Physically Based Sky, Atmosphere and Cloud Rendering in Frostbite

Sébastien Hillaire
EA Frostbite

Contents

1 Introduction

Video games are becoming more and more demanding in terms of visual quality and dynamism. Open world games, for instance, may require lots of dynamic elements, such as *time of day* lighting and real-time evolving weather. The dynamic and often global nature of these elements make them difficult to simulate and render in real time. Sky, atmosphere and clouds are the three main components we need to simulate in order to achieve dynamic time of day and weather conditions. They are difficult to render due to their very detailed and specific volumetric nature. These elements also interact together, such as clouds affecting atmospheric lighting and vice versa.

In this document, we present the practical physically based solutions we have researched and developed for Frostbite to simulate and combine of all these elements, as well as to render the complex interactions between them. We will also describe how this fits into Frostbite's physically based shading framework as well as how artists author such elements (for both released and currently in production titles), along with their performance characteristics.

1.1 Context

In 2014, Frostbite[1] was significantly evolved to become a physically based rendering engine [LR14]. Among other things, this resulted in decoupled light and material definitions, which changed the way they are specified by artists as well as how they interact. These changes permeated all the way up to the camera receptors and the way that lighting information is transformed and presented on screen. This resulted in a huge increase in visual quality, as demonstrated in Figure 1. It can be hard to tell which picture is real or computer generated: left or right?

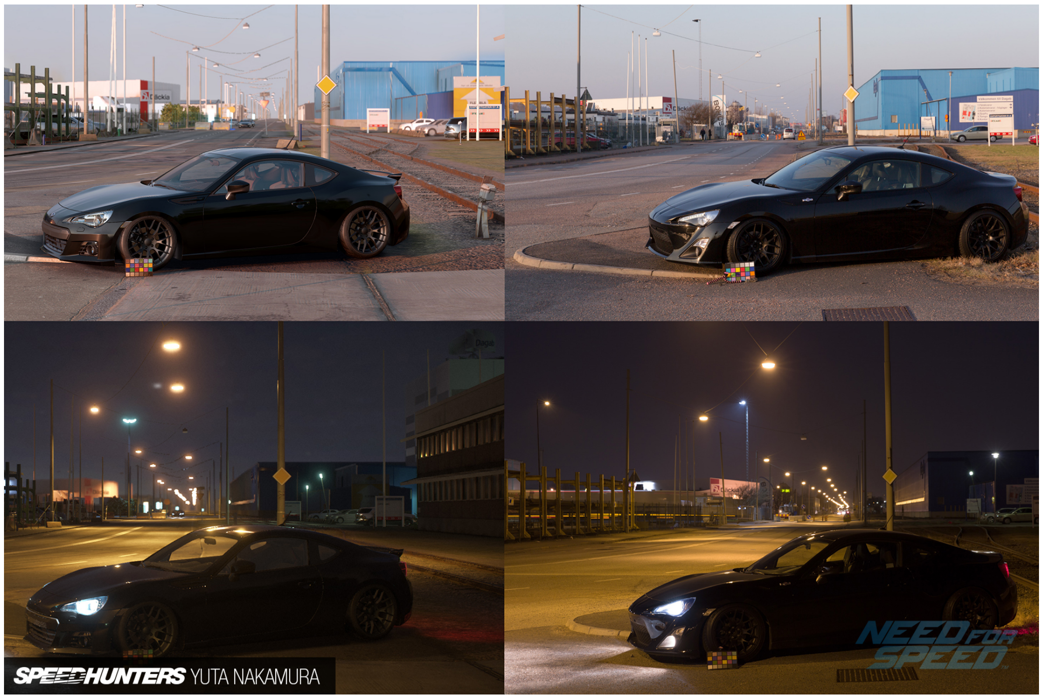


Figure 1: Need for speed [Gho17] comparisons: *reality* versus *Frostbite engine*. Can you guess which is which?

Taking a physically based approach with decoupled lighting and material information means that once a material has been authored, it should look consistent under any lighting condition. This is

[1]Frostbite is a game development platform that powers most titles at Electronic Arts (EA). See `http://www.frostbite.com` for details.

especially relevant to us now that a lot of EA titles are more *open world* in nature, with time of day and dynamic weather. However, at the time, Frostbite's sky, fog, cloud and participating media systems were still very static elements that could only be linearly blended. These elements included the panoramic sky texture, sky color gradient, fog color, fog curves, etc. It was time consuming for artists to ensure that all of these colors and curves were consistent for each time of day, especially as each state can be linearly interpolated and contain high dynamic range content. It is also lots of time to invest when re-authoring is required after art direction feedback. There was nothing inherently wrong with these techniques since they did allow games such as *Battlefield 4* and *Star Wars Battlefront* to looks gorgeous (Figure 2). That being said, more advanced solutions were needed to enable games to support more dynamic use cases.

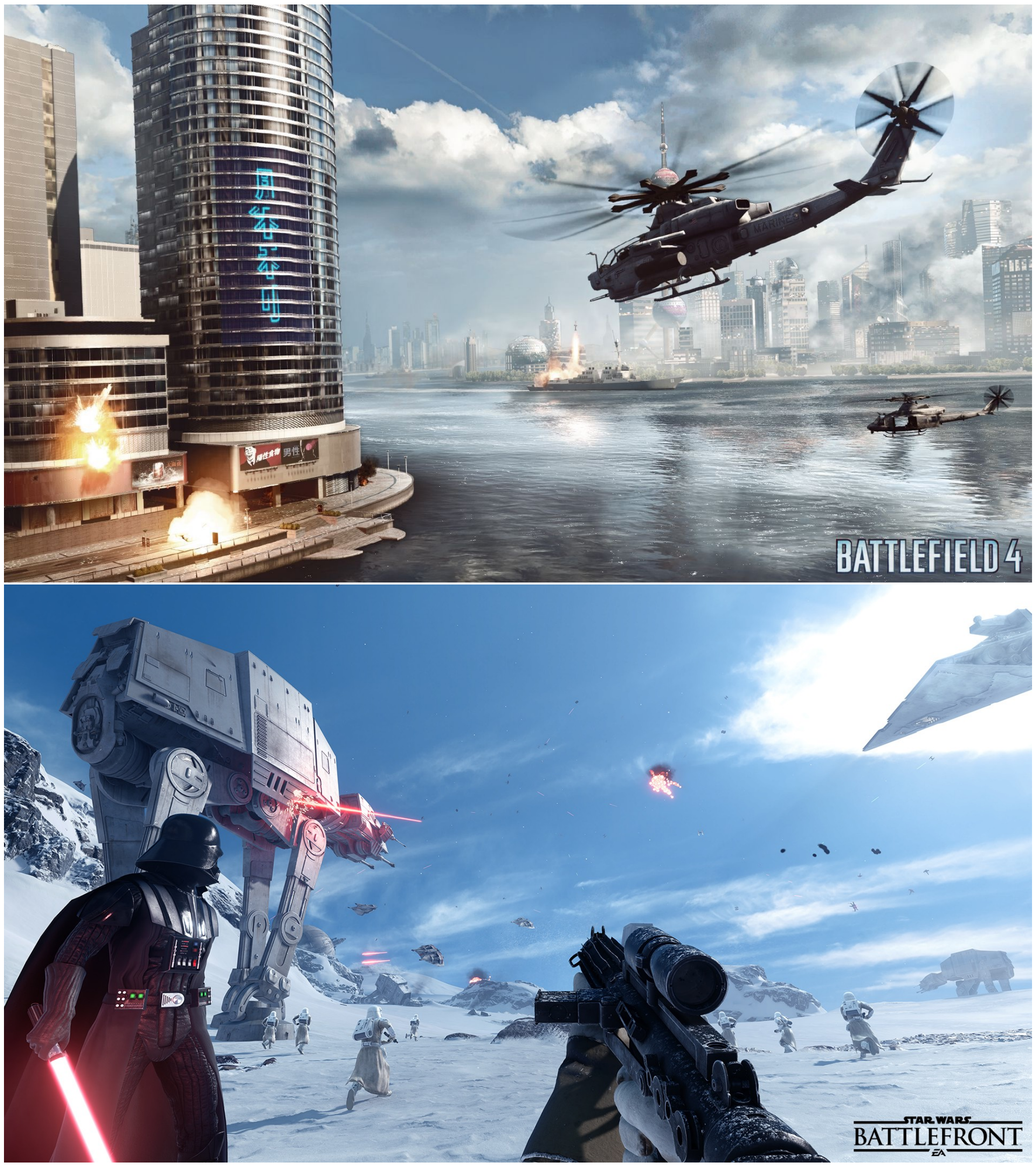


Figure 2: Top and bottom: Battlefield 4 [DIC13] and Star Wars$^{\text{TM}}$ Battlefront [DIC15].

1.2 Scope and objective

Figure 3 represents what we wanted game artists to be able to achieve in Frostbite:

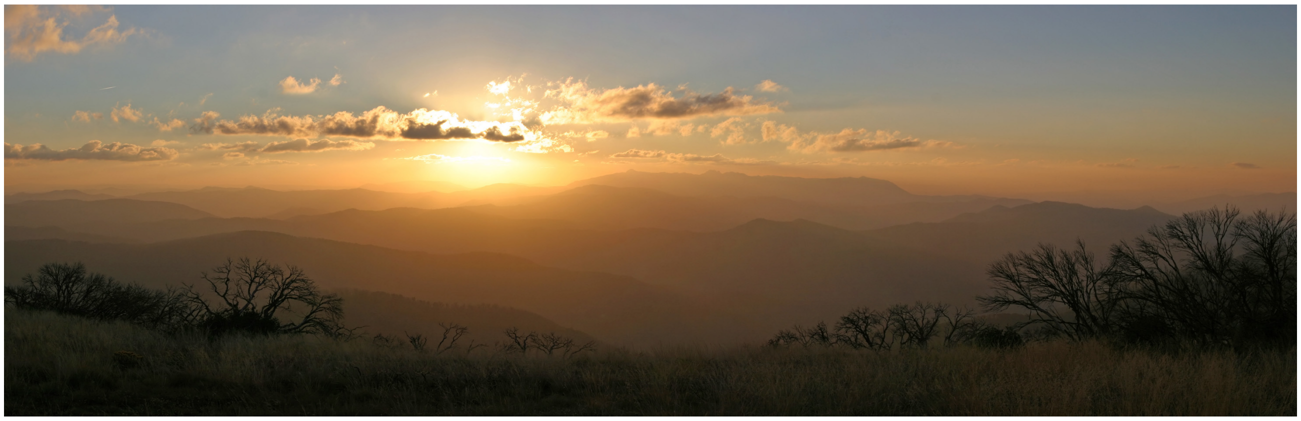


Figure 3: Aerial perspective photo featuring sky and atmosphere scattering, sun and clouds. (Photo by David Iliff. License: CC-BY-SA 3.0.)

- a realistic sky supporting dynamic time of day
- dynamic lighting according to the sun position
- atmospheric scattering
- clouds evolving according to the weather

We wanted these techniques to be scalable in order to be used by both 30 and 60 FPS titles. We also wanted these new techniques to fit within Frostbite's physically based framework. This meant decoupling lighting parameters from material representation, and using physically based parametrizations to represent real physical properties of participating media materials.

Additionally, many EA games need to reach a stable 60 frames per second, with high visual quality. This is already challenging by itself but shipping games with that constraint while having dynamic time of day and weather is a lot more complex. Especially when you want all of the different systems to stay synchronized between each other.

Also, we wanted all these visual features to interact together coherently, and be as realistic and unified as possible by default, across participating media, opaque and transparent materials. For instance, clouds becoming larger and thicker should affect the light scattered through the atmosphere, as well as sun shadows, which in turn also affects global illumination and thus global reflection on transparent surfaces, etc.

1.3 Contributors

The work presented in this document is the result of many collaborations between Frostbite, master students as well as game team engineers and artists.

Concerning the sky and atmosphere systems, we would like to thanks Gustav Bodare (Ghost) and Edvard Sandberg (Ghost) who were master student in 2014, Bioware (Mass Effect Andromeda^{TM}), DICE (Mirror's Edge Catalyst^{TM}), Ghost (Need for Speed^{TM}).

Concerning the volumetric cloud systems, we would like to thanks Rurik Högfeldt (2015 Master student) as well as Bioware: Marc-Andre Loyer (Programmer), Soren Hesse (Tech Environment Art) and Don Arceta (Lead Environment Art).

Thanks to Per Einarsson, Charles de Rousiers and Tomasz Stachowiak and the Frostbite rendering team for all the discussion and help about those rendering techniques.

Thanks to the rendering research community for sharing such incredible knowledge, useful research and practical result. And with source code sometime! Please continue to do so. Thanks to Fabrice Neyret, Eric Bruneton and Antoine Bouthors for the discussion around sky and cloud rendering.

Last but not the least, thanks To Stephen Hill and Stephen McAuley for their help during the SIGGRAPH 2016 *Physically Based Shading in Theory and Practice* course [MH16] preparation, presentation and the review of this document.

2 Participating Media

Participating media is the term used to describe volumes filled with particles. Such particles can be large impurities, *e.g.* dust, pollution, water droplets, or simply particles, *e.g.* molecules. Depending on its composition, the media will interact differently with light traveling through it and bouncing on particles, typically referred to as *light scattering.* The density of particles per volume can also vary spatially and take different form at different scale. For instance, in the case of water droplets, a wide and light density distribution volume could represent uniform fog, *i.e.* homogeneous media, whereas locally dense volume could represent clouds, *i.e.* heterogeneous media.

In this document, we will focus on participating media represented with spherical particles of varying radius. We will not discuss about micro flakes representation and theory. This means we assume isotropic participating media, *i.e.* scattering probability does not depend on incoming light direction, while still allowing asymmetric scattering, i.e. the scatter amount depends on the incoming light direction and is driven by the phase function (see Section 2.3). We will also ignore participating media emitting light, *e.g.* black body simulation. The notations used in this document are presented in Table 1.

Symbol	**description**	**unit**
σ_a	absorption coefficient	m^{-1}
σ_s	scattering coefficient	m^{-1}
σ_t	extinction coefficient	m^{-1}
ρ	albedo	unitless
p	phase function	sr^{-1}
L	luminance	$cd.m{-}2$
$L(x, \omega)$	luminance at point x in direction ω	$cd.m{-}2$
E	illuminance	$cd.sr^{-1}.m{-}2$

Table 1: Notation used in this document.

In this section, we present the theory behind participating media volumetric lighting and shadowing. We will also present the different parameters as well as how they interact and influence the final result. The goal is to help you understand the behaviour of each of the different parameters.

To go further, one can also visit the **ShaderToy** website to get access to volumetric scattering and shadowing example code (See Figure 4). Frostbite is using such code to simulate volumetric light interactions.

Figure 4: Shader toy example presenting volumetric scattering and shadowing [Hilb][Hilc].

2.1 Single scattering

We will focus on the simulation and rendering of *single scattering* in participating media. In case you are still hungry for more after this chapter, your could read the following very well written resources: [PH10], [dEo16] or [Wre11]. One of the best explanation of scattering event is accessible in chapter 4 of [Jar08].

Considering light traveling straight through a medium, different events can make the outgoing luminance be different as compared to the luminance that went in:

- **absorption** σ_a: photons are absorbed by the medium matter.
- **out-scattering** σ_s: photons are scattered away by bouncing off particle in the medium matter. This will be done according to the phase function p describing the distribution of bounce direction (See Section 2.3).
- **in-scattering** σ_s: photons can scatter in the current light path after bouncing off particles and contribute to the final luminance. This will also be done according to the phase function p.
- **emission**: light can be emitted when media reach high heat, *e.g.* fire. We ignore this component in this document.

To sum it up, adding photons on a path is a function of σ_s and removing photon is a function of extinction $\sigma_t = \sigma_a + \sigma_s$ representing both absorption and out-scattering.

Each of these events are wavelength dependent. It means that the way different light frequencies will be absorbed or scattered with different probabilities. For the sake of real-time efficiency, we will only simplify this by considering the red ($680nm$), green ($550nm$) and blue ($440nm$) light spectrum. The final per spectrum luminance integration can be performed using equation 1 considering punctual lights. A sketch presenting the different components is visible in Figure 5.

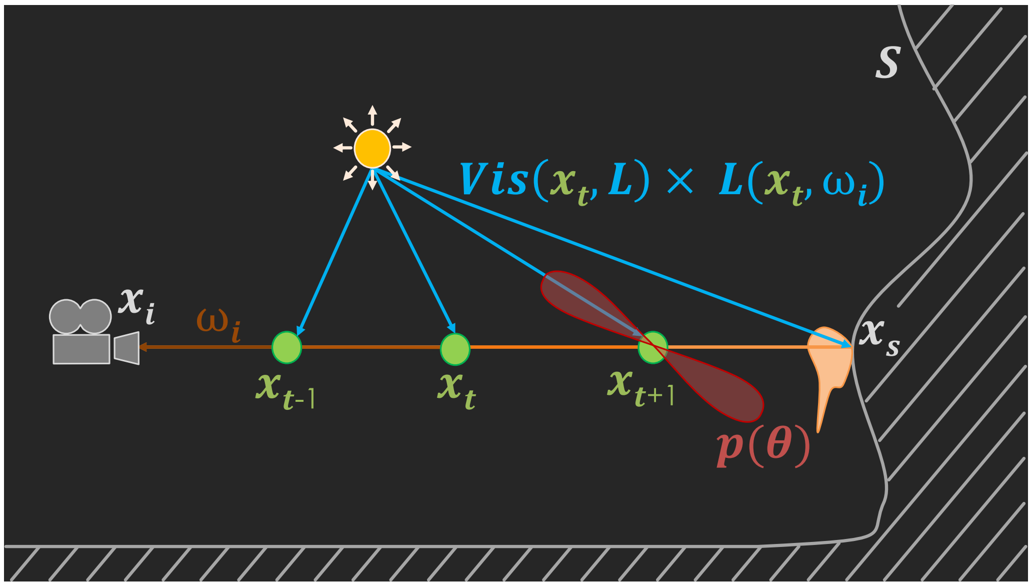


Figure 5: Sketch illustrating the integration of scattered light in a virtual world with a single point light using parameters and equations presented in this section.

$$L_i(x_i, \omega_i) = Tr(x, x_s)L(x_s, \omega_i) + \int_{t=0}^{S} Tr(x, x_t)Lscat(x_t, \omega_i)\sigma_s dt \qquad (1)$$

The **transmittance** $Tr(x, x_t)$ in Equation 2 is a function of extinction. The higher the extinction, or the distance, the higher the **optical depth** $\tau = \sigma_t(x_t)dt$ will be and, in turn, the less light will travel through the medium section. The behaviour of the transmittance function is presented in Figure 6. Transmittance needs to be applied on (1) the luminance $L(x_s, \omega_i)$ from opaque surface, (2) the luminance $Lscat(xt, \omega_i)$ resulting from an **in-scattering** event and also (3) each path from a scattering event to the light source. (1) will result in some visual fog-like occlusion of surface, (2) will result in self-occlusion of participating media and (3) will result in volumetric shadows within the participating media. Since $\sigma_t = \sigma_a + \sigma_s$, it is expected that the transmittance is influenced by both the scattering, *i.e.* out-scattering, and absorption components.

$$Tr(x_a, x_b) = \exp(-\int_{x=x_a}^{x_b} \sigma_t(x)dt) \tag{2}$$

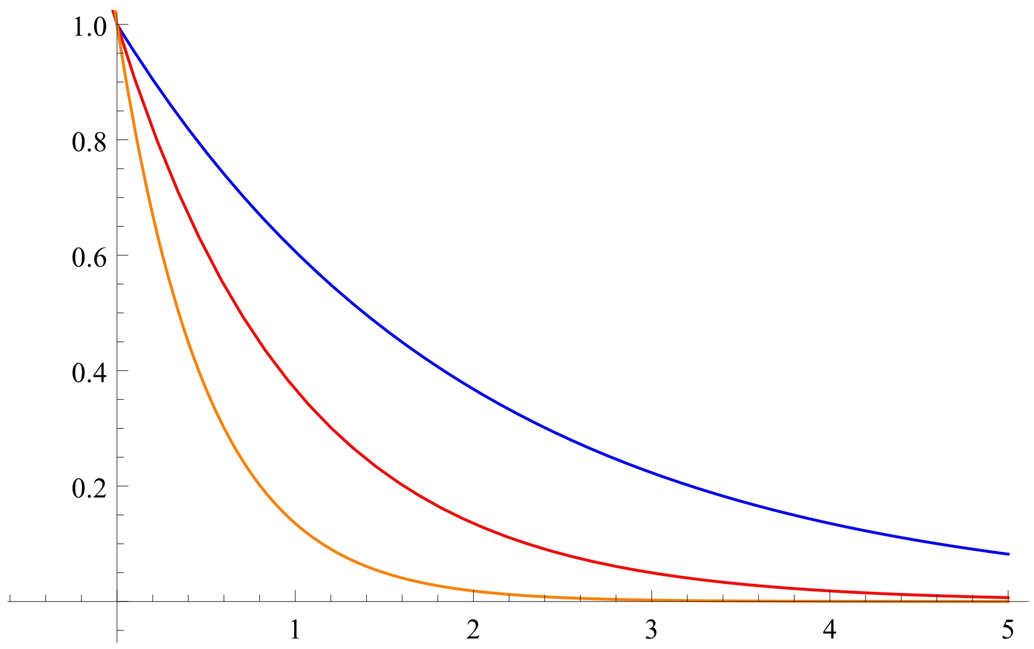


Figure 6: Transmittance function as a function of depth and extinction $\sigma_t = 0.5$, $\sigma_t = 1.0$ and $\sigma_t = 2.0$.

A single **in-scattering** event is represented by $Lscat(x, \omega_i)$ in Equation 3, describing the amount of luminance scattered back to a direction according to all the punctual light sources of a scene, the visibility function $Vis(x, L)$ as well as the phase function p described in Section 2.3.

$$Lscat(x, \omega_i) = \sum_{i=0}^{lights} p(\omega_i, L)Vis(x, L)L_i(x, L) \tag{3}$$

The **visibility** function $Vis(x, L)$ from Equation 4 represents the amount of light reaching the light source. For instance, if an opaque object is sitting in between a light source and the sample point x, thus occluding the sample, the $shadowMap(x, L)$ will simply return 0 instead of 1 (in the case of an infinitesimally small punctual light, ignoring soft shadow). This is traditionally achieved using hardware shadow mapping relying light view depth textures.

The $volumetricShadow(x, L)$ represents the transmittance from the sample point x to the light position x_L with values in $[0, 1]$ per wavelength, thus allowing the participating media to self-shadow. This is usually achieved using secondary ray marching toward each light source. As a quality performance trade-off, specific volumetric shadow sampling/storage techniques [Hil15][JB10][Sal+10] can be used to store transmittance for out-going direction from a light.

$$\begin{aligned} Vis(x,L) &= shadowMap(x,L) * volumetricShadow(x,L) \\ volumetricShadow(x,L) &= Tr(x,x_L) \end{aligned} \tag{4}$$

2.2 Albedo

The albedo Equation 5 is a value representing the relative importance of scattering relatively to absorption in a medium for each considered spectrum band. The value is within the $[0,1]$ range:

- An albedo close to 0 indicates that most of the light is absorbed, resulting in a very dark medium (*e.g.* dark exhaust smoke)
- An albedo close to 1 indicates that most of the light is scattered instead of being absorbed, resulting in a brighter medium (*e.g.* air, cloud or earth atmosphere)

$$\begin{aligned} \rho &= \sigma_s/(\sigma_s + \sigma_a) \\ \rho &= \sigma_s/\sigma_t \end{aligned} \tag{5}$$

2.3 Phase function

As mentioned before, a participating medium is composed particle with varying radius. The distribution of these particles radius will influence the distribution of light scattering direction at any point within participating media. Describing such probability distribution is achieved using a phase function used when evaluating in-scattering as shown in equation 3.

$$x = \frac{2\pi r}{\lambda} \tag{6}$$

A phase function will change the in-scattering at a point x as a function of the directional luminance information reaching that point. Different types of scattering can be identified from x the **relative size** of a particle as defined by equation 6 where r is the particle radius and λ the considered wavelength [Hul57][Wikg]:

- $x \ll 1$: For example Rayleigh scattering (*e.g.* air)
- $x \approx 1$: Mie scattering
- $x \gg 1$: Geometric scattering

More components can influence the scattering result such as the index of refraction or the participating media content, *etc.* We will ignore them for this version of the document.

2.3.1 Isotropic scattering phase

In this case, light will be scattered uniformly in all direction. Surely not a very realistic scenario but it is commonly used due to its simplicity. The phase function is presented in equation 7 where θ is the angle between incoming light direction and out scattering direction.

$$p(\theta) = \frac{1}{4\pi} \tag{7}$$

2.3.2 Rayleigh scattering phase

$$p(\theta) = \frac{3}{16\pi}(1 + \cos^2\theta) \tag{8}$$

Rayleigh derived expressions for the scattering of light off molecules in the air [Ray71]. For instance, Rayleigh scattering is used to describe light scattering happening in the earth atmosphere and is reported as having very low to no absorption. This phase is a two-lobe function as visible in Figure 7 and can be evaluated using equation 8.

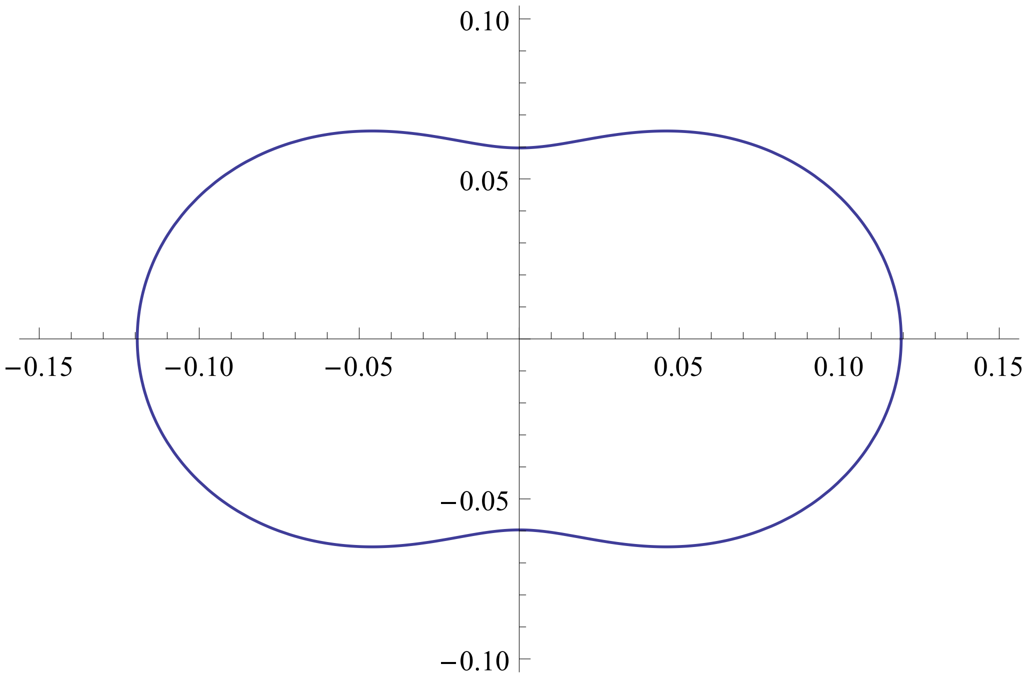


Figure 7: Polar plot of the Rayleigh phase function.

Rayleigh scattering is also highly dependent on the wavelength of light. This is represented by a wavelength dependent scattering coefficient σ_s equation presented in [Jar08]. An approximation commonly used is to set constant σ_s for each of the reduced R, G and B light spectrum band commonly used when evaluating/rendering such scattering events.

2.3.3 Mie scattering phase

Mie scattering [Mie08] is the model that can be used when the size of particles is similar to the light wavelength. However, Mie scattering is complex to simulate and requires many power functions.

An alternative is to use the Henyey-Greenstein phase function 8 with g in $]0,1[$. It has been proposed to represent the scattering by interstellar dust. It can also be used to represent any smoke, fog or dust like participating media. Such media can exhibit a very strong backward or forward scattering resulting in large visual halos around light sources, *e.g.* spot lights in fog, or the strong silver lining effect at the edge of clouds in the sun direction.

$$p_{hg}(\theta, g) = \frac{1 - g^2}{4\pi(1 + g^2 - 2g\cos\theta)^{1.5}} \tag{9}$$

This Henyey-Greenstein phase function can feature more complex shape than Rayleigh scattering and is evaluated using Equation 9. It can result in varied shape as shown in Figure 8. The strength of forward or backward scattering is controlled using the g parameter.

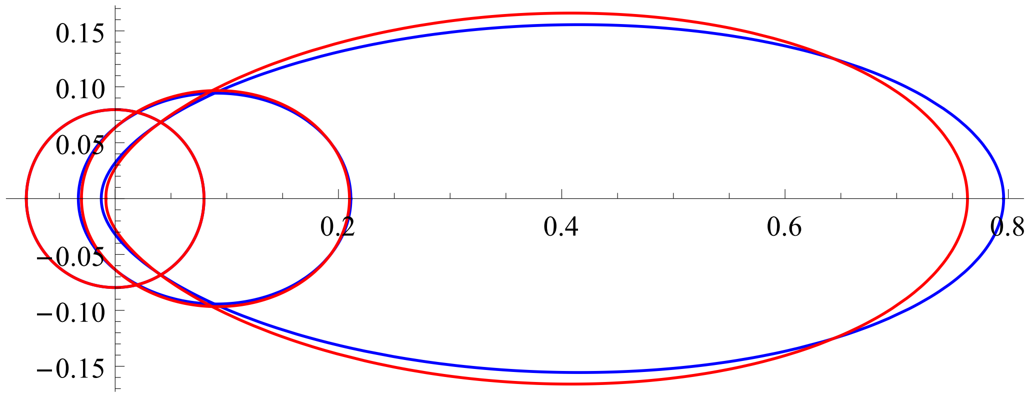


Figure 8: Polar plot of the Henyey-Greenstein and Schlick appoximation phase functions.

A fast way to approximate the Henyey-Greenstein phase function is to use an approximation proposed by Schlick. This equation 10 does not feature any complex power function but instead only a square which is a lot faster to evaluate. To be able to map that function onto the original Henyey-Greenstein phase function, the k parameter need to be computed using g. This only has to be done once for participating media having a constant g value. It is interesting to note that for very strongly positive and negative g values, the error can become quite large (see Figure 9) and result in lower silver lining effect.

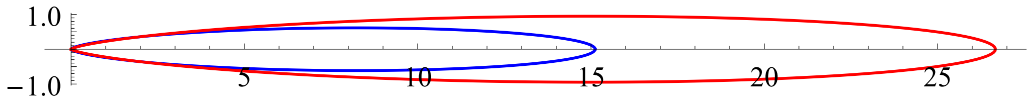


Figure 9: Polar plot of the Henyey-Greenstein and Schlick appoximation phase functions showing lager error for large g values.

$$\begin{aligned} k &\approx 1.55g - 0.55g^3 \\ p(\theta, k) &= \frac{1-k^2}{4\pi(1+k\cos\theta)^2} \end{aligned} \tag{10}$$

2.3.4 Geometric scattering phase

Geometric scattering happens for very large particles. In this case, light can refract and reflect within each particles. This can result in complex scattering phase function also depending on light polarisation. For instance, a real life example of that, is the visual rainbow effect. It is caused by internal reflection of light inside water particles in the air, dispersing the sun light into a visible spectrum on a small visual angle (≈ 3 degrees) of the resulting backward scattering.

Such complex phase function can be evaluated using MiePlot software [Lav15]. This software uses the Mie scattering theory, Debye series and ray tracing to evaluate phase functions. The resulting phase function can be visualised and output into a file for usage in your applications. As shown in Figure 10.

2.4 Examples

This Section presents the different components of volumetric rendering and how they can influence the final visual look of a volume. If you are starting experimenting with these types of algorithm, that

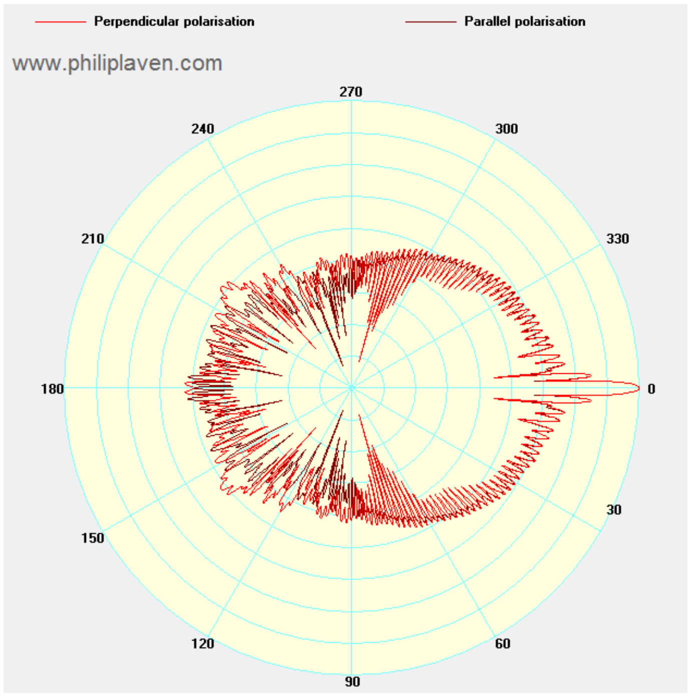


Figure 10: Exemple of a complex phase function generated using MiePlot [Lav15].

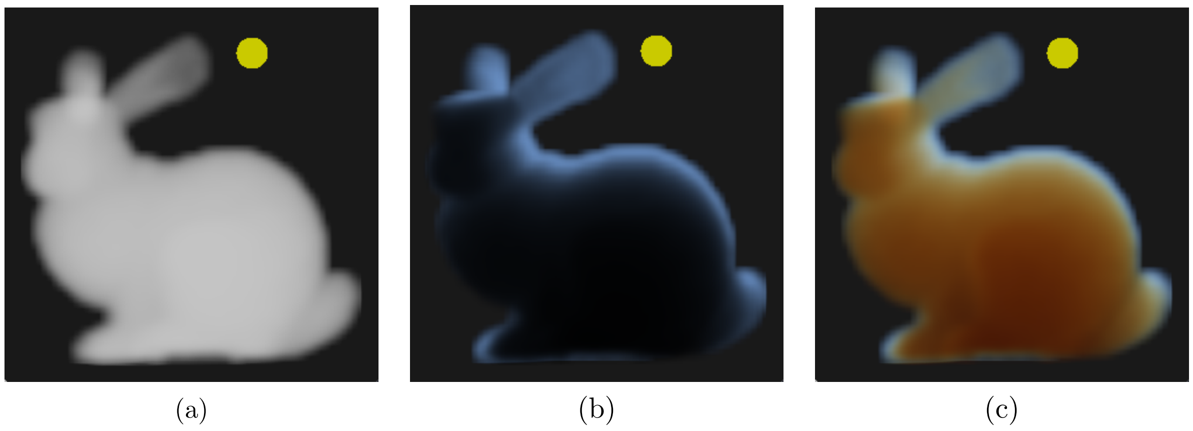
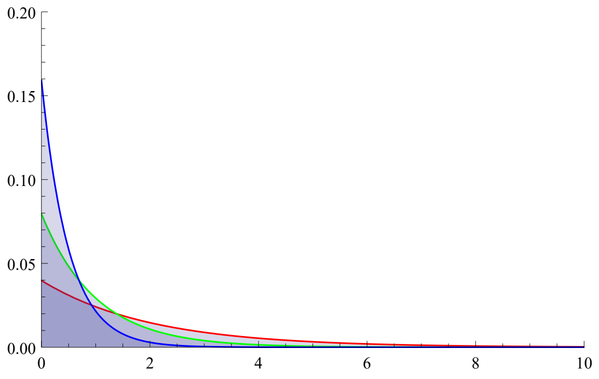


Figure 11: Enabling different volumetric rendering components: (a)$\sigma_s = 1$ scattering without volumetric shadow, (b) $\sigma_s = \{0.5, 1.0, 2.0\}$ RGB scattering with grey volumetric shadows achieved with $\sigma_t = mean(\sigma_s)$ and (c) with colored volumetric shadow achieved with $\sigma_t = \sigma_s = \{0.5, 1.0, 2.0\}$. In each examples, absorption has been set to 0 $\sigma_a = 0$. (a) and (b) are not physically correct but used here to present component contributions.

Section should hopefully give you a good intuition of the role and influence of the different parameters. If you think anything is missing, please do get in touch [2] and the section will be updated.

Figure 11 presents different components being enabled separately. Image (a) shows a Stanford bunny shaped participating medium under a white light and white scattering $\sigma_s = 1$. Image (b) shows the same medium but now with $\sigma_s = \{0.5, 1.0, 2.0\}$. Since the blue color scatters more, the bunny has an overall blue color. The volumetric shadow is a grey shadow evaluated using equation 4. The grey scale shadow is achieved using $\sigma_t = 2.0$. This is just to show the addition of volumetric shadow, even though this is physically incorrect. To be physically correct, volumetric shadow should be evaluated using $\sigma_t = \sigma_s + \sigma_a = \{0.5, 1.0, 2.0\}$ (given the fact that $\sigma_a = 0$ in this case). The resulting visual is

[2]sebastien.hillaire@frostbite.com



(a)

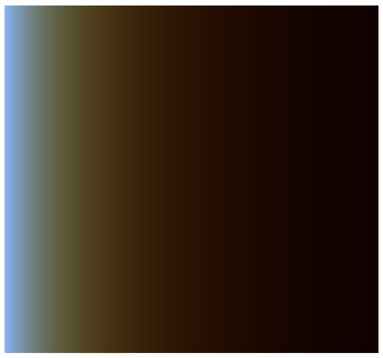

(b)

Figure 12: (a) graph representing the amount of light scattered after traveling through a uniform medium, taking into account transmittance and assuming a niform phase function, over a distance represented by the x axis. (b) The corresponding color gradient visualization of the graph.

presented in Figure 11 (c). At the interface between air and the medium, when light has not traveled a long path, blue light is scattered more resulting in a blue color. For long light path deep from the entry point, the blue light has been scattered more. As a result, only the other components remain: that is why the red color is then more visible in this material configuration. This behavior is in fact similar to the one of light traveling in the atmosphere. It only happens at a larger scale because the concentration of air molecule in the atmosphere is a lot lower (see Section 3). In order to illustrate that idea using the same coefficient, we give the one-dimensional scattering profile in Figure 12 as a graph for R, G and B wavelength and color gradient.

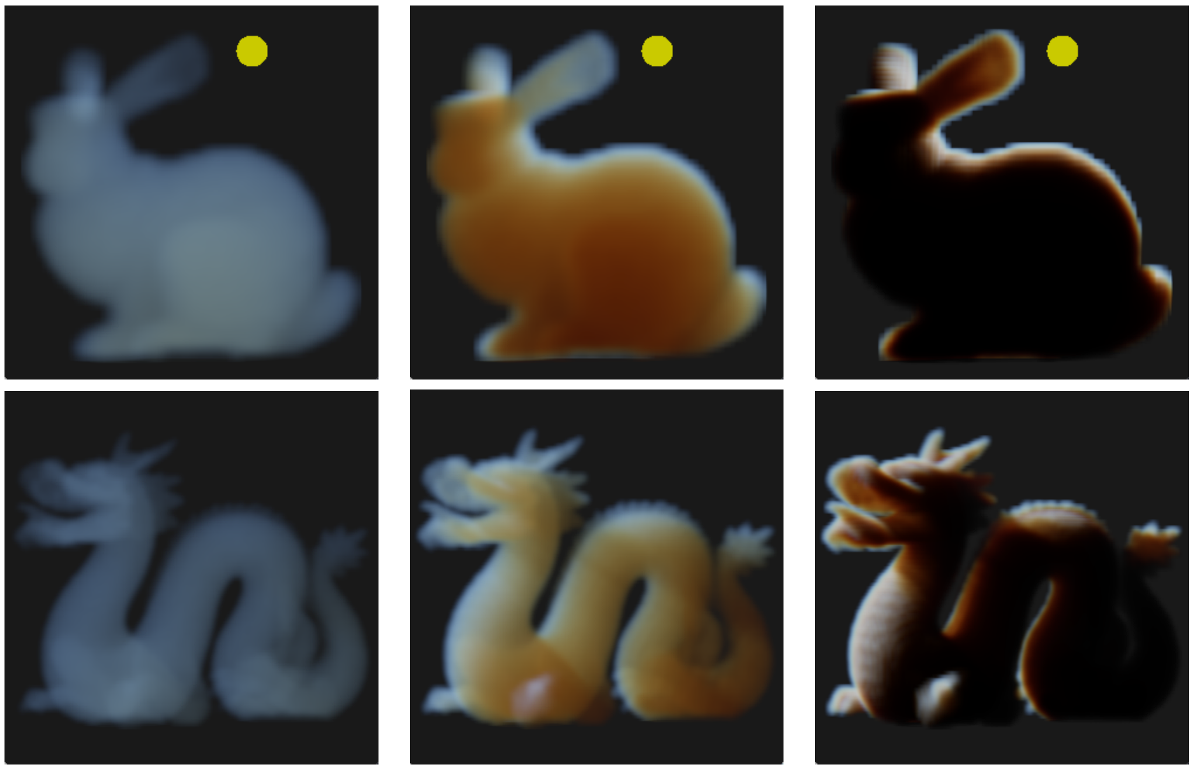


Figure 13: Stanford bunny and dragon with increasing density (from left to right: 0.1, 1.0 and 10.0) where $\sigma_s = \{0.5, 1.0, 2.0\}$.

Figure 13 presents Stanford bunny and dragon with the same participating media material ($\sigma_s = 0.5, 1.0, 2.0$) but varying densities. For low density media, scattering coefficient will give the dominant participating medium color. When the density is increased, the resulting behaviour will be more complex as described previously, *i.e.* due to absoprtion and out-scattering. In the very dense case for the rightmost picture, the objects starts to almost look like flesh/skin. This is because our skin also has similar overall characteristic: red light travel further within our flesh. Skin is also a more complex volumetric material with many layers and thus can be expensive to evaluate using ray marching. Instead it is usually approximated with *cheaper* diffusion profiles ignoring any form of single scattering phase function or anisotropy [Jen+01b] [DL07].

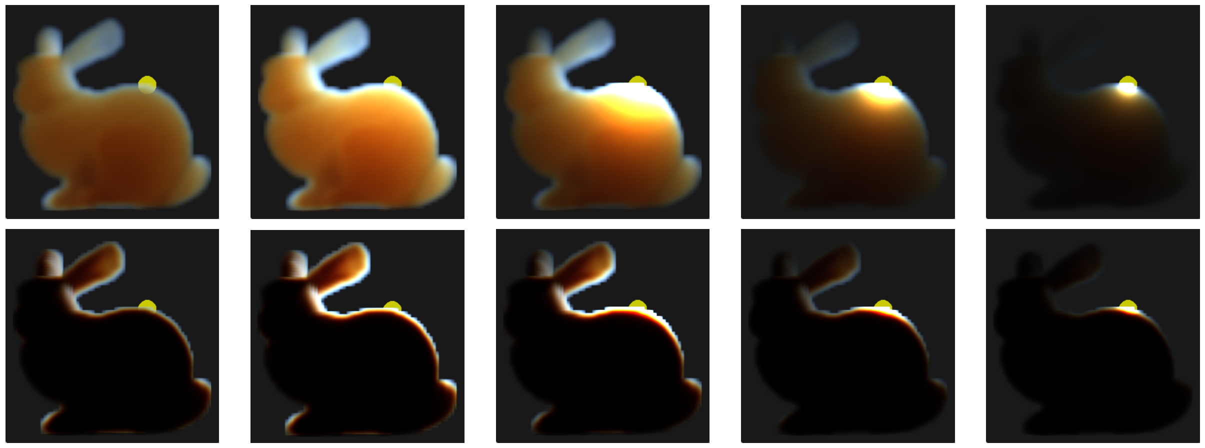


Figure 14: Stanford bunny using a the Henyey-Greenstein phase function 9 with g ranging from isotropic to strong forward scattering (from left to right $g = 0.0, 0.5, 0.9, 0.99, 0.999$) and dnesity of 1.0 (top) and 10.0 (bottom).

The phase function will also greatly influence final look of participating media. For instance, it is critical to achieve the characteristic look of clouds in order to get their characteristic sliver lining visual effect. And this is the same for many types of smoke for instance. This phenomenon is visible in Figure 14 especially for dense material blocking the light from traversing it (bottom row). You can also notice that for strong forward scattering, $g > 0$, the medium will look brighter when looking towards the light source. Indeed, strong forward scattering media will scatter more and more light toward it traveling direction only, leaving other areas darker. This generates halos around light sources, generating the glow everyone knows, for instance when looking at street lights in foggy days. Then the thicker the medium gets, the less light will be able to travel through it. However, the strong forward scattered light will still be get through for small optical depth near the edge of the volume, resulting in the silver lining visual effects.

2.5 Related chapters

The participating media material and light interaction described in this section is the foundation of all the techniques and result presented in this document. The concept presented here will be discussed the following Section:

- Rayleigh and Mie scattering and the atmosphere medium: Section 3.2
- Cloud phase function: Section 5.7
- Integration improvement: Section 5.6
- Multi-scattering approximation: Section 5.8

3 Sky and Atmosphere

This section describes how sky and atmosphere scattering can be simulated and rendered. There are already plenty of very detailed resources on that area. Thus this section will be short and we will reference outside articles and open source code as much as possible.

The research and development presented in this section have been conducted as a joint effort together with:

- Ghost: Gustav Bodare and Edvard Sandberg (2015 Frostbite/DICE/Chalmers Master students [BS])
- Ghost (Need for Speed$^{\text{TM}}$)
- DICE (Mirror's Edge Catalyst$^{\text{TM}}$)
- Bioware (Mass Effect Andromeda$^{\text{TM}}$)

3.1 Previous work

Rendering a world inherently requires the rendering of a planet sky and atmospheric effects. On Earth, what we call the blue sky is the result of the sun light scattering in the atmosphere participating media. The atmosphere is also a key visual cue: its color is linked to the current time of day, *i.e.* sun direction, and its foggy visual appearance helps with the perception of distance and relative size. As such, it is important to be able to accurately render these components required by an increasing amount of games needing dynamic time of day and large open world to explore, drive or even fly over. The first physically based atmosphere rendering model from Nishita *et al.* [Nis+93] was dedicated to the rendering of the earth from space. Since then, many sky rendering methods have been proposed to render atmosphere and skies fro, the ground up to space. They can be split in two categories:

- Analytic models [PSS99][HW12][CIE95]
- Iterative models [Ril+04][Wen07][ONe07][BN08][Ele09][HW12][Yus13a]

Analytic models build a set of parameters used to evaluated the sky look. For instance [PSS99] relies on turbidity, a measure of the fraction of scattering due to haze as opposed to molecules (Mie/Rayleigh scattering), luminance at zenith and of course view and sun directions. These models however are limited to ground view or atmosphere parameters can't be changed freely to simulate extra terrestrial planets, or reach specific art driven visuals.

The spectral rendering of the sky can be used for an improved accuracy in many models but we cannot really afford that for real time games on today's platforms. In this case we will restrict ourselves to the usual 3 wavelength of the visible light range: red ($\lambda = 680nm$), green ($\lambda = 550nm$) and blue ($\lambda = 440nm$).

From now on, we are going to focus on iterative atmosphere models relying on LUTs for atmosphere simulation and rendering. If you want more details about how these sky simulation models can be implemented, we strongly recommend you to read [BN08], [Ele09] and [Yus13a] in details. Their implementations are nicely described, detailed and **open source code** is also provided at the following addresses [Bru17] and [Yus13b] respectively.

Iterative models mainly rely on ray marching in order to integrate scattered light. This result is usually stored in look-up tables (LUT) in order to avoid the expensive cost of ray marching. Those textures can then simply efficiently, leveraging hardware filtering features of graphic cards. For instance, Bruneton et al. [BN08] are generating the following LUTs:

- 2D Transmittance LUT: only indexed on view height and azimuth angles thanks to earth spherical symmetry.
- 4D scattering LUT: since this depends on height, view and light direction, it is indexed based on a custom remapping of these values to also avoid certain visual artifact at the horizon.

The 4D scattering LUT can then be used to iterate on multiple order of scattering and thus pre-integrating a LUT already containing multi-scattering [BN08]. That is a very important property of these approach since multi-scattering is especially important when the sun is at the horizon in order to not get too saturated or dark.

Elek *et al.* [Ele09] proposed to reduce the 4D scattering LUT dimensionality by ignoring the change of scattering as a fucntion of the horizontal/azimuthal angle betwen the view direction and the sun direction. This simplification basically remove the earth shadow from the atmosphere multi-scattering solution. This result in a simpler 3D LUT that is faster to evaluate in real-time on GPU. Additionaly, Yusov [Yus13a] proposed an improved parametrization helping getting more details at the horizon and reducing some visual artifacts that can appear at the horizon.

3.2 Sky and atmosphere participating media definition

To be able to render sky and atmosphere, we need to take into account several components. We must first consider the atmosphere as a constant height slab around the earth with a exponential distribution of air molecules in it.

Light interacting with air particles that are much smaller than the light's wavelength results in the highly wavelength dependent **Rayleigh** scattering. Considering the earth atmosphere, blue light are scattered more and that is why the sky appears blue during the day. However when the sun is at the horizon, light will have to travel a longer distance in the atmosphere and most blue light will be scattered away. Blue light will not travel as far as the green and red light in the atmosphere. That is why sunset and sunrise appear reddish.

Another important component of the atmosphere is the large particles concentrated near the ground. The concentration of these particles depends a lot on weather conditions, or pollution for instance. These particles cause wavelength independent **Mie** scattering. So the phase function describing how light will scatter is usually not uniform but biased toward the direction of the light travel direction, *i.e.* forward scattering. This phenomenon will cause the bright halo we usually see around the sun.

3.3 Atmosphere composition

In this sub-section we describe what coefficients and distribution should be used. After discussing with [BN08], we have learned that these coefficients do not represent scattering coefficient gathered from all wavelengths and integrated with respect to the RGB visible spectrum according to the Human perception. Instead the scattering coefficients for R, G and B were only taken for the corresponding wavelengths 680, 550 and 440 nanometres.

We follow the usual description of the atmosphere from [Ril+04] and [BN08]. On top of which we also add Ozone contribution which is important for the look of that sky a sunset and sunrise. Table 2 summarizes all the coefficients and their distribution in the atmosphere.

We have chosen to use the same Rayleigh scattering coefficient as [Ril+04] and [BN08], even though the evaluation of Equation 11 gave us different numbers $(5.47e^{-6}, 1.28e^{-5}, 3.12)^{-5}$ for air refractive iindex $n = 1.0003$, a number of molecule per meter cube $N = 2.545 \times 10^{25}$ and a standard air depolarisation factor $p_n = 0.035$ [PSS99]. The Mie coefficient is really up to the atmosphere status: clarity, pollution, dust, sand storm, *etc.*

Type	**Scattering** (m^{-1})	**Extinction** (m^{-1})	**Distribution**
Rayleigh (molecule)	$\sigma_s^{Ray} = (5.8e^{-6}, 1.35e^{-5}, 3.31e^{-5})$	σ_s^{Ray}	$e^{\frac{-h}{8.0km}}$
Mie (dust)	$\sigma_s^{Mie} >= 2e^{-6}$	$1.11\sigma_s^{Mie}$	$e^{\frac{-h}{1.2km}}$
Ozone	0	$\sigma_a^{O_3}$	$e^{\frac{-h}{8.0km}}$

Table 2: Default earth atmosphere properties.

$$\sigma_s^{Ray} = \frac{8\pi^2(n^2-1)^2}{3N\lambda^2} \times \frac{6+3p_n}{6-7p_n} \tag{11}$$

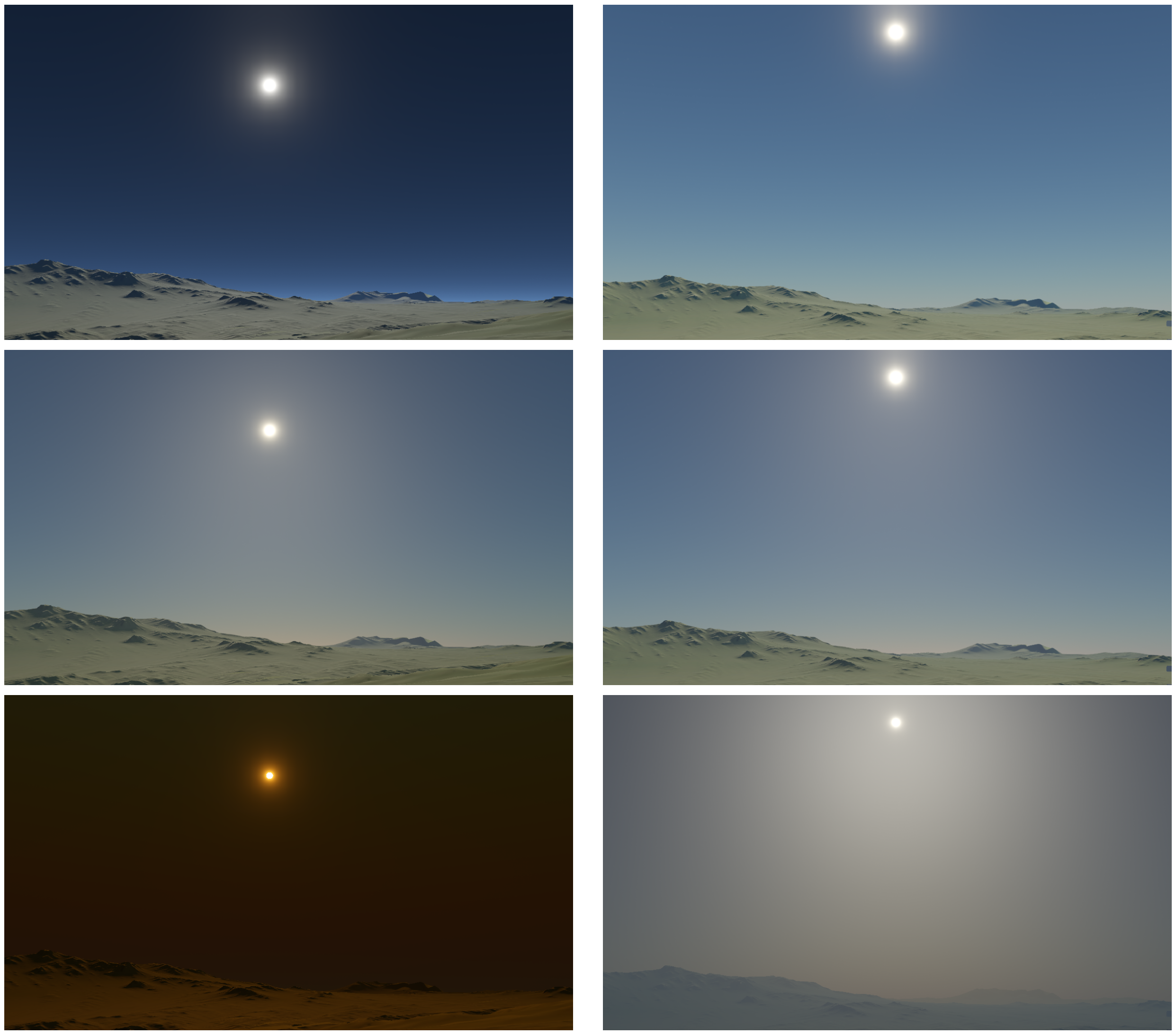


Figure 15: Left: light, earth-like and heavy Rayleigh scattering. Right: default earth-like sky with no, default and heavy Mie scattering.

The result of using such coefficients is visible in Figure 15. You can notice that increasing the Rayleigh scattering will increase the blueness of the sky until light extinction become more important due to out-scattering, as shown in participating media example Section 2.4. Increasing Mie scattering simply makes the atmosphere look more dusty as if there would heavy pollution or a sand storm.

3.4 Ozone absorption

As reported by Adams [CK74], taking into account ozone particle absorption is *Essential [...] to reproduce the blue of the zenith sky.* Kutz in his master thesis blog present visual improvement resulting from taking into accoutn ozone [Kut13]. Unfortunately, the absorption coefficients are no shared: we present here how we recovered them. We recover $\sigma_a^{O_3}$ using Equations 12 and 13. We first recover the air molecule per unit volume ($molecule/m^3$) using Equation 12 where $airConcentration = 41.58 mol/m^3$ [Wikk] is the air density at sea level and $N_A = 6.022140857 \times 10^{23}$ is the Avogadro constant [Wiki]. Then using Equation 13 the absorption coefficient is evaluated from ozone cross section and air density. The ozone cross section is taken from measured data from [Ser13]: the value is an average of each R, G and B wavelength range for all emasured temperatures. According to overall ozone percentage in the atmosphere air [Kut13], the final recovered values for ozone absorption are $\sigma_a^{O_3} = (3.426, 8.298, 0.356) \times 0.06 \times 10^{-5}$.
Ozone should be concentrated 32km up in the sky. But this was giving us unexpected results (probably due to using RGB instead of a more complete spectrum). Instead, we have chosen ozone to follow the same atmosphere distribution as the Rayleigh scattering particle distribution and thus the absorption coefficients can simply be added to the Rayleigh extinction coefficient used for the scattering simulation.

$$N_{air} = airConcentration \times N_A \tag{12}$$

$$\begin{aligned}
crossSection^{O_3R} &= 1.36820899679147 \times 10^{-21} && (cm^2/molecule)\\
crossSection^{O_3G} &= 3.31405330400124 \times 10^{-21} &&\\
crossSection^{O_3B} &= 1.42214627365509 \times 10^{-21} &&\\
crossSection^{O_3R} &= crossSection^{O_3R}/10000.0 && (m^2/molecule)\\
crossSection^{O_3G} &= crossSection^{O_3G}/10000.0 &&\\
crossSection^{O_3B} &= crossSection^{O_3B}/10000.0 &&\\
\sigma_a^{O_3R} &= N_{air} * crossSection^{O_3R} && (1/m)\\
\sigma_a^{O_3G} &= N_{air} * crossSection^{O_3G} &&\\
\sigma_a^{O_3B} &= N_{air} * crossSection^{O_3B} &&
\end{aligned} \tag{13}$$

The result of using ozone is visible in Figure 16. Without ozone, the sky can appear too yellow overall. Taking into account ozone in the extinction coefficient (see Section 2.1) can bring back a more consistent blue sky color at sunset and sunrise.

As a *side note*, we have tried to use the wavelength dependent ozone absorption coefficients from [PSS99] and the spectrum to XYZ functions from [WSS13]. We have tried different transform and sRGB gamut space clipping without being able to recover a consistent absorption coefficients with respect to simulated distances. This approached proved to be unstable and we would be interested in any feedback and why/what we might have done wrong. The attempt to recover these coefficients is available publicly in a ShaderToy [Hila]. Figure 17 show the ShaderToy presenting multiple graphics: the wavelength to RGB weights, the ozone absorption curve in grey per wavelength range, as well as recovered RGB colour after distance based absorption per wavelength band and transformation back into RGB space. You can notice that negative absorption coefficients are sometimes recovered and that is completely invalid. That is why help and/or suggestions are welcome.

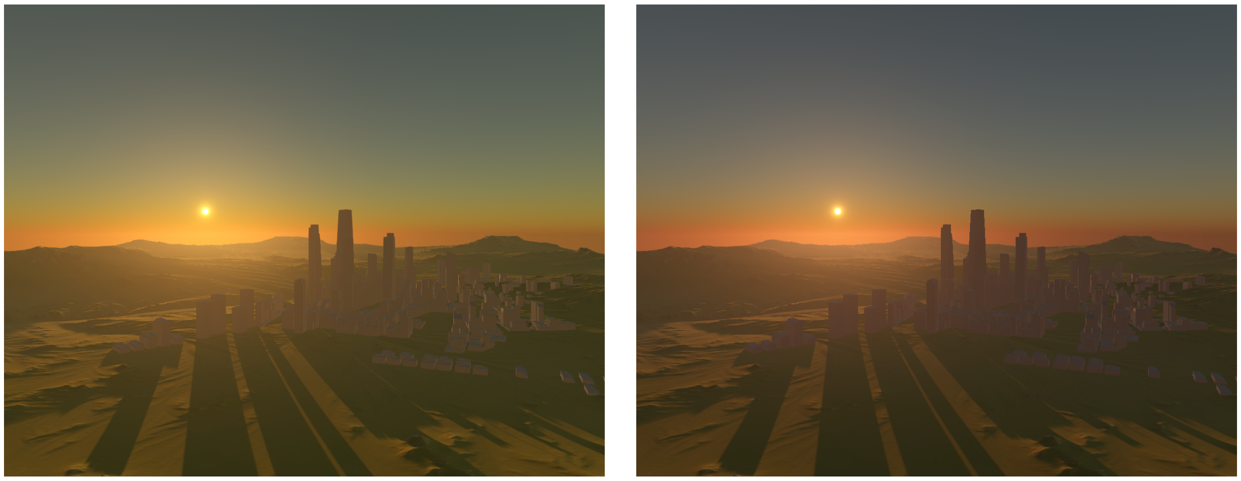


Figure 16: Left: default sky. Right: default sky with added ozone absorption.

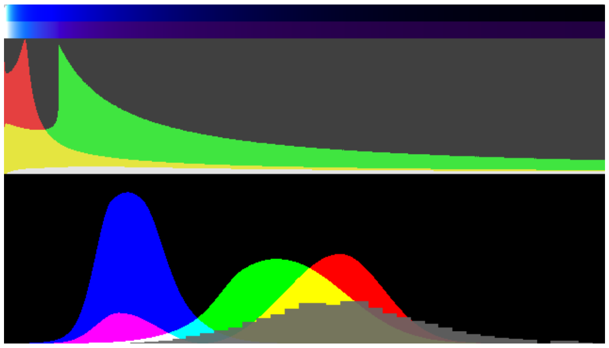


Figure 17: The spectrum to RGB ozone absorption ShaderToy [Hila].

3.5 Our approach

The physically based sky system available in Frostbite borrows from many research results: Bruneton [BN08], Elek [Ele09] and Yusiv [Yus13a]. Here is a list of choice:

- We use a 3D look up table as in [Ele09] instead of the 4D original one [BN08]. This only means that we ignore the view/sun azimuth angle: for instance we cannot represent the shadow of the earth in the scattering look-up table. We found that this is a reasonable assumption for most use cases we encountered so far. A comprehensive list of all assumptions is available in Section 4.1.1 of [Ele09].

- As described by Bruneton *et al.* [BN08] the scattering LUT can have accuracy issue at the horizon, resulting is visual artefacts for elevation angle of 0 when the view is near the ground. We rely on the parameterisation improvement proposed by Yusov [Yus13c]. We give simple non-optimized reference hlsl code for the parametrization we use in appendix A.

- Evaluating the scattered luminance using LUTs per pixel multiple times when rendering the aerial perspective on opaque surface could be expensive depending on your budget. To reduce the cost, we evaluate each frame the scattered luminance for current view in a low resolution 3D texture fitted on the camera frustum (default resolution: 32x32 with 16 depth slices). This makes the fog evaluation cheaper and has the advantage of being easy to evaluated and apply on all transparent meshes to ensure consistency. On Frostbite, we sample this volume texture on transparent per vertex. Aerial perspective rendering is visible in Figure 18.

- We give artists a way to also add height fog to their scenes [Wen07]. For the final image to looks consistent under time of day, we color the height fog according to a luminance taken from the LUT at the horizon. This is valid thanks to the use of a 3D scattering LUT instead of the 4D one mentioned above (it owuld have required to sample the value accros the full horiwon in this case). This lumminance is taken into account together with the phase functions evaluated per pixel when when applying the heigh fog on the scene [Hil15]. This results in seamless transition from height fog near the camera, at the horizon and transition to the sky as in Figure 18.

- Once the LUTs have been computed, the sun can be moved freely. But changing some atmosphere parameters such as *height*, *scattering* or *extinction* coefficients will trigger a LUT update that is too costly. We counter that issue by temporally amortize the look-up table update cost over multiple frames (see Section 3.5.1.)

3.5.1 Performance

In order to result in coherent and unified lighting and shadowing, the physically-based sky must be rendered in multiple views, *e.g.* main, planar reflection, environment map. In this section we give our latest performance results on XBox One.

Pass	**Performance**
720p Main view	0.42 ms
AP volume 32x32x16 (Section 3.5)	0.05 ms
LUT update on one frame	3.50 ms
LUT update 19 frames	0.22 ms per frame

Table 3: Physically based sky rendering performance on XBox One.

The performance given in Figure 3 are given for a full screen sky. When atmosphere properties are changed, we must update the transmittance and scattering look-up tables. This could take as much as 3.5ms, due to the fact that multi-scattering is also integrated. To avoid this cost, we are distributing the evaluation of the LUT on multiple frames while lerping between the last two valid results [BS]. This is a key point which allowed us to make the technique affordable for 60 frame per second titles. It does add a little bit of latency but that has been deemed acceptable for all of our use cases..

3.5.2 Results

Results from the dynamic sky are visible in this entire document, this section and also the volumetric cloud Section 5. The first games to ship with that technology were Need for Speed 2016 [Gho15] and Mirror's Edge catalyst [DIC16] [Chr16].

Since the sky simulation takes scattering and extinction coefficients as input, it is possible to render any extra-terrestrial planets. For instance, Mars is called the red planet because it appears as a red star due to its rusty ground. It also appears that Mars atmosphere would scattered more the red and green component of the light spectrum. This is why the mars atmosphere at day time appears yellow or orange [NAS16]. A direct consequence of that fact is that sunsets on Mars are not red but blue as shown by NASA on their website [NAS05] and as visible in Figure 19. We have not been able to find Mars atmosphere scattering properties but Figure 20 shows a series image from a time-lapse render of a sunset on Mars rendered with Frostbite and using an eye-balled set of atmosphere properties.

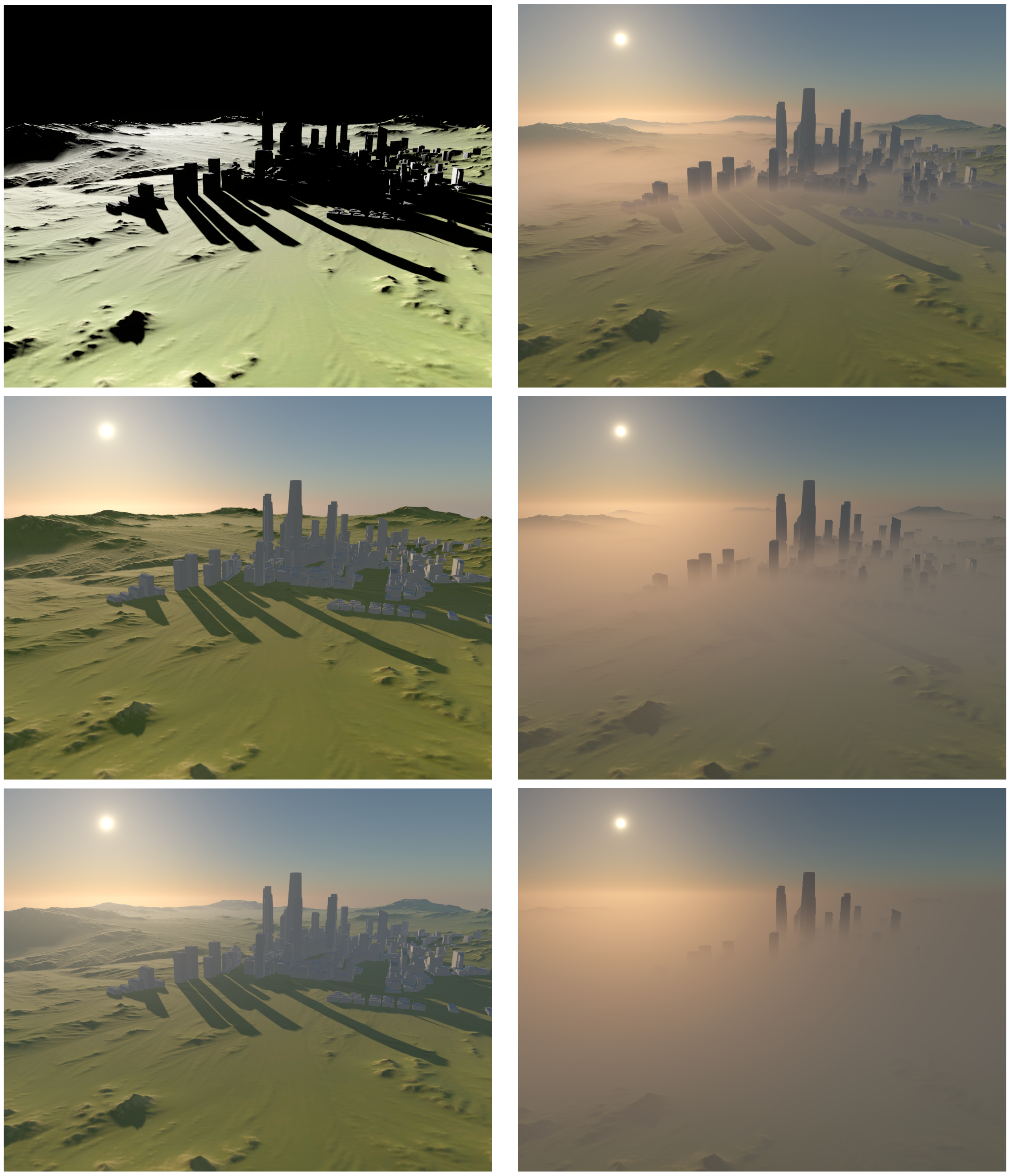


Figure 18: Sky, atmosphere scattering and height fog. From top to bottom; left: only sun light, added sky rendering, added aerial perspective; Right: progressively adding a thicker height fog.

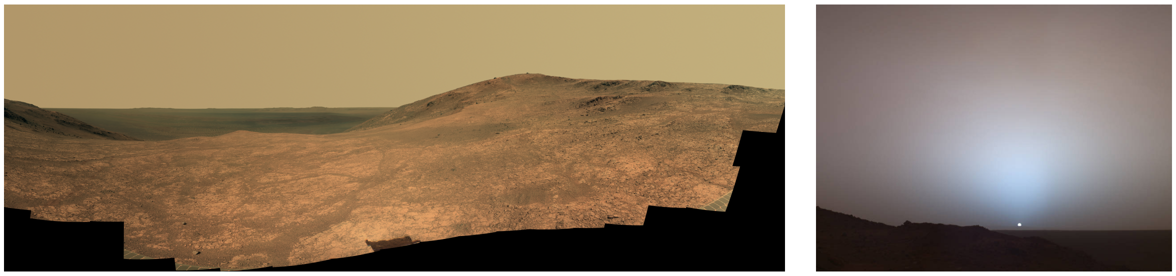


Figure 19: Left: Mars view during day light [NAS16]. Right: Mars view during sunset [NAS05].

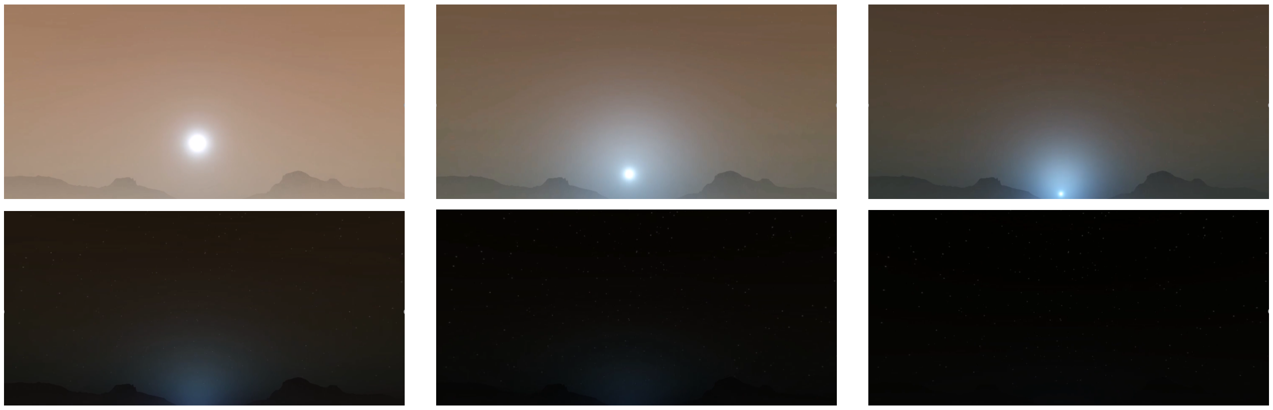


Figure 20: Time-lapse of a Mars atmosphere sunset simulation in Frostbite.

4 Sun, Moon and Stars

Rendering skies involves the rendering of many other far away elements:

- Sun
- Moon
- Stars
- Celestials

When rendering these elements involves paying attention to many small details. One also needs to know their properties such as luminance or angular diameters [Wika] to faithfully represent them.

4.1 Sun

The sun is the stars the earth is orbiting around. From the see level, Its angular diameter is between 32.7 to 31.6 minutes of arc depending on time of year [NAS16], i.e. according to its orbit position. It corresponds to an angular diameter of 0.527 deg to 0.545 deg.

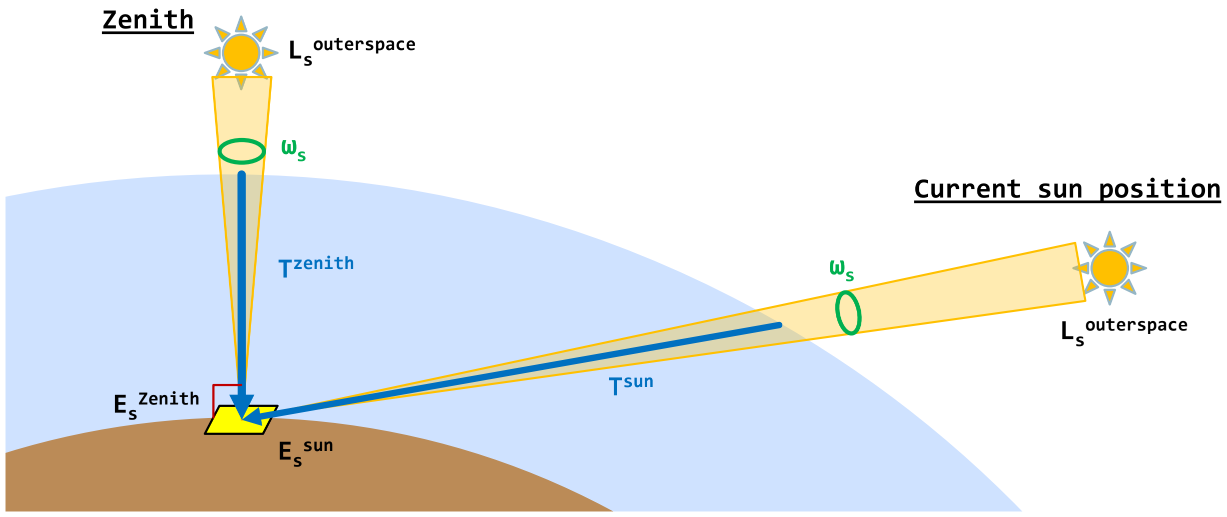


Figure 21: Sketch presenting the different elements and quantities discussed in this Section: earth, atmosphere, transmittance, sun luminance and illuminance at zenith and current sun position.

4.1.1 Sun illuminance

The sun illuminance E_s at ground level (Figure 21) is reported as being a value between 100000 to 120000 Lux [Wikf].

In Frostbite, artists author the sun contribution by giving its illuminance at zenith E_s^{zenith}. This is more convenient for them as it becomes easier and more intuitive to compare results again real world values (see [LR14] Section 4.6). It is also given for the sun at zenith after the atmosphere transmittance has been applied to it. This is then easier for artists to abstract away earth transmittance resulting form non trivial distribution of particles in the atmosphere and scattering/absorption coefficients.

4.1.2 Sun luminance

In Frostbite, the light buffer stores luminance L, not illuminance E. As such, we need to convert the sun illuminance E_s^{zenith} given by artist to its luminance L_s that will be applied on the sun disk. In order to achieve this, the following process is applied:

1. Considering the sun as a perfect disk, evaluate its solid angle ω_s (assumed constant on earth)
2. Evaluate sun luminance L_s^{zenith} at ground level according to ω_s and E_s^{zenith}
3. Considering the earth transmittance and sun at zenith, evaluate the sun outer space luminance $L_s^{outerspace}$
4. Render the sun using $L_s^{outerspace}$ and apply atmosphere transmittance to it

For a cone with aperture θ radians, the solid angle can be evaluated using Equation 14. It is thus possible to recover sun the solid angle ω_s (between $0.0000664sr$ and $0.0000711sr$ for physical angular diameters reported above).

$$2\pi(1 - \cos(0.5 \times \theta)) \tag{14}$$

In the case of the sun, illuminance at ground level is given by artist for a sun at zenith and independently of its subtended solid angle. If we consider the sun having a relatively small solid angle and of a relatively uniform luminance, we can approximate its illuminance E_s^{zenith} as the integral over its solid angle using equation 15. Then we can simply recover the sun luminance L_s^{zenith} using equation 16

$$\begin{aligned} E_s^{zenith} &= \int_{\omega_s} L_s^{zenith} \\ L_s^{zenith} &\approx \omega_s L_s^{zenith} \end{aligned} \tag{15}$$

$$L_s^{zenith} \approx \frac{E_s^{zenith}}{\omega_s} \tag{16}$$

For a given earth/atmosphere setup, it is possible to easily compute transmittance at zenith $Tr_{atmosphere}^{zenith}$ by integrating extinction from the ground along the up vector until the considered atmosphere upper boundary. Outer space luminance can now be computed as $L_s^{outerspace} = L_s^{zenith}/Tr_{atmosphere}^{zenith}$. the reasonable assumption we are making here is that atmosphere transmittance never reach 0 for each wavelength components.

Having the sun outer space luminance, we can thus render the sun sprite as a perfect disk, matching its angular diameter, and add its luminance $L_s^{outerspace}$ contribution to the light buffer. However, if only this is done, the sun will simply look like a very bright disk because the atmosphere transmittance $Tr_{atmosphere}^{sun}$ is ignored. This can be resolved by using the atmosphere transmittance lookup table described in Section 3. Once sampled per pixel, we simply evaluate the sun correct final luminance as $L_s = Tr_{atmosphere}^{sun} \times L_s^{outerspace}$. This final correct and matching result is visible in Figure 22.

With this process done, the sun will have a correct appearance matching its zenith angle as well as the atmosphere properties. This is especially important when the sun moves to simulate time of day. It is also important on low exposure scene such as dusk or dawn setup for the sun to not bloom out the picture when visible. Also, since the sun luminance is recovered from its illuminance and solid angle, making the sun larger will automatically dim its luminance. This process automatically makes sure that the overall visual scene lighting remains consistent with the sun appearance.

Example of values for the sun on earth:

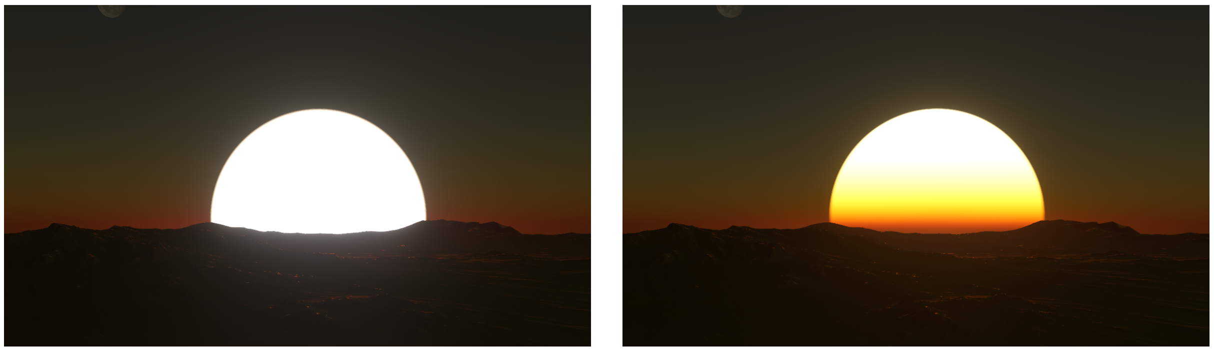


Figure 22: Screenshots showing the sun disk at horizon (dusk setup) rendered without (left) and with (right) atmosphere transmittance applied per pixel.

- Illuminance on ground $E_s^{zenith} = 120000$ Lux
- Angular diameter of 0.545 deg corresponds to a solid angle of $0.0000711sr$
- Luminance $L_s^{zenith} = 1.69 \times 10^9 cd.m^{-2}$
- Outer space luminance $L_s^{outerspace} = L_s^{zenith} / Tr_{atmosphere}^{zenith}$

Using Frostbite default physically based sky simulation from Section 3, transmittance can get a value between $0.925, 0.861, 0.755$ at zenith and $0.0499, 0.004, 4.10e^{-5}$ when the sun is at the horizon. Figure 23 present the different properties of the transmittance according to the sun elevation.

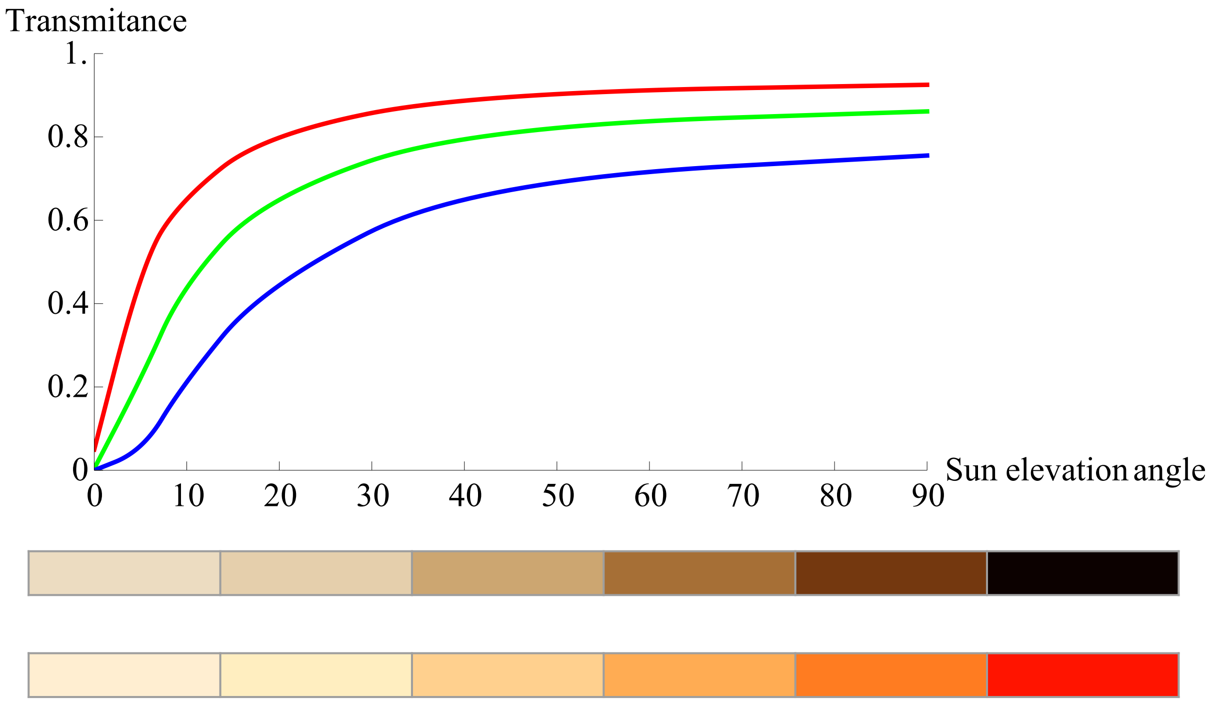


Figure 23: Top: transmittance curve for each RGB channel of perceptible light wavelength. Middle: transmittance as a color for elevation of 90, 45, 20, 10, 5 0 degrees. Bottom: scaled transmittance according to highest wavelenght channel to visualize transmittance tint, *i.e.* hue.

4.1.3 Limb darkening

The sun appears as a disk in the sky. But because the sun is a sphere, the disk will not have a uniform luminance [Wikh]. This is due to the fact that, for a given point of view, more light will be visible when viewing the surface along it normal (center of the disk) than tangent to it (edges of the disk). Indeed, in tangent areas, light has to travel more trough the sun gas and thus will also get more absorbed. This phenomenon results in a sun disk being visually more intense at its center than at its edge. Astrophysics researchers have measured the luminance variation of emitted light on the sun disk and proposed some models [Nec96].

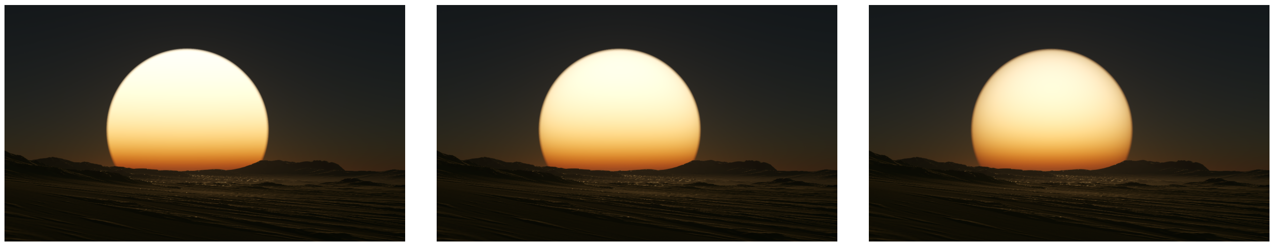


Figure 24: Screenshots showing the sun disk at horizon (dusk setup) rendered (a) without limb darkening, (b) with limb darkening matching earth solar system sun [HM98] and (c) an even strong limb darkening effect obtained by changing the parameters.

The implementation of [Nec96] and [HM98] models is given in this document. The HLSL source implementation is available in Appendix B. The gradient resulting from the model proposed in [Nec96] is visible in Figure 25. It is possible to initialise the model to the earth solar system sun and still give artists a way to author limb darkening. In a real-time context, you may even want to simplify and optimise these models to a simple gradient texture lookup.

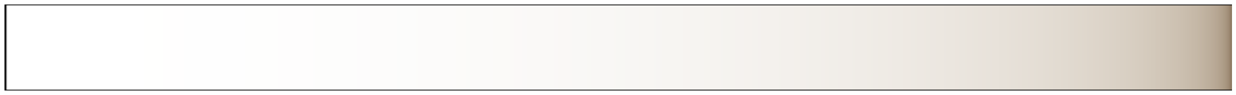


Figure 25: The sun limb darkening gradient resulting from [Nec96] with the sun disk center on the left and its outer edge on the right.

4.2 Moon

The moon is a satellite orbiting the earth at a mean distance of $384000km$. Its angular diameter is between 29.3 to 34.1 minutes of arc depending on time of year [Wikj]. This corresponds to an angular diameter between 0.488 deg to 0.568 deg.

- Illuminance on ground $E_{moon} = 0.26$ Lux [Wikf]
- Angular diameter of 0.568 deg corresponds to a solid angle of $0.0000711sr$
- Luminance $L_{moon} = 3658cd.m^{-2}$

Since E_{moon} is illuminance at ground level (after atmosphere transmittance), L_{moon} could be used to render the moon luminance after multiplication by a texture presenting its albedo.

One tricky effect to render is to make the moon and other orbiting objects have a lit and shadowed side with respect to single and/or multiple sun(s). This case is not handled today in Frostbite: artists are responsible for the setup. On top of that, as a fun fact, one must keep in mind the moon terminator illusion [VSc].

4.3 Stars

Stars are light emitter bodies scattered in the universe. The most well known of them being the sun. We have not found any mean solid angle, illuminance or luminance data for stars. We have only been able to gather the following data:

- Typical value for stars contribution to earth lighting have been reported in [Jen+01a] ($E_{starts} = 3e^{-2}W/m^2$)
- The angular diameter of constelations, solar systems, a few stars and other object in space are reported in [Wika]

A way to render stars is also presented in one of Neyret's shader toy [Ney]. It is rendering coloured stars according to their temperature. The colour is recovered using Plank's law describing spectral density of electromagnetic radiation for each temperature [Wikl][Wikb].

4.4 Results

Figure 26 presents the result when rendering sun, moon and stars sprites at dawn and night time in Frostbite. Environment map and local reflection volumes cube-map can capture the moon, stars and other space and celestial elements. This result in the moon and start being visible in reflections, increasing the realism and overall coherency of a scene. When these cube maps are convolved, these elements will be part of the convolution, also resulting in more consistent lighting.

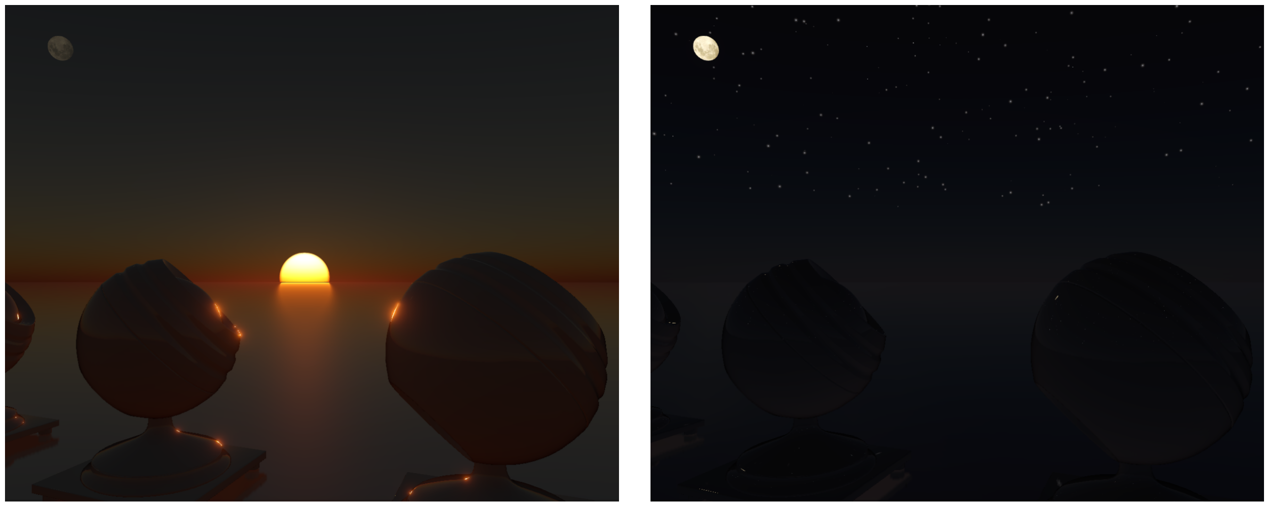


Figure 26: Screenshots showing the rendering of the sun disk (enlarged), moon and stars at (left) dawn time, (right) night time also their presence in reflections.

5 Clouds

The research and development presented in this section have been conducted as a joint effort with:

- Rurik Högfeldt (2015 Frostbite Master student [Hög])
- Bioware: Marc-Andre Loyer (Programmer), Soren Hesse (Tech Environment Art) and Don Arceta (Lead Environment Art)

5.1 Background and Previous work

Clouds are a very complex and a very expressive visual feature of skies. Art wise, one can make them look menacing, representing incoming storm, epic or discreet, thin or massive, *etc.* Cloud usually move slowly but needs to be dynamic for large open world game with dynamic weather changes. Different techniques can be used to achieve these looks depending on the level of complexity of a game setup and budget.

When considering the classic approach of rendering a sky and clouds using a single panoramic texture, Guerette proposed to use a well known visual flow technique in order, to give an illusion of motion in the sky [Gue14]. The cloud would them appear to move in a direction set for instance the global wind direction. This is a very effective method that however do not involved any variation of cloud shape, weather nor lighting.

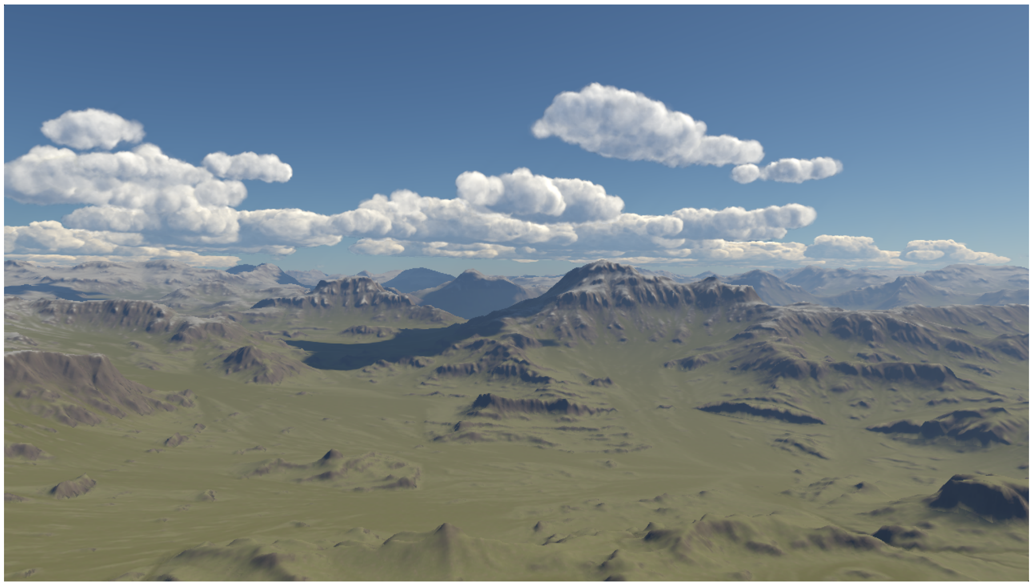


Figure 27: Results obtained by Yusov *et al.* [Yus14].

In the case of flight simulator, Harris posoposed to render clouds as volumes of particles [Har02]. The method was made very efficient by not rendering all particles all the time but impostors representing groups of particles when far away. This gives the possibility to update impostors at a lower rate according to the camera distance and relative displacement. Another particle based cloud rendering method is the one presented by Yusov [Yus14]. Strato-cumulus like clouds can be rendered by taking into account the dynamic lighting of the sun and sky using per particle pre-integrated lighting. The simplistic particle-like look was avoided by using depth aware blending made possible using a new hardware feature called *Rasterizer Ordered Views*, see Figure 27. These two particles based approaches can be very efficient at rendering clouds but are mostly limited to cumulus-like shapes.

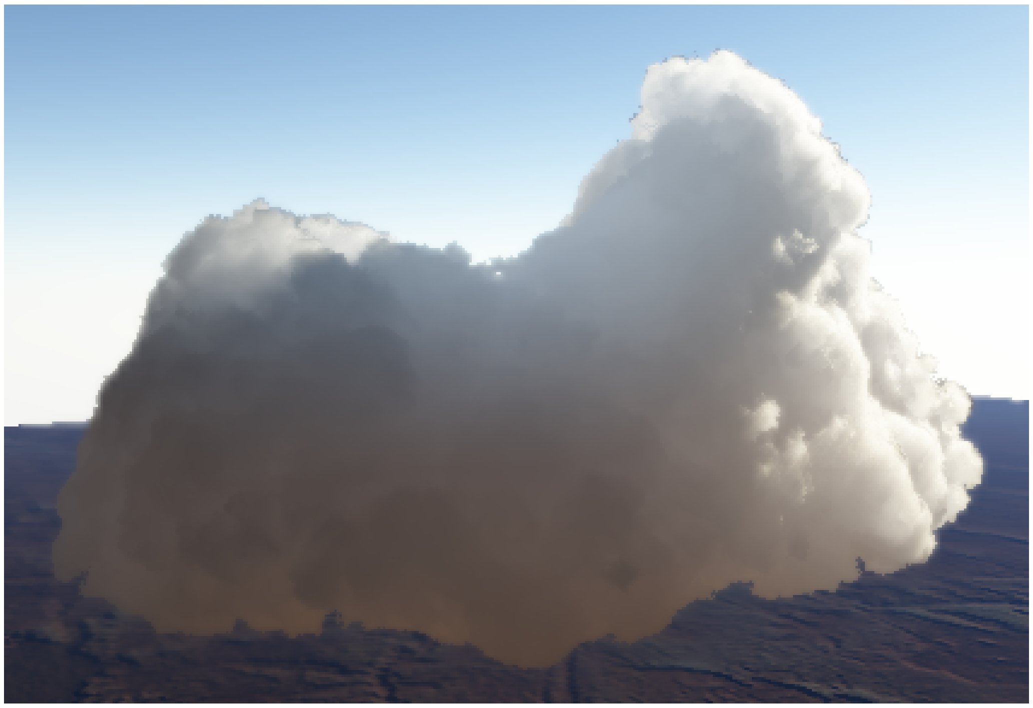


Figure 28: Results obtained by Bouthors *et al.* [Bou+08].

A few volumetric based cloud rendering techniques have also been researched [Ril+04][Bou+08][Sch15]. For instance, Bouthors render clouds with a mix of meshes and ray mached *hyper* textures [Bou+08]. The final scattering light is gather using disk-like shaped light collectors positioned at the surfaces of the cloud shape. The light transfert is integrated while ray-marching in real-time and accelerated using off-line pre-computed transfer tables. The final result is of very high visual quality as visible in Figure 28 but it also has a non-negligible GPU cost. Furthermore, the combined mesh and hyper texture data are not straightforward for artists to comprehend, create and edit.

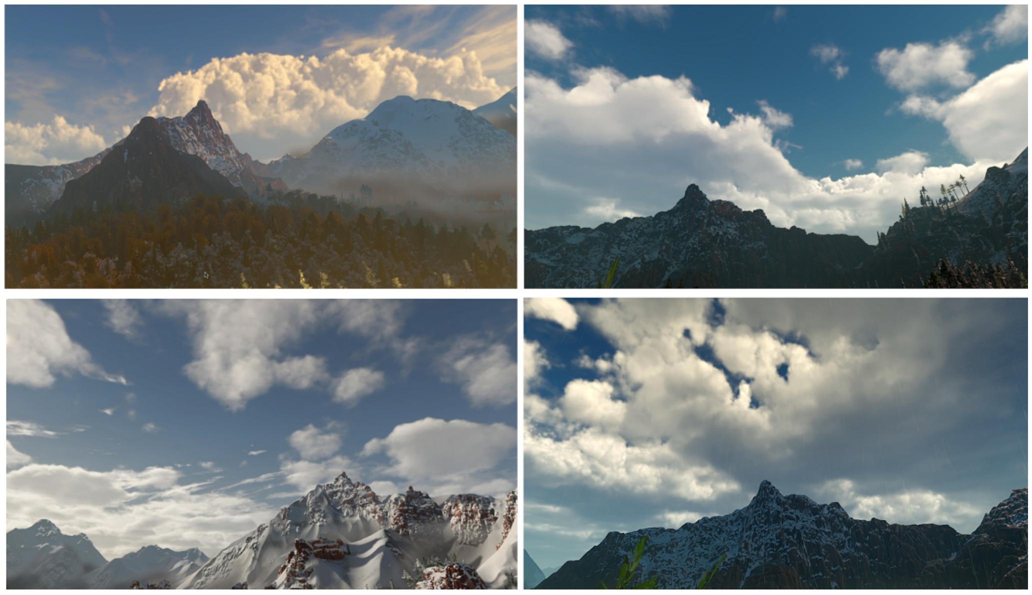


Figure 29: Results obtained by Schneider *et al.* [Sch15].

In the context of real-time game, Reset is the first game that has demonstrated advanced cloud rendering together with atmosphere interaction [Ltd]. However, not a lot have been disclosed publicly about the algorithm detail. Schneider presented a visually similar ray-marched approach [Sch15], allowing to render dynamically lit volumetric clouds. With few parameters, the method allows the rendering of complex cloud shapes with many details as seen in Figure 29. The use of volumetric textures containing *Perlin-Worley* noise has been suggested as a very good fit to better represents the cauliflower-like shape of cumulus-like clouds. The resulting clouds are completely dynamic and can evolve according to time and weather constrains. This technique is very applicable to real-time games thanks to the use of temporal integration of the scattered light solution allowing to to temporally integrate the final scattering result.

For Frostbite we decided to follow the path of [Ltd] and [Sch15] since they have the following advantage we needed:

- Realistic cloud shapes
- Large scale clouds possible
- Dynamic, so weather change can be happen
- Dynamic volumetric lighting and shadowing support

We want our implementation to fit into the Frostbite physically based framework: have material information decoupled from lighting and be energy conserving. This to ensure coulds would fit within any lighting environment which is a must have when dealing with dynamic time of day and weather.

5.2 Cloud participating media material

Clouds are made of very thick participating media. Hess *et la.* [HKS98] measured water clouds and reported a single scattering albedo $\rho = 1$ and high extinction σ_t coefficient in the $[0.04, 0.06]$ range for stratus, $[0.05, 0.12]$ for cumulus (for the 550um wavelength corresponding to perceptible green). Given the fact that ρ is very close to 1, $\sigma_s = \sigma_t$ can be assumed.

Cloud single scattering is a very important part of their defining look together with their very specific phase function discussed in Section 5.7. With only single scattering, and due to their thickness, clouds would only look like dirty/smoky element with only scattered light at their surface. To avoid this, Another defining component of clouds look must be taken into account: the many scattering events taking place within them. Details on how to approximated this characteristic is given in Section 5.7.

5.3 Cloud authoring

The way artist can author clouds and their distribution in the world is very similar to the one in [Sch15] with some extra control that were needed for our games and uses cases. This volumetric approach used to generate cloud shapes is called procedural, it uses algorithms to generate content from a few parameters. Using algorithm to generate artistic data can be hard to control and also not always compatible with artists visions. that is why defining a set of meaningful input parameters for artists to achieve their vision is important. This section explains the controls exposed to artists authoring volumetric clouds in Frostbite.

Being a procedural approach, one can easily think of **tens to thousands** of ways to produce parameters and controls that will blend together using formulas to produce volumetric clouds. We present here our approach that matches our games and our artists desires. It is up to you to find what suits your need and you games best.

5.3.1 Cloud distribution and density

The clouds are assumed to remain within a single slab of constant height around the earth. They are made of a single participating media material with varying density only. Artists create a *weather texture* having a world space size and extent over the world. That texture is scaled and repeated all over the world if necessary. And example is visible in Figure 30 each channel represents:

- Red channel: 2d projected cloud density.
- Green channel: 2d projected cloud type index.

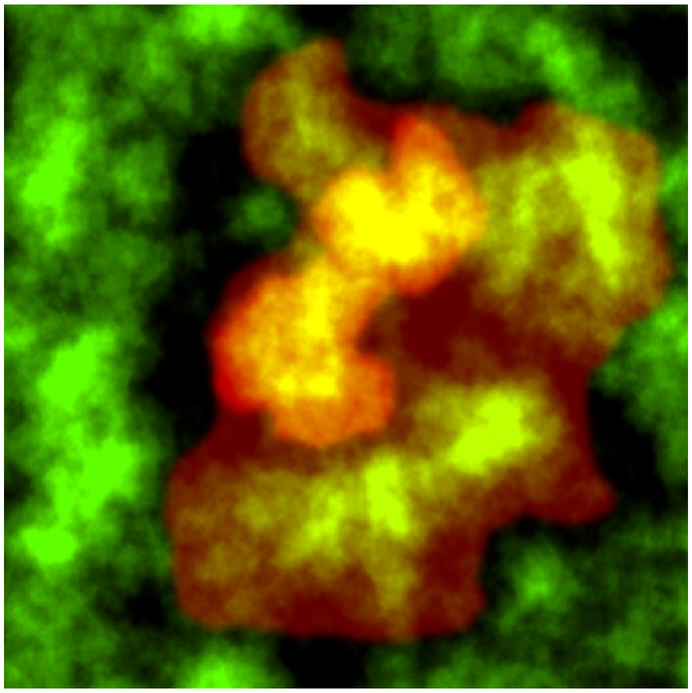


Figure 30: Example of cloud weather texture. Artists can paint the world space distribution of clouds as well as their type.

The cloud type is used to index another *cloud type* texture along the x axis in texture space. The Y texture space axis is the normalised height within the cloud layer. And example such as texture is visible on in Figure 31 each channel represents:

- Red channel: the density of the cloud within the layer height.
- Green channel: the erosion amount applied (small scale noise eroding large scale noise). This directly maps to the amount of turbulence of the cloud surface. 0 maps to smooth, 1 maps to fully eroded by the 3D erosion texture according to parameters similar to [Sch15].

The cloud type texture allows artist to specify cloud profile along the atmosphere height. Using such a texture also allows artists lots of freedom. They can for instance represent multiple layers of clouds or Anvil clouds (see Section 5.11). We simply ask artists to keep these textures as small as possible to help reduce texture cache miss as much as possible. Both textures can be statically assigned to a level and also dynamically updated if necessary.

While ray-marching the cloud layer, we evaluate the cloud density according to the weather texture, type texture but also according to two different volume noise textures in a similar fachion to [Sch15]. A low frequency $noise_L$ is first used to give a base shape to clouds and break down the repeatability

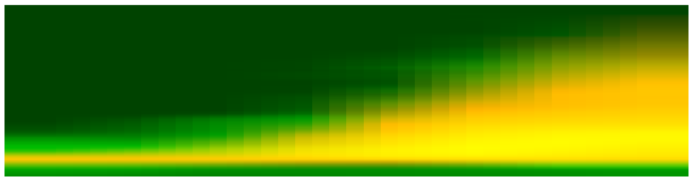


Figure 31: Example of cloud type texture. Artist can paint it in order to control the height based density as well as erosion.

of the weather texture, also called *base shape*. A high frequency one $noise_H$ is then use to erode that base shape and add details at edges. We present a way to generate these volume noise textures in the next Section 5.4.

5.4 Cloud noise

The cloud rendering algorithm described in [Sch16] propose to use a specific setup of tile-able volume noise textures but no source code or texture is given. We describe the texture generation in this section and link to an open source repository where source code can be accessed.

The $noise_L$ volume texture is generated as a combination of Perlin-Worley noise and multiple octaves of Perlin noise. The $noise_H$ texture is generated as multiple octaves of Worley noise. Worley noise is very interesting when it comes to cloud rendering since it helps representing the cauliflower like shape they can take at times.

This textures can be presented as 4-component RGBA textures that are combined using in shader math [Sch16]. In Frostbite we simply use a single component volume texture representing the final single channel noise. This made cloud a lot faster to render thanks to the reduction of required memory bandwidth still giving the same final visual result.

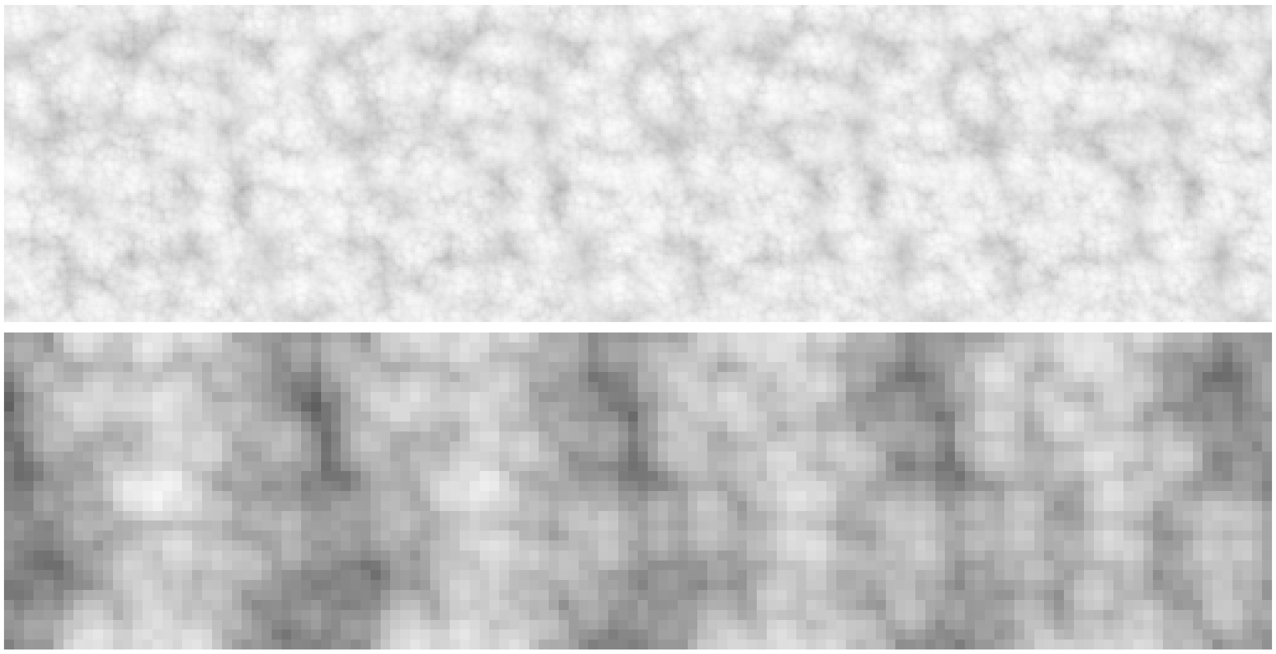


Figure 32: These image show 4 slices of tile-able volume noise: Top, *shape* noise containing cauliflower like Perlin-Worley noise shape, bottom: *erosion* noise made of multiple octaves of Perlin noise.

We give away a small program to generate such noise using open source libraries. Please refer to Appendix D for description and more details on how to access the code.

5.4.1 Cloud material

The cloud participating media material is described as a single participating media material. It consists of the following parameters:

- Absorption σ_a described in Section 2
- Scattering σ_s described in Section 2
- Dual lobe phase function described in Section 5.7

Only the density of the material is spatially varying. This density is built using the procedural approach described in the previous sub-section. In order for cloud to look convincing, the choice of noise is however crucial.

5.5 Cloud rendering

Using the material and equations described in Section 2 we integrate different lighting components presented in Figure 33.

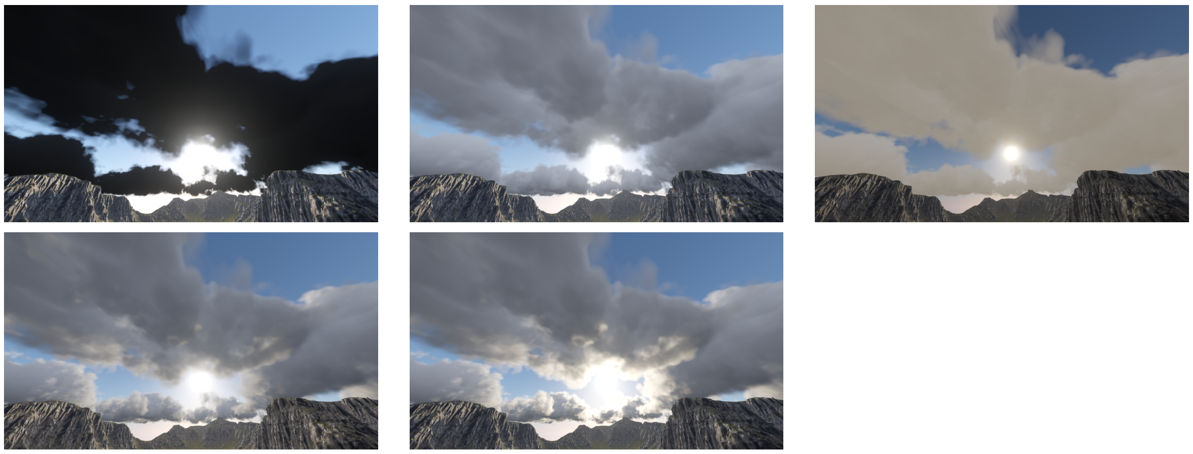


Figure 33: The different lighting components taken into account when ray marching the cloud layer (from top-left to bottom-right): (1) background transmittance, (2) ambient scattering, (3) not shadowed sun light scattering, cloud self shadowed sun light scattering without (4) and with (5) forward scattering phase function. 21 samples were used to generate these images.

5.5.1 Ambient lighting

Cloud ambient lighting is sampled using a global light probe represented as Spherical Harmonics [LR14]. For the sake of performance, ambient lighting occlusion is not taken into account when evaluating ambient lighting. This is would have not been practical given our current game budget and pre-integrating occlusion would also be tricky due to the very procedural nature of the volumetric cloud involving complex noise shapes and erosion processes. We only take into account the first non directional term of Frostbite global probe. Thus, luminance resulting for ambient lighting contribution can often be too bright. To counter this effect, we give artist a way to scale down the ambient component according to scale in $[0, 1]$. Taking into account the sky, i.e. atmosphere scattering, can result in slightly blue cloud if no multi-scattering solution is used. To resolve that issue, we also give artists a way to desaturate the luminance resulting from ambient lighting.

We also weight ambient lighting using a linear gradient in $[0, 1]$ from the bottom to the top of the cloud layer. This approach assumes that the sky is the only contribution to the ambient lighting and that it ignores bounce lighting from the earth. This can be approximated with two different approaches:

- Bias the gradient range to $[a, 1]$ in order to take into account that $a\%$ of the ambient lighting is due to bounce of the earth ground.
- Sample the ambient contribution from the global probe coming from the top and bottom of the hemisphere. This is an improved version of the above one if the global probe is taking into account some overall tint when integrating the luminance coming from the bottom hemisphere / earth.

The ambient contribution control presented in this sub-section are not physically based but allowed our game teams to reach their visual target.

5.5.2 Sun shadow sampling

In order to generate volumetric shadow, we ray-march toward the sun for each samples taken along the view vector. Ray-marching is done in a straight line toward the sun according to the current sample jittered position (See next Section focusing on *temporal scattering integration*). Shadow samples are taken four times according to a base shadow sample distance that is multiplied by a constant factor for each sample. This in order to progressively sample further away from the source sample. This progressive shadow sample scheme together with the temporal jittering result in smooth/soft shadow estimation.

5.5.3 Temporal scattering integration

We render the clouds in a single pass. Each frame, samples are randomly offset within their sampling step/depth range. Once the current frame solution has been estimated, it is blended with the previous frame solution according to a constant blend factor. This result in a temporal integration of the scattering solution achieved using an exponential moving average. For the final frame to not look blurry when the camera is moving moving/rotating very fast, we re-project the previous result according to previous and current camera properties, *i.e.* projection and transform.

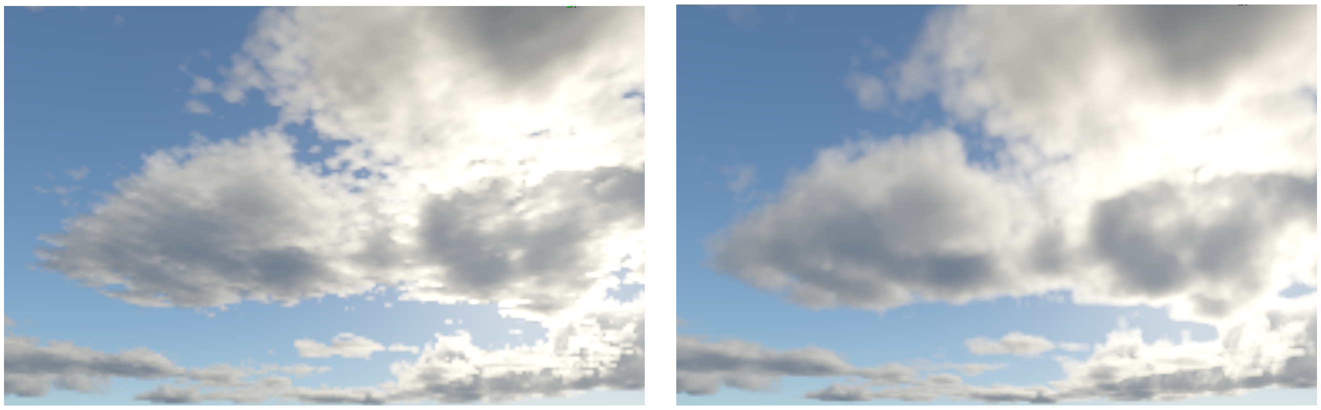


Figure 34: Left: 14 cloud samples without temporal integration. Right: same view and sample count with temporal scattering integration.

This techniques allows us to us a lot less samples per frame while maintaining visual quality. The difference and improvement is visible in Figure 34.

5.6 Improved scattering

When integrating scattered lighting in participating media in real-time, one has to use as few samples as possible to be efficient. One issue that arise in this case is that a single sample will then represents an integration over a larger distance. The larger the distance, the less representative this sample will be, decreasing the accuracy of the integration. We describe here different integration approaches and also propose a new one which give higher quality and is also energy conserving.

5.6.1 Usual scattering integration

The usual and simple way to integrated scattered and transmittance along a ray is presented in Listing 5.6.1. The scattered lighting is initalised to $(0, 0, 0)$ and transmittance to 1. Each step a material sample and a light sample are taken as used to update scattered light and transmittance according to Equation 1.

```

// Contains integrated scattered luminance (in rgb) and trasmittance (in a) along a ray.
float4 intScattTrans = float4(0.0, 0.0, 0.0, 1.0);
for (uint samplerIt = 0; samplerIt < smapleCount; ++samplerIt)
{
    float4 scatteringExtinction        = takeMediaSample(coord);
    const float3 scattering            = scatteringExtinction.rgb;
    const float transmittance          = exp(-scatteringExtinction.a * ds);

    // Get sun luminance according to volumetric shadow and phase function
    const float3 luminance             = sunLuminance(coord, sunDir, viewDir);

    intScattTrans.rgb += scattering * luminance * intScattTrans.a * ds; // (S) step
    intScattTrans.a   *= transmittance;                                 // (T) step
}
```

Listing 1: Simplified code presenting a way to integrate scattered light and transmittance along a ray.

There is one problem with that formulation: inScattTrans.rgb is updated using inScattTrans.a **(S)** and then inScattTrans.a is updated **(T)**. But is the correct order of this steps? In fact, as presented in Figure 35, none is correct. If **(S)** is executed before **(T)**, then scattering will be added without taking into account transmittance over the sampling range ds, resulting in non-energy-conserving integration. If **(T)** is executed before **(S)**, then the resulting participating media will look too bright as the scattered lighting will be over occluded using the entired ds range.

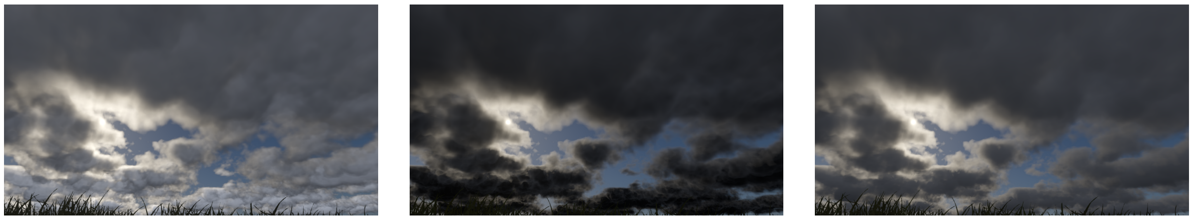


Figure 35: Issues when integrating scattered light with left: **(S)** is executed before **(T)** (not energy conserving), middle: **(T)** is executed before **(S)** (too much absorption of energy), right: a reference integration with many step showing the expect result.

Overall, for non dense material, the error presented in Figure 35 will remain very small. A very simple integration such as the one presented here would work for light fog for instance. However, it will start to break for very dense material, when σ_s becomes high.

5.6.2 Better numerical integrations

To better approximate the definite integral of a curve using a low number of samples, it is possible to use different numerical approaches such as the trapezoidal method [Wikn] or the Simpson's rule [Wikm]. We will take the example of the trapezoidal rule in this section.

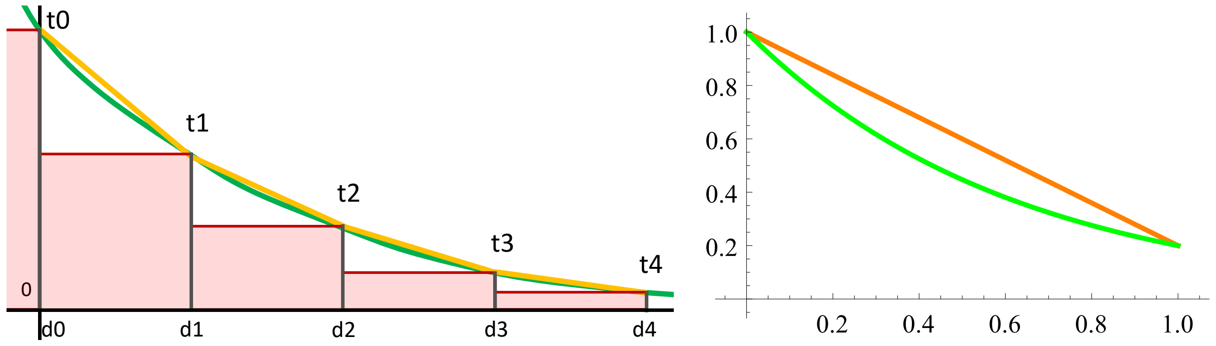


Figure 36: Left: different integration solution, Right: difference between linear and $exp(-x)$ function.

Trapezoide rule describe how a curve integral can be approximated using a few samples and the trapezoidal surface area equation. On the left image of Figure 36, Let's consider the transmittance as the green curve and a scattered light luminance of 1. The transmittance as evaluated by code presented in Listing 5.6.1 with **(T)** executed before **(S)** (*e.g.* $\sum(t_x/(d_{x+1} - d_x)))$ will result in too much absorption: the red squares Y-axis top-cap will always always under the green curve we want to integrate. However, executing **(S)** before **(T)** (*e.g.* $\sum(t_{x+1}/(d_{x+1} - d_x)))$ will result in a non-energy-conserving integral since the Y-axis top-cap of the red squares will always be above the green curve.

Using the trapezoide curve will allow to integrate between each interval $[d_x, d_{x+1}]$ using a piece-wise linear top cap [Wikn]. The trapezoidal integration represent the integration of the orange curve in Figure 36. You can see that the orange curve is a closer match to the green reference curve we want to integrate. However, we can still notice that the integration will not be energy conserving: as visible on the right image of Figure 36, the orange curve is still always above the green reference curve. As a result participating media material will still scatter more light than they should, and the discrepancy will be higher for high value of σ_s, *i.e.* the green curve would converge quicker towards 0 in this example but not the orange curve.

5.6.3 Energy-conserving analytical integration

To solve this issue, we propose to analytically integrate the scattered light over a range according to both extinction σ_t and a scattered light sample $S = Lscat(xt, \omega_i)$ as well as an integration depth d. If we consider taking a single sample for the scattered light sample, we would only have to integrate it for each point on the piece of curve according to transmittance to the front depth of the range. This is achieved using Equation 17 [Hil15].

$$\int_{x=0}^{d} \exp^{-\sigma_t x} \times S dx = \frac{S - S \exp^{-\sigma_t d}}{\sigma_t} \tag{17}$$

Using Equation 17 is staightforward: one only need to take a single σ_t and S per slab. The integrated scattered light can be evaluated using the given aforementioned equation while applying and updating the transmittance of previous integration as shown in Listing 5.6.3. This will result in energy conserving scattering over the considered depth range d. You might have noticed that the result of Equation 17 is undefined when extinction $\sigma_t = 0$. We simply resolve that issue by clamping extinction to a small epsilon.

```

// Contains integrated scattered luminance (in rgb) and trasmittance (in a) along a ray.
float4 intScattTrans = float4(0.0, 0.0, 0.0, 1.0);
for (uint samplerIt = 0; samplerIt < smapleCount; ++samplerIt)
{
    float4 scatteringExtinction       = takeMediaSample(coord);
    const float3 scattering           = scatteringExtinction.rgb;
    const float extinction            = scatteringExtinction.a;
    const float clampedExtinction     = max(extinction, 0.0000001);
    const float transmittance         = exp(-scatteringExtinction.a * ds);

    // Get sun luminance according to volumetric shadow and phase function
    const float3 luminance            = sunLuminance(coord, sunDir, viewDir);
    const float3 integScatt           = (luminance - luminance * transmittance) /
        clampedExtinction;

    intScattTrans.rgb += intScattTrans.a * integScatt;
    intScattTrans.a   *= transmittance;
}
```

Listing 2: Improved analytical scattering integration pseudo code.

Using this integration improvement, it is possible to get participating media to look correct without too many samples when increasing the material density. Figure 37-left show that using the sampling presented in Section 5.6.1, 512 samples needs to be taken for the result to converge towards a correct result mathing the material (as compared to the wrong 21 samples pictures presented in Figure 35). However, on the right, when the scattering integration equation is used, only 21 samples are enough to reach the expected result. As you would expect, the image using 512 samples will result in a more accurate representation of the cloud shape but at least now the lighting result a more independent of the number of sampler.

This analytical integration is simply more correct that the trapezoidal integration in this particular case. This is especially important when for physically based HDR lighting and rendering when contrast and luminance difference can be very large within a scene. For more details about this improved scattering integration formula, please refer to Appendix C.

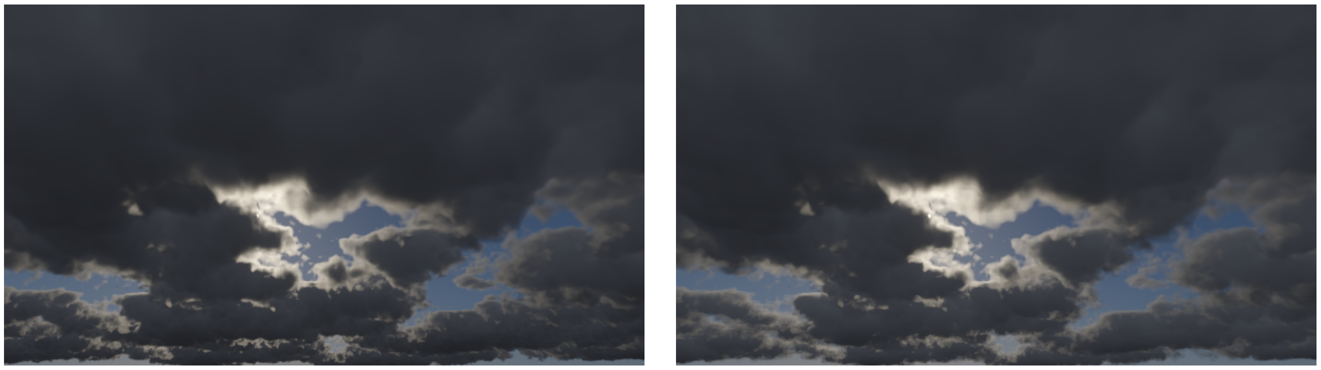


Figure 37: Left: Rendering clouds without Equation 17: 512 samples are needed to converge, and Right: with Equation 17 improved integration showing only 21 samples are needed to converge to an correct/acceptable visual result.

5.7 Cloud phase function

In Section 2.3, we have described the phase function as a mathematical tool representing the bouncing light direction distribution when scattered. This is an important properties of participating media as it defines some very important visual features: from forward scattering resulting in strong silver lining on cloud to subtle wavelength dependent geometric scattering resulting in colored fogbow.

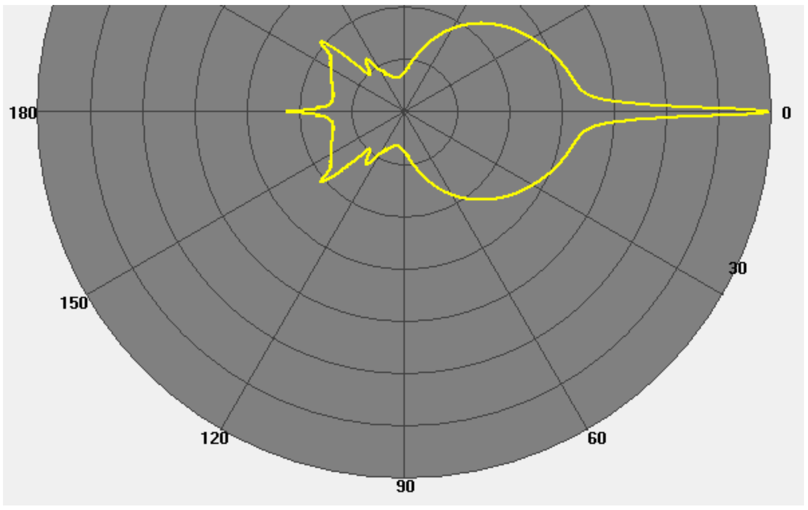


Figure 38: Example of cloud phase function generated with MiePlot [Lav15] (also discussed in [Bou+08]. This shows the scattering for a single light wavelength with all polarised light averaged on a logarithmic scale.

As presented in [Bou+08], the cloud phase function can be very complex. Clouds, being composed of relatively large water droplet, can feature geometric scattering (see Section 2.3). An example such a cloud phase function is visible in Figure 38. You can notice the strong forward scattering spike on the right or also the complex fogbow visual effect at around 120 deg [Wikc]. Other cloud visual features are pseudo specular and glory halo [Wikd].

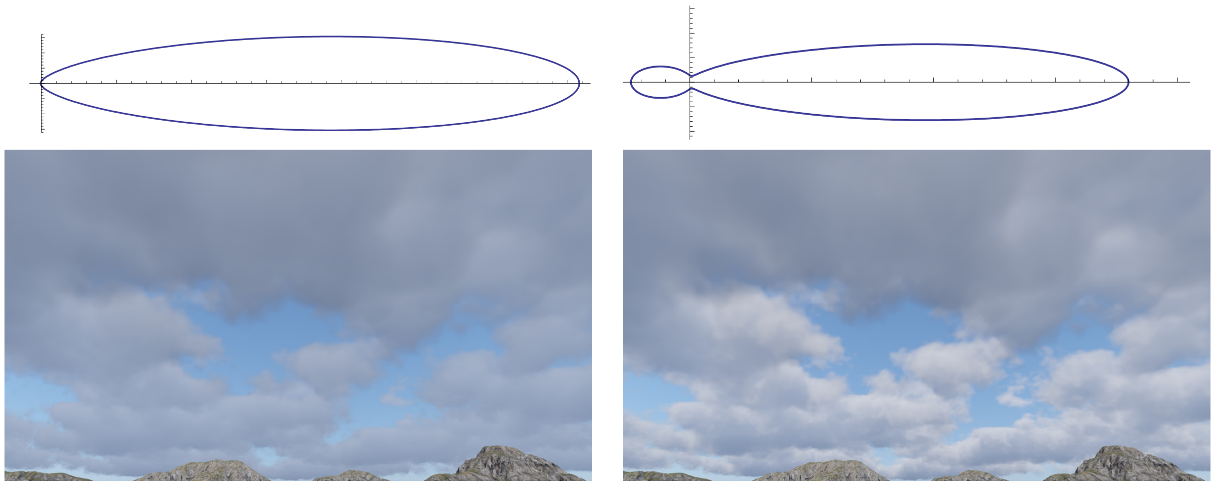


Figure 39: Cloud rendering with sun behind the camera with Left: single-lob p_{hg} and right: dual lobe p_{dual} phase function. The top image show corresponding 2d phase function shapes assuming forward is toward the right.

In a real-time context such as video games, we do not have the luxury of evaluating such complex phase function shape. Thus a single phase function is usually evaluated, (*e.g.* a Henyey-Greenstein phase function). But when representing material featuring strong forward scattering such as cloud, the back scattering component can them become missing and cloud view from an opposite direction from the sun direction can look dull and lack details as visible in Figure 39 left image. Indeed, with a strong forward peak, only ambient lighting would remain when looking at clouds when the sun is behind the camera. To resolve this issue, we use a dual-lob phase function p_{dual} consisting of two

Henyey-Greenstein phase functions blended together according to a weight $w \in [0,1]$ as shown in Equation 18.

$$p_{dual}(\theta, g_0, g_1, w) = lerp((p_{hg}(\theta, g_0), p_{hg}(\theta, g_1), w) \tag{18}$$

Using a dual lobe phase function gives artists a lot more control over the way light will bounce in the cloud participating media. It is now possible to better balance forward and backward scattering. Figure 39, right image, shows that a lot more details can be achieved by using a dual lobe phase function allowing both a strong forward scattering peak while also maintaining some amount of backward scattering to bring our more details from the clouds shape.

5.8 Multiple scattering

Clouds scatter lights many and a huge part of their bright and white look is the result of multi-scattering. Without multi-scattering cloud would mostly be lit sun and ambient at their edges, and would be very dark anywhere else. Multi-scattering is also a key component for clouds to not look like smoke. With the massive amount of water suspended in the air, puffy clouds can look very white and bright, even when lit by the very strongly blue tinted environment sky light scattering. As reported at he begining of this Section, cloud albedo is very clost to 1.

Different methods can be used to evaluate multi-scattering solutions:

- Path tracing (recursing): it would largely be out of budget for real time game use cases.
- Pre-computed: this is similar to the collector based approach proposed by [Bou+08]. However it would likely be hard to pre-compute for procedural content.
- Iterative: Similar to [Ele+14] One could propagate multi-scattered light in volume. Although automatic, we would however likely still end up out-of-budget with this technique due to the amount of memory required.

In the end we settled to use the very simple multi-scattering approximation proposed by Wrenninge *et al.* [WKL13]. The method basically integrate multiple *octave* of scattering and sum them.

So basically, the final integrated scattering is:

$$Lmultiscat(x, \omega_i) = \sum_{0}^{(N-1)} Lscat(x, \omega_i) \tag{19}$$

Where the following substitutions are made:

$$\begin{aligned} \sigma_s' &= \sigma_s \times a^n \\ \sigma_e' &= \sigma_e \times b^n \\ p'(\theta) &= p(\theta \times c^n) \end{aligned} \tag{20}$$

In order to make sure this technique energy conserving when evaluating $Lmultiscat(x, \omega_i)$, one must ensure that $a <= b$. Otherwise more light can be scattered than expected because equation $\sigma_t = \sigma_a + \sigma_s$ would not be respected any more since σ_s could end up being larger than σ_t.

The advantage of this solution is that one can integrate the scattered light for each of the different octaves *while* raymarching, all at once. The drawback is that it does not represent well complex multi-scattering behavior: for instance side or backward scattering, no cone spread, *etc.* Despite these drawbacks, the technique works very well in practice and gives artists a fine grained control of the look of volumetric clouds. It is now possible for them to generate highly scattering, *i.e.* thick, participating media while still making sure the scattered light can punch through the medium in order to reveal inner details on the shadowed sides 40.

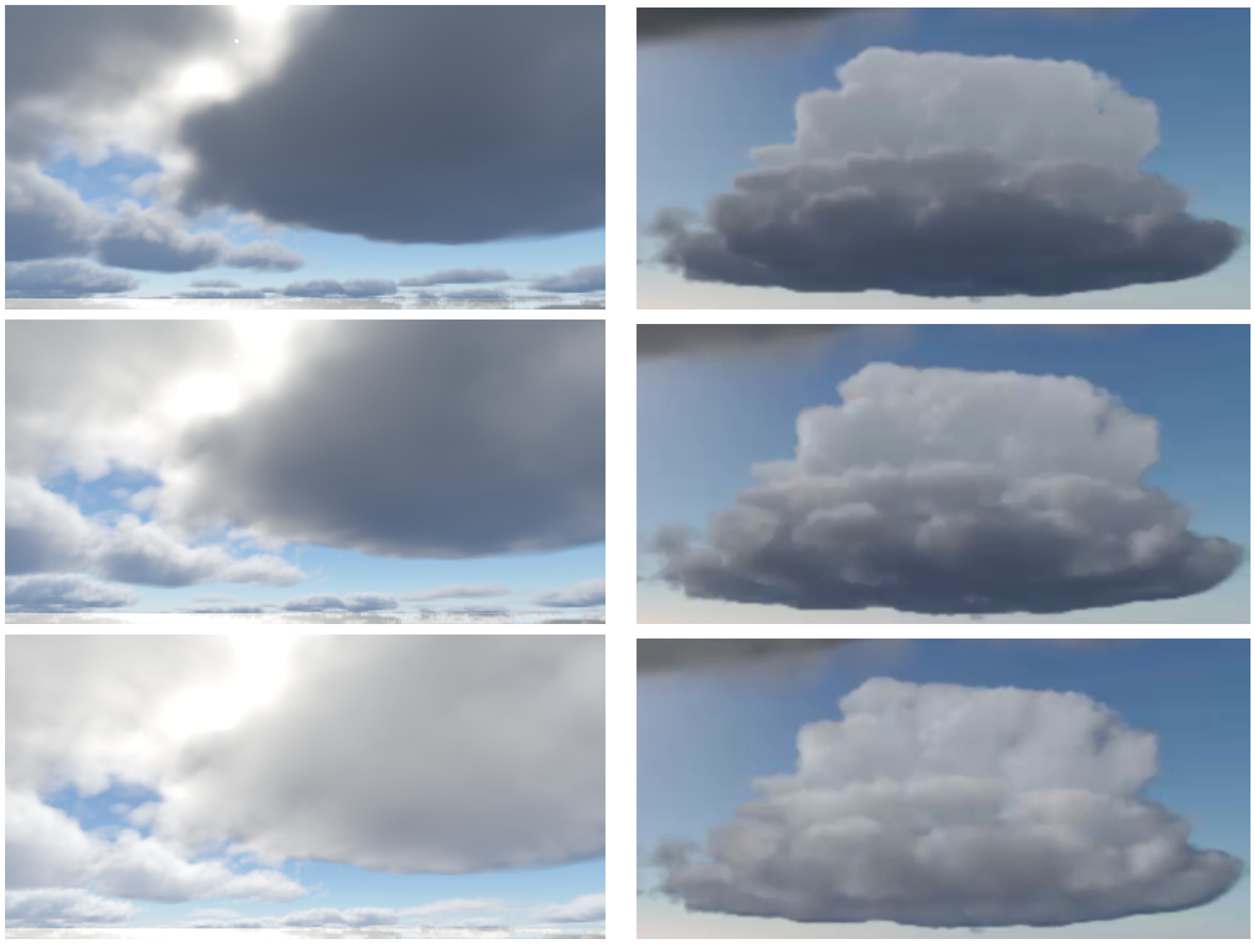


Figure 40: Exemple of cumulus (left) and cumulonimbus (right) clouds rendered with single scattering N=1 (top) and multi scattering with N=2 (middle) and N=3 with exaggerated multi scattering (bottom)

5.9 Other interactions

Frostbite volumetric cloud shadow is built in completely part of the engine. It is expected that cloud will then interacts consistently with every elements of a scene. That in order to get everything to look consistent, whatever the planet, the time of day and the weather are.

Clouds are taken into account when evaluating shadows or atmosphere scattering which enable many global effects to happen:

- Shadow: volumetric cloud shadows are baked into a 2D texture storing transmittance and projected onto the world. It is applied on all opaque, transparent surfaces, and also sampled by our emitter system to correctly lit particles. The projection is very simple and assumes a overall flat planet around the camera, which is reasonable to assume for typical planets.
- GI: the volumetric cloud shadow map is also sampled when updating input to our dynamic global illumination system (achieved using Enlighten [Geo17]). This results in dynamic global illumination being influenced by the cloud, weather and sun direction.
- Clouds affecting aerial perspective: described in Subsection 5.9.2.
- Aerial perspective affecting clouds: described in Subsection 5.9.1.

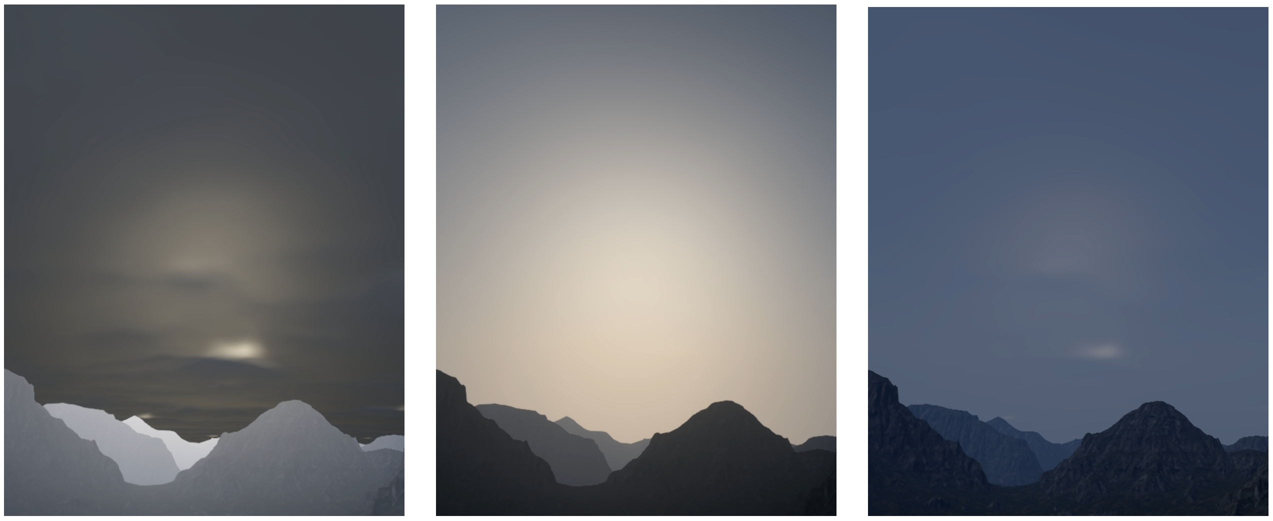


Figure 41: Important clouds and aerial perspective interactions. Left: heavy cloud layer simply composited over a bright day sky, Middle: aerial perspective applied on clouds, and Right: the correct result with clouds layer coverage affecting the aerial perspective.

5.9.1 Aerial perspective affecting clouds

Clouds are usually very high in the atmosphere and far away from the camera. As a result their looks is affected by the aerial perspective, *i.e.* atmospheric light scattering happens in between cloud particle and the view point. Figure 41-left shows that if clouds are rendered without taking into account the aerial perspective, a visual discrepancy happens at the horizon between clouds and earth. The middle image shows the same scene but with aerial perspective applied to the cloud. The issue at the horizon is no longer visible.

$$Depth_{cloud} = \frac{\sum_{n=0}^{N} Tr(x_n) * Depth(x_n)}{\sum_{n=0}^{N} Tr(x_n)} \tag{21}$$

On straightforward way to apply the aerial perspective on the cloud is to sample it while integrating the scattered light and transmittance during the cloud layer ray-marching. However, all those extra texture samples would make cloud passes more expensive. Instead we propose to achieve that goal in two steps:

- Compute the cloud mean front depth weighted by the transmittance to the view point using Equation 21). This allows to evaluate a smooth front depth while taking into account the visibility of each sample and ignoring the depth of occluded samples. We simply skip samples if no cloud particle have been hit, *e.g.* when transmittance is 1. This result in a smooth depth buffer as visible in Figure 42.
- Evaluate the aerial perspective scattering/transmittance texture only once and apply it onto the final integrated cloud scattered luminance and transmittance.

5.9.2 Clouds affecting aerial perspective

Clouds covering the sky will influence the aerial perspective look due to their coverage and the light they scatter around. Under the cloud layer:

- a thick cloud layer will block sky light from scattering in the atmosphere.

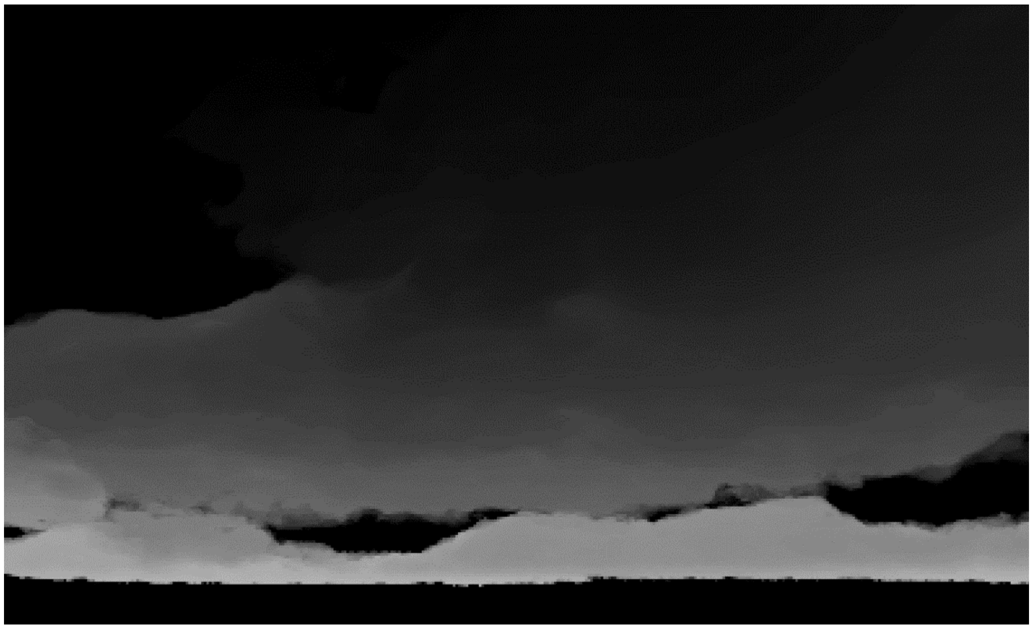


Figure 42: Weighted mean of cloud samples depth. This depth is used to sample the aerial perspective scattering and transmittance applied on clouds as a post-process.

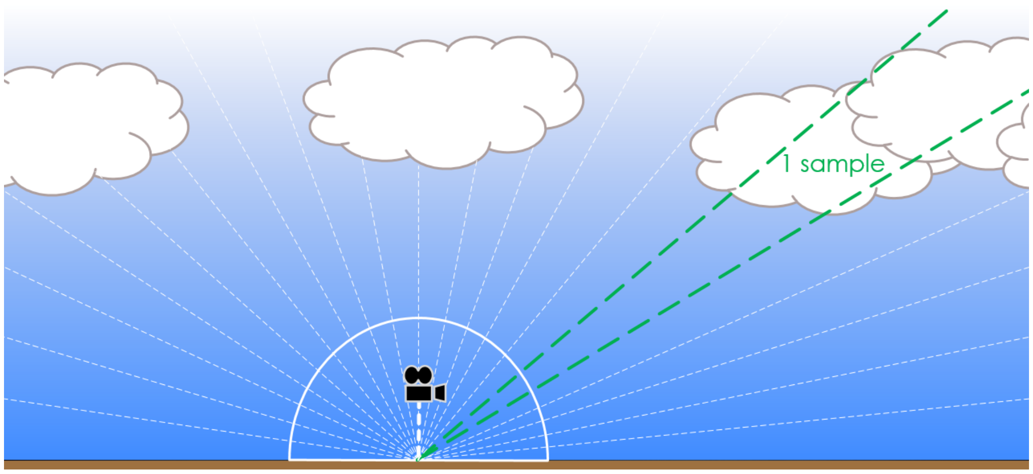


Figure 43: Sketch illustrating integration of cloud scattered luminance and transmittance.

- a bright cloud resulting from high albedo will propagate more light in the atmosphere. The participating media and density coefficients will influence the amount and color of the scattered light.

$$L_{AP} = L_{AP} \times Tr_{cloud} + L_{cloud} \tag{22}$$

As presented in [Hil16], see Figure 43, we sample the cloud *mean transmittance* Tr_{cloud} and *integrate scattered luminance* L_{cloud} over the hemisphere from the camera point of view at the ground level. This is achieved by simply rendering the clouds over a black background and storing per pixel scattered luminance and transmittance.

This result is then used to affect the aerial perspective texture presented in Section 3.5 using equation 22. We simply attenuates aerial perspective scattered sun luminance according to the cloud layer mean transmittance, while adding cloud integrated scattered luminance contribution instead.

The improvement is visible in Figure 41 where the middle image have the aerial perspective unaffected by the cloud layer and the right image has it affected. You can notice how a lot less scattering is present in the atmosphere and that a dark blue scattered color form the dark opaque cloud layer is also visible.

5.10 Performance

In order to achieve coherent visuals, unified lighting and shadowing, clouds must be rendered in multiple views: main, planar reflection, environment map or shadow. In this section we give our latest performance result on XBox One.

View	Performance
720p Main	0.91 ms
720p planar reflection	0.14 ms
1080p Main	1.60 ms
1080p planar reflection	0.20 ms

Table 4: Cloud rendering performance on XBox One.

The performance given in Figure 4 are given for the worst case when looking at the horizon, 3/5 of the screen covered by clouds and 16 samples per pass. Main cloud view is rendered at half resolution and planar reflection a fourth of the the resolution. This performance is also given for multi scattering enabled with N=2 (see Section 5.8).

5.11 Results

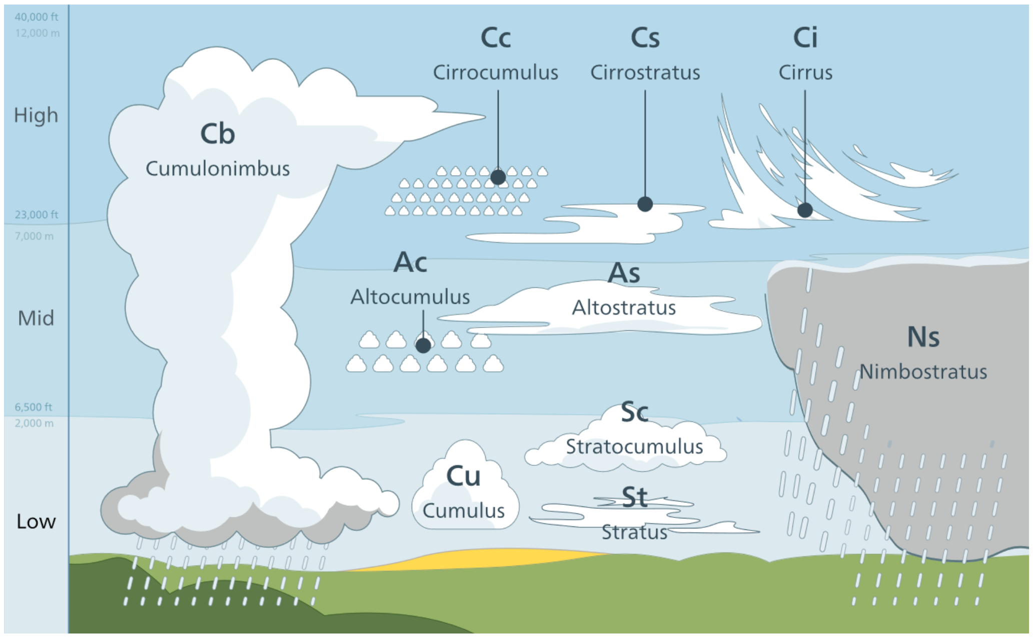


Figure 44: Illustration by Valentin de Bruyn [Wike]. Licence: Creative Commons Attribution-Share Alike 3.0.

Additionally to the results shown in previous sections, we present here some extra visuals obtained after attempting to match different cloud types. Indeed, as visible in Figure 44 clouds can really get a wide variety of shape, density, height, *etc.* Thanks a lot to Soren Hesse for sharing some of his work here.

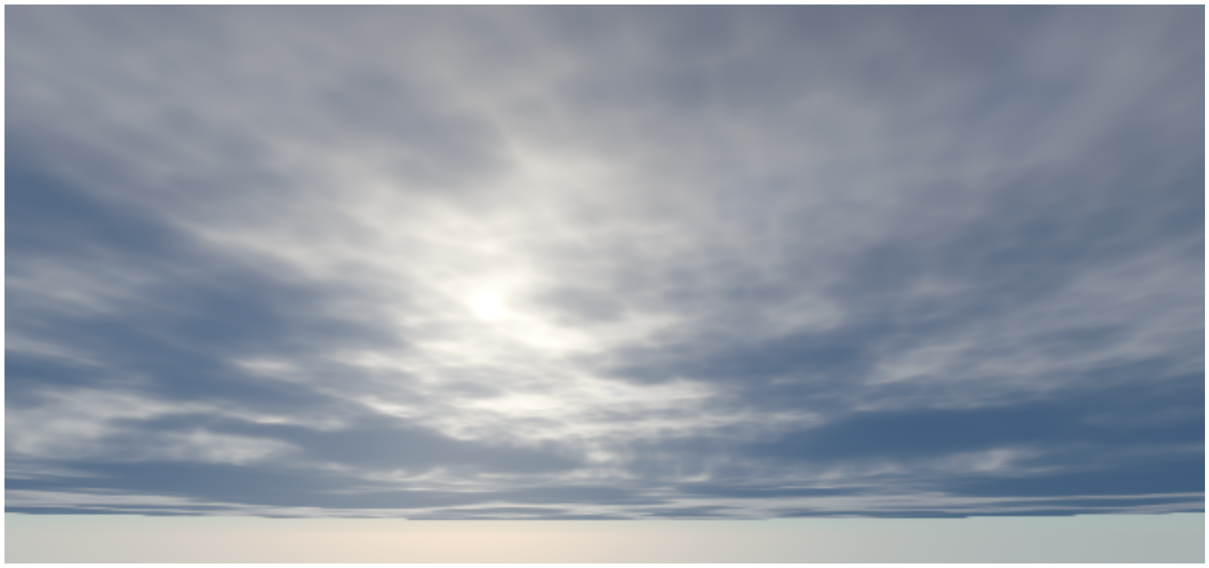


Figure 45: Altostratus clouds.

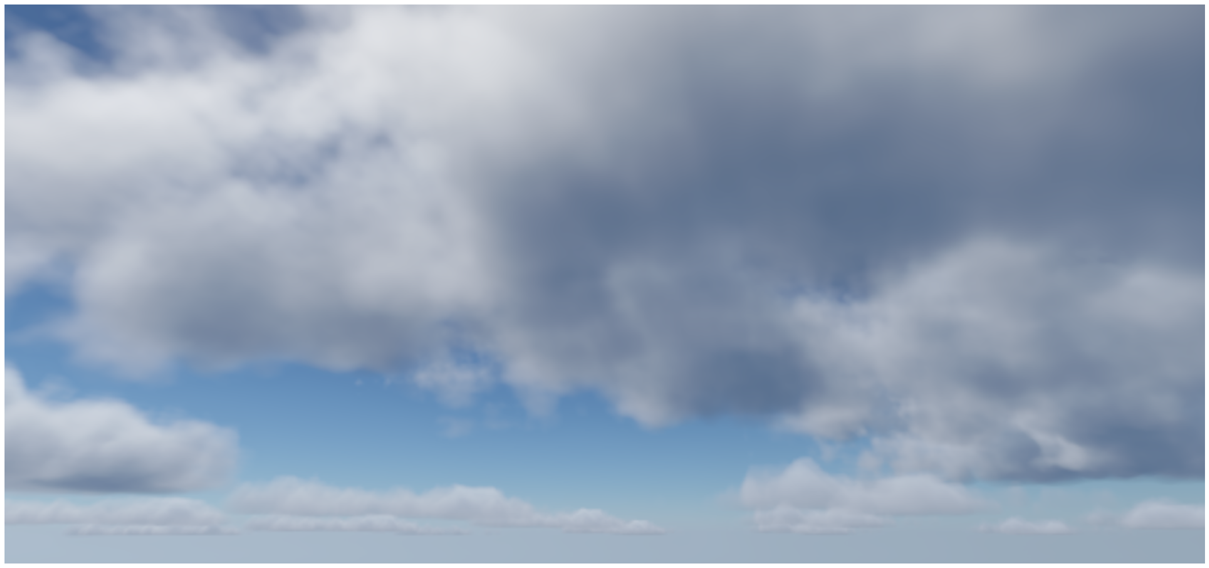


Figure 46: Cumulus clouds.

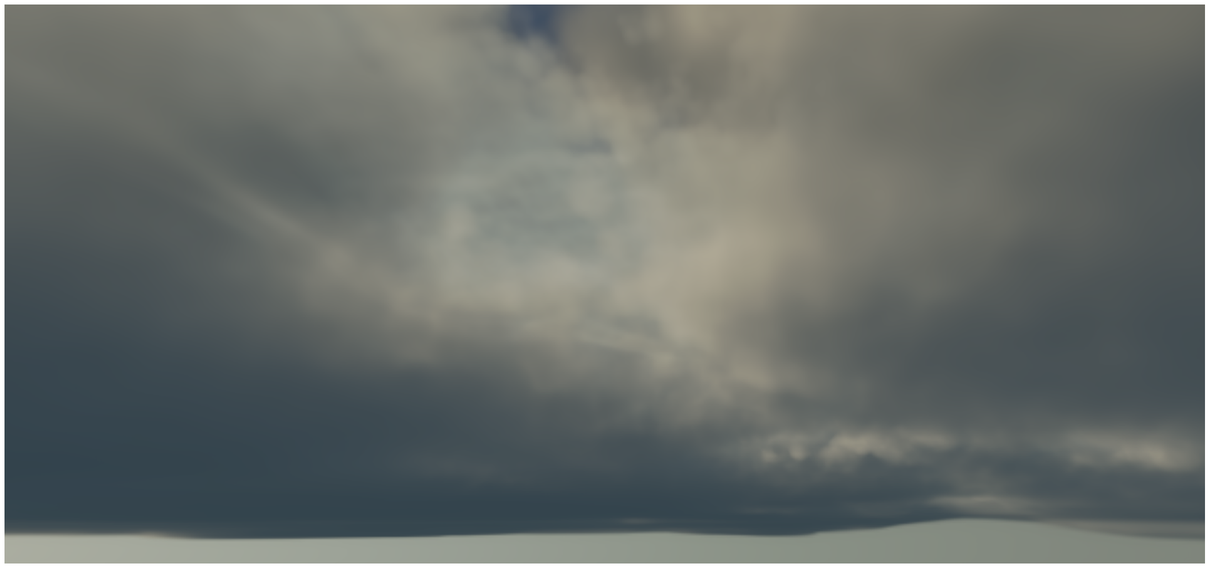


Figure 47: Some menacing large and thick cumulus clouds announcing heavy rain at dusk.

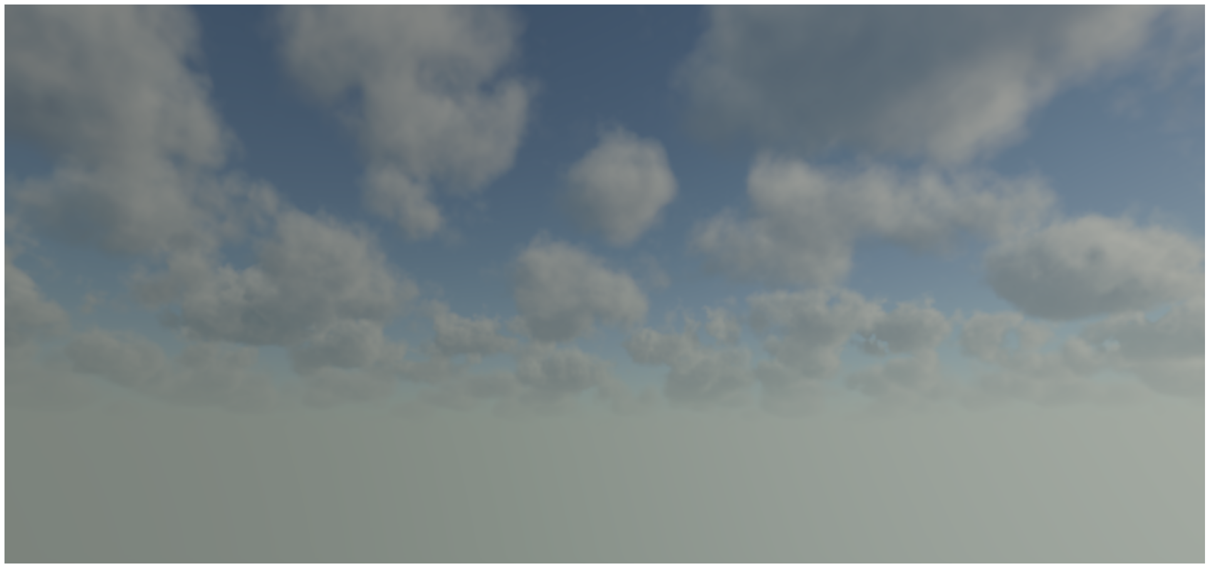


Figure 48: Many stratocumulus at dusk.

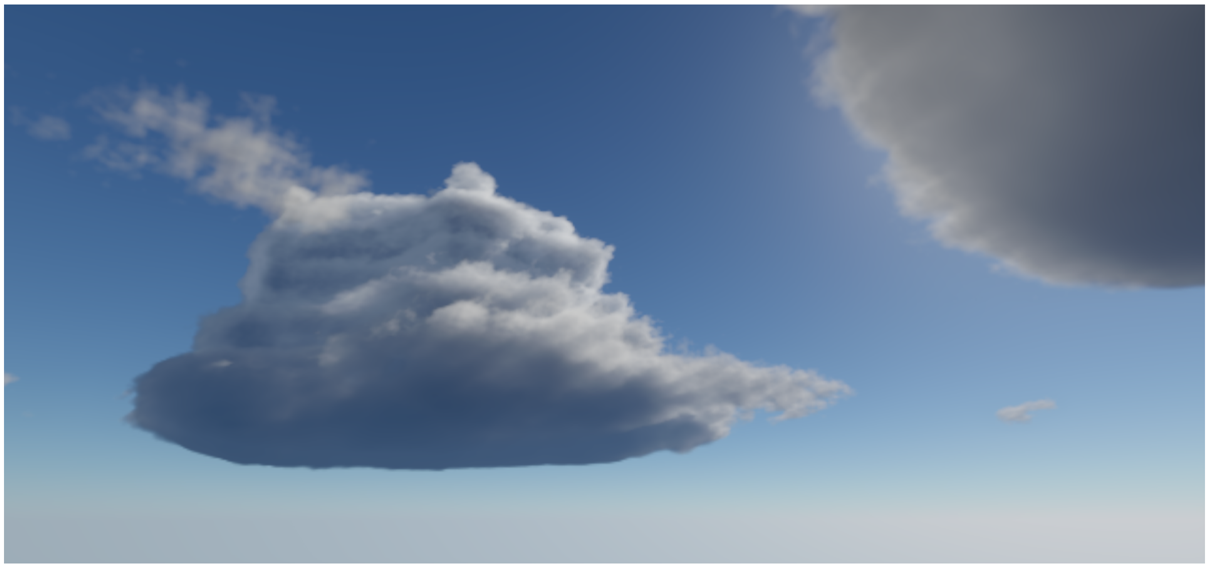


Figure 49: Large and very dense cumulonimbus cloud.

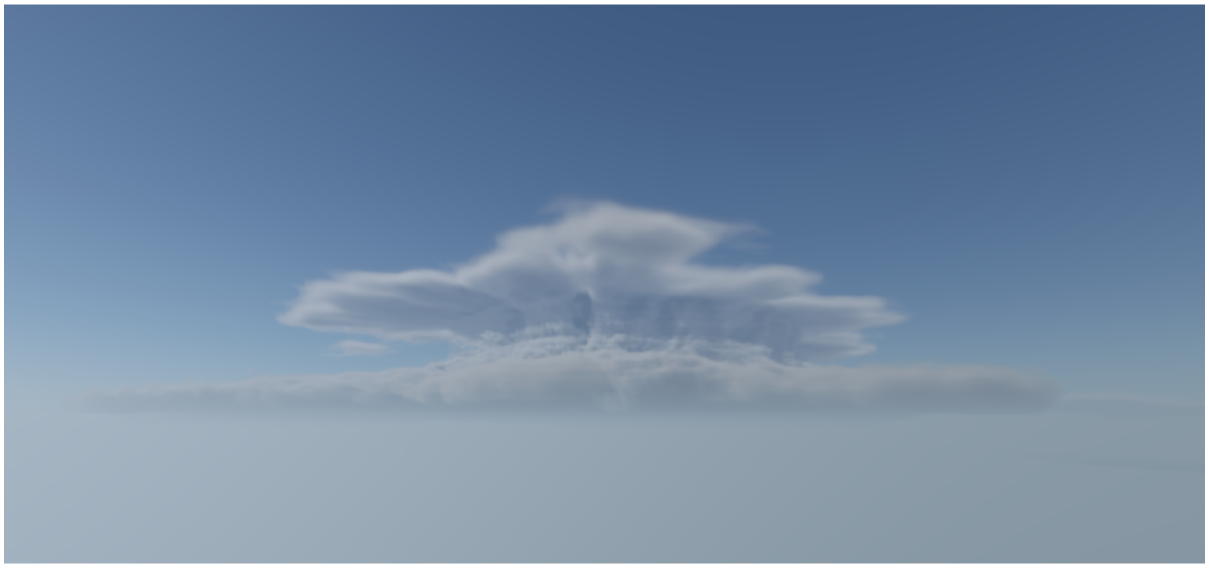


Figure 50: Far away massive and tall cumulonimbus anvil cloud coming our direction. Storm incoming!

Need for Speed Payback [Gho17] will be the first Electronic Arts game shipping with the volumetric cloud rendering technology. Results are visible in Figure 51.

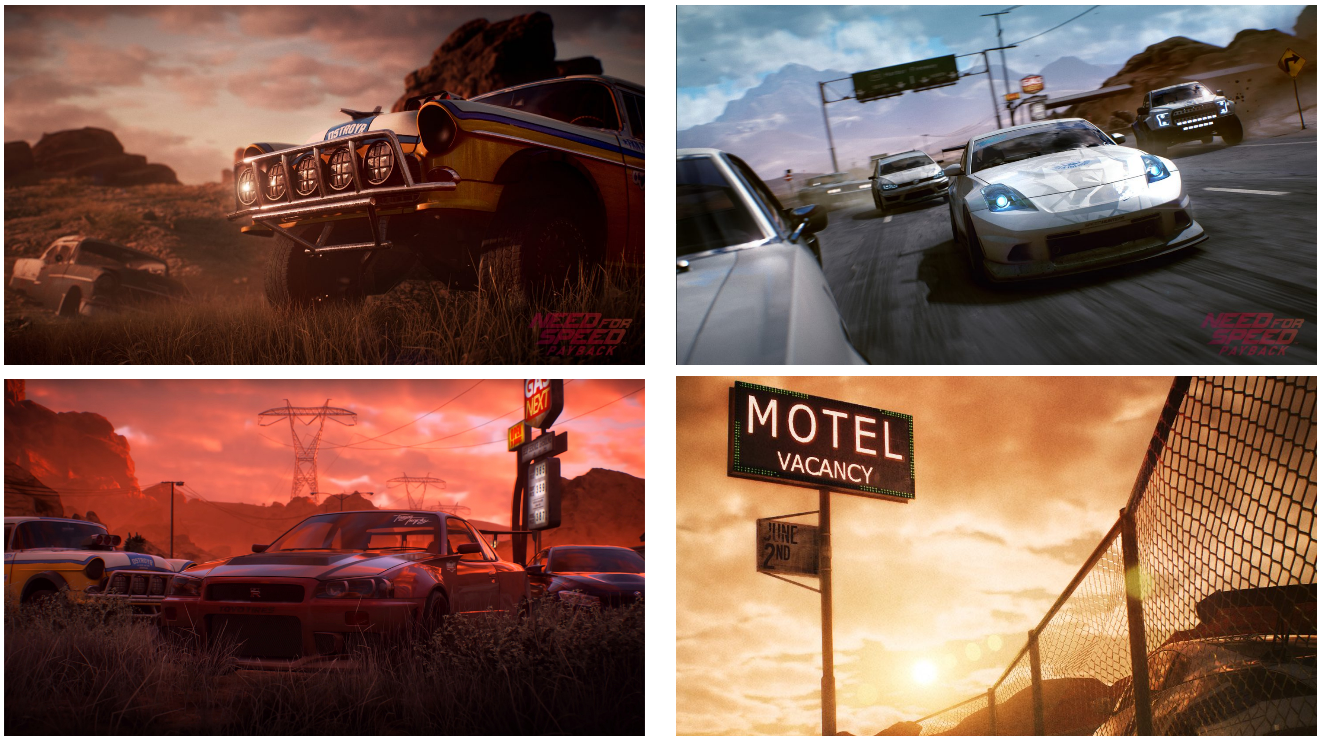


Figure 51: Need for Speed Payback official screen shots featuring Frostbite physically based volumetric clouds.

6 Conclusion

We have presented techniques to render a physically based sky, atmosphere, celestials and clouds. We have detailed the particular implementation in Frostbite used by many EA games in production. The techniques presented in this document have also been used in many recent Electronic-Arts games visible in Figure 52. The cloud rendering technique is the only one that has not be shipped yet but is already used in production for Need for Speed Payback [Gho17] and Anthem games [Bio17].

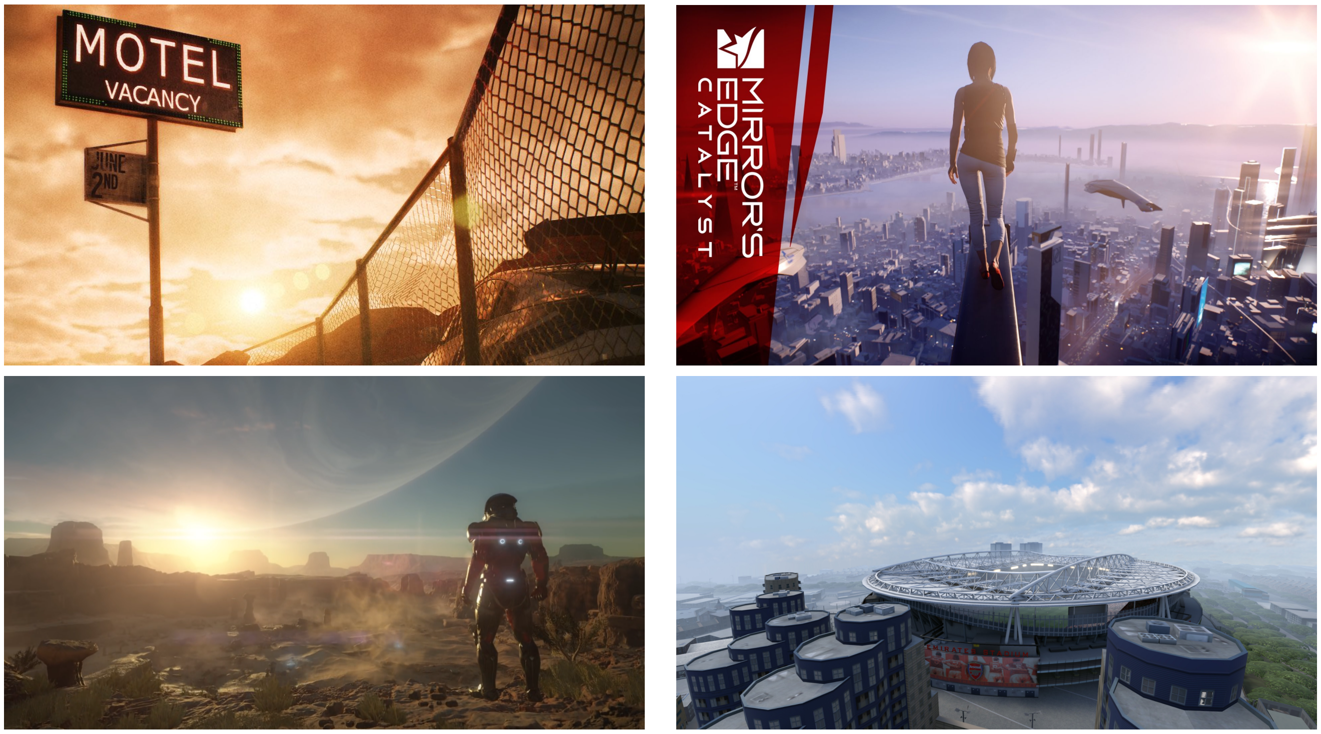


Figure 52: Recent Electronic Arts / Frostbite games has shipped using the techniques presented in this document. Top-Left to Bottom-Right: Need for Speed Payback [Gho17], Mirror's Edge Catalyst [DIC16], Mass Effect Andromeda [BIO17] and FIFA 17 [Art16]

The main challenge encountered during the development of all these games was to maintain a high visual quality under strong constraints. Those constraints drove many of the technical choices we had to do and presented in this document. Firstly performance: Mirror's Edge and FIFA both are games running at 60 frames per seconds, so a minimal GPU cost is necessary. Secondly the interaction between many systems: changing cloud parameters which in turn influence atmosphere scattering, global illumination, *etc* was also challenging from an implementation and visual coherence point of view. Finally all these systems had to be design for real-time preview by artists with minimal updates latency to maintain visual coherence while conditions, *e.g.* time of day or weather, evolve in real-time.

We hope this document will be helpful: (1) that new comers have found the participating media section to be complete and detailed enough to understand all the basic concept behind volumetric rendering, whether you are an artist or programmer, and (2) that the remaining parts of this document will help you implement and/or improve your sky, cloud and celestial rendering systems. If you would like to get more details about some of the discussions, equations or techniques presented, or if you have found any issue or a typo: please do not hesitate to reach out: sebastien.hillaire@frostbite.com or https://twitter.com/SebHillaire.

6.1 Future work

We give here ideas and areas of investigations that could be researched to improve the techniques presented in this documents:

- Aerial perspective sun volumetric shadow from clouds. Solutions have been proposed in previous papers [BN08][Yus13a] but they were aimed at volumetric shadow from opaque geometry only. We need a solution that would also take into account opqaue as well as participating media, e.g. clouds or local fog volumes [Hil15]. Maybe using a camera wrapped froxel volume or a special shadow projection mapped on the frustum.
- Clouds are rendered in a layer behind everything. If one would want clouds to intersects with opaque geometry such a big mountain, it would need to only ray-march up to closest depth. Since clouds are rendered at lower resolution, extra compositing steps would be required: downsampled depth and bilateral upsampling from low resolution to full resolution.
- Cloud ambient is a single color as of today without directionality nor occlusion. One could improve the quality by using Frostbite spherical harmonic probe main incoming light incoming direction, or a single ambient occlusion could be temporally integrated according to different direction each frame in the cloud volume. One could also taking into account the bounce color from the terrain.
- Implement Bruneton's model (4D multi-scattering LUT) or find a cheap way to approximate the earth shadowing of the atmosphere would be interesting to have a an option. In some cases, we have find that having such an visual option would be interesting.

References

[Art16] E. Arts. *FIFA 2017*. 2016. URL: https://www.easports.com/fifa.

[BIO17] BIOWARE. *Mass Effect Andromeda*. 2017. URL: https://www.masseffect.com/.

[Bio17] Bioware. *Anthem*. 2017. URL: https://www.ea.com/games/anthem.

[BN08] E. Bruneton and F. Neyret. "Precomputed Atmospheric Scattering". In: *Proceedings of the Nineteenth Eurographics Conference on Rendering*. EGSR '08. Sarajevo, Bosnia and Herzegovina: Eurographics Association, 2008, pp. 1079–1086. DOI: 10.1111/j.1467-8659.2008.01245.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2008.01245.x.

[Bou+08] A. Bouthors, F. Neyret, N. Max, E. Bruneton, and C. Crassin. "Interactive Multiple Anisotropic Scattering in Clouds". In: *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*. I3D '08. Redwood City, California: ACM, 2008, pp. 173–182. ISBN: 978-1-59593-983-8. DOI: 10.1145/1342250.1342277. URL: http://doi.acm.org/10.1145/1342250.1342277.

[Bru17] E. Bruneton. *Precomputed Atmospheric Scattering*. 2017. URL: https://github.com/ebruneton/precomputed_atmospheric_scattering.

[BS] G. Bodare and E. Sandberg. *Efficient and Dynamic Atmospheric Scattering*. URL: http://publications.lib.chalmers.se/records/fulltext/203057/203057.pdf.

[Chr16] F. Christin. "Lighting the City of Glass". In: Game Developers Conference. 2016. URL: http://www.gdcvault.com/play/1023284/Lighting-the-City-of-Glass.

[CIE95] CIE. "Spatial distribution of daylight-luminance distributions of various reference skies". In: *Color Research and Application* 20.1 (1995), pp. 80–81. ISSN: 1520-6378. DOI: 10.1002/col.5080200119. URL: http://dx.doi.org/10.1002/col.5080200119.

[CK74] G. P. Charles Adams and G. Kattawar. "The influence of Ozone and Aerosol on the Brightness and Color of the twilight Sky". In: *Journal of the Atmospheric Sciences* 31 (1974), pp. 1662–1674. URL: http://journals.ametsoc.org/doi/pdf/10.1175/1520-0469(1974)031%3C1662:TIOOAA%3E2.0.CO%3B2.

[dEo16] E. d'Eon. *A HitchHicker's Guide to Multiple Scattering*. 2016. URL: http://www.eugenedeon.com/project/a-hitchhikers-guide-to-multiple-scattering/.

[DIC13] DICE. *Battlefield4*. 2013. URL: https://www.battlefield.com/games/battlefield-4.

[DIC15] DICE. *Star Wars Battlefront*. 2015. URL: http://starwars.ea.com/starwars/battlefront.

[DIC16] DICE. *Mirror's Edge catalyst*. 2016. URL: http://www.mirrorsedge.com/.

[DL07] E. D'Eon and D. Luebke. *Advanced techniques for realistic real-time skin rendering*. Addison Wesley, 2007. URL: http://http.developer.nvidia.com/GPUGems3/gpugems3_part03.html.

[Ele+14] O. Elek, T. Ritschel, C. Dachsbacher, and H.-P. Seidel. "Principal-Ordinates Propagation for Real-Time Rendering of Participating Media". In: *Computers & Graphics* 45 (2014). DOI: 10.1016/j.cag.2014.08.003.

[Ele09] O. Elek. "Rendering Parametrizable Planetary Atmospheres with Multiple Scattering in Real-time". In: *CESCG* (2009). URL: http://www.cescg.org/CESCG-2009/papers/PragueCUNI-Elek-Oskar09.pdf.

[Geo17] Geomerics. *Enlighten*. 2017. URL: http://www.geomerics.com/enlighten/.

[Gho15] Ghost. *Need for Speed*. 2015. URL: https://www.needforspeed.com.

[Gho17] Ghost. *Need for Speed Payback*. 2017. URL: https://www.ea.com/games/need-for-speed/need-for-speed-payback.

[Gue14] K. Guerrette. “Moving the heavens”. In: Game Developers Conference. 2014. URL: http://www.gdcvault.com/play/1020146/Moving-the-Heavens-An-Artistic.

[Har02] M. J. Harris. “Real-Time Cloud Rendering for Games”. In: Game Developers Conference. 2002. URL: http://www.markmark.net/PDFs/RTCloudsForGames_HarrisGDC2002.pdf.

[Hila] S. Hillaire. *O3 Spectrum to RGB absorption*. URL: https://www.shadertoy.com/view/ldySWV.

[Hilb] S. Hillaire. *Volumetric Stanford Bunny*. URL: https://www.shadertoy.com/view/MdlyDs.

[Hilc] S. Hillaire. *VolumetricIntegration*. URL: https://www.shadertoy.com/view/XlBSRz.

[Hil15] S. Hillaire. “Physically Based and Unified Volumetric Rendering in Frostbite”. In: *Advances in Real Time Rendering, Part I, ACM SIGGRAPH 2015 Courses*. SIGGRAPH ’15. Los Angeles, California: ACM, 2015. ISBN: 978-1-4503-3634-5. DOI: 10.1145/2776880.2787701. URL: http://advances.realtimerendering.com/s2015/.

[Hil16] S. Hillaire. “Physically Based Sky, Atmosphere and Cloud Rendering in Frostbite”. In: *SIGGRAPH 2016 Course: Physically Based Shading in Theory and Practice, ACM SIGGRAPH 2016 Courses*. SIGGRAPH ’16. Los Angeles, California: ACM, 2016. URL: http://blog.selfshadow.com/publications/s2016-shading-course/.

[HKS98] M. Hess, P. Koepke, and I. Schult. “Optical Properties of Aerosols and Clouds: The Software Package OPAC”. In: *Bulletin of the American Meteorological Society* 79.5 (1998), pp. 831–844. DOI: 10.1175/1520-0477(1998)079<0831:OPOAAC>2.0.CO;2. eprint: http://dx.doi.org/10.1175/1520-0477(1998)079<0831:OPOAAC>2.0.CO;2. URL: http://dx.doi.org/10.1175/1520-0477(1998)079%3C0831:OPOAAC%3E2.0.CO;2.

[HM98] D. Hestroffer and C. Magna. “Wavelength dependency of the Solar limb darkening”. In: *Astronomy Astrophysics* 333.1 (1998), pp. 338–342. URL: https://www.researchgate.net/publication/234185016_Wavelength_dependency_of_the_Solar_limb_darkening.

[Hög] R. Högfeldt. *Convincing Cloud Rendering: An Implementation of Real-Time Dynamic Volumetric Clouds in Frostbite*. URL: http://publications.lib.chalmers.se/records/fulltext/241770/241770.pdf.

[Hul57] H. C. V. de Hulst. *Light Scattering by Small Particles*. Reprint, edition (December 1, 1981). Dover Publications, 1957. ISBN: 978-0486642284.

[HW12] L. Hosek and A. Wilkie. “An Analytic Model for Full Spectral Sky-dome Radiance”. In: *ACM Trans. Graph.* 31.4 (July 2012), 95:1–95:9. ISSN: 0730-0301. DOI: 10.1145/2185520.2185591. URL: http://doi.acm.org/10.1145/2185520.2185591.

[Jar08] W. Jarosz. “Efficient Monte Carlo Methods for Light Transport in Scattering Media”. PhD thesis. UC San Diego, Sept. 2008. URL: https://www.cs.dartmouth.edu/~wjarosz/publications/dissertation/.

[JB10] J. Jansen and L. Bavoil. “Fourier Opacity Mapping”. In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’10. Washington, D.C.: ACM, 2010, pp. 165–172. ISBN: 978-1-60558-939-8. DOI: 10.1145/1730804.1730831. URL: http://doi.acm.org/10.1145/1730804.1730831.

[Jen+01a] H. W. Jensen, F. Durand, J. Dorsey, M. M. Stark, P. Shirley, and S. Premože. “A Physically-based Night Sky Model”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. New York, NY, USA: ACM, 2001, pp. 399–408. ISBN: 1-58113-374-X. DOI: 10.1145/383259.383306. URL: http://doi.acm.org/10.1145/383259.383306.

[Jen+01b] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan. “A Practical Model for Subsurface Light Transport”. In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’01. New York, NY, USA: ACM, 2001, pp. 511–518. ISBN: 1-58113-374-X. DOI: 10.1145/383259.383319. URL: http://graphics.ucsd.edu/~henrik/papers/bssrdf/.

[Kut13] P. Kutz. *The Importance of Ozone*. 2013. URL: http://skyrenderer.blogspot.se/2013/05/the-importance-of-ozone.html.

[Lav15] P. Laven. *MiePlot*. 2015. URL: http://www.philiplaven.com/mieplot.htm.

[LR14] S. Lagarde and C. de Rousiers. “Moving Frostbite to PBR”. In: *Physically Based Shading in Theory and Practice, ACM SIGGRAPH 2014 Courses*. SIGGRAPH ’14. Vancouver, Canada: ACM, 2014, 23:1–23:8. ISBN: 978-1-4503-2962-0. DOI: 10.1145/2614028.2615431. URL: http://www.frostbite.com/2014/11/moving-frostbite-to-pbr/.

[Ltd] T. I. Ltd. *Reset blog*. URL: http://reset-game.net/?p=284.

[MH16] S. McAuley and S. Hill. “Physically Based Shading in Theory and Practice”. In: *ACM SIGGRAPH 2016 Courses*. SIGGRAPH ’16. Anaheim, California: ACM, 2016. ISBN: 978-1-4503-4289-6. DOI: 10.1145/2897826.2927353. URL: http://doi.acm.org/10.1145/2897826.2927353.

[Mie08] G. Mie. “Beiträge zur optik trüber medien, speziell kolloidaler metallösungen”. In: *Annalen der Physik*. 1908, pp. 377–445. URL: http://www.dca.iag.usp.br/www/material/akemi/radiacao-I/Mie_Horvath%20(2009).pdf.

[NAS05] NASA. *Sunset on Mars*. 2005. URL: http://www.nasa.gov/multimedia/imagegallery/image_feature_347.html.

[NAS16] NASA. *Rover Opportunity Wrapping up Study of Martian Valley*. 2016. URL: https://www.nasa.gov/feature/jpl/rover-opportunity-wrapping-up-study-of-martian-valley.

[Nec96] H. Neckel. “On the wavelength dependency of solar limb darkening ($\lambda\lambda$303 to 1099 nm)”. In: *Solar Physics* 167.1 (1996), pp. 9–23. ISSN: 1573-093X. DOI: 10.1007/BF00146325. URL: http://dx.doi.org/10.1007/BF00146325.

[Ney] F. Neyret. *Realistic display of star in Hubble images*. URL: https://www.shadertoy.com/view/XdsGWs.

[Nis+93] T. Nishita, T. Sirai, K. Tadamura, and E. Nakamae. “Display of the Earth Taking into Account Atmospheric Scattering”. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’93. Anaheim, CA: ACM, 1993, pp. 175–182. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166140. URL: http://doi.acm.org/10.1145/166117.166140.

[ONe07] S. O’Neil. *Accurate Atmospheric Scattering*. Addison Wesley, 2007. URL: http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter16.html.

[PH10] M. Pharr and G. Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN: 0123750792, 9780123750792. URL: http://www.pbrt.org/.

[PSS99] A. J. Preetham, P. Shirley, and B. Smits. “A Practical Analytic Model for Daylight”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 91–100. ISBN: 0-201-48560-5. DOI: 10.1145/311535.311545. URL: http://dx.doi.org/10.1145/311535.311545.

[Ray71] J. W. S. L. Rayleigh. “On the scattering of light by small particles”. In: *Philosophical Magazine* (1871), pp. 447–454. URL: http://journals.ametsoc.org/doi/pdf/10.1175/1520-0469(1974)031%3C1662:TIOOAA%3E2.0.CO%3B2.

[Ric] C. Riccio. *GLM*. URL: http://glm.g-truc.net/.

[Ril+04] K. Riley, D. S. Ebert, M. Kraus, J. Tessendorf, and C. Hansen. “Efficient Rendering of Atmospheric Phenomena”. In: *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*. EGSR’04. Norrkoping, Sweden: Eurographics Association, 2004, pp. 375–386. ISBN: 3-905673-12-6. DOI: 10.2312/EGWR/EGSR04/375-386. URL: http://dx.doi.org/10.2312/EGWR/EGSR04/375-386.

[Sal+10] M. Salvi, K. Vidimče, A. Lauritzen, and A. Lefohn. “Adaptive Volumetric Shadow Maps”. In: *Proceedings of the 21st Eurographics Conference on Rendering*. EGSR’10. Saarbr&#252;cken, Germany: Eurographics Association, 2010, pp. 1289–1296. DOI: 10.1111/j.1467-8659.2010.01724.x. URL: http://dx.doi.org/10.1111/j.1467-8659.2010.01724.x.

[Sch15] A. Schneider. “The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn”. In: *Advances in Real Time Rendering, Part I, ACM SIGGRAPH 2015 Courses*. SIGGRAPH ’15. Los Angeles, California: ACM, 2015. ISBN: 978-1-4503-3634-5. DOI: 10.1145/2776880.2787701. URL: http://doi.acm.org/10.1145/2776880.2787701.

[Sch16] A. Schneider. *Real-Time Volumetric Cloudscape*. Ed. by W. Engel. CRC Press, 2016, pp. 97–127.

[Ser13] A. Serdyuchenko. *O3 spectra*. 2013. URL: http://www.iup.physik.uni-bremen.de/gruppen/molspec/databases/referencespectra/o3spectra2011/index.html.

[VSc] VScauce. *The Moon Terminator Illusion*. URL: https://www.youtube.com/watch?v=Y2gTSjoEExc.

[Wen07] C. Wenzel. “Real time atmospheric effects in game revisited”. In: Game Developers Conference. 2007. URL: http://developer.download.nvidia.com/presentations/2007/D3DTutorial_Crytek.pdf.

[Wika] Wikipedia. *Angular diameter*. URL: https://en.wikipedia.org/wiki/Angular_diameter.

[Wikb] Wikipedia. *Black body radiation*. URL: https://en.wikipedia.org/wiki/Black-body_radiation.

[Wikc] Wikipedia. *Cloud fog bow*. URL: https://en.wikipedia.org/wiki/Fog_bow.

[Wikd] Wikipedia. *Cloud glory halo*. URL: https://en.wikipedia.org/wiki/Glory_(optical_phenomenon).

[Wike] Wikipedia. *Cloud Types*. URL: https://en.wikipedia.org/wiki/Cloud.

[Wikf] Wikipedia. *Day light*. URL: https://en.wikipedia.org/wiki/Daylight.

[Wikg] Wikipedia. *Light Scattering by Particles*. URL: https://en.wikipedia.org/wiki/Light_scattering_by_particles.

[Wikh] Wikipedia. *Limb Darkening*. URL: https://en.wikipedia.org/wiki/Limb_darkening.

[Wiki] Wikipedia. *Mole unit*. URL: https://en.wikipedia.org/wiki/Mole_(unit).

[Wikj] Wikipedia. *Moon*. URL: https://en.wikipedia.org/wiki/Moon.

[Wikk] Wikipedia. *Number density*. URL: https://en.wikipedia.org/wiki/Number_density.

[Wikl] Wikipedia. *Planck s law*. URL: goo.gl/PqSGLi.

[Wikm] Wikipedia. *Simpson s rule*. URL: https://en.wikipedia.org/wiki/Simpsons_rule.

[Wikn] Wikipedia. *Trapezoidal rule*. URL: https://en.wikipedia.org/wiki/Trapezoidal_rule.

[WKL13] M. Wrenninge, C. Kulla, and V. Lundqvist. "Oz: The Great and Volumetric". In: *ACM SIGGRAPH 2013 Talks*. SIGGRAPH '13. Anaheim, California: ACM, 2013, 46:1–46:1. ISBN: 978-1-4503-2344-4. DOI: 10.1145/2504459.2504518. URL: http://doi.acm.org/10.1145/2504459.2504518.

[Wre11] M. Wrenninge. "Production Volume Rendering". In: *ACM SIGGRAPH 2011 Courses*. SIGGRAPH '11. ACM, 2011.

[WSS13] C. Wyman, P.-P. Sloan, and P. Shirley. "Simple Analytic Approximations to the CIE XYZ Color Matching Functions". In: *Journal of Computer Graphics Techniques (JCGT)* (2013). URL: http://jcgt.org/published/0002/02/01/paper.pdf.

[Yus13a] E. Yusov. "Outdoor Light Scattering". In: Game Developers Conference. 2013. URL: https://software.intel.com/en-us/blogs/2013/06/26/outdoor-light-scattering-sample.

[Yus13b] E. Yusov. *Outdoor Light Scattering Sample*. 2013. URL: https://software.intel.com/en-us/blogs/2013/06/26/outdoor-light-scattering-sample.

[Yus13c] E. Yusov. *Outdoor Light Scattering Sample Update*. 2013. URL: https://software.intel.com/sites/default/files/blog/473591/outdoor-light-scattering-update_1.pdf.

[Yus14] E. Yusov. "High-Performance Rendering of Realistic Cumulus Clouds Using Pre-computed Lighting". In: *Eurographics/ ACM SIGGRAPH Symposium on High Performance Graphics*. Ed. by I. Wald and J. Ragan-Kelley. The Eurographics Association, 2014. ISBN: 978-3-905674-60-6. DOI: 10.2312/hpg.20141101.

A Sky look-up table parametrization

In the appendix, we give functions we used to convert input parameters to LUT coordinates, and vice-versa. These listings represent a version that can be used as reference path (not optimized). Listing 3 gives information about the contextual data, parametrization and LUT coordinate structure we use in Frostbite. Listing 4 gives the function transforming parameters into the look-up table coordinates. Listing 5 gives the inverse functions, transforming look-up table coordinates into parameters.

```

struct SkyLutContext
{
    // Earth properties
    float atmosphereRadius;
    float earthRadius;

    // Look up table resolution
    float resolutionHeight;
    float resolutionView;
    float resolutionSun;
};

struct SkyLutCoord
{
    // Transmitance is only 2d texture
    float2 transCoord;

    // Scattering is a 3d texture.
    float3 scattCoord;
};

struct SkyLutParameter
{
    float height;           // camera height between ground level 0 and atsmosphere height
    float cosViewAngle;     // cos of view zenith angle
    float cosSunAngle;      // cos of sun  zenith angle
};

//
// [1] Yusov, Outdoor Light Scattering Sample Update
// [2] Elek, Rendering Parametrizable Planetary Atmospheres with Multiple Scattering
// [3] Bruneton, Precomputed Atmospheric Scattering
//
```

Listing 3: Unoptimized reference code for physically based sky look-up table mapping.

```

SkyLutCoord convertSkyParamsToLutCoords(in SkyLutContext context, in SkyLutParameter params)
{
    SkyLutCoord output;

    // Normalised coordinates based on camera height between ground level 0 and atsmosphere
        height
    // Used in [1][2][3], Eq. 4 in [1]
    float normalisedheight = clamp(params.height, 0.f, (context.atmosphereRadius - context.
        earthRadius));
    normalisedheight = saturate(normalisedheight / (context.atmosphereRadius - context.
        earthRadius));
    normalisedheight = pow(normalisedheight, 0.5f);

    // Normalised coordinates based on angle between zenith direction and view direction
    // Eq 6 in [1]. Used for view direction but here used for sun direction.
    float normalisedViewZenithTrans = 0.5*(atan(max(params.cosViewAngle, -0.45f)*tan(1.26f
        *0.75f)) / 0.75f + (1.0 - 0.26f));

    // Normalised coordinates based on andgle between zenith direction and view direction
    // Eq. 7 in [1]
    float height = max(params.height, 0.f);
    float cosHorizon = -sqrt(height*(2.f*context.earthRadius + height)) / (context.
        earthRadius + height);
    float normalisedViewZenithScatt;
    if (params.cosViewAngle > cosHorizon)
    {
        float cosViewAngle = max(params.cosViewAngle, cosHorizon + 0.0001f);
        normalisedViewZenithScatt = saturate((cosViewAngle - cosHorizon) / (1.f - cosHorizon
            ));
        normalisedViewZenithScatt = pow(normalisedViewZenithScatt, 0.2f);
        normalisedViewZenithScatt = 0.5f + 0.5f / float(context.resolutionView) +
            normalisedViewZenithScatt * (float(context.resolutionView) / 2.f - 1.f) / float(
            context.resolutionView);
    }
    else
    {
        float cosViewAngle = min(params.cosViewAngle, cosHorizon - 0.0001f);
        normalisedViewZenithScatt = saturate((cosHorizon - cosViewAngle) / (cosHorizon -
            (-1.f)));
        normalisedViewZenithScatt = pow(normalisedViewZenithScatt, 0.2f);
        normalisedViewZenithScatt = 0.5f / float(context.resolutionView) +
            normalisedViewZenithScatt * (float(context.resolutionView) / 2.f - 1.f) / float(
            context.resolutionView);
    }

    // Sun an texcoord
    // Eq 6 in paper [1]
    float normalisedSunZenith = 0.5*(atan(max(params.cosSunAngle, -0.45f)*tan(1.26f*0.75f))
        / 0.75f + (1.0 - 0.26f));

    // Map normalised coordinates into in-between pixel range according to resolution
    output.transCoord = float2(normalisedheight, normalisedViewZenithTrans);
    output.transCoord = ((output.transCoord * (float2(context.resolutionHeight, context.
        resolutionView) - 1) + 0.5) / float2(context.resolutionHeight, context.
        resolutionView));

    // Map normalised coordinates into in-between pixel range according to resolution
    output.scattCoord = float3(normalisedheight, normalisedViewZenithScatt,
        normalisedSunZenith);
    output.scattCoord.xz = ((output.scattCoord * (float3(context.resolutionHeight, context.
        resolutionView, context.resolutionSun) - 1) + 0.5)
        / float3(context.resolutionHeight, context.resolutionView, context.resolutionSun)).
            xz;

    return output;
}
```

Listing 4: Unoptimized reference code for physically based sky look-up table mapping.

```

// Transmittance look up is only a float3 so SkyLutParameter.cosSunAngle will be 0.
SkyLutParameter convertTransmittanceLutCoordsToSkyParams(in SkyLutContext context, in float2
     coords)
{
    // Invert valid pixel range mapping
    float2 texCoords = saturate((coords * float2(context.resolutionHeight, context.
        resolutionView) - 0.5) / (float2(context.resolutionHeight, context.resolutionView) -
         1));

    // Invert height mapping
    texCoords.x *= texCoords.x; // squared
    float height = texCoords.x * (context.atmosphereRadius - context.earthRadius);

    // Invert view mapping
    float cosViewAngle = tan((2.0*texCoords.y - 1.0 + 0.26)*0.75) / tan(1.26*0.75);
    cosViewAngle = clamp(cosViewAngle, -1, 1);

    // Output
    SkyLutParameter output = (SkyLutParameter)0;
    output.height       = height;
    output.cosViewAngle = cosViewAngle;
    return output;
}

SkyLutParameter convertScatteringLutCoordsToSkyParams(in SkyLutContext context, in float3
    coords)
{
    // Invert valid pixel range mapping
    float3 texCoords = saturate((coords * float3(context.resolutionHeight, context.
        resolutionView, context.resolutionSun) - 0.5) / (float3(context.resolutionHeight,
        context.resolutionView, context.resolutionSun) - 1));

    // Invert height mapping
    texCoords.x *= texCoords.x; // squared
    float height = texCoords.x * (context.atmosphereRadius - context.earthRadius);

    // Invert view mapping
    height = max(height, 0.0);
    float cosHorizon = -sqrt(height * (height + 2.0 * context.earthRadius)) / (context.
        earthRadius + height);
    float cosViewAngle;
    if (texCoords.y > 0.5)
    {
        texCoords.y = saturate((texCoords.y - (0.5 + 0.5 / context.resolutionView)) *
            context.resolutionView / (context.resolutionView / 2.0 - 1.0);
        texCoords.y = pow(texCoords.y, 5.0);
        cosViewAngle = max((cosHorizon + texCoords.y * (1 - cosHorizon)), cosHorizon + 1e-4)
            ;
    }
    else
    {
        texCoords.y = saturate((texCoords.y - 0.5 / context.resolutionView)) * context.
            resolutionView / (context.resolutionView / 2.0 - 1.0);
        texCoords.y = pow(texCoords.y, 5);
        cosViewAngle = min((cosHorizon - texCoords.y*(cosHorizon - (-1))), cosHorizon-1e-4);
    }
    cosViewAngle = clamp(cosViewAngle, -1.0, 1.0);

    //Parameterization for sun angle
    float cosSunAngle = tan((2.0 * texCoords.z - 1. + 0.26) * 0.75) / tan(1.26 * 0.75);
    cosSunAngle = clamp(cosSunAngle, -1.0, 1.0);

    SkyLutParameter output = (SkyLutParameter)0;
    output.height       = height;
    output.cosViewAngle = cosViewAngle;
    output.cosSunAngle  = cosSunAngle;
    return output;
}
```

Listing 5: Unoptimized reference code for physically based sky look-up table mapping.

B Sun limb darkening astro-physical models

We present the implementation of two astro-physical models representing the sun limb darkening model. In these models:

- **centerToEdge**: normalised distance from center to edge of the sun.
- **finalLuminance**: the final sun luminance contribution to the pixel.

```

// Model from http://www.physics.hmc.edu/faculty/esin/a101/limbdarkening.pdf
float3 u = float3(1.0, 1.0, 1.0);        // some models have u!=1
float3 a = float3(0.397, 0.503, 0.652); // coefficient for RGB wavelength (680,550,440)

centerToEdge = 1.0 - centerToEdge;
float mu = sqrt(1.0 - centerToEdge*centerToEdge);

float3 factor = 1.0 - u * (1.0 - pow(mu, a));
finalLuminance *= factor;
```

Listing 6: Sun limb darkening model according to [Nec96].

```

// Model using P5 polynomial from http://articles.adsabs.harvard.edu/cgi-bin/nph-
    iarticle_query?1994SoPh..153...91N&defaultprint=YES&filetype=.pdf

centerToEdge = 1.0 - centerToEdge;
float mu = sqrt(1.0 - centerToEdge*centerToEdge);

// coefficient for RGB wavelength (680,550,440)
float3 a0 = float3( 0.34685, 0.26073, 0.15248);
float3 a1 = float3( 1.37539, 1.27428, 1.38517);
float3 a2 = float3(-2.04425,-1.30352,-1.49615);
float3 a3 = float3( 2.70493, 1.47085, 1.99886);
float3 a4 = float3(-1.94290,-0.96618,-1.48155);
float3 a5 = float3( 0.55999, 0.26384, 0.44119);

float mu2 = mu*mu;
float mu3 = mu2*mu;
float mu4 = mu2*mu2;
float mu5 = mu4*mu;

float3 factor = a0 + a1*mu + a2*mu2 + a3*mu3 + a4*mu4 + a5*mu5;
finalLuminance *= factor;
```

Listing 7: Sun limb darkening model according to [HM98].

C Energy-conserving analytical scattering integration

We give more details about the energy-conserving scattered light integration equation presented and discussed in Section 5.6.3. The function basically gives the amount of light scattered out of a slab of homogeneous participating media of depth $d = b - a$, assuming a uniform incoming light to scatter S and extinction σ_t.

$$\begin{aligned} \int_{x=0}^{d} \exp^{-\sigma_t x} \times S dx &= \frac{S - S\exp^{-\sigma_t d}}{\sigma_t} \\ \int_{x=a}^{b} \exp^{-\sigma_t (x-a)} \times S dx &= \frac{S - S\exp^{-\sigma_t (b-a)}}{\sigma_t} \end{aligned} \tag{23}$$

Here is the mathematical proof of this result (where line 3 is obtained using the fact that $\exp(u)' = u' \exp(u)$):

$$\begin{aligned} \int_{x=0}^{d} \exp^{-\sigma_t x} \times S dx &= S \times \int_{x=0}^{d} \exp^{-\sigma_t x} dx \\ &= S \times [\exp^{-\sigma_t x}]_0^x \\ \text{Using} \exp(u)' = u' \exp(u), & \\ &= S \times (\frac{\exp^{-\sigma_t x}}{-\sigma_t} - \frac{\exp^{-\sigma_t \times 0}}{-\sigma_t}) \\ &= S \times \frac{\exp^{-\sigma_t x} - 1}{-\sigma_t} \\ &= \frac{S - S\exp^{-\sigma_t d}}{\sigma_t} \end{aligned} \tag{24}$$

Section 5.6.3 present improvements resulting from using this energy formulation of scattering for cloud rendering. Figure 53 present how it is useful for the more general participating media rendering use case presented in [Hil15].

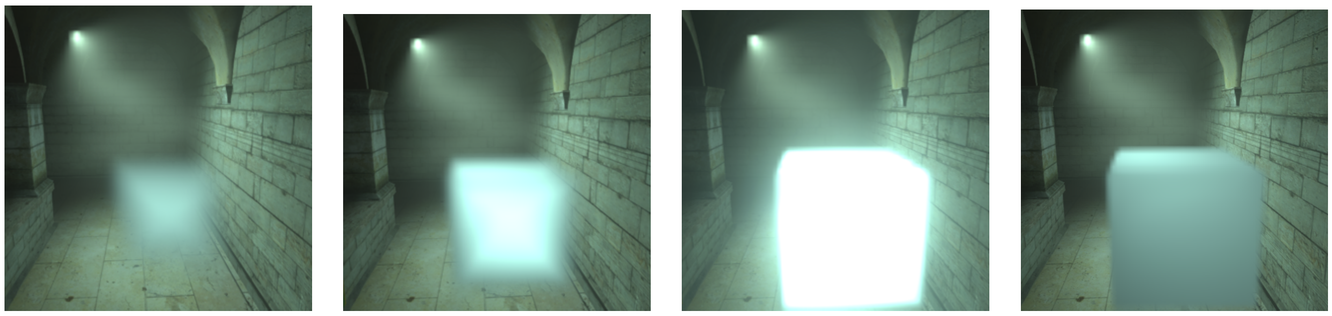


Figure 53: When increasing σ_s, a participating media material should converge towards looking like a solid material. However, integrating scattered light without equation 23 using an iterative approach can result in too much energy send back to the camera as shown in Section 5.6 (from left to right: $\sigma_s = 5$, $\sigma_s = 50$ and $\sigma_s = 5000$). Using energy-conserving scattering equation 23 the thick participating media cube on the right having $\sigma_s = 5000$ is looking more like perfect diffuse surface which is expected when using a uniform phase function.

D Tile-able volume noise library

We have presented important noise type required for the rendering of volumetric cloud in Section 5.4. We provide the source code to such functions:

1. https://github.com/sebh/TileableVolumeNoise
2. Multiple octaves of volume Worley noise (Marc-Andre Loyer)
3. Multiple octaves of volume Perlin noise using GLM [Ric]
4. Perlin-Worley noise as described in [Sch15]