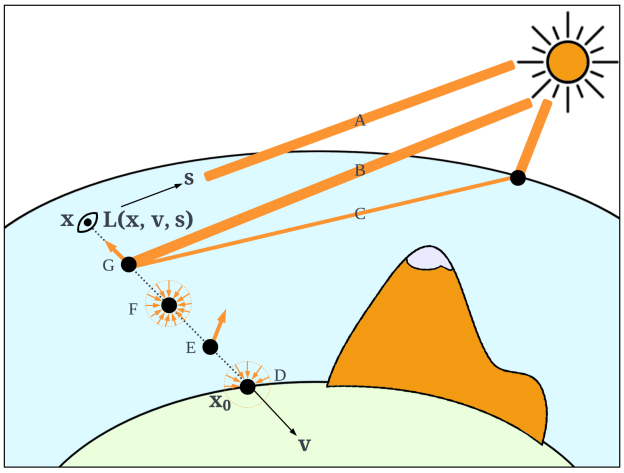

Figure 2. ***An overview of atmospheric scattering.*** *A: zero scattering. B: single scattering. C: multiple scattering. Solving for a physically accurate radiance of light L reaching* $\mathbf{x}$ *from direction* $\mathbf{v}$ *when the sun is in direction* $\mathbf{s}$ *requires: D: light reflected at* $\mathbf{x}_0$*. E: transmittance T results from absorption and out-scattered light. F: is the light scattered in direction* $-\mathbf{v}$*. G: light scattered towards* $\mathbf{x}$ *between* $\mathbf{x}_0$ *and* $\mathbf{x}$*, from any direction. Note that occlusion, like the mountain in the figure, can significantly affect the final calculated L. As a result, we account for occlusion in zero scattering but disregard it in other cases to minimize performance overhead.*

coefficient $\beta_M^e = \beta_M^a + \beta_M^s$ [1]. The ozone layer also has a large impact on the appearance of the Earth's atmosphere, and the absorption of ozone contributes to the sky's blue hue when the sun is near the horizon. We approximate the absorption rate by the following equation:

$$\beta_O^a(h,\lambda) = \beta_O^0(\lambda)\max\left\{0, 1 - \frac{\|h-25km\|}{15km}\right\}$$

In real-time rendering, we can not afford to compute light scattering for each individual wavelength. Instead, we approximate the energy distribution across the spectrum using the three RGB components, so the λ term is assigned to a certain value, referred to as λ_0. Therefore, we have the following functions for scattering coefficient β_s, extinction coefficient β_t and phase function P:

$$\beta_s(h,\lambda) = \beta_R^s(h) + \beta_M^s(h)$$
$$\beta_e(h,\lambda) = \beta_R^s(h) + \beta_M^e(h) + \beta_O^a(h)$$
$$P(\mu) = \frac{\beta_R^s(h)}{\beta_s(h)}P_R(\mu) + \frac{\beta_M^s(h)}{\beta_s(h)}P_M(\mu)$$

which we will use for calculating atmospheric scattering and discuss in later sections.

[1] $\beta_R^e = \beta_R^s$ for air molecules.

2.2 Rendering equation

In the work by Eric Bruneton and Fabrice Neyret, the rendering equation is presented as follows:

$$L(\mathbf{x}, \mathbf{v}, \mathbf{s}) = (L_0 + \mathcal{R}[L] + \mathcal{S}[L])(\mathbf{x}, \mathbf{v}, \mathbf{s})$$

where $L(\mathbf{x}, \mathbf{v}, \mathbf{s})$ represent the radiance of sunlight in direction $\mathbf{s}$, reaching point $\mathbf{x}_0$ from the viewing direction $\mathbf{v}$ (see Figure 2). In order to better understand the rendering equation, we need to specify a few additional terms:

$$T(\mathbf{x}, \mathbf{x}_0) = \exp\left(-\int_{\mathbf{x}}^{\mathbf{x}_0} \sum_{i \in \{R,M\}} \beta_i^e(\mathbf{y})\, \mathrm{d}y\right)$$

$$\mathcal{I}[L](\mathbf{x}_0, \mathbf{s}) = \frac{\alpha(\mathbf{x}_0)}{\pi} \int_{2\pi} L(\mathbf{x}_0, \boldsymbol{\omega}, \mathbf{s}) \boldsymbol{\omega}.\mathbf{n}(\mathbf{x}_0)\, \mathrm{d}\omega \text{ or } 0^2$$

$$\mathcal{J}[L](\mathbf{y}, \mathbf{v}, \mathbf{s}) = \int_{4\pi} \sum_{i \in \{R,M\}} \beta_i^s(\mathbf{y}) P_i(\mathbf{v}, \mathbf{w}) L(\mathbf{y}, \boldsymbol{\omega}, \mathbf{s})\, \mathrm{d}\omega$$

Assume the ray $\mathbf{x} + t\mathbf{v}$ extends to the atmosphere's boundary, where $\mathbf{x}_0$ is either at the ground or the atmospheric boundary where $r = R_t$, then T represents the transmittance between $\mathbf{x}_o$ and $\mathbf{x}$, $\mathcal{I}$ represents the radiance of light reflected at $\mathbf{x}_0$, and the radiance $\mathcal{J}$ of light scattered at $\mathbf{y}$ in direction $-\mathbf{v}$. With these notions, the terms in the rendering equation can be expanded as follows:

$$L_0(\mathbf{x}, \mathbf{v}, \mathbf{s}) = T(\mathbf{x}, \mathbf{x}_0) L_{sun} \text{ or } 0^3$$

$$\mathcal{R}[L](\mathbf{x}, \mathbf{v}, \mathbf{s}) = T(\mathbf{x}, \mathbf{x}_0) \mathcal{I}[L](\mathbf{x}_0, \mathbf{s})$$

$$\mathcal{S}[L](\mathbf{x}, \mathbf{v}, \mathbf{s}) = \int_{\mathbf{x}}^{\mathbf{x}_0} T(\mathbf{x}, \mathbf{y}) \mathcal{J}[L](\mathbf{y}, \mathbf{v}, \mathbf{s})\, \mathrm{d}y$$

where L_0 represents the attenuation value of direct sunlight L_{sun} as it reaches $\mathbf{x}$ via $T(\mathbf{x}, \mathbf{x}_0)$. $\mathcal{R}[L]$ denotes the light reflection at $\mathbf{x}_0$ and the attenuation value of the light before it reaches $\mathbf{x}$. Meanwhile, $\mathcal{S}[L]$ is the internal scattering towards $\mathbf{x}$ between $\mathbf{x}$ to $\mathbf{x}_0$.

2.3 Related works

Given the intricate and recursive characteristics of the previously mentioned rendering equation, solving the L is extremely difficult. Therefore, using naive approaches to attain real-time planetary atmospheric scattering in applications like game engines is impractical. Previous works have made numerous simplifying assumptions to approximate solutions requiring much less computational power. For instance, we can reduce the rendering equation to $L = (L_0 + \mathcal{R}[L_0] + \mathcal{S}[L_0])$ to only include the zero scattering. However, even $\mathcal{S}[L_0]$ is hard to solve in reality, leading some works to propose analytical solutions that rely on certain assumptions, such as a flat Earth with constant[3] atmospheric density or excluding Mie scattering[4]. Besides making simplifications to the physical model, there are approaches utilize precomputations[6]. Similarly, the works by Eric Bruneton and Fabrice Neyret[1] as well as Sébastien Hillaire[2], which we adopted for this project, relies on precomputation but renders the sky and aerial perspective in real-time. It considers all viewpoints, from ground level to space, and takes multiple scattering into account, which sets apart approaches that ignore multiple scattering (see [7] for a comprehensive survey). In addition, the technique can compute up to four scatterings in real-time, which is a significant improvement compared to the double scattering Monte-Carlo simulation with an analytical model[5] for multiple scattering computations. In the later sections, we will explain each precomputed look-up table in detail.

[2] $\mathcal{I}$ is null on the top atmosphere boundary.

[3] If $\mathbf{v} \neq \mathbf{s}$ or the sun is occluded by terrain, L_0 becomes null.

3 Transmittance

The transmittance T represents the proportion that remains unabsorbed or unattenuated after a beam of light travels from one point to another. By definition, T depends only on the positions of the two points and is independent of terrain. Therefore, we can examine it under the assumption of a smooth planetary surface (i.e., all distances from the surface to the sphere's center are equal to R) when solving for T. Upon determining T, we store it in a precomputed texture that will be accessed during the rendering process.

3.1 Transmittance model

In terms of computation, the transmittance between two points p and q is the transmittance between p and the nearest intersection i of the half-line $[p, q)$ with either the top or bottom atmospheric boundary (See Figure 3). We then divide this value by the transmittance between q and i (or use 0 if the segment $[p, q]$ intersects the ground). Because the transmittance values between p and q and between q and p are the same, it is sufficient to know the transmittance between a point p in the atmosphere and points i on the top atmosphere boundary to compute the transmittance between arbitrary points. The transmittance then depends on only two parameters, which can be taken as the radius $r = \|op\|$ and the cosine of the "view zenith angle," $\mu = op(\cdot pi / \|op\|\|pi\|)$. Based on the definition and the Beer-Lambert law, solving for transmittance involves integrating the density of air molecules, aerosols, and air molecules that absorb light (e.g., ozone) along the same segment $[p, i]$.

3.2 Transmittance precomputation

To store the calculated transmittance, we need to map the two parameters (r, μ) to the texture coordinates (u, v), and perform the inverse mapping when retrieving the solution during the rendering process. The original paper provided a generic mapping technique that works for any atmosphere and provides an increased sampling rate near the horizon: $r = \|x\|$, and μ equals a value x_μ between 0 and 1 by considering the distance d to the top atmosphere boundary, where $d_{\min} = r_{\text{top}} - r$ and $d_{\max} = \rho + H$ (See Figure 3).

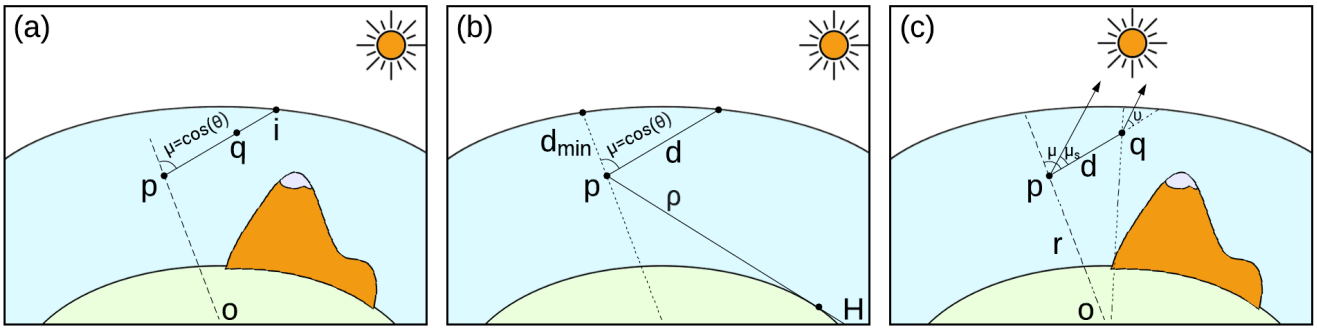

Figure 3. ***Details about transmittance, texture mapping, and single scattering.*** *(a): the transmittance between p and q. (b): mapping between (r, μ) and the texture coordinates (u, v). (c): the single scattered radiance at p after the scattering event at q.*

3.3 Transmittance look-up

While rendering, we can get the transmittance between two arbitrary points p and q inside the atmosphere using just two texture look-ups. This is because the transmittance between p and q equals the transmittance between p and the top atmospheric boundary divided by the transmittance between q and the top atmospheric boundary, and vice versa. It is worth to mention that we also need the atmospheric transmittance to the Sun when calculating single scattering and sky irradiance. During which step, we approximate by integrating over its disc, assuming constant transmittance except below the horizon. We calculate it using top atmosphere boundary transmittance multiplied by the visible Sun disc fraction. It is important to note that we do not perform the ray-ground intersection testing step here. Instead, we implement the function and use it on the caller side. This is because results could be inaccurate for rays close to the horizon, as finite precision and rounding errors in floating-point operations can cause discrepancies.

4 Single scattering

By single scattering, we mean that there has only been one scattering event since the light left the light source before it reached the observer's eye. The single scattered radiance is responsible for the majority of the color we see because the atmosphere scatters light at a relatively low rate.

4.1 Single scattering model

Consider the Sun light scattered at a point q by air molecules or aerosols particles before arriving at another point p, the radiance arriving at p is the product of:

- the solar irradiance at the top of the atmosphere.
- the fraction of the Sun light at the top of the atmosphere that reaches q, $T(\text{Sun}, q)$.
- the Rayleigh or Mie scattering coefficient at q.
- the Rayleigh or Mie phase function.
- the fraction of the light scattered at q towards p that reaches p, $T(q, p)$.

where a total of 4 parameters needed (See Figure 3):

- r is the distance from p to the center of the sphere.
- μ is the angle formed by extending the radius from point p to atmosphere boundary and the line segment connecting points p and q.
- μ_s is the angle formed by extending the radius from point p to atmosphere boundary and the unit direction vector towards the Sun ω_s.
- ν is the angle formed by the same unit direction vector for μ_s and line segment connecting points p and q.

4.2 Single scattering precomputation

Single scattering is quite expensive, and many evaluations are needed to compute multiple scattering due to the recursive nature of the rendering equation. We, therefore, want to precompute the solution and save the results to texture, which requires a mapping from the 4 function parameters to texture coordinates. This requires a 4D texture, and a function that maps (r, μ, μ_s, ν) to texture coordinates (u, v, w, z).

The mapping between r, ν and u, z roughly follows the same procedure we mentioned in the previous section. However, the mapping between μ, μ_s and v, w is more complex. The mapping for μ takes the minimal distance to the nearest atmosphere boundary into account to map it to [0,1] interval. On the other hand, the mapping for μ_s relies on the distance to the top atmosphere boundary and uses a configurable parameter, but still maintains a higher sampling rate near the horizon.

4.3 Single scattering look-up

With the help from precomputed texture generated using the method discussed previously, we can now get the scattering between a point and the nearest atmosphere boundary with two texture look-ups in real-time, which guarantees efficient multiple scattering calculations described in the next section.

In particular, we employ two 3D texture look-ups to simulate a single 4D texture lookup using quadrilinear interpolation.

5 Multiple scattering

By multiple scattering, we mean that the light from the Sun reaches the observer's eye after undergoing two or more bounces within the atmosphere. These bounces consist of scattering events, where light interacts with atmospheric particles, or reflections off the ground.

5.1 Multiple scattering model

Multiple scattering can be broken down into the sum of double scattering, triple scattering, and so on, with each term representing light reaching a point in the atmosphere after exactly two, three, etc., bounces. In addition, each term can be calculated based on the previous one; that is, the light arriving at some point p from direction ω after n bounces is an integral over all the possible points q for the last bounce, which involves the light arriving at q from any direction, after $n-1$ bounces.

Unfortunately, the calculation at each scattering order requires a triple integral based on the previous scattering order: one integral over all the points q on the line segment from p to the nearest atmosphere boundary in direction ω, as well as a nested double integral over all directions at each point q. It will be extremely inefficient if we perform the calculation naively. As suggested by Eric Bruneton and Fabrice Neyret in their work[1], we use the following algorithm when dealing with multiple scattering:

Algorithm 1 Multiple scattering computation

```
1:  Precompute single scattering in a texture.
2:  for scattering order ≥ 2 do
3:      for each point q do
4:          for each direction ω do
5:              Look up (n − 1)-th scattering texture
6:              Compute the scattered light at q towards −ω
7:          end for
8:      end for
9:      for each point p do
10:         for each direction ω do
11:             Look up the texture computed on line 6
12:             Compute the scattered light at p towards ω
13:         end for
14:     end for
15: end for
```

where the calculations for p involve only a double integral, and the calculations for q involve only a single integral, since it is based on the precomputed texture in the previous step.

5.2 Multiple scattering precomputation

To save computation time when calculating the next order, we must precompute each scattering order in a texture, which requires a mapping from function parameters to texture coordinates. Fortunately, all scattering orders rely on the same (r, μ, μ_s, ν) parameters as the single scattering described in Section 4. Therefore, we can conveniently reuse the mappings defined for single scattering.

5.3 Multiple scattering look-up

Likewise, we can simply reuse the same look-up procedure for single scattering to read a value from the precomputed textures for multiple scattering.

6 Ground irradiance

Ground irradiance, which is the sunlight received at the planet's surface after $n \geq 0$ bounces, plays an important role in precomputing the n-th order of scattering, where $n \geq 2$, in determining the light path contributions with their $(n-1)$-th bounce on the ground. In this case, we require ground irradiance only for horizontal surfaces situated at the bottom of the atmosphere for a uniform albedo spherical planet. Furthermore, during the rendering process, we need to compute the contribution of light paths with their final bounce on the ground to achieve an accurate result. Different from the previous case, we need ground irradiance for varying altitudes and surface normals, and precomputation is desired for efficiency. In line with the original work, we precompute irradiance only for horizontal surfaces at any altitude, using 2D textures as a substitute for the more complex 4D textures required in the general case.

6.1 Ground irradiance model

Irradiance is calculated as the integral over a hemisphere of the incident radiance, multiplied by a cosine factor. For ground irradiance computation, we divide it into direct and indirect components. The direct ground irradiance is determined by the Sun's incident radiance at the top of the atmosphere, multiplied by the transmittance through the atmosphere. Given the small solid angle of the Sun, we can approximate transmittance as a constant and move it outside the irradiance integral, which should be performed over the visible fraction of the Sun's disc rather than the entire hemisphere. For indirect ground irradiance, the integral over the hemisphere must be calculated numerically. Specifically, we need to compute the integral over all directions ω of the hemisphere, considering the product of the radiance arriving from direction ω after n bounces and the cosine factor ω_z.

6.2 Ground irradiance precomputation

As mentioned in the previous section, the irradiance depends only on r and μ_s because we precompute the ground irradiance only for horizontal surfaces. Therefore, we only need a mapping from ground irradiance parameters (r, μ_s) to texture coordinates (u, v), and a simple affine mapping is sufficient because of the smooth ground irradiance function.

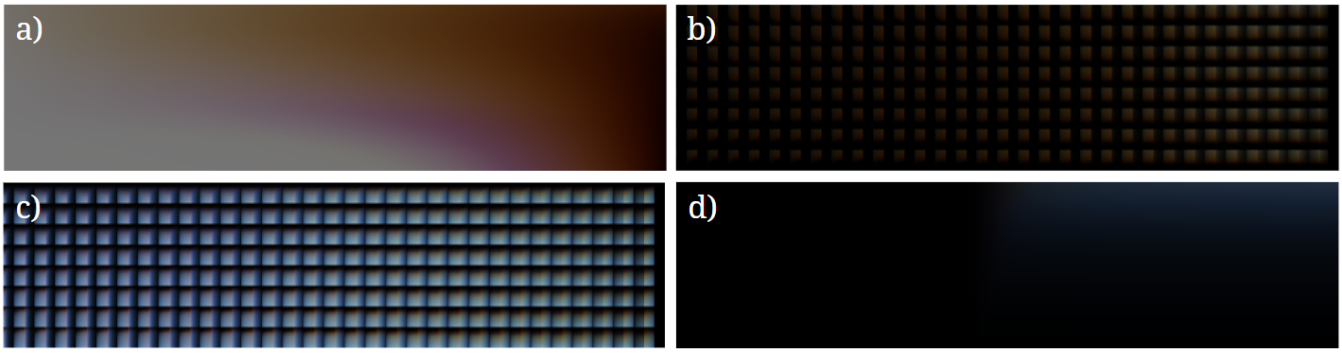

Figure 4. ***Look-up tables/textures.*** *(a): The 2D transmittance texture. (b): The 4D single scattering texture. (c): The 4D scattering texture that combines all scattering orders (4 scattering orders in this case). (d) The 2D ground irradiance texture.*

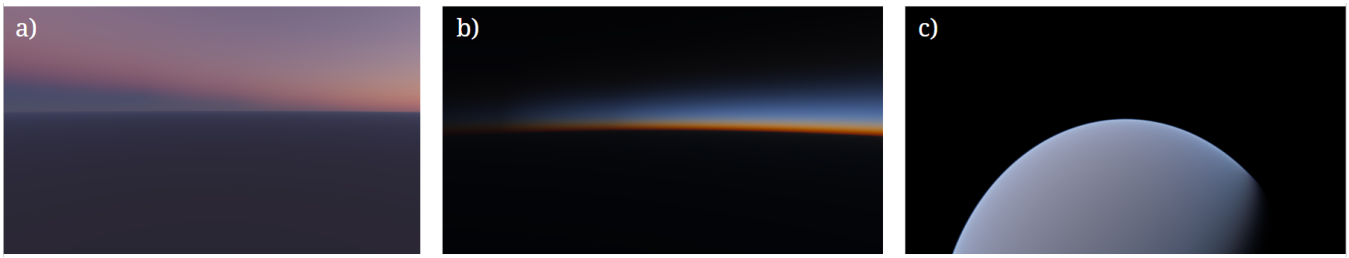

Figure 5. ***Planetary atmospheric scattering results from different camera angles and positions.*** *(a): A dawn scene from the planetary surface [Radiance]. (b) A sunrise scene from the planetary surface. (c) An outer space scene that showcases planetary atmospheric scattering on a larger scale.*

6.3 Ground irradiance look-up

The ground irradiance look-up is relatively straightforward, as it can be accomplished with just a single texture look-up similar to what we did for transmittance textures.

7 Miscellaneous

The section briefly discusses the engine we created to generate the results, our testing strategies and the work distribution for the project.

7.1 Engines

We have developed two distinct engines to showcase the outcomes of our project: a command-line-based engine and a graphical user interface (GUI) based engine. The first engine, the command-line-based variant, is capable of generating a two-dimensional image of a predefined scene in real-time, with users having the ability to customize various elements such as camera angles and positioning. The second engine, utilizing a graphical interface, offers users an enhanced level of control over the scene, including adjustments to sun position, color balance, and even vertex and fragment shaders modifications within the engine. These features enable users to manipulate a range of aspects, creating a more immersive and interactive experience. Both engine utilizes precomputed LUTs from the atmosphere model we created.

7.2 Testing

Our testing approach is relatively simple: We examine the resulting image to determine if it makes sense. Given the calculations' complexity and the project's limited debugging time, we couldn't guarantee that the programs' calculations were entirely accurate. We thoroughly reviewed the model multiple times, noting that minor errors have minimal impact on the image and can sometimes be challenging to detect. Additionally, we incorporated numerous assertions within the code to ensure that basic calculations and look-ups were in the correct ranges. Nevertheless, we still encountered many errors during the process. Figure 6 shows some of our incorrect results.

7.3 Work distribution

As the project advanced, our team worked diligently to organize and assign tasks to efficiently and effectively accomplish the core components mentioned in the proposal. Each team member dedicated around 50 hours to the project, and while we did not complete everything outlined in the proposal due to limited time, we successfully implemented the basic planetary atmospheric scattering, which was the project's primary focus.

Jerry Lu developed the CPU renderer, a command-line-based engine that produces 2D images of a scene based on

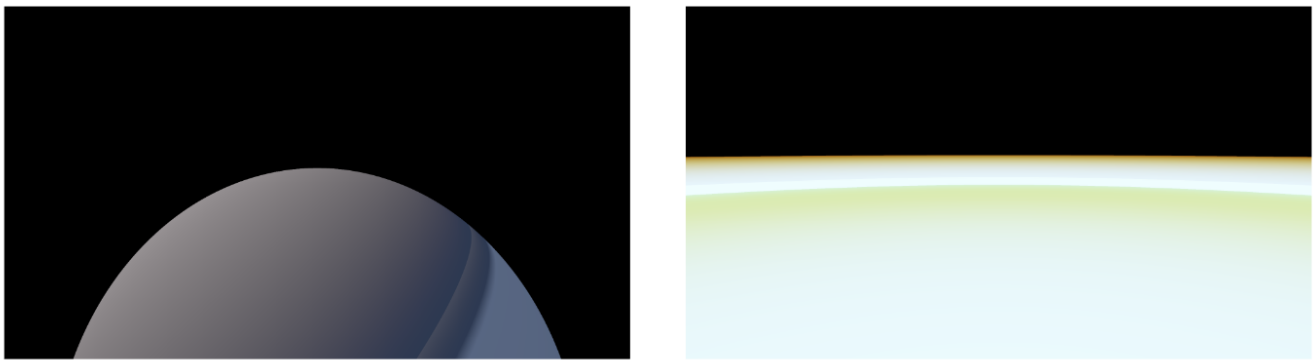

Figure 6. ***Precomputation failures.*** *These were created by our first functional CPU renderer. The precomputation for the planetary atmospheric scatterings was incorrect in the first place, which resulted in incorrect LUTs and broken scenes. The first image shows a broken dawn scene caused by errors in scattering calculation. The second image shows a broken planet caused by errors in texture look-up.*

user-defined parameters. Jerry concentrated on single and multiple scattering calculations, as well as ground irradiance for planetary atmospheric scattering. Moreover, he optimized look-up tables and developed essential utilities for loading and saving precomputed parameters, which enables fast access to precomputed data for rendering purposes.

Meanwhile, Evan Shi centered his efforts on completing the GPU renderer for the project, a graphics-based engine that renders scenes in real-time using precomputed textures. This engine allows users to adjust various parameters to achieve different visual outcomes. In addition, Evan contributed to the project by implementing transmittance and multiple scattering calculations, as well as doing most of the report writing.

8 Results and discussion

All rendering of this paper is provided by the CPU renderer, which samples the entire spectrum to give a radiance spectrum of pixels. Based on this radiance spectrum, we can calculate the color of the pixels in different ways. If not explicitly stated, the result is to first integrate the radiance spectrum with the CIE XYZ basis, converting it to CIE XYZ luminance and then to linear sRGB luminance. The output images are saved as PNG at 8-bit for each channel, after the luminance is gamma corrected ($\gamma = 2.2$) and tone mapped. Figure 5 (a) shows a less-accurate result with only 3 radiance samples without converting to sRGB. The GPU renderer aims to be real-time by sacrificing the color accuracy, which only samples at RGB wavelengths for radiance.

Precomputation takes about 200 seconds to generate all the LUTs in 20 threads with fourth order multiple scattering. Once generated, the LUTs are simply copied to memory and rendered immediately. The GPU renderer can read half precision LUTs and render the atmosphere at more than 144 frames per second, and can adjust the camera and the position of the sun in real time.

With the limited time, we could only work on implementing Eric Bruneton and Fabrice Neyret's work. As outlined in the introduction, an efficient model [2] further speed up the computation. Another future work that could fully exploit the ability of rendering full radiance spectrum is to enable the engine to write RGBE files, which could preserve all the radiance data for high dynamic range displays. Although we also mentioned cloud rendering in the proposal as a side quest, this could also be a future work that could render the interactions between the atmosphere and realistic clouds.

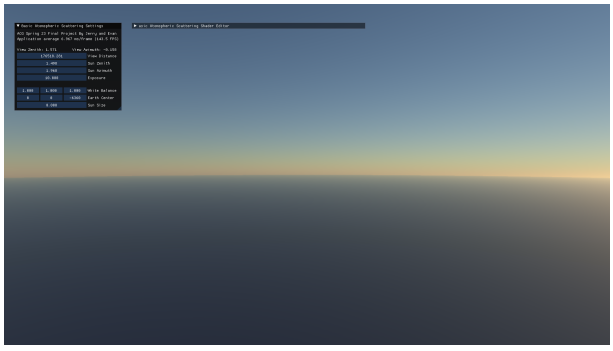

Figure 7. ***Screenshot of the OpenGL version***

Acknowledgments This project, conducted for RPI Advanced Computer Graphics Spring 23, is primarily built upon the prior works of Eric Bruneton and Fabrice Neyret. We express our gratitude to them for supplying valuable references.

References

[1] Eric Bruneton and Fabrice Neyret. 2008. Precomputed Atmospheric Scattering. *Computer Graphics Forum* 27, 4 (June 2008), 1079–1086. https://doi.org/10.1111/j.1467-8659.2008.01245.x

[2] Sébastien Hillaire. 2020. A Scalable and Production Ready Sky and Atmosphere Rendering Technique. *Computer Graphics Forum* 39 (2020), 13–22. https://doi.org/10.1111/cgf.14050 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14050

[3] Naty Hoffman and Arcot J. Preetham. 2002. Rendering outdoor light scattering in real time. *Proceedings of Game Developer Conference* (2002).

[4] H. Jensen, Kirk Riley, David Ebert, Martin Kraus, Jerry Tessendorf, and Charles Hansen. 2004. Efficient Rendering of Atmospheric Phenomena. (07 2004).

[5] A. J. Preetham, Peter Shirley, and Brian Smits. 1999. A Practical Analytic Model for Daylight. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99)*. ACM Press/Addison-Wesley Publishing Co., USA, 91–100. https://doi.org/10.1145/311535.311545

[6] T Schafhitzel, Martin Falk, and Thomas Ertl. 2007. Real-Time Rendering of Planets with Atmospheres. *Journal of WSCG 2007* 15 (01 2007), 91–98.

[7] Jaroslav Sloup. 2002. A Survey of the Modelling and Rendering of the Earth's Atmosphere. In *Proceedings of the 18th Spring Conference on Computer Graphics* (Budmerice, Slovakia) *(SCCG '02)*. Association for Computing Machinery, New York, NY, USA, 141–150. https://doi.org/10.1145/584458.584482

[8] Knut Stamnes, Gary E. Thomas, and Jakob J. Stamnes. 2017. *Radiative Transfer in the Atmosphere and Ocean* (2 ed.). Cambridge University Press. https://doi.org/10.1017/9781316148549