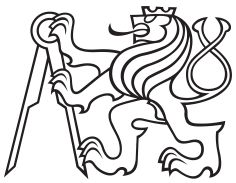
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of computer graphics and interaction

Daylight model for system VRUT

Michal Hvězda

Supervisor: Ing. Jaroslav Sloup
Field of study: Open Informatics
Subfield: Computer games and graphics
January 2020

Acknowledgements

Zde bych chtěl vzdát dík všem, kteří mi pomáhali při práci na tomto projektu. Hlavně bych chtěl poděkovat mému vedoucímu Ing. Jaroslavu Sloupovi za jeho rady a připomínky. Dále bych rád poděkovat svým rodičům za jejich podporu v průběhu mého bakalářského studia.

Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 2. January 2020

Abstract

This thesis deals with design and implementation of a module, for sky-light simulation, which can be used in system VRUT. Furthermore, it focuses on implementation of a standalone application which supports aerial perspective for sky-light simulation.

Keywords: Sky, Skylight, Daylight, VRUT, Computer graphics, Hosek-Wilkie, Bruneton

Supervisor: Ing. Jaroslav Sloup

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací modulu pro simulaci oblohy, který může být použit v systému VRUT. Práce se dále zaměřuje na implementaci samostatné aplikace, která podporuje vzdušnou perspektivu při simulaci oblohy.

Klíčová slova: Obloha, Skylight, Daylight, VRUT, Počítačová grafika, Hošek-Wilkie, Bruneton

Překlad názvu: Model oblohy pro systém VRUT

Contents

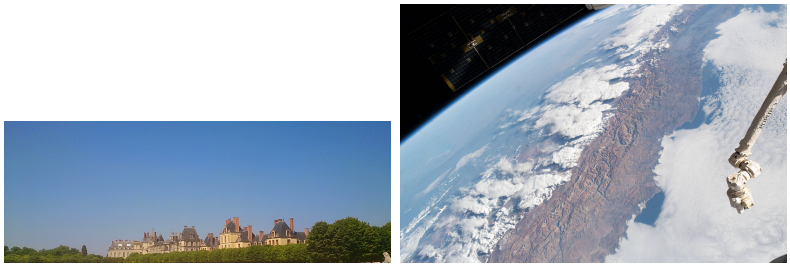
Figures

Tables

Chapter 1

Introduction

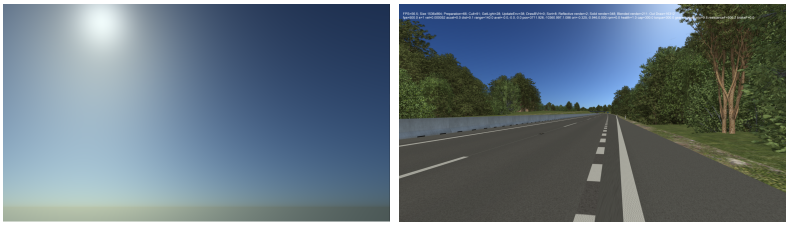
The sky is a big part of every realistic outdoor scene, and it is usually the first thing that makes an impression on a viewer (see Figure 1.1). Therefore it should behave and look as the observer would expect, and do not create the sensation of disbelief.



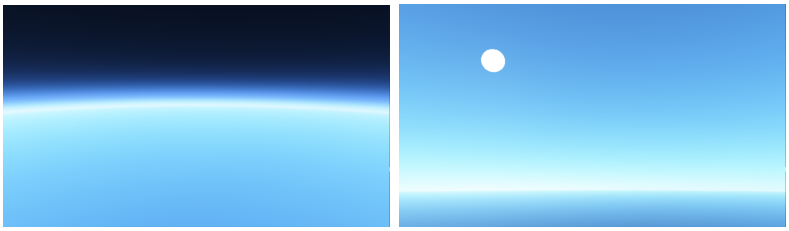
(a)

Figure 1.1: Reference photographs of the sky and atmosphere[NAS].

In computer graphics there are two popular methods to render the skydome. The first method would be via a fixed environment map, which might be a photograph or a hand-drawn image. Even though this method can be visually pleasing, it also has drawbacks. The main problem is the used image can not be changed if someone decides the sun should be a little bit higher or lower. The more flexible approach is to use a skylight model where the parameters of the sky, for example air turbidity or position of the sun, can be specified (see Figure 1.2). Existing skylight models vary in computation complexity and may not be feasible for real-time applications, but some models make simplifying assumptions to be viable for real-time rendering. This work focuses on those skylight models that can be computed in real-time.



(a)



(b)

Figure 1.2: Sample images generated by implemented models a) Hosek-Wilkie model b) Bruneton model.

1.1 Goal

The first goal of this project was to implement an skylight model as a module for a system VRUT which is a abbreviation of Virtual Reality universal toolkit. The second goal was to design and implement a standalone application with a skylight model that supports an aerial perspective. There are plenty of models that achieve a realistic sky and are rendered continuously in real-time [Bru17]. This work covers two of those models, which seemed the most promising for its goals. These two models are Lukas Hosek's and Aleksander Wilkie's model for Full Spectral Sky-Dome Radiance [HW12], and Eric Bruneton's model of precomputed atmospheric scattering[BN08]. Before description of the implantation of these models can be made, it is important to first describe how these models function and also define the clear sky model from which they are derived in the chapter (2). Then in chapters (3) and (4) are described the implementation details of the VRUT module and the standalone application, respectively. Last but not least, the results of the implementations are discussed in the chapter (6).

Chapter 2

Skylight Models

This chapter goes over the theory behind skylight models, but first it is necessery to define a few basic terms from radiometry, such as radiance, irradiance, and transmittance. It is also important to define light scattering and its two types, Rayleigh and Mie scattering. Then using these terms, a physical model (2.1), on which each of the models that follow is based on, can be described; however, each of them makes different assumptions to simplify it to be more feasible for real-time rendering. Last but not least, using the gathered knowledge, Hosek-Wilkie and Bruneton model are covered in sections (2.2) and (2.3), respectively.

As mentioned above, few basic terms from radiometry need to be defined. The first one is radiance L, defined as the quantity of radiation $[W]$ emitted by a surface $[m^2]$ while falling within a solid angle of $[sr]$. In other words, its the total amount of energy emitted, received, transmitted by a surface, per unit area, per unit solid angle. Its unit is $[W \cdot m^{-2} \cdot sr^{-1}]$. Spectral radiance L_λ describes radiance as a function of wavelength. This means that for certain wavelength intervals, the total amount of radiation is given for all wavelengths in that interval. Spectral radiance is denoted as $[W \cdot m^{-2} \cdot sr^{-1} \cdot nm^{-1}]$. Irradiance $\mathcal{E}$ is defined as the quantity of radiation received per unit area $[W \cdot m^{-2}]$. Similarly to spectral radiance the spectral irradiance has unit $[W \cdot m^{-2} \cdot nm^{-1}]$ [Kol12]. Finally, the transmittance T is the material's effectiveness in transferring radiation. In other words, it describes energy loss by absorption or scattering of the passing light through a material.

When light rays pass through a medium, they can collide with the medium's particles in such a way that their direction becomes altered. This physical

phenomenon is called scattering. If the light scatters only once then it is called single scattering. However, the light can also bounce between the medium's particles more than once before it finally escapes the medium. In this case, it is called multiple scattering (See Figure 2.1). The scattering

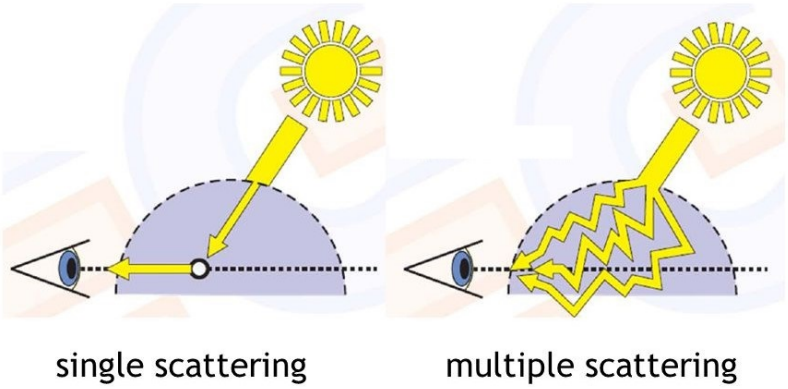


Figure 2.1: Comparison between single and multiple scattering [YYCM06].

has two main types that are the most prevalent. The first one is called the Rayleigh scattering. It is a type of elastic scattering, which means that the light does not lose energy in the scattering process. In other words, the light might change direction after being scattered, but energy remains constant. Elastic scattering is, of course, an idealistic idea since the light's energy changes, but the energy loss is so small that it can be neglected. Rayleigh scattering describes how the light is scattered by particles smaller than the light's wavelength λ. Since it is symmetrical around the axis of incoming light, it is not anisotropic, in other words its not dependant on the light's direction. The other type of scattering is called the Mie scattering. This

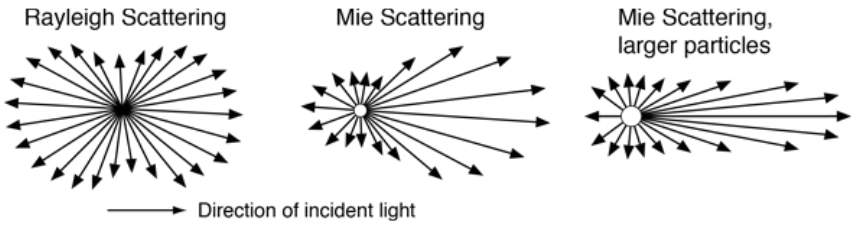


Figure 2.2: Difference in directional dependency of the Mie and the Rayleigh scattering [Nav].

type of scattering is caused by particles of the same or larger size than the light wavelength. In contrast to the Rayleigh scattering, the Mie scattering is highly anisotropic (see Figure 2.2)[Kol12].

2.1 Physical Model

In CG, the most widely used physical model of the atmosphere is the clear-sky model. The atmosphere, in the model, is described as a thin spherical layer of decreasing density containing only two components, air molecules and aerosol particles. The distribution of these particles is not uniform, but since the atmosphere is spherical, density can be determined by height r. The atmospheric layer that envelops the Earth starts at the sea level $R_g = 6360km$ and ends approximately at the height of the stratosphere $R_t = 6420km$ [BN08]. At each point of the atmosphere, light is scattered by air molecules and aerosol particles from its incidental direction by an angle θ. The amount of the scattered or absorbed light is given by a product of scattering coefficient β^s and a phase function P. The scattering coefficient β^s gives us the fractional rate in the transmission of radiation through a scattering medium and depends on the particle density of the medium [BN08]. Phase function P determines how much radiation is scattered in a particular direction by a particle. For air molecules which are smaller than a wavelength of light the scattering coefficient β^s_R and the Rayleigh phase function (see Figure 2.3) are given by the Rayleigh theory as [BN08]:

$$P_R(\theta) = \frac{3}{16\pi}(1 + cos^2(\theta)) \tag{2.1}$$

$$\beta^s_R(h, \lambda) = \frac{8\pi^3(n^2 - 1)^2}{3N\lambda^4} e^{-\frac{h}{H_R}} \tag{2.2}$$

where θ is a scattering angle from light's incidental direction, $h = r - R_g$ is

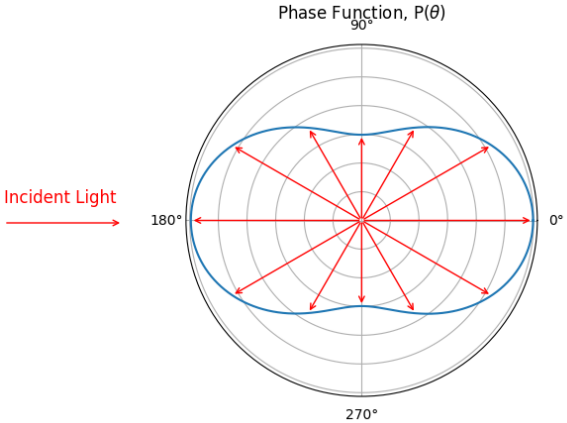


Figure 2.3: Rayleigh phase function for scattering angle θ.

the altitude, λ is wavelength of light, n is the index of refraction of air, N the molecular density at sea level R_g, and $H_R = 8km$ is a thickness of the part of the atmosphere that is affected by the Rayleigh scattering if it was uniform. On the other hand, aerosol particles are big enough to be affected

by the Mie scattering. Therefore their phase function P_M is given by the Mie theory, approximated by the Cornette-Shanks phase function (see Figure 2.4) as [BN08]:

$$P_M(\theta) = \frac{3}{8\pi} \frac{(1-g^2)(1+cos^2(\theta))}{(2+g^2)(1+g^2-2gcos(\theta))^{\frac{3}{2}}} \tag{2.3}$$

$$\beta_M^s(h,\lambda) = \beta_M^s(0,\lambda)e^{-\frac{h}{H_M}} \tag{2.4}$$

where $g \in (-1,1)$ is an asymmetry factor denoting the width of the forward

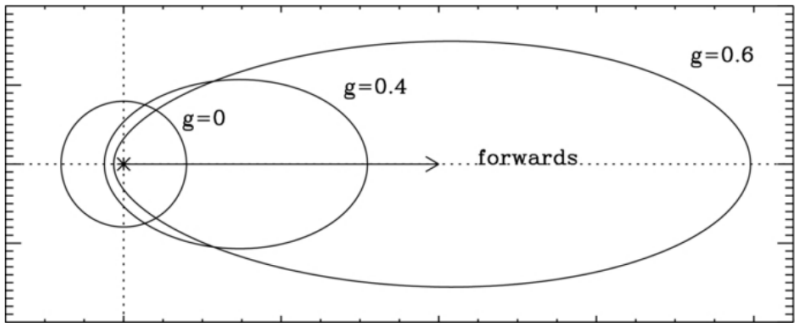


Figure 2.4: The Mie phase function consists of multiple lobes. Prevalence of these lobes is given by the size of the scattering particle.

lobe (see Figure 2.4), $H_M \simeq 1.2km$ is a height scale of aerosol particles within the atmosphere with exponentially decreasing density. The final portion of scattered and absorbed light at each point of the atmosphere is given by the sum of products of phase functions and extinction coefficients[BN08].

$$P_R\beta_R^e + P_M\beta_M^e \tag{2.5}$$

Where β_R^e and β_M^e are extinction coefficients. For small particles that are affected by the Rayleigh scattering, can be said that β_R^e is equal to β_R^s, since light scattered by the Rayleigh scattering does not lose energy. However, because the Mie scattering's behavior is not elastic, it is crucial to introduce an absorption coefficient of β_M^a to β_M^e, which would lead to following[BN08]:

$$\beta_M^e = \beta_M^s + \beta_M^a \tag{2.6}$$

2.1.1 Rendering equation

Now with the defined atmosphere, and how light is scattered and absorbed within it, the rendering equations to obtain light's radiance values at each point of the atmosphere can be defined. Note that light can also bounce off the ground, which can be modeled as a Lambertian surface with height field

of reflectance $\boldsymbol{\alpha}(\mathbf{x}, \boldsymbol{\lambda})$ and normal $\mathbf{n}(\mathbf{x})$. For the following rendering equations note that $L(\mathbf{x}, \mathbf{v}, \mathbf{s})$ is the radiance of light reaching point $\mathbf{x}$ from direction $\mathbf{v}$ and with sun's direction $\mathbf{s}$. The extremity of a ray $\mathbf{x}_0(\mathbf{x}, \mathbf{v})$ with origin in $\mathbf{x}$ and direction $\mathbf{v}$. Where $\mathbf{x}_0$ is either located on the ground R_g or at the top of the atmosphere boundary R_t. $T(\mathbf{x}, \mathbf{x}_0)$ describes transmittance of atmosphere between points $\mathbf{x}$ and $\mathbf{x}_0$. $\mathcal{I}(\mathbf{x}_0, \mathbf{s})$ is the radiance of light reflected from point $\mathbf{x}_0$ with sun's direction $\mathbf{s}$. $\mathcal{J}(\mathbf{y}, \mathbf{v}, \mathbf{s})$ is radiance of light at height y in direction $-v$ and sun's direction s. Values for each of the mentioned terms are given by equations bellow [BN08]:

$$T(\mathbf{x}, \mathbf{x}_0) = \exp(-\int_{\mathbf{x}}^{\mathbf{x}_0} \sum_{i \in \{R,M\}} \beta_i^e(\mathbf{y}) dy) \tag{2.7}$$

$$\mathcal{I}[L](\mathbf{x}_0, \mathbf{s}) = \begin{cases} 0 & \text{top of the atmosphere} \\ \frac{\boldsymbol{\alpha}(\mathbf{x}_0)}{\pi} \int_{2\pi} L(\mathbf{x}_0, \boldsymbol{\omega}, \mathbf{s}) \boldsymbol{\omega} \cdot \boldsymbol{n}(\mathbf{x}_0) d\omega & \text{otherwise} \end{cases} \tag{2.8}$$

$$\mathcal{J}[L](\mathbf{y}, \mathbf{v}, \mathbf{s}) = \int_{4\pi} \sum_{i \in \{R,M\}} \beta_i^{\mathbf{s}}(\mathbf{y}) P_i(\mathbf{v} \cdot \boldsymbol{\omega}) L(\mathbf{y}, \boldsymbol{\omega}, \mathbf{s}) d\omega \tag{2.9}$$

Using introduced notations the rendering equations for obtaining radiance values are defined as [BN08]:

$$L(\mathbf{x}, \mathbf{v}, \mathbf{s}) = (L_0 + R[L] + S[L])(\mathbf{x}, \mathbf{v}, \mathbf{s}) \tag{2.10}$$

$$L_0(\mathbf{x}, \mathbf{v}, \mathbf{s}) = \begin{cases} T(\mathbf{x}, \mathbf{x}_0) L_{sun} & \text{otherwise} \\ 0 & \mathbf{v} \neq \mathbf{s} \text{ or Sun is occluded} \end{cases} \tag{2.11}$$

$$R[L](\mathbf{x}, \mathbf{v}, \mathbf{s}) = T(\mathbf{x}, \mathbf{x}_0) \mathcal{I}[L](\mathbf{x}_0, \mathbf{s}) \tag{2.12}$$

$$S[L](\mathbf{x}, \mathbf{v}, \mathbf{s}) = \int_{\mathbf{x}}^{\mathbf{x}_0} T(\mathbf{x}, \mathbf{y}) \mathcal{J}[L](\mathbf{y}, \mathbf{v}, \mathbf{s}) dy \tag{2.13}$$

where L_0 is the direct sunlight L_{sun} attenuated by T, R is the light reflected at $\mathbf{x}_0$ attenuated by T before reaching $\mathbf{x}$, S is the portion of light between $\mathbf{x}$ and $\mathbf{x}_0$ which scattered towards $\mathbf{x}$ (see Figure 2.5).

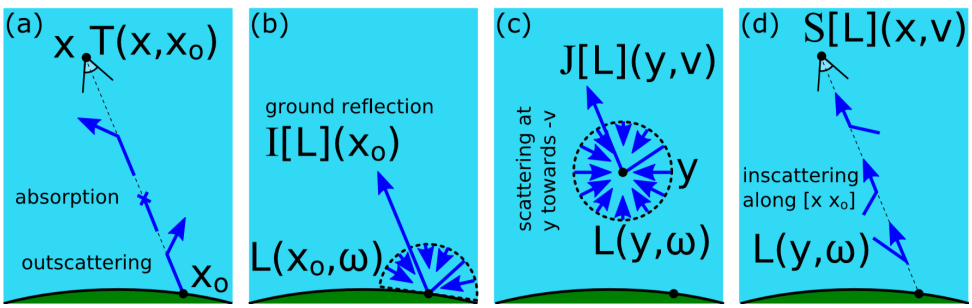


Figure 2.5: Visual representations of all the defined rendering equations [BN08].

As can be seen the equation 2.10 is very complex to solve, and therefore not suitable for real-time rendering. For this reason, many analytical assumptions have been made to simplify it. One of these assumptions, which is also the most widely used by real-time applications, is to ignore multiple scattering. With only single scattering in mind, the equation 2.10 would be reduced into the equation bellow [BN08]:

$$L = L_0 + R[L_0] + S[L_0] \tag{2.14}$$

However, this still leaves us with a problem that $S[L_0]$ is also too complicated to solve. Therefore some analytical models proposed further simplifications, such as flat earth with constant atmosphere density or without the Mie scattering, to tackle this problem. These simplifications, however, come at a price of idealization. For example, the flat earth hypothesis limits the observer only to the ground level [HW12]. A popular method in CG is to use an analytical formula in which parameters are fitted to externally obtained reference data. This kind of method is usually low cost, thus ideal for real-time. Another general approach to solve $S[L_0]$ is to use numerical integration with the use of low sampling [O'N05].

2.2 Hosek-Wilkie Model

The Hosek and Wilkie's model [HW12] is an example of an analytical approach for solving the physical model assuming only single scattering and flat earth hypothesis. The model is inspired by an already existing Preetham model [AJP99], which it tries to improve. To develop such a model, they first needed reference data. Thus they created a brute force path tracer that simulates interactions between particles in the atmosphere and light. Then they used the path tracer to generate a large number of reference images of the sky-dome for different atmospheric conditions (see Figure 2.6). With access to reference data and the knowledge of how Perez formula, which is the core of Preetham's model [AJP99], is derived, they devised an extended Perez formula and fitted the distribution parameters to the reference data.

2.2.1 Calculating spectral radiance

The core of the Hosek-Wilkie model is an extended Perez formula with included anisotropic term χ, which places a localized glow around solar point. This anisotropic term used is to simulate the zero-order glow of the Mie scattering, which produces a phenomenon called circumsolar ring (see Figure

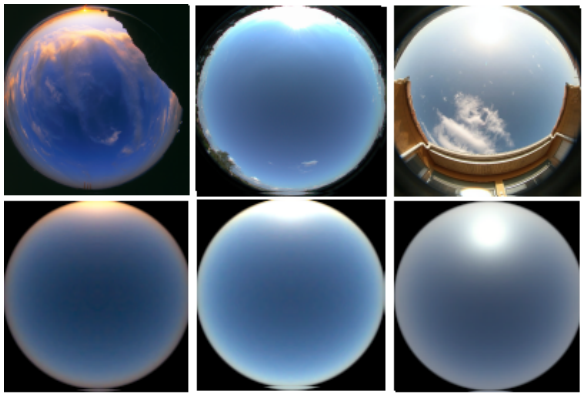


Figure 2.6: Three comparison images between actual photose and results of the Hosek-Wilkie brute force path tracer [HW12].

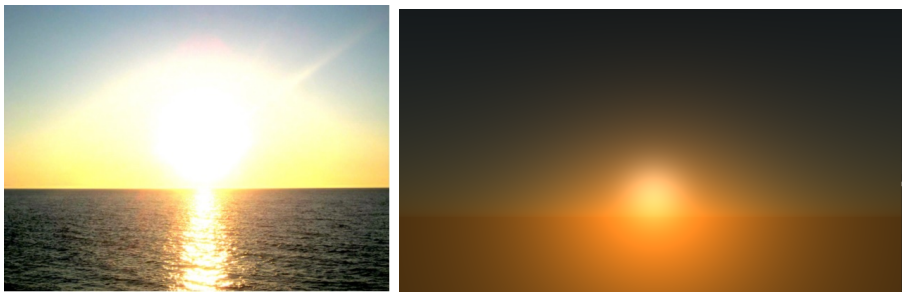


Figure 2.7: Comparison between actual photo of circumsolar ring (aureole) and result of of the Hosek-Wilkie model[Kol12]

2.7). Extended Perez formula and anisotropic term χ are defined bellow [HW12].

$$F(\theta,\gamma) = (1+Ae^{\frac{B}{cos\theta+0.01}})(C+De^{E\gamma}+F\cos^2\gamma+G\chi(H,\gamma)+Icos^{\frac{1}{2}}\theta) \quad (2.15)$$

$$\chi(g,\alpha) = \frac{1+cos^2\alpha}{1+g^2-2g\cdot cos\alpha} \quad (2.16)$$

The original Perez formula which served as base for the previusly defined extended model is given by [HW12]:

$$F_{Perez}(\theta,\gamma) = (1+Ae^{B/\cos\theta})(1+Ce^{D\gamma}+E\cos^2\gamma) \quad (2.17)$$

where θ is the zenith view angle, γ is the angle formed by viewing ray and solar point. Parameters A through I are distribution coefficients to tune the luminance distribution and do not directly translate to any physical quantities.

Note that coefficient C has replaced the value of 1 in the second parenthesize. This is to obtain absolute luminance values at zenith instead of relative ones, since they decided, as opposed to the Preetham model, not to normalize against zenith luminance. The decision not to normalize by zenith luminance

is due to a complication that arose, with the inclusion of the anisotropic term, when they tried to do it. Instead, they decided to multiply the result by the expected value of spectral radiance. The final value of the spectral radiance is calculated as follows [HW12]:

$$L_\lambda = F(\theta, \gamma) \cdot L_{M\lambda} \tag{2.18}$$

where $L_{M\lambda}$ is an expected value of spectral radiance in a point randomly selected in the upper hemisphere with uniform distribution.

Radiance distribution parameters A through I in Hosek-Wilkie's extended formula are calculated using bezier curves, as opposed to Preetham's linear function of turbidity. Thus they provide a four dimensional table ($10\times2\times9\times6$) M^λ of values $m_{T,\alpha,p,c}$ - for each integer values of turbidities $T \in \{1,\ldots,10\}$, two values of albedo $\alpha \in \{0,1\}$ and nine distribution parameters $p \in \{1,\ldots,9\}$ there is total of six controls points $c \in \{1,\ldots,6\}$ of a bezier curve. The resulting vector of distributional parameters A through I is calculated as follows [HW12]:

$$\begin{aligned} v_p^\lambda = {} & m^\lambda_{T,\alpha,p,1} \cdot (1-x)^5 + \\ & m^\lambda_{T,\alpha,p,2} \cdot 5x(1-x)^4 + \\ & m^\lambda_{T,\alpha,p,3} \cdot 10x^2(1-x)^3 + \\ & m^\lambda_{T,\alpha,p,4} \cdot 10x^3(1-x)^2 + \\ & m^\lambda_{T,\alpha,p,5} \cdot 5x^4(1-x)^1 \\ & m^\lambda_{T,\alpha,p,6} \cdot x^5 \end{aligned} \tag{2.19}$$

where x is the solar elevation normalized to an interval [0,1] as:

$$x = \sqrt[3]{\frac{\eta}{\frac{\pi}{2}}}$$

The reason for the cube root is to spread the values more evenly along the interpolation interval since most of the changes in radiance distribution pattern happen abruptly at low solar elevations.

The expected value of spectral radiance $L_{M\lambda}$ is calculated similarly as distribution parameters; the only difference being that the result is a single value. This means there is a three dimensional ($10 \times 2 \times 6$) table R^λ of values $r_{T,\alpha,c}$ for which the same quantic bezier interpolation is used to obtain a single value of $L_{M\lambda}$.

2.2.2 Limitations and Complexity

The Hosek-Wilkie model limits the observer to the ground since the analytical formulas were fitted only for views at the ground level. Another limitation of the model is that it supports only the sun's directions above the horizon. The reason being that it would need a different analytical model since the sky radiance pattern is significantly different at sunrise or sunset than during the day.

Time and memory complexity to render a pixel is O(1). There is no precomputation phase if a user wants to use the provided radiance data. However, it is needed to recompute the sky radiance for many views, sun directions, and perform a new nonlinear fitting, if one wants to change atmospheric parameters [Bru17].

2.3 Bruneton Model

In contrast to the previously described model, which uses externally obtained reference data for fitting distributional coefficients inside an analytical formula, the Bruneton method [BN08] is to solve equation (2.10) directly. This is done via a precomputation phase, where the model tries to precompute equation (2.10) as much as possible, with minimal approximations or idealism. Zero and single scatterings are computed precisely; however, multiple scattering is only approximated by a perfect sphere of constant reflectance to allow for mentioned precomputations.

First, since it is assumed the ground is a perfect sphere it is important to define notations L', L'_0,L'_{sun},P'_M, P'_R, R', $\mathcal{S}'$, T', $\mathcal{J}'$, $\mathcal{E}'$, α', x'_0 as spherical approximations of their counterparts from (2.1). Also because of our spherical approximations a position $\mathbf{x}$ and a direction $\mathbf{v}$ can be reduced to an altitude r and a view zenith angle μ. Sun direction $\mathbf{s}$ can be also mapped to spherical coordinates μ_s and ν. With the defined notations the functions that depend on $\mathbf{x}, \mathbf{v}, \mathbf{s}$ can be expressed only by four parameters r, μ, μ_s and ν, where each value is given by following parametrization:

$$r = ||x|| \tag{2.20}$$

$$\mu = \frac{x \cdot v}{r} \tag{2.21}$$

$$\mu_s = \frac{x \cdot s}{r} \tag{2.22}$$

$$\nu = v \cdot s \tag{2.23}$$

The main idea behind the precomputation phase and its algorithm (see algorithm 0) is that the radiance value L can be express with a series in the linear operators R and S as:

$$\begin{aligned} L &= L_0 + (R+S)[L_0] + (R+S)[(R+S)[L0]] + ... \\ &= L_0 + L_1 + L_2 + ... \\ &= L_0 + \sum_{i=1}^{n} L_i \\ &= L_0 + L_* \end{aligned} \tag{2.24}$$

where each L_i is the contribution of light reflected and/or scattered exactly i times. R is for L_0 computed exactly during rendering using equation (2.12), S on the other hand is more complicated since it is an integral between $\mathbf{x}$ and $\mathbf{x}_0$, but due to the oclusion term in L_0, the integral is null at all points $\mathbf{y}$ that are in shadow. Using this fact it can be assumed that these points are between $\mathbf{x}_s$ and $\mathbf{x}$, which means we can ignore the occlusion. Therefore the integral can be reduced to lit segment $[\mathbf{x}, \mathbf{x}_s]$. Thus $S[L_0]$ can be computed as $S[L_0] = \int_{\mathbf{x}}^{\mathbf{x}_s} T\mathcal{J}[L'_0] = \int_{\mathbf{x}}^{\mathbf{x'}_0} T\mathcal{J}[L'_0] - \int_{\mathbf{x'}_0}^{\mathbf{x}_s} T\mathcal{J}[L'_0]$, which can be rewritten as a equation of two precomputed functions T' and $S'[L'_0]$ [BN08]:

$$S[L_0](\mathbf{x}, \mathbf{v}, \mathbf{s}) = S'[L'_0](\mathbf{x}, \mathbf{v}, \mathbf{s}) - T'(\mathbf{x}, \mathbf{x_s}) S'[L'_0](\mathbf{x_s}, \mathbf{v}, \mathbf{s}) \tag{2.25}$$

For multiple scattering it is more difficult to account for occlusion in the terms $L_2 + L_3 + ... + L_n = R[L_*] + S[L_*]$. Fortunately effects of the multiple scattering are small compared to the single scattering during the day, and the contribution of the ground is small when it's not directly lit by the sun [BN08]. Therefore Bruneton proposes to approximate occlusion effects in $S[L_*]$ by integrating the multiple scattering contributions, without occlusion, between $\mathbf{x}$ and $\mathbf{x}_s$. This gives positive and negative bias. The approximation of $S[L_*]$ gives $S[L_*] \simeq \int_{\mathbf{x}}^{\mathbf{x}_s} T\mathcal{J}[L'_*]$. $R[L_*]$ approximates occlusion effects with the ambient occlusion of horizontal hemisphere due to the ground's tangent plane, $\frac{1+\mathbf{n}\cdot\mathbf{n}'}{2}$. This gives $R[L_*] \simeq \hat{R}[L_*]$ with [BN08]:

$$\hat{R}[L'_*] = T(\mathbf{x}, \mathbf{x}_0) \frac{\alpha(\mathbf{x}_0)}{\pi} \frac{1 + \mathbf{n}(\mathbf{x}_0) \cdot \mathbf{n}'(\mathbf{x}_0)}{2} \mathcal{E}'[L'_*](\mathbf{x}_0, \mathbf{s}) \tag{2.26}$$

$$\mathcal{E}'[L'_*](\mathbf{x}_0, \mathbf{s}) = \begin{cases} \int_{2\pi} L'_*(\mathbf{x}_0, \boldsymbol{\omega}, \mathbf{s}) \boldsymbol{\omega} \cdot \mathbf{n}'(\mathbf{x}_0) d\omega \\ 0 \end{cases} \tag{2.27}$$

Finally using this idea the result of the algorithm (0) are three tables $\mathbb{T}$, $\mathbb{E}$ and $\mathbb{S}$. Table $\mathbb{T}$, which after our previous parametrization depends on altitude r and view zenith angle μ, stores transmittance $T'(r, \mu)$ for all r and μ. Because $T'(r, \mu)$ corresponds to $T(\mathbf{x}, \mathbf{x}_0(\mathbf{x}, \mathbf{v}))$ the following identity needs to be used in order to compute the transmittance between two points x and y.

$$T(x, y) = \frac{T'(r_x, \mu_x)}{T'(r_d, \mu_d)} \tag{2.28}$$

where r_x and μ_x are calculated using equations 2.20 and 2.21 respectively. r_d and μ_d are given by formula:

$$r_d = \sqrt{d^2 + 2d\mu_x r_x + r_x^2} \tag{2.29}$$

$$\mu_d = \frac{r_x \mu_x + d}{r_d} \tag{2.30}$$

Note that d is simply the distance between points x and y. The table $\mathbb{E}$ holds the accumulated irradiance $\mathcal{E}'(r, \mu_s, \nu)$ of all scattering orders. The value of irradiance for each scattering order is given by evaluating the equation 2.27. Last but not least the third table $\mathbb{S}$ stores the scattering values $S'(r, \mu, \mu_s, \nu)$. To precompute $\mathbb{E}$ and $\mathbb{S}$ the algorithm uses three intermediate tables ΔE, ΔS and ΔJ containing after each iteration i $\mathcal{E}'[L'_i]$, $S'[L'_i]$ and $J'[L'_i]$. At the end of each iteration ΔE and ΔS are added to the result tables $\mathbb{E}$ and $\mathbb{S}$ (see algorithm 0).

Algorithm 1 *Precompute*(n_{orders})

Input:
A . . . Atmospheric Parameters
Output:
$\mathbb{S}$ **. . . Multiple scattering table**
$\mathbb{E}$ **. . . Irradiance table**
$\mathbb{T}$ **. . . Transmittance table**

$\mathbb{T}(r, \mu) \leftarrow T'(r, \mu)$;
$\Delta\mathbb{E}(r, \mu_s, \nu) \leftarrow \mathcal{E}'[L'_0](r, \mu_s, \nu)$
$\Delta\mathbb{S}(r, \mu, \mu_s, \nu) \leftarrow \mathcal{S}'[L'_0](r, \mu, \mu_s, \nu)$
$\mathbb{E}(r, \mu) \leftarrow 0$
$\mathbb{S}(r, \mu, \mu_s, \nu) \leftarrow \Delta\mathbb{S}(r, \mu, \mu_s, \nu)$
for $(i \leftarrow 1\ to\ i < n_{orders})$ **do**
  $\Delta\mathbb{J}(r, \mu, \mu_s, \nu) \leftarrow \mathcal{J}'[T\frac{\alpha'}{\pi}\Delta\mathbb{E} + \Delta\mathbb{S}](r, \mu, \mu_s, \nu)$
  $\Delta\mathbb{E}(r, \mu_s, \nu) \leftarrow \mathcal{E}'[T\frac{\alpha'}{\pi}\Delta E + \Delta S](r, \mu_s, \nu) = \mathcal{E}'[\Delta S](r, \mu_s, \nu)$
  $\Delta\mathbb{S}(r, \mu, \mu_s, \nu) \leftarrow \int_x^{x'_0} T(x, y)\Delta\mathbb{J}(||y||, \mu, \mu_s, \nu)dy$
  $\mathbb{E}(r, \mu_s, \nu) \leftarrow \mathbb{E}(r, \mu_s, \nu) + \Delta\mathbb{E}(r, \mu_s, \nu)$
  $\mathbb{S}(r, \mu, \mu_s, \nu) \leftarrow \mathbb{S}(r, \mu, \mu_s, \nu) + \Delta\mathbb{S}(r, \mu, \mu_s, \nu)$

To store values into our N dimension tables, a special mapping between coordinates and their physical counterparts is needed. For tables $\mathbb{T}$ and $\mathbb{E}$ which depend only on r and μ, a naive mapping of normalizing the values r and μ into a table of indices in $[0, 1]^2$ is sufficient. However, for $\mathbb{S}$ an issue arises with mapping from (r, μ, μ_s, ν) into a table indices in $[0, 1]^4$. With the usage of the simple mapping, a problem of rapidly growing memory usage would appear, because of the high resolutions needed for μ to get a good sampling for aerial perspective. Lower resolutions for μ would result in visible artifacts, especially near the horizon. To avoid this problem and better exploit

the range of possible values, Bruneton proposes the mapping bellow [BN08]:

$$u_r = \frac{\rho}{H} \tag{2.31}$$

$$u_\mu = \begin{cases} \frac{1}{2} + \frac{r\mu + \sqrt{\Delta}}{2\rho} & \text{if } r\mu < 0 \text{ and } \Delta > 0 \\ \frac{1}{2} - \frac{r\mu + \sqrt{\Delta + H^2}}{2\rho + 2H} & \text{otherwise} \end{cases} \tag{2.32}$$

$$u_{\mu_s} = \frac{1 - e^{-3\mu_s - 0.6}}{1 - e^{-3.6}} \tag{2.33}$$

$$u_\nu = \frac{1 + v}{2} \tag{2.34}$$

where

$$\rho = ||r - R_g|| \tag{2.35}$$

$$H = ||R_t - R_g|| \tag{2.36}$$

$$\Delta = r^2\mu^2 - \rho \tag{2.37}$$

Furthermore, to save space, Bruneton proposes storing the red value of the Mie scattering into the alpha channel of $\mathbb{S}$. To approximate the rest of the values of the the Mie scattering using the red channel the formula below is used [BN08].

$$S_M \approx S_R \frac{S_{M,r}}{S_{R,r}} \frac{\beta^S_{R,r}}{\beta^S_{M,r}} \frac{\beta^S_R}{\beta^S_M} \tag{2.38}$$

Finally, to obtain the radiance values to render the sky and aerial perspective, equations bellow are evaluated for each pixel.

$$L \simeq L_0 + R[L_0] + \hat{R}[L'_*] + S'[L']_{|\mathbf{x}} - T(\mathbf{x}, \mathbf{x}_s) S'[L']_{\mathbf{x}_s} \tag{2.39}$$

Where $\mathbf{x}$ is a camera position, $\mathbf{x}_s$ depends on terrain shadows and gives light shafts. L_0 is computed as in (2.11) using only precomputed values of transmittance from table $\mathbb{T}$. Computing $R[L'_0]$ requires transmittance table $\mathbb{T}, \alpha(\mathbf{x}_0)$ and $\mathbf{n}(\mathbf{x}_0)$ and a shadow test to determine if $\mathbf{x}_0$ is lit. Lastly to compute $\hat{R}[L'_*]$ and $S'[L']_{|x} = S'[L'](\mathbf{x}, \mathbf{v}, \mathbf{s})$ the irradiance table $\mathbb{E}$ and the scattering table $\mathbb{S}$ are used.

The model is also able to generate light shafts but since this thesis focuses only on the sky the algorithm to do so will not be covered. The algorithm can be found in the original paper [BN08].

2.3.1 Limitations and Complexity

In contrast to the Hosek-Wilkie model, the Bruneton model supports all viewing positions from the ground to space. Its main limitation, which

authors want to remove in the future, is rooted in the dependency on the clear-sky model. Since the model assumes homogeneous particle density in the atmosphere, dependent only on the distance from the Earth center, it is not affected by clouds or other pollution in the atmosphere.

Time and memory complexity to render a pixel is O(1). However Precomputation phase involves computing integral over all $n_\theta \times n_\phi$ directions for every cell of the 4D table ΔJ, yielding a $O(n_r n_\mu n_{\mu_s} n_{\nu_s} n_\theta n_\phi)$ memory complexity, and $O(n_r n_\theta n_{\mu_s} n_\nu)$ time complexity [Bru17].

Chapter 3

Skylight model for system VRUT

Virtual reality universal toolkit or VRUT [Kyb08] for short was created in collaboration with the department of computer graphics and interaction ČVUT FEL, and Škoda Auto. It is a universal flexible tool for working with graphical data. The core application serves as a link between multiple, relatively independent modules. These modules can assign different tasks regarding displaying or other functionalities over graphical data and thus extend the functionality of the application as a whole. The communication between modules is handled via messages which are distributed by the VRUT's kernel. One of the assignments for this project was to extend the library of these modules by one that helps to bring more realism into scenes created in VRUT through one of the existing skylight models.

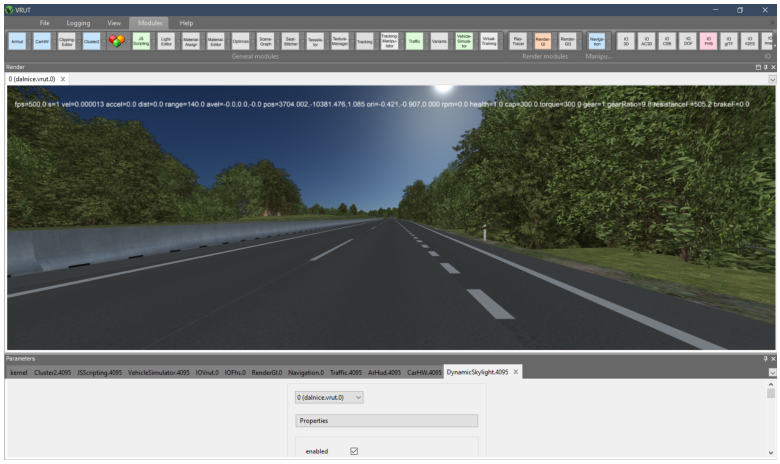


Figure 3.1: VRUT's graphical interface with DynamicSkylight module enabled for turbidity $T = 3$ and albedo $\alpha = 0.1$.

3.1 Implemenatation Design

Skylight model for system VRUT had to be computed in real-time, preferably without any precomputations. Also, since a viewer in a scene is limited to the ground, the picked model did not have to support an aerial perspective. This fact led to the decision to implement the Hosek-Wilkie model (2.2), which supports only the ground perspective and has computation complexity to render a pixel O(1) with no precomputations.

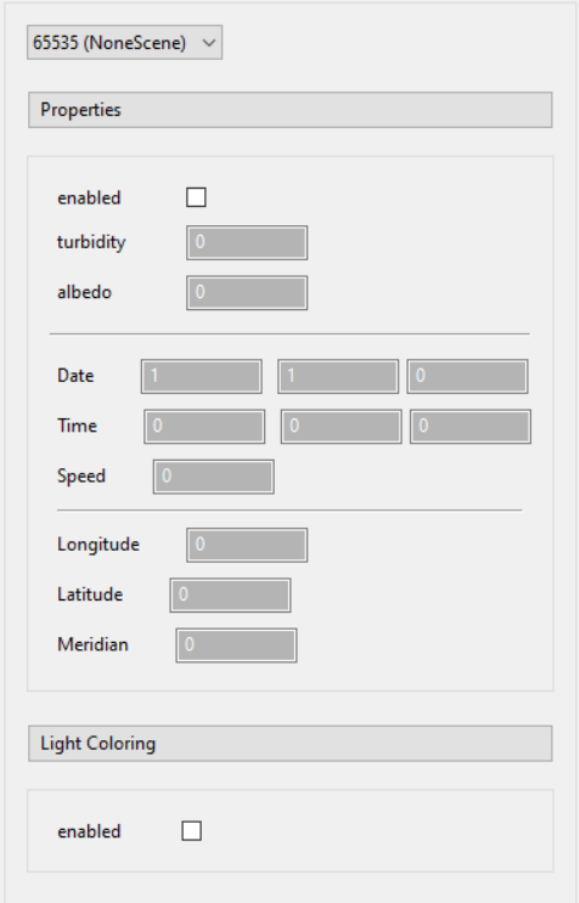


Figure 3.2: Graphical user interface (GUI) of the implemented module.

The skylight model is implemented into the system VRUT as a module with a graphical user interface. The GUI gives a user the freedom to choose the time of the day, speed of elapsing time, GPS coordinates of the scene, and tweak parameters of the Hosek-Wilkie model, such as turbidity and ground albedo (see Figure 3.2).The choice of light coloring is also included, which sets the color of the directional light, that would be the scene's sun, to the computed color of the sun by the extended Perez formula.

3.2 Calculation of sun position

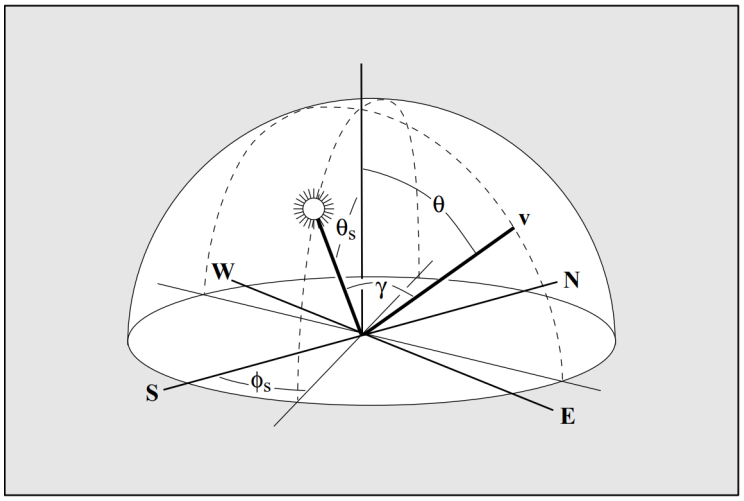


Figure 3.3: Coordinate system used for implementing Hosek-Wilkie model [AJP99].

One of the requirements for our module was to be able to automatically determine the sun's position based on observers selected GPS coordinates, date, and the time of the day. To do that, the sun's position can defined by two angles, the sun's altitude θ_s and azimuth ϕ_s. Values of these angles are obtained in radians by evaluation following equations [AJP99]:

$$\theta_s = \frac{\pi}{2} - arcsin(sinX_\phi \cdot sin\delta - cosX_\phi \cdot cos\delta \cdot cos\frac{\pi t_s}{12}) \tag{3.1}$$

$$\phi_s = arctan(\frac{-cos\delta \cdot sin\frac{\pi t_s}{12}}{cosX_\phi \cdot sin\delta - sinX_\phi \cdot cos\delta \cdot cos\frac{\pi t_s}{12}}) \tag{3.2}$$

$$\delta = 0.4093 \cdot sin(2\pi\frac{J-81}{368}) \tag{3.3}$$

where X_ϕ is latitude, t_s is true solar time and δ solar declination. True solar time can be obtained using user inputted local time t_m, date which is then internally converted into Julian calendar, longitude X_λ and lastly SM is standard meridian which determines the timezone.

$$t_s = t_m + 0.170 \cdot sin(4.0\pi\frac{J-80}{373}) - 0.129 \cdot sin(2\pi\frac{J-8}{355}) + 12\frac{SM - X_\lambda}{\pi}$$

GPS parameters and standard meridian are put into the formula in radians, but for ease of use the GUI takes parameters in degrees.

During testing, a flaw with the above formula was revealed when the azimuth reaches its limit on the interval $[0, \pi]$. When this event happens, the azimuth jumps to the other side of the range, even though the altitude still

has not reached the horizon, which leads to an error in the sun position. This error is, of course, an issue; however, solving this problem would most likely require a different formula for calculation of solar angles.

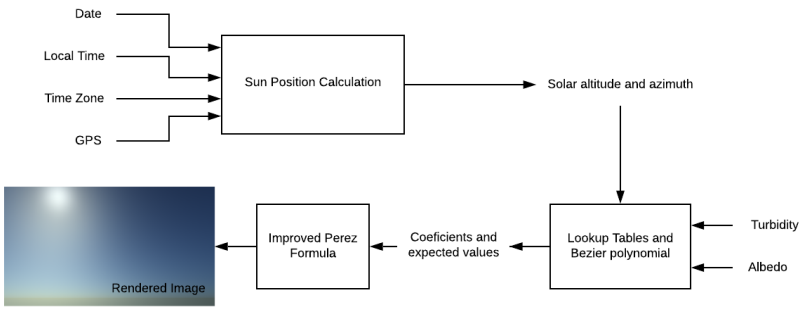


Figure 3.4: The rendering pipeline of the Hosek-Wilkie model.

3.3 Implementation details

Since system VRUT can be run on machines with or without a graphics card, the module runs purely on CPU with sequential rendering. The plan was to also make use of the OpenGL library for drawing the environmental map, however that would require create a new rendering pass inside the VRUT. Beause of this fact , it was decided that the VRUT implementation would only use the pure CPU rendering and the rendering using OpenGL would be part of the standalone application.

First, the implementation has the whole pipeline implemented inside a single class, DynamicSkyLight. The calculation is done in three stages (see Figure 3.4). Firstly the solar angles are calculated using the formulas from (3.2), which returns a vector of two elements, the sun's altitude, and azimuth. Then using the provided tables of spectral radiance, coefficients A through I and the expected value of spectral radiance $L_{M\lambda}$ are calculated as in formula (2.19). For each side of the cube map, the previously acquired values are used to determine the value of the extended Perez formula at each pixel. After the cube map is drawn, it is sent to the scene to replace a previous one. The module also proceeds to update the background's material to alert the SceneManager that a change was made. Also, if enabled, the module uses the Perez formula to acquire the sun's color, which is then used as a diffuse color for directional light (see Figure 3.5). The CPU version produces six images of a cube environment map on each draw call. Each side of the cube map has

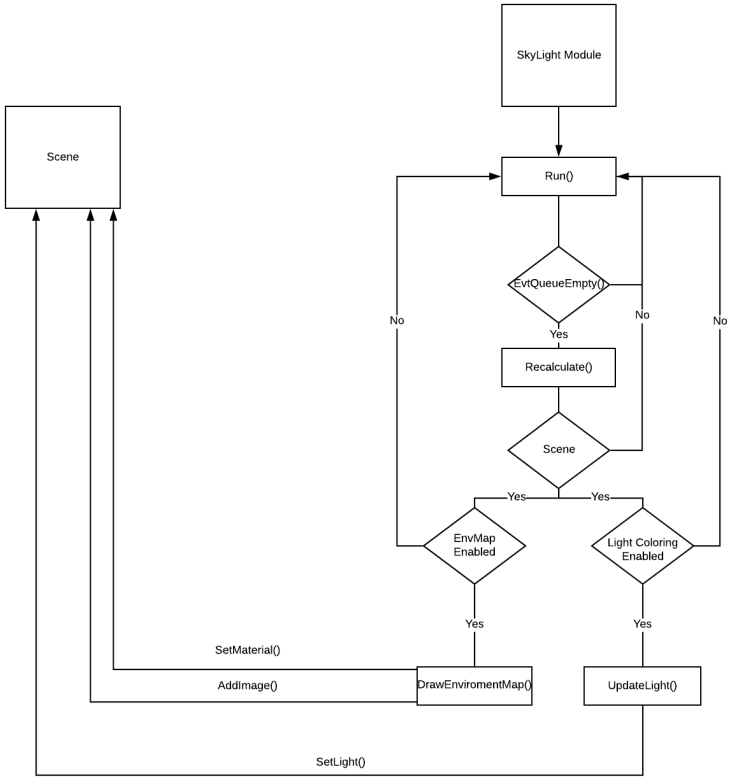


Figure 3.5: Diagram of module's run cycle.

proportions of 256×256 with three unsigned char channels. The texture size is set this way due to the limitation of sequential drawing. Higher resolutions led to a noticeable slowdown (see Table ...) without any appreciable increase in visual quality.

The OpenGL version in the standalone application uses the same computational pipeline as the CPU one, with one difference. Since the implementation has access to the GPU, the drawing can outsourced to it, and not compute the image sequentially as in the CPU method. Distribution parameters and the expected value of spectral radiance are sent to the fragment shader as a ten component vector where each cell has three float values. Also since the implementation is not limited by sequential rendering, it can use a bigger cube map. The used environment map's sides have each size 1024×1024 with 3 float channels.

Chapter 4

Standalone Application

Second goal of this project was to design and implement a standalone application, which uses a skylight model that is able to render an aerial perspective. For this purpose the application uses the Bruneton model which not only supports the aerial perspective but also views from space (see Figure 4.1).

Figure 4.1: Examples of different observer positions from the ground to the space.

4.1 Implementation Design

The application did not have to be very complex therefore it uses the Model-View-Controller design (MVC). The MVC design separates the data model, user interface, and control management into three separate components (see Figure 4.2).

- Model represents data used by an application.
- View converts given data into an view-able form for an user.
- Controller reacts to events and translates them into actions to be performed by a model.

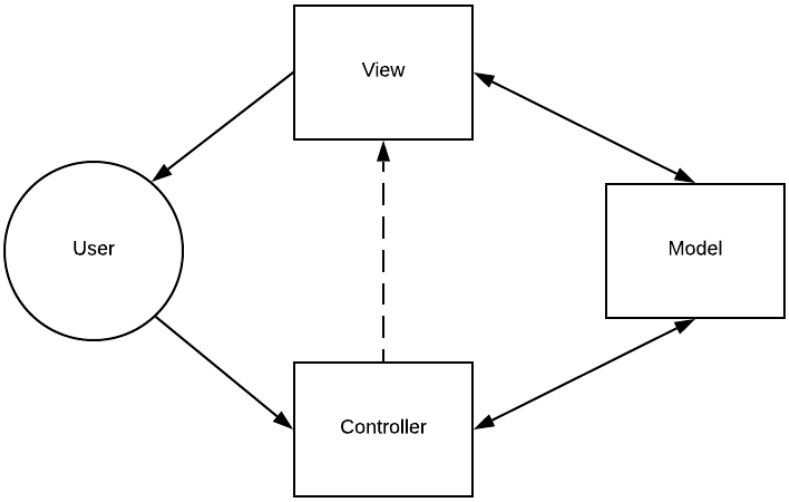


Figure 4.2: Diagram of MVC architecture

4.2 Implemetation of Bruneton model

Since OpenGL does not support 4D textures, the implemantion stores the 4-dimensional tables, which depend on parameters r, μ, μ_s, and ν, as 3D textures. Implementation of these 3D textures is done similarly as in the original Eric Bruneton's implementation. The tables are defined as 3D tables, but each 2D layer stores 3D information as sub-layers (see Figure. 4.3) positioned next to each other. Also since atmospheric parameters are not changed during run time they are directly baked into each shader where they can be accessed by a constant.

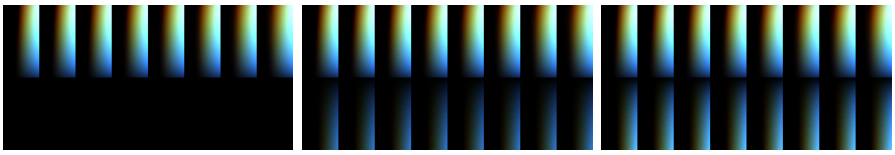


Figure 4.3: 4D table $\Delta\mathbb{S}$ as a 3D texture.

Our implementation uses the same texture sizes as were used in the original implementation. Transmittance texture T is 256×64, has 3 float channels. Irradiance textures $\mathbb{E}$ and $\Delta\mathbb{E}$ are 64×16, each has 3 float channels. Scattering textures $\mathbb{S}$ and $\Delta\mathbb{S}$ have width 32×8 (see Figure 4.3), height 128, and depth 32. Each of them has 4 float channels. The original implementation makes use of geometry shaders to write into the individual layers of 3D textures. Instead of that, OpenGL's compute shaders are used to write into each layer

of a framebuffer directly. This approach directly bypasses the unnecessary vertex and geometry shader stages.

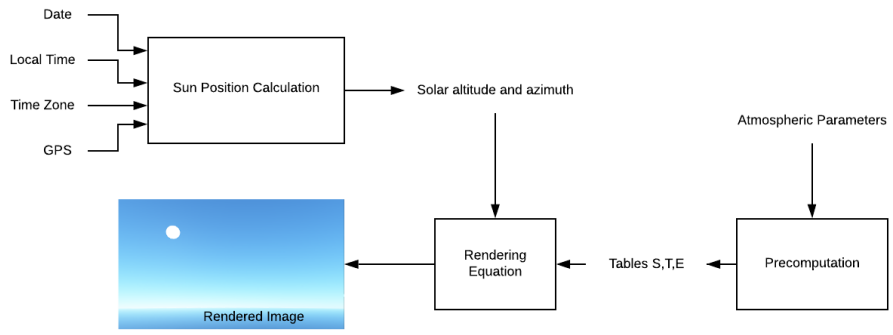


Figure 4.4: Rendering Pipeline of the Bruneton Model

As mentioned above, the precomputations are done via OpenGL's computation shaders and follow the same pipeline as in algorithm 0 (see Figure 4.4). The input for these shaders are atmosphere parameters which are as mentioned above baked into their source code. These parameters state the bottom and top radius of the atmospehere, solar irradiance, solar angular radius, scattering coefficients, absoption coefficient and so on. The outputs of these shaders are 2D and 3D textures (see Figure 4.5), which are used for rendering a final background image.

Transmittance shader. This shader computes the transmittance between points $\mathbf{x}$ and $\mathbf{y}$. This involves three integrals. The first one is the integral of the number density of air molecules along the segment $[\mathbf{x}, \mathbf{y}]$. Then there is the integral of the number density of aerosol particles along the segment $[\mathbf{x}, \mathbf{y}]$. Last but not least, the integral of the number density of air molecules that absorb light along the segment $[\mathbf{x}, \mathbf{y}]$. Results are stored into the 2D texture $\mathbb{T}$ (see Algorithm 2) [BN08].

Direct Irradiance shader. Direct Irradiance shader computes the direct irradiance (see Algorithm 3) and stores it in $\Delta\mathbb{E}$. It also prepares $\mathbb{E}$ for later computations by setting all cells to null. The direct irradiance is computed as the Sun radiance at the top of the atmosphere, tuned by transmittance. Since the solar solid angle is small, the transmittance can be approximated by a constant. Then the integral becomes equivalent to the ambient occlusion due to a sphere, also called view factor [BN08].

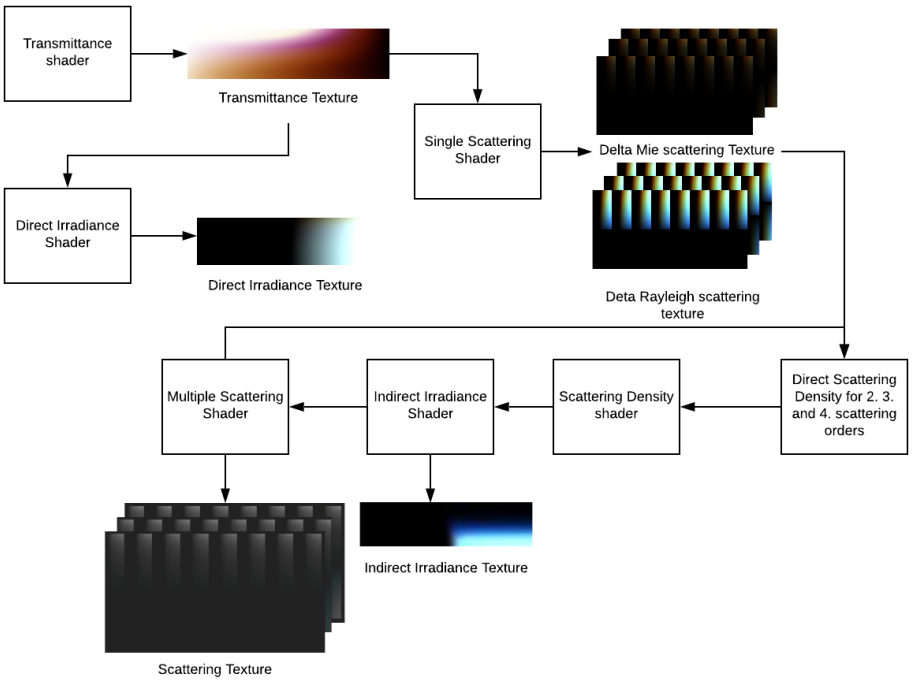


Figure 4.5: Diagram of the precomputation algorithm. Note that the output texture of the multiple scattering shader $\mathbb{S}$ has red value of the Mie scattering as alpha channel. Therefor the table has in fact clear background, but for clarity the background is set to black.

Single Scatteing shader. Computes the single Rayleigh and Mie scattering values and stores the values in Rayleigh scattering table $\Delta\mathbb{S}_R$ and Mie scattering table $\Delta\mathbb{S}_M$ and then stores them into $\mathbb{S}$ (see Algorithm 4).

Scattering Density shader. This shader evaluates the equation (2.9), which is needed for the calculation of multiple scattering. The shader computes the radiance, which is scattered at some point $\mathbf{x}$ inside the atmosphere, towards some direction $-\boldsymbol{\omega}$. The scattering event is also assumed to be the k-th bounce.

The radiance is integral over all incidental the possible directions ω_i, of the product of, the incidental radiance L_i arriving at $\mathbf{x}$ from direction $\boldsymbol{\omega}_i$ after k-1 bounces (see Algorithm 5)[BN08].

Indirect Irradiance shader. The indirect ground irradiance is computed numerically over all directions $\boldsymbol{\omega}$ of the hemisphere. The shader produces

Algorithm 2 $T'(r, \mu)$

Input:
$A \ldots$ **Atmosphere parameters**
Output:
$\mathbb{T} \ldots$ **Transmittance table**

procedure $OpticalLengthtoR_t(r, \mu, DensityLayer)$
  $N \leftarrow StepCount$ ▷ Number of intervals for numerical integration
  $StepSize \leftarrow DistanceToRt(r, \mu)/N$
  $Acc \leftarrow 0$
  for $(i \leftarrow 0$ to $N)$ **do** ▷ Integration Loop
    $d_i \leftarrow i * StepSize$ ▷ Distance from origin
    $r_i \leftarrow DistanceFromPlanetCenter(d_i, r, \mu)$
    $y_i \leftarrow DensityAtPoint(r_i - R_g, DensityLayer)$
    if $i = 0$ *or* $i = N$ **then** ▷ Trapezoidal Rule
      $Weight \leftarrow 0.5$
    else
      $Weight \leftarrow 1$
    $Acc \leftarrow Acc + y_i * Weight * StepSize$
  return Acc
$\mathbb{T} \leftarrow \exp\left(-\sum_{\{R,M\}} \beta_i^e * OpticalLengthToR_t(r, \mu, i)\right)$

two 2D tables: $\Delta\mathbb{E}$ of the k-th order and accumulated irradiance table $\mathbb{E}$ (see Algorithm 6).

Multiple scattering Shader. The Shader computes the k-th order of scattering for each point $\mathbf{x}$ and direction $\boldsymbol{\omega}$, the radiance coming from $\boldsymbol{\omega}$ after k bounces, using the table $\Delta\mathbb{J}$ precomputed in the previous shader [BN08] (see Algorithm 7). The results are scattering table $\Delta\mathbb{S}_k$ of the k-th order and accumulated table $\mathbb{S}_k$ of k orders.

Finally, the precomputed textures are used to render an environment map as a cube map with the size for each side 1024×1024 with 4 float channels. The formula bellow is evaluated for each pixel of the cube map.

$$L \simeq L'_{sun}(\mu, \mu_s, \nu_s)T(r, \mu_s) + \sum_{R,M} P'_i(\nu)S'(r, \mu, \mu_s, \nu_s) \tag{4.1}$$

The result of this equation is the final sky color, which is stored in our cube map. Since the implementation only renders the sky, it does not utilize shadow volumes. Therefore the implementation does not make use of the light shafts which the model is capable of.

Algorithm 3 $\mathcal{E}'[L'_0](r, \mu, \mu_s, \nu)$

Input:
$\mathbb{T}$ ... **Transmittance table**
A ... **Atmosphere parameters**
Output:
$\Delta\mathbb{E}_0$ ... **Direct Irradiance Table**

$\mathcal{E}_s \leftarrow SolarIrradiance$
$\alpha_s \leftarrow SunAngularRadius$
$T \leftarrow \mathbb{T}(r, \mu_s)$
$c \leftarrow AvarageCosFactor(\mu_s, \alpha_s)$
$\Delta\mathbb{E}_0 \leftarrow \mathcal{E}_s * T * c$

Algorithm 4 $\mathcal{S}'[L'_0](r, \mu, \mu_s, \nu)$

Input:
$\mathbb{T}$ ... **Transmittance table**
A ... **Atmosphere parameters**
Output:
$\Delta\mathbb{S}_{\mathbb{R}}$ ... **Single Rayleigh scattering table**
$\Delta\mathbb{S}_{\mathbb{M}}$ ... **Single Mie scattering table**
$\Delta\mathbb{S}_0$ ... **Combined scattering table of 0 order**

$N \leftarrow StepCount$ ▷ Number of intervals for numerical integration
if Ray intersects ground **then**
  $Step \leftarrow R_g/N$
else
  $StepSize \leftarrow R_t/N$
$Acc \leftarrow 0$
for $(i \leftarrow 0$ to $N)$ **do** ▷ Integration Loop
  $d_i \leftarrow i * StepSize$
  $r_{d_i} \leftarrow r_d(r, \mu, d_i)$ ▷ get r_{d_i}, μ_{d_i} and $\mu_{s_{d_i}}$ parameters at a current step
  $\mu_{d_i} \leftarrow \mu_d(r, \mu, d_i, r_{d_i})$
  $\mu_{s_{d_i}} \leftarrow \mu_d(r, \mu_s, d_i, r_{d_i})$
  if Ray intersects ground **then** ▷ Sample $\mathbb{T}$
    $T \leftarrow \mathbb{T}(r, -\mu)/\mathbb{T}(r_{d_i}, -\mu_{d_i})$
  else
    $T \leftarrow \mathbb{T}(r, \mu)/\mathbb{T}(r_{d_i}, \mu_{d_i})$
  $T_s \leftarrow \mathbb{T}(r_{d_i}, \mu_{s_{d_i}})$ ▷ Sample $\mathbb{T}$ to the sun
  $T \leftarrow T * T_s$
  $y_i \leftarrow T * DensityAtPoint(r_d - R_g)$
  if $i = 0$ *or* $i = N$ **then** ▷ Trepezoidal Rule
    $Weight \leftarrow 0.5$
  else
    $Weight \leftarrow 1$
  $Acc \leftarrow y_i * Weight$
$\Delta\mathbb{S}_l \leftarrow Acc * StepSize * SolarIrradiance * ScatteringCoefficient_l$ ▷ l is either Rayleigh or Mie. The algorithm is the same for both.
$\Delta\mathbb{S}_0 \leftarrow (\Delta\mathbb{S}_R.r, \Delta\mathbb{S}_R.g, \Delta\mathbb{S}_R.b, \Delta\mathbb{S}_M.r)$

Algorithm 5 $\mathcal{J}'[T\frac{\alpha'}{\pi}\Delta\mathbb{E} + \Delta\mathbb{S}](r, \mu, \mu_s, \nu)$

Input:
$\mathbb{T}$ ... **Transmittance table**
$\Delta\mathbb{S}_{\mathbb{R}}$ ... **Single Rayleigh scattering table**
$\Delta\mathbb{S}_{\mathbb{M}}$ ... **Single Mie scattering table**
$\Delta\mathbb{S}_{k-1}$ ... **Combined scattering table of n-1 order**
$\Delta\mathbb{E}_{k-1}$ **Irradiance table of (k-1)th order**
A ... **Atmosphere parameters**
Output:
$\Delta\mathbb{J}_k$ ... **Scattering density table of k-th order**

$N \leftarrow StepCount$
$Acc \leftarrow 0$
for i to N **do**
  $\theta \leftarrow i * \theta StepSize$
  if $RayIntesectsGround(r, cos(\theta))$ **then**
    $d_g \leftarrow DistanceToR_g(r, cos(\theta))$ ▷ Distance to the bottom boundry
    $T_g \leftarrow \mathbb{T}(r_{d_g}, -cos(\theta)_{d_g})/\mathbb{T}(r, -cos(\theta))$ ▷ Transmittance to the ground
    $\boldsymbol{\alpha_g} \leftarrow GroundAlbedo$
  else
    $d_g \leftarrow 0$
    $T_g \leftarrow 0$
    $\boldsymbol{\alpha}_g \leftarrow 0$
  for j to $2N$ **do**
    $\phi \leftarrow j * \phi StepSize$
    $\boldsymbol{\omega}_i \leftarrow SphericalCoords(\theta, \phi)$
    $d\omega \leftarrow \theta StepSize * \phi StepSize * \sin(\theta)$ ▷ SolidAngle
    $\nu_1 \leftarrow \boldsymbol{\omega}_s \cdot \boldsymbol{\omega}_i$
    if $k == 1$ **then**
      $incidentRadiance \leftarrow P_R(\nu_1) * \Delta S_R(r, \omega.z, \mu_s, \nu_1) + P_M(\nu_1) * \Delta S_M(r, \omega.z, \mu_s, \nu_1)$
    else
      $incidentRadiance \leftarrow S_{n-1}(r, \boldsymbol{\omega}_i.z, \mu_s, \nu_1)$
    $\mathcal{E}_g \leftarrow \Delta\mathbb{E}_{k-1}(R_g, \mathbf{n}_g \cdot \boldsymbol{\omega}_s)$ ▷ $Sample\Delta\mathbb{E}_{k-1}$ on the ground
    $incidentalRadiace \leftarrow incidentalRadiace + (T_g * \boldsymbol{\alpha}_g * \mathcal{E}_g)/\pi$
    $\nu_2 \leftarrow \boldsymbol{\omega} \cdot \boldsymbol{\omega_i}$
    $density_R \leftarrow GetDensity_R(r - R_g)$
    $density_M \leftarrow GetDensity_M(r - R_g)$
    $Acc \leftarrow Acc + d\omega * incidentalRadiance * \sum_{R,M}^{i} \beta_i^S * density_i * P_i(\nu_2)$
$\Delta\mathbb{J}_k \leftarrow Acc$

Algorithm 6 $\mathcal{E}'[T\frac{\alpha'}{\pi}\Delta E + \Delta S](r, \mu_s, \nu)$

Input:
$\mathbb{T}$ ... **Transmittance table**
$\Delta\mathbb{S}_{\mathbb{R}}$ ... **Single Rayleigh scattering table**
$\Delta\mathbb{S}_{\mathbb{M}}$ ... **Single Mie scattering table**
$\Delta\mathbb{S}_{k-1}$ ... **Combined scattering table of k-1 order**
$\mathbb{E}_{k-1}$ **Irradiance table of (k-1)th order**
A ... **Atmosphere parameters**
Output:
$\Delta\mathbb{E}_k$ ... **Irradiance table of k-th order**
$\mathbb{E}_k$ ... **Accumulated irradiance table of k-th order**

$N \leftarrow StepCount$ ▷ Number of intervals for numerical integration
$Acc \leftarrow 0$
$\phi StepSize \leftarrow \pi/N$ ▷ Integration over a sphere
$\theta StepSize \leftarrow \pi/N$
for i to $N/2$ **do** ▷ Integration cycle
  $\theta = i * \theta StepSize$
  for j to $2N$ **do**
    $\phi = i * \phi StepSize$
    $\boldsymbol{\omega} \leftarrow SphericalCoords(\theta, \phi)$
    $d\omega \leftarrow \theta StepSize * \phi StepSize * \sin(\theta)$ ▷ SolidAngle
    $\nu \leftarrow \boldsymbol{\omega} \cdot \boldsymbol{\omega}_s$
    if $k == 1$ **then**
      $y \leftarrow P_R(\nu) * \Delta S_R(r, \omega.z, \mu, \nu) + P_M(\nu) * \Delta S_M(r, \omega.z, \mu, \nu)$
    else
      $y \leftarrow S_{n-1}(r, \boldsymbol{\omega}.z, \mu, \nu)$
    $Acc \leftarrow Acc + y * d\omega * \boldsymbol{\omega}.z$
$\Delta\mathbb{E}_k \leftarrow Acc$
$\mathbb{E}_k \leftarrow \mathbb{E}_{k-1} + \Delta\mathbb{E}_k$

Algorithm 7 $\int_x^{x'_0} T(x,y)\Delta\mathbb{J}(||y||,\mu,\mu_s,\nu)dy$

Input:
$\mathbb{T}\ldots$ **Transmittance table**
$\Delta\mathbb{J}_k$ **Scattering density table of k-th order**
$\mathbb{S}_{k-1}\ldots$ **Scattering table of (k-1)th order**
$A\ldots$ **Atmosphere parameters**
Output:
$\Delta\mathbb{S}_k\ldots$ **Scattering table of k-th order**
$\mathbb{S}_k\ldots$ **Accumulated scattering table of k-th order**

$N \leftarrow StepCount$ ▷ Number of intervals for numerical integration
if $Rayintersectsground(r,\mu)$ **then**
  $StepSize \leftarrow R_g/N$
else
  $StepSize \leftarrow R_t/N$
$Acc \leftarrow 0$
for $(i \leftarrow 0$ to $N)$ **do** ▷ Integration Loop
  $d_i \leftarrow i * StepSize$
  $r_{d_i} \leftarrow r_d(r,\mu,d_i)$ ▷ Get r_d, μ_d and μ_{sd} for current integration step
  $\mu_{d_i} \leftarrow \mu_d(r,\mu,d_i,r_{d_i})$
  $\mu_{sd_i} \leftarrow \mu_d(r,\mu_s,d_i*\nu)$
  if $RayIntersectsGround(r,\mu)$ **then** ▷ Sample $\mathbb{T}$
    $T \leftarrow \mathbb{T}(r_{d_i},-\mu_{d_i})/\mathbb{T}(r,-\mu)$
  else
    $T \leftarrow \mathbb{T}(r,\mu)/\mathbb{T}(r_{d_i},\mu_{d_i})$
  $y_i \leftarrow T * \Delta\mathbb{J}(r_i,mu_i,mu_{si})$ ▷ Sample $\Delta\mathbb{J}_k$ and tune it by transmittance
  if $i = 0\ or\ i = N$ **then** ▷ Trepezoidal Rule
    $Weight \leftarrow 0.5$
  else
    $Weight \leftarrow 1$
  $Acc \leftarrow y_i * Weight$
$\Delta\mathbb{S}_k \leftarrow Acc$
$\mathbb{S}_k \leftarrow \mathbb{S}_{k-1} + \Delta\mathbb{S}_k$

Chapter 5

Results

This chapter presents results of this work's imprementations. Benchmarks were done on a PC running Windows 10, with Intel Core i7-8750H and NVIDIA GeForce GTX 1060. Results were obtained by averaging 100 iterations.

5.1 Skylight for VRUT

In this section are shown results of implementation of Hosek-Wilkie's model into the system VRUT compared to the results of the same sky-light model implemented using the OpenGL in the standalone application. Rendering of

Cube side's size	CPU rendering time[ms]	GPU rendering time[ms]
256×256	108.5	1.74
512×512	383.65	3.07
1024×1024	1564.39	6.96
2048×2048	6360.62	6.01
4096×4096	26079.375000	9.65

Table 5.1: Comparison between rendering times of the CPU implementation and the GPU implementation of Hosek-Wilkie's model for different cube map sizes.

the sky with varying times of the day, and values of turbidity and albedo are shown in figures A.1, A.2, A.3, A.4 and A.5.

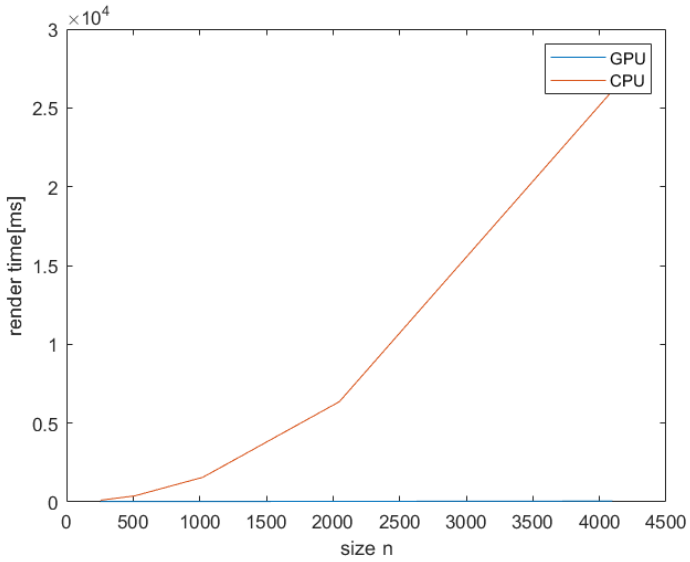


Figure 5.1: Plot of the table 5.1. As can be seen CPU rendering times grow exponentially with texture size but GPU rendering times increase linearly with minor deviations.

5.2 Bruneton model

In this section are shown results of implementation of Bruneton model in the standalone application. Rendering of the resolution 1920×1080 with cube map's side size 1024×1024 takes 3.64 ms. Precomputation phase takes 1138.03 ms. Offscreen rendering times of the environmental map for different cube map sizes can be seen in table 5.2 (see Figure 5.2). Rendering of the sky with observer positions and sun's altitudes are shown in figures A.9, A.6, A.7 and A.8.

Cube side's size	rendering time[ms]
256×256	0.89
512×512	2.87
1024×1024	4.48
2048×2048	10.58
4096×4096	30.05

Table 5.2: Off-screen rendering times of the enviromental map for different cube map sizes.

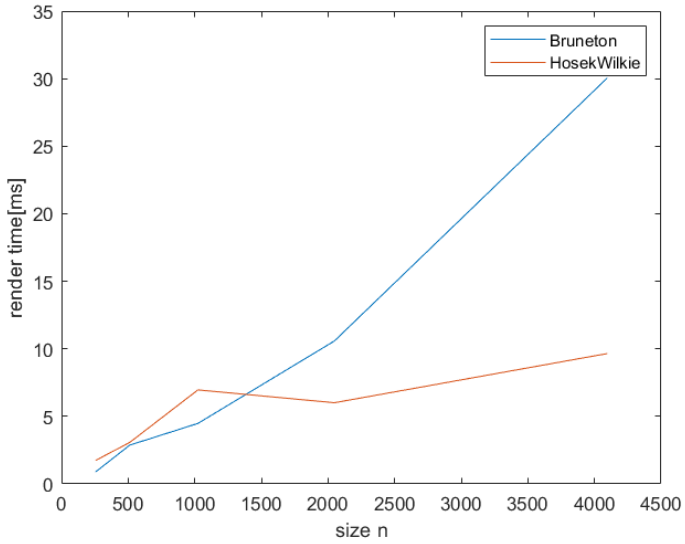


Figure 5.2: Off-screen GPU rendering times comparison between Hosek-wilkie and Bruneton skylight models for different cube map sizes. Both models' rendering times grow linearly with texture size.

Chapter 6

Conclusion

This thesis extends the library of VRUT's modules by a module providing a sky-light model that runs in real-time with low computational and memory complexity. The implemented module broadens the user's toolset for creating realistic outdoor scenes. The model was also implemented as a part of the standalone application using library OpenGL.

Furthermore the standalone application implements the Bruneton sky-light model that supports all observer positions, from the ground to space. The model has rendering complexity for each pixel O(1) with the precomputation phase, which is implemented using OpenGL's computation shaders.

6.1 Future Work

In the future, we would like to include the support for shadow volumes into the standalone application, which could be subsequently used for rendering light shafts. Possible future work might also be the inclusion of real-time rendered volumetric clouds.

For VRUT's DynamicSkylight module, the future implementations could try to find a better formula to obtain solar angles.

Bibliography

[AJP99] B. Smits A. J. Preetham, P. Shirley, *A practical analytic model for daylight*, In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (1999), 91–100.

[BN08] Eric Bruneton and Fabrice Neyret, *Precomputed atmospheric scattering*, Computer Graphics Forum, vol. 27, Wiley Online Library, 2008, pp. 1079–1086.

[Bru17] E. Bruneton, *A qualitative and quantitative evaluation of 8 clear sky models*, IEEE Transactions on Visualization and Computer Graphics (2017), 2641–2655.

[HW12] Lukas Hosek and Alexander Wilkie, *An analytic model for full spectral sky-dome radiance*, ACM Transactions on Graphics (TOG) **31** (2012), no. 4, 95.

[Kol12] Timothy R. Kol, *Analytical sky simulation*, `http://timothykol.com/pub/sky.pdf`, 12 2012.

[Kyb08] Václav Kyba, *Modulární 3d prohlížeč*, ČVUT FEL (2008).

[NAS] NASA, *Nasa earth image*, `http://fs.fish.govt.nz/Page.aspx?pk=7&sc=SUR`, (accessed: 2019-10-30).

[Nav] R. Nave, *Mie scattering*, `http://hyperphysics.phy-astr.gsu.edu/hbase/atmos/blusky.html`.

[O'N05] Sean O'Neil, *Accurate atmospheric scattering*, Gpu Gems **2** (2005), 253–268.

[YYCM06] Pat Hanrahan Yung Yu Chuang and Torsten Moller, *Volume and Participating Media digital image synthesis*, https://www.csie.ntu.edu.tw/~cyy/courses/rendering/06fall/lectures/handouts/lec12_volume.pdf, 2006, (accessed: 2019-10-30).

Appendix A

Results

Figure A.1: Sunrise for T = 5 $\alpha = 0.1$.

Figure A.2: Sundown for T = 5 $\alpha = 0.1$.

Figure A.3: Sun at zenith T = 5 $\alpha = 0.1$.

Figure A.4: Sun at zenith T = 5 $\alpha = 0.1$.

Figure A.5: Sundown for T = 3 $\alpha = 0.9$.

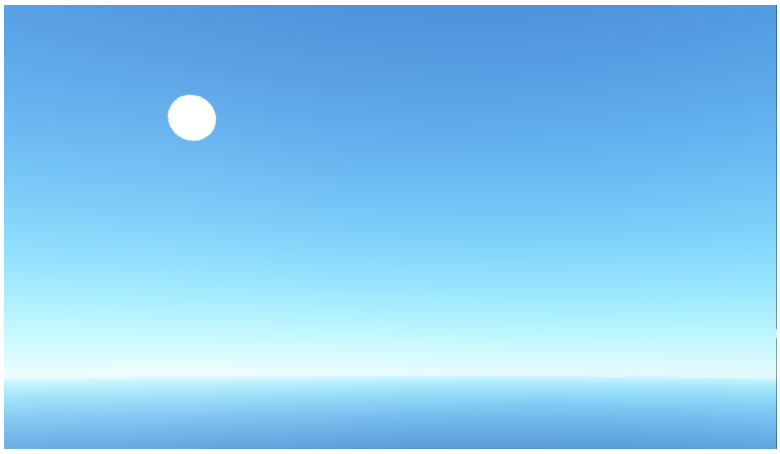


Figure A.6: Observer on the ground, altitude 30 degrees.

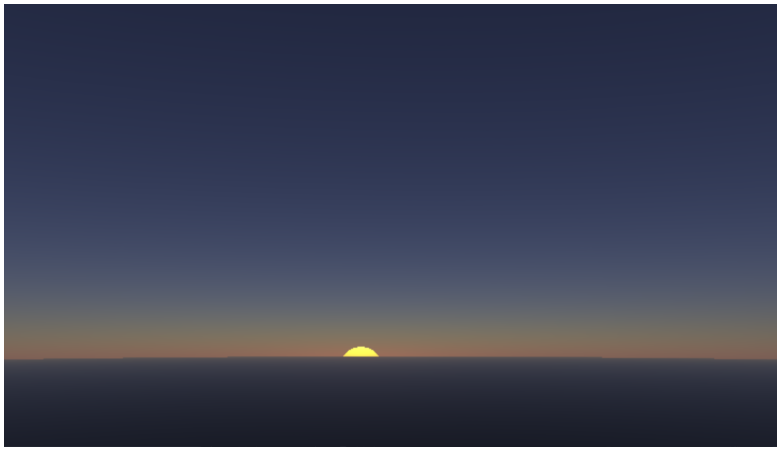


Figure A.7: Observer on the ground, altitude -2.03 degrees.

Figure A.8: Observer inside the atmosphere.

Figure A.9: Observer in space.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hvězda** Jméno: **Michal** Osobní číslo: **459888**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Model oblohy pro systém VRUT

Název bakalářské práce anglicky:

Daylight model for system VRUT

Pokyny pro vypracování:

Seznamte se s existujícími modely osvětlení oblohy [1-6]. Zaměřte se na modely schopné v reálném čase spočítat osvětlení/barvu celé oblohy (např. [2-4]). Prostudujte existující možnosti jak tyto modely obohatit o další efekty jako mraky či mlha.
Na základě prostudované literatury navrhněte a implementujte modul pro systém VRUT (Virtual Reality Universal Toolkit), který umožní začlenit model oblohy jak do path-traceru, tak do OpenGL rendereru v podobě mapy prostředí (environment map). Dále navrhněte a implementujte samostatnou aplikaci v OpenGL, která bude schopna zobrazit model oblohy pro zvolené datum, čas a pozici na zeměkouli. Aplikaci obohaťte o sadu GLSL shaderů pro implementaci vzdušné perspektivy a vrstvy mraků [6,7].
Funkčnost vytvořené aplikace ověřte alespoň na třech různých scénách a výsledky porovnejte s reálnými fotografiemi oblohy.

Seznam doporučené literatury:

[1] E. Bruneton: A Qualitative and Quantitative Evaluation of 8 Clear Sky Models. IEEE Transactions on Visualization and Computer Graphics, 23(12), 2017, p.2641-2655.
[2] L. Hošek, A. Wilkie:.Adding a Solar-Radiance Function to the Hošek-Wilkie Skylight Model. IEEE Computer Graphics and Applications, 33(3), 2013, p.44-52.
[3] L. Hošek, A. Wilkie:.An Analytic Model for Full Spectral Sky-dome Radiance. ACM Transactions on Graphics (TOG), 31(4), 2012, Article No. 95.
[4] A. J. Preetham, , P. Shirley, B. Smits:. A Practical Analytic Model for Daylight. In Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, 1999, pp. 91-100, ACM Press/Addison-Wesley Publishing Co..
[5] H.W. Jensen, F. Durand, J. Dorsey, M.M. Stark, P. Shirley, S. Premože: A physically-based night sky model. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques (pp. 399-408). 2001, August, ACM.
[6] D. Müller, J. Engel, J. Döllner: Single-Pass Rendering of Day and Night Sky Phenomena. In Proceedings of the Vision, Modeling, and Visualization Workshop 2012, pages 55–62, Eurographics Association, 2012.
[7] T.Kment, M. Rauter, G. Zotti: Modelling of Daylight for Computer Graphics. Technical report, Institut für Computergraphik und Algorithmen, Wien, 2006.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Jaroslav Sloup, Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **14.02.2019** Termín odevzdání bakalářské práce: **07.01.2020**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Jaroslav Sloup podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta