Simulating the Colors of the Sky

Contents

Keywords: simulating colors of the sky, atmospheric scattering, Rayleigh scattering, Mie scattering, sky color, simulation of the sky, scattering coefficients, absorption coefficients, extinction coefficients, phase function, optical depth, volume rendering, sunlight, single scattering, multiple scattering, atmosphere, daylight, sunlight.

In this chapter we will learn about atmospheric scattering. We recommend that you also read the lesson on volume rendering and subsurface scattering. They share with this topic almost the same concepts. In fact, atmospheric scattering can be seen as an extension of volume rendering.

Introduction

For centuries, the sky has surely been a subject of fascination to many artists who tried to depict its colors with as much accuracy as possible. Many generations of physicists and mathematicians before the 19th-20th century also probably got obsessed with trying to figure out what could cause the sky to appear orange at sunset and sunrise and blue during the day. Historically, the discovery of atmospheric scattering is attributed to **Rayleigh** (also known as John Strutt), a Nobel prize English physicist whose main body of work was produced in the last 19th century (Rayleigh succeeded to **Maxwell** at the Cambridge University who is known for his work on electromagnetism. A large portion of the computer graphics research relates to Maxwell's work. Rayleigh also studied black bodies with **Max Planck**). Atmospheric scattering also has an effect on an object's appearance. This effect is known as **aerial perspective** and was first observed, studied and reproduced by Leonardo Da Vinci in his paintings (in fact this technique can be found in paintings prior to Da Vinci's work). As the distance between an object and an observer increases, the colors of the object are replaced with the colors of the atmosphere (for the sky it is usually blue during the day and red-orange at sunrise and sunset). This is an important visual clue as we, humans, are used to evaluate the distance of objects in a landscape by comparing how blue they appear in relation to each other. In a (digital) painting, aerial perspective can greatly help adding depth to an image (the brain can more

easily process that an object is further away than another) as illustrated with the following reproduction of a Da Vinci painting (Madonna of the Rocks).

In more recent history, that is history of computer graphics, **Nishita** wrote a seminal paper in 1993 entitled "**Display of the Earth Taking into account Atmospheric Scattering**" in which he describes an algorithm to simulate the colors of the sky. Interestingly enough his paper and the following one he wrote in 1996 on the same topic ("**Display Method of the Sky Color Taking into Account Multiple Scattering**") was not entirely about atmospheric simulation but more about realistic rendering of the Earth (including rendering of ocean surfaces, clouds and continents) seen from outer space.

This reminds us that in these years, the development of computer graphics techniques was driven by manufacturing industries for accurate simulation of their products or to provide a 3D virtual training environment, rather than by the entertainment industries (films and games). The technique described by Nishita to simulate the sky hasn't changed much since his time. Most of the research made in this area focused on implementing his algorithms on the GPU without any noticeable

improvement to the simulation quality (focus was more on speed and real time simulation). Another sky model exists though which was described by **Preetham & al.** in a paper published at Siggraph in 1999 (**"A Practical Analytic Model for Daylight**"). As its title suggests, this is an analytical model which provides an accurate simulation of the sky colors however it is more restrictive than the technique proposed by Nishita (for instance the model only works for an observer located on the ground). It is worth mentioning a paper published by **Jensen & al.** at Siggraph in 2011 on the simulation of a night sky ("**A Physically-Based Nightsky Model**"). This chapter will propose an implementation of the Nishita model. In the following chapters, we will study the other algorithms.

Figure 1: top a real photograph of the Earth from outer space. Bottom, a simulation using Nishita's model. These images have been copied from the paper "Display of the Earth taking into account Atmospheric Scattering" written by Nishita in 1993 (c) Siggraph.

In the next paragraph, we will describe the various phenomenons which once put together are responsible for the colors of the sky. We will then show how Nishita's algorithm simulates these phenomenons. Once we have the model implemented in a C++ program, we will show that we can easily extend the technique to simulate aerial perspective and that by varying the parameters of the model, we can also create alien skies.

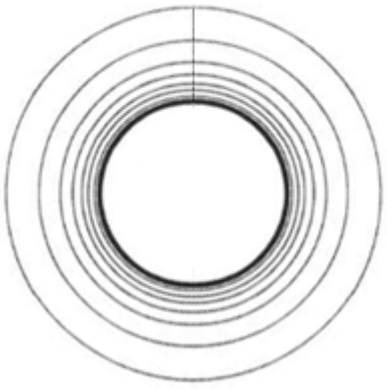
Atmospheric Model

An atmosphere is a layer of gas surrounding a planet. This atmosphere stays in place because of the planet's gravity. What defines this atmosphere is mainly its thickness and its composition (the different elements the atmosphere is made of). The earth atmosphere thickness is about 100 km (delimited by the Kármán line) and is made of oxygen, nitrogen, argon (a gas discovered by Rayleigh), carbon dioxide and water vapor. However in the simulation we will be using a thickness of 60km (we will only consider scattering in the first two layers of the Earth's atmosphere, the troposphere and the stratosphere). If you read the lesson on volume rendering (we invite you to do it now before you continue if you haven't yet) you will know that photons are scattered when they collide with particles. When light travels through a volume (the atmosphere) in a certain direction, photons are deflected in other directions when they collide with particles. This phenomenon is called scattering. How frequently light is scattered by particles both depends of the particles properties (mainly their size) and their density in the volume (how many particles/molecules are in one unit of volume). We have now the

size of the molecules that the Earth atmosphere is made of and we also know their concentration/density. We will use these scientific measurements for our atmospheric scattering simulation. One more important aspect of the atmosphere surrounding the planet, is that the density of these particles decrease with height. In other words, we find many more molecules per unit cube of air at the sea level than 2 km above the sea. This is an important fact as the amount of scattering depends on the molecules density as we just mentioned. As light travels in the atmosphere from the upper layer to the ground, we will need to sample the atmosphere density along the ray at regular intervals and compute the amount of scattering based on the atmosphere density at these sample positions (we perform a numerical integration which we explain in detail further down). Most papers (including Nishita's) make the assumption that the atmosphere density decreases exponentially with height. In other words it can be modelled with a simple equation of the form:

$$density(h) = density(0)e^{-\frac{h}{H}}$$

where density(0) is the air density at sea level, h is the current height (altitude) where we measure the atmosphere density and H is the thickness of the atmosphere if its density were uniform (in scientific papers H is called the **scale height** and in fact relates to the altitude by which the density of the atmosphere decreases by a factor of e. H depends on temperature). Some papers claim that this model is a good approximation, however some others claim it to be incorrect and prefer to model the sky density as a series of concentric layers which are each determined by their thickness and density (XX please quote paper here XX).



Modelling the sky densities as a series of layers (called spherical shell in Nishita's paper) also presents an advantage from a computational point of view. As we will perform a numerical integration along the path of the light towards the viewer, samples taken at position in the atmosphere where the density is high are more important than samples taken where the density is low. Performing a "constant step" integration along the ray would therefore result in a poor convergence. Nowadays computers are so fast that we can brute force this computation by taking regular steps but in Nishita's time optimising the algorithm was necessary. He therefore proposed a model where the atmosphere is represented as series of spherical shells, separated by small intervals at low altitude and long intervals at high altitude. Note however in the digram that was published in the paper that the thickness of the layers very much seem to have an exponential variation. This is not so surprising as the radius of each shell is precisely determined in his model by the density distribution of air molecules (and the formula he uses to compute this density is very similar to the one we have written above).

The atmosphere is made of small particles (air molecules) which are also mixed up at low altitude with bigger particles called **aerosols**. These particles can be anything from dust or sand raised by wind or can be there because of air pollution. They certainly have an impact on the atmosphere appearance especially as they don't scatter light the same way air molecules do. The scattering of light by air molecule is called **Rayleigh scattering** and the scattering of light by aerosols is called **Mie scattering**. In short, Rayleigh scattering (the scattering of light by air molecules) is responsible for the blue color of the sky (and it's red-orange colors at sunrise and sunset) while the Mie scattering (the scattering of light by aerosols) is usually responsible for the white-grey haze that you typically see above polluted cities (haze obscures the clarity of the sky, see figure 2).

Figure 2: aerosols present in the atmosphere are responsible for haze.

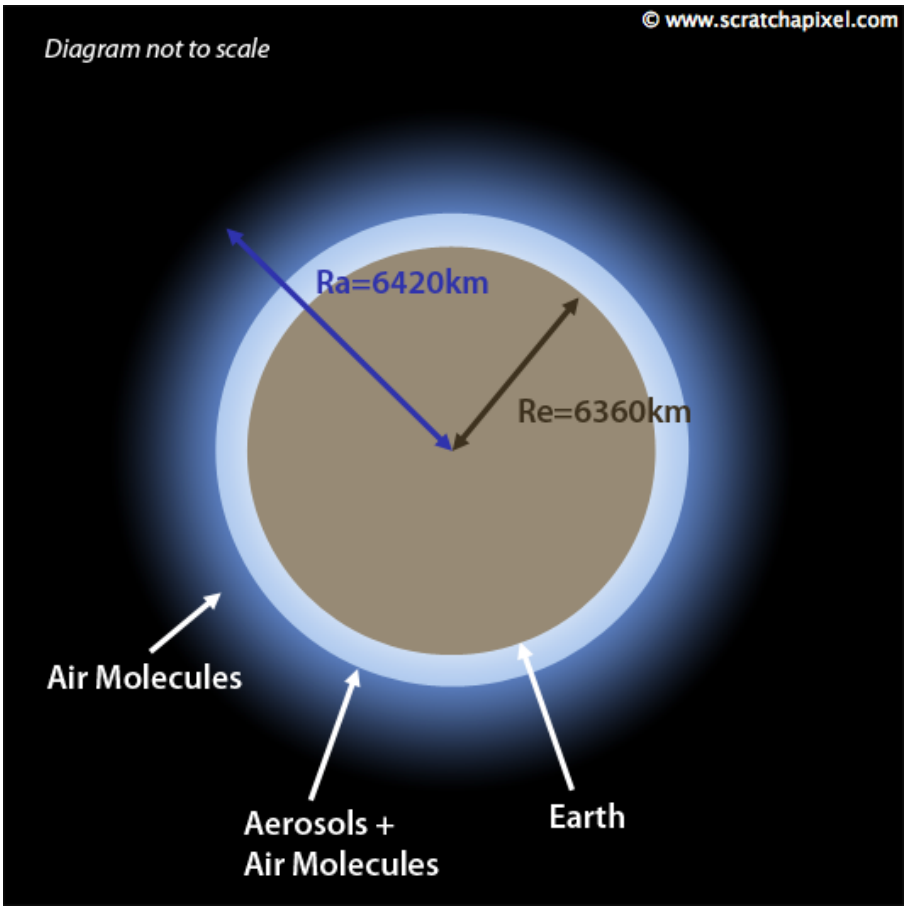


Figure 3: a simple graphical representation of the atmospheric model we will be using in this lesson. It is defined by the radius of the planet (Re) and the radius of the atmosphere (Ra). The atmosphere is made of aerosols (mainly present at low altitude) and air molecules. Density of particles decreases exponentially with altitude. This diagram is not to scale.

The model would be incomplete without mentioning that it will be required from the user to

specify the radius of the planet and the sky. These numbers will be used to compute the altitude of the sample positions taken along the view and light rays. For Earth we will be using Re = 6360 km (e for Earth) and Ra = 6420 km (a for atmosphere). All distances and parameters from our model which relate to distance should be expressed in the same unit system (either km, m, etc.).

> For atmospheric models, this can be tricky as the size of the planets can be large. For such dimensions, kilometer is usually a more suitable choice. However scattering coefficients are very small and they are more easily expressed in terms of meters or millimeters (even nanometers for light wavelengths). Choosing a good unit is also driven by floating precision limitations (that is if the numbers are too big or too small, the floating representation of these numbers by the computer can become inaccurate which would result in calculation errors). Input parameters to the model can be expressed in different units and internally converted by the program.

Finally, we will finish the description of our model by noting that the main source of illumination of our sky is the sun. The sun is so far away from the Earth that we can assume that all the light rays reaching the atmosphere are parallel to each other. We will show further down how this observation can simplify the implementation of our model.

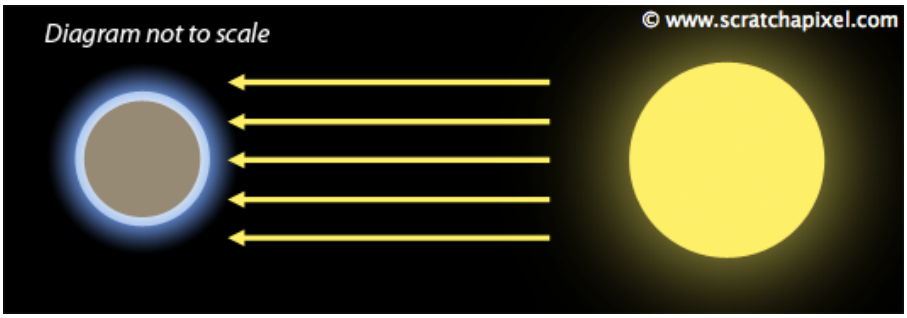


Figure 4: the sun is so far away from the Earth, that each light ray reaching the Earth's atmosphere can be considered as being parallel to each other.

Rayleigh Scattering

The scattering of light by air molecules was discovered by Rayleigh in the late 19th century. He showed in particular that this phenomenon has a strong wavelength dependency and more precisely that air molecules scatter blue light more than green and red light. This form of scattering and the equation he came up with to compute the scattering coefficients for volumes made of molecules such as the one we find in the atmosphere, only apply to particles which sizes are much smaller than the wavelengths making up visible light (the particle should be at least one tenth smaller than the scattered wavelength). Wavelengths for visible light vary from 380 to 780 nm, 440, 550 and 680 being considered as the peaks for blue, green and red light respectively. We will be using these values for the rest of this lesson. The Rayleigh scattering equation provides the scattering coefficients of a volume for which we know the molecular density. In the literature on atmospheric scattering the scattering coefficients are denoted by the greek letter β (beta).

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

The superscript *S* holds for scattering and the subscript R holds for Rayleigh (to differentiate these coefficients from the Mie scattering coefficients). In this equations h is the altitude, λ (lambda) is the wavelength, N is the molecular density at sea level, n is the index of refraction of air, and H_R is the scale height. In our simulation we will be using H_R = 8km (see scale height on the web for more accurate measurements if needed). As you can see with this equation, light with short wavelength (such as blue light) will give higher value for β; than light with long wavelength (red) which explains why the sky appears blue during the day. As light from the sun travels through the atmosphere, more blue light is scattered towards the observer than green and red light. Why does it appear red-orange at sunrise and sunset then ? In that particular configuration, the light from the sun has to travel a much longer portion of the atmosphere to reach the observer's position than it does when the sun is above your head (the **zenith** position). That distance being considerably longer, most of the blue light has been scattered away before it can reach the observer's location, while some of the red light which is not scattered as often as the blue light still remains. Hence the red-orange appearance of the sky at sunrise and sunset (which sometimes can either be purple or slightly green).

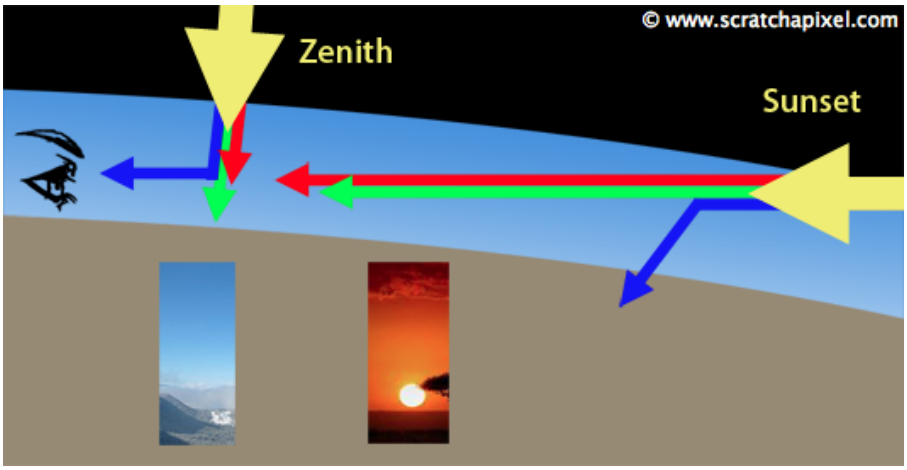


Figure 5: when the sun is at the zenith, sunlight travels a short distance before it reaches the eye. At sunset (or sunrise) sunlight travels a much bigger distance before it reaches the observer's eyes. In the first case blue light is scattered towards the eye and the sky appears blue. In the second case, most of the blue light has been scattered away before it can have a chance to reach the eye. Only red-green light reaches the observer's position which explain why the sky looks red-orange when the sun is at the horizon.

We could use some measurements for N, n, etc in order to compute a value for β; but we will use precomputed values which correspond to the scattering coefficients of the sky at sea level ($33.1\text{e}^{-6}\text{m}^{-1}$, $13.5\text{e}^{-6}\text{m}^{-1}$ and $5.8\text{e}^{-6}\text{m}^{-1}$ for wavelengths 440, 550 and 680 respectively. Note how the scattering coefficients for blue light is greater than the coefficients for green and red light). By just applying the exponential part of the equation (the right term) we can adjust these coefficients for any given altitude h.

> One of the main problems with Nishita's papers and the other papers we mention in this lesson, is that they either don't give the values that were used for N, n, etc. to produce the images that are shown in the papers. When measurements are used, authors of the papers don't usually cite their sources. Finding scientific data for molecular density at sea levels is not obvious and if you have any useful link or information on this topic that you could provide us with we would like to hear from you. The sea level scattering coefficients that we are using in this lesson can be found in two papers: "*Efficient Rendering of Atmospheric Phenomena*", by Riley et al. and "*Precomputed Atmospheric Scattering*" by Bruneton et al.

We won't go into too much details about the density of air, but interesting information can be found on the internet on this matter. In this lesson, all we really need are average scattering coefficients at sea level that work well for recreating the colors of the sky, but learning how to compute these coefficients will be of interest to any curious reader who wishes to have more control over the atmospheric model.

If you read the lesson on Volume Rendering, you will know that the physical models used to render participating media are the **scattering coefficients**, the **absorption coefficients** and the **phase function** which describes how much and in which directions light is scattered when it collides with particles. In this lesson so far, we have only given values (and explained how to compute) scattering coefficients for the Earth's atmosphere. For atmospheric scattering it is usually admitted that absorption is negligible. In other words we will assume that atmosphere doesn't absorb light. Remember from the lesson on Volume Rendering, that the **extinction coefficient** that we need in the physical model used to render a participating media, is the sum of the absorption and scattering coefficients, therefore (since the absorption coefficient is zero) we can write for the extinction coefficient:

$$\beta_R^e = \beta_R^s$$

The Raleigh phase function looks like this:

$$P_R(\mu) = \frac{3}{16\pi}(1+\mu^2)$$

where μ is the cosine of the angle between the light and the view directions (see figure 8).

Mie Scattering

Mie scattering is similar to the Rayleigh equation in a way but applies to particles which size is greater than the scattered wavelength. It is the case of aerosols which we find in the low altitudes of the Earth's atmosphere. If we were to apply the Rayleigh equation to aerosols we would not get a convincing image. The Mie equation to render the scattering coefficients looks like this:

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

where the subscript M holds for Mie. Note that there is a specific scale height value H_M for the Mie scattering which is usually set to 1.2 km. To the contrary of the Rayleigh scattering, we don't need an equation to compute the Mie scattering coefficients. We will use values from measurements made at sea level instead. For Mie scattering we will use $\beta_S = 210\mathrm{e}^{-5}\mathrm{m}^{-1}$. Aerosols too have their density decreasing exponentially with altitude and like for the Rayleigh scattering we will be simulating this effect by including an exponential term on the right inside of the equation (modulated by the scale height factor H_M). Mie extinction coefficient is about 1.1 times the value of its scattering coefficient. And the Mie phase function equation is:

$$P_M(\mu) = \frac{3}{8\pi} \frac{(1-g^2)(1+\mu^2)}{(2+g^2)(1+g^2-2g\mu)^{\frac{3}{2}}}$$

The Mie phase function includes a term g (the Rayleigh phase function doesn't) which controls the anisotropy of the medium. Aerosols exhibit a strong forward directivity. Some papers use g=0.76 (which we will also be using as our default value).

The Concept of Optical Depth

Before we look at a practical implementation of the algorithm in C++ we will put all the pieces of the puzzle together. FIrst, we should note that the sky is nothing else than a spherically shaped volume surrounding a solid sphere. Because it is nothing else than a volume, to render the sky, we can use the ray-marching algorithm which is the most common technique to render participating medium. We studied that algorithm in detail in the lesson on Volume Rendering. When we render a volume, the observer (or the camera) can either be inside the volume or outside.

When the camera is inside we need to find the point where the viewing ray exits the volume. When it is outside, we need to find the points where the viewing ray enters and exits the volume. Because the sky is a sphere, we will use a ray-sphere intersection routine to analytically compute these points. We have presented a couple of techniques to compute the intersection of a ray and a sphere in the basic section (Ray-Quadratic Shapes Intersection). We know if the camera is inside or outside the atmosphere by testing its altitude using the ground as a reference. If this altitude is greater than the atmosphere thickness then the camera is outside the atmosphere and the viewing ray might intersect the atmosphere in two places.

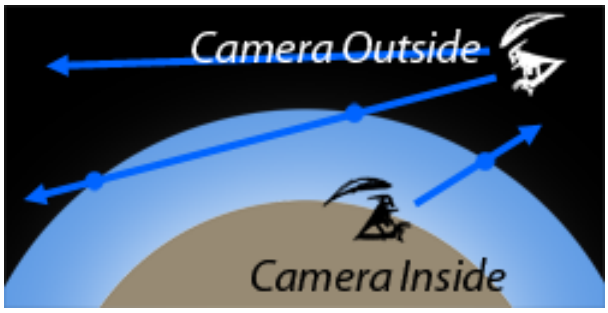


Figure 6: the camera can either be inside the atmosphere or outside. When it is inside, we are only interested in the intersection point between the camera ray and the atmosphere. When the camera is outside it can intersect the atmosphere in two points.

For now we will assume that the camera is on the ground (or one meter above the ground) but the algorithm will work for any arbitrary camera position (we will render an image of the sky from outer space at the end of this lesson). Lets assume that we have a viewing ray (corresponding to one pixel in the frame) and that we know where this ray intersects with the upper limit of the atmosphere. The problem we need to solve at this point, is to find out how much light is traveling along this ray in the direction of the viewer. As we can see in following figure, it is unlikely that our camera or observer will be directly looking at the sun (which is dangerous for your eyes).

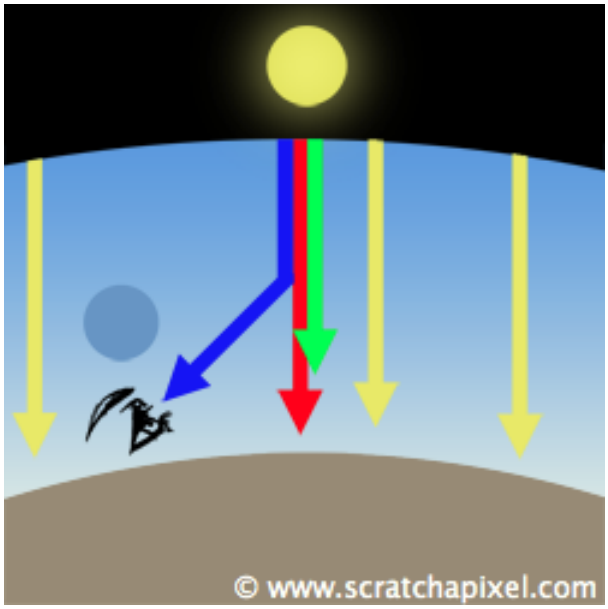


Figure 7: single scattering is responsible for the sky color. A viewer is rarely directly looking at the sun (which is dangerous). However when we are looking away from the sun the atmosphere has a color which is the result of blue light from the sunlight being scattered in the direction of the observer's eyes.

In all logic, if you were looking in the direction of the sky, you shouldn't see anything because you are looking at the outer space which is empty (space between celestial bodies). However, the fact is that you see a blue color. This is caused by light from the sun entering the atmosphere and being deflected by air molecules in the viewing direction. In other words, there's no direct light coming from the sun traveling along the viewing direction (unless the viewing direction is pointing directly at the sun) but as the light coming from the sun is scattered by the atmosphere, some of that light ends up traveling in the direction of your eye. This phenomenon is called

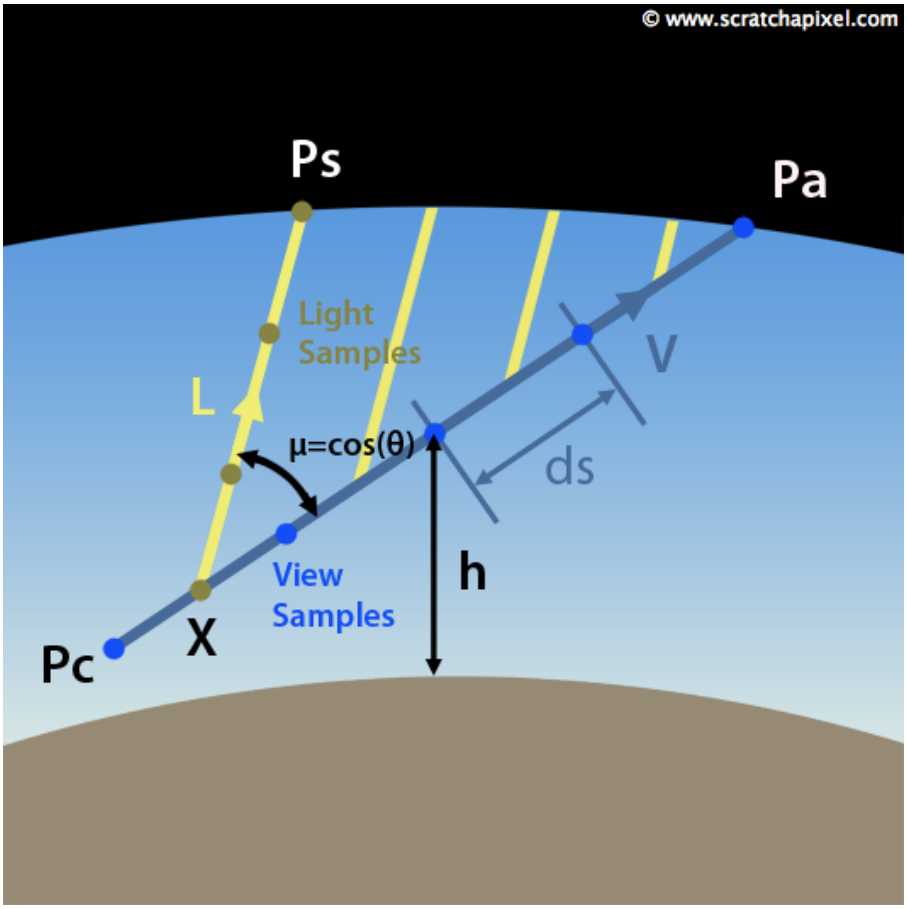


Figure 8: to compute the sky color we first trace a ray from Pc to Pa and then integrate the amount of light coming from the sun (with direction L) that is

reflected back along that ray (V).

single scattering (which is explained in the lesson on Volume Rendering).

What do we know so far? We know that to render the sky color for one pixel in the frame we will first cast a viewing ray (V) from the point Pc (the camera position) in the direction of interest. We will compute the point Pa where this ray intersects with the atmosphere. Finally, we need to compute the amount of light that is traveling along this ray due to single scattering. This value can be computed with the volume rendering equation which we studied in the lesson on Volume Rendering:

$$L(P_c, P_a) = \int_{P_c}^{P_a} T(P_c, X) L_{sun}(X) ds$$

Where T is the transmittance between point Pc and X (the sample position along the viewing direction) and L is the amount of light in the volume at X. All it says, is that the total amount of light reaching the observer at Pc is equal to all the sunlight being scattered along the viewing direction V. This quantity can be obtained by summing up the light at various positions (X) along V. This technique is called a numerical integration. The more samples we take along the ray the better the result but the longer it takes to compute. The transmittance term (T) in the equation accounts for the fact that light scattered along V in the direction of the viewer at each sample position (X), is also attenuated as it travels from X to Pc.

Recall from the lesson on volume rendering that light is attenuated as it travels from a point to another in the volume due to absorption and out-scattering. Lets call the amount of light we have at Pb, Lb and the amount of light arriving at Pa from Pb, La. In the presence of a participating medium, La (light received from Pb) will be lower than Lb. Transmittance represents the ratio La over Lb and T is therefore in the range zero to one. In equation form, transmittance looks like this:

$$T(P_a, P_b) = \frac{L_a}{L_b} = exp(-\sum_{P_a}^{P_b} \beta_e(h) ds)$$

In plain english this equation means that we need to measure the extinction coefficients β_e of the atmosphere at various sample positions along the path ray Pa-Pb, sum these values up, multiply them by the length segment ds (that is the distance Pa-Pb divided by the number of samples used), take the negative of this value and feed it to an exponential function.

Going back to what we said about Rayleigh scattering you will

remember that β_s (from which we will compute β_e) can be computed using the scattering coefficients at sea level modulated by the exponential of the altitude h over the scale height (H).

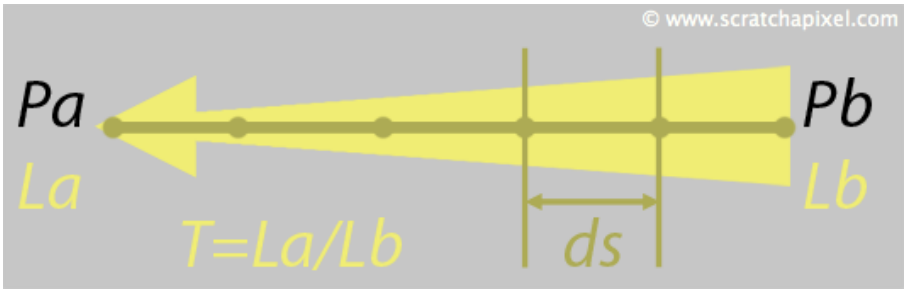


Figure 9: as light (Lb) travels from Pb to Pa, it gets attenuated because of out-scattering and absorption. The result is La. Transmittance is the amount of light received at Pa from Pb, after it was attenuated while traveling through the atmosphere (that is T=La/Lb).

$$\beta_s(h) = \beta_s(0)exp(-\frac{h}{H})$$

For Rayleigh scattering, absorption can be ignored and we can write:

$$\beta_e(h) = \beta_s(h) + \beta_a(h) = \beta_s(h) + 0 = \beta_s(h)$$

You can move $\beta_e(0)$ out of the integral (because it is a constant) and you can re-write the transmittance equation as:

$$T(P_a, P_b) = exp(-\beta_e(0) \sum_{P_a}^{P_b} exp(-\frac{h}{H})ds)$$

The sum inside the exponential function can be seen as the average density value between Pa and Pb and the equation itself is usually referred to in the literature as the **optical depth** of the atmosphere between points Pa-Pb.

Adding the Sunlight

Finally we will use the rendering equation that we have presented in the previous paragraph to compute the sky color. Lets write it again:

$$L(P_c, P_a) = \int_{P_c}^{P_a} T(P_c, X)L_{sun}(X)ds$$

We have explained how to render the transmittance. Lets now have a look at the term Lsun(X) and explain how to evaluate it. Lsun(X) corresponds to the amount of sunlight scattered at the sample position X along the viewing direction. To evaluate it, first we need to compute the amount of light arriving at X. Taking the sunlight intensity directly is not enough. Indeed, if light enters the atmosphere at Ps it will also be attenuated as it travels to X. We therefore need to compute this attenuation using a similar equation to the one we have been using to compute the attenuation of light traveling along the viewing ray from Pa to Pc (equation 1):

$$L_{sun}(X) = Sun\ Intensity * T(X, P_s)$$

Equation 1

Technically for each sample position X along the viewing ray we will need to cast a ray in the sunlight direction (L) and find where this light ray intersects with the atmosphere (Ps). We will then use the same numerical integration technique we used for the viewing ray to evaluate the transmittance (or optical depth) term in equation 1. The light ray will be cut into segments and the density of the atmosphere at the centre of each light segment will be evaluated.

One of the last elements in building an algorithm to compute the sky color, is to account for the amount of light that is scattered in the viewing direction based on the light direction itself and the view direction. This is the role of the phase function. Rayleigh and Mie scattering have their own phase function which we gave further up. If you need a refresher on what the phase function is, read the lesson on Volume Rendering. In short, the phase function is a function that describes how much light coming from direction L is scattered in direction V. The Mie phase function contains an extra term g, called the **mean cosine** (among many other possible names) which defines if the light is mainly scattered along the forward direction (L) or the backward direction ($-L$). For forward scattering g is in the range [0:1] and for backward scattering g is in the range [-1:0]. When g equals zero, light is equally scattered in all directions and we say that the scattering is isotropic. For Mie scattering we will set g to 0.76.

$$L_{sun}(X) = Sun\ Intensity * T(X, P_s) * P(V, L)$$

Finally we need to account for the fact that mainly blue light for instance for Rayleigh scattering is scattered along the view direction. To reflect this we will multiply the result of the previous equation by the scattering coefficients (β_s) which gives the full equation for the light part of the equation. However recall that β_s changes with the altitude (its value is a function of height) where h is the height of X in relation to the ground (sea level more precisely):

$$L_{sun}(X) = Sun\ Intensity * T(X, P_s) * P(V, L) * \beta_S(h)$$
$$L(X) = \int_{4\pi} Light\ Intensity * T(X, P_s) * P(V, L) * \beta_S(h)$$

For the Earth Atmosphere the sun is the only light source in the sky however, to write a generic form of the previous equation we should account for the fact that light might come from many directions. In scientific paper you will usually see this equation being written as an integral over 4π; which describes a sphere of incoming directions. This more generic equation would also account for sunlight reflected by the ground (multiple scattering) which we will ignore in this chapter.

Computing the Sky Color

Putting all the elements together we have (equation 2):

$$Sky\,Color(P_c, P_a) = \int_{P_c}^{P_a} T(P_c, X) L_{sun}(X) ds$$

Equation 2

with (equation 3):

$$L_{sun}(X) = Sun\ Intensity * T(X, P_s) * P(V, L) * \beta_S(h)$$

Equation 3

If we replace 3 in 2 we get:

$$Sky\,Color(P_c, P_a) = \\ \int_{P_c}^{P_a} T(P_c, X) * Sun\ Intensity * P(V, L) * T(X, P_s) * \beta_s(h) ds$$

The result of the phase function is a constant as well as the sunlight intensity. We can therefore move these two terms out of the integral (equation 4):

$$Sky\,Color(P_c, P_a) = \\ Sun\ Intensity * P(V, L) \int_{P_c}^{P_a} T(P_c, X) * T(X, P_s) * \beta_s(h) ds$$

Equation 4

This is the final equation for rendering the color of the sky for a particular view and light direction. As such it's not so complicated. It's more computationally demanding since we compute an integral and do many calls to the exponential function (through the transmittance term). Plus, you need to remember that they sky color is actually the result of Rayleigh and Mie scattering. Therefore we need to compute this equation twice for each type of scattering:

$$Sky\,Color(P_c, P_a) = \\ Sky\,Color_{Rayleigh}(P_c, P_a) + Sky\,Color_{Mie}(P_c, P_a)$$

Remember from calculus that the multiplication of two exponential functions is equal to one exponential function which argument is the sum of the arguments from the first two functions:

$$e^a e^b = e^{a+b}$$

We can use this property at our advantage (we compute one exponential instead of two) and re-write the multiplication of the two transmittance terms in equation 4 with:

$$T(P_c, X) = e^{-\beta_{e0}} \\ T(X, P_s) = e^{-\beta_{e1}} \\ T(P_c, X) * T(X, P_s) = e^{-\beta_{e0}} * e^{-\beta_{e1}} = e^{-(\beta_{e0} + \beta_{e1})}$$

Equation 5

Lets now have a look at the implementation of the atmospheric model in a C++ program.

Implementation (C++)

We have now all the elements needed to implement a version of Nishita's algorithm in a C++ program. As usual, the program will be written so that the algorithm can easily be understood. Some optimisations will be suggested at the end of this paragraph that could make the program run faster. However real time is not the goal here and images of the sky can still be created with this program in a matter of seconds.

First we will create an Atmosphere class that we will use to specify all the parameters of our system: the radius of the planet and of the atmosphere (Re, Ra), the Rayleigh and Mie scattering coefficients at sea level, the Rayleigh and Mie scale height (Hr and Hm), the sun direction, the sun intensity and the mean cosine. All distances are expressed in meters (as well as the scattering coefficients).

```
001 class Atmosphere
002 {
003 public:
004     Atmosphere(
005         Vec3f sd = Vec3f(0, 1, 0),
006         float er = 6360e3, float ar = 6420e3,
007         float hr = 7994, float hm = 1200) :
008         sunDirection(sd),
009         earthRadius(er),
010         atmosphereRadius(ar),
011         Hr(hr),
012         Hm(hm)
013     {}
014
015     Vec3f computeIncidentLight(const Vec3f& orig, const Vec3f& dir, float tmin, float t
016
```

```
    Vec3f sunDirection;     // The sun direction (normalized)
    float earthRadius;      // In the paper this is usually Rg or Re (radius ground, ea
    float atmosphereRadius; // In the paper this is usually R or Ra (radius atmosphere)
    float Hr;               // Thickness of the atmosphere if density was uniform (Hr)
    float Hm;               // Same as above but for Mie scattering (Hm)

    static const Vec3f betaR;
    static const Vec3f betaM;
};

const Vec3f Atmosphere::betaR(3.8e-6f, 13.5e-6f, 33.1e-6f);
const Vec3f Atmosphere::betaM(21e-6f);
```

We will render the sky as if it was seen by a fisheye lens. The camera looks straight up and captures a 360 degrees view of the sky. To create an animation of the sky rendered for different position of the sun, we will render a series of frames. At the first frame, the sun is at the zenith (centre frame). At the last frame, the sun is slightly under the horizon.

```
void renderSkydome(const Vec3f& sunDir, const char *filename)
{
    Atmosphere atmosphere(sunDir);
    auto t0 = std::chrono::high_resolution_clock::now();

    const unsigned width = 512, height = 512;
    Vec3f *image = new Vec3f[width * height], *p = image;
    memset(image, 0x0, sizeof(Vec3f) * width * height);
    for (unsigned j = 0; j < height; ++j) {
        float y = 2.f * (j + 0.5f) / float(height - 1) - 1.f;
        for (unsigned i = 0; i < width; ++i, ++p) {
            float x = 2.f * (i + 0.5f) / float(width - 1) - 1.f;
            float z2 = x * x + y * y;
            if (z2 <= 1) {
                float phi = std::atan2(y, x);
                float theta = std::acos(1 - z2);
                Vec3f dir(sin(theta) * cos(phi), cos(theta), sin(theta) * sin(phi));
                // 1 meter above sea level
                *p = atmosphere.computeIncidentLight(Vec3f(0, atmosphere.earthRadius +
            }
        }
        fprintf(stderr, "\b\b\b\b\%3d%c", (int)(100 * j / (width - 1)), '%');
    }

    std::cout << "\b\b\b\b" << ((std::chrono::duration)(std::chrono::high_resolution_cl
    // Save result to a PPM image (keep these flags if you compile under Windows)
    std::ofstream ofs(filename, std::ios::out | std::ios::binary);
    ofs << "P6\n" << width << " " << height << "\n255\n";
    p = image;
    for (unsigned j = 0; j < height; ++j) {
```

```
030             for (unsigned i = 0; i < width; ++i, ++p) {
031 #if 1
032                 // Apply tone mapping function
033                 (*p)[0] = (*p)[0] < 1.413f ? pow((*p)[0] * 0.38317f, 1.0f / 2.2f) : 1.0f -
034                 (*p)[1] = (*p)[1] < 1.413f ? pow((*p)[1] * 0.38317f, 1.0f / 2.2f) : 1.0f -
035                 (*p)[2] = (*p)[2] < 1.413f ? pow((*p)[2] * 0.38317f, 1.0f / 2.2f) : 1.0f -
036 #endif
037                 ofs << (unsigned char)(std::min(1.f, (*p)[0]) * 255)
038                     << (unsigned char)(std::min(1.f, (*p)[1]) * 255)
039                     << (unsigned char)(std::min(1.f, (*p)[2]) * 255);
040             }
041         }
042         ofs.close();
043         delete[] image;
044 }
045
046 int main()
047 {
048 #if 1
049     // Render a sequence of images (sunrise to sunset)
050     unsigned nangles = 128;
051     for (unsigned i = 0; i < nangles; ++i) {
052         char filename[1024];
053         sprintf(filename, "./skydome.%04d.ppm", i);
054         float angle = i / float(nangles - 1) * M_PI * 0.6;
055         fprintf(stderr, "Rendering image %d, angle = %0.2f\n", i, angle * 180 / M_PI);
056         renderSkydome(Vec3f(0, cos(angle), -sin(angle)), filename);
057     }
058 #else
059     ...
060 #endif
061
062     return 0;
063 }
064
```

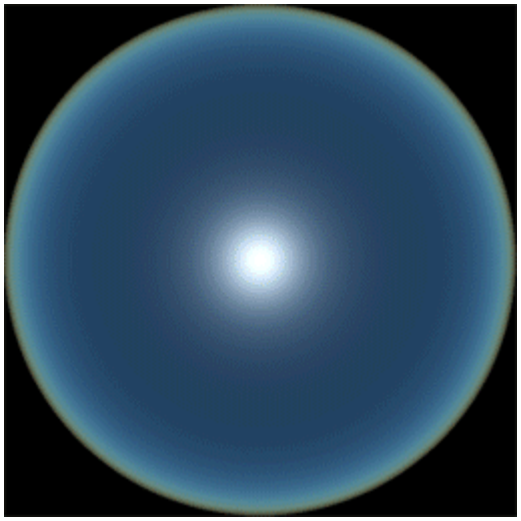
Finally here is the function used to compute equation 4 (that is the color of the sky for a particular camera ray). The first thing we do is finding the intersection point of the camera ray with the atmosphere (line 4). Then we compute the value of the Rayleigh and Mie phase function (using the sun and camera ray direction. Line 14 and 16). The first loop (line 17) creates samples along the camera ray. Note that the sample position (X in the equations) is the segment middle point (line 19). From there, we can compute the sample (X) height (line 19). We compute exp(-h/H) multiplied by ds for Rayleigh and Mie scattering (using Hr and Hm). These values are accumulated (line 23 and 24) to compute the optical depth at X. We will also use them later to scale the scattering coefficients $\beta_s(h)$ in equation 4 (line 42 and 43). Then we compute the amount of light coming from the sun at

X (line 31 to 38). We cast a ray in the direction of the sun (rays are parallel) and find the intersection with the atmosphere. This ray is cut into segments and we evaluate the density in the middle of the segment (line 35 and 36). Accumulating these values gives us the optical depth of the light ray. Note that we test if each light sample is above or below the ground. If it is below the ground, the light ray is in the shadow of the earth; we can then safely discard the contribution of this ray (line 34 and 39). Note that the Mie extinction coefficient is about 1.1 times the value of the Mie scattering coefficient (line 40). Finally using the trick from equation 5, we can compute the accumulated transmission of the light and the camera ray by accumulating their optical depth in one single exponential call (line 40 and 41). At the end of the function, we return the final color of the sky for this particular ray, which is the sum of the Rayleigh and Mie scattering transmittance, multiplied by their respective phase function and scattering coefficients. This sum is also multiplied by the sun intensity (line 48).

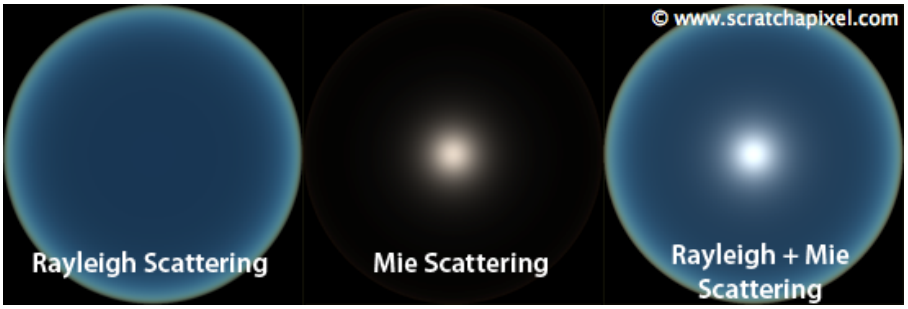
At this point in time, the sun intensity is just a magic number (we used the value 20). But in a future revision of this lesson we will learn how to use actual physical data).

```
001 Vec3f Atmosphere::computeIncidentLight(const Vec3f& orig, const Vec3f& dir, float tmin,
002 {
003     float t0, t1;
004     if (!raySphereIntersect(orig, dir, atmosphereRadius, t0, t1) || t1 < 0) return 0;
005     if (t0 > tmin && t0 > 0) tmin = t0;
006     if (t1 < tmax) tmax = t1;
007     uint32_t numSamples = 16;
008     uint32_t numSamplesLight = 8;
009     float segmentLength = (tmax - tmin) / numSamples;
010     float tCurrent = tmin;
011     Vec3f sumR(0), sumM(0); // mie and rayleigh contribution
        float opticalDepthR = 0, opticalDepthM = 0;
012     float mu = dot(dir, sunDirection); // mu in the paper which is the cosine of the ar
013     float phaseR = 3.f / (16.f * M_PI) * (1 + mu * mu);
014     float g = 0.76f;
015     float phaseM = 3.f / (8.f * M_PI) * ((1.f - g * g) * (1.f + mu * mu)) / ((2.f + g *
016     for (uint32_t i = 0; i < numSamples; ++i) {
017         Vec3f samplePosition = orig + (tCurrent + segmentLength * 0.5f) * dir;
018         float height = samplePosition.length() - earthRadius;
019         // compute optical depth for light
020         float hr = exp(-height / Hr) * segmentLength;
021         float hm = exp(-height / Hm) * segmentLength;
022         opticalDepthR += hr;
023         opticalDepthM += hm;
024         // light optical depth
025         float t0Light, t1Light;
026         raySphereIntersect(samplePosition, sunDirection, atmosphereRadius, t0Light, t1L
027         float segmentLengthLight = t1Light / numSamplesLight, tCurrentLight = 0;
028         float opticalDepthLightR = 0, opticalDepthLightM = 0;
029         uint32_t j;
```

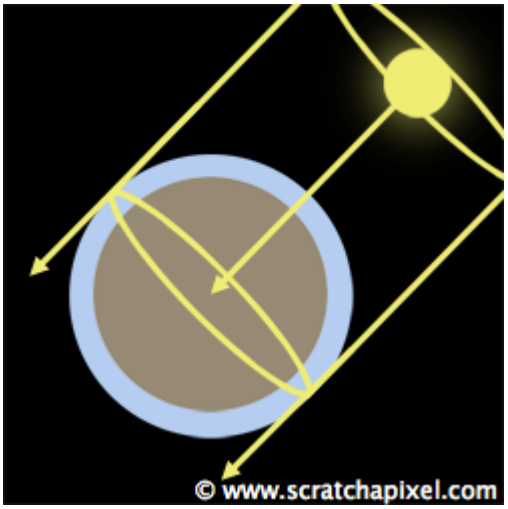
```
        uint32_t j;
        for (j = 0; j < numSamplesLight; ++j) {
            Vec3f samplePositionLight = samplePosition + (tCurrentLight + segmentLengt
            float heightLight = samplePositionLight.length() - earthRadius;
            if (heightLight < 0) break;
            opticalDepthLightR += exp(-heightLight / Hr) * segmentLengthLight;
            opticalDepthLightM += exp(-heightLight / Hm) * segmentLengthLight;
            tCurrentLight += segmentLengthLight;
        }
        if (j == numSamplesLight) {
            Vec3f tau = betaR * (opticalDepthR + opticalDepthLightR) + betaM * 1.1f * (
            Vec3f attenuation(exp(-tau.x), exp(-tau.y), exp(-tau.z));
            sumR += attenuation * hr;
            sumM += attenuation * hm;
        }
        tCurrent += segmentLength;
    }

    // We use a magic number here for the intensity of the sun (20). We will make it mo
    // scientific in a future revision of this lesson/code
    return (sumR * betaR * phaseR + sumM * betaM * phaseM) * 20;
}
```



Each frame only takes a few seconds to compute on a 2.5GHz processor (this time depends on the number of view and light samples you use). A few features can be added to this program. Some papers run a tone mapping on the resulting image before saving it out. This can help reducing the contrast between the brightness of the sky around the sun (very bright and white) and the rest of the sky. In addition, you can save the resulting image to a floating point image format (HDR, EXR) as values can easily be greater than one depending on your sun intensity (remapping and clipping the values is not a great choice). This function can easily be added to any existing renderer (see the paragraph on Aerial Perspective for more details about this). Note in the images, how the color of the sky turns red-orange when the sun is above or slightly below the horizon. You can also comment out the contribution of the Mie scattering to see the contribution of the Rayleigh scattering independently (and vice and versa). However by just looking at the resulting images and remembering what you have learned about these two scattering models, you can easily guess that Mie scattering is responsible for the main white halo around the sun and the brightening/desaturation of the sky at the horizon while Rayleigh scattering is responsible for the blue/red/orange colors of the sky.



An interesting project could consist of passing a date and time to the program, then compute the sun position based on this data, render a frame with this sun direction and compare the result to a photograph of the real sky taken at the same time on the same day. How closely would they match? The sun position is related to its declination and the solar hour angle (in other words the time of the day).



Optimisation: Nishita observed that the sky is symmetrical in relation to the axis defined by the sun position and the centre of the Earth. Based on this observation, the paper proposes a technique to bake the optical depth in a 2D table which you can access using μ (the cosine of the angle between the view and the light direction) and h (the height of the sample point). You will need to create a table for each scattering model (Rayleigh and Mie). These tables can be computed before hand saved into a file and reused each time the program is run. Using precomputed values for the optical depth rather than computing it on the fly (which requires a few expensive exponential calls) would speed the render quite a lot. If speed is important for your application, this optimisation is probably the first one you could implement (see Nishita's paper for the details).

Light Shafts

Atmospheric effects can sometimes create spectacular visual effects. Light shafts usually make these effects more dramatic. If light was traveling freely through the volume, the volume would appear uniformly lit. But if we were to place some objects in the scene, areas of the volumes shadowed by these objects would appear darker than areas which are not occluded. Light shafts are beams of light which corresponds to regions of the volume illuminated by a light source (the sun). But they are only visible because regions of this volume are shadowed by objects in the scene (in the case of the Earth's atmosphere, mainly mountains and clouds).

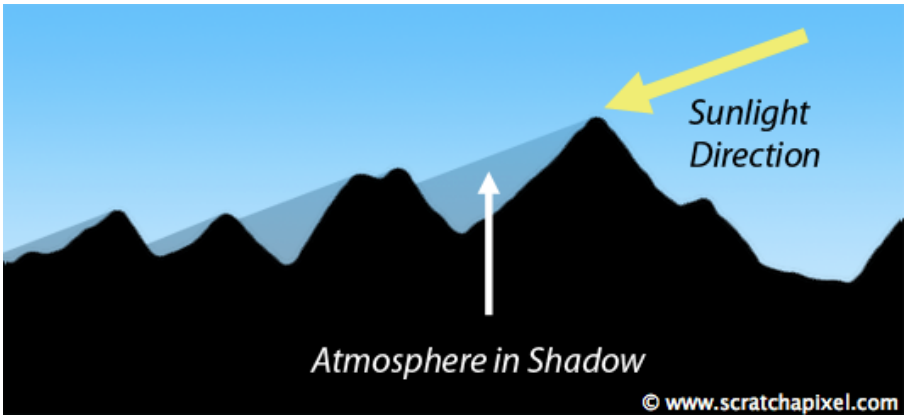


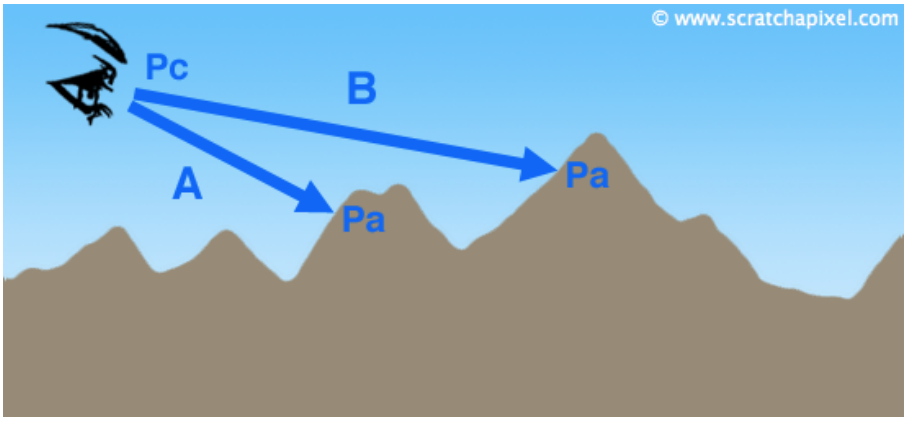
Figure 10: light shafts appear when some regions of the volume are shadowed by objects in the scene. In this example, some area of the atmosphere are shadowed by the mountains.

Simulating Aerial Perspective

Computing aerial perspective actually doesn't require much changes to our code. As you can see with the following figure, the color of the mountain is affected by the presence of the blue atmosphere. Because the distance from the eye to the ground is shorter for ray A than it is for B, the contribution of the atmosphere is less noticeable on the color of ray A than it is on the color of ray B.

First you should render the color of the geometry (the terrain) for the ray. Then you should render the transmittance (aka the opacity) and the color of the atmosphere (using the eye position and the intersection point of the ray with the geometry for Pc and Pa)

and composite the two colors together using the following alpha blending formula:




```
Ray Color = Object Color * (1 - Transmittance) + Atmosphere Color
```

In this lesson though, to keep the code simple, we won't be rendering the color of the ground which by default will be black. If you wish to understand in details how this compositing operation can be done, check the lesson on volume rendering. Here are a couple of results we got with two different positions for the sun. The camera is so far away that the Earth is visible in the bottom half of the frame. We first find if the primary ray intersects with the Earth. If it does we then compute the color of the atmosphere from the eye position to this intersection point.

These images can be computed with a very basic ray-tracer. If the camera ray hits the Earth, all we need to do is update the ray tmin/tmax variables (lines 36-39) before we call the function that computes the atmosphere color.

```
void renderSkydome(const Vec3f& sunDir, const char *filename)
{
    Atmosphere atmosphere(sunDir);
    ...
#if 1
    // Render fisheye
    ...
#else
    // Render from a normal camera
    const unsigned width = 640, height = 480;
    Vec3f *image = new Vec3f[width * height], *p = image;
    memset(image, 0x0, sizeof(Vec3f) * width * height);
    float aspectRatio = width / float(height);
    float fov = 65;
    float angle = std::tan(fov * M_PI / 180 * 0.5f);
    unsigned numPixelSamples = 4;
    Vec3f orig(0, atmosphere.earthRadius + 1000, 30000); // camera position
    std::default_random_engine generator;
    std::uniform_real_distribution distribution(0, 1); // to generate random floats in
    for (unsigned y = 0; y < height; ++y) {
        for (unsigned x = 0; x < width; ++x, ++p) {
            for (unsigned m = 0; m < numPixelSamples; ++m) {
                for (unsigned n = 0; n < numPixelSamples; ++n) {
                    float rayx = (2 * (x + (m + distribution(generator)) / numPixelSamp
                    float rayy = (1 - (y + (n + distribution(generator)) / numPixelSamp
                    Vec3f dir(rayx, rayy, -1);
                    normalize(dir);
                    // Does the ray intersect the planetory body? (the intersection tes
                    // not against the atmosphere). If the ray intersects the Earth boc
                    // is ahead of us, then the ray intersects the planet in 2 points,
                    // only want to comupute the atmosphere between t=0 and t=t0 (where
                    // the Earth first). If the viewing ray doesn't hit the Earth, or c
                    // is then bounded to the range [0:INF]. In the method computeIncic
                    // compute where this primary ray intersects the atmosphere and we
                    // of the ray to the point where it leaves the atmosphere.
                    float t0, t1, tMax = kInfinity;
                    if (raySphereIntersect(orig, dir, atmosphere.earthRadius, t0, t1) &
                        tMax = std::max(0.f, t0);
                    // The *viewing or camera ray* is bounded to the range [0:tMax]
                    *p += atmosphere.computeIncidentLight(orig, dir, 0, tMax);
                }
            }
            *p *= 1.f / (numPixelSamples * numPixelSamples);
        }
        fprintf(stderr, "\b\b\b\b%3d%c", (int)(100 * y / (width - 1)), '%');
    }
#endif
```

```
047         ...
048     }
049
```

Alien Skies

By changing the parameters of the Atmosphere model, it's quite easy to create skies which have very different looks than our Earthly sky. One could for instance imagine re-creating the atmosphere of the planet Mars or simply create his/her own imaginary sky. The scattering coefficients and the thickness of the atmosphere are the most obvious parameters you can play with to change its look. Increasing the contribution of Mie scattering quickly turns the atmosphere into a very foggy-hazy sky, which combined with light shafts, can create expressive/moody images.

What about Multiple Scattering ?

The color of the sky is the result of light being scattered once or multiple times in the atmosphere towards the viewer. However, in the literature, it is emphasised that single scattering predominates. Therefore rendering images of the sky by ignoring multiple scattering still gives very plausible results. Most models in the literature ignore or do not provide a technique to take multiple scattering into account. Bruneton however suggests that the amount of light reflected by the ground is large enough to influence the sky color. He proposes a model in which light scattered by the ground is taken into account (in this model, Earth is assumed to be a perfectly spherical shape).

Conclusion

The main few ideas you need to remember from this chapter is that the sky can be rendered as a very large volume in the shape of a sphere (surrounding another larger sphere representing the Earth). Any volume (as learned in the lesson on Volume Rendering) can be defined by absorption and scattering coefficients as well as a phase function. The sky appearance is the combination of Rayleigh and Mie scattering. Rayleigh scattering is responsible for the blue color of the sky (and its red-orange color at sunrise and sunset) and is caused by light being scattered by air molecules which size are much smaller than the light wavelength. Air molecules scatters blue light more than green and blue light. Mie scattering is responsible for the whitish hazy look of the atmosphere. It is caused by light being scattered by molecules bigger than the light wavelength. These molecules are known as aerosols. Aerosols scatter light of all wavelength equally.

Ideas of Improvements/Exercises

If you are looking to improve/extend the code, here is a list of ideas:

- Render the surface of the Earth (land and ocean) using procedural texturing or mip-mapping
- Add clouds to the sky model
- Precompute the optical depth between the sun and an arbitrary point along the ray (use this technique to speed up the program - for more details on the method, check Nishita's paper).
- Add multiple-scattering
- Compare this model with other Sky models (check the paper "A Framework for the Experimental Comparison of Solar and Skydive Illumination")

References

Display of The Earth Taking into Account Atmospheric Scattering, Nishita et al., Siggraph 1993.

Display Method of the Sky Color Taking into Account Multiple Scattering, Nishita et al., Siggraph 1996.

Precomputed Atmospheric Scattering, Eric Bruneton and Fabrice Neyret, Eurographics 2008.

A Practical Analytic Model for Daylight, A. J. Preetham et al. Siggraph 1999.

A Framework for the Experimental Comparison of Solar and Skydive Illumination, Joseph T. Kider Jr et al., Cornell University 2014.

Chapter 1 of 2 Next Chapter →