Dereyne Kobe

Ozone Absorption in Single Atmospheric Scattering

Supervisor: Goussaert Thomas

Coach: Defoort Stephanie

Graduation Work 2025-2026

Digital Arts and Entertainment

Howest.be

1 CONTENTS

2 ABSTRACT & KEY WORDS

(EN)

Atmospheric scattering can play a key role in the visual realism of outdoor scenes in computer graphics. While many real-time implementations focus on Rayleigh and Mie scattering, the absorption effects of the ozone layer are often overlooked or ignored. This thesis will investigate how the ozone layer affects the colour of the sky in a single scattering atmospheric model. The focus will lie on the perceived realism ozone absorption adds, as well as the computational cost it brings with it, if any. In addition, the visual and performance impact of different phase functions for Mie scattering is evaluated.

A physically-based single scattering model was implemented using the Vulkan graphics API. Visual differences were assessed through a blinded, pairwise comparison user survey across varying sun elevations, while performance was measured using a GPU profiling tool. The results show that the inclusion of ozone absorption does not introduce a significant performance cost but does influence perceived realism in a context dependent manner. Ozone absorption was preferred at higher sun elevations, while its inclusion at low sun angles reduced perceived realism within a single scattering framework. Differences between phase functions were found to be subtle, with no meaningful impact on performance.

Key words: atmospheric scattering, single scattering, ozone absorption, Rayleigh scattering, Mie scattering, phase functions

Dereyne Kobe

(NL)

Atmosferische verstrooiing speelt een belangrijke rol in het visueel realisme van buitenscènes in computer graphics. Hoewel veel real-time implementaties zich richten op Rayleigh- en Mie-verstrooiing, worden de absorptie-effecten van de ozonlaag vaak genegeerd of vereenvoudigd. Deze thesis onderzoekt hoe de ozonlaag de kleur van de lucht beïnvloedt binnen een enkelvoudig verstrooiingsmodel. De focus ligt op de subjectieve realiteit die ozonabsorptie toevoegt, evenals op de eventuele rekenkosten die hiermee gepaard gaat. Daarnaast wordt ook het visuele effect en de systeembelasting van verschillende fasefuncties voor Mie-verstrooiing onderzocht.

Een fysisch gebaseerd enkelvoudig verstrooiingsmodel werd geïmplementeerd met behulp van de Vulkan graphics API. Visuele verschillen werden geëvalueerd aan de hand van een geblindeerde, paarsgewijze vergelijkingsenquête bij verschillende zonnestanden, terwijl systeembelasting werd gemeten met behulp van een GPU-profiling tool. De resultaten tonen aan dat het toevoegen van ozonabsorptie geen significante impact heeft op de systeembelasting, maar wel een contextafhankelijke invloed heeft op de subjectieve realiteit. Ozonabsorptie werd verkozen bij hogere zonnestanden, terwijl de opname ervan bij lage zonnestanden de waargenomen realiteit binnen een enkelvoudig verstrooiingsmodel verminderde. Verschillen tussen fasefuncties bleken subtiel en hadden geen betekenisvolle invloed op de systeembelasting.

Kernwoorden: atmosferische verstrooiing, enkelvoudige verstrooiing, ozonabsorptie, Rayleigh-verstrooiing, Mie-verstrooiing, fasefuncties

3 ABBREVIATIONS AND SYMBOLS

Abbreviation	Meaning
API	Application Programming Interface
CPU	Central Processing Unit
FPS	Frames per Second
GPU	Graphics Processing Unit
HDR	High Dynamic Range
LDR	Low Dynamic Range
LUT	Look-Up Table
MFS	Microsoft Flight Simulator
OS	Operating System

Table 1: Abbreviations

4 PREFACE

For well over the majority of my life so far, I have been interested in movies and games, more specifically in the underlying technology and processes that make them possible. Since starting my first bachelor at Digital Arts and Entertainment with a specialization in Game Development, this interest developed into a strong focus on graphics programming. Researching rendering techniques, watching videos, programming myself, all to be able to learn more.

During this period, I became particularly interested in cloud and sky rendering, as well as volumetric techniques such as ray marching. These methods aim to realistically simulate atmospheric phenomena and play a crucial role in achieving visual realism in modern games and simulations. The visual complexity of skies, combined with the physical processes that govern light interaction in the atmosphere, makes atmospheric rendering both a challenging and compelling topic.

For these reasons, this thesis focuses on atmospheric scattering, exploring the physical principles behind the colour of the sky and how the Earth's ozone layer plays a role in that.

5 LIST OF FIGURES

6 LIST OF EQUATIONS

7 LIST OF TABLES

8 LIST OF CODE BLOCKS

9 INTRODUCTION

The sky is something you see in movies or games more often than you might realise, it's something that you usually don't pay much attention to most of the time because it feels familiar and natural. However, at the same time, an inaccuracy might throw you off, as we can be quite sensitive to subtle variations. Accurately simulating the sky in real-time applications needs to be looked at with care if we are going for realism. This is especially important in contexts where the sky plays a dominant role, such as flight simulators, space simulations, or cinematic environments, where its appearance can significantly influence immersion and perception.

Figure 1: Microsoft Flight Simulator from [1]

Figure 2: MFS Weather Add-On from [2]

This thesis investigates the physics behind atmospheric scattering and how different physical concepts effect the visuals and realism of the sky. More specifically, this paper takes a deeper look at how ozone absorption can affect the colour of the sky in a single scattering atmospheric model, as well as how phase functions influence the appearance of the sky and sun.

The first part of this paper will delve deeper into atmospheric scattering and its effects, as well as some already existing models and the math behind them.
The second part will present the scattering model used in this paper, with ozone absorption, as well as the visual and performance results. Furthermore, these results will be analysed, discussed, and a conclusion to answer the research question will be formulated.

The research question this paper seeks to answer is:
How does a single-scattering atmospheric scattering model compare visually and computationally when extended with ozone absorption and alternative phase function approximations?

The aim is to investigate how incorporating an additional physical effect, such as ozone absorption, influences the perceived colour of the sky, and how using different phase function approximations affects both the visual result and the computational performance of real-time rendering models.

To be able to conduct this research and be able to answer the research question, the following hypotheses have been formulated:

H0: *Including ozone absorption or using different phase functions results in no significant visual or performance differences compared to the baseline single scattering model.*

H1: *Ozone absorption produces measurable sky-colour differences, while variations in phase functions do not. Any performance impact comes primarily from ozone absorption, not from phase-function changes.*

H2: *Ozone absorption produces significant visual differences only at low sun elevations (e.g., sunrise and sunset), with negligible impact at higher sun angles.*

H3: *Different phase function approximations produce subtle visual differences, but do not significantly affect GPU performance.*

10 LITERATURE STUDY / THEORETICAL FRAMEWORK

10.1 INTRODUCTION TO ATMOSPHERIC SCATTERING

"*Atmospheric Scattering*" refers to the interaction between sunlight and particles in Earth's atmosphere, resulting in the redirection and modification of the incoming sunlight. The way this light is scattered depends primarily on the size and composition of the particles it encounters, as well as the wavelength of the light itself. All these interactions are responsible for a variety of observable atmospheric phenomena, such as the colour of the sky, and the appearance of the sun itself.

Figure 3: Colour of the sky (example 1) from [3]

Figure 4: Colour of the sky (example 2) from [4]

Figure 5: Appearance of the Sun (example 1) from [5]

Figure 6: Appearance of the Sun (example 2) from [6]

There are different processes that contribute to atmospheric scattering. Rayleigh scattering is when light interacts with particles much smaller than the wavelength of the light [4], while Mie scattering is more significant for larger particles [7]. Additionally, the ozone layer also absorbs a small fraction of the incoming light [8].

It is also important to point out the difference between single and multi-scattering. With single scattering, light undergoes exactly one scattering event before reaching the viewpoint, while with multi-scattering that light can undergo several scattering events. In real life multi-scattering takes place, but a lot of atmospheric scattering models simplify to single scattering to reduce computational cost.

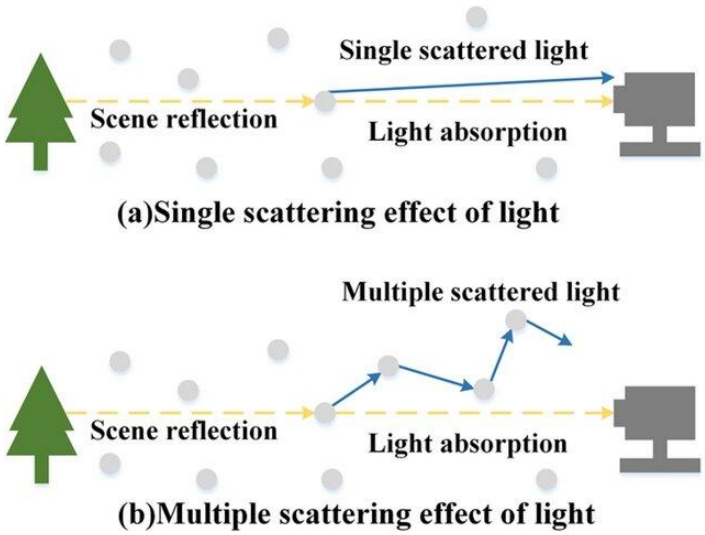

Figure 7: Single vs Multiple Scattering from [9]

The following sections will provide an overview of these atmospheric effects and their significance, as well as the mathematical model and physical principles behind them.

10.2 OVERVIEW OF THE ATMOSPHERIC EFFECTS

10.2.1 RAYLEIGH SCATTERING

Rayleigh scattering is the scattering of electromagnetic radiation, such as light, by particles that are smaller than the wavelength of the radiation itself [4]. It is named after physicist Lord Rayleigh after he published a paper describing the phenomenon in 1871 [10].

The amount of light dispersion due to Rayleigh scattering is inversely proportional to the fourth power of the wavelength [4]:

$$I(\lambda) \propto \frac{1}{\lambda^4}$$

Equation 1: Proportion of Rayleigh scattering to wavelength.

Where I is the scattered light intensity, and λ the wavelength. This means that shorter wavelengths such as blue ($450 - 485\ nm$) or violet ($380 - 450\ nm$) get scattered significantly more than longer wavelengths such as red ($625 - 750\ nm$) or green ($500 - 565\ nm$) [11].

It is mainly because of this wavelength dependent scattering that explains why the Earth's sky appears blue during the day. As sunlight enters the atmosphere it interacts with the small air particles along its path that cause Rayleigh scattering. The red and green light get scattered a little bit, but not nearly as much as the blue light [12]. The heavy scattering of blue light makes the sky appear blue.

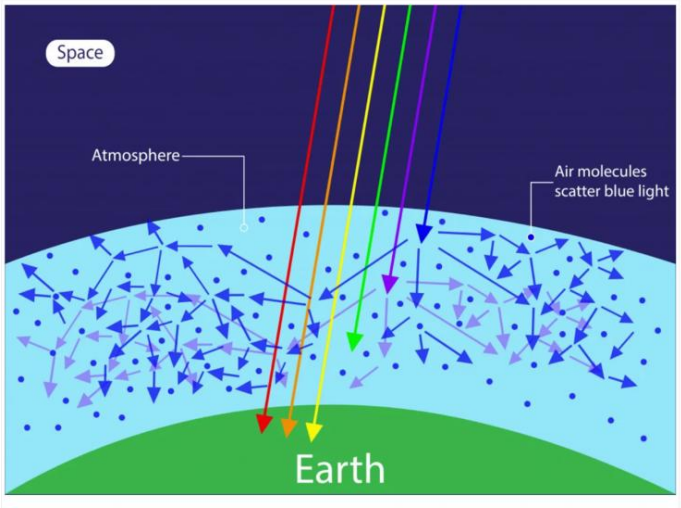

Figure 8: Scattering of light in the atmosphere from [12]

But why does the sky look reddish orange at sunset then? The reason is the exact same, however when the sun is at a lower angle relative to the viewpoint, the sunlight needs to travel through much more atmosphere, and thus much more scattering occurs. Nearly all the blue and even green light gets scattered away, leaving only the yellow red and orange light to reach your eye [12], [13].

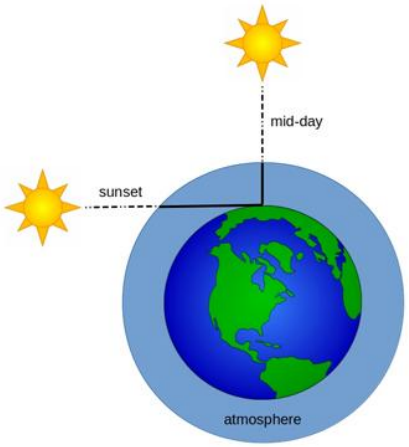

Figure 9: Light travelling longer through the atmosphere during sunset from [13]

10.2.2 MIE SCATTERING

Mie scattering is the scattering of light by particles that have a similar or larger diameter to the wavelength of the incident light. These bigger particles are called aerosols. An aerosol in this context is a particle such as fog, dust, or mist [14]. Mie scattering is named after physicist Gustav Mie after he published a paper on the phenomenon in 1908 [15].

In contrary to Rayleigh scattering, Mie scattering does not strongly depend on the wavelength of the incident light. This means that Mie scattering is not predominantly the reason why the sky is blue, however it is the reason why we see a white glare around the sun [7].

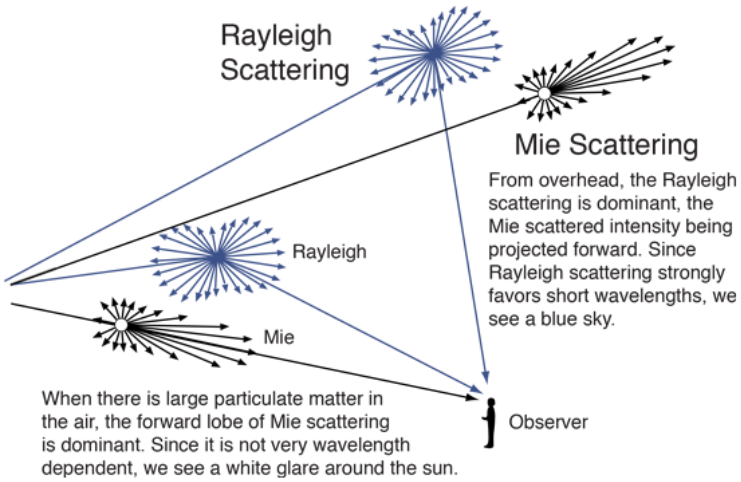

Figure 10: Rayleigh vs Mie scattering from [7]

Another difference between Rayleigh and Mie scattering is *how* the scattering happens. While Rayleigh has a much more uniformly distributed scattering, Mie is more prone to forward scattering. This means that the incident light will be scattered in a forward direction much more than in in backward or perpendicular direction [7].

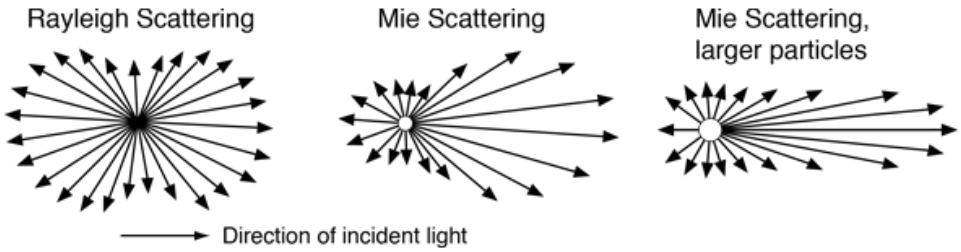

Figure 11: Scattering difference between Rayleigh and Mie from [7], [16]

10.2.3 OZONE ABSORPTION

The ozone layer is a region within the atmosphere of the Earth, it contains a high concentration of ozone (O_3) and absorbs most of the Sun's ultraviolet radiation [17]. Chappuis absorption, which is the absorption of electromagnetic radiation by ozone, is noticeable in the ozone layer as it absorbs a small amount of sunlight in the visible spectrum [8]. The absorption occurs between wavelengths of $400 - 650\ nm$, with similar peaks at $575\ nm$ and $603\ nm$. The absorption effect is named after the discoverer of the effect, James Chappuis [8].

Furthermore, Chappuis absorption is significantly weaker than the ultraviolet absorption that happens by the ozone layer. Mainly being visible during early and late hours when the sunlight needs to travel through more atmosphere, Chappuis absorption also contributes to the blue colour of the sky together with Rayleigh scattering [8]. That is because ozone absorbs more red and green light, than it does blue light, meaning less blue light is absorbed so more can scatter [8].

10.3 ATMOSPHERIC SCATTERING IN COMPUTER GRAPHICS

10.3.1 EXISTING MODELS

There exist many implementations of atmospheric scattering in computer graphics, each with a varying level of complexity, different simplifications and assumptions, as well as varying optimizations. This section aims to provide a summary of these models and compare them, before diving into the mathematics behind atmospheric scattering.

In 1993, Nishita et al. proposed a single scattering atmospheric model [18]. Due to the limited hardware then, it was too difficult to compute single scattering without simplifications at interactive framerates. The paper specifically looked at a rendering algorithm to display the earth from within the atmosphere as well as outside of the atmosphere, from outer space. They assume that the Earth is perfectly spherical, that sunlight travels parallel due to the distance of the Sun, and ignore the effects of the ozone layer. Furthermore, they also propose the usage of a precalculated scattering LUT. This model has very often been used as a base for atmospheric scattering research in computer graphics.

Based on Nishita et al.'s model from 1993 [18], O'Neil explains how to implement that model in GPU Gems 2 chapter 16 [19]. The main difference between the two models is that O'Neil doesn't use the precalculated LUT that Nishita et al. proposed. Instead, to run the model at interactive framerates, O'Neil analyses the results of the scattering equations and proposes approximate functions. The drawback in his model being that Rayleigh and Mie scattering aren't separated, and the scale height cannot be changed at will, without changing one of the approximate functions O'Neil uses.

Hoffman and Preetham proposed another model, focussing on an approach tailored for games, where efficiency is above physical accuracy. Using analytical approximations for Rayleigh and Mie scattering to achieve convincing results at low cost [20].

In the paper published by Bruneton and Neyret [21], another model is proposed that takes multiple scattering into account. To make sure the model runs at interactive frame rates, they precompute equations and store them in multidimensional LUTs. Furthermore, their precomputations also include shadows which allows for effects such as mountain light shafts.

The model in Unreal Engine and described by Hillaire [22] is another multi scattering approach that uses LUTs to their advantage. Since the model is used in Unreal Engine, it is highly important for it to be efficient and keep up with the visual quality of Unreal Engine.

10.4 MATHEMATICAL MODEL

Let's look at a ray going through the atmosphere:

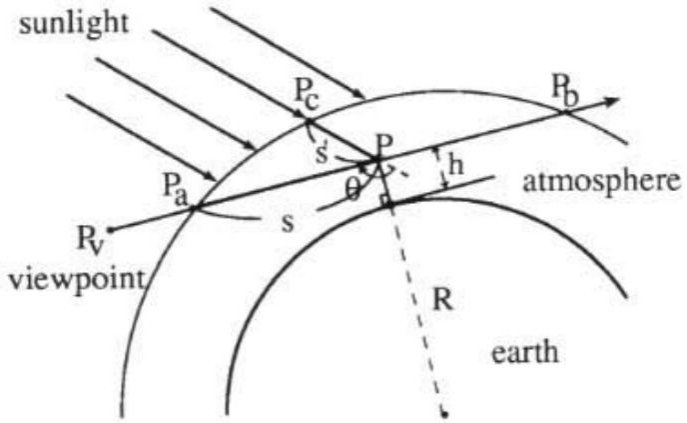

Figure 12: A view ray passing through the atmosphere from [18]

From viewpoint P_b to point P_v we have a view ray $\vec{v}$. Point P_a is the point where the view ray enters the atmosphere, with P_b the exit point. Point P is any point along the view ray $\vec{v}$ and P_c is the point where the light ray enters the atmosphere towards point P.

10.4.1 SCATTERING

If we want to know how much light has undergone scattering at a given point P towards the direction of the view point P_v, we can calculate that with the following equation [23]:

$$I(h,\theta,\lambda) = I_0 * \rho_R(h) * F_R(\theta) * \frac{\beta_R(\lambda)}{4\pi} + I_0 * \rho_M(h) * F_M(\theta) * \frac{\beta_M(\lambda)}{4\pi})$$

Equation 2: Scattering Equation in *P*

Where h is the altitude of point P, θ is the angle between the incident light and the direction towards the viewpoint, and λ the wavelength. The meaning of $\rho_{R,M}$, $F_{R,M}$, and $\beta_{R,M}$ will be explained in subsequent sections.

10.4.1.1 PHASE FUNCTIONS

The phase functions compute how much of the incident light is scattered towards the viewpoint. Rayleigh and Mie scattering each have their own phase function [19].

10.4.1.1.1 RAYLEIGH PHASE FUNCTION

The Rayleigh phase function can be described as follows:

$$\boldsymbol{F_R(\theta) = \frac{3}{4}(1 + cos^2(\theta))}$$

Equation 3: Rayleigh Phase Function

Where θ is the angle between the incident light and the direction towards the viewpoint [18], [24]. This is the angle between PP_v and PP_c in Figure 12.

This phase functions describes an almost uniformly distributed scattering as shown in this image:

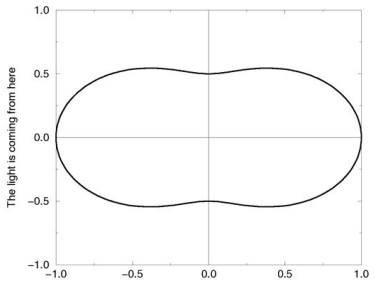

Figure 13: Rayleigh phase function distribution from [16]

10.4.1.1.2 MIE PHASE FUNCTION

The Mie phase function is more complex than the Rayleigh one. As mentioned before, Mie scattering happens much less uniformly than Rayleigh scattering, and instead the distribution is more in a forward lobe direction. The size of the particle also plays a significant role in the distribution.

10.4.1.1.2.1 HENYEY-GREENSTEIN

In a paper published by Henyey and Greenstein, a formulation for Mie scattering was proposed:

$$F_{HG}(\theta, g) = \frac{1 - g^2}{(1 + g^2 - 2 * g * \cos(\theta))^{3/2}}$$

Equation 4: Henyey-Greenstein Mie Phase Function

Where θ is the angle between the incident light and the direction towards the viewpoint, and g a parameter that measures the asymmetry of the scattering [24], [25]. This means that for $g > 0$ we have stronger forward scattering, while if $g < 0$ we have a stronger backward scattering. When $g = 0$ we end up with an isotropic distribution [25].

As pointed out by Cornette and Shanks, while the Henyey-Greenstein phase function can represent a forward or backward scattering peak, it can not simplify to the Rayleigh phase function from Equation 3.

10.4.1.1.2.2 DOUBLE HENYEY-GREENSTEIN

The Double Henyey-Greenstein phase function is a four-parameter phase function that applies the Henyey-Greenstein function twice:

$$F_{DHG}(\theta, f, g_1, g_2) = (1 - f) * F_{HG}(\theta, g_1) + f * F_{HG}(\theta, g_2)$$

Equation 5: Double Henyey-Greenstein Mie Phase Function

Where θ is the angle between the incident light and the direction towards the viewpoint, f a parameter that influences the strength of the forward and backward scattering peak, g_1 must be positive for the forward scattering peak, and g_2 must be negative for the backward scattering peak.

The downside of this phase function is that it is parameterized by four parameters. In order for the Double Henyey-Greenstein phase function to approximate the Rayleigh phase function from Equation 3, the forward and backward scattering peaks must be equal, meaning $g_1 = -g_2$, and $f = 1/2$ [24].

10.4.1.1.2.3 CORNETTE-SHANKS

Cornette and Shanks proposed a new phase function in 1992. Their proposed phase function is more physically realistic and overcomes the aforementioned problems with the Henyey-Greenstein and Double Henyey-Greenstein phase functions [24]:

$$F_{CS}(\theta, g) = \frac{3}{2}\frac{1-g^2}{2+g^2}\frac{1+cos^2(\theta)}{(1+g^2-2*g*\cos(\theta))^{3/2}}$$

Equation 6: Cornette-Shanks Mie Phase Function

Where θ is the angle between the incident light and the direction towards the viewpoint, and g a parameter that measures the asymmetry of the scattering [24]. This phase function has the interesting property that as g approaches 0 the phase function approaches the Rayleigh phase function. While as g approaches 1, it approaches the Henyey-Greenstein phase function [19], [24].

10.4.1.2 DENSITY FUNCTION

The density function scales the amount of scattered light depending on the altitude of point P. This needs to be done since the density of the atmosphere isn't constant. We assume that it is exponential and define the density function as follows [16], [18], [23]:

$$\rho(h) = \exp\left(\frac{-h}{H_0}\right)$$

Equation 7: Density ratio function

Where h is the altitude, and H_0 the scale height, which is the thickness of the atmosphere if the density were uniformly distributed [18], [23]. In the case of Rayleigh scattering where we focus on air molecules, the scale height is $H_0 = 7994\ m$ [16], [18]. For Mie scattering where the focus lies on aerosols, the scale height is $H_0 = 1200\ m$ [18], [23].

10.4.1.3 SCATTERING COEFFICIENT

Together with the phase function, the scattering coefficient describes how much light gets scattered in a certain direction. As mentioned before, for Rayleigh scattering this highly depends on the wavelength, while for Mie scattering it doesn't that much. That's why for the scattering coefficient for Mie scattering, the wavelength dependency is often left out.

For Rayleigh scattering this coefficient looks as follows [18], [23]:

$$\beta_R = 8\pi^3 \frac{(n^2-1)^2}{3N_s\lambda^4}$$

Equation 8: Rayleigh scattering coefficient.

Where n is the index of refraction of air (1.0003 [23]), N_s is the molecular density of the standard atmosphere at sea level ($2.545e25$ $\frac{molecules}{m^3}$ [23]), and λ the wavelength.

For Mie scattering this coefficient is more complex to calculate according to Bodare and Sandberg [23]. As they state, most models base it on measured data or approximate it. But the default value they use, as well as the implementation from Bruneton and Neyret [21], is $\beta_M = (2e-6, 2e-6, 2e-6)$.

10.4.2 ATTENUATION AND TRANSMITTANCE (OUT-SCATTERING)

When light travels through the atmosphere, a part of it also gets out-scattered or absorbed. The attenuation or transmittance function for light travelling from point P_a to P_b through the atmosphere is as follows [18], [19], [23]:

$$t(P_a, P_b, \lambda) = \left(\beta_R^e * \int_{P_a}^{P_b} \rho_R\big(h(P)\big)\, dP\right) + \left(\beta_M^e * \int_{P_a}^{P_b} \rho_M\big(h(P)\big)\, dP\right)$$

Equation 9: Attenuation/Transmittance equation

Where $h(p)$ is a function that calculates the altitude of the integration variable point P, and β^e is the extinction coefficient. We can say that $\beta_R^e = \beta_R$ since the particles that cause Rayleigh scattering, such as molecules, only out-scatter the light [23]. For β_M^e we can assume $\beta_M^e = \frac{\beta_M}{0.9}$ as shown by Bodare and Sandberg [23]. We also call this integration the optical depth [18], [19].

In the final scattering equation, we will need to use Equation 9 twice, since the light that will eventually reach viewpoint P_v will have been attenuated twice. Once, the incident light from PP_c will be attenuated from the distance it travels through the atmosphere to reach P, and then a second time when the light scatters and needs to travel through the atmosphere from point P to P_a.

10.4.3 FINAL SCATTERING EQUATION

To evaluate how much light reaches the viewpoint P_v after a single scattering event and attenuation, we can use the following equation [16], [18], [19], [23]:

$$I_v(\theta, \lambda) = I_0 * \frac{\beta(\lambda)}{4\pi} * F(\theta) * \int_{P_a}^{P_b} \rho(h(P)) * \exp\big(-t(P_c, P, \lambda) - t(P, P_a, \lambda)\big)\, dP$$

Equation 10: Atmospheric Scattering Equation

Where θ is the scattering angle, λ the wavelength, β the scattering coefficient, F the phase function, ρ the density function, and t the attenuation function.

From this we can conclude that the total amount of light reaching viewpoint P_v from both Rayleigh and Mie scattering is:

$$I_{total}(\theta,\lambda) = I_{v,R}(\theta,\lambda) + I_{v,M}(\theta,\lambda)$$

Equation 11: Total scattered light reaching viewpoint P_v

Which just means that the total scattered light reaching viewpoint P_v is the sum of the Rayleigh and Mie scattering equations.

10.4.4 OZONE ABSORPTION

The final thing that hasn't been discussed yet in this mathematical model is the inclusion of ozone absorption in the atmosphere. Since ozone doesn't scatter light and only absorbs [8], [17], [23], it only has to be accounted for in the attenuation function from Equation 9.

$$t(P_a,P_b,\lambda) = \left(\beta_R^e * \int_{P_a}^{P_b} \rho_R\big(h(P)\big)\,dP\right) + \left(\beta_M^e * \int_{P_a}^{P_b} \rho_M\big(h(P)\big)\,dP\right)$$

Equation 9: Attenuation/Transmittance equation

This function has already been discussed for Rayleigh and Mie, as well as looking at their extinction coefficients and respective scale heights. For ozone we can assume that the density function $\rho_O = 6e-7 * \rho_R$ [23]. This equation for ρ_O approximates the density by using the average concentration of ozone relative to air, which is 0.00006% [23], [26], [27]. It's an approximation since ozone in the atmosphere does not decrease exponentially. When it comes to getting the extinction coefficient for ozone β_O^e, Kutz pointed out how we can extract it from the ozone absorption cross sections shown in Figure 14 [27].

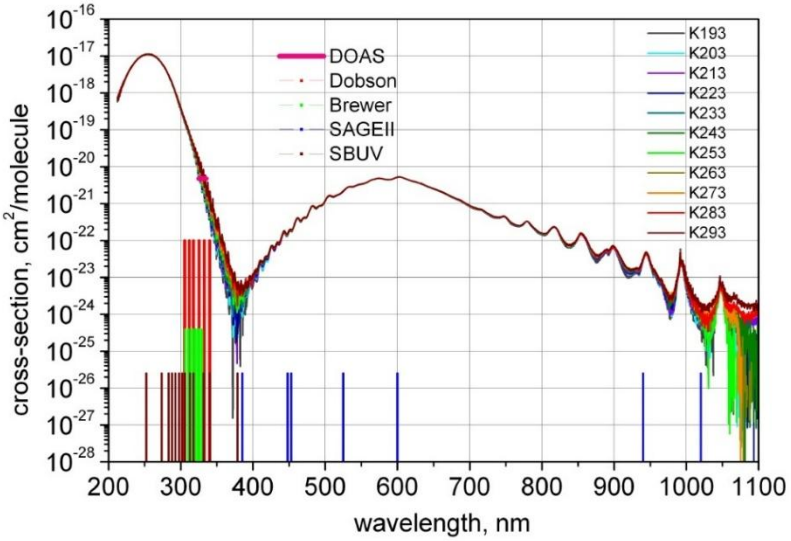

Figure 14: Ozone absorption cross-sections from [27]

This means that the equation from Equation 9 now looks like this if we were to include ozone absorption:

$$t(P_a, P_b, \lambda) = \left(\beta_R^e * \int_{P_a}^{P_b} \rho_R\big(h(P)\big)\, dP\right) + \left(\beta_M^e * \int_{P_a}^{P_b} \rho_M\big(h(P)\big)\, dP\right) + \left(\beta_O^e * \int_{P_a}^{P_b} \rho_O\big(h(P)\big)\, dP\right)$$

Equation 12: Attenuation/Transmittance equation with ozone absorption

11 RESEARCH

This research aims to implement an atmospheric single scattering model for real-time rendering. Which will be extended with the inclusion of ozone absorption as well as the use of different phase functions. The research will be done in a custom code base, tailored for this research, using the Vulkan graphics API (1.3.296.0) and C++ as the programming language. The source code is available here (https://github.com/Kobazaaa/Ashen).

The following sections will go over the base implementation, as well as the inclusion of ozone absorption and different phase functions. Afterwards looking at the performance differences by profiling different setups and finishing off by conducting a small survey for perceived differences in sky colour.

11.1 DESIGN DECISIONS AND DEVIATIONS FROM PRIOR WORK

As discussed in section 10.3.1, many existing models use precomputed LUTs for calculations such as optical depth. This is primarily done to reduce the computational cost to run at real-time interactive framerates. However, in this implementation, LUTs are omitted, as they primarily serve to avoid computing the double integral along the view and light rays.

O'Neil similarly avoided LUTs but relied on approximate functions that are hard-coded to a specific scale height, preventing the separation of Rayleigh and Mie scattering [19]. For those reasons, and for simplicity, this research will not use LUTs, or the approximate functions proposed by O'Neil. Instead, the double integral is computed directly in the shader, while still running at interactive framerates.

11.2 ASSUMPTIONS AND SIMPLIFICATIONS

This section serves to provide the assumptions and simplifications that are made for the model:

1. *Single scattering* - Multiple scattering is ignored to reduce the computational cost.
2. *Parallel sunlight* - It is assumed sunlight travels in a straight line and parallel to each other due to the distance of the Sun.
3. *Independent scattering components* - Rayleigh and Mie scattering are treated independently, each using their own scale heights, for accuracy.
4. *Spherical Earth* - The Earth is assumed to be a perfect sphere to simplify geometric computations.
5. *Exponential Density* - The density of the atmosphere varies exponentially with altitude.
6. *RGB approximation* - Scattering is computed per colour channel rather than full spectral wavelengths, to reduce complexity.
7. *Neglecting ground effects* - There is no interaction with terrain in the scattering equation.
8. *Sky conditions* - The experiments take place on a perfect spherical world with a perfectly clear sky, and fixed sun elevations.

11.3 IMPLEMENTATION DETAILS

11.3.1 RENDERING PIPELINE INTEGRATION

Scattering computations are performed per-vertex of the sky dome in the vertex shader, except for phase functions, which are applied in the fragment shader. This separation prevents visual artifacts that occur when phase functions are evaluated per vertex. Other parts of the scattering equation were chosen to be evaluated per vertex, and then be interpolated, to reduce computational load while still providing smooth results across the sky dome.

The vertex shader ("*SkyFromAtmosphere.vert*") receives atmospheric and camera parameters through a uniform buffer. These include the camera position and height, light direction, Rayleigh, Mie, and ozone coefficients, scale heights, sun intensity, and rendering radius and thickness. See Code Block 1 for the full descriptor that the shader expects.

```
layout(set = 0, binding = 0) uniform Parameters
{
    vec3 cameraPos;                // current camera pos
    float cameraHeight;            // current camera height

    vec3 lightDir;                 // direction to sunlight
    float sampleCount;             // nr of samples along the ray

    vec3 betaR;                    // Rayleigh coefficient
    float atmosphereThickness;     // thickness of the atmosphere

    vec3 betaM;                    // Mie coefficient
    float planetRadius;            // planetary radius

    vec3 betaO;                    // ozone coefficient
    float rayleighScaleHeight;     // scale height Rayleigh (thickness if atmosphere uniform)
    float mieScaleHeigh;           // scale height Mie (thickness if atmosphere uniform)
    float sunIntensity;            // intensity of the sun

    float renderRadius;            // the radius of the planet at which it is rendered
    float renderThickness;         // the thickness of the atmosphere at which it is rendered
};
```

Code Block 1: Vertex shader input descriptor set for sky shader

After numerical integration, the vertex shader outputs three values to the fragment shader ("*SkyFromAtmosphere.frag*"): the accumulated Rayleigh colour, the accumulated Mie colour, and the direction from the current vertex to the camera. These outputs are used in the fragment shader to apply the phase functions as described in section 10.4.1.1.

```
layout(location = 0) out vec3 outRayleighColor;
layout(location = 1) out vec3 outMieColor;
layout(location = 2) out vec3 outDirectionToCam;
```

Code Block 2: Vertex shader output

Lastly, there is a post-process shader to allow for HDR rendering and scale down the resulting image back to LDR using an exposure function. This was done since the atmospheric scattering equation easily generates values that are bigger than one. The exposure function used is the one proposed by O'Neil [19]:

$$E(x, c) = 1 - \exp(-x * c)$$

Equation 13: Exposure correction function

Where x is the exposure value, and c the HDR colour output. You can see here a comparison of HDR vs no HDR output of the final render, with a default exposure value of 2 [19].

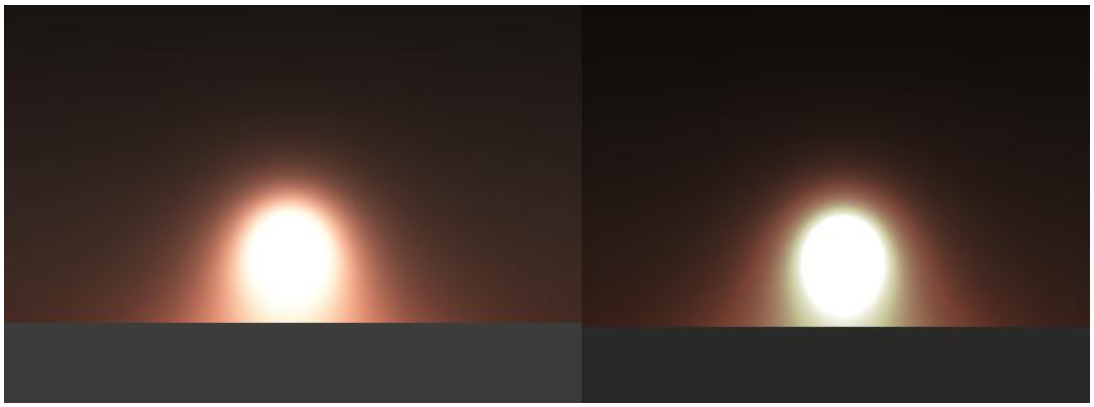

Figure 15: Comparison between HDR and no HDR

The implementation is quite simple. When HDR is enabled, the output image is rendered to a separate render texture with an `R32G32B32A32_SFLOAT` format. When it is done rendering, a full screen triangle gets drawn and the texture gets sampled per fragment in the fragment shader and Equation 13 gets applied to this HDR value for it to be converted to LDR and presented to the swap chain.

11.3.2 NUMERICAL INTEGRATION

Unfortunately, the scattering equations from Equation 9 and Equation 10 are impossible to solve analytically, due to it being an integral equation. Instead, a different approach needs to be taken.

Recalling Equation 10, we need a starting point (P_a), an exit point out of the atmosphere (P_b), and a point where the incoming light enters the atmosphere (P_c) toward a given point P. These are all computable, the start position is the position of the camera if we are inside the atmosphere, the exit point can be computed by doing a ray-sphere intersection test since we have a direction from the camera and a sphere from the sky dome. The same can be done for the light entry point. With the required parts at hand, we can look at solving the equation.

We can solve the equations numerically by applying a Riemann sum. A Riemann sum is an approximation of an integral by performing a finite sum, and is often used in numerical integration [28]. One such way to solve a Riemann sum is the midpoint rule. The definition which is defined as [28]:

$$S_{mid} = \Delta x \left[f\left(a + \frac{\Delta x}{2}\right) + f\left(a + \frac{3\Delta x}{2}\right) + \cdots + f(b - \frac{\Delta x}{2}) \right]$$

Equation 14: Midpoint rule Riemann sum

Where a is the starting value of the integral, b the end value, and Δx the interval at which you evaluate the function f. This means that the smaller the interval is, the more accurate the approximation will be. If we were to visualize the midpoint rule on the graph itself, it would look like this:

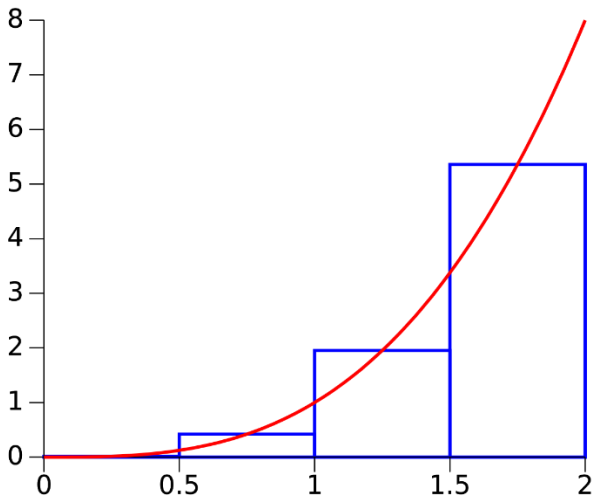

Figure 16: Midpoint rule graph visualization from [28]

In Equation 10's outer integral, we need to integrate from P_a to P_b. The length of this ray is known, and a constant sample count is passed via a uniform buffer as shown in Code Block 1. Using these two values we can compute an interval. In code this looks as follows:

```
1.      // Get the ray from the Camera to the current Vertex,
2.      // the length of this ray is the far point of the ray passing through the atmosphere
3.      vec3 startPos = cameraPos;
4.      vec3 endPos = vertexPosition;
5.      vec3 ray = endPos - startPos;
6.      float farDistance = length(ray);
7.      ray /= farDistance;
8.
9.      // Initialize the scattering loop variables
10.     float travelDistance = farDistance;
11.     float sampleLength = travelDistance / sampleCount;
12.     vec3 sampleRay = ray * sampleLength;
13.     vec3 samplePoint = startPos + sampleRay * 0.5;
```

Code Block 3: Midpoint rule in code

The integration proceeds by iterating from the initial sample point to the endpoint, advancing the ray by the interval "`sampleLength`" at each step. Within this loop, the integral from Equation 10 is approximated using the midpoint rule by accumulating the Rayleigh and Mie contributions, each weighted by the "`sampleLength`".

```
1.      // Loop through the sample points
2.      vec3 frontColorR = vec3(0);
3.      vec3 frontColorM = vec3(0);
4.      for(int i = 0; i < sampleCount; ++i)
5.      {
6.          // Calculate the sample depth for rayleigh and mie
7.          float sampleHeightOffGround = length(samplePoint) - planetRadius;
8.
9.          float sampleDepthR = DensityFunction(sampleHeightOffGround, rayleighScaleHeight);
```

```
10.         float sampleDepthM = DensityFunction(sampleHeightOffGround, mieScaleHeigh);
11.
12.         // Calculate the attenuation
13.         vec3 exitPos = samplePoint + lightDir * DistanceToAtmosphereExit(samplePoint, lightDir);
14.         vec3 attenuation = CalculateAttenuation(startPos, samplePoint);
15.         attenuation *= CalculateAttenuation(samplePoint, exitPos);
16.
17.         // Add Color
18.         frontColorR += sampleDepthR * attenuation * sampleLength;
19.         frontColorM += sampleDepthM * attenuation * sampleLength;
20.
21.         // Advance to next sample
22.         samplePoint += sampleRay;
23.     }
```

Code Block 4: Solving the outer integral of the scattering equation

After this numerical integration, the result needs to be multiplied by the intensity of the sunlight and the scattering coefficients. Phase function multiplication is delayed to the fragment shader as stated before.

```
1.      outRayleighColor = sunIntensity * betaR * INV_FOUR_PI * frontColorR;
2.      outMieColor = sunIntensity * betaM * INV_FOUR_PI * frontColorM;
```

Code Block 5: Final Rayleigh and Mie colour from vertex shader

Equation 10 also contains the attenuation function t (Equation 9) inside its integral, this is the function "`CalculateAttenuation`" in Code Block 4. This function itself also contains an integral that needs to be solved numerically, twice even. Once from sample point P to start point P_a and once from entry point P_c to the sample point P. This is similar to what was done in Code Block 3 and Code Block 4. The main difference being what is computed inside of the for-loop. The entire shader can be found at the end of this paper in the Appendices section.

This leaves us with finding the value for β_R, β_M, β_R^e, and β_M^e for the attenuation function.

Calculating β_R is given by Equation 8, with $n = 1.0003$, $N_s = 2.545e25\ \frac{molecules}{m^3}$, and $\lambda = (6.5e-7, 5.1e-7, 4.75e-7)$ for RGB, the computed value for β_R is $\beta_R = (6.56\text{e}-6, 1.73\text{e}-5, 2.30\text{e}-5)$.
Additionally, $\beta_R^e = \beta_R$ [23].

Since β_M is more complex to calculate, we can base ourselves off of the value used by Bodare and Sandberg's model $\beta_M = (2e-6, 2e-6, 2e-6)$ [21], [23]. Furthermore $\beta_M^e = \frac{\beta_M}{0.9}$.

11.4 PHASE FUNCTIONS

11.4.1 PHASE FUNCTION IMPLEMENTATION

As mentioned in section 11.3.1, the phase functions are delayed until the fragment shader to prevent artifacts. This fragment shader ("*SkyFromAtmosphere.frag*") receives the accumulated Rayleigh and Mie colours, as well as the direction vector from the vertex shader. Furthermore, it also receives a uniform buffer with the following elements:

```
1. layout(set = 0, binding = 1) uniform PhaseParameters
2. {
3.     vec3 lightDir;                  // direction of the sunlight
4.     float g;                        // constant that affects symmetry of the scattering
5.     float g2;                       // g^2
6.     uint phaseType;                 // Which phase function to use
7. };
```

Code Block 6: Fragment shader input descriptor set for sky shader

The "phaseType" parameter is there to enable switching phase functions at runtime. However, for profiling this parameter is removed to avoid any if-statement inside of the shader. The phase functions this paper will look at are the same as described in section 10.4.1.1, being the Henyey-Greenstein, Cornette-Shanks, and Double Henyey-Greenstein phase functions.

Specifically, the fragment shader computes the final equation, Equation 11, right after multiplying the incoming Rayleigh and Mie colours with their respective phase functions. Refer to Appendices to see the full fragment shader.

11.5 OZONE ABSORPTION

11.5.1 OZONE MODEL

As seen in section 10.2.3, ozone absorption only needs to be accounted for in the attenuation function from Equation 9, turning that equation into Equation 12:

$$t(P_a, P_b, \lambda) = \left(\beta_R^e * \int_{P_a}^{P_b} \rho_R\big(h(P)\big)\, dP\right) + \left(\beta_M^e * \int_{P_a}^{P_b} \rho_M\big(h(P)\big)\, dP\right) + \left(\beta_O^e * \int_{P_a}^{P_b} \rho_O\big(h(P)\big)\, dP\right)$$

Equation 12: Attenuation/Transmittance equation with ozone absorption

The way these integrals are solved numerically, has already been explained in section 11.3.2. The Rayleigh and Mie extinction coefficients were already described in section 10.4.1.3 and 11.3.2. This means that the only things left to complete this equation are β_O^e and ρ_O. But as seen in section 10.4.4, both these values can be directly computed.

Starting off with ρ_O, this value can be approximated with $\rho_O = 6e-7 * \rho_R$. While ozone doesn't drop off exponentially, as shown in the Literature Study / Theoretical Framework, it can be approximated by multiplying the Rayleigh density function with the average density of ozone relative to air, which is 0.00006% [23], [26], [27].

The value for β_O^e can be calculated as pointed out by Peter Kutz [27]. The absorption cross-sections for ozone are obtained from the figure below. The corresponding values for the wavelengths $\lambda = (6.5e-7, 5.1e-7, 4.75e-7)$ are $cross\_section = (3.1e-21, 1.9e-21, 4.5e-22)$. Note that the cross-sections from the graph are in $\frac{cm^2}{molecule}$, but need to be converted to $\frac{m^2}{molecule}$. This can be achieved by multiplying the cross-sections by 10^{-4}. Doing this conversion, we end up with $cross\_section = (3.1e-25, 1.9e-25, 4.5e-26)$.

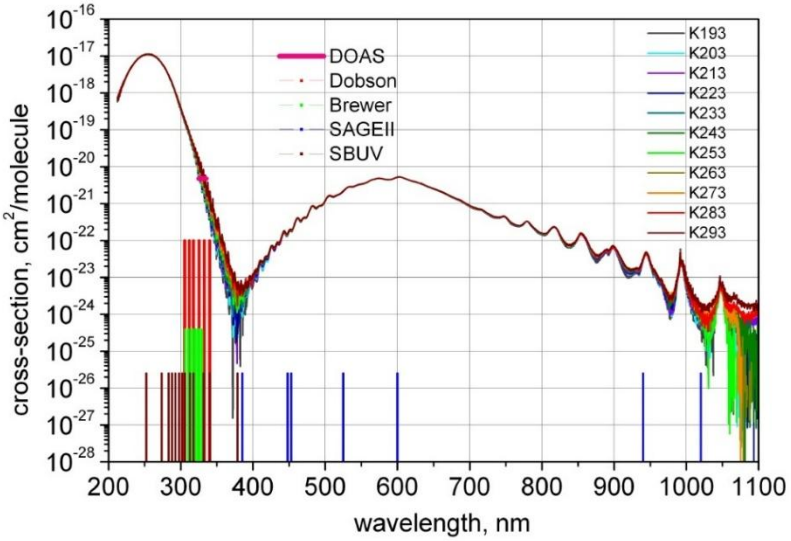

Figure 14: Ozone absorption cross-sections from [27]

Multiplying these with the molecular number density of standard air ($N_S = 2.545e25\ \frac{molecules}{m^3}$ [23]), ending up with a value for the ozone extinction coefficient $\beta_O^e = (7.89, 4.84, 1.15)$.

11.5.2 INTEGRATION INTO THE SCATTERING MODEL

Ozone absorption is incorporated into the scattering model in the vertex shader for the sky dome ("*SkyFromAtmosphere.vert*"). When computing the inner integrals for the attenuation functions, we accumulate the attenuation due to ozone and multiply the result by the ozone extinction coefficient, before adding all the attenuation factors together, as shown in Equation 12.

```
1.      for(int i = 0; i < sampleCount; ++i)
2.      {
3.          float height = length(samplePoint) - planetRadius;
4.
5.          // Calculate density at height
6.          float dR = DensityFunction(height, rayleighScaleHeight);
7.          float dM = DensityFunction(height, mieScaleHeigh);
8.          float dO = dR * 6e-7;
9.
10.         // sum up
11.         resultRayleigh  += dR * sampleLength;
12.         resultMie       += dM * sampleLength;
13.         resultOzone     += dO * sampleLength;
14.
15.         samplePoint += sampleRay;
16.     }
17.
```

```
18.     // calibrate with coefficients
19.     resultRayleigh *= betaR;
20.     resultMie *= betaM / 0.9;
21.     resultOzone *= betaO;
```

Code Block 7: Ozone Absorption integration

11.6 EXPERIMENTAL SETUP

This section covers the setup of the experiments and how the experiments were conducted. The outcomes of these experiments are discussed in more detail in the Results and Discussion section.

11.6.1 DEFAULT PARAMETER VALUES

This section quickly goes over the default values for the constants described under the mentioned sections. Unless stated otherwise, these values are kept constant across all experiments.

Section 11.3:

Variable	Meaning	Value
sampleCount	The number of samples taken along the ray.	16
betaR	The scattering coefficient for Rayleigh.	(6.56e−6, 1.73e−5, 2.30e−5)
atmosphereThickness	The thickness of the atmosphere.	100.000 m
betaM	The scattering coefficient for Mie.	(2e−6f, 2e−6f, 2e−6f)
planetRadius	The radius of the planet.	6.371.000 m
betaO	The absorption coefficient for ozone.	(7.89, 4.84, 1.15)
rayleighScaleHeight	The scale height for Rayleigh.	7994 m
mieScaleHeight	The scale height for Mie.	1200 m
sunIntensity	The intensity of the Sun.	20
renderRadius	The radius at which the planet is rendered.	20
renderThickness	The thickness of the atmosphere at which is rendered.	0.3139

Table 2: Default values for scattering vertex shader

Section 11.4:

Variable	Meaning	Value
g	Constant that affects symmetry of the scattering.	-0.990
g2	g*g	0.9801
phaseType	The current phase function in use.	Henyey-Greenstein

Table 3: Default values for scattering fragment shader

11.6.2 VISUAL SURVEY METHODOLOGY

To measure perceived visual differences in the different render profiles, a small survey has been conducted. People were given two images, they were asked to compare the two images and say which one they think looks more realistic, and which one they prefer. None of the participants were aware which image had a certain feature or not, to avoid any bias.

Participants were asked to compare the following image setups, the participants did not see the name of the images:

Sun Elevation Angle	Image 1	Image 2
Low	No Ozone	No Ozone
Low	Cornette-Shanks	Double Henyey-Greenstein
Low	No Ozone	No Ozone
Low	Cornette-Shanks	Henyey-Greenstein
Low	Henyey-Greenstein	Double Henyey-Greenstein
Low	Cornette-Shanks	Double Henyey-Greenstein
Medium	No Ozone	No Ozone
Medium	Cornette-Shanks	Double Henyey-Greenstein
Medium	Cornette-Shanks	Henyey-Greenstein
Medium	Cornette-Shanks	Henyey-Greenstein
Medium	Henyey-Greenstein	Double Henyey-Greenstein
Medium	No Ozone	No Ozone
High	No Ozone	No Ozone
High	Cornette-Shanks	Double Henyey-Greenstein
High	Cornette-Shanks	Henyey-Greenstein
High	No Ozone	No Ozone
High	Henyey-Greenstein	Double Henyey-Greenstein
High	Henyey-Greenstein	Double Henyey-Greenstein

Table 4: Survey Image Comparison Profiles

You may notice that some comparisons were asked more than once, this was done to identify inconsistent or random responses and to increase the reliability of the collected data.

All the submitted forms were compiled into useable data and graph, which are present in the Appendices section, and whose results are discussed in the upcoming sections.

11.6.3 PERFORMANCE MEASUREMENT

11.6.3.1 CONFIGURATION

All the experiments were run on consistent hardware and software for fairness across tests.

Component	Specification
CPU	AMD Ryzen 7 5800H with Radeon Graphics
GPU	NVIDIA GeForce RTX 3060 Laptop GPU
OS	Windows 11 (24H1)
Vulkan API	1.3.296.0
Resolution	1920x1080

Table 5: List of hardware and software used for performance profiling

11.6.3.2 RENDERING PROFILES

The following rendering setups were profiled:

	Has Ozone Absorption	Phase Function	Sun Angle
Profile 01	Yes	Cornette-Shanks	Medium
Profile 02	No	Cornette-Shanks	Medium
Profile 03	No	Henyey-Greenstein	Medium
Profile 04	No	Double Henyey-Greenstein	Medium

Table 6: Performance Profiles

To avoid any bias, for the profiles that do not use ozone absorption, all the code related to ozone absorption was removed. The implemented model has a toggle to enable and disable ozone, and similarly to cycle between phase functions. However, for profiling this toggle will be removed to not have any if-statements in the shader. The same thing has been done for the phase functions. This is to ensure performance differences reflect actual computational cost and not branching behaviour inside the shaders.

Each profile renders two meshes, a ground mesh and a sky dome mesh. Each mesh consists of $63k$ vertices.

11.6.3.3 MEASURING

For the measuring of performance itself, a tool called "*FrameView*" [29] was used. Each render profile from Table 6 was measured over the same time period. To account for minor variations, each render profile was measured five times and the average was taken.

The results of these measurements can be found in the Appendices section and will be discussed in the upcoming sections.

11.7 RESULTS

11.7.1 VISUAL RESULTS

Here are all the different render profiles shown. All the individual images can be found at the end of this paper in the Appendices section.

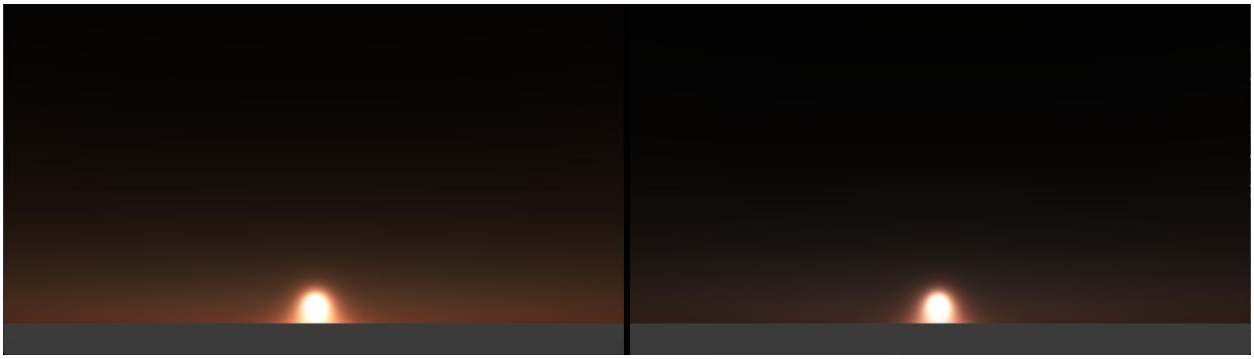

Figure 17: No Ozone vs Ozone | Low Sun Angle

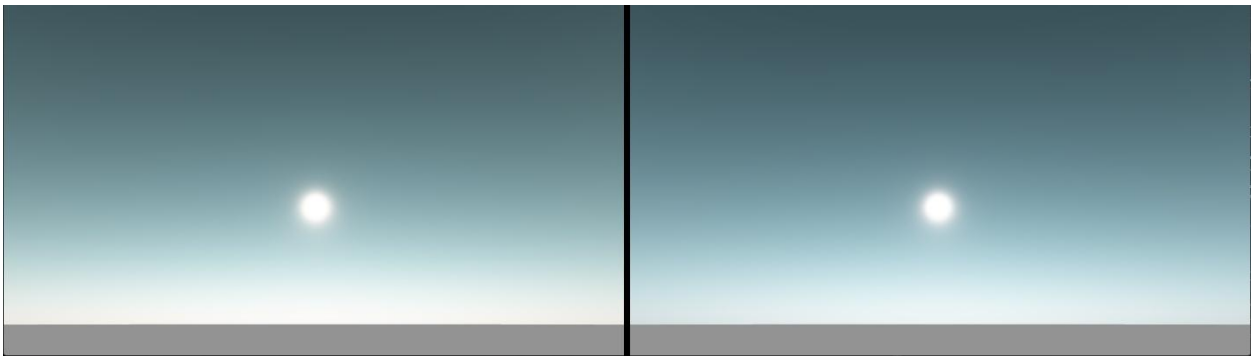

Figure 18: No Ozone vs Ozone | Medium Sun Angle

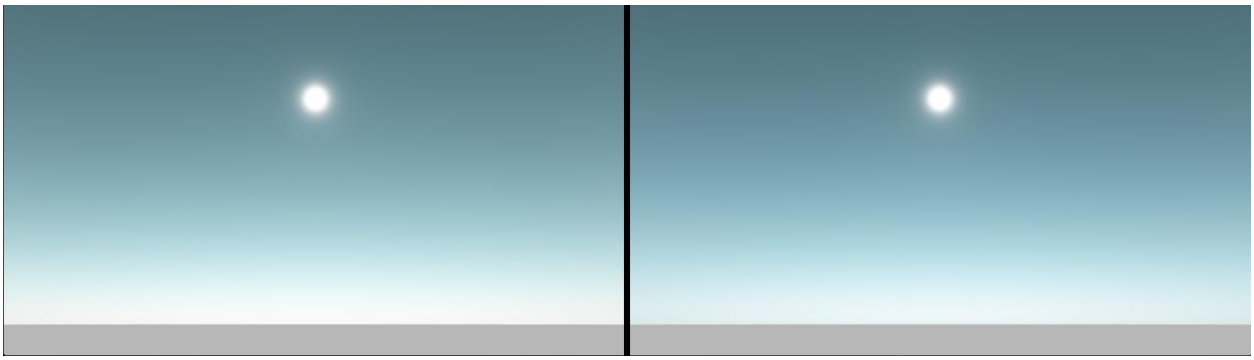

Figure 19: No Ozone vs Ozone | High Sun Angle

11.7.2 SURVEY RESULTS

A total of 43 participants filled out the survey. Due to the many graphs and results, all the graphs for the survey can be found in the Appendices section. The results will be further discussed in the Discussion and Conclusion section.

Shortly looking at the statistics for ozone absorption turned on or off, participants rated “No Ozone” more at lower sun angles in both perceived realism and preference, while at high sun angles “Ozone” was the preferred option. The participants of course did not know they were rating ozone absorption or which image was which.

For the phase function comparisons, when comparing Cornette-Shanks to Double Henyey-Greenstein, generally Cornette-Shanks was preferred. When comparing Cornette-Shanks to normal Henyey-Greenstein, a lot of people said that they could not see a difference between the two. Based on these results, we would assume that when comparing Henyey-Greenstein to Double Henyey-Greenstein, people would also preferer the normal one over the double one. When looking at the results for Henyey-Greenstein vs Double Henyey-Greenstein, this does indeed appear to be the case, with actually getting quite similar answers to Cornette-Shanks vs Double Henyey-Greenstein.

11.7.3 PROFILING RESULTS

11.7.3.1 AVERAGE FRAME TIME

The average frame time (in ms) of each profile from Table 6 was measured over the same period. This was done several times to account for variations and all the results were averaged. The results are as follows:

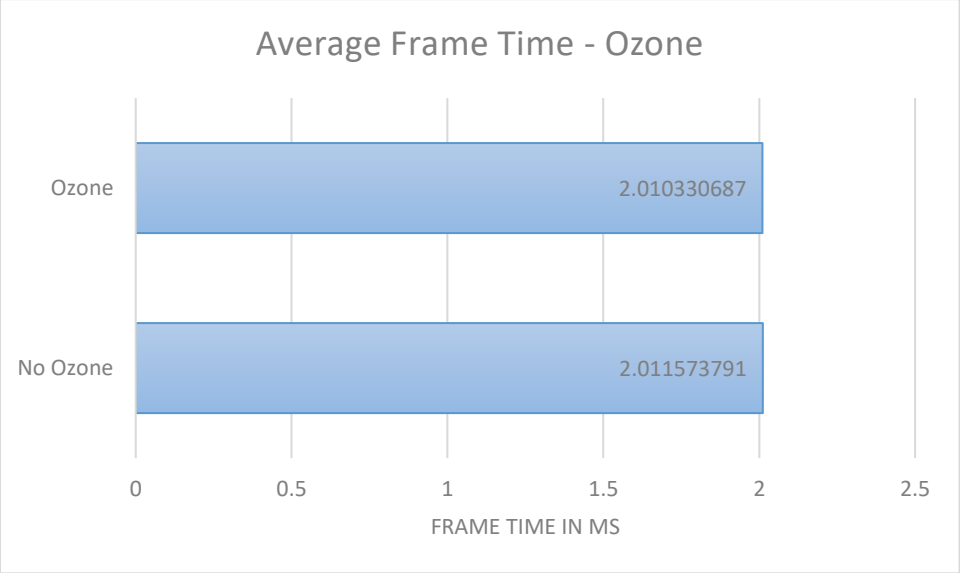

Figure 33: Average Frame Time | Ozone Absorption

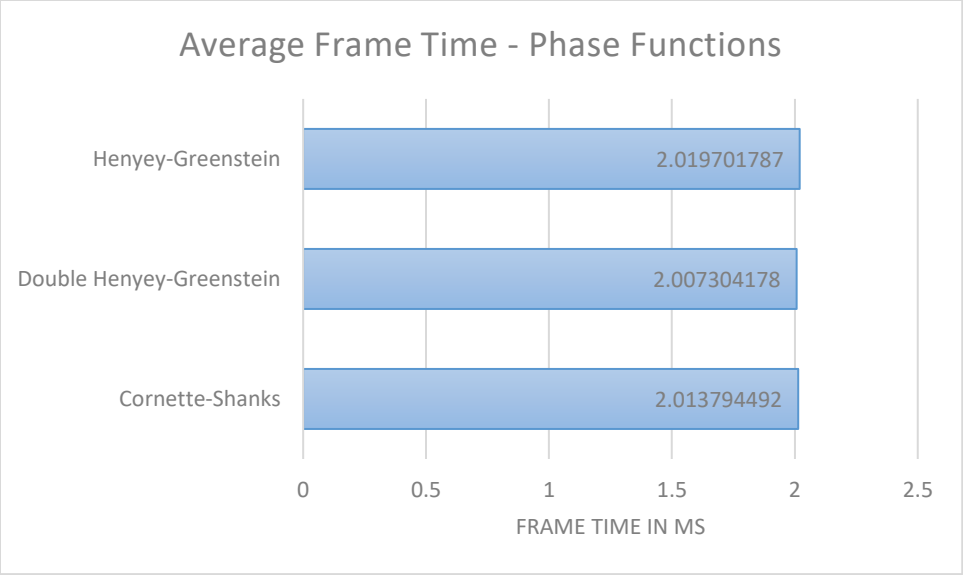

Figure 34: Average Frame Time | Phase Functions

There is no distinct difference between the different profiles. Neither the phase function, nor the inclusion of ozone absorption makes a significant enough difference. The minor differences observed are likely due to measurement noise or small variations in GPU/CPU scheduling rather than the choice of phase function or the inclusion of ozone absorption.

11.7.3.2 AVERAGE GPU USAGE

The average GPU usage of each profile from Table 6 was measured over the same period. This was done several times to account for variations and all the results were averaged. The results are as follows:

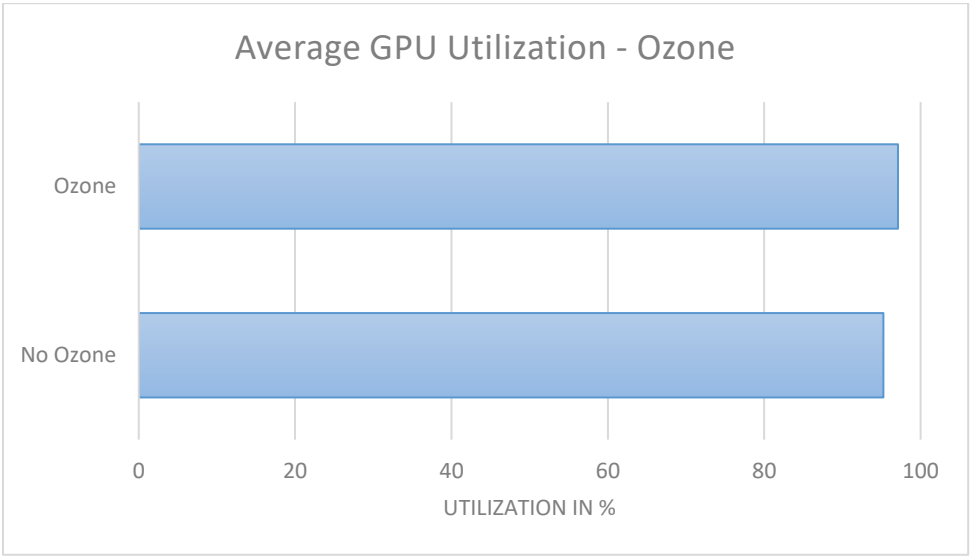

Figure 35: Average GPU Utilization | Ozone Absorption

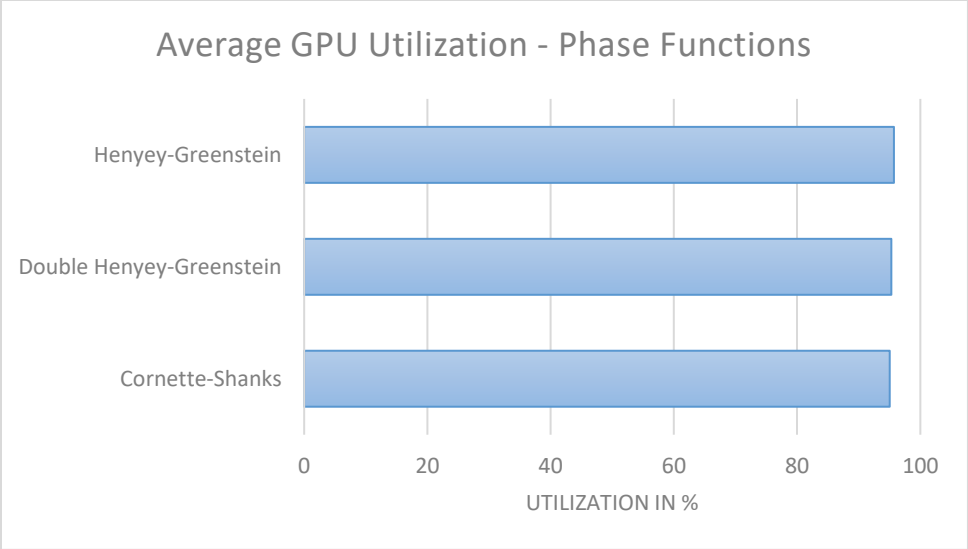

Figure 36: Average GPU Utilization | Phase Functions

There is no distinct difference between the different profiles. Neither the phase function, nor the inclusion of ozone absorption makes a significant enough difference. The minor differences observed are likely due to measurement noise or small variations in GPU/CPU scheduling rather than the choice of phase function or the inclusion of ozone absorption.

12 DISCUSSION

12.1 OZONE ABSORPTION

Looking at the performance measurements taken for including ozone absorption, visible in Figure 33 and Figure 35, there is no significant difference in the inclusion or exclusion of ozone absorption. If you are striving for realism, performance should thus not be a reason to not include the effects the ozone layer has on the colour of the sky. Whether it is worth it to include ozone absorption in a single scattering model is a separate question. Looking at the results from the survey in Figure 21, Figure 22, and Figure 23, it is safe to say that there is a clear distinction.

When the sun is at a lower angle, such as sunrise or -set, participants overwhelmingly preferred the image without ozone absorption, as well as thinking it was more realistic. The medium sun angle is more evenly distributed between “ozone” and “no ozone”, while the high sun angle shows that the participants preferred the image with ozone. These results may seem odd at first, but there is a logical explanation behind them.

In the low sun angle category, nearly all participants both thought the image without ozone absorption was more realistic and preferred it. If we look at these renders side by side:

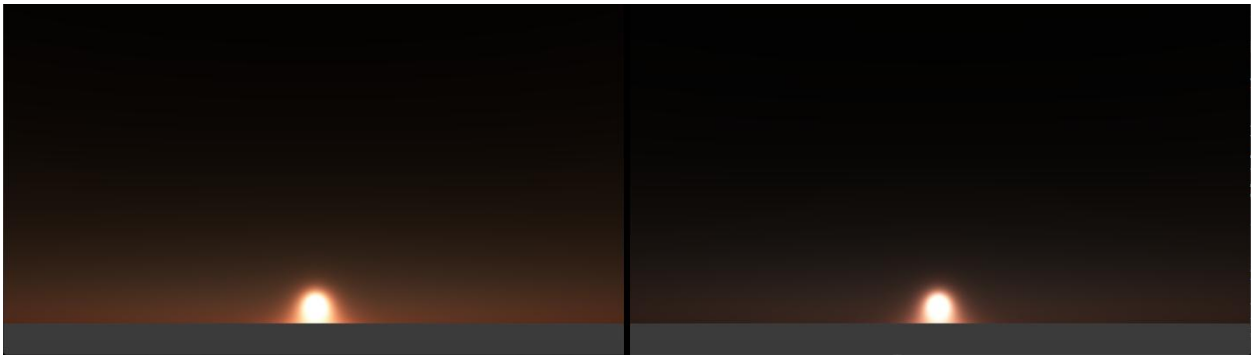

Figure 17: No Ozone vs Ozone | Low Sun Angle

Comparing these two images to a real-life photo of a sunset:

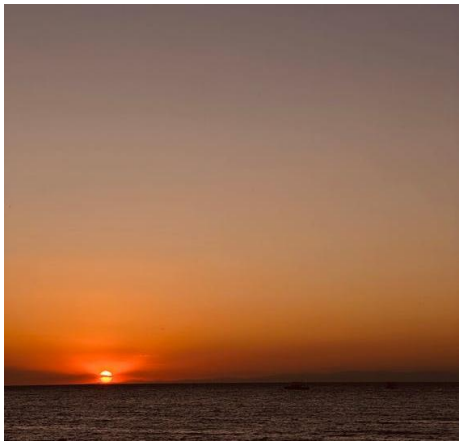

Figure 20: Sunset view from [30]

From the original render the leftmost image, the one without ozone, does resemble real-life sunsets more. The Chappuis absorption causes more red and green light to be absorbed than blue [8], meaning that we lose that red glow that the left image has in Figure 17, compared to the right image. But Figure 20 clearly shows that red glow is something real. The answer is multiple scattering. This implementation only considers single scattering, where every light ray undergoes one scattering event before reaching the viewpoint, this does not mimic reality. In reality each light ray undergoes several scattering events before reaching your eye, accumulating light and causing the sky to still appear slightly bright at sunset and sunrises [23], unlike this implementation where at sunset the top of the sky already looks very dark. Another key role is the conditions of the sky, this model assumed no special conditions, but in real life there might be more dust, mist, haze, or other atmospheric effects that change the colours you perceive in the sky, due to more or less scattering taking place [14].

When it comes to the medium or high sun angle categories, a significant more amount of people chose the image with ozone absorption to be more realistic and also preferred it. Comparing these side to side, we can see a very subtle difference in colour, where the image with ozone absorption appears ever so slightly bluer.

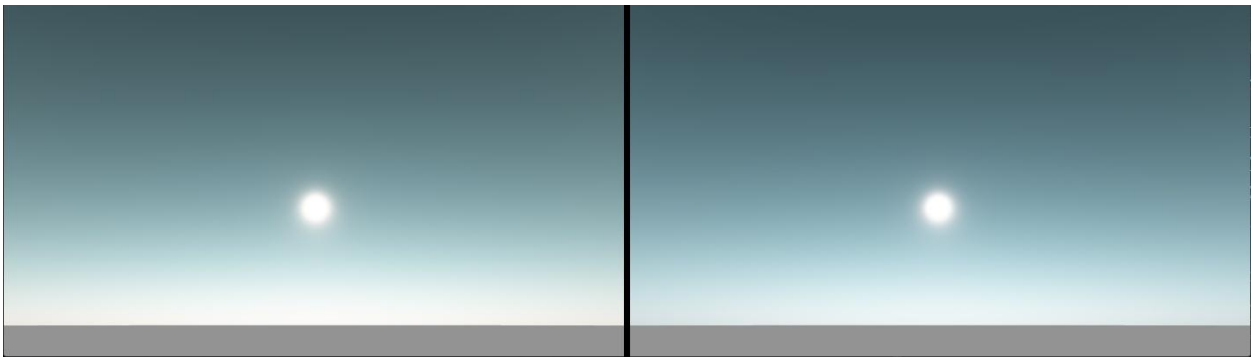

Figure 18: No Ozone vs Ozone | Medium Sun Angle

12.2 PHASE FUNCTIONS

The performance results for the different phase functions showed almost no difference whatsoever. The only minor differences are not significant enough and can be written off as noise or small variations in scheduling, rather than the phase function itself having a significant effect on it.

The survey results do look more interesting. Across all sun elevations, the *Cornette-Shanks* phase function was generally preferred over the *Double Henyey-Greenstein* function. When comparing *Cornette-Shanks* to *Henyey-Greenstein*, the majority of participants noted that they did not see any noticeable difference between the two. As expected based on these results, *Henyey-Greenstein* was also generally preferred and perceived as more realistic when compared to the *Double Henyey-Green*stein phase function.

These results do suggest that the visual differences between *Cornette-Shanks* and *Henyey-Greenstein* are minimal, which is reasonable since *Cornette-Shanks* is an enhanced version of *Henyey-Greenstein* [24]. The fact that *Double Henyey-Greenstein* did not perform as well, could be due to the choice of parameters, as *Double Henyey-Greenstein* expects three parameters (ignoring the angle), instead of the same one parameter that *Cornette-Shanks* and *Henyey-Greenstein* expect. You might get more favourable results when using different parameters for Equation 5.

13 CONCLUSION

“How does a single scattering atmospheric scattering model compare visually and computationally when extended with ozone absorption and alternative phase function approximations?”, that was the research question this paper sought to answer. To address this, a real-time single scattering model was implemented and evaluated for both performance and visual perception.

From a computational perspective, the results show that neither the inclusion of ozone absorption nor the choice of phase function has a measurable impact on performance. Across all profiled configurations, average frame times and GPU utilization remained effectively unchanged, indicating that these changes do not have a significant computational overhead in the context of single scattering.

Visually, the inclusion of ozone absorption was shown to influence perceived realism and user preference, but in a context dependent manner. Survey results indicate that at low sun elevations, images rendered without ozone absorption were more frequently preferred and perceived as more realistic. At medium sun elevations, preferences were more evenly distributed, while at high sun elevations, images including ozone absorption were clearly favoured. These findings suggest that ozone absorption contributes positively to perceived realism when the sun is high in the sky but may reduce realism at low sun angles within a single-scattering framework.

This behaviour can be attributed to the limitations of single scattering. Since effects such as multiple scattering are not captured by the implemented model, ozone absorption alone can overly attenuate red and green wavelengths, resulting in skies that appear darker and less realistic than expected, especially at low sun angles.

Overall, this research demonstrates that both ozone absorption and alternative phase functions can be incorporated into a single-scattering atmospheric model without performance penalties. However, their visual impact depends strongly on viewing conditions and model limitations. The decision to include ozone absorption in a single-scattering context should therefore be guided by the intended visual outcome and lighting conditions rather than computational constraints.

14 FUTURE WORK

14.1 MULTIPLE SCATTERING

A natural extension of this research would be modifying the model to use multiple scattering instead of single scattering. As discussed in the Discussion and Conclusion sections, perceived realism does suffer due to the limitations of single scattering, especially at low sun angles. Investigating the effects of ozone absorption in a multiple scattering setup would provide an even more realistic representation of the atmospheric effects. Although ozone absorption in multiple scattering has already been explored in other works [23], [27], there remains room to study the perceptual impact and computational cost in modern real-time pipelines.

14.2 OPTIMIZATIONS

For simplicity and clarity, this work deliberately omitted optimization techniques such as precomputed lookup tables and analytical approximations commonly used in atmospheric scattering models. Future work could explore how ozone absorption can be integrated into these optimized approaches, including for example transmittance LUTs. Evaluating the trade offs between performance and memory usage when incorporating ozone into these optimized models would be relevant for real-time rendering applications such as games and simulations.

14.3 OTHER ATMOSPHERIC EFFECTS

This paper did not cover other major atmospheric effects, such as clouds, haze, or weather conditions. All these effects have an impact on the atmosphere and can change the appearance of the sky. Looking into these effects, possibly in combination with ozone absorption, could be interesting research. More broadly, extending the model to include additional atmospheric effects would further improve realism and provide a more complete representation of the sky.

15 CRITICAL REFLECTION

I feel like this research project taught me a lot of different things.

First and foremost, it has quite obviously taught me a lot about atmospheric scattering, both in physics and computer graphics. But also, about project planning and scope. I know of myself that I tend to over scope quite fast, this project started out the same, but ultimately, I feel like I scoped it down to a good and manageable level. I am glad I had the opportunity to stick with my custom Vulkan / C++ implementation, as this allowed me to keep learning the Vulkan Graphics API, instead of falling back to a pre-existing framework. Developing this custom framework actually gave me a lot of new insights that I want to apply to my other projects.

I also improved upon some of my soft skills throughout this project. Reading research papers was something I had not done often in the past, though this project forced me to learn this skill over time to make reading papers as efficient as possible. Another one those skills is writing. I enjoyed the process of writing this paper, writing about the theoretical framework, explaining concepts, or analysing the results. I believe writing improves only through practice, so this project also helped me develop that skill further.

I was also really surprised by the survey results. I did not expect the results for ozone absorption at low sun angles to be so dominant on the non-ozone side. But after thinking about it, there was a logical reason behind it as I explained in the Discussion section. I ended up having 43 participants complete the survey, and while I secretly hoped for slightly more, I am happy with the results I got. Overall, I am happy with the results and the work I delivered.

Finally, ever since enrolling into the "*Graphics Programming 1*" course at Digital Arts and Entertainment, I knew Graphics Programming was what I wanted to pursue further. This research project allowed me to learn more about graphics programming and it further confirmed that this is truly what I want to do.

16 REFERENCES

[1] "Microsoft Flight Simulator," Microsoft Flight Simulator Celebrates 10 Million Pilots. [Online]. Available: https://www.flightsimulator.com/microsoft-flight-simulator-celebrates-10-million-pilots/

[2] "Active Sky FS," HiFi Simulation Technologies. [Online]. Available: https://hifisimtech.com/asfs/

[3] E. Belorizky, "The colours of the sky," *Encyclopedia of the Environment*. May 01, 2025. [Online]. Available: https://www.encyclopedie-environnement.org/en/air-en/colours-sky/

[4] Wikipedia contributors, "Rayleigh scattering — Wikipedia, The Free Encyclopedia." 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Rayleigh_scattering&oldid=1329027339

[5] *Sunrise to Sunset*. [Online Video]. Available: https://www.pbslearningmedia.org/resource/buac18-k2-sci-ess-sunposition/changing-position-of-the-sun-in-the-sky/

[6] *Observe Sun Patterns*. [Online Video]. Available: https://www.pbslearningmedia.org/resource/buac18-k2-sci-ess-sunposition/changing-position-of-the-sun-in-the-sky/

[7] *Blue Sky*. [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/atmos/blusky.html

[8] Wikipedia contributors, "Chappuis absorption — Wikipedia, The Free Encyclopedia." 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Chappuis_absorption&oldid=1313426015

[9] S. An, L. Wang, and L. Wang, "Semi-supervised dehazing network using multiple scattering model and fuzzy image prior," *Appl. Intell.*, vol. 54, pp. 1–19, Apr. 2024, doi: 10.1007/s10489-024-05443-9.

[10] J. W. Strutt, "XV. On the light from the sky, its polarization and colour," *Lond. Edinb. Dublin Philos. Mag. J. Sci.*, vol. 41, no. 271, pp. 107–120, 1871, doi: 10.1080/14786447108640452.

[11] Wikipedia contributors, "Visible spectrum — Wikipedia, The Free Encyclopedia." 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Visible_spectrum&oldid=1319443846

[12] L. Vočadlo, "Why is the sky blue?," UCL Culture Online. [Online]. Available: https://www.ucl.ac.uk/culture-online/case-studies/2022/mar/why-sky-blue

[13] W. Hordijk, "Why are sunsets red?," Plus. [Online]. Available: https://plus.maths.org/content/why-are-sunsets-red

[14] Wikipedia contributors, "Aerosol — Wikipedia, The Free Encyclopedia." 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Aerosol&oldid=1328714671

[15] G. Mie, "Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen," *Ann. Phys.*, vol. 330, no. 3, pp. 377–445, Jan. 1908, doi: 10.1002/andp.19083300302.

[16] D. Lopes and A. R. Fernandes, "Atmospheric Scattering - State of the Art," in *21o Encontro Português de Computação Gráfica*, A. Goncalves, A. R. Fernandes, and N. Rodrigues, Eds., The Eurographics Association, 2020. doi: 10.2312/pt.20141310.

[17] Wikipedia contributors, "Ozone layer — Wikipedia, The Free Encyclopedia." 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Ozone_layer&oldid=1318147102

[18] 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*, in SIGGRAPH '93. New York, NY, USA: Association for Computing Machinery, 1993, pp. 175–182. doi: 10.1145/166117.166140.

[19] S. O'Neil, "Accurate Atmospheric Scattering," in *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, Addison-Wesley Professional, 2005, pp. 253–268.

[20] N. Hoffman and A. J. Preetham, "Photorealistic Real-Time Outdoor Light Scattering," *Game Dev. Mag. CMP Media Inc*, vol. 9, pp. p32-38, Aug. 2002.

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

[22] S. Hillaire, “A scalable and production ready sky and atmosphere rendering technique,” in *Computer Graphics Forum*, Wiley Online Library, 2020, pp. 13–22.

[23] G. Bodare and E. Sandberg, “Efficient and Dynamic Atmospheric Scattering,” 2014.

[24] W. Cornette and J. Shanks, “Physically reasonable analytic expression for the single-scattering phase function,” *Appl. Opt.*, vol. 31, pp. 3152–3160, Jun. 1992, doi: 10.1364/AO.31.003152.

[25] L. G. Henyey and J. L. Greenstein, “Diffuse radiation in the Galaxy.,” *apj*, vol. 93, pp. 70–83, Jan. 1941, doi: 10.1086/144246.

[26] “Ozone facts,” NASA Ozone Watch. [Online]. Available: https://ozonewatch.gsfc.nasa.gov/facts/SH.html

[27] P. Kutz, “Ozone Absorption.” [Online]. Available: https://skyrenderer.blogspot.com/2012/10/ozone-absorption.html

[28] Wikipedia contributors, “Riemann sum — Wikipedia, The Free Encyclopedia.” 2025. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Riemann_sum&oldid=1322381525

[29] *FrameView*. (2025). NVIDIA. [Online]. Available: https://www.nvidia.com/en-us/geforce/technologies/frameview/

[30] Bilge Şeyma Kütükoğlu, *Clear Sky over Sea Coast at Sunset*. [Photography]. Available: https://www.pexels.com/photo/clear-sky-over-sea-coast-at-sunset-17359252/

17 ACKNOWLEDGEMENTS

I would like to thank some people that contributed meaningfully to this research paper.

First, I would like to thank Thomas Goussaert, my supervisor for this research project, and Stephanie Defoort, my coach throughout the process, for all their valuable feedback, guidance, and help during the development of this bachelor thesis.

Secondly, I would like to thank the teachers of Digital Arts and Entertainment for equipping me with the skills and knowledge that made this research possible. Prior to enrolling in the Game Development major, I had no coding experience, yet through their instructions, support, and feedback, I was able to reach this stage of the study program.

Furthermore, I would like to thank my colleagues for their support, as well as everyone who took the time to proofread this paper and provide constructive feedback. Specifically I want to thank Thalia Tritar for her support and help in spreading the survey, Ivans Minajevs for his support and help throughout the project, and Gilles Durnez for helping me get on track during the early stages of this research.

Finally, I would like to thank all participants who completed the survey for this research. Without their contributions and insights, this study would not have been possible.

18 APPENDICES

18.1 SHADERS

For all source code visit the GitHub page of this research project here (https://github.com/Kobazaaa/Ashen).

18.1.1 SKYFROMATMOSPHERE.VERT

```
#version 450
#extension GL_GOOGLE_include_directive : require

layout(push_constant) uniform PushConstants
{
    mat4 view;
    mat4 proj;
} pc;

#include "Helper_Scattering.glsl"

layout(location = 0) in vec3 inPosition;
layout(location = 1) in vec3 inColor;

layout(location = 0) out vec3 outRayleighColor;
layout(location = 1) out vec3 outMieColor;
layout(location = 2) out vec3 outDirectionToCam;


// This shader is used to render the sky dome when the camera is in the atmosphere
// This means that the ray along which we will sample starts at the camera and ends at the current vertex
void main()
{
    // Transform the current vertex position to a world-size position instead of the scaled down render version
    float scaledHeight = planetRadius + atmosphereThickness;
    vec3 scaledPos = normalize(inPosition) * scaledHeight;

    // Get the ray from the Camera to the current Vertex,
    // the length of this ray is the far point of the ray passing through the atmosphere
    vec3 startPos = cameraPos;
    vec3 endPos = scaledPos;
    vec3 ray = endPos - startPos;
    float farDistance = length(ray);
    ray /= farDistance;

    // Initialize the scattering loop variables
    float travelDistance = farDistance;
    float sampleLength = travelDistance / sampleCount;
    vec3 sampleRay = ray * sampleLength;
    vec3 samplePoint = startPos + sampleRay * 0.5;

    // Loop through the sample points
    vec3 frontColorR = vec3(0);
    vec3 frontColorM = vec3(0);
    for(int i = 0; i < sampleCount; ++i)
    {
        // Calculate the sample depth for rayleigh and mie
        float sampleHeightOffGround = length(samplePoint) - planetRadius;

        float sampleDepthR = DensityFunction(sampleHeightOffGround, rayleighScaleHeight);
        float sampleDepthM = DensityFunction(sampleHeightOffGround, mieScaleHeigh);

        // Calculate the attenuation
```

```
54.         vec3 exitPos = samplePoint + lightDir * DistanceToAtmosphereExit(samplePoint, lightDir);
55.         vec3 attenuation = CalculateAttenuation(startPos, samplePoint);
56.         attenuation *= CalculateAttenuation(samplePoint, exitPos);
57.
58.         // Add Color
59.         frontColorR += sampleDepthR * attenuation * sampleLength;
60.         frontColorM += sampleDepthM * attenuation * sampleLength;
61.
62.         // Advance to next sample
63.         samplePoint += sampleRay;
64.     }
65.
66.     // Finally, scale the Mie and Rayleigh Colors
67.     gl_Position = pc.proj * pc.view * vec4(inPosition, 1.0);
68.
69.     outRayleighColor = sunIntensity * betaR * INV_FOUR_PI * frontColorR;
70.     outMieColor = sunIntensity * betaM * INV_FOUR_PI * frontColorM;
71.
72.     outDirectionToCam = cameraPos - scaledPos;
73. }
```

Code Block 8: Sky dome vertex shader

18.1.2 SKYFROMATMOSPHERE.FRAG

```
 1. #version 450
 2. #extension GL_GOOGLE_include_directive : require
 3. #include "Helper_PhaseFunctions.glsl"
 4.
 5. layout(location = 0) in vec3 inRayleighColor;
 6. layout(location = 1) in vec3 inMieColor;
 7. layout(location = 2) in vec3 inDirectionToCam;
 8.
 9. layout(location = 0) out vec4 outColor;
10.
11. void main()
12. {
13.     // Angle between light and -view direction
14.     float cosine = dot(lightDir, inDirectionToCam) / length(inDirectionToCam);
15.     float cosine2 = cosine * cosine;
16.
17.     float phaseR = GetRayleighPhase(cosine2);
18.     float phaseM = GetMiePhase(cosine, cosine2, g, g2);
19.
20.     vec3 c = phaseR * inRayleighColor
21.             + phaseM * inMieColor;
22.
23.     outColor = vec4(c, c.b);
24. }
```

Code Block 9: Sky dome fragment shader

18.1.3 HELPER_SCATTERING.GLSL

```
layout(set = 0, binding = 0) uniform Parameters
{
    vec3 cameraPos;                // current camera pos
    float cameraHeight;            // current camera height

    vec3 lightDir;                 // direction of to sunlight
    float sampleCount;             // nr of samples along the ray

    vec3 betaR;                    // rayleigh coefficient
    float atmosphereThickness;     // thickness of the atmosphere

    vec3 betaM;                    // mie coefficient
    float planetRadius;            // planetary radius

    vec3 betaO;                    // ozone coefficient
    float rayleighScaleHeight;     // scale height rayleigh (the altitude at which the average atmospheric density is found)
    float mieScaleHeigh;           // scale height mie (the altitude at which the average atmospheric density is found)
    float sunIntensity;            // intensity of the sun

    float renderRadius;            // the radius of the planet at which it is renderer
    float renderThickness;         // the thickness of the atmosphere at which it is renderer
};

// -- Constants --
const float PI = 3.14159265359;
const float INV_PI = 0.31830988618;
const float FOUR_PI = 12.5663706144;
const float INV_FOUR_PI = 0.07957747154;

// -- Density Function --
float DensityFunction(float heightOffGround, float scaleHeight)
{
    return exp(- heightOffGround / scaleHeight);
}

// -- Attenuation --
vec3 CalculateAttenuation(vec3 start, vec3 end)
{
    // Make the ray
    vec3 ray = end - start;
    float travelDistance = length(ray);
    ray /= travelDistance;

    // Setup loop variables
    float sampleLength = travelDistance / sampleCount;
    vec3 sampleRay = ray * sampleLength;
    vec3 samplePoint = cameraPos + sampleRay * 0.5;

    // -- Intagration time
    vec3 resultRayleigh = vec3(0);
    vec3 resultMie = vec3(0);
    vec3 resultOzone = vec3(0);
    for(int i = 0; i < sampleCount; ++i)
    {
        float height = length(samplePoint) - planetRadius;

        // Calculate density at height
        float dR = DensityFunction(height, rayleighScaleHeight);
        float dM = DensityFunction(height, mieScaleHeigh);
        float dO = dR * 6e-7;

```

```
62.         // sum up
63.         resultRayleigh  += dR * sampleLength;
64.         resultMie       += dM * sampleLength;
65.         resultOzone     += dO * sampleLength;
66.
67.         samplePoint += sampleRay;
68.     }
69.
70.     // calibrate with coefficients
71.     resultRayleigh *= betaR;
72.     resultMie *= betaM / 0.9;
73.     resultOzone *= betaO;
74.
75.     // return attenuations
76.     return exp(-(resultRayleigh + resultMie + resultOzone));
77. }
78.
79. float DistanceToAtmosphereExit(vec3 start, vec3 dir)
80. {
81.     float b = 2.0 * dot(start, dir);
82.     float c = dot(start, start) - pow(planetRadius + atmosphereThickness, 2);
83.     float discriminant = b*b - 4.0*c;
84.
85.     if(discriminant < 0.0)
86.         return 0.0;
87.
88.     float t0 = (-b - sqrt(discriminant)) * 0.5;
89.     float t1 = (-b + sqrt(discriminant)) * 0.5;
90.
91.     return max(t0, t1);
92. }
```

Code Block 10: Scattering function helpers

18.1.4 HELPER_PHASEFUNCTIONS.GLSL

```
 1. // input
 2. layout(set = 0, binding = 1) uniform PhaseParameters
 3. {
 4.     vec3 lightDir;       // direction of the sunlight
 5.     float g;             // constant that affects symmetry of the scattering
 6.     float g2;            // g^2
 7.       uint phaseType;    // Which phase function to use
 8. };
 9.
10. // Different Phase Functions
11. float Phase_HenyeyGreenstein(float g, float g2, float cosine)
12. {
13.       float denom = pow(1 + g2 - 2*g*cosine, 1.5);
14.       float num = 1 - g2;
15.       return num / denom;
16. }
17. float Phase_DoubleHenyeyGreenstein(float cosine, float g)
18. {
19.     float alpha = 0.5;
20.
21.     float g1 = abs(g);
22.     float g2 = -g1;
23.
24.     return  alpha * Phase_HenyeyGreenstein(g1, g1*g1, cosine)
25.     + (1 - alpha) * Phase_HenyeyGreenstein(g2, g2*g2, cosine);
26. }
27. float Phase_CornetteShanks(float g, float g2, float cosine, float cosine2)
28. {
```

```
29.       return 1.5 * ((1.0 - g2) / (2.0 + g2)) * (1.0 + cosine2) / pow(1.0 + g2 - 2.0 * g *
cosine, 1.5);
30. }
31.
32.
33.
34. // Calculates the Mie phase function
35. float GetMiePhase(float cosine, float cosine2, float g, float g2)
36. {
37.     switch(phaseType)
38.     {
39.         case 0:
40.             return Phase_HenyeyGreenstein(g, g2, cosine);
41.             break;
42.         case 1:
43.             return Phase_CornetteShanks(g, g2, cosine, cosine2);
44.             break;
45.         case 2:
46.             return Phase_DoubleHenyeyGreenstein(cosine, g);
47.             break;
48.         default:
49.             return 0;
50.     }
51. }
52.
53. // Calculates the Rayleigh phase function
54. float GetRayleighPhase(float cosine2)
55. {
56.       return 0.75 + 0.75 * cosine2;
57. }
```

Code Block 11: Phase functions in code

18.2 GRAPHS

18.2.1 OZONE COMPARISONS

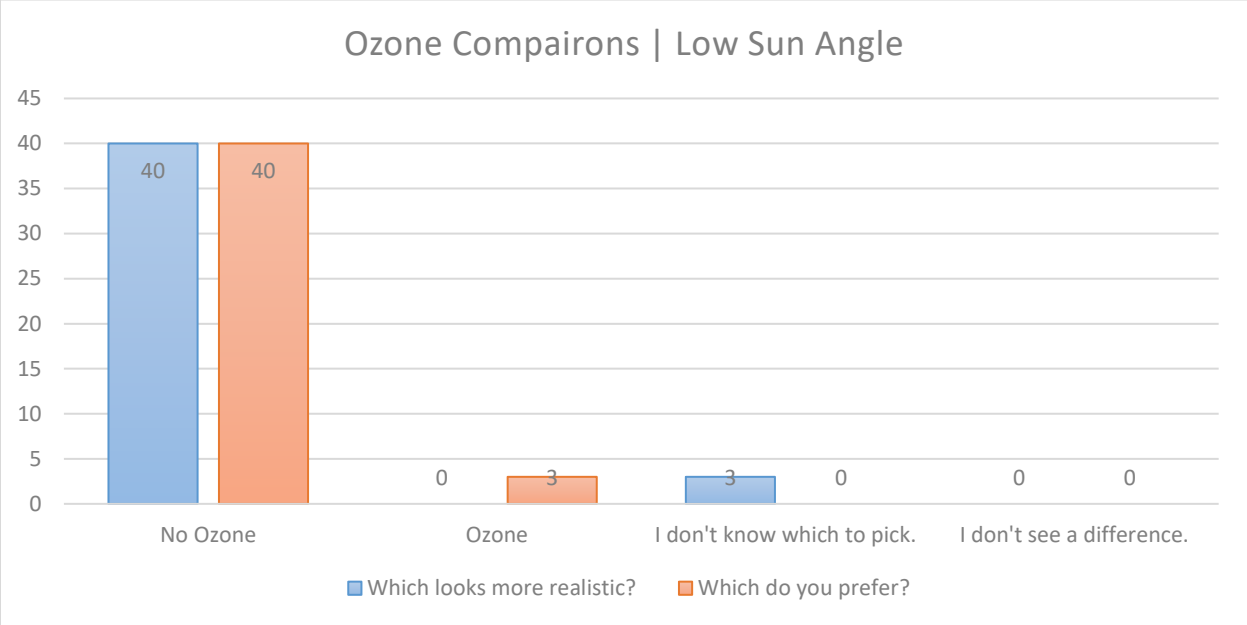

Figure 21: Ozone Comparison | Low Sun Angle

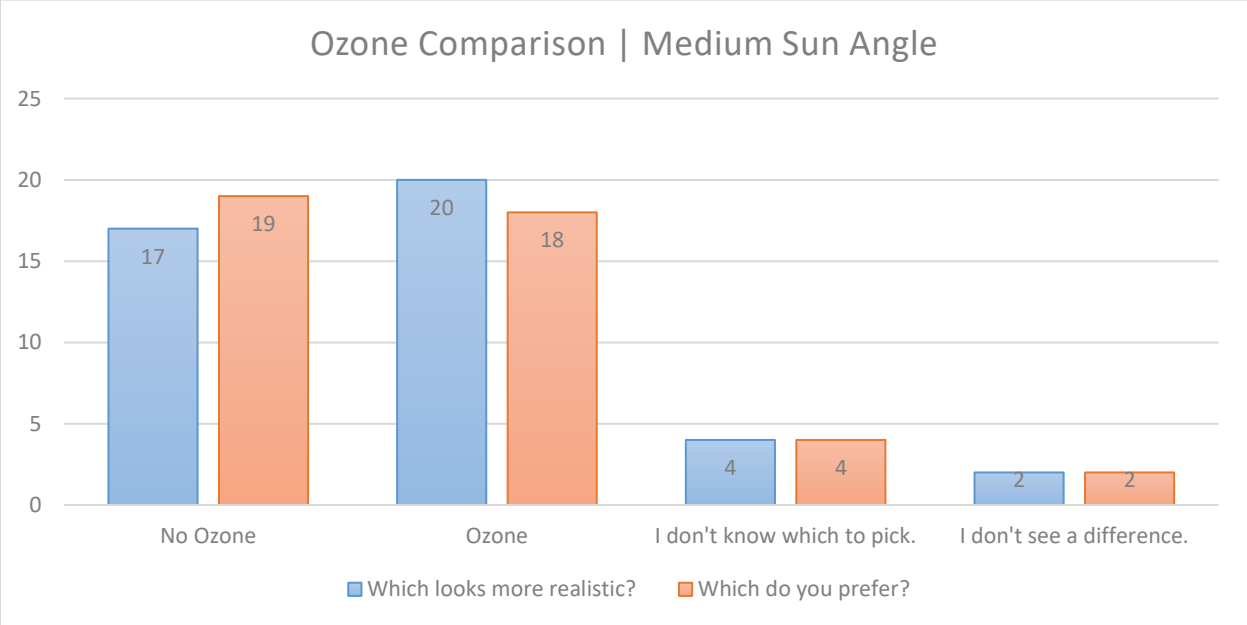

Figure 22: Ozone Comparison | Medium Sun Angle

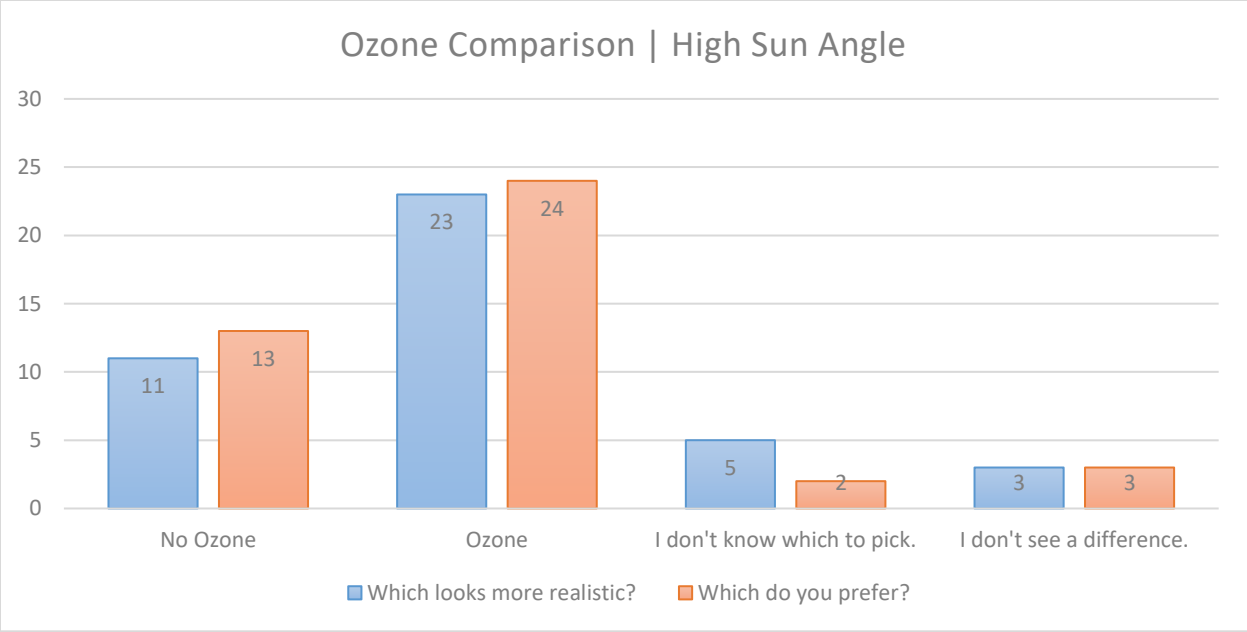

Figure 23: Ozone Comparison | High Sun Angle

18.2.2 PHASE FUNCTION COMPARISON

18.2.2.1 CORNETTE-SHANKS VS DOUBLE HENYEY GREENSTEIN

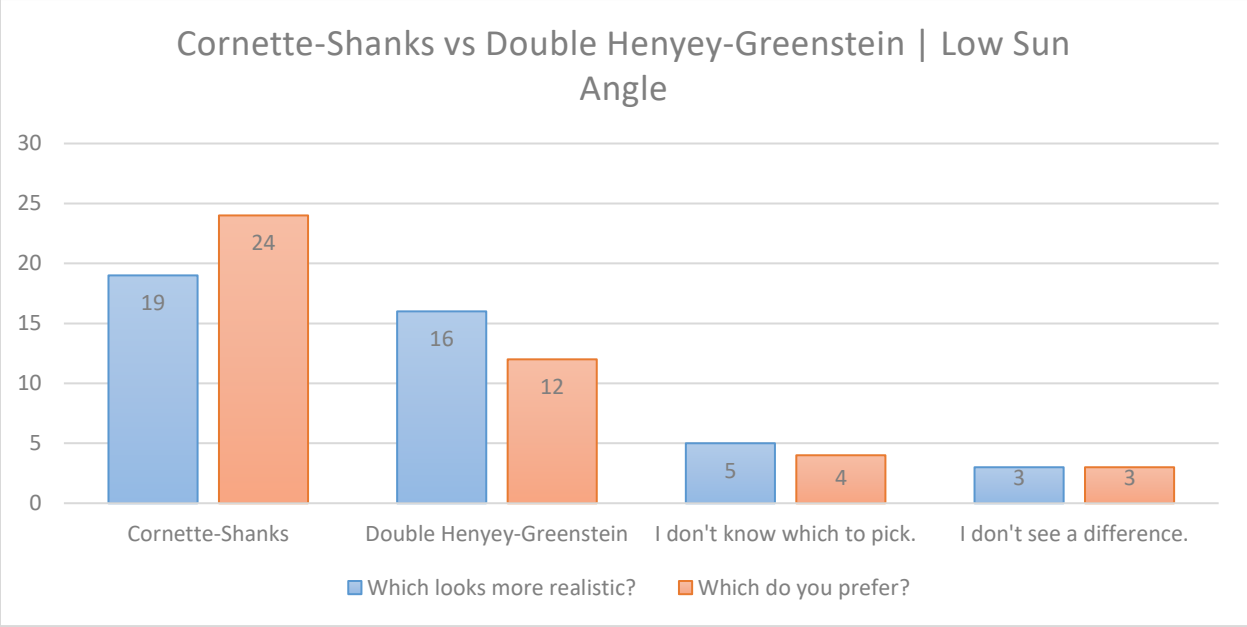

Figure 24: Cornette-Shanks vs Double Henyey-Greenstein | Low Sun Angle

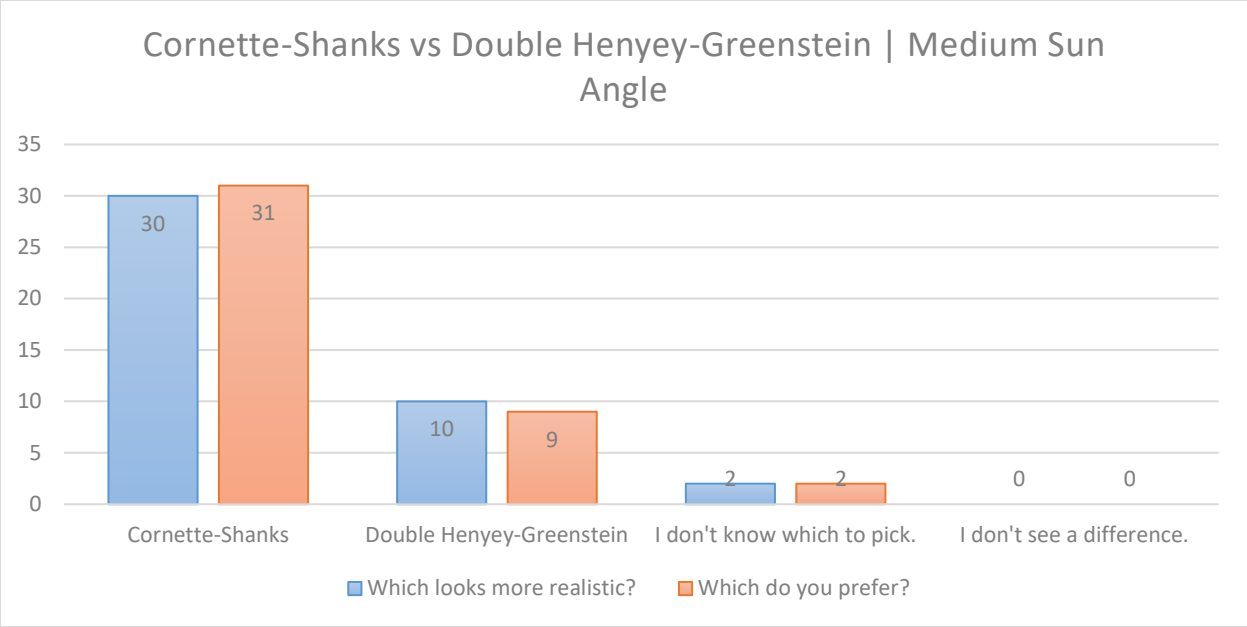

Figure 25: Cornette-Shanks vs Double Henyey-Greenstein | Medium Sun Angle

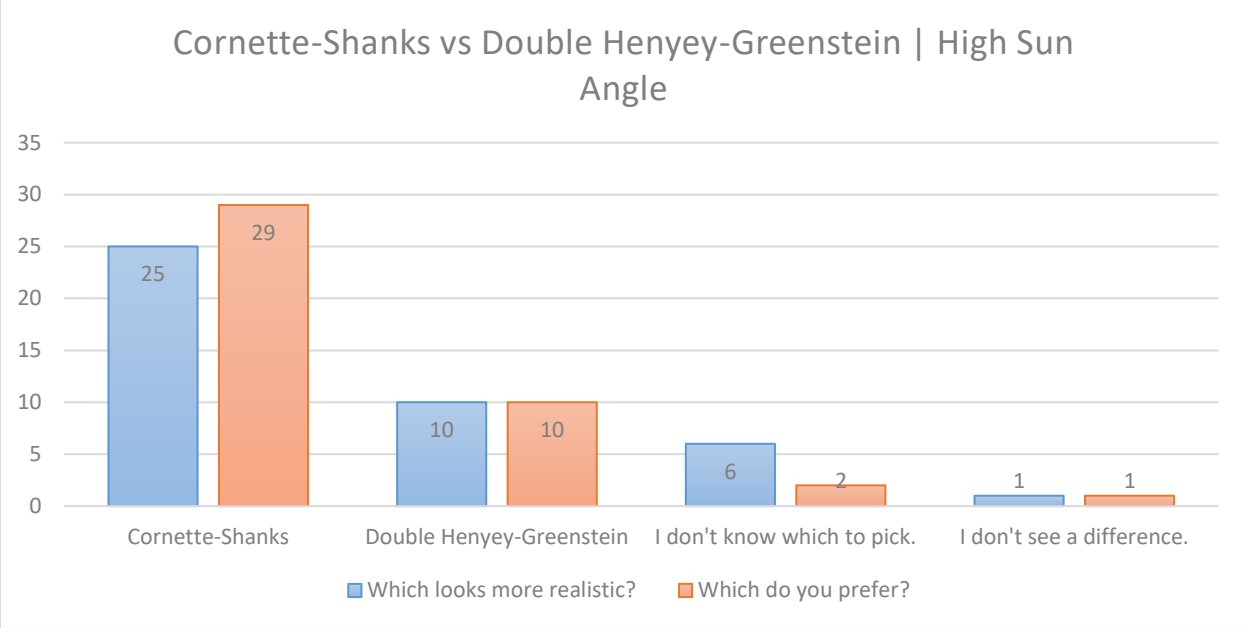

Figure 26: Cornette-Shanks vs Double Henyey-Greenstein | High Sun Angle

18.2.2.2 CORNETTE-SHANKS VS HENYEY GREENSTEIN

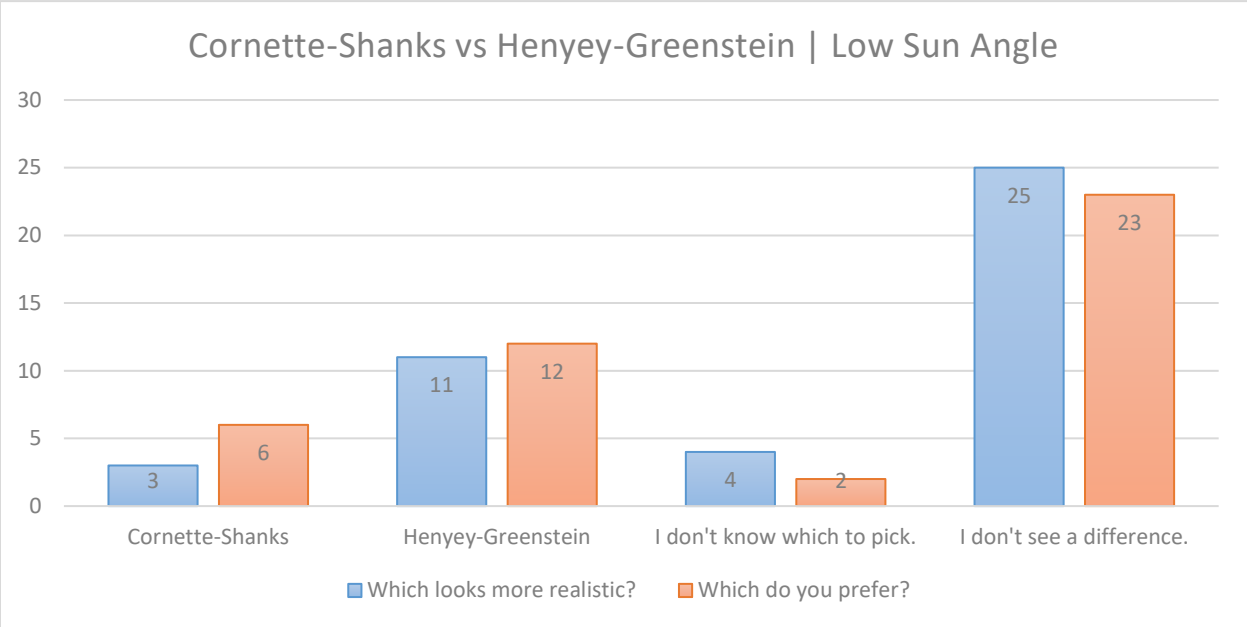

Figure 27: Cornette-Shanks vs Henyey-Greenstein | Low Sun Angle

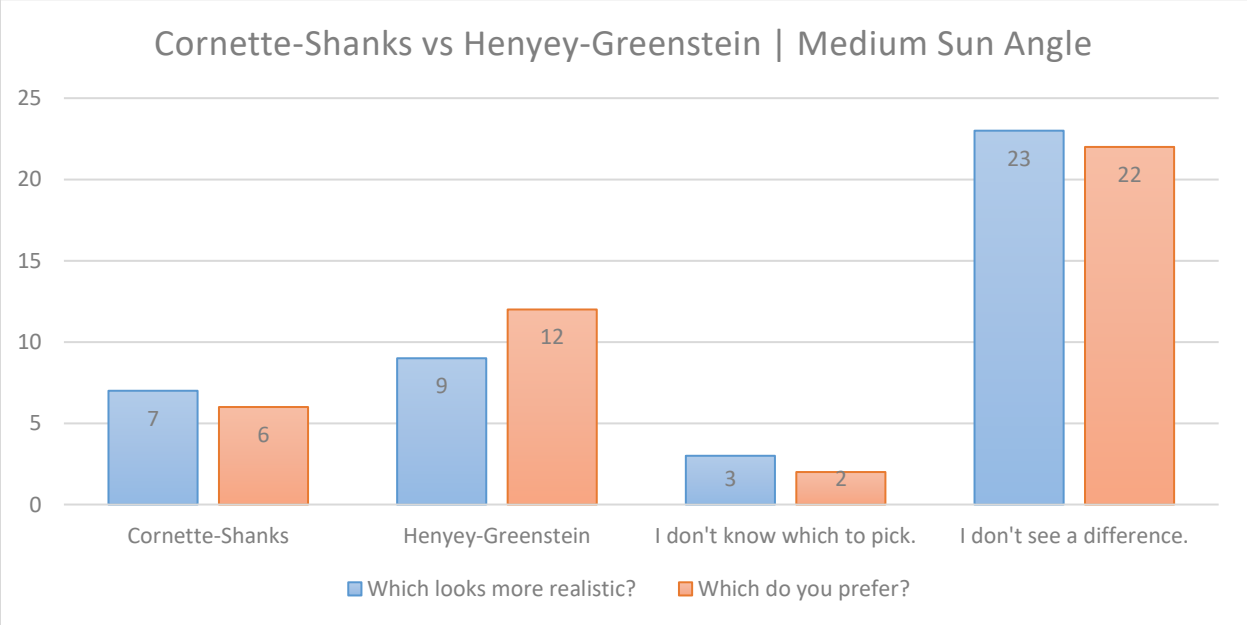

Figure 28: Cornette-Shanks vs Henyey-Greenstein | Medium Sun Angle

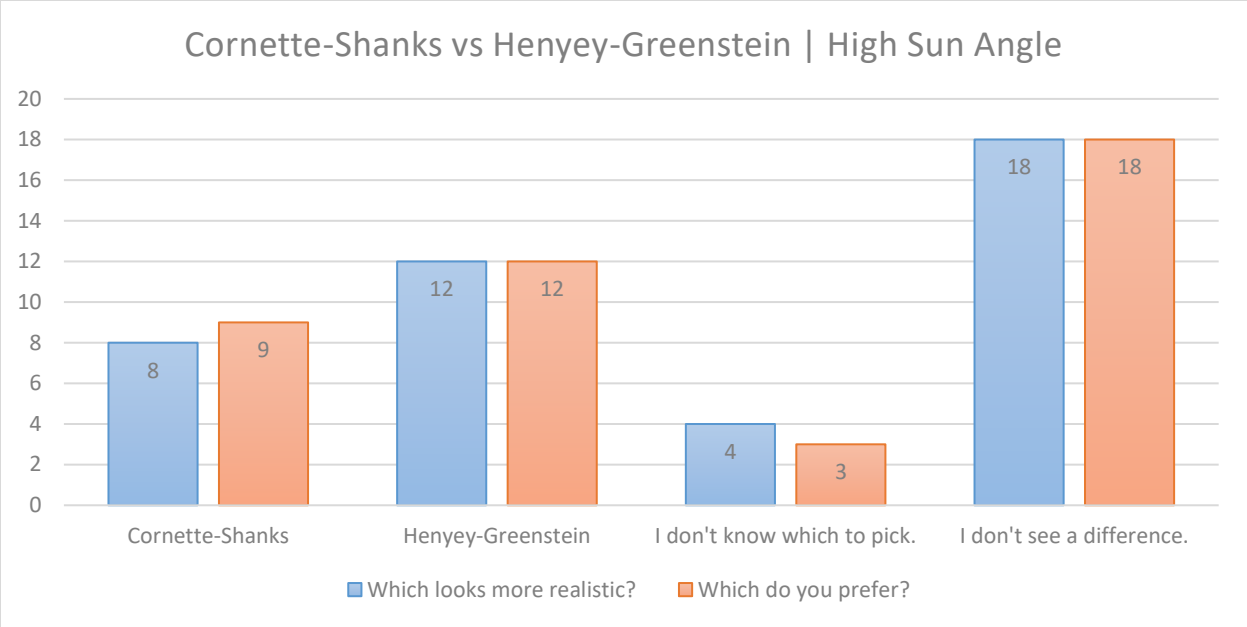

Figure 29: Cornette-Shanks vs Henyey-Greenstein | High Sun Angle

18.2.2.3 HENYEY GREENSTEIN VS DOUBLE HENYEY GREENSTEIN

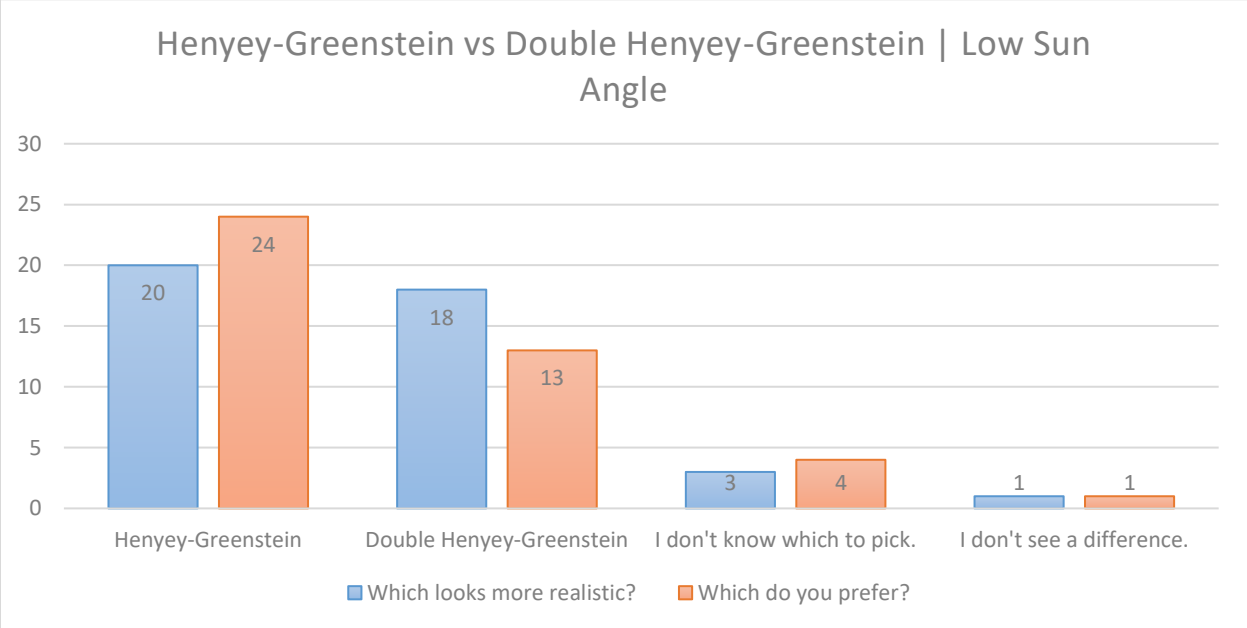

Figure 30: Double Henyey-Greenstein vs Henyey-Greenstein | Low Sun Angle

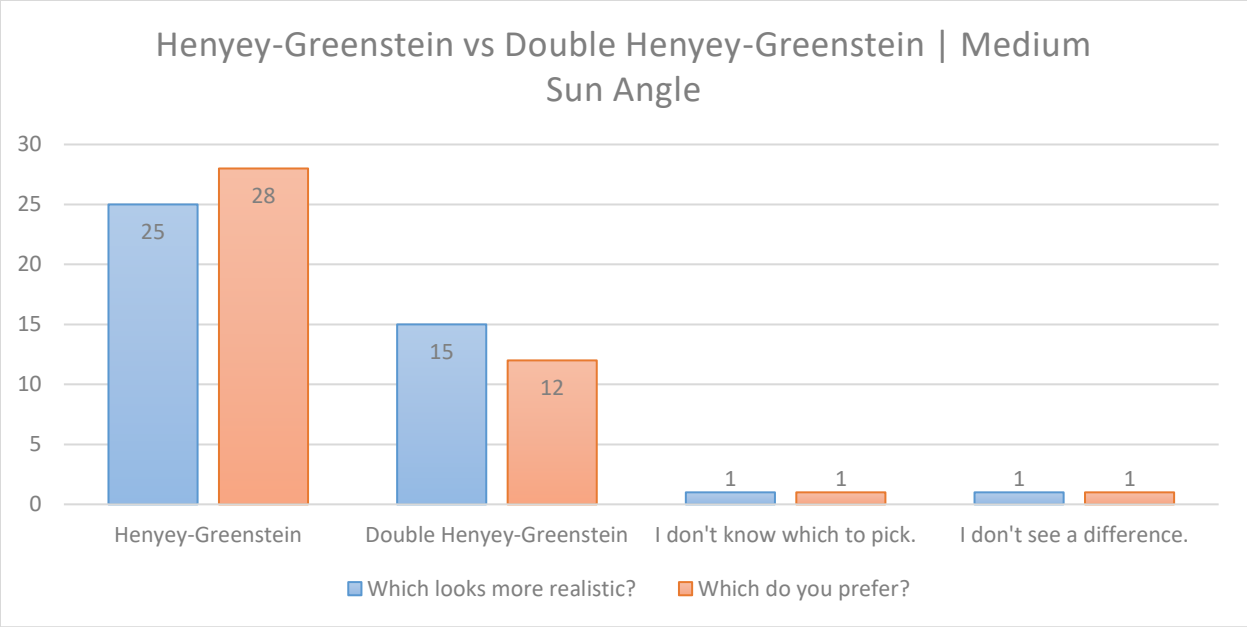

Figure 31: Double Henyey-Greenstein vs Henyey-Greenstein | Medium Sun Angle

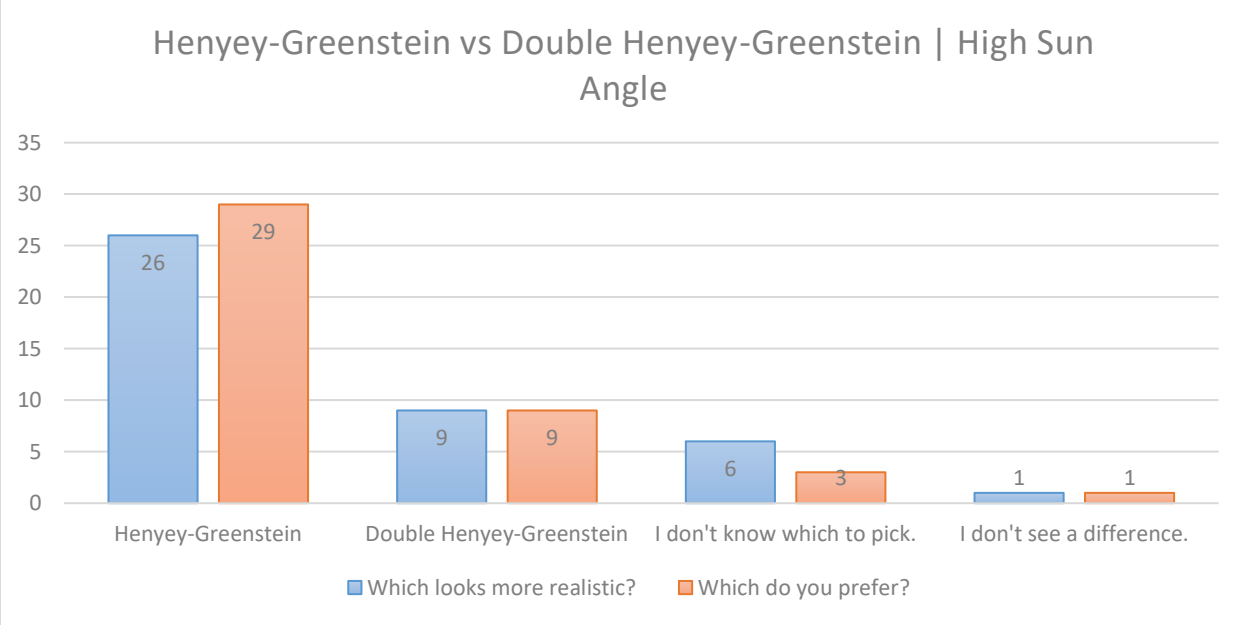

Figure 32: Double Henyey-Greenstein vs Henyey-Greenstein | High Sun Angle

18.2.3 PERFORMANCE TABLES

18.2.3.1 AVERAGE FRAME TIME

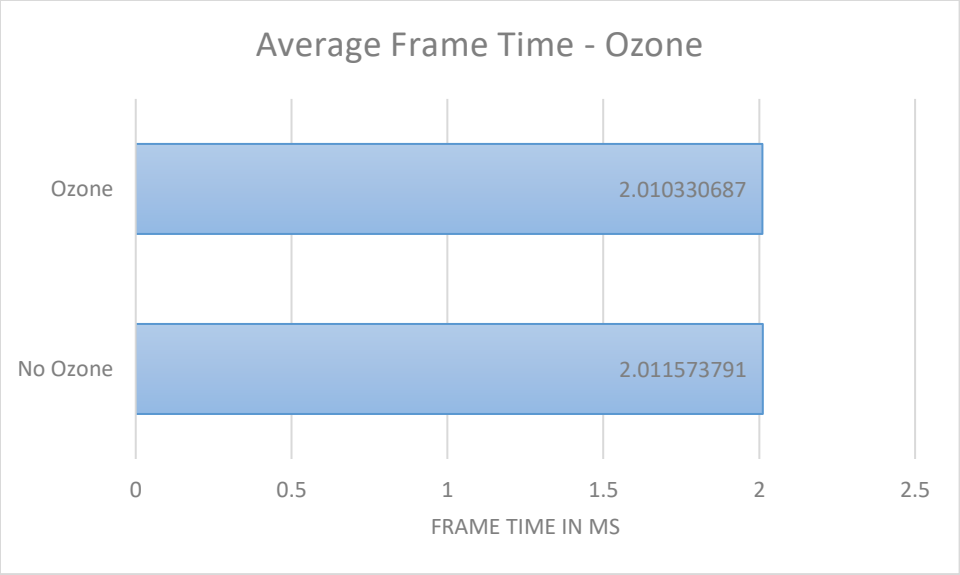

Figure 33: Average Frame Time | Ozone Absorption

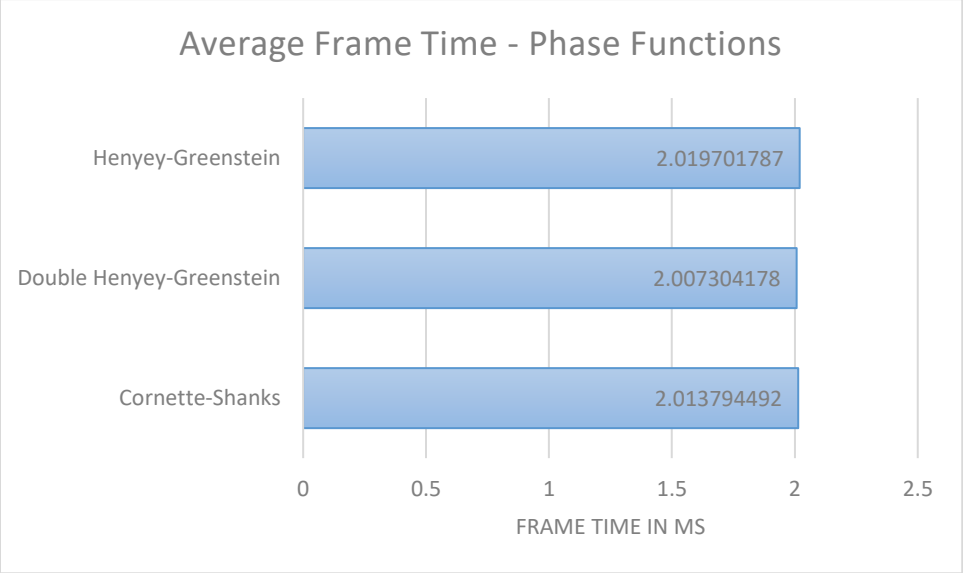

Figure 34: Average Frame Time | Phase Functions

18.2.3.2 AVERAGE GPU UTILIZATION

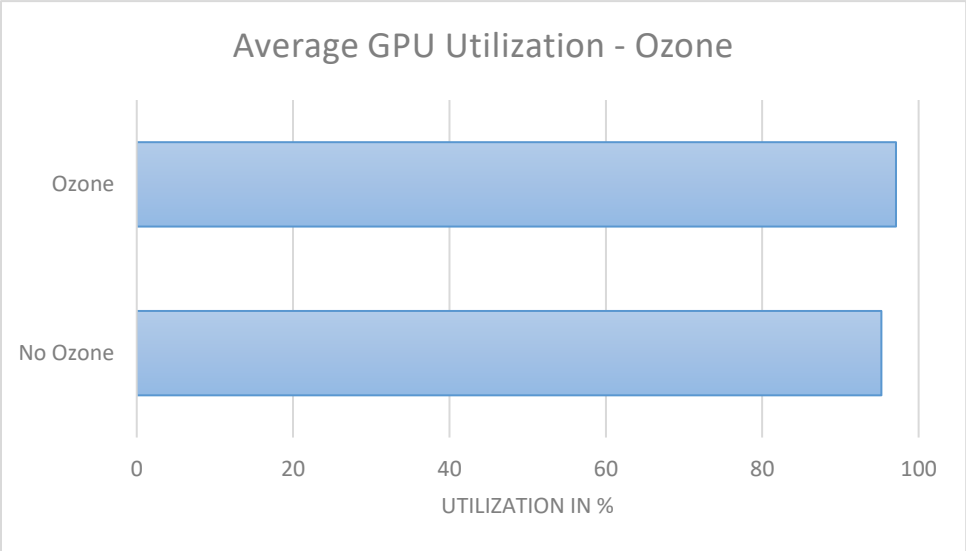

Figure 35: Average GPU Utilization | Ozone Absorption

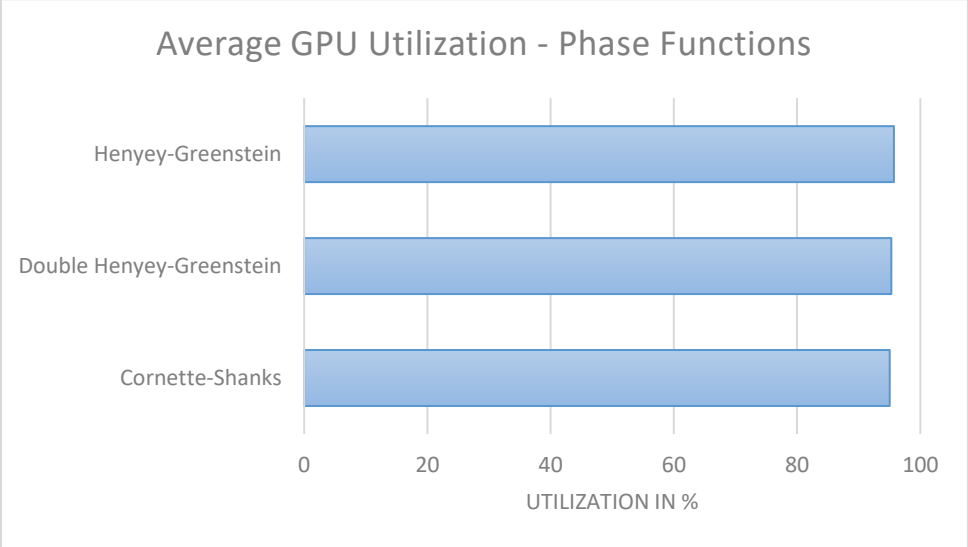

Figure 36: Average GPU Utilization | Phase Functions

18.3 IMAGES

18.3.1 OZONE ABSORPTION

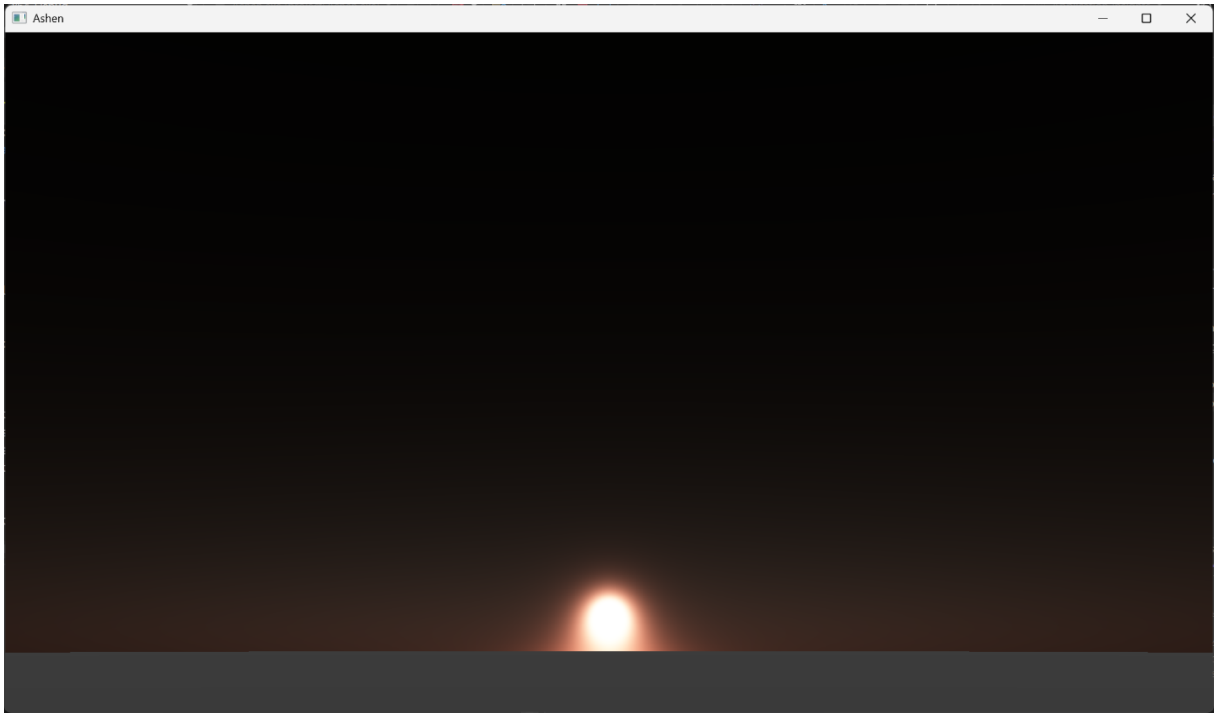

Figure 37: Ozone | Cornette-Shanks | Low Sun Angle

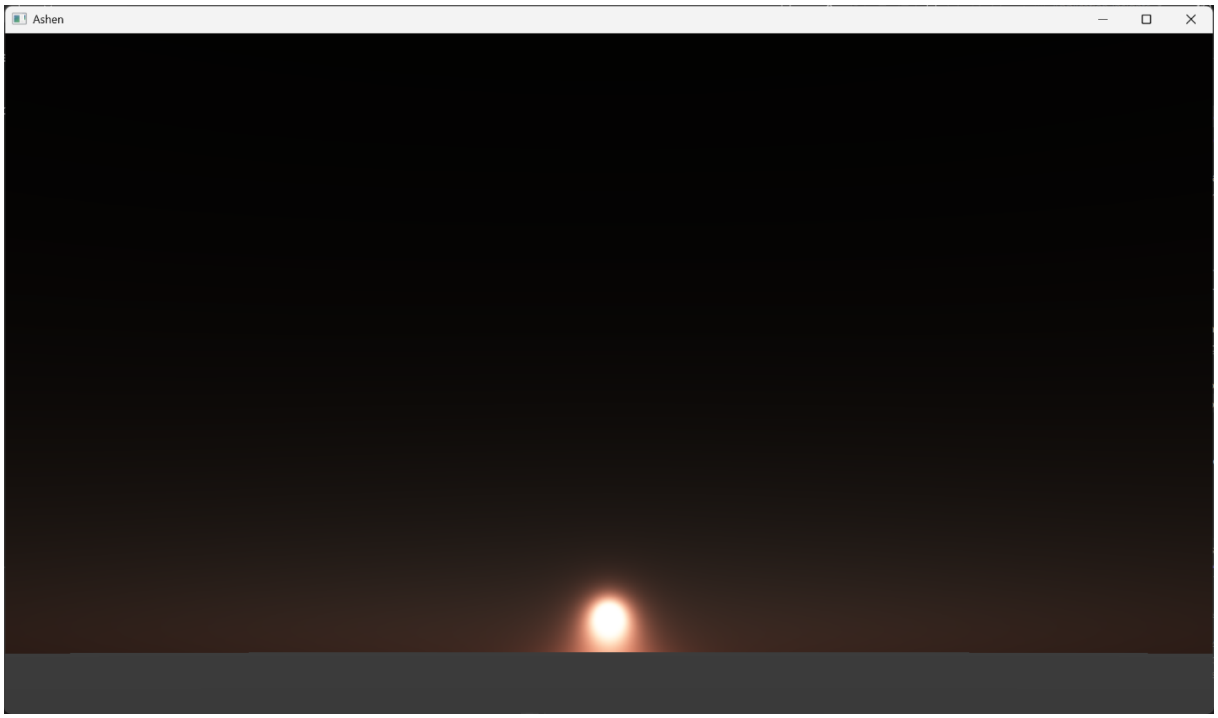

Figure 38: Ozone | Double Henyey-Greenstein | Low Sun Angle

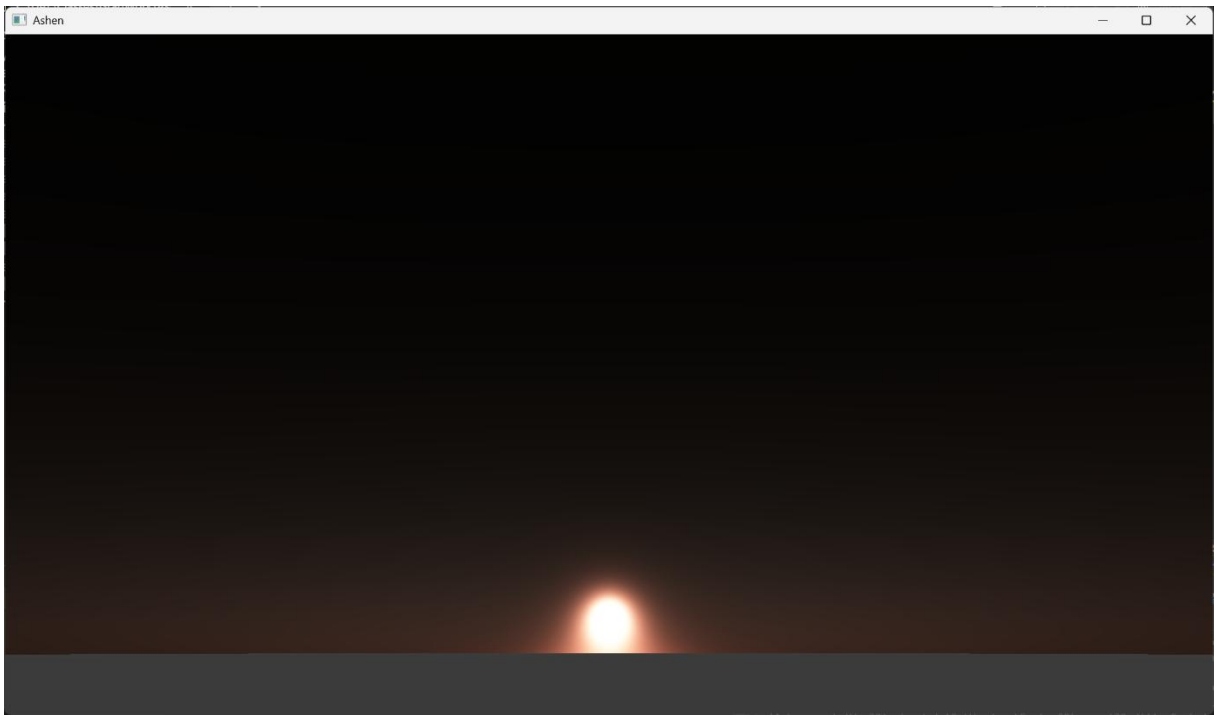

Figure 39: Ozone | Henyey-Greenstein | Low Sun Angle

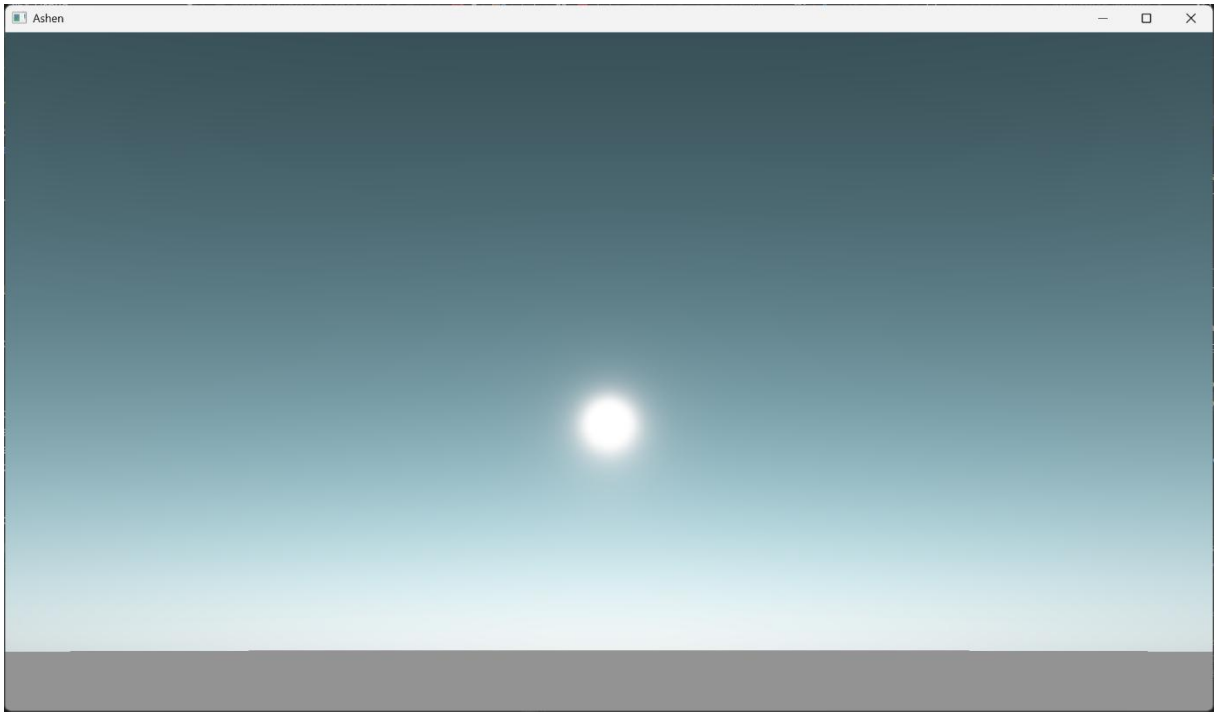

Figure 40: Ozone | Cornette-Shanks | Medium Sun Angle

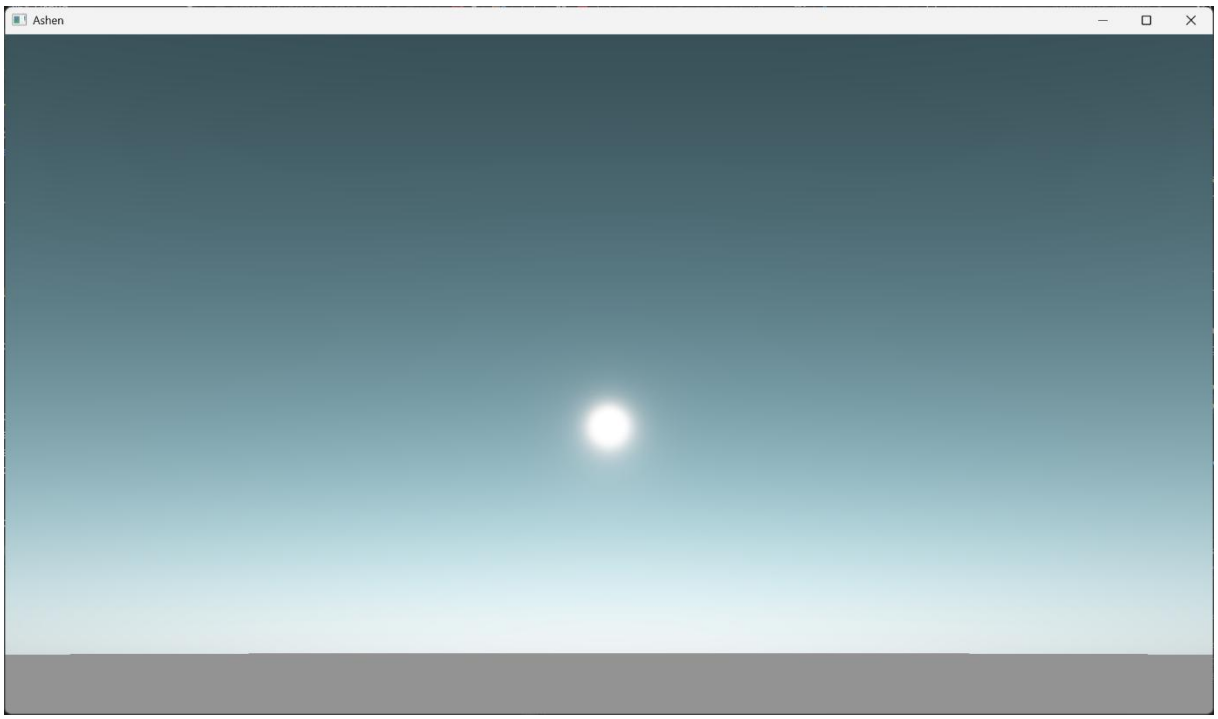

Figure 41: Ozone | Double Henyey-Greenstein | Medium Sun Angle

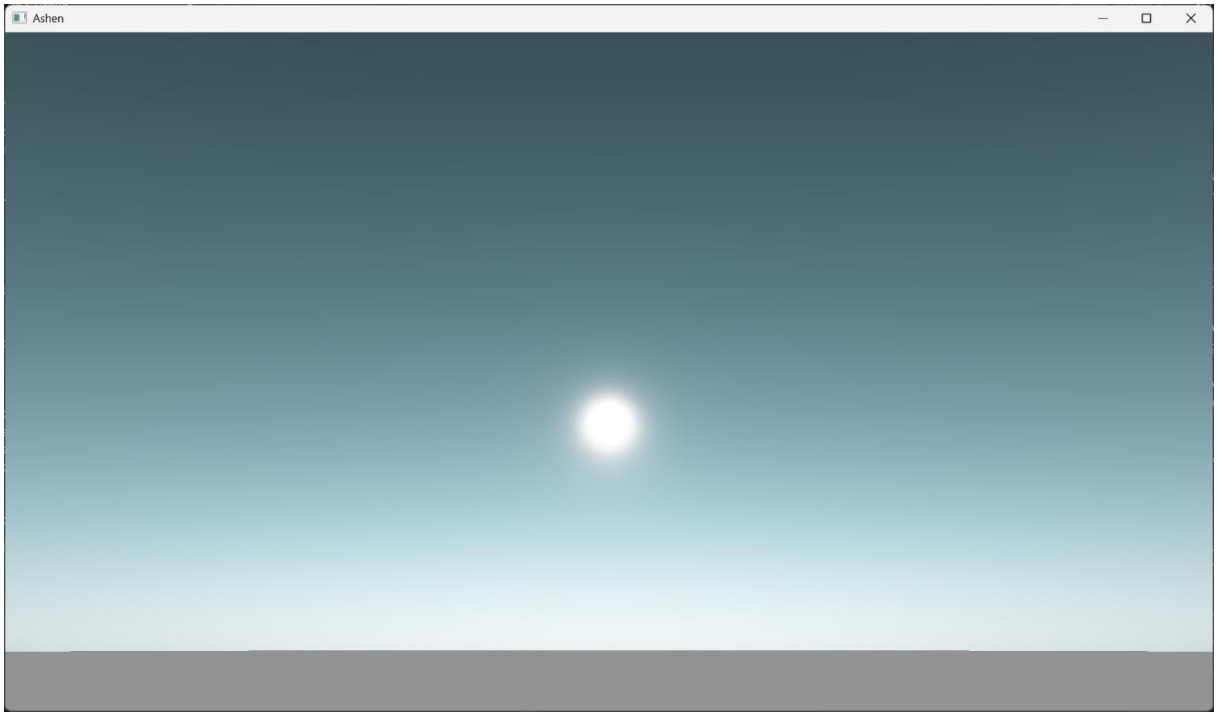

Figure 42: Ozone | Henyey-Greenstein | Medium Sun Angle

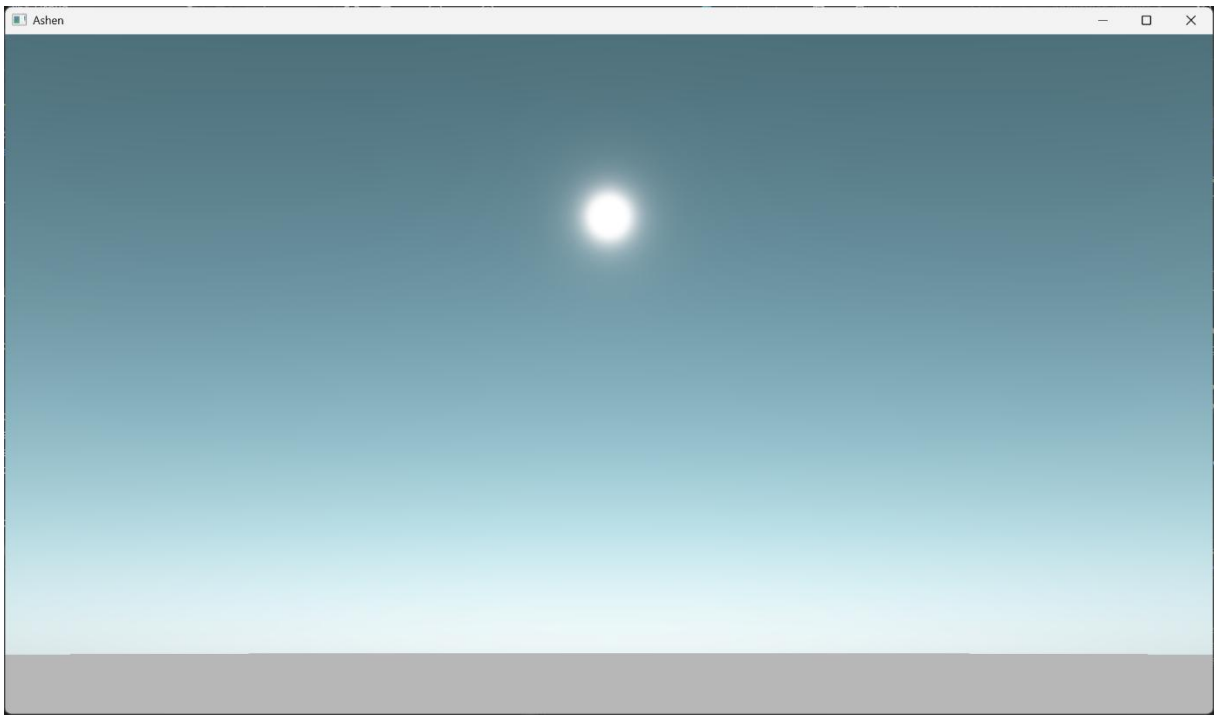

Figure 43: Ozone | Cornette-Shanks | High Sun Angle

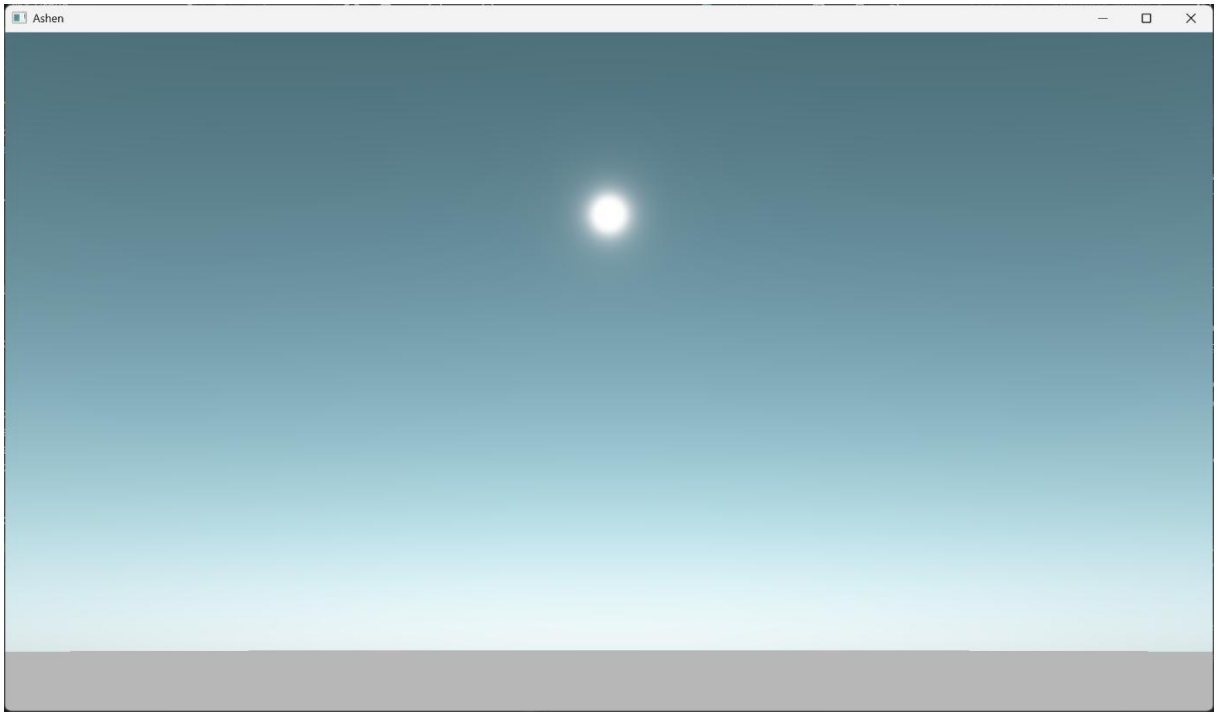

Figure 44: Ozone | Double Henyey-Greenstein | High Sun Angle

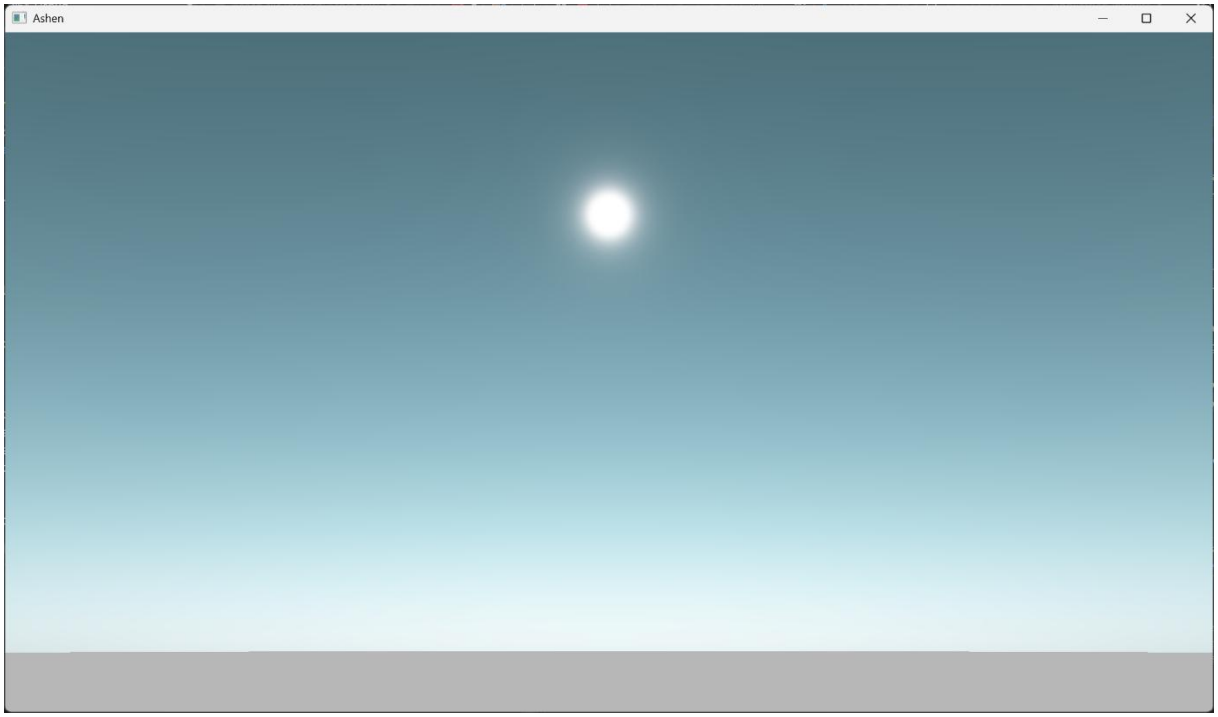

Figure 45: Ozone | Henyey-Greenstein | High Sun Angle

18.3.2 NO OZONE ABSORPTION

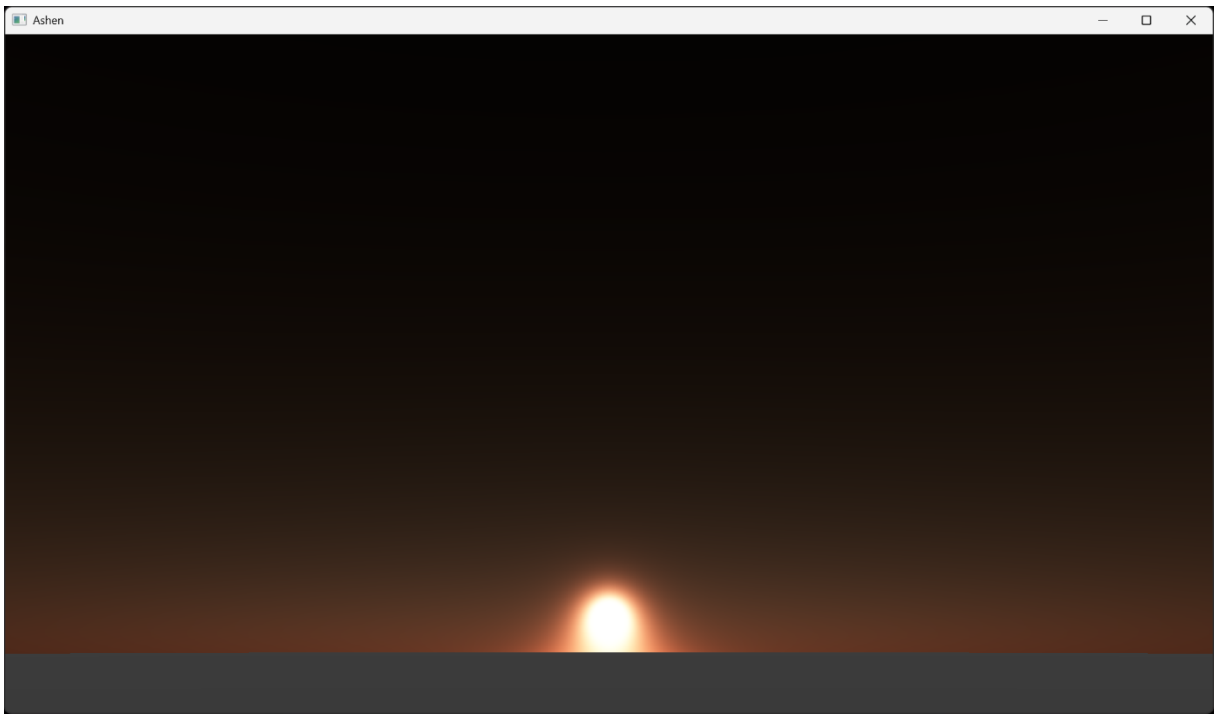

Figure 46: No Ozone | Cornette-Shanks | Low Sun Angle

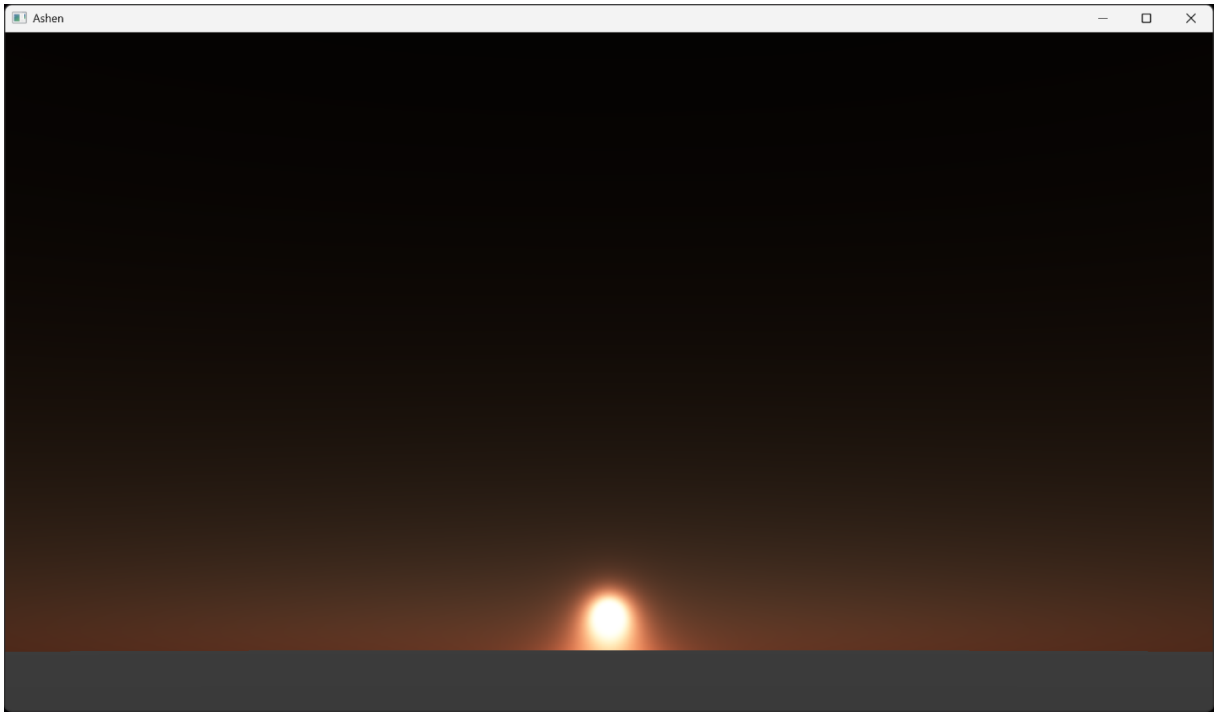

Figure 47: No Ozone | Double Henyey-Greenstein | Low Sun Angle

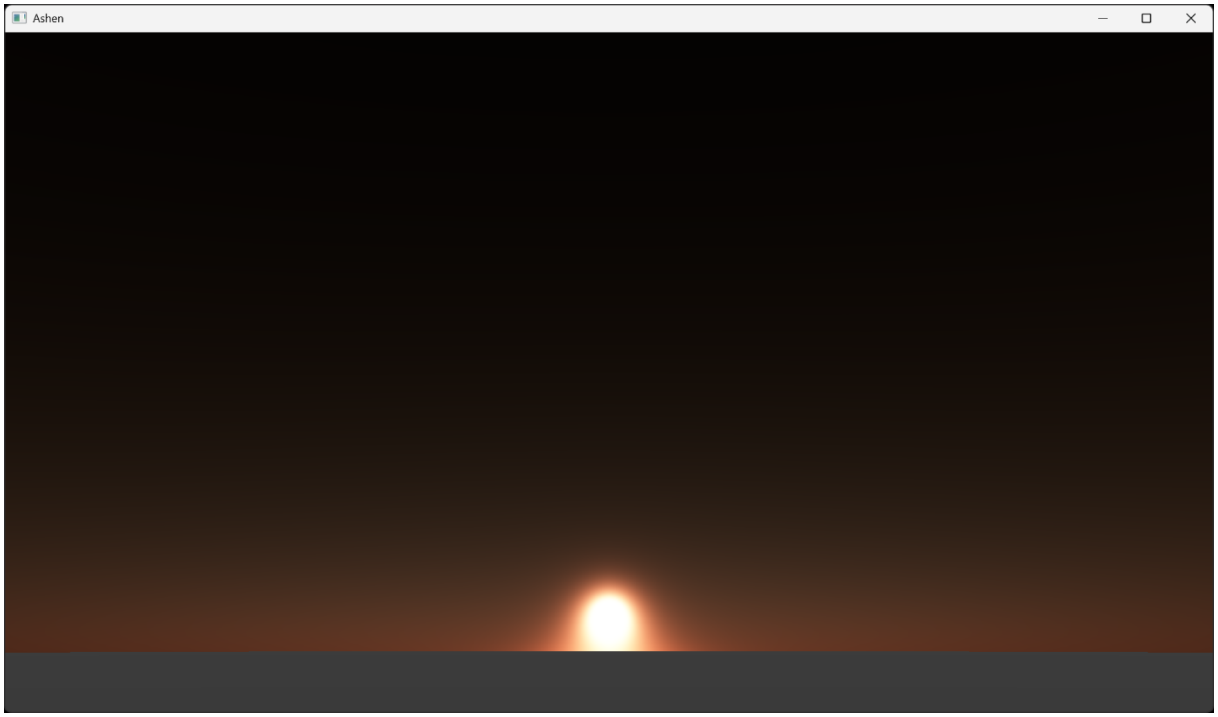

Figure 48: No Ozone | Henyey-Greenstein | Low Sun Angle

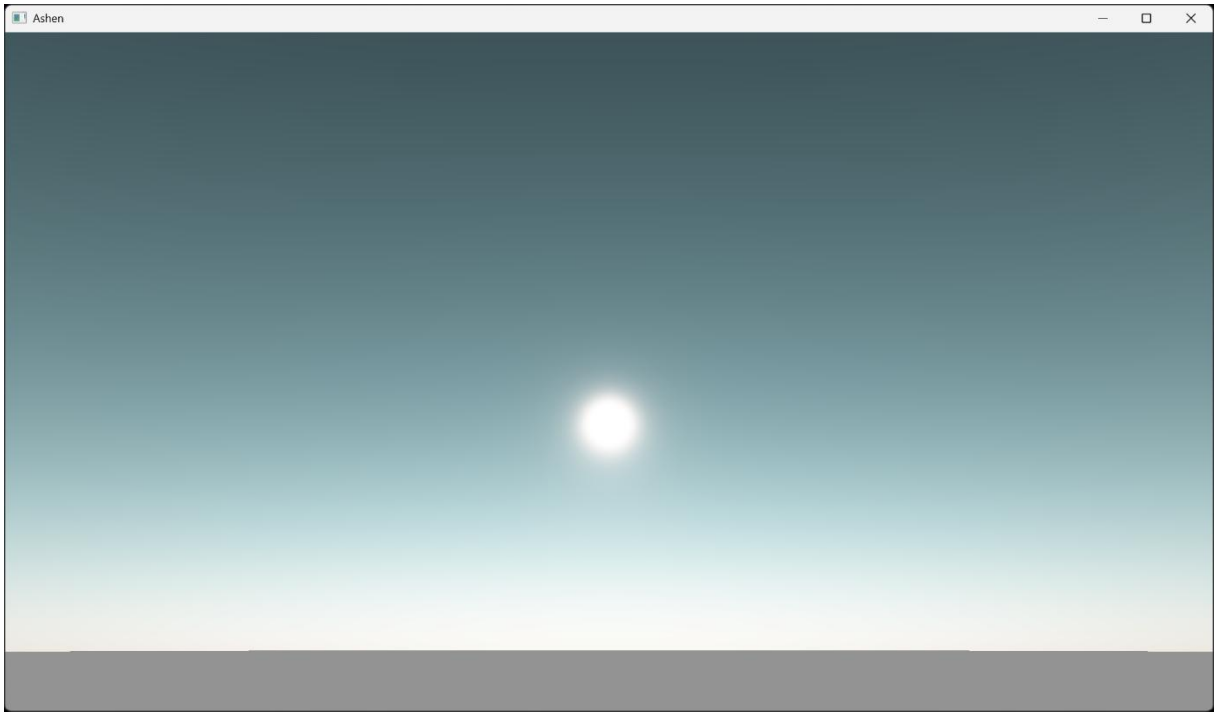

Figure 49: No Ozone | Cornette-Shanks | Medium Sun Angle

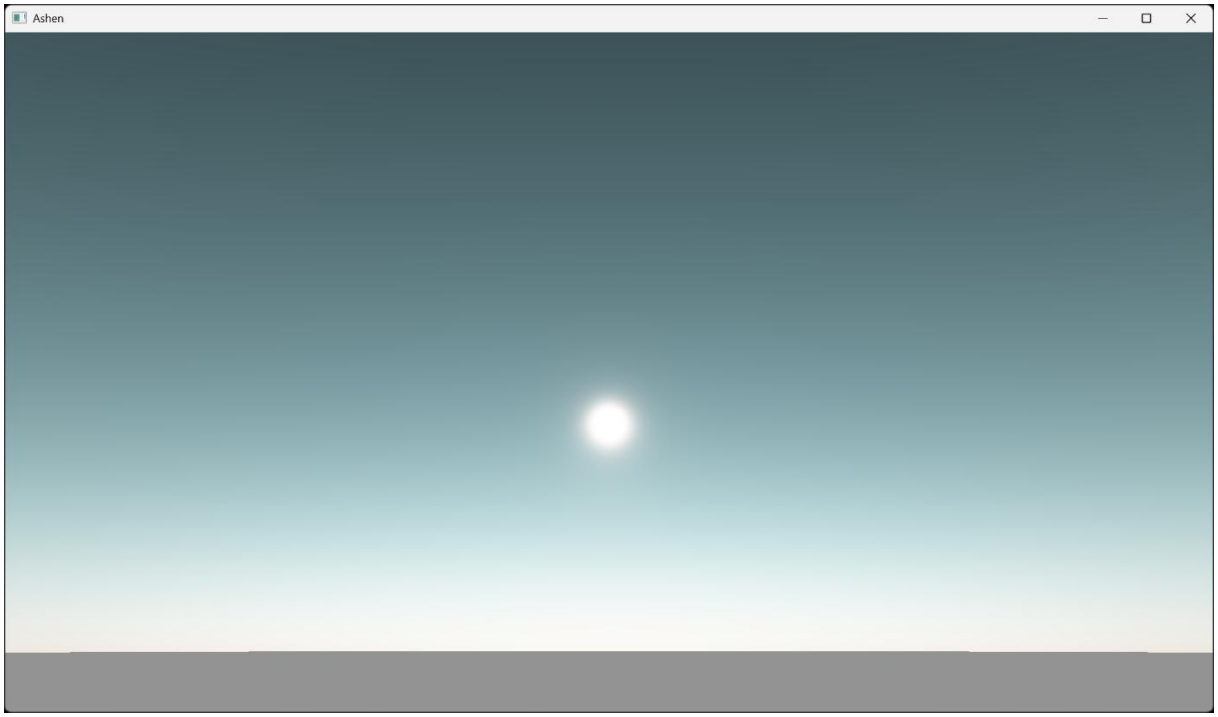

Figure 50: No Ozone | Double Henyey-Greenstein | Medium Sun Angle

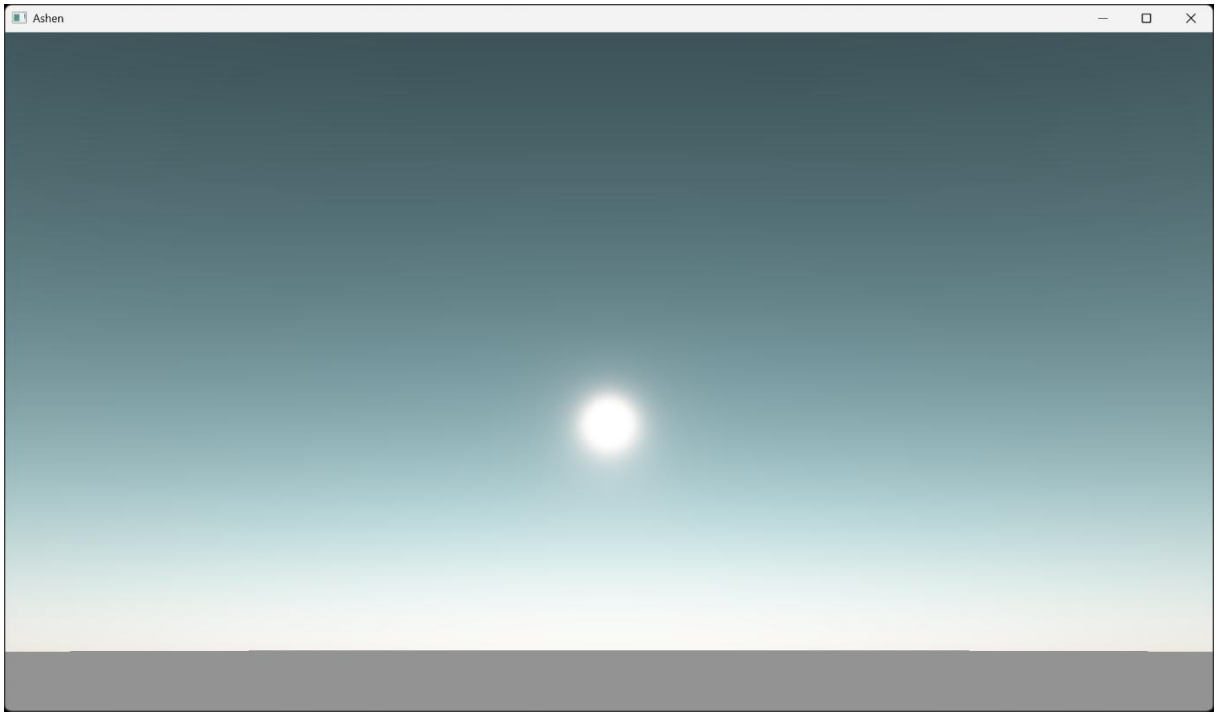

Figure 51: No Ozone | Henyey-Greenstein | Medium Sun Angle

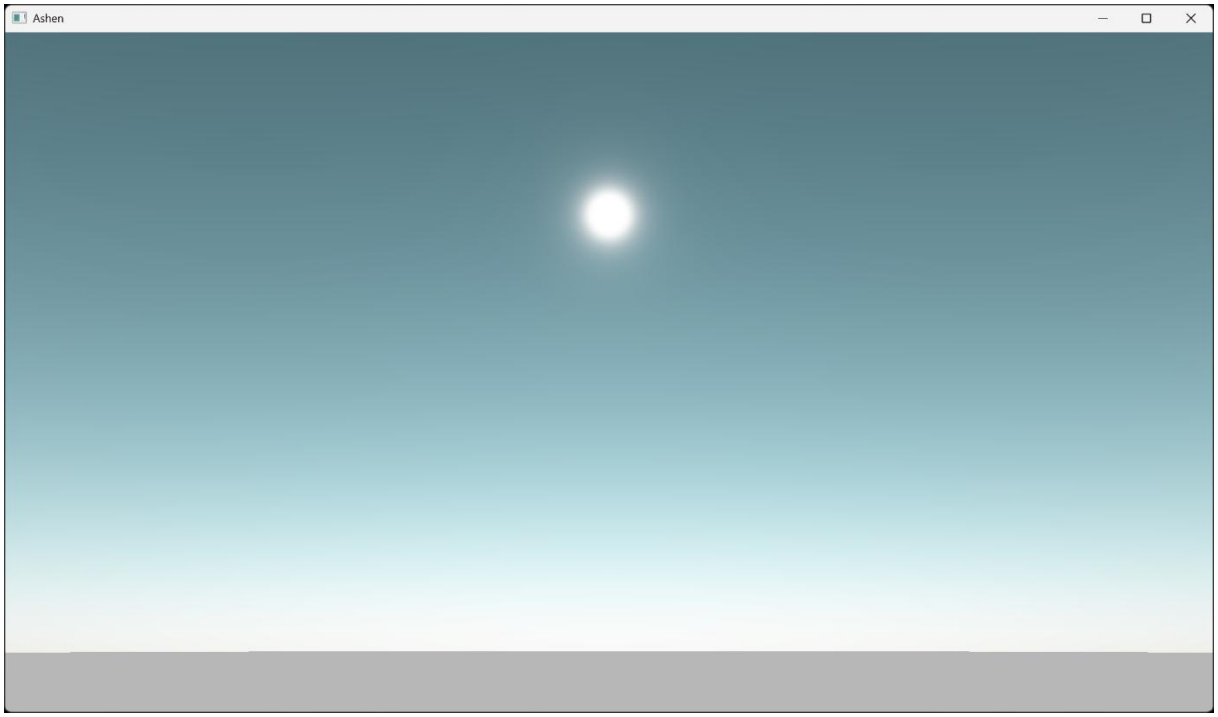

Figure 52: No Ozone | Cornette-Shanks | High Sun Angle

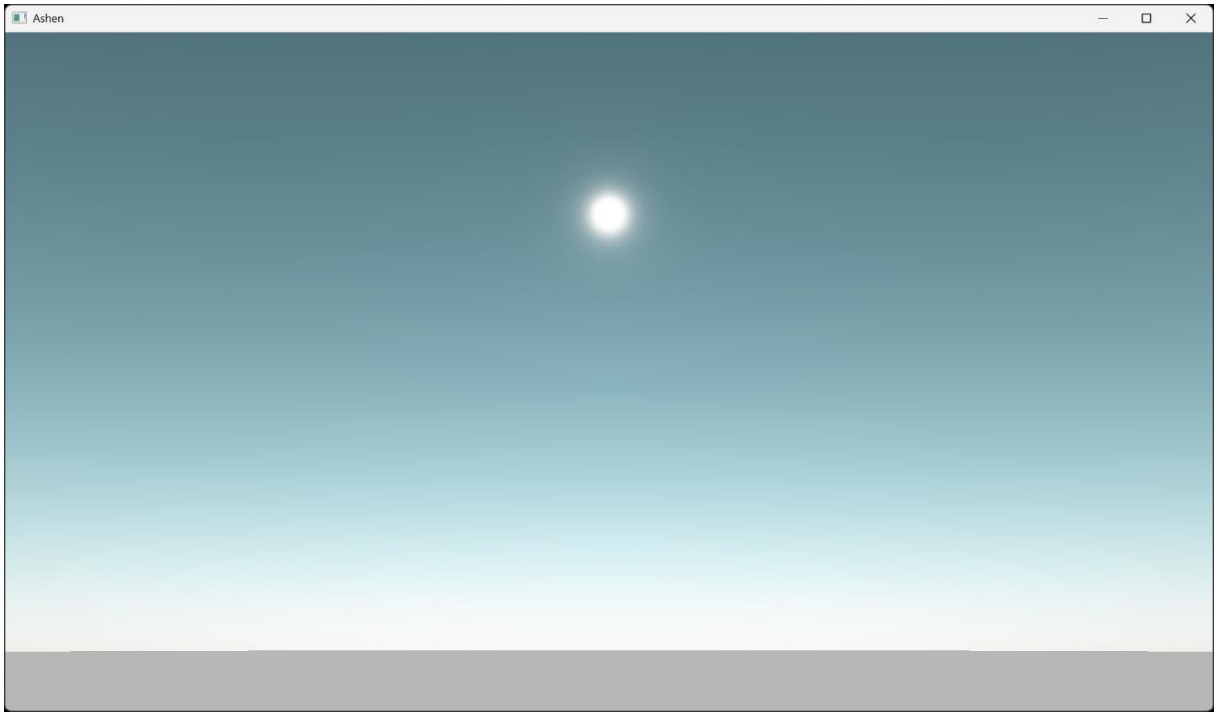

Figure 53: No Ozone | Double Henyey-Greenstein | High Sun Angle

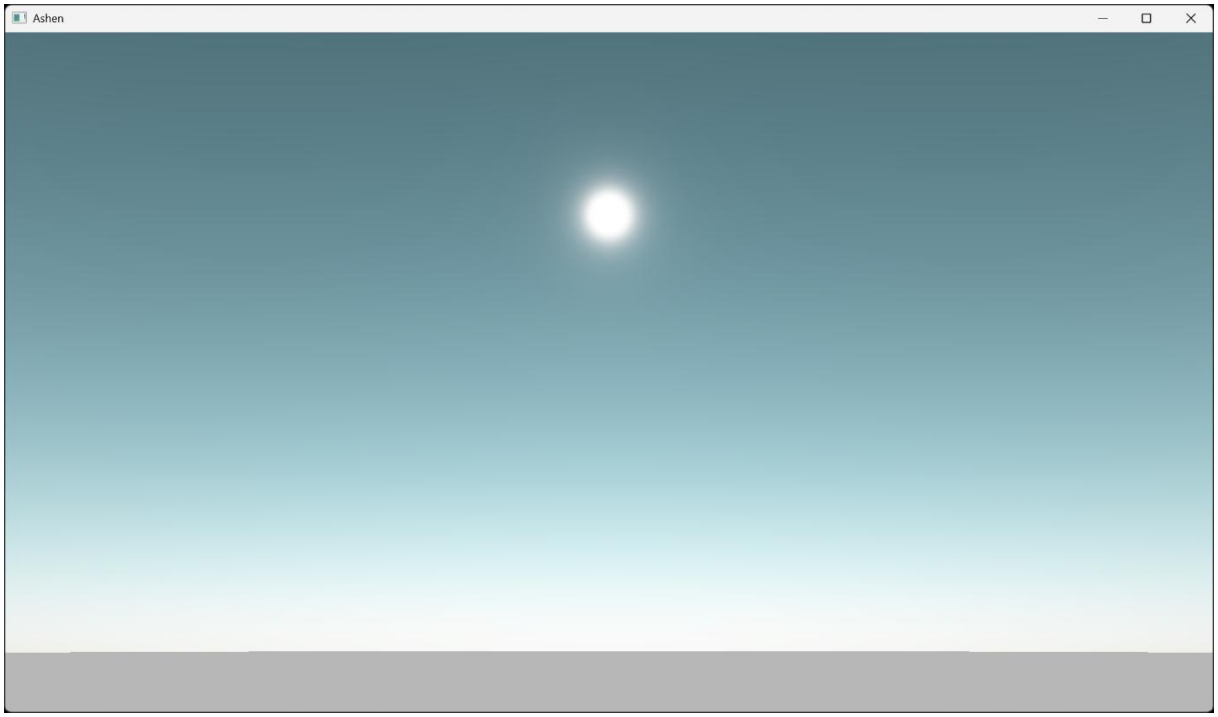

Figure 54: No Ozone | Henyey-Greenstein | High Sun Angle