Efficient and Dynamic Atmospheric Scattering

Master of Science Thesis in Computer Science – Algorithms, Languages and Logic

GUSTAV BODARE
EDVARD SANDBERG

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2014

Efficient and Dynamic Atmospheric Scattering

Gustav Bodare
Edvard Sandberg

Examiner: Ulf Assarsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover: A rendered image displaying the sky and aerial perspective as viewed from the ground.

Department of Computer Science and Engineering
Göteborg, Sweden June 2014

Abstract

This thesis explores the physical background of atmospheric light scattering, as well as the modeling and implementation of a physically based sky rendering system. We build upon existing state of the art research to develop a fast and memory-efficient technique, viable for use in AAA games. Our mathematical model extends previous work by including the contribution of ozone, an important addition for a realistic sky. A new method used for the calculations of aerial perspective is suggested to provide higher quality. By utilizing the high parallelizability of the GPU, we show how precomputed scattering tables can be updated dynamically to provide transitions between atmospheric weather conditions with minimal impact on real time performance.

Acknowledgements

We would like to thank *everyone at Ghost Games* for making us feel welcome, and for giving us the opportunity to work alongside such talent. We would especially like to thank our supervisor *Andreas Brinck*, Sr Software Engineer at Ghost Games, for his continuous support and help throughout the thesis work.

Mikael Uddholm, Rendering Engineer at DICE, for providing guidance and help with integrating the module into the existing engine, and for giving us the opportunity to visit and experience DICE.

Fabien Christin, Lighting Artist at DICE, for testing and providing much appreciated artistic feedback.

Erik Sintorn, our supervisor at Chalmers, for helping with various problems and solutions, and for proof reading our report.

Finally, we want to thank *Ulf Assarsson*, Head of Research at the Chalmers Graphics Department, for inspiring us to pursue a career within the most interesting area of work there is.

Contents

Contents **3**

1 Introduction **5**
1.1 Motivation 5
1.2 Problem Statement 5
1.3 Our work 5

2 Previous Work **6**

3 Atmospheric Scattering **7**
3.1 Physical Basis 7
3.2 Mathematical Model 8
3.2.1 Introduction 8
3.2.2 Scattering Coefficients 9
3.2.3 Phase Functions 10
3.2.4 Density Function 10
3.2.5 Transmittance 11
3.2.6 Single Scattering 12
3.2.7 Multiple Scattering 13

4 Precomputing Scattering **16**
4.1 Overview 16
4.2 Texture Parametrization 17
4.2.1 Height Parametrization 17
4.2.2 View Angle Parametrization 18
4.2.3 Sun Angle Parametrization 19
4.3 Inverse Parametrization 20
4.4 Transmittance 20
4.5 Single Scattering 21
4.6 Multiple Scattering 22
4.7 Optimization 23
4.7.1 Precomputing on the GPU 24
4.7.2 Transmittance 24
4.7.3 Precomputing Gathering 25
4.7.4 Dropping half the LUT 25
4.7.5 Approximating Mie 26

5 Dynamic Atmospheric Conditions **27**
5.1 Theory 27
5.2 Implementation 28

6 Rendering **29**
6.1 Overview 29
6.2 Rendering the Sky 29
6.3 Ground Lighting 30
6.4 Aerial Perspective 31
6.4.1 Previous Method 32
6.4.2 Our Method 34

7 Results **36**
7.1 Textures 36
7.2 Precomputation and Rendering Performance 37
7.3 Visual Results 38
7.4 Validation 41

8 Discussion and Conclusion **43**

9 Future Work **44**

References **45**

A Appendix **46**
A.1 System Overview 46
A.2 Additonal Visual Results 47

1 Introduction

Atmospheric scattering contributes to many everyday phenomena that we experience. It is the reason the sky is blue at noon and red at sunset. It is also the reason why an object far in the distance appears to have lost a portion of its original color and adopted the color of the sky. In short, atmospheric scattering describes what happens with sunlight when it enters the atmosphere.

With this thesis, we intend to continue the work of researchers within the field of atmospheric scattering for computer graphics.

1.1 Motivation

While many older sky and haze rendering techniques involve sky cubes and a simple depth-dependant color modifier, it is difficult to create realistic environments and dynamic time of day transitions using these approaches. Instead, we aim to provide a way of calculating what is actually happening with the light emitted by the sun when it enters the atmosphere of a planet.

This topic has been the subject of study for graphics researchers in the past. However, we believe that there is still room for improvement, not only when it comes to quality, but also in terms of performance. We intend to utilize modern computation techniques, such as compute shaders, to develop a dynamic sky and haze rendering module that provides accurate results in real time, on current generation hardware, while remaining memory efficient.

The module will primarily be targeting current generation gaming consoles and PCs of equivalent hardware, utilizing the strengths of DirectX 11.

1.2 Problem Statement

The purpose of our research is to build upon previous state of the art work on the topic of atmospheric scattering, improving both the quality and the performance, to allow use in AAA games. We want to investigate the possibility of providing dynamic weather transitions, while maintaining the high quality provided by previous research. Furthermore, although the visual results of previous research are already of high quality, we would like to investigate how the quality and realism can be improved even further without sacrificing computation time.

1.3 Our work

In this thesis, we suggest implementation changes and optimizations made to provide dynamic weather transitions; something previous implementations of precomputed atmospheric scattering tables have not supported.

We also provide a higher quality approach of calculating aerial perspective, utilizing a flexible method that could be extended to include scattering in additional types of participating media - such as local fog, clouds and water.

In addition, the often ignored but important presence of ozone in the atmosphere is explicitly included in our model.

2 Previous Work

Over the years, a number of different approaches have been used to solve the problem of atmospheric scattering in computer graphics. This section aims to summerize and compare the numerous previous attempts at solving the problem.

Nishita et al present a model in [NSTN93], that has since been the basis for most computer graphic research, on the topic of atmospheric scattering. By utilizing assumptions that the earth is perfectly spherical and that rays of sunlight can be treated as parallel, they suggest the use of precomputed scattering look-up tables.

In [PSS99], Preetham et al. suggest calculating skylight by numerically solving the scattering integrals for different sun positions and mapping approximative functions to the data. Preetham et al. also suggests an analytic solution to aerial perspective, through various assumptions - such as the earth being a flat surface.

In [O'N05], Sean O'Neil presents a real time approximation of the work by Nishita et al. by analyzing the results from parts of the scattering equations, and creating approximative functions.

In *[SFE07],* Schafhitzel et al. present an approach of precomputing single scattering into a 3D table, parametrized by height, view angle, and sun angle. They also present an approach towards solving aerial perspective, utilizing the same look up tables.

In [BN08], Bruneton and Neyret present an extension to the work by Shafhitzel et al., using four dimensions instead of three for their look-up tables, including the sun-view azimuth in addition to height, view angle and sun angle. Bruneton and Neyret also extends the previous work by including multiple scattering, shadowed scattering, and improved look-up table parametrization.

In [ELE09], Oskar Elek presents an approach much similar to the one by Schafhitzel et al. Through partially adopting the parametrization by Bruneton and Neyret, while still using a 3D table, Elek produces a quick and memory efficient solution. Elek takes various actions to make up for the lack of a fourth dimension in his precomputed tables, such as deferring the phase functions to the fragment shader and deriving an ad hoc Rayleigh phase function that aims to emulate the earth's shadow on the atmosphere.

In [YUS14], Egor Yusov presents an extension to the work by Bruneton and Neyret, further improving look-up table parametrization. Additionally, Yusov suggests calculating the scattering in epipolar space.

3 Atmospheric Scattering

3.1 Physical Basis

This section describes the physical background for how light behaves in an earth-like planetary atmosphere. Like many other graphics applications we assume a clear sky, ignoring light scattering in clouds.

The earth's atmosphere consists of a mixture of many different compounds. Generally these can be divided into two main types - gas and aerosol. Gas refers to air molecules, whereas aerosol consists of larger liquid or solid particles suspended in the gas, such as water droplets, dust, or ice crystals. Considering these compounds are vital, as they directly affect how light behaves in the atmosphere. As a light ray from the sun hits a gas molecule or aerosol particle, a portion of light will deviate from its original path. This is called light scattering, and is the very reason the atmosphere appears colored and enlightened from directions other than the source of light.

Light scattering behaves differently for gas molecules and aerosol particles, allowing us to recognize two types of scattering - Rayleigh and Mie. Rayleigh scattering occurs on gas molecules, or more specifically, on particles fulfilling the following criterion:

$$d \ll \frac{\lambda}{2\pi} \tag{1}$$

where d is the diameter of the particle and λ is the wavelength of the incoming light [ELE09]. As the diameter of the particle grows larger than the wavelength, the scattering smoothly transitions to Mie scattering.

The behavioral difference of Rayleigh and Mie scattering is mainly that the amount of Rayleigh-scattered light depends largely on the wavelength of the incoming light, whereas Mie scattering has a very weak dependence on wavelength. While some CG applications include the wavelength dependency for Mie scattering, many choose to ignore it for simplicity. For Rayleigh scattering, light with shorter wavelength has a tendency to scatter more frequently, generating the blue color of a clear mid-day sky. The orange-red twilight sky can be explained by light passing close to the horizon traveling a greater distance through the atmosphere, making most of the blue light scatter away, while the red light still makes it through.

For a realistic simulation of the earth's atmosphere it is also important to consider the effect of ozone. Even tough ozone does not contribute to scattering, it absorbs a fraction of the incoming light. Furthermore, it is dependent on wavelength, which means it will affect the color of the light that passes through it. Although contribution of ozone is usually ignored in computer graphics sky models, its presence is responsible for an important visual enhancement to our model (see Section 7.3, Figure 20).

When discussing light scattering models, the terms single and multiple scattering are often present. Single scattering depicts the situation where one and only one scattering event has taken place before the light reaches the eye, and accordingly multiple scattering means that the light has scattered several times, and possibly undergone both Rayleigh and Mie scattering.

We also need to introduce the term aerial perspective, also commonly known as haze or distance fog. This phenomena is most noticeable on far away objects which seem to fade away and accumulate a blueish hue the further from the observer they appear. Aerial perspective is very important to consider, as its absence will make objects appear disjoint from the scene, and it allows the observer to better appreciate distance. The reason for the aerial perspective is exactly the same as the reason for the color of the sky. At each point between the observer and the object, light will scatter into the viewing ray, and generate the blue hue. Furthermore, the light from the light source bouncing off the object, creating the original color, will be attenuated whilst traveling towards the observer.

3.2 Mathematical Model

In this chapter, we describe the mathematical model used to calculate the scattering in our system. Such a model can be very complex, especially when taking multiple scattering into account. Thus, certain approximations must be used in order to produce model feasible for use in a computer application. The model is the result of the work initially done by Nishita et al. [NSTN93] and further extended by Oskar Elek to include multiple scattering [ELE09]. We extend this model to include ozone and its contribution to atmospheric scattering.

3.2.1 Introduction

The intensity of light that has undergone scattering in a certain point $\mathbf{p}$, towards a direction $\boldsymbol{v}$, with an incident light direction $\boldsymbol{l}$, can be described by the following equation:

$$I_S(\mathbf{p}, \boldsymbol{v}, \boldsymbol{l}, \lambda) = I_I(\lambda) \left(\rho_R(h) F_R(\theta) \frac{\beta_R(\lambda)}{4\pi} + \rho_M(h) F_M(\theta) \frac{\beta_M(\lambda)}{4\pi} \right) \tag{2}$$

where I_I is the intensity of the incident light, $\rho(h)$ is the density function for height h, above the earth surface, of point $\mathbf{p}$, $F(\theta)$ is the phase function for the angle θ between $\boldsymbol{v}$ and $\boldsymbol{l}$, and $\beta(\lambda)$ is the scattering coefficient.

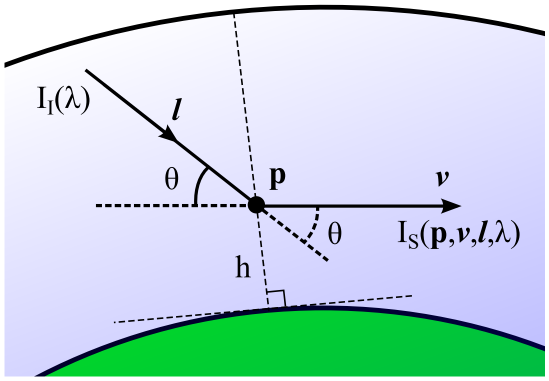

Figure 1: Light traveling in direction $\boldsymbol{l}$, scattering towards direction $\boldsymbol{v}$ in point $\mathbf{p}$.

Throughout this chapter, we will further describe the necessary equations and constants. Please refer to Table 1 for used notation.

Notation	Description
I_I	Intensity of incident light
I_S	Intensity of scattered light
$\beta_{R,M}$	Rayleigh/Mie scattering coefficient
$\beta^e_{R,M}$	Ryaleigh/Mie extinction coefficient
$F_{R,M}$	Rayleigh/Mie phase function
$\rho_{R,M}$	Rayleigh/Mie density function
T	Transmittance function
$I^{(1)}_{S_{R,M}}$	Intensity of Rayleigh/Mie single scattered light
$G^{(k)}_{R,M}$	Gathered Rayleigh/Mie light intensity of order k
$I^{(k)}_{S_{R,M}}$	Intensity of Rayleigh/Mie scattered light of order k

Table 1: Mathematical model notations.

3.2.2 Scattering Coefficients

For all points at sea level, the Rayleigh and Mie scattering coefficients combined with their respective phase function (see Section 3.2.3) describe the proportion of incident light scattered in a certain direction. In our model, the Rayleigh scattering coefficient depends on the density of air molecules and the wavelength of the incident light, while the Mie scattering coefficient only depends on density of aerosol, and not the wavelength. Some models include dependency on wavelength for Mie scattering as well, although the dependency is then much weaker than for Rayleigh[PSS99].

The Rayleigh scattering coefficient is defined as follows:

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

Where n is the refractive index of air (≈ 1.0003), N_e ($molecules \cdot m^{-3}$) is the molecular density of air at sea level, λ is the wavelength (m) of the incident light. As default settings in our system we use $(6.5e{-}7, 5.1e{-}7, 4.75e{-}7)$ for the wavelengths of RGB, $2.545e25$ for N_e, resulting in a scattering coefficient of $\beta_{R_{RGB}} = (6.55e{-}6, 1.73e{-}5, 2.30e{-}5)$.

The Mie scattering coefficient is a bit more complex to calculate. Other models either base it on measured data or approximate it with a formula [PSS99]. The default Mie scattering coefficient in our implementation is, as in Bruneton and Neyret's implementation[BN08], $\beta_{M_{RGB}} = (2e{-}6, 2e{-}6, 2e{-}6)$. Note that the values are identical for all three wavelength, due to the Mie scattering coefficient being independent of wavelength.

3.2.3 Phase Functions

The phase functions for Rayleigh and Mie represent the angular dependency of the scattering coefficients. They are parametrized by θ, which denotes the angle between the direction of the incident light and the direction of the scattered light. The Rayleigh phase function has an almost uniform distribution, while the Mie phase function generally has a strong forward lobe.

The Rayleigh phase function is described by:

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

The Mie phase function is much more complex than the Rayleigh phase function, as the distribution varies heavily depending on the size of the particles. However, Cornette and Shanks[CS92] suggest an improvement of the Henyey-Greenstein function, allowing approximation of the mie phase function for particles with a diameter larger than the wavelength of the light:

$$F_M(\theta, g) = \frac{3(1-g^2)}{2(2+g^2)} \frac{\left(1 + \cos^2(\theta)\right)}{(1 + g^2 - 2g\cos^2(\theta))^{3/2}} \tag{5}$$

where $g \in [-1, 1]$ is an asymmetry factor depicting the width of the forward lobe of Mie scattering.

3.2.4 Density Function

With the previous two functions, we are able to calculate scattering for a single point at sea level. The density function scales the scattering coefficients to an arbitrary height, by representing the change in molecular and aerosol density. We make the assumption that the molecular density decreases with an exponential rate with respect to the height above the earth surface, and define the density function as follows:

$$\rho_{R,M}(h) = \exp\left(-\frac{h}{H_{R,M}}\right) \tag{6}$$

where h is the height above sea level, $H_R \approx 8000m$ and $H_M \approx 1200m$ are the Rayleigh and Mie scale heights, i.e. the thickness of the atmosphere if the molecules and aerosol were uniformly distributed.

By combining the density function with scattering coefficients and phase functions we can now solve equation (2).

3.2.5 Transmittance

We have now defined the amount of light scattered in a certain direction, but we also need to represent how this light is attenuated. A fraction of the light traveling along a certain path through a participating media such as the atmosphere will outscatter, or be absorbed by aerosol. The transmittance function describes the attenuation of light between two points $\mathbf{p_a}$, $\mathbf{p_b}$ inside the atmosphere:

$$T(\mathbf{p_a}, \mathbf{p_b}, \lambda) = \exp\left(-\sum_{i \in R,M} \beta_i^e(\lambda) \int_{\mathbf{p_a}}^{\mathbf{p_b}} \rho_{R,M}(h(\mathbf{p}))d\mathbf{p}\right) \tag{7}$$

where $h(\mathbf{p})$ is the height above the earth surface in point $\mathbf{p}$, and $\beta_{R,M}^e$ is the respective extinction coefficients for Rayleigh and Mie. Since molecules only outscatter light, $\beta_R^e = \beta_R$. However, as aerosol particles both scatter and absorb light, $\beta_M^e = \beta_M + \beta_M^a$, where β_M^a is an absorption coefficient. In our system, we use $\beta_M^e = \frac{\beta_M}{0.9}$ as suggested by Bruneton and Neyret[BN08]. Note that special care has to be taken for cases where the path between $\mathbf{p_a}$ and $\mathbf{p_b}$ intersects the earth, in which case the light is completely attenuated.

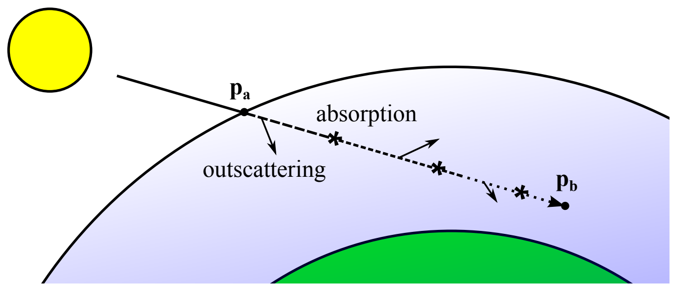

Figure 2: Transmittance from $\mathbf{p_a}$ to $\mathbf{p_b}$.

While this is generally sufficient to produce a realistic looking sky, it does not take the effects of ozone into account. However, it has been shown by Peter Kutz [PK13] that the addition of ozone to the model produces more realistic skies, especially for sunset and sunrise (see Section 7.3, Figure 20).

Ozone does not scatter light, but absorbs a fraction depending on wavelength of the incoming light. This means that ozone only needs to be accounted for in the transmittance equation:

$$T(\mathbf{p_a}, \mathbf{p_b}, \lambda) = \exp\left(- \sum_{i \in R,M,O} \beta_i^e(\lambda) \int_{\mathbf{p_a}}^{\mathbf{p_b}} \rho_{R,M,O}(h(\mathbf{p}))d\mathbf{p} \right) \tag{8}$$

where β_O^e is the ozone extinction coefficient, and the ozone density function $\rho_O = 6e{-}7 * \rho_R$. While ozone, in reality, does not decrease exponentially with altitude - the mentioned equation for ρ_O approximates the density by using the average overall concentration of ozone relative to air molecules [NASA09].

As shown by Kutz, β_O^e can be acquired using values from Figure 3.

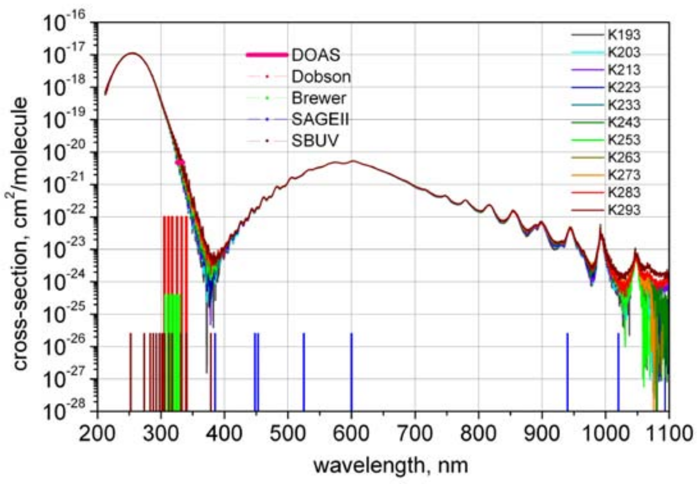

Figure 3: Ozone absorption cross section for different wavelengths[UBR].

Note that the cross section values of Figure 3 are in $cm^2/molecule$, and other values of our model are in m. Thus, all that is required to convert these values into absorption coefficient is to simply scale the values into $m^2/molecule$.

3.2.6 Single Scattering

We now have everything we need to compute the intensity of light scattering towards, and reaching, an observer. To evaluate single scattering - light that has undergone one and only one scattering event before reaching the observer - we define the following equation:

$$I^{(1)}_{S_{R,M}}(\mathbf{p_o},\boldsymbol{v},\boldsymbol{l},\lambda) = I_I(\lambda)F_{R,M}(\theta)\frac{\beta_{R,M}(\lambda)}{4\pi}\int\limits_{\mathbf{p_a}}^{\mathbf{p_b}}\rho_{R,M}(h(\mathbf{p}))T(\mathbf{p_c},\mathbf{p},\lambda)T(\mathbf{p},\mathbf{p_a},\lambda)d\mathbf{p} \tag{9}$$

where $\mathbf{p_a}$ is the point where a ray from the observer position, $\mathbf{p_o}$, enters the atmosphere in the viewing direction, $\mathbf{p_b}$is the point where the same ray exits the atmosphere or intersects the ground. $\mathbf{p_c}$ is the point where the incident light enters the atmosphere before reaching point $\mathbf{p}$, the point at which light scatters into the viewing ray. $\boldsymbol{v}$ is the direction from point $\mathbf{p}$ to point $\mathbf{p_a}$, i.e. the inverse viewing direction, and $\boldsymbol{l}$ is the incident light direction from the light source. Note that if the observer is situated inside the atmosphere, $\mathbf{p_a} = \mathbf{p_o}$.

Worth mentioning is that the incident light direction $\boldsymbol{l}$ is in fact slightly different between $\mathbf{p_a}$ and all the other points along the ray. However, due to the vast distance to the sun, we make the assumption that all light rays can be treated as parallel.

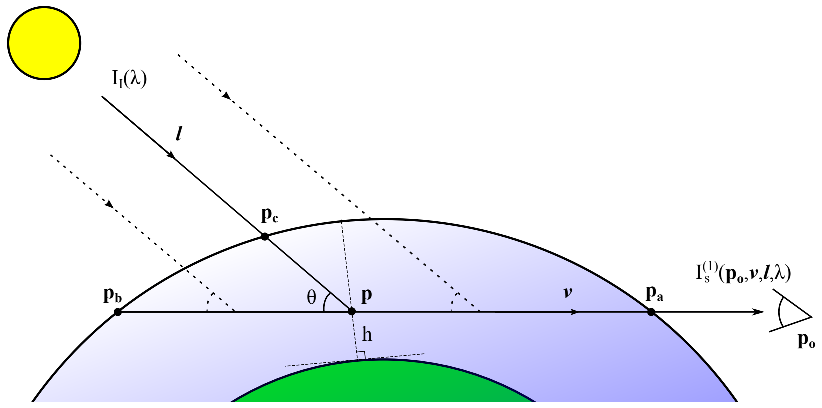

Figure 4: Single scattering towards $\mathbf{p_o}$ between $\mathbf{p_a}$ and $\mathbf{p_b}$.

To acquire the total intesntiy of the light scattered into direction $\boldsymbol{v}$ and reaching the observer, we simply sum the results of Rayleigh and Mie single scattering.

$$I^{(1)}_{S_R} + I^{(1)}_{S_M} = I^{(1)}_S \tag{10}$$

3.2.7 Multiple Scattering

The most straightforward solution to computing multiple scattering would be to obtain the higher scattering orders through nested recursive multidimensional integrals, which would be very slow. Instead, we will make use of data from previously calculated scattering orders, allowing us to use an iterative algorithm rather than the aforementioned recursive method. We define a function expressing the amount of light of scattering order k reaching a certain point $\mathbf{p}$, and scattering in direction $\boldsymbol{v}$:

$$G^{(k)}_{R,M}(\mathbf{p},\boldsymbol{v},\boldsymbol{l},\lambda) = \int\limits_{4\pi} F_{R,M}(\theta) I^{(k)}_{R,M}(\mathbf{p},\boldsymbol{\omega},\boldsymbol{l},\lambda) d\boldsymbol{\omega} \tag{11}$$

where θ is the angle between $\boldsymbol{v}$ and $-\boldsymbol{\omega}$.

For example, the gathering function for order one would be to integrate single scattering over all directions and apply the phase function. Thus, the gathering function of order k expresses the light gathered in point $\mathbf{p}$, where the light has undergone exactly k scattering events.

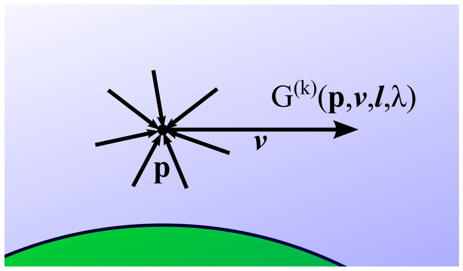

Figure 5: Light of order k scattering towards $\boldsymbol{v}$.

Using the gathering function, we can express the intensity of light reaching the observer of scattering order k. We define the function for light of order k reaching the observer as follows:

$$I^{(k)}_{S_{R,M}}(\mathbf{p_o},\boldsymbol{v},\boldsymbol{l},\lambda) = \frac{\beta_{R,M}(\lambda)}{4\pi} \int\limits_{\mathbf{p_a}}^{\mathbf{p_b}} G^{(k-1)}_{R,M}(\mathbf{p},\boldsymbol{v},\boldsymbol{l},\lambda)\rho_{R,M}(h(\mathbf{p}))T(\mathbf{p},\mathbf{p_a},\lambda)d\mathbf{p} \tag{12}$$

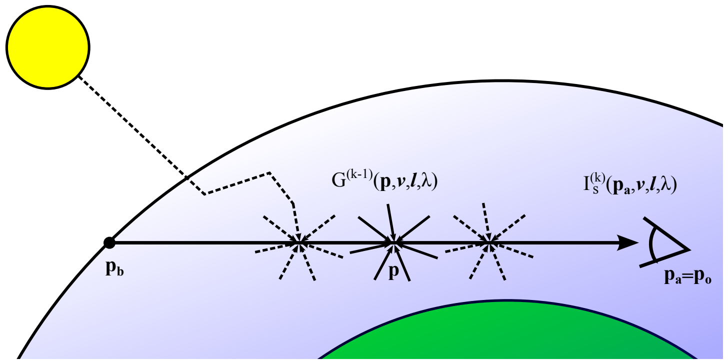
$G^{(k-1)}(\mathbf{p},\boldsymbol{v},\boldsymbol{l},\lambda)$
$I_S^{(k)}(\mathbf{p_a},\boldsymbol{v},\boldsymbol{l},\lambda)$
$\mathbf{p_b}$
$\mathbf{p}$
$\mathbf{p_a}=\mathbf{p_o}$

Figure 6: Light of order k scattering towards $\mathbf{p_o}$ between $\mathbf{p_a}$ and $\mathbf{p_b}$.

Again, when the observer is situated inside the atmosphere $\mathbf{p_a} = \mathbf{p_o}$. As with single scattering, the total intensity is acquired by summing Rayleigh and Mie.

$$I_S^{(k)} = I_{S_R}^{(k)} + I_{S_M}^{(k)} \tag{13}$$

Finally, to calculate all light reaching the observer after K scattering orders, all we need to do is sum the results from single scattering and each higher order scattering.

$$I_S = \sum_{i=1}^{K} I_S^{(i)} \tag{14}$$

4 Precomputing Scattering

In this chapter, we aim to describe how the mathematical model can be utilized to compute and store atmospheric scattering reaching the observer in a way that is efficient and feasible to implement in computer software. We show a procedural way of creating scattering look-up tables, represented as textures, and how this procedure can be optimized.

4.1 Overview

Providing scattering information for every observer position $\mathbf{p}[x, y, z]$, every view direction $\boldsymbol{v}[x, y, z]$ and every light source direction $\boldsymbol{l}[x, y, z]$ would require a look-up table with nine dimensions. This would not only cause the texture to become huge, but would also require an infeasible amount of time to compute the scattering.

In order to reduce the problem into a more viable one, we adopt the assumptions made by Bruneton and Neyret[BN08];

1. The planet can be treated as perfectly spherical.
2. The density of particles in the air change only with height, and does not depend on geographical location.
3. All light that reaches the atmosphere can be treated as parallel, due to the vast distance between the earth and the sun.

By making these assumptions, the problem is reduced to four degrees of freedom; the height, the sun-zenith angle, the view-zenith angle and the sun-view azimuth angle. Elek[ELE09] suggests the removal of the sun-view azimuth, further reducing the problem to only three degrees of freedom, by making the assumption that we can model the azimuth dependent difference in hue using a modified Rayleigh phase function:

$$F_R(\theta) = \frac{8}{10}\left(\frac{7}{5} + \frac{1}{2}\cos(\theta)\right) \tag{15}$$

The motivation behind this decision is that the sun-view azimuth mainly contributes to the earth's shadowing of the atmosphere, a phenomena barely visible due to multiple scattering and often hidden by terrain. This will also cause the sky to appear identical for all horizontal viewing angles, but deferring the evaluation of the phase functions to the pixel shader solves this. The deferral of the phase functions also removes artifacts that would otherwise appear as blockiness in the Mie scattering around the sun.

By reducing our look-up tables to three dimensions, both the texture sizes and the time required to precompute the scattering tables are greatly reduced. Figure 7 elucidates the three parameters used for our look-up tables; height, view-zenith angle and sun-zenith angle.

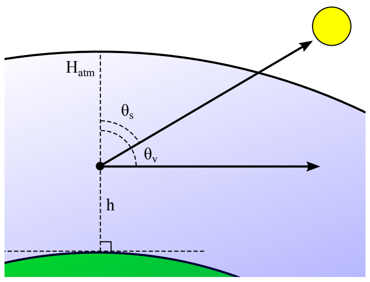

Figure 7: Precomputation parameters, sun-zenith angle θ_s, view-zenith angle θ_v, and the height h.

4.2 Texture Parametrization

In order to fetch values from our 3D textures, we need a way of translating the sun angle, view angle and height into the range $[0, 1]$. For this purpose, a function of the form

$$f : (h \in [0, H_{atm}], \theta_v \in [0, \pi], \theta_s \in [0, \pi]) \rightarrow (u_h \in [0, 1], u_v \in [0, 1], u_s \in [0, 1])$$

is required to map our values into texture coordinates. The most straightforward implementation of such a function would be to use a linear parametrization:

- $u_h = \frac{h}{H_{atm}}$
- $u_v = \frac{cos(\theta_v)+1}{2}$
- $u_s = \frac{cos(\theta_s)+1}{2}$

While this is sufficient if the texture resolution is large enough, Bruneton and Neyret [BN08] shows that better precision can be achieved, even with a smaller texture, by improving the parametrization. As the sky differs mostly in hue at view angles near the horizon, the parametrization must provide higher precision in these regions. The atmosphere is also denser near the ground, and the density then decreases with height, meaning that the height parametrization must focus low heights. Furthermore, it is unnecessary to fill the textures with data where the scattering is occluded by the earth and does not reach the observer. The remapping functions suggested by Bruneton and Neyret have since been further improved by Yusov [YUS14].

In the following sections we use the notation $c_s = \cos(\theta_s)$ and $c_v = \cos(\theta_v)$.

4.2.1 Height Parametrization

Because the density of the atmosphere exponentially decreases with altitude, the difference in sky color is largest at low altitudes. This means that the height parametrization should be focusing these low altitudes. This is achieved by using the following function:

$$u_h = \left(\frac{h}{H_{Atm}}\right)^{0.5} \tag{16}$$

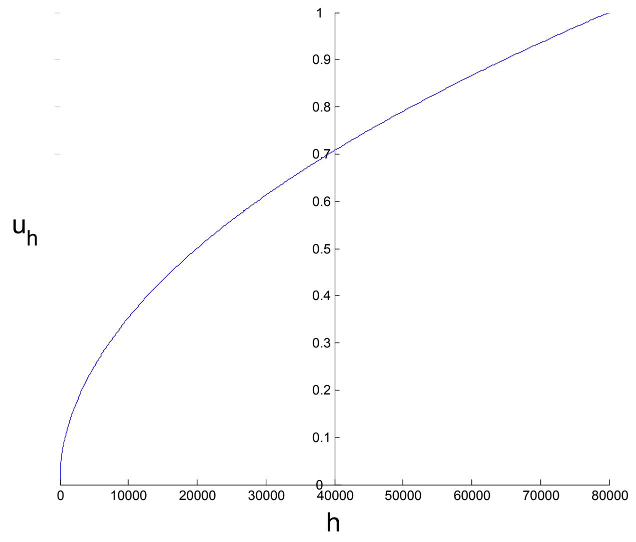

Figure 8: Height parametrization for $h \in [0, H_{atm} = 80000]$

4.2.2 View Angle Parametrization

As mentioned, the difference in hue is at its largest near the horizon. The suggested view parametrization solves this by calculating the angle between the horizon and zenith for the current height of the observer, and provides higher precision for similar angles.

$$u_v = \begin{cases} 0.5\left(\frac{c_v - c_h}{1 - c_h}\right)^{0.2} + 0.5, \ c_v > c_h \\ 0.5\left(\frac{c_h - c_v}{c_h + 1}\right)^{0.2}, \ c_v \leq c_h \end{cases} \qquad where\ c_h = -\frac{\sqrt{h(2R_{Earth} + h)}}{R_{Earth} + h} \tag{17}$$

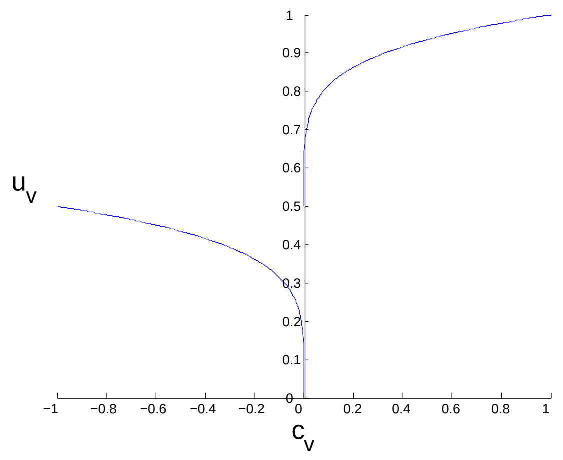

Figure 9: View angle parametrization for $c_v \in [-1, 1]$

4.2.3 Sun Angle Parametrization

While the importance of the direction of the sun does not change drastically during daytime, we do not need to include sun angles below a certain threshold in our table because during night time the inscattered light from the sun is almost nonexistent.

$$u_s = 0.5(\frac{\tan^{-1}(\max(c_s, -0.1975)\tan(1.26 \cdot 1.1))}{1.1} + (1 - 0.26)) \qquad (18)$$

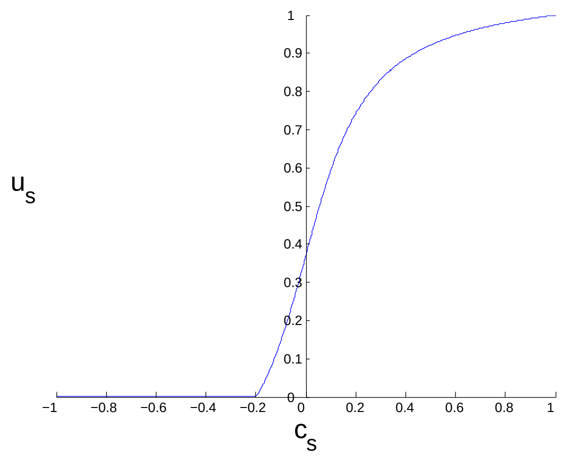

Figure 10: Sun angle parametrization for $c_s \in [-1, 1]$.

4.3 Inverse Parametrization

When filling the look-up tables, a function that maps the coordinate of the given texture cell into an actual combination of viewing direction, sun direction and height is required.

$$f : (u_h \in [0,1], u_v \in [0,1], u_s \in [0,1]) \rightarrow (h \in [0, H_{atm}], \theta_v \in [0,\pi], \theta_s \in [0,\pi])$$

Finding these values can be done by simply using the inverse of the parametrizations (Equations 16, 17, 18), but because it can be quite complicated to evaluate and calculate these inverse functions by hand - we provide the inverse parametrization, as suggested by Yusov [YUS14] and Bruneton and Neyret[BN08], below:

- $h = u_h^2 H_{atm}$
- $c_v = \begin{cases} c_h + (u_v - 0,5)^5(1 - c_h), \ u_v > 0.5 \\ c_h - u_v^5(1 + c_h), \ u_v \leq 0.5 \end{cases} \quad where \ c_h = -\frac{\sqrt{h(2R_{Earth}+h)}}{R_{Earth}+h}$
- $c_s = \frac{\tan((2u_s - 1 + 0.26)0.75)}{\tan(1.26 * 0.75)}$

4.4 Transmittance

The transmittance calculations are a major part of solving the single and multiple scattering integrals. This section aims to present the most basic approach of algorithmically solving the transmittance integral. Bruneton and Neyret suggest, in their implementation[EVA08], the use of an analytically derived solution to the integral. However, during testing we noticed that while this algorithm is often sufficient, it fails to provide accurate data in many cases where the relation between planet radius and atmosphere radius is unlike that of earth.

Due to this, and for additional reasons described in Section 4.7.2, we are instead solving the transmittance integral using trapezoidal numerical integration. While the algorithm itself is simple to implement using this approach, it is provided below as it is referenced in following chapters.

Algorithm 1 Transmittance between $\mathbf{p_a}$ and $\mathbf{p_b}$

```
stepSize = distance($\mathbf{p_a}$, $\mathbf{p_b}$)/INTEGRATION_STEPS
$\boldsymbol{dir}$ = normalize($\mathbf{p_b} - \mathbf{p_a}$)
for step=0; step < INTEGRATION_STEPS ; ++step
   s = $\mathbf{p_a}$ + stepSize*step*$\boldsymbol{dir}$
   currentDensityMie = getDensityMie(height(s))
   currentDensityRayleigh = getDensityRayleigh(height(s))
   totalDensityMie += (currentDensityMie+
      previousDensityMie)/2*stepSize
   totalDensityRayleigh += (currentDensityRayleigh+
      previousDensityRayleigh)/2*stepSize
   previousDensityMie = currentDensityMie
   previousDensityRayleigh = currentDensityRayleigh
end loop
transmittance = exp(-
   (totalDensityRayleigh*$\beta^e_{R_{RGB}}$ +
   totalDensityMie*$\beta^e_M$))
```

In the following two sections, 4.5 and 4.6, and their respective algorithms, this algorithm will be referenced using the following syntax:

`transmittance(`$\mathbf{p_x}$`, `$\mathbf{p_y}$`)`

4.5 Single Scattering

In order to calculate single scattering and store the result in a 3D texture, we iterate each cell of the texture and use the previously described inverse parametrization (see Section 4.3) to translate the cell coordinate into view direction, sun direction and height. Trapezoidal integration is then used to calculate the amount of single scattering along the viewing ray, for the given height and sun direction.

Algorithm 2 describes the numerical integration for single scattering.

Algorithm 2 Single Scattering reaching $\mathbf{p_a}$ from direction $\boldsymbol{v}$ with incident light direction $\boldsymbol{l}$. Recall that phase functions are deferred to the pixel shader.

```
$\mathbf{p_b}$ = intersection($\mathbf{p_a}$, $-\boldsymbol{v}$, earthRadius, atmosphereRadius)
stepSize = distance($\mathbf{p_a}$, $\mathbf{p_b}$)/INTEGRATION_STEPS
for step=0 ; step < INTEGRATION_STEPS ; ++step
   $\mathbf{p}$ = $\mathbf{p_a}$ + stepSize*step*$-\boldsymbol{v}$
   transmittance = transmittance($\mathbf{p_a}$, $\mathbf{p}$)
   $\mathbf{p_c}$ = intersection($\mathbf{p}$, $-\boldsymbol{l}$, earthRadius, atmosphereRadius)
   transmittance *= transmittance($\mathbf{p}$, $\mathbf{p_c}$)
   currentInscatteringMie =
      getDensityMie(height($\mathbf{p}$))*transmittance
   currentInscatteringRayleigh =
      getDensityRayleigh(height($\mathbf{p}$))*transmittance
   totalInscatteringMie +=
      (currentInscatteringMie+previousInscatteringMie)
      /2*stepSize
   totalInscatteringRayleigh +=
      (currentInscatteringRayleigh+previousInscatteringRayleigh)
      /2*stepSize
   previousInscatteringMie = currentInscatteringMie
   previousInscatteringRayleigh = currentInscatteringRayleigh
end loop
totalInscatteringMie *= $\beta_M$/(4*$\pi$)*$I_{I_{RGB}}$
totalInscatteringRayleigh *= $\beta_{R_{RGB}}$/(4*$\pi$)*$I_{I_{RGB}}$
```

4.6 Multiple Scattering

Thanks to Equations 11 and 12, it is possible to calculate each order of multiple scattering iteratively. Again, the first step required is to convert from a texture coordinate to view angle, sun angle and height, using the inverse parametrization functions. Trapezoidal numerical integration is then used to integrate along the view direction of the observer. The main difference between single and multiple scattering is that in the case of multiple scattering, the sun is not used as a light source. Instead, for each integration point, we gather the amount of incoming light, from the previous scattering order, that scatters in direction $\boldsymbol{v}$.

We will first define an algorithm used to compute part of Equation 11 ($G_{R,M}^{(k-1)}$), the amount of incoming light, from the previous scattering order, in a certain point.

Algorithm 3 Multiple Scattering Gathering, gatheredLight($\mathbf{p}$, $\boldsymbol{v}$, $\boldsymbol{l}$)

```
for $\theta_v$=0 ; $\theta_v$<$2\pi$ ; $\theta_v$+=($2\pi$/INTEGRATION_STEPS) loop
   gathered += fetchScattering(height($\mathbf{p}$), $\theta_v$, $\theta_s$)
end for
gathered *= $4\pi$/INTEGRATION_STEPS
```

Note that, while the gathering function is an integral over a sphere, we only integrate over the view angles stored in our look up tables. The above algorithm is, as mentioned, part of the multiple scattering integral. Algorithm 4 describes how to calculate scattering of order k (Equation 12, $I_S^{(k)}$).

Algorithm 4 Multiple Scattering reaching point $\mathbf{p_a}$ from direction direction $\boldsymbol{v}$ with incident light direction $\boldsymbol{l}$

```
$\mathbf{p_b}$ = intersection($\mathbf{p_a}$, $-\boldsymbol{v}$, earthRadius, atmosphereRadius)
stepSize = distance($\mathbf{p_a}$, $\mathbf{p_b}$)/INTEGRATION_STEPS
for step=0 ; step < INTEGRATION_STEPS ; ++step
   $\mathbf{p}$ = $\mathbf{p_a}$ + stepSize*step*$-\boldsymbol{v}$
   transmittance = transmittance($\mathbf{p_a}$, $\mathbf{p}$)
   currentInscatteringMie =
      gatheredLight($\mathbf{p}$, $\boldsymbol{v}$, $\boldsymbol{l}$)*
      getDensityMie(height($\mathbf{p}$))*transmittance
   currentInscatteringRayleigh =
      gatheredLight($\mathbf{p}$, $\boldsymbol{v}$, $\boldsymbol{l}$)*
      getDensityRayleigh(height($\mathbf{p}$))*transmittance
   totalInscatteringMie += (currentInscatteringMie+
      previousInscatteringMie)/2*stepSize
   totalInscatteringRayleigh += (currentInscatteringRayleigh+
      previousInscatteringRayleigh)/2*stepSize
   previousInscatteringRayleigh = currentInscatteringRayleigh
end loop
totalInscatteringMie *= $\beta_M$/($4\pi$)
totalInscatteringRayleigh *= $\beta_{R_{RGB}}$/($4\pi$)
```

In accordance with Equation 14, when the desired number of scattering have been computed, single scattering and each higher order scattering are summed, creating the final 3D look-up table.

4.7 Optimization

In order to reduce the time required to precompute our scattering look-up textures we have taken multiple actions, of varying importance. With the following section we intend to describe the most important optimizations and improvements used in our implementation. We will not go into detail about standard shader optimization tricks, but will instead focus on changes to the structure and flow that greatly reduce the time needed to solve the scattering integrals.

4.7.1 Precomputing on the GPU

The algorithms used to calculate the values of each cell in the scattering textures are identical. Theoretically, the only difference between cells is the input; the observer height $\mathbf{p_o}$, direction $\boldsymbol{v}$ and direction $\boldsymbol{l}$. Since the inverse parametrization described in Section 4.3 allows us to convert texture coordinates into the aforementioned parameters, a GPU implementation is a very suitable choice. We utilize the vast amount of processing units on the GPU to parallelize the procedure, instead of iteratively computing each texture cell on the CPU.

We suggest using one thread for each cell in the textures, finding the thread ID and thus also the texture coordinate via the thread ID variable. It is possible, and preferable, to use compute shaders for all precomputations needed.

4.7.2 Transmittance

One of the most time consuming operations in the scattering integrals for a basic implementation is calculating transmittance. For example, as described by Algorithm 2, transmittance must be computed not only between the observer and the point, but also from the point to the atmosphere boundary in the direction towards the sun. Both of these computations are heavily optimizable, using the following two approaches.

Calculating transmittance between observer and integration point Instead of using the algorithm described in Section 4.5, we can utilize the fact that the scattering integrals integrate over the same ray that is used by the transmittance. Thanks to this, we can calculate the transmittance between the observer and the given point while we calculate the scattering. All that is needed is to solve, and store the result of, the density integral up to the current point. When transmittance is needed, simply calculate according to Algorithm 5.

Algorithm 5 Calculating transmittance from density

```
    transmittance = exp(-($\beta^e_M$*mieParticleDensitySum
      + $\beta^e_{R_{RGB}}$*rayleighParticleDensitySum)
```

Precomputing transmittance to the atmosphere Transmittance between the integral point and the atmosphere can not use the approach described in the previous section. However, it is possible to optimize these calculations almost as much, using a separate precomputed transmittance texture. Transmittance is precomputed using only two degrees of freedom - height and sun-zenith angle. The process of filling this texture is very straightforward.

The integration is performed using the steps described in Algorithm 6.

Algorithm 6 Precomputing transmittance to the atmosphere

```
1) For each texture cell, find the viewing direction and
height using inverse parametrization.
2) Integrate along viewing ray from the corresponding height.
   2.1) For each integration step, calculate the particle
   density of Mie and Rayleigh.
3) Calculate exp(-($\beta^e_M$*mieParticleDensitySum+
$\beta^e_{R_{RGB}}$*rayleighParticleDensitySum)).
4) Save the result in the texture
```

The transmittance table uses the parametrization described in Section 4.2 for view angle and height.

4.7.3 Precomputing Gathering

The process of calculating multiple scattering, described in Section 4.6, is slow and repetitive. The fact that integration is done over many view directions for every height and all sun directions means that many very similar gathering calculations are computed. With this information, we can understand that it would be possible to precompute this data into an auxiliary look-up texture, parametrized only by height and sun direction. This process is identical to Algorithm 3, but with the advantage of using much fewer inputs. Henceforth, this look-up table will be referenced to as the gathering LUT.

While calculating the gathering LUT, we store two different values in two different textures. The first one being the gathered light scattering from the previous scattering order, i.e. we are doing a pre-pass for scattering of order k, which means that this texture will contain light scattered from the $k-1$ order into the given point. The second texture contains the sum of the gathering for all scattering orders, up to and including the current order. This second LUT is later used to include multiple scattering in aerial perspective (see Section 6.4).

Algorithm 7 describes the new process for calculating scattering of order k, replacing the use of Algorithm 3 in Algorithm 4.

Algorithm 7 Multiple Scattering using the Gathering LUT

```
gathreedRayleigh = fetchRayleigh(height, sunAngle)
gatheredMie = fetchMie(height, sunAngle)
```

By only sacrificing minimal precision, this optimization heavily improves the performance (see Section 7.2).

4.7.4 Dropping half the LUT

As a result of the view angle parametrization, the scattering look-up table is divided into two parts - view angles above and below the horizon. Previous implementations of atmospheric scattering tables only use the lower half of the texture for calculations of aerial perspective and multiple scattering. Because we suggest a new way of calculating aerial perspective (see Section 6.4), we no longer have any use for the lower half of the texture when the precomputations for multiple scattering are completed.

By making use of this fact, it is possible to disregard calculating values for the lower half of the texture during the last scattering order. Thus, making the precomputations roughly 13% faster when using four scattering orders[1]. Furthermore, this allows us to improve the performance of the parametrization during rendering, again due to the fact that we no longer need the lower half of the texture.

4.7.5 Approximating Mie

As shown by Bruneton and Neyret[BN08], it is possible to reduce the amount of data required to store inscattering, by approximating Mie_{RGB} using only Mie_R in combination with $Rayleigh_{RGB}$. This means that it is possible to store all scattering information in one look-up table, rather than having to use one for Rayleigh and one for Mie, by storing the red value of Mie scattering in the alpha channel of the Rayleigh look-up texture.

The formula used to retrieve the Mie scattering is then defined as follows:

$$Mie_{RGB} = Rayleigh_{RGB} \frac{Rayleigh_A}{Rayleigh_R} \frac{\beta_{R_R}}{\beta_{M_R}} \frac{\beta_M}{\beta_R} \tag{19}$$

As can be seen on the Mie scattering around the sun in Figure 11, the approximation works very well and the results are almost identical.

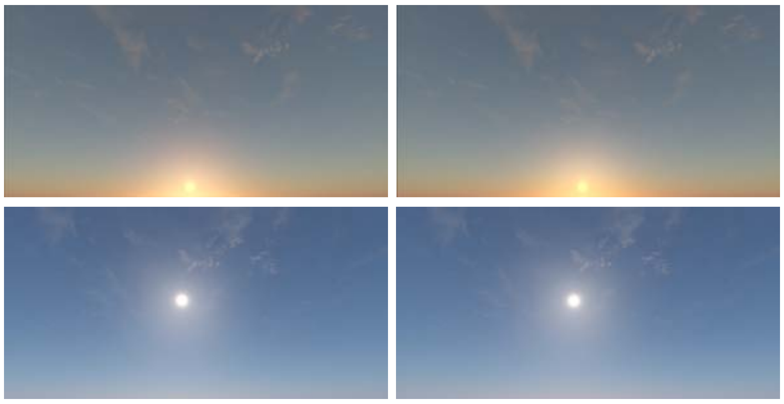

Figure 11: Left column: Mie and Rayleigh scattering stored separately. Right column: Mie approximated from $Rayleigh_{RGB}$ and Mie_R.

[1]Each scattering order takes about the same time. I.e. each scattering order takes 25% of the time

5 Dynamic Atmospheric Conditions

A very important limitation of the previous work on the topic of precomputed atmospheric scattering is that neither of the previously suggested methods provide the option to enable dynamic atmospheric conditions. We have tried to solve this problem, and this chapter will describe our method.

5.1 Theory

In order to provide dynamic weather transitions in the atmosphere, we must change the properties of the atmosphere. In order to represent this change in our final scene, we must update the scattering tables. While our optimizations, combined with the constantly improving hardware of current generation computers, allow us to recompute the scattering tables each frame while still providing interactive frame rates (see Section 7.2), it is simply not justifiable to spend that much time each frame on something as small as weather transitions in the atmosphere for AAA games.

In reality, weather conditions do not change instantly from perfectly clear to foggy. Taking this into consideration, we rewrote our system to provide dynamic update of the look-up tables over multiple frames. In order to distribute the workload evenly over frames, we investigated the amount of time required for each part of the precomputations. We found that each scattering order, whether it is single scattering or multiple scattering, takes about the same amount of time. We also found that the transmittance precomputations are almost instantaneous compared with the time required for scattering orders, and that the precomputation of the gathering LUT takes roughly five percent of the time required to complete one scattering order.

In order to make our system as dynamic as possible, without sacrificing too much useability, we wanted to provide an option for setting the amount of frames to be used, with certain restrictions due to ease of implementation. To do this, and while taking the timing of the various stages of precomputing into consideration, we developed the following formula:

$$F = ((K - 1) * X) + X/2 + (K - 1)$$

where F is the total amount of frames that the system should be precomputing over, K is the number of scattering orders and X is a variable free for the user to choose with the restriction of $X \in \{2, 4, 8, 16, 32\}$.

This means that we divide each of the first $K - 1$ scattering orders into X amount of frames, add $X/2$ frames for the final scattering order (due to the optimization described in Section 4.7.4), and then add one frame for each gathering table that needs to be precomputed. The transmittance precomputations are run during the first frame of each precomputation cycle, together with the first part of the single scattering.

5.2 Implementation

Simply implementing a system according to the theory above would not be sufficient to providing nice looking weather transitions. The biggest problem occurs because we are updating the texture used for rendering part by part. This means that when half of the new single scattering calculations have been computed, half the texture will contain data of only single scattering for the new atmospheric conditions while the other half will contain multiple scattering data calculated for the previous conditions of the atmosphere.

In order to solve this, we changed our system to contain two different final textures. We then toggle which of these two textures is used for precomputations, and let the other one be used for rendering. This nicely solves the problem of updating parts of the texture while it is being used for rendering. However, one problem still remains. By considering a system where we divide the work over 115 frames ($X = 32$), according to the formula described above, it would take almost two seconds[2] to update the look-up tables. In certain extreme cases, it is possible that the change in the precomputed scattering tables is large enough to be noticed by the user. In order to solve this problem we implemented a system, working in parallel with the precomputations, that interpolates between the previous and the second to previous atmospheric scattering tables during the precomputation of the first $(K - 1)$ scattering orders.

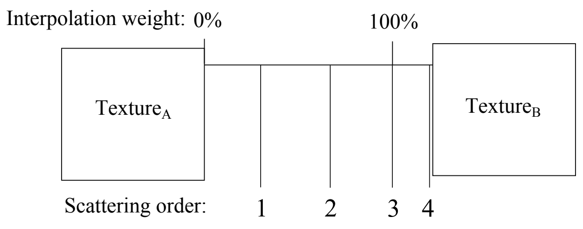

Figure 12: Interpolation run in parallel to precomputations

Simply put, the first three scattering orders in the above example are stored in temporary textures. When the fourth, and last, scattering order is about to be precomputed, the interpolation between texture A and texture B has reached the case where the resulting texture is 100% of texture B. After this, the precomputations store the new atmospheric scattering data in texture A, and no interpolation occurs until texture A has been filled with its new data. When the precomputations are completed, we interpolate in the other direction, and the process repeats itself if the atmospheric weather conditions are not up to date.

The process of interpolating is used for the scattering tables, as well as for the extra textures required by rendering in order to provide fast and fully dynamic weather transitions.

[2]Assuming that the application runs with 60 frames per second.

6 Rendering

This chapter will describe the sky module rendering procedure, and how it uses the precomputed data stored in look-up tables. We first show how to render the sky, from any observer position, followed by the rendering of the ground and aerial perspective.

6.1 Overview

In our system, the rendering is done by creating a quad located at the far plane of the camera frustum and drawing all scattering to it, while using a depth texture, containing all previously rendered geometry, to render the aerial perspective. Depth testing and drawing to the depth-buffer is turned off when rendering the atmospheric scattering to preserve the depth data of the scene. Furthermore, the view vector for each pixel is acquired by interpolating the vectors from the near plane to the far plane of the frustum, for each corner. The zenith vector can be found by subtracting the camera position with the position of the earth center. The light source vector is not dependent on the camera position due to Assumption 3 in Section 4.1 (in which we state that all light rays are treated as parallel) and thereby only depends on relative positions of the earth and the sun.

Note that while a global tone mapping is needed to correct over-saturation, our report does not cover this area.

6.2 Rendering the Sky

To render the sky, we simply need to fetch the Rayleigh- and Mie-scattered light from the look-up texture, multiply it with the intensity of the incident light, and apply their respective phase functions.

Fetching from the texture is performed by calculating the texture coordinates using the parametrization provided in Section 4.2. Since the look-up texture is parametrized by $\cos(\theta_s)$, $\cos(\theta_v)$, and h, we need to find these to use as input for the remapping functions. Both $\cos(\theta_s)$ and $\cos(\theta_v)$ can easily be calculated by the dot product of the zenith vector, and the view and sun vector respectively. The height h is calculated by the distance from the earth center to the camera position, subtracted by the planet radius. Special care has to be taken when the camera is outside the atmosphere. In this situation the camera position has to be moved to the closest intersection between the viewing ray and the atmosphere, before the parametrizations are computed. Note that if the camera is inside the atmosphere, we know that the sun direction and height of the observer are constant during the rendering of each pixel in the current pass. This means that the parametrization for these variables can be calculated on the CPU and sent directly to the shader.

It is important to also consider the fact that the look-up table only contains light that has scattered, meaning we also have to render direct sunlight; direct light that has only had a portion of its intensity attenuated along its path to the observer. A simple way to do this is to render a disc, where the color is calculated by the intensity of the incident light multiplied with the transmittance along the path from the point where the light enters the atmosphere, to the camera position.

$$I_S^{(0)}(\mathbf{p_o}, \boldsymbol{l}) = I_I(\lambda)T(\mathbf{p_o}, \mathbf{p_c}) \tag{20}$$

where $\mathbf{p_o}$ is the position of the observer, $\boldsymbol{l}$ is the direction to the light source, and $\mathbf{p_c}$ is the intersection with atmosphere boundary in the direction $\boldsymbol{l}$ from the camera position $\mathbf{p_o}$. The transmittance between $\mathbf{p_o}$ and $\mathbf{p}_c$ can be obtained by fetching from the precomputed transmittance table, at the height of $\mathbf{p_o}$ in direction $\boldsymbol{l}$.

Even though our system does not include light scattering in clouds, fairly realistic clouds can be rendered with simple cloud layer textures. By taking advantage of the attenuation of the direct light reaching the clouds, the same way the direct light reaching the observer is calculated, we can achieve the reddish hue on the clouds during a sunset (see Figure 13).

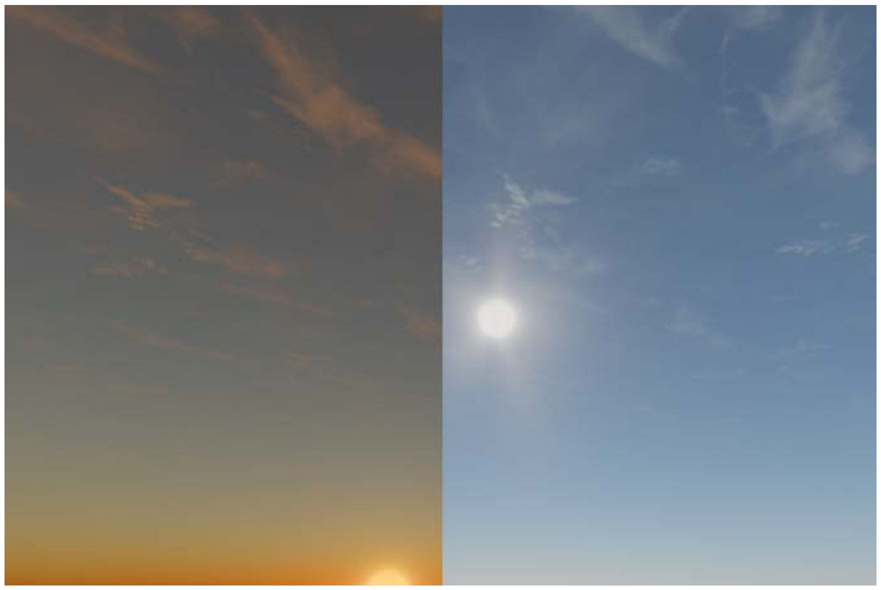

Figure 13: Simple cloud layer lit by direct sunlight

6.3 Ground Lighting

Lighting on the ground mainly consists of two parts - direct sunlight and ambient sky illumination. The light reaching a certain point on the earth surface will also be affected by transmittance, as it is reflected off the surface and travels towards the observer (see Figure 14).

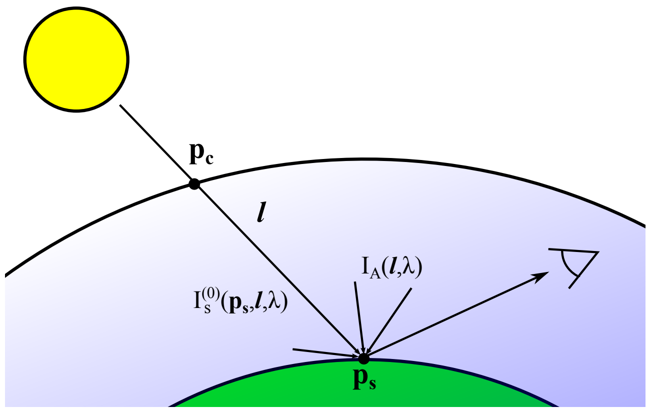
$\mathbf{p_c}$
$\boldsymbol{l}$
$I_A(\boldsymbol{l},\lambda)$
$I_S^{(0)}(\mathbf{p_s},\boldsymbol{l},\lambda)$
$\mathbf{p_s}$

Figure 14: Direct sunlight and ambient sky illumination reaching the camera.

However, the attenuation of light between an object and the observer can be deferred to the rendering of aerial perspective as shown in Section 6.4.2. An added bonus of this is that light from other sources than the sun, reflected off a surface, will also be attenuated by the atmosphere. Direct sunlight can then be calculated with Equation 20, by replacing the observer position $\mathbf{p_o}$ with the surface position $\mathbf{p_s}$.

Oskar Elek[EK10] shows that the ambient sky illumination can be precomputed into a 1D look-up table by fetching from the main 3D scattering texture. The function for ambient light reaching a certain point $\mathbf{p}$ on the ground, when the sun is in direction $\boldsymbol{l}$, is defined as follows:

$$I_A(\mathbf{p}, \boldsymbol{l}, \lambda) = \int_{2\pi} (\boldsymbol{n} \bullet \boldsymbol{\omega}) I_S(\mathbf{p}, \boldsymbol{\omega}, \boldsymbol{l}, \lambda) d\boldsymbol{\omega} \tag{21}$$

where $\boldsymbol{n}$ is the surface normal. Since Elek assumes a perfectly spherical earth, and thereby ignoring terrain, the function is sufficient to calculate the ambient light at ground level. However, in our case, we want to be able to calculate the ambient light for arbitrary heights, as well as for any surface normal, which could be done by adding dimensions to the look-up table. The system in which we implemented our sky rendering module already had support for calculating ambient light for any scene, which allowed us to ignore this. Had that not been the case, adding an extra dimension for height to the look-up table would hopefully have been sufficient.

6.4 Aerial Perspective

Aerial perspective consists of two core parts. Firstly, the color of the underlying object must be attenuated according to the outscattering, the transmittance, between camera and object. Secondly, we must add the color corresponding to the total amount of inscattering between the observer and the object. Therefore, we must calculate T_{as} as well as I_{as}, the transmittance and inscattering between the observer and the object in question.

6.4.1 Previous Method

Previous research [SFE07] suggest calculating Aerial Perspective using the same look-up table as for sky rendering. To understand the process of doing this, we first express the inscattered light reaching the observer (which is stored in our look-up table) as follows:

$$I_{ab}T_{ab} = I_{sb}T_{ab} + I_{as}T_{as} \tag{22}$$

I_{ab} represents light scattering towards the observer, between the observer and the atmosphere (or earth) behind the object, T_{ab} is the transmittance between the same two points. I_{sb} is the light scattering towards the observer between the object and the atmosphere behind it. I_{as} is the light scattering towards the observer between the observer and the object, and finally T_{as} is the transmittance between observer and object.

What is important to note is that I_{sb} is multiplied with T_{ab}, and not just T_{sb}. This is because the light scattering towards the observer between the object and the atmosphere must also be attenuated on its path from the object to the observer.

Equation 22 can be rewritten to:

$$I_{as}T_{as} = I_{ab}T_{ab} - I_{sb}T_{ab} \tag{23}$$

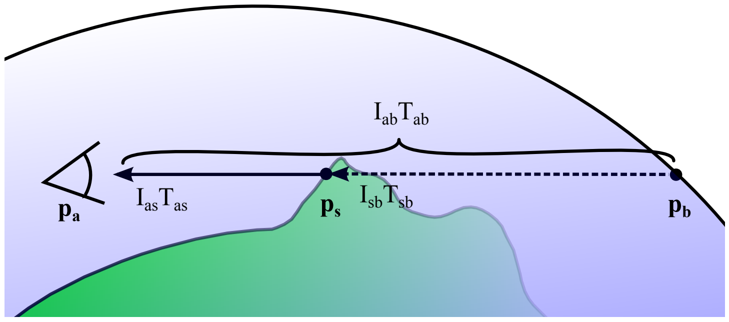

Figure 15: Aerial Perspective between $\mathbf{p_a}$ and $\mathbf{p_s}$.

By using Equation 23, we can now define an algorithm (Algorithm 8) to calculate inscattering between the observer and the object, using only values from our precomputed look-up tables:

Algorithm 8 Aerial Perspective

```
1. Fetch inscattering from the observer to the atmosphere
or earth behind the object, $I_{ab}T_{ab}$.
2. Move the camera to the object, and fetch inscattering
from the object to the atmosphere or earth behind it, in
the same viewing direction, $I_{sb} * T_{sb}$.
3. Calculate transmittance from the observer to the object,
and multiply that with $I_{sb} *$
$T_{sb}$ to get $I_{sb}T_{ab}$ ($T_{as}$ can be found
either by real-time integration to the object, or by using
a look-up table containing transmittance to the atmosphere
or ground, $T_{as} = T_{ab}/T_{sb}$).
4. Use Equation 23 to calculate $I_{as}T_{as}$.
```

In theory, this sounds like a good way of solving the problem of aerial perspective. In practice, however, we came across many small problems with this approach that made it less than ideal. The worst problem was a discontinuity at the horizon, as seen in Figure 16. Bruneton and Neyret do not solve this problem in their paper[BN08], but suggest a fix in their source code[EVA08].

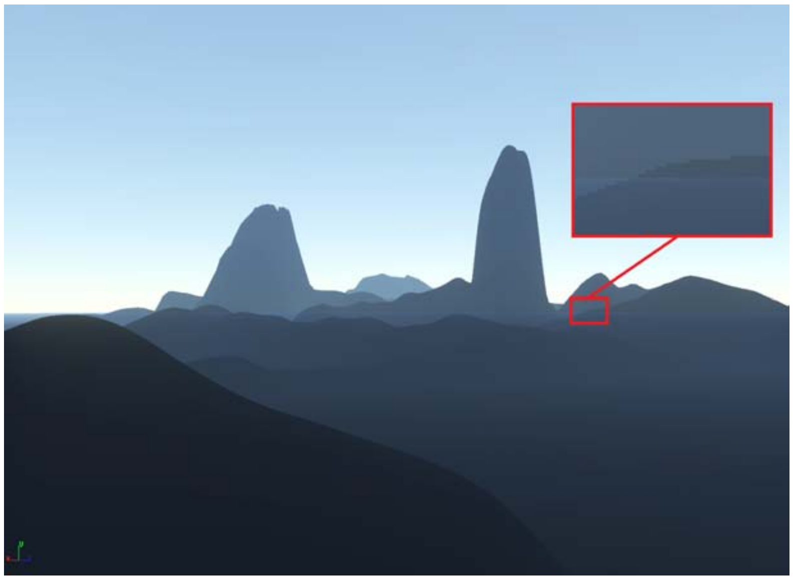

Figure 16: Aerial Perspective inscattering, with artifacts due to the discontinuity at the horizon.

The fix suggested is to interpolate each value, within a certain threshold angle, above the horizon with a value below the horizon in order to smoothen the discontinuity. However, even by doing so, the results are not as good and accurate as can be wished for, and the performance suffers from the extra work.

6.4.2 Our Method

Instead, we started to investigate other options. Our first attempt was to simply sacrifice multiple scattering and integrate single scattering between the viewer and the object in the pixel shader. Even though the visual result was satisfactory, this method proved, as expected, to be very expensive even with just a few integration steps.

Finally, we came up with an idea to build two low resolution 3D textures from the camera frustum, and use a compute shader to fill each cell with the amount of inscattering or transmittance from the center of the cell to the eye. This is done every frame, as a pre-pass to the rendering, but still turned out to be a viable option performance wise due to the low resolution. When rendering the aerial perspective, we simply fetch the scattering, $I_{as}T_{as}$, from the scattering texture for the given position $\mathbf{p_s}$ within the frustum. The already existing color of the object is attenuated according to the transmittance, T_{as}, contained in the second texture. To find the position of the object within the frustum, we use a depth buffer containing all previously rendered geometry. Furthermore, the multiple scattering is included by adding a texture, containing the sum of all orders of gathered light, to the precomputation procedure (see Appendix A.1 for a full system overview). For each cell in the frustum texture, we fetch the gathered sum for the given height and sun direction, and include as incoming light in the scattering calculations.

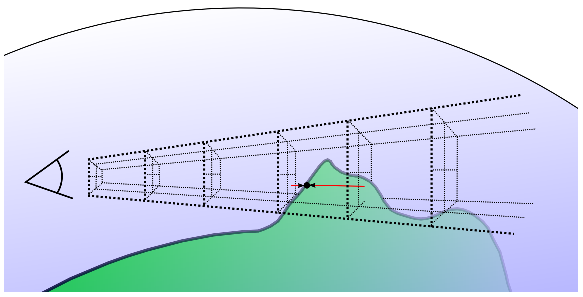

Figure 17: Our method of computing aerial perspective, using a low resolution 3D texture mapped to the camera frustum.

The process of filling these textures is described in Algorithm 9.

Algorithm 9 Aerial Perspective - Our method

```
1. Dispatch a compute shader, with $Resolution_X *$
$Resolution_Y$ threads.
2. For each thread, use the frustum to calculate a viewing
direction.
3. Calculate single scattering and transmittance along the
aforementioned direction, using numerical integration with
$Resolution_Z$ steps.
   3.1 After each step, store the intermediate result in
   the corresponding texture cell.
   3.2 Multiple scatterng is included by fetching from a
   texture containing the sum all orders of gathered light.
```

This proved very successful, and the visual results were excellent. As shown in Section 7.3, they are at least on par with integrating in real time, and in some situations even better, because we can afford a much higher number of integration steps.

Another benefit of this approach is that we no longer need the half of the precomputed table that contains inscattering from viewing directions below the horizon, as mentioned in Section 4.7.4.

7 Results

The following chapter will go into detail about the visual results as well as the performance of our system. To express the system's viability in games targeting current generation hardware, all precomputation and rendering times are measured on Xbox One.

7.1 Textures

As suggested by Elek[ELE09], our main scattering lookup texture has a precision of 32x128x32 for height/view-angle/sun-angle, with 16 bit floats for each channel. Thanks to our method of computing the aerial perspective each frame, we could drop half that look-up table and thereby use a texture with dimensions 32x64x32 (~0.5MB) for the rendering pass. This is considerably less than the 128x128x128 (~16MB) texture used by Shafhitzel et. a. - thanks to the improved texture parametrizations proposed by Bruneton and Neyret[BN08], and further improved by Yusov[YUS14]. However, due to our aerial perspective method, we also require the 3D textures mapped to the camera frustum, representing the inscattering and transmittance. Furthermore, the aerial perspective pre-pass also needs the gathering sum texture to include multiple scattering. We found that these textures produce satisfying results with dimensions 32x32x16 for the aerial perspective and transmittance, and 32x32 for the gathering sum texture (with a combined size of ~0.28MB). Lastly, our rendering pass uses the transmittance to atmosphere texture to calculate the zero-scattered light (direct sunlight), with the dimensions 32x128 (~0.031MB).

In the case of dynamic weather, the aforementioned values need to be doubled, as we use two of each texture, to interpolate between. An exception, however, is the aerial perspective and transmittance from camera textures. Furthermore, since we recompute scattering tables over several frames, a few additional and temporary textures are required during the precomputation process. Table 2 lists the total amount of textures used for rendering, when using dynamic weather transitions, as well as their dimensions and sizes.

Texture	Dimensions	Memory
Transmittance interpolation target A	32x128	32kB
Transmittance interpolation target B	32x128	32kB
Scattering interpolation target A	32x64x32	512kB
Scattering interpolation target B	32x64x32	512kB
Gathering sum interpolation target A	32x32	8kB
Gathering sum interpolation target B	32x32	8kB
Aerial Perspective	32x32x16	128kB
Transmittance from camera	32x32x16	128kB
Total		~1.33MB

Table 2: Textures used for rendering.

7.2 Precomputation and Rendering Performance

In our first attempt, we decided to perform the precomputations on the CPU. This proved to be feasible, but by using roughly one minute to complete the process of precomputing - it by far exceeded our limitations regarding the possibility of dynamically recomputing the look-up table in real time. Worth mentioning is that, at this stage, our method was very straight forward, and did not utilize any of the possible optimizations explained in Section 4.7.

When moving the precomputations to the GPU, the total time required drastically dropped to about 300ms. By using the optimizations previously described, this was further reduced to about 3ms. The largest time gain comes from the use of auxiliary gathering textures (see section 4.7.3). Please refer to Figure 18 for timings of the whole precomputation process for four scattering orders.

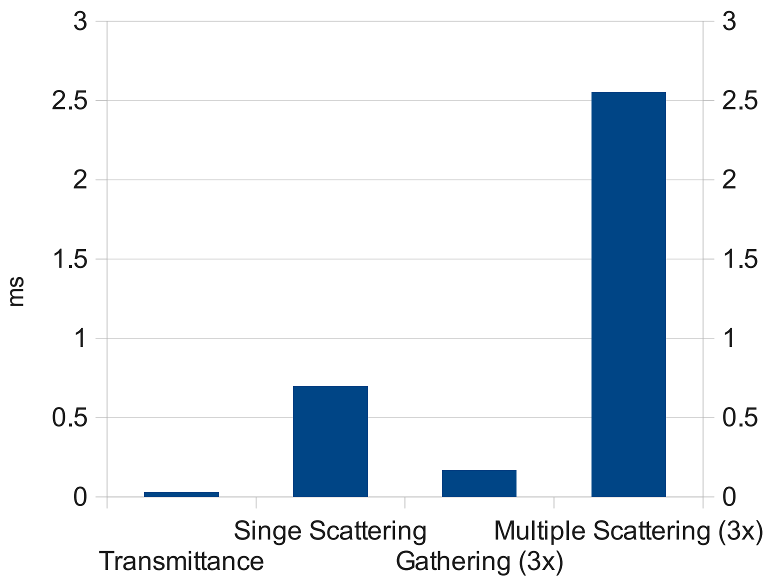

Figure 18: Transmittance: 0.03ms, Single Scattering: 0.7ms, Gathering: 0.057ms per scattering order, Multiple Scattering: 0.85ms per scattering order. Total time of about 3.45ms.

To change the weather dynamically, we found that by recomputing the scattering tables over 115 frames, the real-time performance of the system remained virtually unaffected. The time required for computations each frame never exceeded 0.1ms.

By rendering the scattering to a single quad located at the far plane of the camera frustum (as explained in Section 6.1), the sky module is in no way limited by the amount of vertices in the scene. Even though our method for calculating the aerial perspective requires a separate compute shader pre-pass, the real time performance does not suffer greatly, as the pre-pass is executed over only 0.03ms. Figure 19 contains timing comparisons for the different parts of the rendering pass.

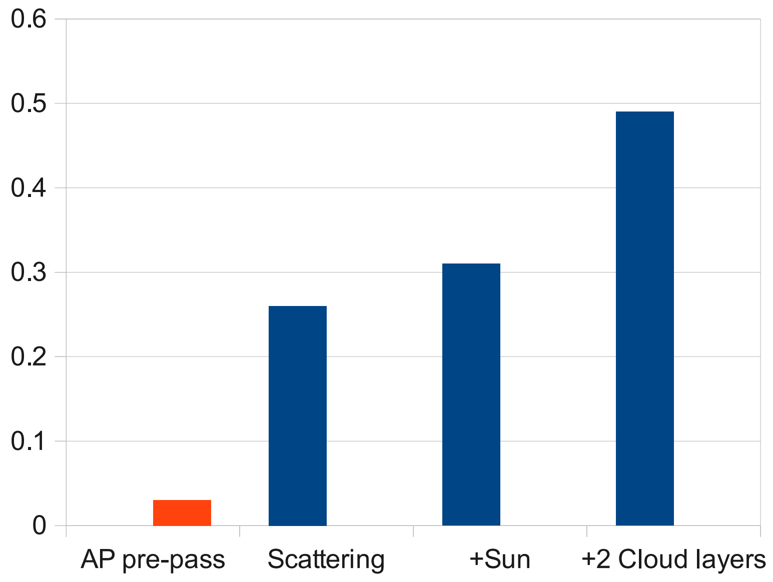

Figure 19: Average rendering times for scattering only, scattering and direct sunlight, and scattering with direct sunlight and two cloud layers. Also, the aerial perspective pre-pass.

Worth noting is that, while the performance numbers used in this section are measured on Xbox One, we do not utilize the ESRAM - something that would improve performance even further.

7.3 Visual Results

The system is capable of producing realistic looking skies. We found that by including ozone we could create far more realistic colors of the sky, compared to previous research where the contribution of ozone is ignored, especially during sunsets (see Figure 20).

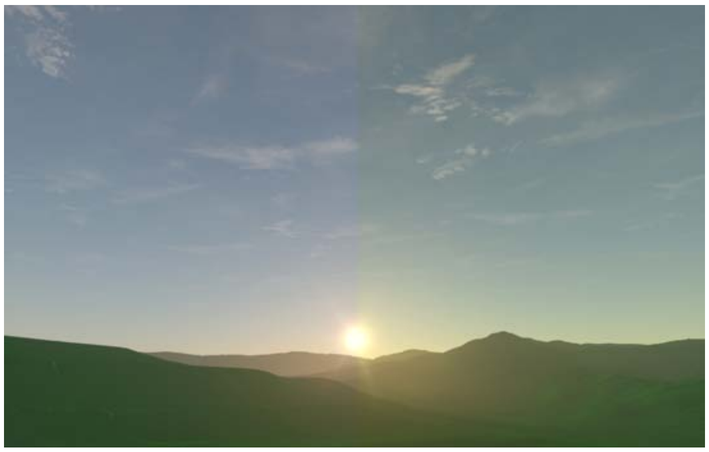

Figure 20: Left: Sunset with ozone. Right: Sunset without ozone.

The system can also produce otherworldly skies. For instance, with a minor performance loss, we could include a second light source, and thereby simulate a hypothetical alien planet. Thanks to the fact that the look-up tables contain scattering values without the sun intensity, the only additional work is to perform texture look-ups for the second light direction, and multiply with its intensity.

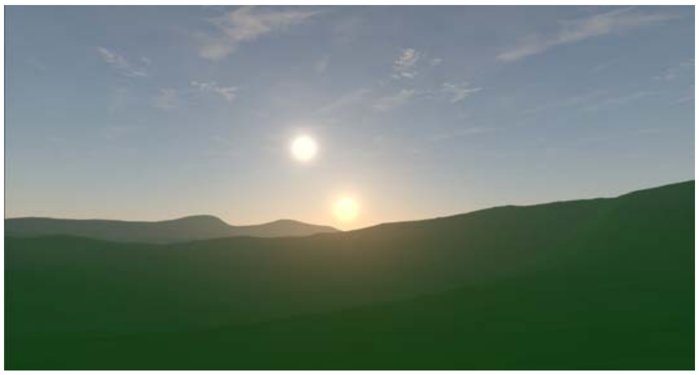

Figure 21: Multiple light sources contributing to the scattering in the atmosphere.

Furthermore, the atmospheric conditions can be changed to simulate other planets, as shown in Figure 22.

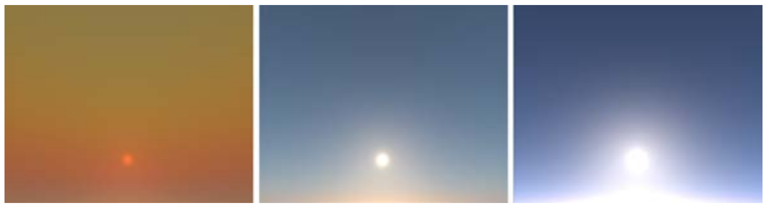

Figure 22: Left: 10 times higher molecular density. Middle: Earth-like atmosphere. Right: 10 times lower molecular density.

As can be seen in Figure 23, the visual precision of inscattered light, calculated with our aerial perspective method, does not suffer from the problems described in Section 6.4.1, and the result is at least on par with real time numerical integration.

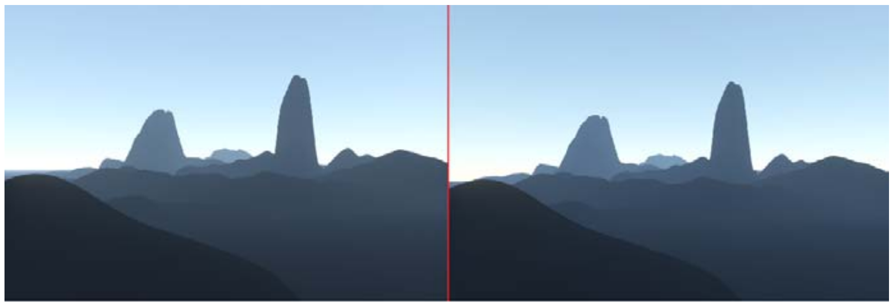

Figure 23: Aerial perspective, inscattering only. Left: Our method (multiple scattering excluded). Right: Numerical integration in the pixel shader, using 4 samples.

Calculating the ground lighting as described in Section 6.3 adds to the nice transition between different times of day, as seen in Figure 24.

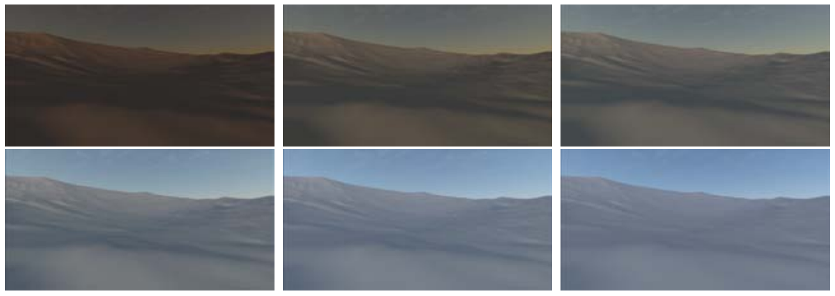

Figure 24: From left to right and top to bottom, ground lighting for $\theta_s = 90°, 88°, 86°, 82°, 75°, 65°$.

Additional images rendered with our system can be found in Appendix A.2 - Additonal Visual Results.

7.4 Validation

Bruneton and Neyret validate the results of their implementation by plotting the sky luminance for all view angles, where the sun-view azimuth is zero[BN08]. They then compare their resulting graph with measured data preovided by CIE. Because we want to validate our implementation against both measured data, but also against previously implemented approaches, we embrace this approach. Figure 25 displays the result of the measurements in our model, and compares them to the same measured data, as used by Bruneton and Neyret[BN08], provided by CIE[CIE04].

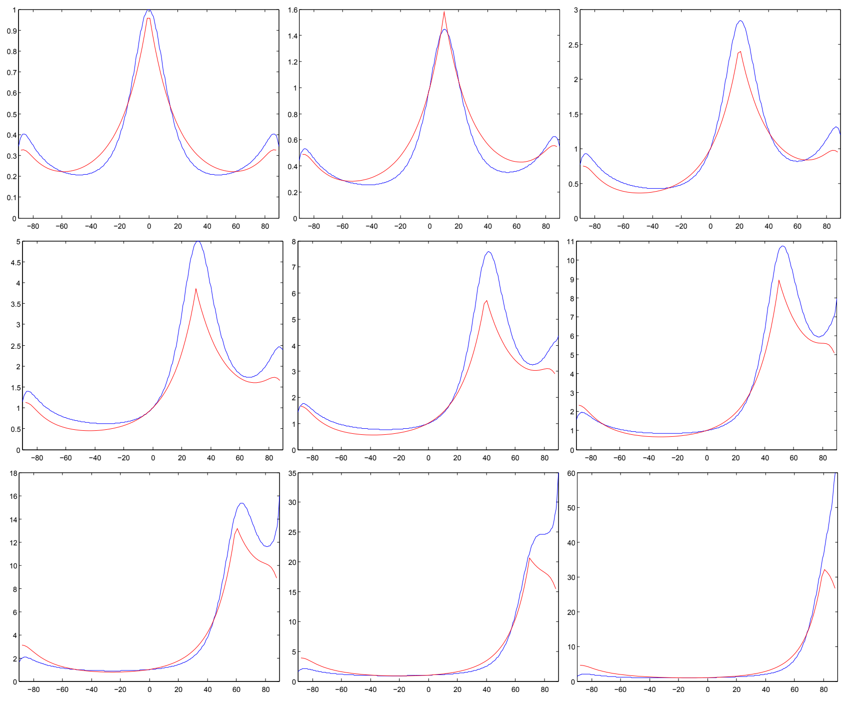

Figure 25: Left to right, and top to bottom: θ_s = $0°$, $10°$, $20°$, $30°$, $40°$, $50°$, $60°$, $70°$, $80°$. Red line: CIE sky model. Blue line: Our model.

As Bruneton and Neyret, we use $\beta_M = 2.2e^{-5}$, $\beta_M^e = \frac{\beta_M}{0.9}$, $g = 0.73$ and $H_M = 1200m$. Because Bruneton and Neyret provide the same measurements for their implementation[BN08], it is possible for us to compare our results with theirs, as well as the CIE model.

We can see an over approximation, although smaller than in Bruneton and

Neyret's data, close to the horizon. As shown by Zotti et al.[ZWP07] the Preetham sky model also suffers from this problem. It is also worth noting that the modified Rayleigh phase function (Equation 15), derived by Elek, provides fairly good results, although slightly underapproximating the scattering at the opposite side of the sun for lower sunangles. Even though the values used for our measurements are adopted from Bruneton and Neyret, in order to compare our results with theirs more precisely, we believe that it is possible to match the CIE model more closely by careful selection of atmospheric properties.

8 Discussion and Conclusion

We have presented several extensions to previous research, including the addition of ozone to the mathematical model and an efficient way of computing artifact free aerial perspective. While the addition of ozone increases the realism of the rendered atmosphere, it is possible that the level of realism could be even further increased by performing fully spectral calculations[EK10].

Additionally, the utilization of the current generation rendering pipeline, combined with many optimizations, allows the system to recompute the scattering look-up tables with virtually no impact on real time performance, and thereby provide seamless transitions between any atmospheric conditions.

While using a 3D texture requires some approximations, aesthetically pleasing results are achieved with a reduced cost. Multiple scattering suffers from the lack of a fourth dimension, as the phase function cannot be applied correctly. Instead, a modified phase function is used on the combined light from single and multiple scattering. If increased precision is desired, the system could easily be extended to utilize a 4D look-up table, in which case the precomputation performance would suffer while the rendering performance would only be marginally impaired. When using 4D tables, the phase function of each multiple scattering order could be included during the precomputations, and the single scattering could be stored in a separate table and still use deferred phase functions for precision reasons. Worth mentioning is that our aerial perspective method would still be perfectly viable, and preferable, as it already includes the sun-view azimuth during single scattering calculations, and the multiple scattering could also be included by increasing the dimensions of the gathering LUT to include a scattering direction.

9 Future Work

We believe that the area that could benefit the most from future research is the aerial perspective computations, in which we would like to calculate additional effects, such as local fog and scattering in clouds. It would also be possible to extend these calculations by including and utilizing shadow volumes in order to provide physically based volumetric light shafts.

Another area that we believe to require future work is the area of parametrization. While the currently used functions provide excellent results, they are slow to evaluate. This is especially important for the parametrization used by the real time shader.

References

[NSTN93] Tomoyuki Nishita, Takao Sirai, Tadamura Katsumi, Nakamae Eihachiro. “Display of the Earth taking into account atmospheric scattering”. In SIGGRAPH 93.

[PSS99] A. J. Preetham, Peter Shirley, Brian Smits. “A Practical Analytic Model for Daylight”. In SIGGRAPH 99.

[O’N05] Sean O’Neil. “Accurate Atmospheric Scattering”. In GPU Gems 2.

[SFE07] Tobias Schafhitzel, Martin Falk, Thomas Ertl. “Real-Time Rendering of Planets with Atmospheres”. In Journal of WSCG, Volume 15.

[BN08] Eric Bruneton, Fabrice Neyret. “Precomputed Atmospheric Scattering”. Eurographics Symposium on Rendering 2008, Volume 27, Number 4, pp. 1079–1086.

[ELE09] Oskar Elek. “Rendering Parametrizable Planetary Atmospheres with Multiple Scattering in Real-Time”.

[YUS14] Egor Yusov. “Outdoor Light Scattering Sample Update”. URL: https://software.intel.com/en-us/blogs/2013/09/19/otdoor-light-scattering-sample-update

[EK10] Oskar Elek, Petr Kmoch. “Real-Time Spectral Scattering in Large-Scale Natural Participating Media”.

[CS92] William M. Cornette, Joseph G. Shanks. “Physical reasonable analytic expression for the single-scattering phase function”. Applied Optics Volume 31, Issue 16, pp. 3152-3160.

[PK13] Peter Kutz. URL: http://skyrenderer.blogspot.com

[NASA09] URL: http://ozonewatch.gsfc.nasa.gov/facts/ozone.html

[UBR] URL: http://www.iup.physik.uni-bremen.de/gruppen/molspec/databases/referencespectra/o3spectra2011/

[CIE04] International Commission on Illumination (CIE). “Spatial distribution of daylight”. In CIE standard general sky, second edition.

[ZWP07] Georg Zotti, Alexander Wilkie, Werner Purgathofer. “A Critical Review of the Preetham Skylight Model”. In WSCG 2007 Short Communications Proceedings I, pp. 23–30.

[EVA08] Eric Bruneton. URL: http://www-evasion.imag.fr/Membres/Eric.Bruneton/index.en.html

A Appendix

A.1 System Overview

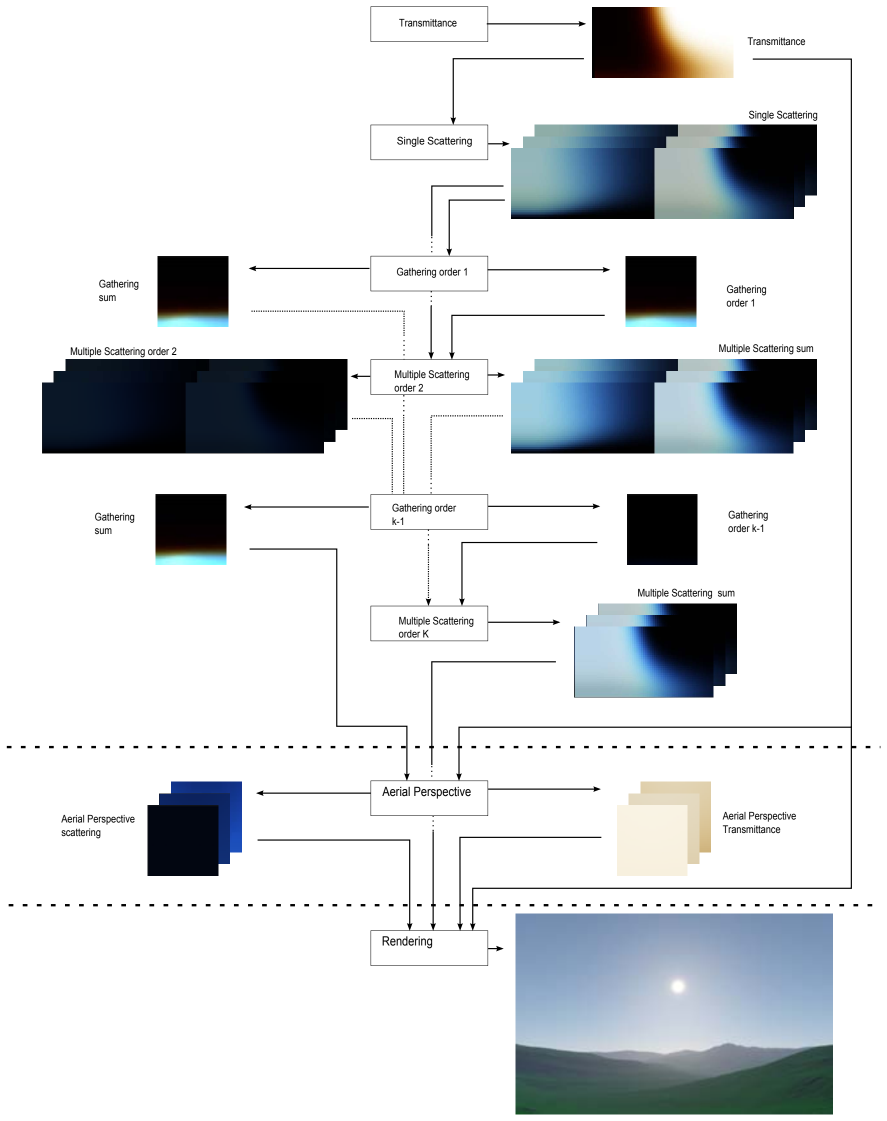

Figure 26: Complete system overview.

A.2 Additonal Visual Results

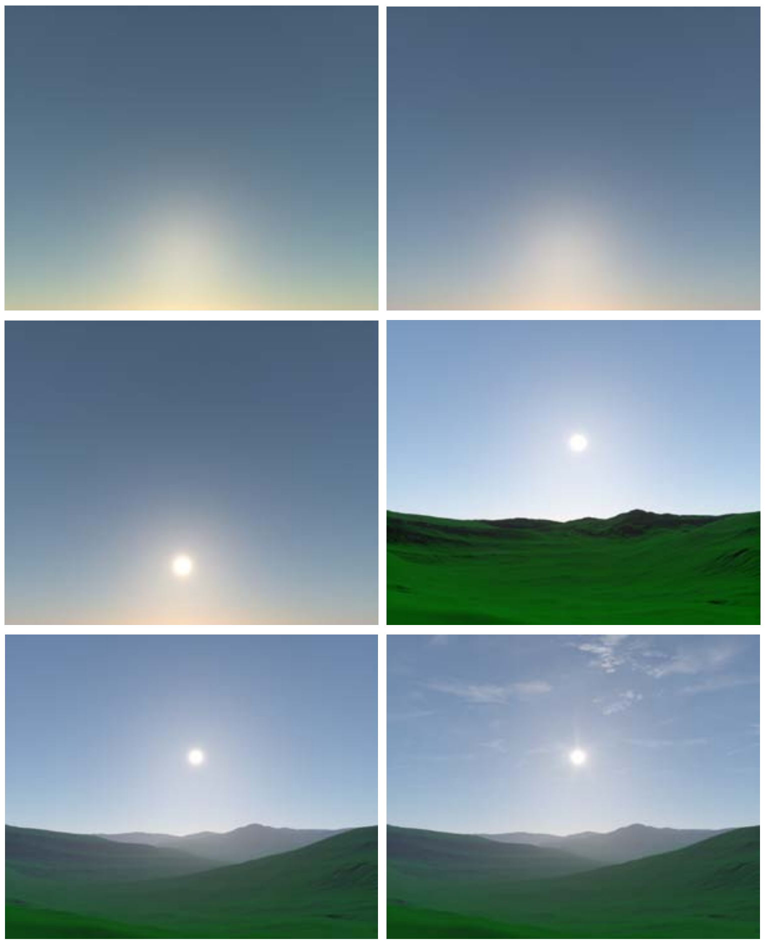

Figure 27: From left to right, and top to bottom: 1. Scattering only 2. Scattering with ozone contribution 3. Direct sunlight 4. Sky in a scene 5. With aerial perspective 6. With cloud layer and sun glare

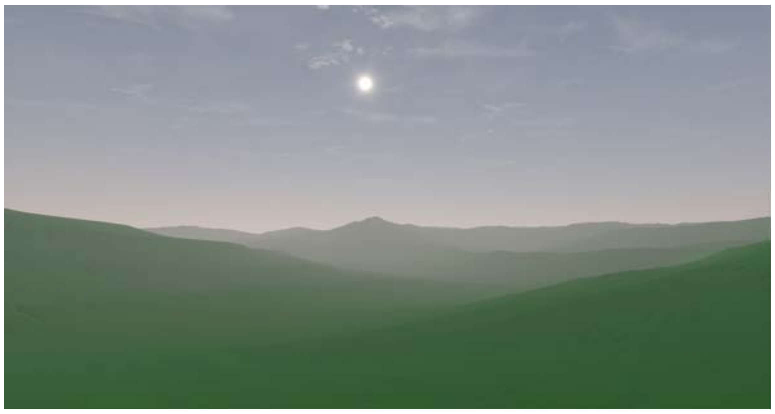

Figure 28: Foggy weather.

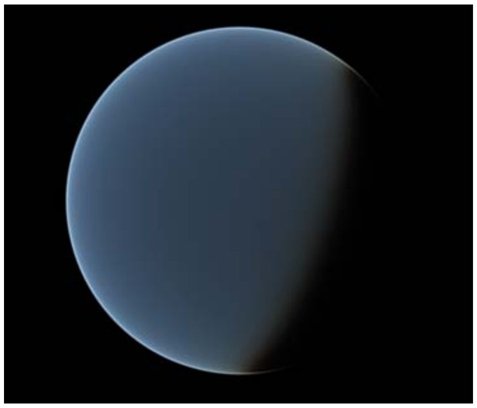

Figure 29: Inscattering only, rendered from space.