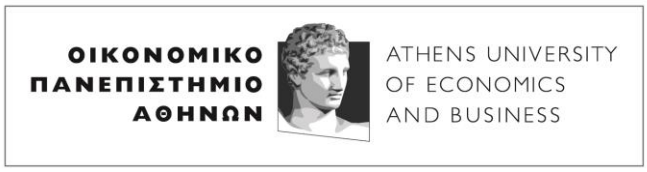
Skylighting Web Application with three.js

A dissertation
submitted in partial fulfillment of the
requirements for the degree of
Master of Science at the
Department of Informatics of the
Athens University of Economics and Business



Author: John Zobolas

Supervisor: Georgios Papaioannou, Assistant Professor, AUEB

Date: 31/5/2015

Abstract

In this thesis, we study and briefly examine different skylight models that have been proposed by the scientific community and we analyze a more recent one which provides an analytic model for full-Spectral Sky-Dome Radiance. This model is used to implement a web-based application which takes input from the user and draws a scene on a canvas element using the JavaScript 3D library three.js. This scene includes a basic 3D-model, a ground plane, the sky Dome, the Sun and the light sources that represent the radiance of the sky. A full explanation is provided for the source code and the functionality of each component of the Web-application as well as some screenshots of the scene, showing different instances of the sky dome and the 3D-model while changing the turbidity, the ground albedo value as well as the solar elevation.

Acknowledgements

Many thanks go to Professor G. Papaioannou for letting me do my “thing” and for giving helpful insights during this project. Also, a huge thanks to all the people I interacted with the previous year – friends, colleagues and more: I learnt a lot from you (I hope you also)! As always, this work is dedicated to those who improvise and want what is not in front of them.

Contents

Table of Figures

Table of Tables

1 Introduction

In the last two decades, there has been a dramatic need for modeling correctly the appearance of the sky in a way that it can be used for the realistic rendering of outdoors scenes. Such a need was formed when design professionals started wanting better sky models to use in their architectural applications; also, in the video game, virtual reality and animation industry, more sky scenes are now generated not by the old-school skybox techniques but from modern colorful models that create natural and breathtaking views of the sky representing different atmospheric conditions. Moreover, the proper capture of the earth's (or other planets) sky colors is a prerequisite of many flight and driving simulations that are used to train pilots, astronauts, etc. and it is also a necessary characteristic of complex astronomical applications like NASA's World Wind, Celestia, Google Earth and Space Engine.

To tackle this need for easier-to-use, faster and realistic sky models, researchers in Computer Graphics have spent a lot of time investigating methods to represent the sky luminance as well as the natural laws that produce the colors of the sky and have subsequently come up with two different kinds of sky models. In the first category, the generation of advanced physical-based models that try to approximately simulate the scattering of the Sun's light in the atmosphere is prominent and although the use of such models produces results as close to reality as possible, they still fail to satisfy the requirement of real-time rendering. On the other hand, the second category involves parametric sky models that are based on simple mathematical formulas for analytically computing the radiance distribution in the sky dome and whose simulation results can be easily modified by adjusting just two to three coefficients.

While in this thesis we will examine sky models that fit on both categories, the main focus will be given to the parametric models, because these are the only ones that can be used for the real-time rendering of outdoors scenes. Another benefit which comes from using an analytical model is that all such models have the same reference coordinate system as the one seen in Figure 1: this system's geometry shows the sky-dome, the Sun's position (the *solar elevation* angle is one of the main parameters that largely affect the radiance results of such models – e.g. it can be tuned to a low value so that the model will represent sunset conditions) and a random viewing direction for which we want to calculate the radiance value upon the sky dome. Moreover, for calculating the sky dome's radiance values throughout different atmospheric/weather setups, the physical-based models have to tune a lot of coefficients that represent size and particle density in the air, humidity, climate-dependent parameters, etc. whereas the most distinguished and easy to tune parameter that is used on most analytical models is the value of *turbidity*: this is a generic single value that

represents the degree of bleariness inside any sky setup from clear blue skies (turbidity equal to 1) to hazy atmospheres (turbidity equal to 7 or more).

To summarize, the main purpose of this document will be first to describe the main ideas of each sky model, discuss their advantages and disadvantages and finally explain why we chose a specific analytical model to fashion the sky appearance in our Web application and how exactly that model was implemented and integrated in our featured scene.

2 State of the Art in Sky Models

In this section, we will present the main ideas and models that influenced the graphics research activity in the field of Sky Luminance.

2.1 Sky Model Categorization

The sky models which we will examine can be briefly categorized in 4 different areas:

1. Luminance-only Sky Models: mainly these models investigate a formula that best fits the sky luminance distribution patterns while accounting for a variety of weather conditions: that is from clear to partly cloudy skies. The most known such model is Perez's mode (Perez, Seals and Michalsky 1993) and subsequently the CIE's (International Commission on Illumination) Standard on Spatial Distribution of daylight (Spatial distribution of daylight - CIE standard general sky 2002) which was pretty much Perez's formula adapted into a standard intended for architecture and lighting-design applications.
2. Sky Models that include Color Data: the new generation of graphics research that intended to add chromaticity to the sky's appearance by producing colorful skylight models which were directly useful for rendering. The main work here and a standard even in our days, is Preetham's skylight model (Preetham, Shirley and Smits 1999) which is also based on Perez's formula.
3. Night-Sky models: there is a single excellent work (Jensen, et al. 2001) that presents a physically-based model of the night sky for realistic image synthesis and which is up until now the best in this category.

4. Daylight simulation tools: this area includes models and tools which are used for accurate daylighting simulation and analysis, embedded in building/architectural applications, while providing extra features such as thermal and energy/cost savings.

2.2 Luminance-only Sky Models

2.2.1 Perez Model

The main purpose of this model - see (Perez, Seals and Michalsky 1993) - is to present a mathematical framework that calculates the luminance of any possible sky profile, by adjusting selected coefficients. The formula characterizes very well clear skies with low turbidity (turbidity is a measure of how much "heavy" or "dark" is a sky – high turbidity means a sky with hazy atmosphere whereas a turbidity equal to 1 is a clear blue sky) and that's why it was used extensively at the time in architecture and lighting-design applications, such as assessing how much light in addition to direct sunlight a given room would receive if its windows faced a certain direction, etc. Owing to the lack of color information, Perez's model can't be directly used for rendering, but it's still useful for its original purpose.

The formula devised by Perez and his associates (which was an improved version of an even older one – see (CIE, Standardization of Luminance Distribution on Clear Skies 1973)) is as follows:

$$F_{Perez}(\theta,\gamma) = \left(1 + Ae^{B/cos\theta}\right)(1 + Ce^{D\gamma} + Ecos^2\gamma) \qquad (1)$$

In formula (1), γ is the angle formed by the view direction and a vector pointing toward the Sun, θ is the angle formed by the zenith and view direction, and e is the Euler number (see Figure 1). The values returned by these formulas are normalized so that the value at the zenith is 1. Multiplying this value by the luminance at the zenith yields the final luminance distribution function:

$$Y = \frac{F(\theta,\gamma)}{F(0,\theta_s)} * Y_z \qquad (2)$$

where θ_s denotes the angle formed by the Sun and zenith and Y_z is the luminance at the zenith.

One can plainly see that from the formula (1) is based on the values of the coefficients A, B, C, D and E: though these values can be used to tune the luminance distribution and they were derived from a large, high-quality experimental set of sky-scan data, they don't directly translate to any physical quantities.

Perez, by experimenting with these 5 parameters, he managed to establish a connection between each one of them and a specific *physical effect* on the sky distribution. And so, he presented graphs which

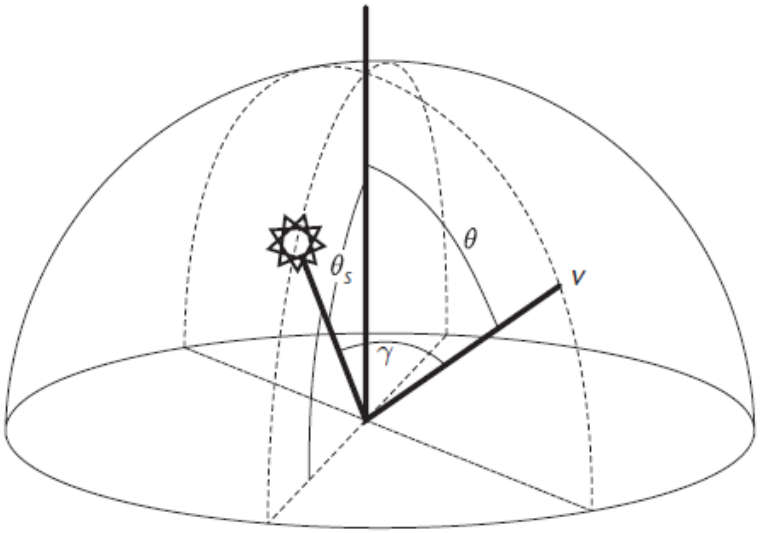


Figure 1: Coordinate System used by CIE and Perez models

show for example, that the A coefficient is responsible for creating either a darkening ($A > 0$) or a brightening ($A < 0$) horizon region with respect to the zenith angle θ (and thus corresponding to overcast and clear sky conditions). Similarly, he showed that the B coefficient relates to the luminance gradient near the horizon, C to the relative intensity of the circumsolar region, D to the width of the circumsolar region and E to relative backscattered light.

Also, because these five coefficients may seem somewhat simple but also intuitive, someone can see from the model's derivation method of the formula (1), that they are functions of three values: the sky's clearness ε, the sky's brightness Δ and the solar zenith value θ_s. These functions are derived via nonlinear least-squares fitting of equation (1) to a large number of experimental data points. As such, if we provide the 3 above simplistic and easy-to-understand values (which in turn represent a sky profile) we can produce the 5 model's parameters which in turn can give us the luminance distribution of the specific sky profile. Note that the term *sky brightness* was first introduced by Perez and has nothing to do with the official term used by the CIE.

Overall, Perez's model provided a simple mathematical framework which can compute a large quantity of sky luminance patterns and it stood as a base upon which many other models where build.

2.2.2 CIE Standard

While the Perez's model was based on a nice and simple mathematical framework, it could not be easily used in design and business applications: you couldn't know beforehand what values to use for the 5 parameters and what would the sky result be without having first indulged in many tests and experimentation on the coefficients. The CIE, which is the global international authority on light, illumination, color and color spaces, managed to establish a list of tabulated values, which comprises 16 models describing luminance distributions for atmospheric conditions varying from clear cloudless sky to heavily overcast weather, with different luminance turbidities. This standard, presented in the report (CIE, Spatial distribution of daylight - CIE standard general sky 2002) was intended for two purposes:

1. to be a universal basis for the classification of measured sky luminance distributions
2. to give a method for calculating sky luminance in daylighting design procedures

The CIE formula was slightly different than the one Perez used (see equation (1)) and it was also found to be less accurate because of its generality in some cases (meaning that if you knew exactly which 5 coefficients gave you the sky profile you wanted, you could get better results with Perez's model than from one of CIE's 16 models):

$$F_{CIE2002}(\theta,\gamma) = \left(1 + Ae^{B/cos\theta}\right)\left(1 + C\left(e^{D\gamma} - e^{D\frac{\pi}{2}}\right) + Ecos^2\gamma\right) \qquad (3)$$

All values from the above formula are subject to the coordinate system of Figure 1. An example of the standardized parameters is given in the next table, in which the 5 coefficients constitute two different models that describe overcast skies[1]:

Type	*Grada-tion group*	*Indica-trix group*	*a*	*b*	*c*	*d*	*e*	*Description of luminance distribution*
1	I	1	4,0	-0,70	0	-1,0	0	**CIE Standard Overcast Sky,** Steep luminance gradation towards zenith, azimuthal uniformity
2	I	2	4,0	-0,70	2	-1,5	0,15	Overcast, with steep luminance gradation and slight brightening towards the sun

Table 1: The first two models of the CIE Standard

[1] Sometimes, instead of the Sky Type 1, the "traditional" Overcast luminance distribution is used: $\frac{L_{(\gamma)}}{L_z} = \frac{1+2\sin\gamma}{3}$, where γ is the angle of elevation of the sky element above the horizon and L_z the zenith luminance.

2.3 Sky Models that include Color Data

2.3.1 Nishita Model

It is a silent convention that the skylight models get the name of the main author of the article that proposed them: so *Nishita's* model was one of the first colorful skylight models that was directly useful for rendering and it was described in the paper (Nishita, Sirai, et al. 1993). Later, a more sophisticated and revised model was introduced in (Nishita, Dobashi and Nakamae 1996) mainly by the same authors.

The Nishita's model is the first physically-based skylight model that displays *the earth as viewed from outer space* (see Figure 2) the intention was for the model to be used in space flight simulators (e.g. reentry to the atmosphere) or for simulation of surveys of the earth (not from the ground but as it would be to observe earth from a satellite for example – a position within the atmosphere). The authors emphasized the need for physically-based accuracy (contrast to the static 3D images/models of earth) in order to create attractive images of our planet: there are some phenomena such as the red coloring of the horizon or the color of the clouds that play a very important role in such an image synthesis and which cannot be correctly visualized unless the model takes into account the optical effects which are caused by particles in the atmosphere (atmosphere scattering) and the relationship between the Sun's position and the viewpoint. Also, when earth is viewed from space, the sea (oceans mostly) play a very important role to the whole appearance and its color is largely affected by the absorption/scattering effects due to water molecules.

So, in conclusion, the Nishita's model was based and formed around these 4 characteristics:

1) Calculation of the *spectrum of the earth* viewed through the atmosphere; the earth is illuminated by direct sunlight and sky light affected by atmospheric scattering. Also, sunlight is absorbed when light passes through the atmosphere, and sky light consists of light scattered by particles in the air. On the way, passing through the atmosphere the light is attenuated, and its spectrum changes.
2) Calculation of the *spectrum of the atmosphere* taking account of absorption/scattering due to particles in the atmosphere. The Nishita's model was the first to mention and include the characteristics of the particles found in earth's atmosphere (e.g. size, density distribution changing exponentially with altitude): the scattering analysis includes small particles such as air molecules

which produce *the Rayleigh scattering effect*[2] and the aerosols such as dust which contribute to *Mie scattering*[3].

3) Calculation of the *spectrum on the surface of the sea* taking into account the radiative transfer (scattering light) of water molecules and the reflected light at the surface.
4) Calculation of the *color of the clouds*: mostly due to sunlight scattering from particles of the clouds (whose sizes are larger than that of air molecules or of aerosols). The geometry of the clouds is simply modeled by applying 2D fractals to multiple imaginary spheres (representing clouds with multiple altitudes) around the earth's geometry sphere.

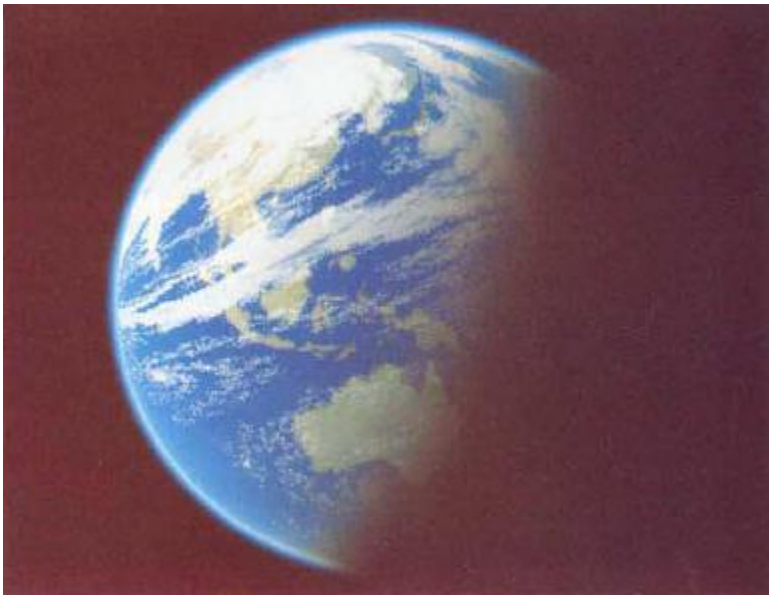


Figure 2: Simulation image showing Earth from space using Nishita's model

For the calculations included in 1) and 2) the authors used numerical integrations which are effectively solved by using several (various) lookup tables and by making assumptions such as the simulation of the earth's shape as a perfect sphere and the hypothesis that the sunlight is a parallel light. It is important to mention though that the model ignores the multiple scattering of light between the air molecules and aerosols in the atmosphere and the interreflection of light between the earth's surface and

[2] **Rayleigh scattering** is the (dominantly) elastic scattering of light or other electromagnetic radiation by particles much smaller than the wavelength of the radiation. Rayleigh scattering causes the blue hue of the daytime sky and the reddening of the Sun at sunset.
[3] **Mie scattering** describes situations where the size of the scattering particles is comparable to the wavelength of the light, such as the atmosphere's aerosols (cloud, mist, haze, dust, fog, etc.)

particles in the air because of its high computational cost. So, Nishita's model simulates the sky's appearance due to *single scattering* only (see Figure 3 for an example of the linearity of the atmospheric zone layers that is a result of that). For 3), they showed that an analytical solution is available instead of numerical integrations (which sped up the calculations).

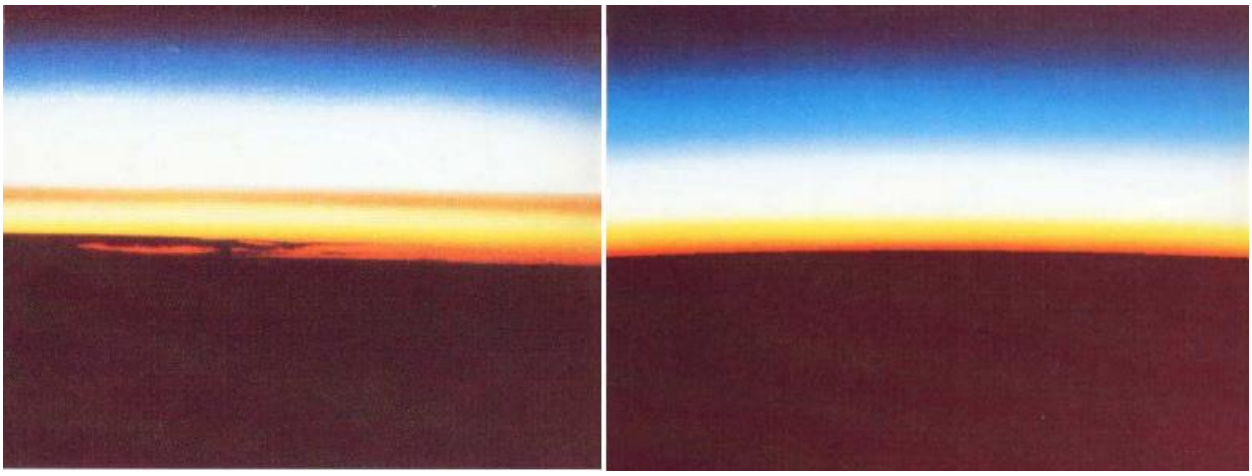


Figure 3: Comparison between NASA's space shuttle photographs (left) and Nishita's model (right)

Nishita and his colleagues later presented a revised model – see (Nishita, Dobashi and Nakamae 1996) – that used discretized precomputation to simulate multiple scattering. The main subject of their work was how to display realistic clouds. To do that, they considered two factors: one is the *multiple scattering* due to particles in clouds (the algorithm proposed calculates at least the 3rd order of multiple scattering) and the other factor is the *sky light* (including ground albedo[4]) due to the large effects that it may have on the color of the clouds when it is stronger than the sunlight (this usually happens at sunset and/or sunrise).

Regarding the computation of the multiple scattering inside the clouds, the authors suggested subdividing a cloud into space voxels (volume elements) and calculating the energy transferred, absorbed and scattered between these voxels. They used the well-known form factor from the radiosity models to represent the ratio of the total scattered and emitted energy leaving one voxel and absorbed or scattered by another voxel. They also used the fact that inside the clouds, scattering of light due to particles (water droplets) obeys Mie scattering and it usually takes the form of *strong forward scattering* (the phase function is narrow – meaning that the original incident intensity is "preserved" in the original incident direction of the light beam): this suggests that if we know beforehand the light direction we could predict which voxels

[4] **Albedo** (for ground, also: reflection coefficient): is the reflected sunlight from the ground or more general the diffuse reflectivity or reflecting power of a surface. In most models, it takes values from the range [0,1].

(which create a sample space – a lobe of space volumes – from the "voxelized" cloud) affect a specified volume element in the cloud (usually we are interested for the one that the viewer's direction is pointing at). Also, it is known that in such a scenario the form factors of distant voxels are small: the ones that contribute more to the energy transfer are the ones that are inside that sample space.

So, the authors pre-compute the light scattering inside this sample space before the calculation of scattering due to every voxel in the whole cloud: this technique is done for all the 2nd and 3rd order scattering paths and the results are stored in a table cell different for each voxel. These results (each voxel's contribution ratio to the total intensity) can then be used as a *reference pattern (or template)* and by doing this, the total calculation cost for the rendering process is highly reduced. Even though the authors minimized the computational cost, their simulations still had to run 20-30 minutes (while pre-calculating no more than the 3rd order of path scattering) to produce images of the clouds such as those seen in Figure 4:

Figure 4: Examples of clouds using Nishita's model

These examples depict beautiful variations in the color of the clouds and sky. One can see the bright edges of the clouds in the left image of Figure 4 because the Sun's position is behind them, as well as the red coloring of the sky and clouds in the right image during a sunset. Finally, the authors produced another work (Nishita, Dobashi and Kaneda, et al. 1996) in which they focused only on the spectral distribution of sky radiation and not the clouds, while taking into account the multiple scattering of light in the atmosphere. Their method was the same as before (all space "voxelized") and their simulation results for up to 2nd order of path scattering (lasting 1-2 minutes) produced nice reference sky images.

2.3.2 Preetham Model

Despite the efforts of Nishita and his team, their model could not produce realistic images of the sky for real-time rendering: a different approach was needed, one that was not so computational expensive, would model the aspects of the atmosphere that produce the color of the sky and could also approximate the hue/shading effects of it. Arcot Preetham and colleagues found that the most significant difference between the indoor and outdoor scenes is that when outside, illumination comes directly from the sun and sky; and the distances involved make *the effects of air visible*. These effects of air are manifested as the desaturation and color shift of distant objects and are usually known as *aerial perspective* (see Figure 5 for rendered images with and without these effects).

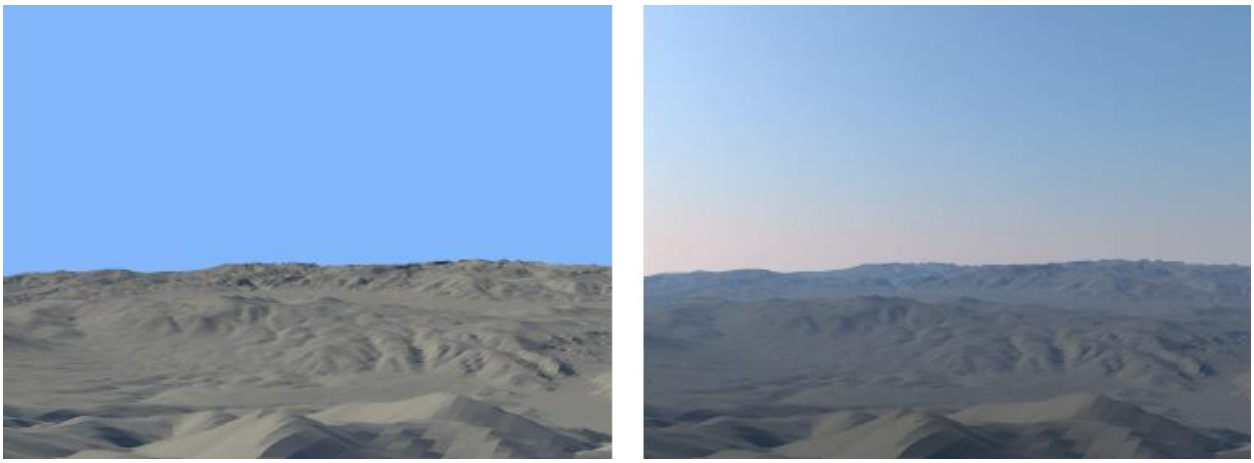


Figure 5: Left: A rendered image of an outdoor scene with a constant colored sky and no aerial perspective. Right: The same image with a physically-based sky model and physically-based aerial perspective

So, a new model had to be made and it needed two formulas. The first should describe the spectral radiance of the Sun and sky in a given direction. The second should describe how the spectral radiance of a distant object is changed as it travels through the air to the viewer. Also, these formulas should be able to take as input, data that is generally available or at least possible to estimate. Such description fits Perez's formula which computed sky luminance: now the new formula should calculate sky spectral radiance instead. So, the widely used Preetham model (Preetham, Shirley and Smits 1999) is based directly on Perez and his colleagues' formula. But unlike the CIE models, it calculates the coefficients A to E, analytically. Actually, Preetham and his colleagues first generated reference images using Nishita Model and then, by nonlinear optimization, fit the Perez formulas to those images. The result was a set of linear functions that

take one parameter, turbidity, as an argument (see next paragraph for an analytical explanation). These functions return all five parameters (A to E), as well as a bicubic function of turbidity and the solar elevation angle for calculating zenith luminance.

Turbidity is a measure of the fraction of scattering due to haze as opposed to molecules. This is a convenient quantity because it can be estimated based on visibility of distant objects. More formally, turbidity is the ratio of the optical thickness of the haze atmosphere (haze particles and molecules) to the optical thickness of the atmosphere with molecules alone:

$$T = \frac{t_m + t_h}{t_m} \qquad (4)$$

where t_m is the vertical optical thickness of the molecular atmosphere, and t_h is the vertical optical thickness of the haze atmosphere. So, if the haze is non-existent $(t_h = 0)$ you get the typical clear blue sky with $T = 1$. Although turbidity is a great simplification of the true nature of the atmosphere, atmospheric scientists have found it a practical measure of great utility: it does not require complex instrumentation to estimate turbidity and so it is particularly well-suited for application in graphics.

The Preetham model also provides two chroma channels (x and y) that are calculated using the same approach (fit the formula to the reference data) and the authors provided a way to convert the outputs to spectral-radiance data. The resulting model is fast and easy to implement (it needs only two values: turbidity and Sun position) and is considered the de facto standard analytic model of spectral sky-dome radiance. An example of a sky profile with high turbidity $(T = 6)$ that this model can produce, can be seen in Figure 6:

Figure 6: Examples of sky radiance with high turbidity using the Preetham's model (left: morning, right: near sunset)

There is a research work (Habel, Mustata and Wimmer 2008) that used the Spherical Harmonic (SH) Lighting technique in conjunction with the Preetham model to produce fast and compact representations of sky domes as well as to efficiently compute the illumination of objects by the sky. Spherical harmonic (SH) lighting is a family of real-time rendering techniques that can produce highly realistic shading and shadowing with comparatively little overhead. All SH lighting techniques involve replacing parts of standard lighting equations with spherical functions that have been projected into frequency space using the spherical harmonics as a basis.

Figure 7: Scene lit at sunset using Preetham's model and the SH rendering technique

In this work, the researchers' goal was to reformulate the Preetham model by representing the spherical harmonic weights as functions of the skylight model parameters. By using the symmetrical geometry of the sky dome, they reduced the parameter space to just two dimensions (Sun's azimuth angle θ_s – see Figure 1 and turbidity) and observed that all SH-weights show a largely polynomial behavior in that space. So, they performed a polynomial two-dimensional non-linear least square fit for the principal parameters to achieve both negligible memory and computation costs while minimizing the mean reconstruction error (since after finding the polynomial coefficient matrix for each SH-weight they needed to reconstruct the SH-weights in real-time – that is once per frame). Additionally, they executed a domain

specific Gibbs phenomena[5] suppression to remove ringing artifacts[6]. An example of how well this model can represent the shadows and hue colors of the environment's objects can be seen in the sun-setting scene of Figure 7.

After Preetham's model became widely used, many people using this specific model decided to go with a turbidity range of 2 to 6 (the simulation and fitting only took these turbidities into account). This at first was not a problem since these values capture a lot of atmospheric conditions and sky profiles. After some years though, an investigation was done by (Zotti, Wilkie and Purgathofer 2007) in which they showed that indeed the Preetham model partly due to the limitations of the Perez formula, and partly due to the simple linear functions chosen for calculating the parameters, is incapable of reproducing correct absolute luminance values in comparison with the recommended values from the CIE Standard. The analysis showed these deficiencies:

1) Preetham's model could produce *negative luminance values* (for turbidities $T \leq 1.6$ and mainly during a sunset/sunrise)
2) For higher turbidities $(T \geq 6)$ the *luminance distribution is considerably off*: the zenith is not dark enough, the Sun does not have the usual bright color and the horizon is painted with a uniform orange-yellow color (see Figure 8)

So, since the Preetham model uses fitting to ad-hoc formulas, it can only work over a limited range of turbidities – and this particular range presents a trade-off between usefulness and accuracy. From the way that it was created, the Preetham Model can be at most as good as the Nishita Model with simulation of at most 2nd order of scattering. This showed the need for a better simulation of atmospheric scattering which would be used as a base to improve or replace Preetham's model, so that the data fit will produce more natural results. Such models are briefly described in the next section.

[5] The **Gibbs phenomenon** here happens because the Skylight models are only defined on a hemisphere, whereas the spherical harmonics require a spherical environment.

[6] In Signal Processing, **ringing artifacts** are artifacts (meaning errors) that appear as spurious signals near sharp transitions in a signal: here the discontinuity happens near the horizon.

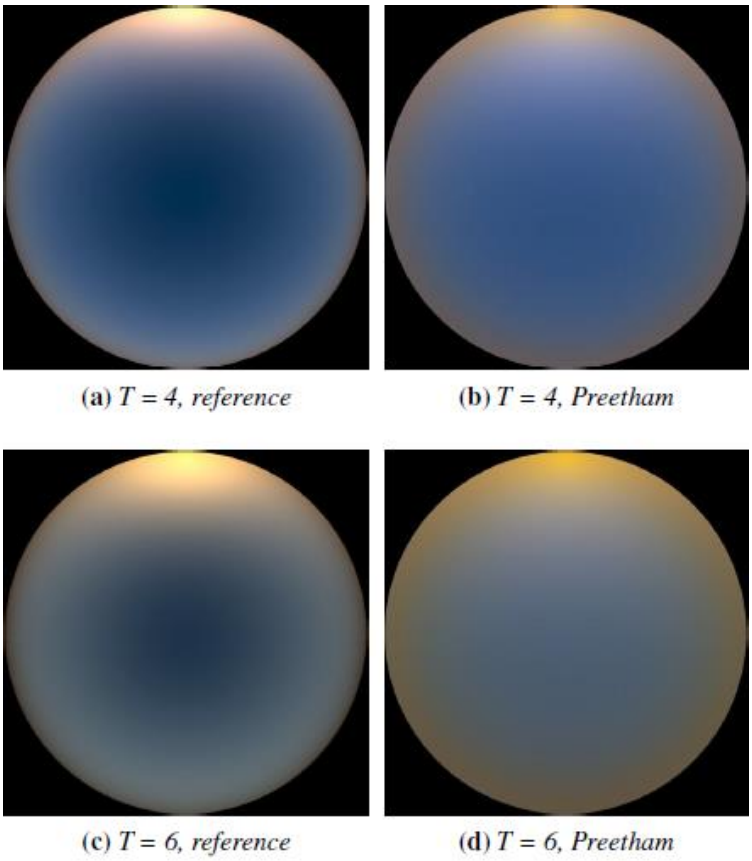


Figure 8: Comparison between reference image (computed with a path tracer for the Hosek-Wilkie Sky Model – see "The path tracer" section) and Preetham's model with a solar elevation of 6° (sunrise)

2.3.3 Physical-based models

In this section, we provide information about relatively new models that use an approach that is based on physics laws as well as smart computational techniques to achieve full spectral radiance models of the sky dome without sacrificing on the beauty and naturalness of the earth's sky at different weather and atmospheric conditions. As we will see, this usually means higher computation times for the simulations, more realistic results and generally complex models.

Such a physically based model (it can be seen as an improvement of Nishita's one) was presented by Haber et al. – see (Haber, Magnor and Seidel 2005). The authors of this model were inspired by the beautiful color of the sky during the twilight period before sunrise and after sunset: they believed that such sky profiles cannot be adequately simulated with a parametric model and in order to produce a more realistic

model based on physical laws and parameters which can be measured for different climate conditions, the model must take into consideration these:

- solar irradiance spectrum and its absorption in the ozone layer,
- wavelength-dependent refraction of direct sunlight in the atmosphere,
- climate-dependent composition and size distribution of aerosols / dust particles,
- height-dependent air, humidity, and aerosol density,
- Rayleigh scattering (air molecules) and Mie scattering (aerosols),
- radiative transfer (high-order multiple scattering), as well as
- The shadow of the Earth.

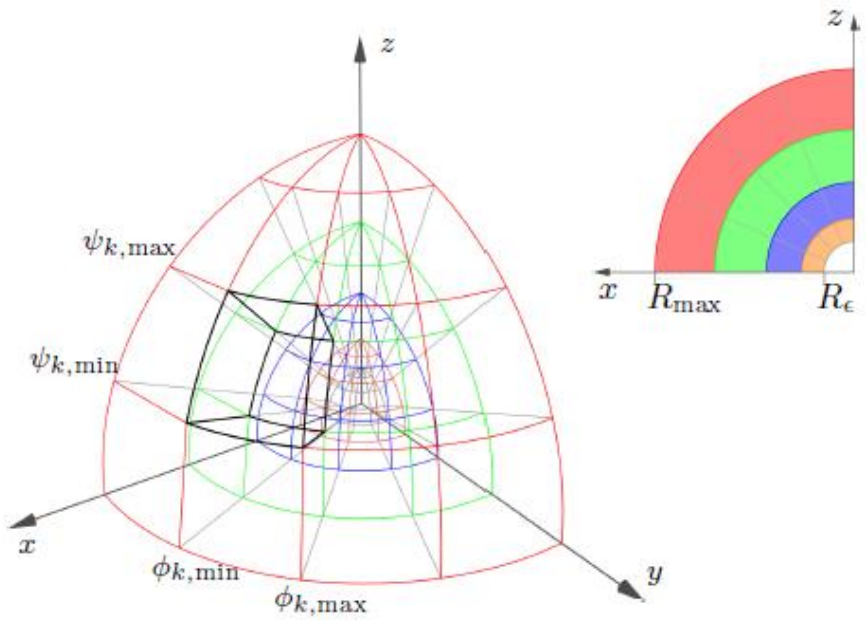


Figure 9: Spherical parametrization of Haber's atmosphere model (one quarter of the hemisphere) with color-coded shells

So, the model included 5 steps to compute the colors of the sky for an arbitrary observer position and observation date and time:

1. Compute the *position of the Sun* from the observer position and the observation date and time.
2. Set up the *atmosphere model* (both atmosphere layers and atmosphere cells – see Figure 9 for the latter) which is the same basic principle used in Nishita's paper (subdivision of the atmospheric body into discrete blocks and then calculation of radiative transfer between them – this procedure is a case of a *volume radiosity algorithm*). Atmosphere layers are used to store

optical properties of the atmosphere, while atmosphere cells are used during the radiative transfer computations (Step 4). The initialization of all wavelength-dependent optical parameters is done by using the *OPAC software* – see (Hess, Koepke and Schult 1998) . The authors take eight samples in their calculations covering the wavelength range of visible light – that is from 380nm to 720 nm and usually use 20-50 atmospheric layers (in each one the parameters change depending on the height).

3. Compute the *direct illumination* contribution of the atmosphere from the Sun taking into account the atmospheric refraction (from air molecules), the ozone layer, and the shadow of the Earth.
4. Compute the *indirect illumination* contribution by simulating multiple scattering events between atmosphere cells (taking into account Rayleigh and Mie scattering) - this is by far the computationally most expensive step and the authors present several algorithmic optimizations and permissible approximations to reduce the computation times to about 1-2 hours.
5. Convert the illuminated atmosphere into an RGB sky texture as seen from the observer.

Given the correct configuration of atmospheric constituents, this model can realistically predict appearance of the sky over a wide range of atmospheric conditions (see Figure 10) although there is a high cost regarding the computation rendering time. All images presented in Haber's paper simulated multiple scattering with five scattering events whereas extensive tests showed that they obtained sky map radiances with less than 1% maximum error. In the form it was presented, the model does not account for the influence of ground reflectance or clouds, although according to the authors, it could easily be modified to do so.

Figure 10: Simulation result of Haber's model at sunrise with different kind of climates. Left to right: continental climate with average pollution, urban climate with heavy pollution, and tropical climate with no pollution

Another physically based model, was presented by Eric Bruneton and Fabrice Neyret – see the paper (Bruneton and Neyret, Precomputed Atmospheric Scattering 2008). The authors proposed a method to render the earth's atmospheric events in *real time*, from any viewpoint from ground to space. This method includes Rayleigh and Mie multiple scattering and ground albedo (reflectance), which is important to correctly render twilight sky colors, the shadow of the Earth inside the atmosphere and even the mountain's shadows and light shafts (that usually can be seen in such cases).

To get to a real-time based solution, the authors based their model on moderate simplifying assumptions that allowed them to get a better approximate solution of the *rendering equation*[7] – one that has pre-computable terms for all viewpoints, view directions and Sun directions. For the multiple scattering, their solution is based on an exact computation for zero and single scattering, and uses an approximation of occlusion effects to compute the higher-order scattering events. The multiple scattering computation is done in an incremental manner: one scattering order is calculated at time, always from the data for the previous order. It can be seen in Figure 11 that the model takes into account the shadow volume and light shafts (which are valuated at the zero and first scattering events) and then it approximates the integration from x to x_0 with the x to x_s, while ignoring occlusion (which is implicit via the use of x_s). So, while light beam (a) is unchanged, (b) is affected by ignoring occlusion of the secondary scatters (this yields both positive and negative bias and the effect is small anyway).

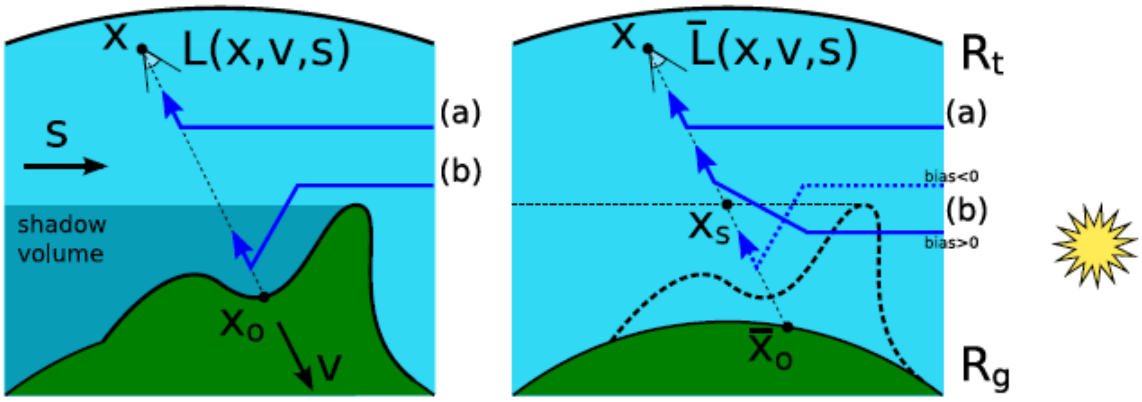


Figure 11: Exact Solution (left) and approximate solution (right) of Bruneton's model

[7] The rendering equation is: $L(x,v,s) = (L_0 + R[L] + S[L])(x,v,s)$, where $L(x,v,s)$ is the radiance of light reaching x from direction v when the sun is in direction s, L_0 is the direct sunlight L_{sun}attenuated before reaching x, $R[L]$ is the light reflected at x_0 and also attenuated before reaching x, and $S[L]$ is the inscattered light – the light scattered towards x between x and x_0.

The pre-computation algorithm was implemented on GPU (with fragment shaders processing the numerical integration) which saves disk space for the 4D-table that results from the algorithm and allows to quickly change atmospheric events (5 scattering orders are computed in 5 *seconds* on a NVidia 8800 GTS). Some results of this model can be seen in Figure 12:

Figure 12: Bruneton's model simulation results: light shafts and the mountains' shadows are visible. The sunset is depicted nicely too.

One last physically based model that incorporates multiple scattering, including scattering in water bodies, and that allows for real-time rendering by utilizing a set of lookup tables was presented by Elek and Kmoch (Elek and Kmoch 2010). In contrast to the previous model (Bruneton and Neyret 2008), this one's focus was on real-time rendering of *arbitrarily dense planetary atmospheres and large bodies/areas of water* (e.g. oceans), although the method which was used was built upon the one from Bruneton's model: that is subdivision of the atmospheric body into discrete blocks, precompute a 4-dimensional look-up table (time-costly) which involves incremental computation of the scattering dataset and then rendering in real time images from any position and at any incident light angle (Sun's position): the pre-computation may take ~1.5 hours (which is a little more than Bruneton's model) but that happens because in this model the authors also evaluate the reflection and scattering from water bodies. Lastly, this model is also fully spectral and takes account of the ground albedo (its value is pre-decided and baked into the look-up table).

Regarding research for real-time rendering of water bodies (which usually involves models that are very similar to the atmospheric ones), there exist two kind of theories: the ones that explain and generate the surface wave geometry – such as in (Bruneton, Neyret and Holzschuch 2010) – and the ones that try to

simulate the light transport in the water volume. The first model that complies in the second category, and which used an analytical expression for computing the first scattering order in water to render oceans (by performing minimum changes to an existing sky model), was the Nishita Model. Later, other models were introduced which included higher scattering orders but remained computationally expensive. Elek's model used the same model for light scattering in the water that he used for the atmospheric scattering, while changing and/or taking into account water-specific attributes such as the existence of additional organic and inorganic compounds (e.g. the phytoplankton), the significant absorption mainly in the red part of the spectrum, the constant water density and the neglecting of Mie scattering – partly because there are not sufficient data for the coefficients used and mainly because the behavior of Mie scattering is strongly anisotropic with a dominant forward lobe, so it would only have noticeable effect for an underwater observer looking towards the light source (which is not in the scope of this model). Comparison of various water types' appearance at morning and afternoon and with diverse types of aquatic environment can be seen in Figure 13:

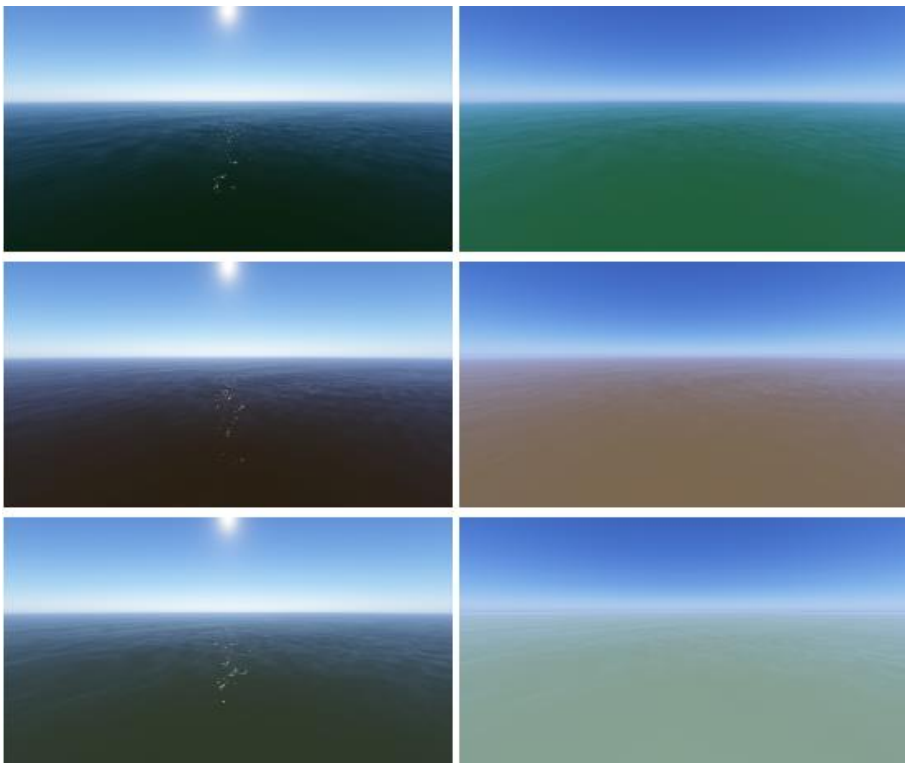


Figure 13: Simulation of different water bodies (top to bottom: high algae concentration, muddy water, high phytoplankton concentration) at morning (left) and afternoon (right) with Elek's model.

2.4 Night Sky Model

The most popular paper that presents a physically based model of the night sky and it pretty much accounts for all significant sources of natural light that exist in a sky night is (Jensen, et al. 2001). It includes the Moon, the stars, the Milky Way[8], the zodiacal light[9], the airglow[10], etc. (see Table 2 for a complete list of these light sources and their respective irradiance values). The model uses astronomical data for its calculations (e.g. position of Sun and moon, astronomical coordinate systems, etc.) and not only simulates the direct appearance of celestial elements (such as the Moon or the stars) but also provides analytical formulas for the illumination that they produce. For example, the Moon is simulated as a geometric model illuminated by the Sun, using recently measured elevation and albedo maps, as well as a specialized BRDF[11]. All the model's components can be seen in Figure 14.

Component	Irradiance $[W/m^2]$
Sunlight	$1.3 \cdot 10^{3}$
Full Moon	$2.1 \cdot 10^{-3}$
Bright planets	$2.0 \cdot 10^{-6}$
Zodiacal light	$1.2 \cdot 10^{-7}$
Integrated starlight	$3.0 \cdot 10^{-8}$
Airglow	$5.1 \cdot 10^{-8}$
Diffuse galactic light	$9.1 \cdot 10^{-9}$
Cosmic light	$9.1 \cdot 10^{-10}$

Table 2: Typical values for sources of natural illumination at night

The final rendering of the night scenes was done with a Monte Carlo ray tracer. Specifically, for simulating the scattering in the atmosphere the authors used a spherical model of the earth similar to that of (Nishita, Dobashi and Kaneda, et al. 1996). The most computational step in the rendering of sky images is the multiple scattering in the atmosphere, which by using Nishita's model, it only includes up to the second order of scattering: that's why the rendering time for most of the individual images presented in the paper ranged from 30 seconds to 2 minutes, except from the ones that included clouds (~2 hours).

[8] **Milky Way** is the galaxy that contains our Solar System. From Earth, the Milky Way appears as a band because its disk-shaped structure is viewed from within.
[9] **Zodiacal light** is a faint, roughly triangular, diffuse white-yellow glow that is seen in the morning night sky and is caused by sunlight scattered by space dust around the earth.
[10] **Airglow** is a faint emission of light by a planetary atmosphere (in our case, Earth's).
[11] **BRDF** (Bidirectional reflectance distribution function - $f_r(\omega_i, \omega_r)$) is a function takes an incoming light direction (ω_i) and outgoing direction (ω_r) and returns the ratio of reflected radiance exiting along (ω_r) to the irradiance incident on the surface from direction (ω_i).

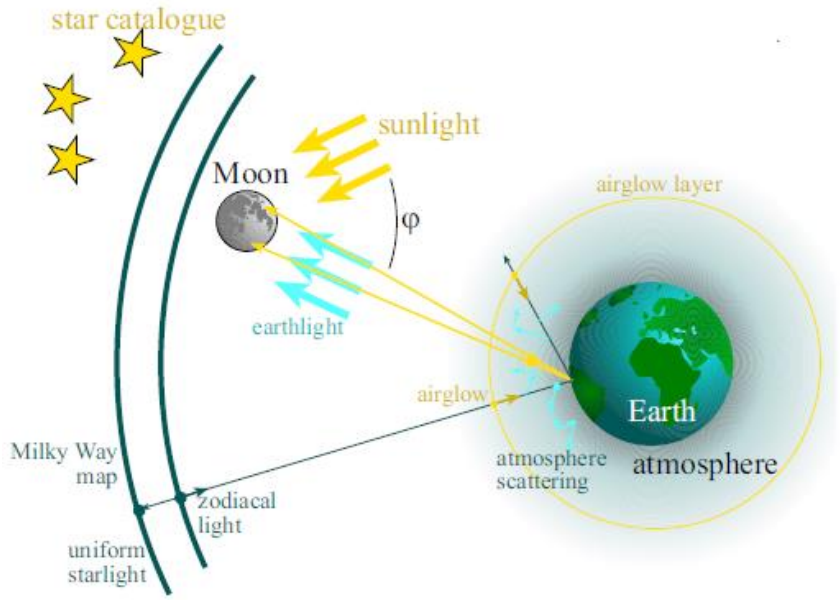


Figure 14: The components of the night sky model

Lastly, the authors emphasized that the "sense of night" is a very important concept when generating images of the night sky: it requires carefully ensuring correct physical values (that's why they used accurate astronomical data measurements) and using a *perceptually based tone-mapping algorithm*. The latter is needed because for a daylight simulation it is less noticeable if the sky intensity is wrong by a factor two, but in the night sky all the components must work together to form an impression of night. Examples of night sky rendered images are seen in Figure 15:

Figure 15: Left: The effect of multiple scattering on the Moon and the sky just before sunrise. Right: A clear moonless sky with the Milky Way and many stars visible.

2.5 Daylight simulation tools

2.5.1 Solene

A high-performance daylight simulation tool is *Solene*, which is the product of research of the CERMA laboratories. It is a software for simulating sunshine, lighting and thermal radiation on urban territories/morphologies and indoor architectural spaces. In the paper (Miguet and Groleau 2002), the authors give a simple description of Solene and its general features and present the model that is used to simulate the environmental (outdoors and indoors) luminance. Their main goal is to present a way which overcomes the technical difficulties that are raised when the light hits many transparent and/or reflective areas and glazing [12] surfaces and is generally able to give realistic results when there are multiple transparencies. This is something new, since most existing tools suffer from disadvantages such as restrictive conditions about the input geometry (rectangular shapes, single window), limited sky conditions (CIE normalized skies – see CIE Standard), and a reduced field of investigation due to a separate approach for indoor and outdoor spaces.

For simulating the solar irradiance, Solene uses a statistical radiance model which can depend on many parameters, such as the solar altitude, turbidity, etc. The sky dome is simulated as a hemisphere of infinite radius and a process of sectorization takes place as seen in Figure 16 – the outcome of which is a triangulated mesh consisting of "sky patches". Having 1024 such elements is a good compromise between precision and calculation speed regarding the sky dome's luminance. Now, for each "sky patch", which can be uniquely described by the ζ azimuth angle and relative γ angle from the Sun's direction, the relative sky luminance is given by Perez's formula (1), where $\theta = \zeta$. The authors find it better to use an all-weather model (Perez's) than the CIE Standard on most cases.

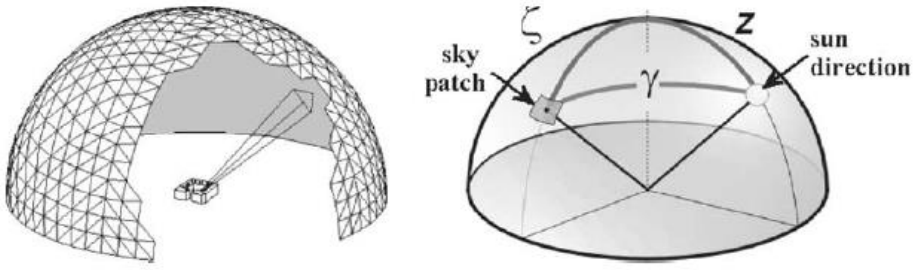


Figure 16: The triangulated sky dome of Solene's sky model

[12] **Glazing** is a part of a wall or a window which is made of glass

For calculating the luminance of the environment, the authors first divide the scene into smaller patches and then for each one of them, sum the luminance of all the sky patches that are visible to it (this depends also on the solid angle that each scene patch is "seen" from the sky patches). After that, the Solene's model must calculate the luminance of areas that are not seen by any sky patch: these areas as mostly illuminated by reflected light from other surfaces. To solve this, the authors make these assumptions (they work for most architectural materials):

1) There is no re-emission of light towards the sky
2) Emitted or received energy and reflectance are constant over a whole patch (true if patches are small enough)
3) Every non-transparent surface is considered as opaque[13] and Lambertian[14]

Solene's method incorporates the *Radiosity model* for calculating the radiative transfer between the scene's patches. The idea is that every patch in the scene (being outdoors or indoors) can exchange luminous energy with every other "scene patch": the way the exchanges of light take place in the environment depends on the incoming energy (which was calculated in the previous step) and the geometrical relationship between surfaces (from the above assumptions every patch has one reflectance ρ only). So, the radiosity of a given patch is defined as the auto-emitted flux increased by the reflected part of the flux coming from all other surfaces as seen in Figure 17 (where B_j is the radiosity of patch j, and F_{ji} is the form-factor[15] of patches j, i):

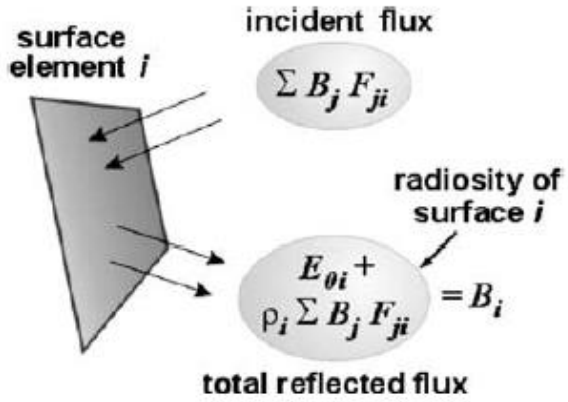


Figure 17: Diagram for the radiative balance of a patch *i*

[13] **Opaque** surface is a surface that you cannot see through it
[14] **Lambertian** is an ideal diffusely reflecting surface in which the luminous intensity observed is directly proportional to the cosine of the angle θ between the direction of the incident light and the surface normal
[15] **Form-factor** is the geometrical relationship that describes the proportion of distributed energy between two patches

For calculating the form factors, which is the most computational expensive step in any Radiosity model, the authors use an efficient analytical contour-integration technique[16] as well as a hidden surface removal process. Lastly, to solve the glazing and multi-transparencies problem, the authors suggested to not use the glazing areas as new diffuse light sources (as many other tools do) but as normal "scene patches" with extra properties: each glazing side will be added a different transmission factor ρ and normal (see Figure 18). In this way, a window patch can act both as a normal opaque patch but also as a transparent surface that permits some light to cross through. With this specific approach, multi-reflections are allowed in Solene and calculated analytically· also there isn't the slightest limitation on the number of glazing surfaces, their position or geometrical shape.

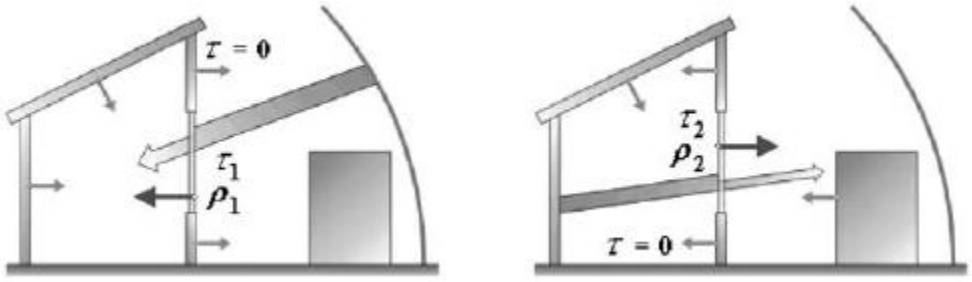


Figure 18: The glazing has different normal and transmission factors depending on the direction of the light's crossing

2.5.2 Daylight in buildings

The (Johnsen and Watkins 2010) is a report build specifically for design and energy engineers that want all the necessary information of how the outside lighting affects the interior of the buildings and how low energy consumption can be achieved with specific design principles. It was created jointly by the International Energy Agency (IEA) "Energy Conservation in Buildings and Community Systems Program" and the "Solar Heating and Cooling Program". Mainly the report describes all the possible ways that daylight and sunlight can be brought inside a building and how well the different options perform. It also examines the ways of controlling daylight and electric light so that internal conditions are maintained within design limits and the energy use is reduced. Also, it gives an analytical review of the most important daylight simulation tools that an engineer can use to help him incorporate daylight into building design and

[16] **Contour integration** is a method of evaluating certain integrals along paths in the complex plane

predict its performance. Lastly, the report gives examples from case studies where both traditional and innovative daylight systems have been demonstrated and evaluated.

3 An Analytic Model for Full Spectral Sky-Dome Radiance

In this section, we will provide extensive details about the Hosek-Wilkie Skylight Model that was first presented in the scientific article (Hosek and Wilkie 2012) and was used as a basis to build our Skylighting Web application (see Sky Model Implementation). The main characteristics of this model (and the reasons why it was chosen) are:

1. It is a full-spectral model; this means that it handles each spectral component independently (including the ultraviolet range)
2. It is a parametric, analytical, flexible and real-time Sky Model; it can be seen as an extension of the Preetham Model with the necessary modifications that make it able to overcome the Preetham's model weaknesses stated in (Zotti, Wilkie and Purgathofer 2007).
3. It was built by fitting basis function coefficients to radiance data which were created from a brute force simulation of radiative transfer in the atmosphere using a C++ path tracer implementation
4. It takes into account the effect of the ground albedo

3.1 The path tracer

The Hosek-Wilkie Skylight Model was derived the same way the Preetham was: first devising an analytic formula which would find the relative radiance at any point in the hemisphere and secondly by fitting the parameters of this model to *reference data* (images) which would be provided either by a newly proposed physical model (just like the ones that were presented in the "Physical-based models" section) either by natural images of the sky (but it is practically impossible to gather data for all kinds of weather conditions and solar elevations) or by another method. In the end, the authors decided that it was best to use an extensive, brute-force path tracing algorithm which would create rendering images of sky profiles with different characteristics.

Actually, path tracing is a computer graphics Monte Carlo method of rendering images of 3D scenes such that the global illumination is faithful to reality. Fundamentally, the algorithm is integrating over all the illuminance arriving to a single point on the surface of an object. This illuminance is then reduced by a surface reflectance function (BRDF) to determine how much of it will go towards the viewpoint camera. This integration procedure is repeated for every pixel in the output image: thus, path tracing can be time-consuming and computational expensive as a rendering method, but it can produce images that are indistinguishable from photographs (and so it was ideal for creating reference images for the Hosek-Wilkie Sky Model).

The path tracing algorithm could simulate interaction between light and participating media inside Earth's atmosphere:

1. Interactions of light with the ground are described by standard models of reflection
2. Scattering events within the atmosphere are affected either by air molecules (*Rayleigh scattering*) or by aerosols (*Mie scattering*).

The authors used analytic mathematical formulas for simulating these two types of scattering events as well as a world geometry model that includes a sphere for the Earth (with a Lambertian diffuse surface) and an atmosphere model (another sphere 100km above the sea level) in which the particles' density decays exponentially with altitude. If a light ray of the path tracer "hits" the upper boundary of the atmosphere, is considered to have exited the atmosphere and is tested for collision with the Sun. Now, because the Sun covers only 0.0005% of the total celestial sphere, the previous argument means that a lot of rays will be wasted. In order to prevent this and make the computations faster, the authors implemented a somewhat parallel rendering of images by putting suns at different elevations at the same time (so a ray that hits a specific Sun provides radiance data only for the reference image that has that specific Sun) or by having multiple suns for the same elevation placed in circle around the zenith (exploiting the azimuth's angle symmetry we can find the path that originated from a pixel and ended in a Sun, turn it as many degrees as it needs to match the reference Sun and reverse the path to find the real affected pixel in the image).

Even with this so called "many suns" optimization, in order to create sufficiently decent quality reference sky images, it requires shooting about one million rays per pixel and so calculating a 128x128 image takes somewhere between *40 minutes and 3 hours* (depending on the wavelength, solar elevation, turbidity, and ground albedo) on a 2.66GHz Core i7 920 machine. Since this is a brute force simulation, such computations times are expected and this is the main reason why such models cannot be efficiently used for real time rendering. In Figure 19 we can see some simulation reference sky images (the camera is

placed 10m above the sea level, and uses a circular fisheye projection that is oriented towards the zenith) for different values of sky turbidity:

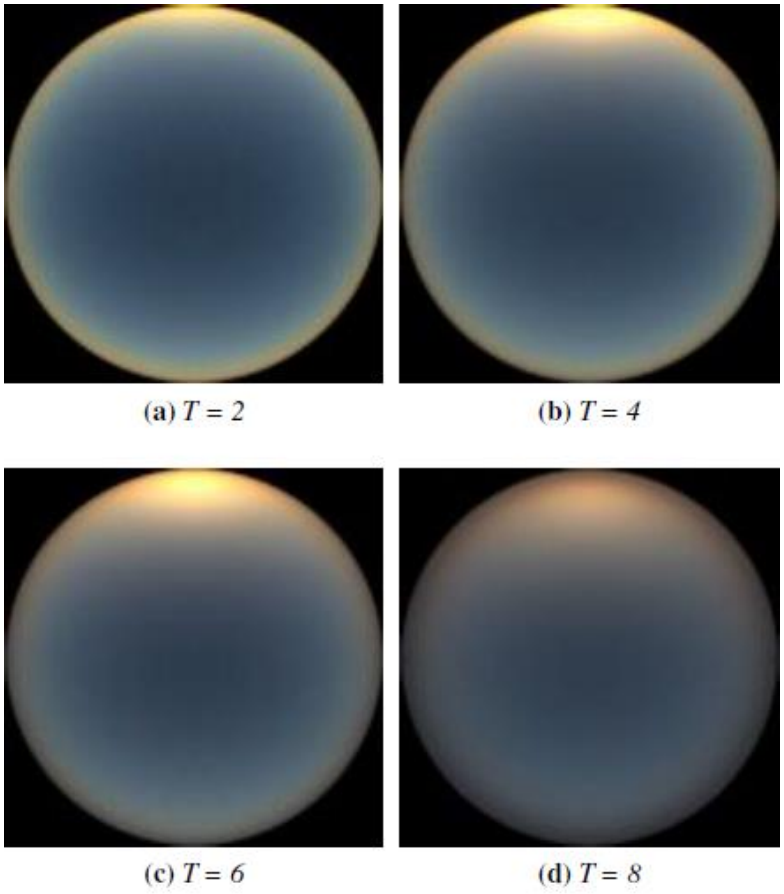
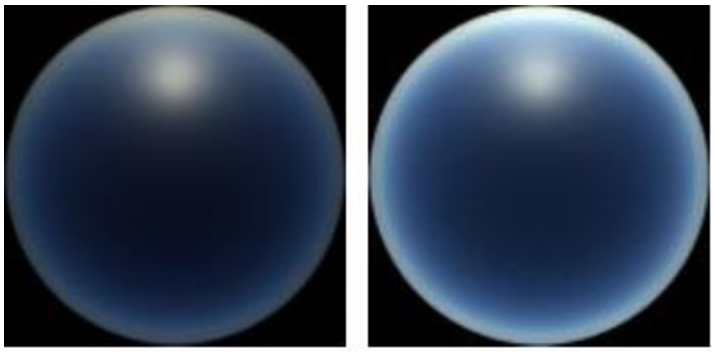


Figure 19: Reference images from the path tracer: solar elevation is 4°

While correcting and improving the Preetham's model weaknesses was the result of the Hosek-Wilkie Skylight Model, it was not the first motive of the authors. They noticed that the Preetham model does not produce nice rendering results regarding *sunset colors*: actually, Preetham's solution was a rough approximation even when compared with the path tracer's reference images (see Figure 8). Also, they were the first to notice the *importance of ground albedo* to the illuminance of the sky dome. High albedo values (close to 1) can occur in winter scenes where the radiation reflected and scattered towards the viewer due to Mie scattering is causing a perceptible change in the brightness of the whole sky dome (the changes are more considerable when high turbidities exist – see Figure 20 for the path tracer results and Figure 22 for the Sky Model's results).



(a) $\alpha = 0.1$ (b) $\alpha = 0.9$

Figure 20: Different path tracer renderings for low and high ground albedo *a*, solar elevation 40° and turbidity 4

The path tracer was programmed to discard rays that didn't undergo any scattering events. In other words, the reference images used for the sky-dome luminance model didn't include rays that directly hit the Sun. In that prospect, the analytic model presented in (Hosek and Wilkie 2012) does not include the energy from these direct hits on the Sun, either. The authors did that because the solar disk creates a very small discontinuous area on the sky dome that can be up to 18 times brighter than its surroundings. The discontinuity and brightness difference would pose a problem for the optimization's stability of the tracer as well as the parametric fitting of the reference images to the radiance distribution mathematical model. That's why they proposed another solution – presented in (Hosek and Wilkie 2013) – in order to include the solar radiance in their model.

3.2 The Sky Model

Using the path tracer, the authors produced the reference dataset used to calculate the parameters of the Hosek-Wilkie Skylight Model. This dataset was generated for a set of turbidities 1 to 10 (higher turbidities would require additional parameters to control), albedo values from 0 to 1 and 11 spectral channels (400nm - 720nm in 40nm increments as well as two in the ultraviolet range - 320nm and 360nm). For each combination of the previous values (T, a, wl), 46 images were generated for solar elevations ranging from 0° to 90°. The authors wanted to have more d for smaller solar elevations (before the sun reached the *15° elevation*) not only because it was their motive to have more reference data for sunset

conditions (and so to make an even better parametric fit to the model), but also because they noticed that after that 15° solar elevation, the sky-dome appearance stabilizes and subsequent changes are minimal.

In order to incorporate the above concept in their method, the reference images were not created for equidistant solar elevations: that means that the model for calculating the solar elevation η was not: $\eta = \left({}^{n}/_{45}\right) * 90°$ (for $n \in \{0,1, \dots ,45\}$ the number of the image), but rather: $\eta = \left({}^{n}/_{45}\right)^{3} * 90°$, a formula which distributes more elevation values when the Sun is near the horizon.

The authors used the equation (1) – see Perez Model – as a base to build their mathematical framework. Perez's formula with the 5 coefficients is basically a single scattering model with Rayleigh phase function and exponential out-scattering. It captures luminance distributions of skies with low turbidities (clear skies) almost flawlessly, providing the A to E parameters are fitted specifically to the setup we want to simulate (e.g. correct fitting to simulate a specific solar elevation and turbidity). The limitations of this formula and subsequently the things that the authors of the Hosek-Wilkie Skylight Model improved upon, can be summarized in these:

1) When the turbidity increases, the contribution of Mie anisotropic scattering becomes more pronounced and produces a phenomenon called *circumsolar ring* (aureole): this cannot be simulated with the Perez or the Preetham model because of the morphology of their formula: a new term needs to be added for that and the authors called it the anisotropic term χ (it uses the same formula for the phase function in Mie scattering simulation in the path tracer).
2) In order to produce a smoother luminance gradient around the zenith at sky profiles with low solar elevations, the authors added the term $I \cdot cos^{1/2}\theta$ (usually with a negative value for I). This term reduces the extent of the aureole[17] around the zenith and the logic behind this is that the higher the viewing angle, the fewer scatterers are in the way to in-scatter light in the viewer's direction.
3) When using Perez's formula for a viewing direction on the horizon ($\theta = 90°$) then the term ${}^{B}/_{cos\theta}$ becomes undefined (division by zero). To tackle this, the authors added a small additional factor on this term (the $+0.01$) which also solves the extremely bright rim around horizon.

Also, the authors didn't incorporate the *dusk conditions* in their formula (negative solar elevations), because they would need to extend the radiance formula even more and substantially complicate the fitting process that we will explain later.

[17] **Aureole** is the luminance halo around the Sun and the horizon, as seen in Figure 8

The general Hosek-Wilkie formula that calculates the relative sky-dome radiance can be seen below. The coordinates system is the same as the one used in the Perez model but we include it anyway since the solar elevation was added in the picture (see Figure 21).

$$F_{Hosek-Wilkie}(\theta,\gamma) = \left(1 + Ae^{\frac{B}{cos\theta+0.01}}\right)(C + De^{E\gamma} + Fcos^2\gamma + G\chi(H,\gamma) + Icos^{1/2}\theta) \quad (5)$$

$$\chi(g,a) = \frac{1+cos^2 a}{(1+g^2-2g\cdot cosa)^{\frac{3}{2}}} \quad (6)$$

$$L_\lambda = F_{Hosek-Wilkie}(\theta,\gamma)\cdot L_{M\lambda} \quad (7)$$

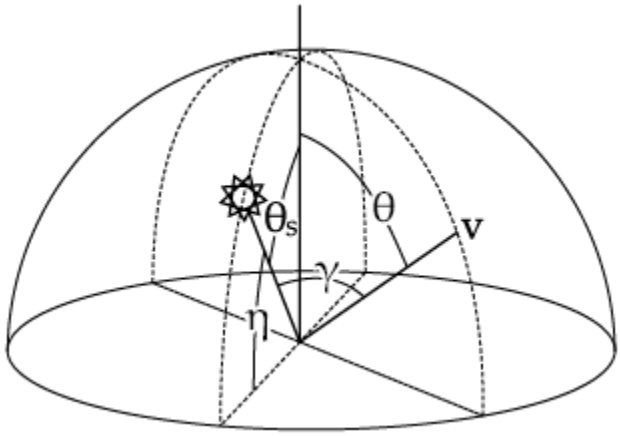


Figure 21: Coordinate System used by the Hosek-Wilkie Skylight Model

The additional C parameter in the formula (5) (instead of 1 that is in Perez's formula – see equation (1)) has to do with the normalization of the radiance: now it is normalized based on the expected value of spectral radiance in a point randomly picked in the upper hemisphere with uniform distribution ($L_{M\lambda}$) and not with regards to the zenith luminance value (as can be seen from equation (2)). This was done because the new formula can produce the luminance spike around the Sun for low solar elevations but cannot do that for the higher ones ($> 85°$) - which is true as to what happens in reality, but it also creates a problem to normalize against the zenith radiance value (since the value would be smaller).

As one can see, formula (7) is wavelength dependent: this means that the computations need to be done each time with different parameters for each band/channel (the price to pay for accurate luminance data and chroma). The model provides three versions: one fully spectral (including the UV range – they are extra wavelengths after all), one that supplies the CIE XYZ values (3 channels) and one that calculates the RGB values (3 channels here also).

3.3 Calculating the radiance parameters

The new model included parameters A to I (9 in total) that have to be analytically calculated using the reference data from the path tracer. The Preetham model calculated the parameters as linear functions of turbidity and the change of solar elevation didn't have any huge effect on the coefficients (which doesn't create realistic sunsets as we have explained). On the contrary, the Hosek-Wilkie model uses *5th order Bezier curves* for calculating these parameters in order to incorporate in the model the non-linear changes of distribution radiance that are produced for different channels when the solar elevation is altered.

So, for each wavelength λ, a 4D-table is created: $M^{\lambda} = \{m_{T,a,p,c}, T \in \{1,\dots,10\}, a \in \{0,1\}, p \in \{1,\dots,9\}, c \in \{1,\dots,6\}\}$, where T is the turbidity, a the ground albedo, p the "id" of the parameters A to I and c is a set of six control points that are used for interpolation in the 5th order Bezier polynomial, using solar elevation η as the interpolation parameter. Also, solar elevation is transformed into an $[0,1]$ interval by using the equation: $x = \sqrt[3]{\eta/\frac{\pi}{2}}$, since the η value was made to include more values at low solar elevations, now by using the cubic root, we spread the x values more evenly across the aforementioned interpolation interval. Then, the following vector is calculated:

$$V_{\lambda} = u_p^{\lambda} = \sum_{i=1}^{6} m_{T,a,p,i}^{\lambda} \cdot f_i \cdot x^{i-1}(1-x)^{6-i} \qquad (8)$$

, with $f_i = \begin{cases} 1\ , i = 1,6 \\ 5\ , i = 2,5 \\ 10, i = 3,4 \end{cases}$

So, the 9-parameter vector V_{λ} has 9 functions with known values (turbidity, albedo, normalized solar elevation) and the 9 parameters A to I as unknowns. The procedure then is to fit this vector for each waveband to the outputs of the reference path tracer using the Levenberg-Marquardt nonlinear least-square fitting method in MATLAB. If someone wants to get radiance data for non-integer turbidities or albedo values in the $(0,1)$ interval, then the model calculates parameter vectors for adjacent turbidities and albedos and then linearly interpolates them (which produces realistic enough results). Also, to calculate the $L_{M\lambda}$ (see equation (7)) the authors use the same procedure described above only that the table is now a 3D one: $R^{\lambda} = \{r_{T,a,c}, T \in \{1,\dots,10\}, a \in \{0,1\}, c \in \{1,\dots,6\}\}$ and the fitting method returns only one number as a result (not a vector).

3.4 Sky-Dome Radiance results

The authors investigated and validated how well the proposed analytic model approximated their reference data. Moreover, they fitted the unmodified Perez formula to the reference dataset to see if their assumptions and changes/intuitions regarding the mathematical formula were indeed able to get simulation results that were closer to reality (that is to the reference data) than Perez's framework (in essence that their model was better). For comparison purposes, they used the logarithmic signal-to-noise ratio (SNR) calculated using the $20 \cdot \log(10)$ rule, which means that larger SNR values correspond to better reproduction of the reference data. Also, for Perez's model, they used specific A-E parameters that fitted specifically for each waveband/turbidity/albedo/solar angle the reference dataset, making the comparison a little unfair for their model. The comparison results can be seen in Table 3:

	New model				Perez model			
	avg	min	T	η	avg	min	T	η
320nm	25.5	18.7	10	$0°12'$	17.4	15.3	5	$0°$
360nm	26.2	19.8	6	$0°1'$	16.1	14.7	2	$1°42'$
400nm	25.5	20.4	3	$0°1'$	16.9	13.4	4	$2°42'$
440nm	24.1	18.4	10	$7°54'$	17.4	12.4	8	$0°$
480nm	22.5	15.2	10	$7°54'$	17.5	12.8	8	$0°12'$
520nm	21.0	13.0	10	$7°54'$	17.4	12.2	9	$5°45'$
560nm	19.5	11.3	10	$5°45'$	17.4	10.4	9	$5°45'$
600nm	18.3	10.0	10	$5°45'$	17.0	8.5	10	$4°2'$
640nm	17.2	9.1	10	$5°45'$	16.3	7.7	10	$4°2'$
680nm	16.3	8.2	10	$4°2'$	16.1	7.0	10	$4°2'$
720nm	15.4	7.6	10	$4°2'$	15.8	6.7	10	$4°2'$
CIE X	20.0	11.2	10	$5°45'$	18.0	11.1	10	$5°45'$
CIE Y	20.0	11.5	10	$5°45'$	17.9	11.5	10	$5°45'$
CIE Z	24.0	17.1	10	$7°54'$	18.1	17.0	10	$7°54'$

Table 3: Comparison between the Hosek-Wilkie and Perez's model for each spectral waveband and different Turbidities

One can observe that the SNR values for the Hosek-Wilkie model are larger than Perez's. The only waveband for which the original formula slightly outperforms the modified version used in the new model is 720nm, but the advantage is not significant (also with zero albedo the superiority of the Hosek-Wilkie

model is clearly apparent). Also, one can notice that the worst SNRs (for both models) happen for higher turbidities and sunset conditions: this is when the performance of the model degrades.

Some simulation results can be seen in Figure 22, where the subtle but noticeable effects of ground albedo on the sky over a reconstructed archaeological site are apparent and in Figure 23, where we notice the nice twilight color of the sunrise in a high turbidity setup and the dark rim that appears around the horizon.

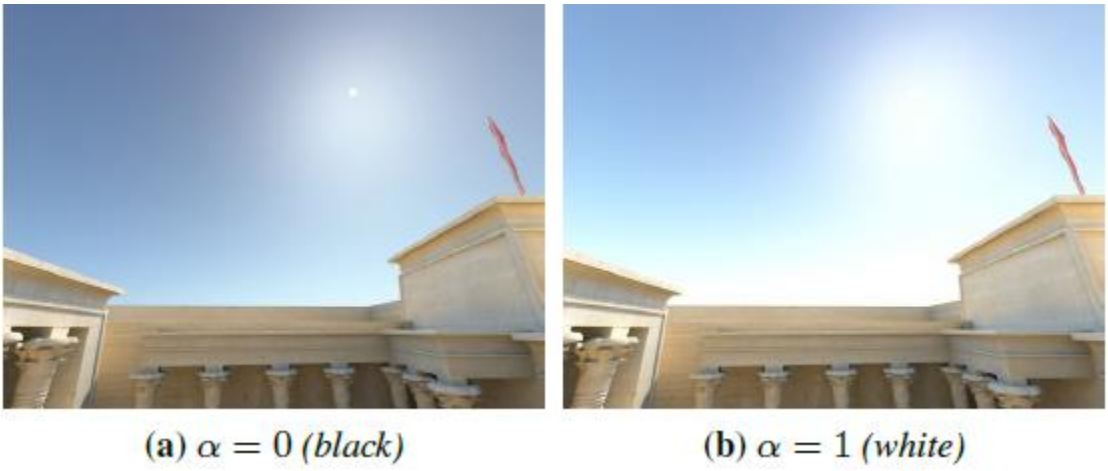


Figure 22: The influence of ground albedo on sky appearance for turbidity 5

Figure 23: A hazy sunrise with turbidity 9. Left is a real photograph and right is the Hosek-Wilkie simulation

3.5 Adding a Solar-Radiance Function

While the Hosek-Wilkie Skylight Model produces genuine spectral data for the entire range from the near ultraviolet through the visible spectrum and it reliably supports turbidities between 1 and 10 as well as the important effect of ground albedo, it does not include the solar disk: the energy from the Sun is not included in the path tracer simulation since every ray that does not scatter at all, is discarded. So, to completely describe an outdoor scene's radiance environment, the authors proposed an enhancement to the Hosek-Wilkie model in (Hosek and Wilkie 2013). The basic idea in this work was that any object's appearance can be perceived from an arbitrary observer based on this equation:

$$L = L_0 \cdot \tau + L_{in} \tag{9}$$

, where L is the intensity of the light that reaches the observer, L_0 is the intensity of the light emitted and reflected at the power source, τ is the participating media's transmittance (the percentage of the L_0 that actually reaches the observer), and L_{in} is the in-scattered light's intensity on the sight-path between the object and the observer. What the previous version of the model calculated analytically was the L_{in}, and it is what plays the most important role since most times we don't look straight in the Sun (and L_0 is zero because no rays come from the darkness of the space): the other times (when we want to have the Sun also in the "picture" – render it also) the term $L_0 \cdot \tau$ represents the solar radiance function that the new model has to calculate.

The atmospheric transmittance τ is independent of the solar elevation, the ground albedo and the viewing direction: only the weather conditions (turbidity) can affect it. The authors used this specific technique to evaluate the transmittance for different turbidities: they used the path tracer and casted rays in one direction and after the simulation was over, they calculated the ratio of rays that weren't scattered before they escaped the atmosphere (the ratio of the rays that went straight ahead to space): this is a very good approximation of the transmittance τ.

The only thing that is missing now is the L_0 term. To generate accurate renderings of the Sun, the authors didn't just take the L_0 tabulated data from NASA: they made extra calculations to simulate the *limb darkening* effect of the Sun: this phenomenon is an optical effect seen in stars (including of course our Sun) where the center part of the disk appears brighter than the edge or "limb" of the star. This drop in light intensity towards the edge of the solar disk is caused because the light which is emitted from the Sun's surface must travel larger distances when it originates from the edges (a distance almost parallel to the Sun's surface in its atmosphere) than from the center.

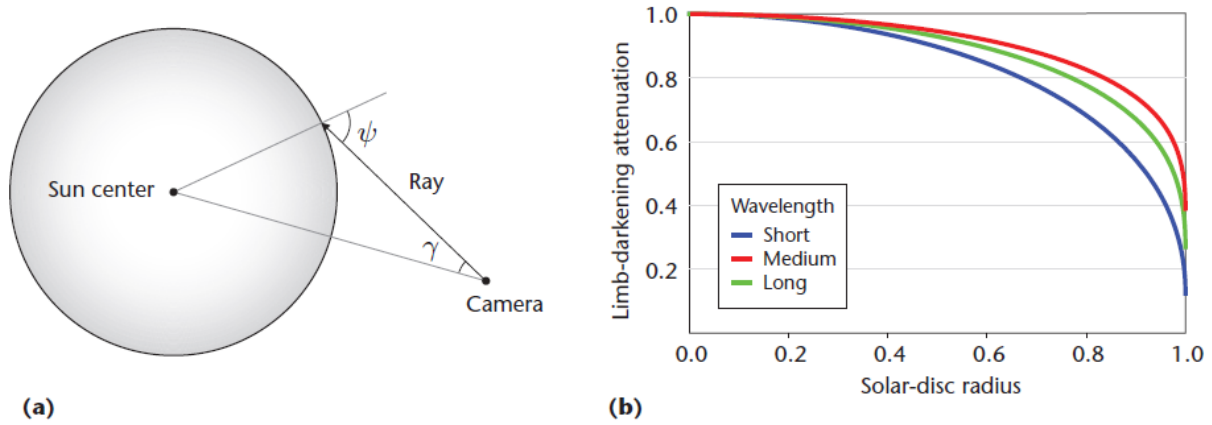


Figure 24: (a) The limb darkening effect. (b) The attenuation for varying wavelengths across the solar disc's radius

The "geometry" of the limb darkening effect can be seen in Figure 24a. Based on that model, astronomers use an approximation formula to find the radiant intensity $I(\psi)$ at a point on the solar-disk, based on the radiant intensity at the center, $I(0)$:

$$\frac{I(\psi)}{I(0)} = \sum_{k=0}^{N} a_k cos^k(\psi) \quad (10)$$

, where $\cos(\psi) = \sqrt{1 - \frac{sin^2\gamma}{sin^2 0.25°}}$ (11)

The a_k coefficients are tabulated for every wavelength available in the spectrum and the authors interpolated the ones necessary in order to get the values they needed for the exact wavebands they use in their model. The 0.25° angle is the *Sun's angular radius* as seen from earth (we will discuss analytically this in the Sky Model Implementation section).

Also, in Figure 24b, we can see that the light attenuation is wavelength-dependent: for a solar-disk radius of 1 (the outer-rim of the solar disk) blue light is attenuated twice as much as red light which leads to the solar rim being redder than the center. Overall, with the inclusion of the limb darkening effect, the Hosek-Wilkie model can generate nice and realistic images of the Sun – see Figure 25 for some sunsets with different turbidities and solar elevations.

Figure 25: Direct Images of the solar disk: left is $(T, \eta) = (1, 0.5)$, center is $(T, \eta) = (6, 0.5)$, **right is** $(T, \eta) = (7, 0.25)$**, where T is the turbidity and η is the solar elevation**

For calculating the term $L_0 \cdot \tau$ from equation (9), the authors used the same technique as the one analyzed in the section "Calculating the radiance parameters" for the L_{in} term: interpolating a cubic (3rd order) Hermite polynomial function based on solar elevation segments x_i where again more points are calculated during the sunset conditions:

$$L_0 \cdot \tau \approx r_{\lambda,T}(\eta) = \sum_{k=0}^{3} m_{\lambda,T,i,k} \cdot (\eta - x_{i-1})^k \quad (12)$$

, where $\eta \in (x_{i-1}, x_i), i = 0,1, \dots, 45$, $x_i = \frac{\pi}{2} \cdot \sqrt[3]{\frac{i}{45}}$ and the $m_{\lambda,T,i,k}$ is the tabulated value we get after we match the approximation value above with the real value $L_0 \cdot \tau$ from NASA. Also, to get a limb-darkening spectral-radiance value, the authors multiply the result of equation (12) with the results from (10) that describe the limb darkening effect.

In Figure 26, we can see the results from the above formula as well as NASA's tabulated values for the Sun's spectral radiance (for turbidity = 4). We notice that the wavebands closer to the red color (right values on the graph) get larger radiance values than the ones that represent the blue color (left values on the graph), which is normal since we describe the Sun's radiance color on different wavelengths. Also, we also notice the expected reduction in direct solar radiance for decreasing solar elevations. Lastly, no value calculated can surpass the NASA's "Sun" labeled value in Figure 26 (which is the solar radiation that hits the top of the atmosphere based on astronomical observations) since the Hosek-Wilkie solar radiance function takes also into account the limb darkening effect.

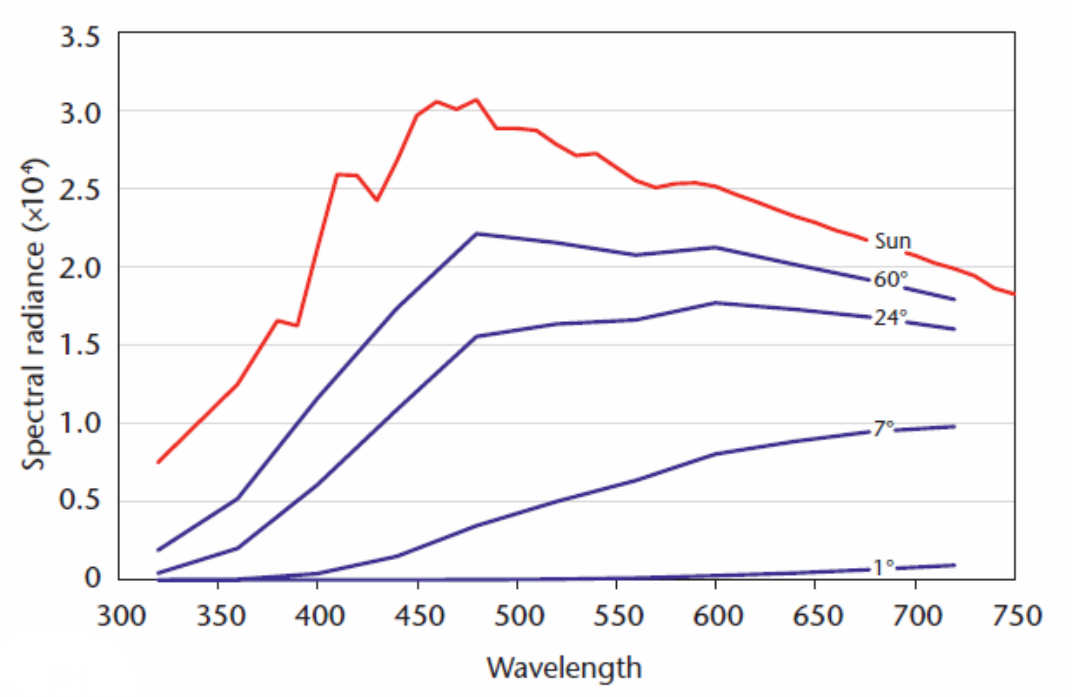


Figure 26: Direct solar radiance for turbidity 4 and different solar elevations for the whole spectrum. The line labeled "Sun" represents the L_o NASA term

Based on these results, the authors warn to take care when determining the solar disc's overall radiant power. In practice, they averaged a sufficiently large number of random samples on the disc to get reasonable results. This happens because of the limb darkening phenomenon and the fact that at lower solar elevations the attenuation of sunlight is modeled by a non-linear progression. So, using just one sample at the solar disc's center yields significantly wrong results. The authors estimated that on average, across wavebands, limb darkening by itself induces an error of approximately 23 percent.

Finally, some results incorporating both the Hosek-Wilkie Skylight Model and the solar-radiance function (we call it the "*complete"* model) can be seen in Figure 27 and Figure 28: in Figure 27 the results from adjusting the proposed model for different solar elevations can be seen as well as the effect on the objects on the scene and in Figure 28, the authors illustrate how changing the value of turbidity affects the ratio between direct sunlight and sky-dome radiance.

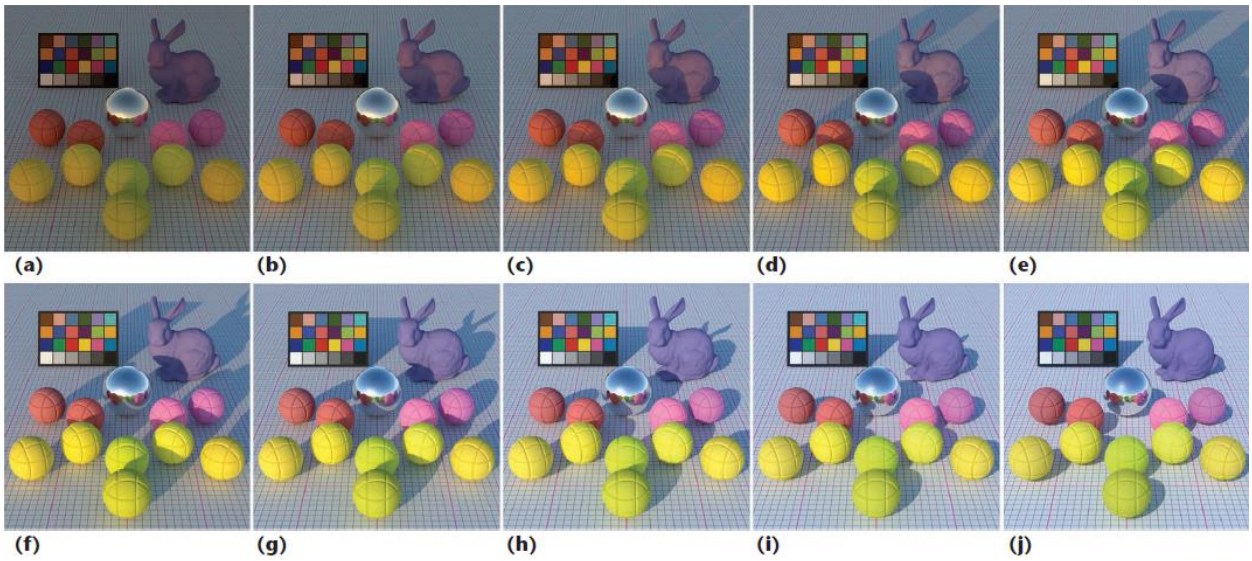


Figure 27: Results of the complete Hosek-Wilkie Skylight Model for turbidity 3 and solar elevations (from left to right): η = 1°, 2°, 3°, 6°, 9°, 12°, 18°, 24°, 32°, 42°

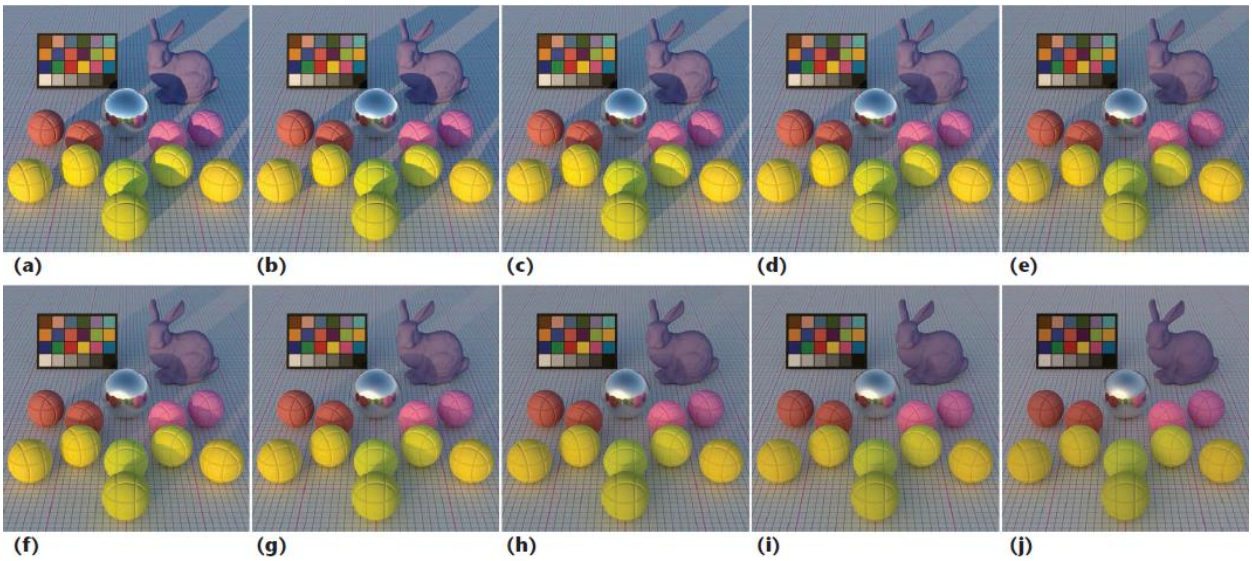


Figure 28: Results of the complete Hosek-Wilkie Skylight Model for solar elevation η = 8° and turbidity values (from left to right): T = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

4 Sky Model Implementation

4.1 General Description

Following the presentation of the Hosek-Wilkie Skylight Model in the previous section, we will now describe analytically how we managed to build our Web application based on the results of the aforementioned model. All this discussion will be centered on the frontend, which is the Web GUI interface the users see and interact with, as well as its backend part. The concept of the application is summarized in these points:

- The users can enter three numbers: Turbidity, ground albedo and solar elevation
- The user can choose which 3D-model will be present in the scene by selecting one from a dropdown list (or even none at all)
- When pressing a specific button, a realistic sky-dome sample is created on a canvas element taking data from the Hosek-Wilkie Skylight Model implementation which is provided from the authors (the backend part)
- Except from the sky-dome, the scene will include a terrain, the Sun, a 3D-model and of course the lights that represent light coming from the Sun and the sky-dome
- The user is able to move inside the scene using the mouse and specific keyboard buttons
- An extra feature is provided when pressing the "demo button" which uses only the values of turbidity and ground albedo and starts a sunrise sequence with solar elevation from 1° to 90° (adding 0.5° every 1 sec)

Regarding the technologies and languages used to build this Web app as well as their roles inside the application, we categorize them in:

- Technologies used in the *frontend* part of the application:
  - **HTML5+ CSS3**: structure of the GUI, make the buttons look fancier, etc.
  - **JavaScript**: taking the values the users enter in the GUI, executing value checking functions (e.g. turbidity ranges from 1 to 10), executing the canvas rendering when pressing a button, etc.
  - **AJAX**: sending the appropriate request to the server/backend part and receiving the response

- **Three.js**: a lightweight cross-browser JavaScript library/API which is used to create and display the animated 3D scene we designed, inside a canvas Web Browser element. Internally, Three.js uses the WebGL standard, but its API is easier to use even for someone who has only a vague idea of computer graphics.

- Technologies used in the *backend* part of the application:
  - **C**: this language is used not only to take the results of the Hosek-Wilkie model but to generate the appropriate geometry and color values that will be used by Three.js to create the sky dome and the lights.
  - **PHP**: this server-side language is used as an intermediate between the AJAX request and the C backend part because ANCI C is not a widely-used server-side language by itself and cannot be easily employed as a receiver of an AJAX request.

4.2 Application Design concepts and Architecture

Regarding the implementation itself, the 2 primary issues that we faced were firstly what technology to use in order to draw inside the canvas element (chose Three.js instead of WebGL for its simplicity and ease of use) and secondly how to get the results we wanted from the Hosek-Wilkie sky-model library. For the latter, we were faced with two options: either re-build the author's model in JavaScript from scratch (which would make our entire application a client-side matter only) or use PHP as an intermediate programming entity whose only work would be to transfer the requests from the frontend (GUI) to the backend (C program) and the responses the other way around.

After making some tests, we managed to port the Hosek-Wilkie C implementation to JavaScript (using emscripten[18]) but the total execution time for getting 3 radiance spectral values (corresponding to RGB values) of a single point in the sky-dome (for fixed solar-elevation, ground albedo and turbidity) was 70 times larger than the corresponding implementation in ANSI C (0.07 sec (JavaScript) as opposed to 0.001 sec (in C)). So, based also on the fact that the authors of the Hosek-Wilkie model implemented their model in C and it would be advisable to also implement our solution for the dome geometry and color values on the same language, we decided to go with the PHP-based intermediate solution.

[18] **Emscripten** is an LLVM-based project that compiles C and C++ into highly-optimizable JavaScript in asm.js format (see http://kripken.github.io/emscripten-site/)

An architectural component diagram showing all the core elements of our Web application can be seen in Figure 29: the canvas element is where the animate function generated by Three.js is drawing the scene, but to do so, the user has to use the GUI to provide the desirable input parameters and activate the entire process with a specific button. Furthermore, before the animation takes place, it needs the geometry and color data for the dome and the lights of the scene which is provided by an AJAX request, which is taken by a simple PHP program (PHP executer), which in turn takes the necessary input from the request (turbidity, solar elevation and albedo), drives it for execution/calculation by the C program and finally takes the results back to the AJAX call, in order to continue with the creation of the scene.

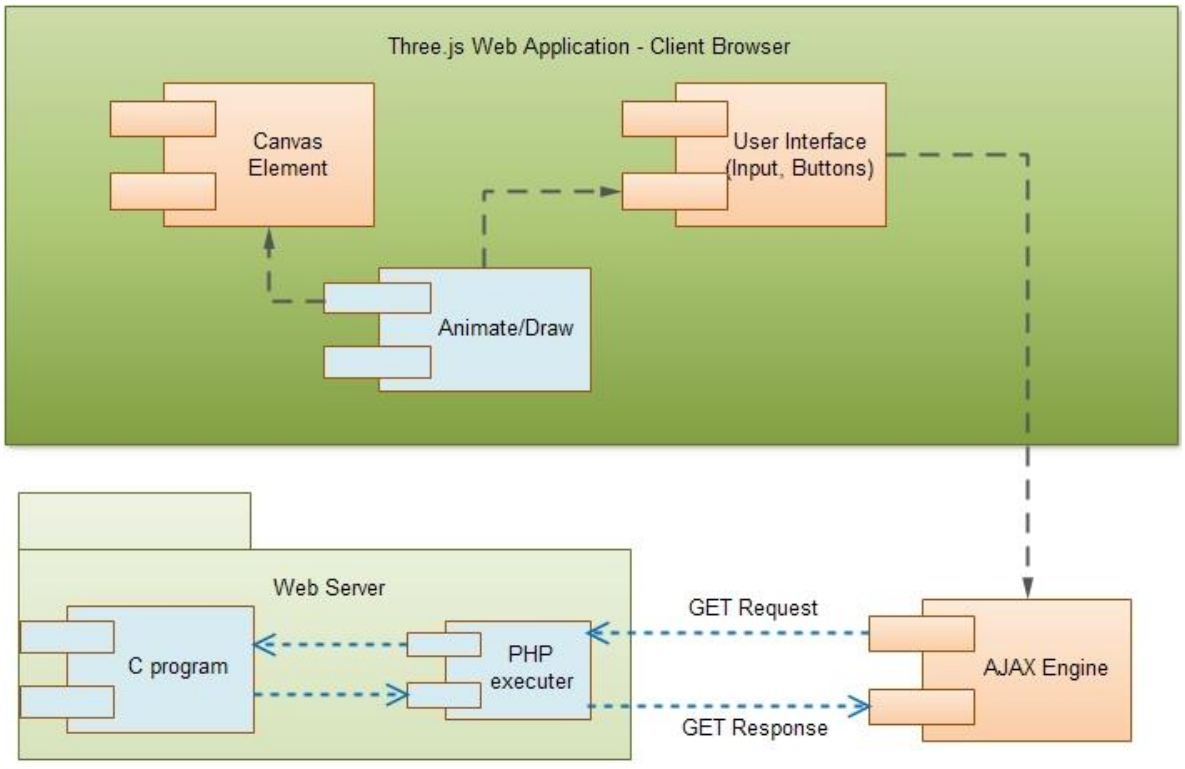


Figure 29: The Web's application architectural diagram

From the above diagram one can already see the *bottleneck* of our application: if the drawing waits for the AJAX call to finish for too long, the canvas element will be stuck. A situation such as this can be caused by a low-spec CPU or even a network-problem (a "slow server" for example). Also, in the "demo playing" feature of our application the same logic is used as the one described in the above diagram: a different AJAX call every 1 sec for a different solar elevation. So, if the calculation of the geometry and

colors of the sky dome is taking more than 1 sec for some solar elevation values, the outcome for that specific value is that the animation will not be created at all! Although, even if the previous sample is not complete before the 1 second interval, the new sample can be drawn on its time, given that the AJAX call (being asynchronous and all) is unobstructedly requesting for the sample of the previous solar elevation + 0.5° and that the total calculation time (fetching the request, computation, returning the response) is under 1 second.

4.3 Web GUI

In Figure 30, we can see a snapshot of our Web application's GUI. It consists of 3 fill-in boxes which include the values for solar elevation, turbidity and the ground albedo. We also check whether the value inserted by the user is within the limits imposed by the Hosek-Wilkie model: that is solar elevation ranges from 0° to 90° degrees, turbidity can be from 1 to 10 and ground albedo from 0 to 1. Also, an absence of any value is also considered wrong input and it is not allowed. Next is the 3D model drop-down list which provides a model to render at the center of the scene (in Figure 30 it's the white sphere). We have selected of a variety of 3D models which we will discuss more analytically in the "Three.js scene" section.

Figure 30: The Web application's GUI

The GUI has also 3 buttons: the first, called "Generate a Sky Dome Sample" does exactly what its name suggests. The other two are demo-specific and are used (the first one) to start a demo sunrise sequence and (the second) to stop that sequence. Also, a user cannot generate a new sky dome sample (while changing or not the turbidity or the ground albedo value) or change the 3D model inside the scene while a demo is playing: first you have to stop it. The right part of the GUI's screen is a Canvas Web Browser element, inside which the actual rendering is taking place. In the snapshot of Figure 30, we can see the terrain, the Sun, the white sphere (3D model) and of course the sky dome, with a haze around the early morning Sun which is a natural characteristic of the high turbidity sky setup such as this.

4.4 Using the Hosek-Wilkie model

In this section, we will describe the components of the backend part (server-side) of our application (the functionality of the *dome_points.c* program).

4.4.1 The Hosek-Wilkie model's interface

The authors of the (Hosek and Wilkie 2012) provided an easy to use C implementation of their model and in their next work (Hosek and Wilkie 2013), they updated their model in order to include the solar radiance function which was described in the section "Adding a Solar-Radiance Function". This implementation provides functions the can be used to get the radiance value of a specific spectral or not channel (can be RGB or CIE's XYZ) given the appropriate parameters. The complete solution, which also adds the solar-radiance function from equation (9), is only implemented for spectral data. So, in our solution we decided to use:

1. The spectral sky dome radiance function for calculating the colors of the points in the sky dome
2. The complete spectral radiance function for calculating the colors of the lights that represent the sky dome
3. The solar radiance to represent the light coming from the Sun (which results from subtracting the sky dome radiance from the "complete" solution)

Since our solution will be used to render a computer-generated screen (which also means that the Three.js will take 3 values for coloring – the RGB values – each one from 0 to 1), we take the result from each specific radiance function mentioned above, *3* times: one for each waveband that represents the RGB values. The wavelengths that were used are these:

- Red wavelength = 685 nm
- Green wavelength = 533 nm
- Blue wavelength = 473 nm

For the calculation of the radiance, we firstly build for each channel a *skymode_state* structure which has all the pre-calculated data that the authors of the Hosek-Wilkie model gathered from the reference dataset, based on the values of turbidity, solar elevation and ground albedo that the user has fill in on the GUI:

```
#define num_channels 3    // RGB
for (unsigned int i = 0; i < num_channels; i++) {
      skymodel_state[i] = arhosekskymodelstate_alloc_init (solarElevationRad, turbidity, albedo);
}
```

Then, for a point on the sky dome, for which we know the θ and γ values (see Figure 21) – which tell us *where* is this point in the sky in relation to the Sun's position – we can calculate the sky-dome radiance or the complete radiance value with these functions:

```
// skymodelStateForChannel different for each channel/wavelength
arhosekskymodel_radiance (skymodelStateForChannel, θ, γ, wavelength)
arhosekskymodel_solar_radiance (skymodelStateForChannel, θ, γ, wavelength)
```

Note that a subtraction operation between the above two values will give the solar radiance which will be used only to model the color of the sunlight in our implementation.

Now, based on the above provided interface (mainly the *arhosekskymodel_radiance* function), what is missing for the calculation of the sky dome radiance, is the calculation of the θ and γ values: this also accounts for how many points in the sky dome we will need and generally the geometry and the exact model that we will use in order to build the sky dome in our scene.

4.4.2 Building the sky-dome

To build the sky-dome, we use as a reference the hemisphere sky setup that is presented in Figure 31. Based on this setup, because we can't find the radiance value of every single point in the hemisphere (computationally impossible) we approximate the computation needed by taking only a fair sample of specific points placed in a $(\theta\varphi)$ grid, where $\theta \in [0, \pi]$ and $\varphi \in [0,2\pi)$ are the known angles from the sphere coordinates. By firstly predefining 3 values: the R value (radius of the hemisphere) and the θ_{step} and φ_{step} in degrees – as seen in Figure 31 – we can find the coordinates of any sky dome point v, given that we start from a reference, which for our setup is the point $p_{ref} = (-R, 0,0)$.

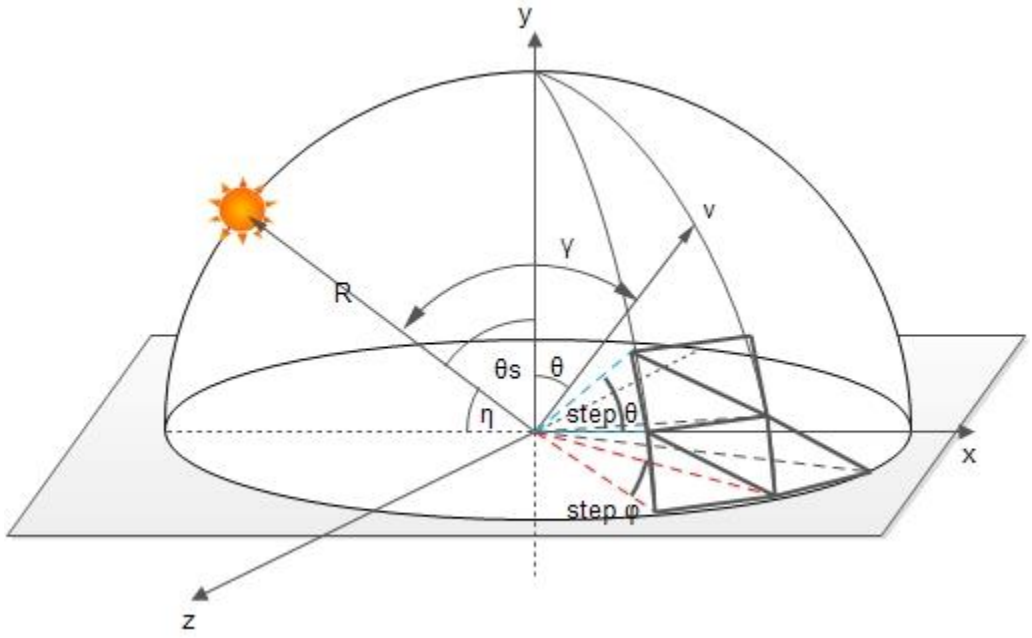


Figure 31: The main concept of the implementation of the Sky Dome

Also, by knowing the Sun's position (which can be easily found when you know the solar elevation η, the radius R and the fact that the Sun will move on the $(-x, y)$ quarter-plane only - as is in our setup in Figure 31), we can calculate the γ angle between the Sun and any given point v. The θ angle is easy to find since it can be directly derived from the number of θ-steps you have used to reach the point v. So, with the θ and γ angles already accounted for every point v, we can use the *arhosekskymodel_radiance* function described in the previous section to calculate the sky dome radiance of these points. In our implementation, we used: $R = 1000, \theta_{step} = \varphi_{step} = 5°$, so this setup accounted for a total of 1225 points in the hemisphere (the largest θ_{step} can be such that $\theta = 10°$ - the zenith point though is included).

Of course, in our implementation someone can change the values of R, θ_{step} or φ_{step}: smaller values for the angle's steps will calculate more dome points and by doing so, the approximation will be better for the sky dome appearance (the caveat here is that more points require more processing time). Nonetheless, after we tested for a while, we found out that the values we chose are a good compromise between calculation speed and sky-dome appearance (it indeed looks like a hemisphere with 1225 points).

So, up until now, our solution provides a series of values representing the (x, y, z) coordinates of the sky dome points, as well as their colors - triplets (R, G, B) which are the result of using the aforementioned sky-dome radiance function for each point. To complete the calculation regarding the sky-dome, we need to think about how the Three.js will represent these values. Just drawing these 1225 points with their respective colors wouldn't create a sky-dome but more like a dotted atmosphere (with a lot of black!). What we need to do is to refine our calculations in order to triangulate the hemisphere as in Figure 31. Actually, what our algorithm does is that it produces all the *triangle indexes*: this means that if we index all the 1225 produced points from 0 to 1224, then all the sky-dome can be covered in triangles, each of which can be defined by its three surrounding points. For a simple triangulation of the sky dome such as the one seen in Figure 32, where $\theta_{step} = 45°$ and $\varphi_{step} = 90°$, we can see the points that are generated (in that specific order) from our algorithm as well as the triangles that will be created by Three.js.

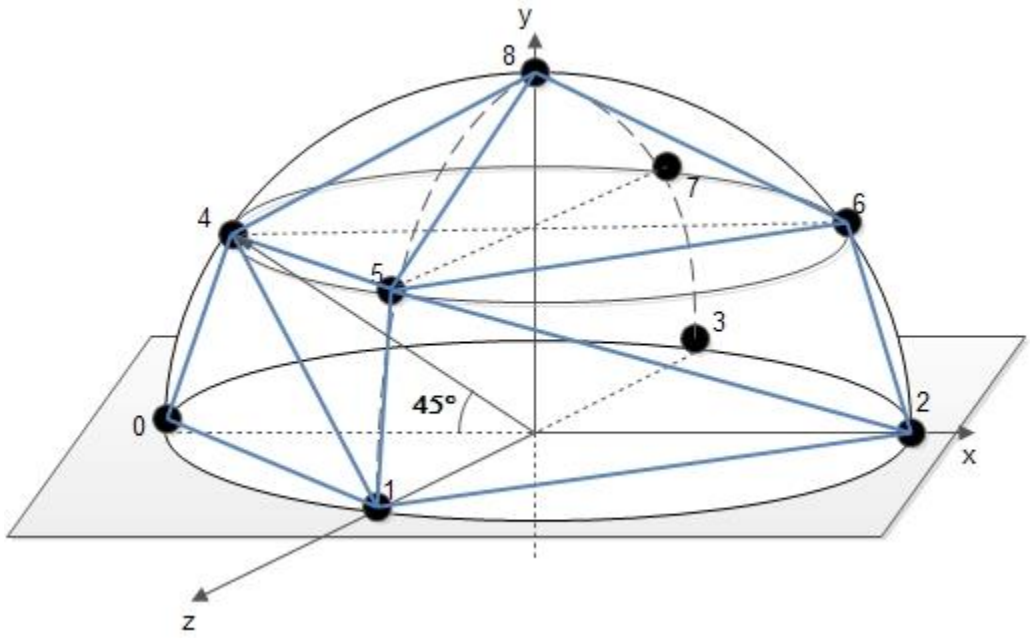


Figure 32: Triangulation of the Sky Dome with θ_{step}=45° and φ_{step}=90°

Essentially, what our algorithm does is to also provide (apart from the coordinates and colors of the 9 points in Figure 32) the triangles indexes which in this case are the 12 triplets (12 is the number of the triangles necessary to "complete" the dome): (0,1,4), (1,2,5), (2,3,6), (3,0,7), (4,5,1), (4,5,8), (5,6,2), (5,6,8), (6,7,3), (6,7,8), (7,4,0), (7,4,8). These 12 triangles will generate a polygonised dome and that's why by choosing enough dome points, we can make the dome surface smoother and more curved. Also, Three.js interpolates the colors of the 3 points of one triangle in order to fill the chroma of the underlying surface. Lastly, the same logic applies for any values of θ_{step} and φ_{step}, which are the values that define the number of the dome points and subsequently the number of triangles that are used to render the sky dome.

4.4.3 Lights

The sky dome is just a huge triangulated mesh structure in our scene defined by points and their colors. To illuminate our scene (that is the terrain and the specified 3D model) we must establish some light sources (as discussed in the section "The Hosek-Wilkie model's interface" we will mainly use the *arhosekskymodel_solar_radiance* function which returns the spectral values for RGB coloring and which will be the colors of the lights that the Three.js will use). Apart from where we are going to put these light sources, we should think about what these lights will represent. There are two sources of illumination under the sky-dome:

1) Light coming from the sky dome itself
2) Light coming from the Sun

About the first point (1), the problem that must be solved is finding a way to represent the full sky-dome luminosity which mathematically can be given from the formula:

$$L_{SkyDome}(T,\alpha,\eta) = \iint_{\Omega} L d\omega = \int_{\varphi=0}^{2\pi}\int_{\theta=0}^{\frac{\pi}{2}} L(\theta,\gamma,T,\alpha,\eta)d\theta d\varphi \qquad (13)$$

, where T is the turbidity, α is the albedo value and η is the solar elevation (see Figure 31). We approximate the equation (13) by representing the sky-dome's luminance with the sum of luminance L_i produced by $N = 5$ light sources explicitly placed as in Figure 33 (angle elevation is 35°):

$$L_{SkyDome}(T,\alpha,\eta) = \iint_{\Omega} L d\omega \approx \frac{2\pi}{N}\sum_{i=1}^{N=5} L_i \qquad (14)$$

The 2π factor in the formula (14) is the solid angle[19] of the hemisphere (4π is for the complete sphere). So, each subsequent light source accounts for a different part of the hemisphere that is "seen" by a solid angle of $\frac{2\pi}{N}$. So, we changed the calculation of the L_{in} term in equation (9) on the Hosek-Wilkie implementation to fit our model:

$$L_{in}^{new} = \frac{2\pi}{N} * L_{in} \tag{15}$$

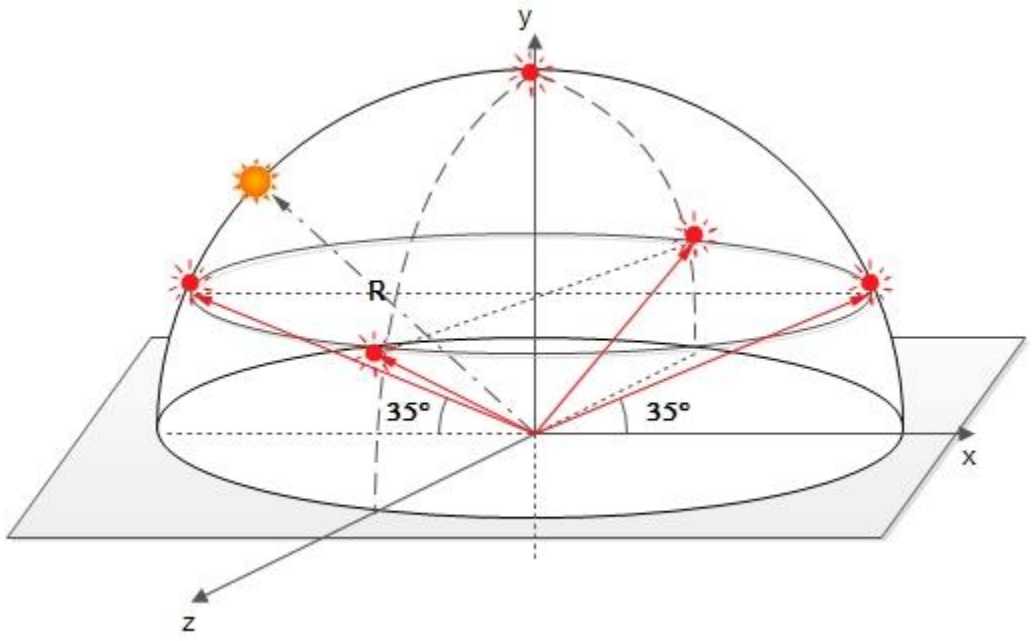


Figure 33: Positions of the light sources that represent the sky-dome light

About the second point (2), the *arhosekskymodel_solar_radiance* function of the Hosek-Wilkie Skylight model which delivers the complete radiance (sky + Sun – including limb darkening) had this particular problem: the part which calculated the solar radiance (the $L_0 \cdot \tau$ term in equation (9)) was producing results with absolute values orders of magnitude larger than the part which calculated the L_{in}^{new} term. This happened because these values represent the solar radiance as seen in Figure 26, which are produced from the path tracer that the authors of the Hosek-Wilkie model used. To bring the solar radiance to the levels of the sky dome radiance values we need to find the *ratio* of the Sun's surface to the surface of the sky dome (we will call it *sunToDomeSurfaceRatio*) and multiply it with the $L_0 \cdot \tau$ term: this will

[19] **Solid angle** is a geometry term that describes a 2D angle (in 3D space) that an object subtends at a point.

bring balance to the two different terms in equation (9) and produce meaningful results. Based on the geometry presented in Figure 34, we have that the angular diameter[20] of the Sun is computed as follows:

$$\delta = 2 \cdot arctan\left({}^{r}/_{R}\right) \Rightarrow r = R \cdot \tan(\delta/2) \quad (16)$$

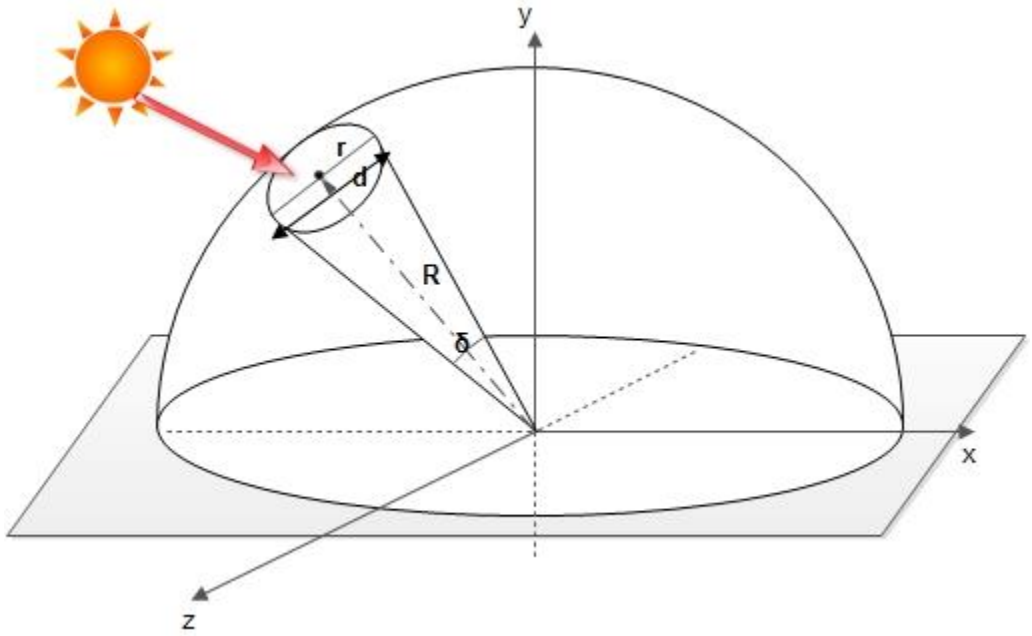


Figure 34: Geometry for computing the angular diameter of the Sun and the *sunToDomeSurfaceRatio* metric

It is also known that[21]: $\delta_{Sun} = 32' = 0.5357°$. The *sunToDomeSurfaceRatio* can thus be calculated as:

$$sunToDomeSurfaceRatio = \frac{Surface_{Sun}}{Surface_{SkyDome}} = \frac{\pi r^2}{2\pi R^2} \overset{(16)}{\Longrightarrow} \frac{\pi R^2 tan^2(\delta_{Sun}/2)}{2\pi R^2} = \frac{tan^2(\delta_{Sun}/2)}{2} \Leftrightarrow$$

$$sunToDomeSurfaceRatio = tan^2(0.26785)/2 = 1.0927 \cdot 10^{-5} \quad (17)$$

The small value of the above result is a correct indication that our procedure is correct. Consequently, we observed that the values between the two terms in equation (9) became balanced (same

[20] **Angular diameter** is an angular measurement describing how large a sphere or circle appears from a given point of view: it can be thought of as the angle an eye or camera must rotate to look from one side of an apparent circle to the opposite side.

[21] $1'$(arcminute) $= 1/60$ degrees

order of magnitude). Adapting the second term which computes the solar radiance (and subsequently completes that calculation performed by the *arhosekskymodel_solar_radiance* function), we have:

$$(L_0 \cdot \tau)^{new} = sunToDomeSurfaceRatio * (L_0 \cdot \tau) \quad (18)$$

So, to summarize, we calculate:

- The colors for the 5 light sources (placed as in Figure 33) using the *arhosekskymodel_solar_radiance* function which is based on an adjusted formula (9): $L^{new} = L_{in}^{new} + (L_0 \cdot \tau)^{new}$.
- The color of the Sunlight using both the *arhosekskymodel_radiance* as well as the *arhosekskymodel_solar_radiance* (since we cannot get the solar radiance with one function only) by using the formula: $(L_0 \cdot \tau)^{new} = L^{new} - L_{in}^{new}$.

4.5 Three.js scene

In this section, we will describe all the three.js functions used in our application to build the elements of our scene. The three.js library is included in a compact file called *three.min.js*, which is included in our html index file, called *skydome.html*. The version of the library that was used was the latest one at that time, revision 71. What we will describe in the following paragraphs (all the three.js-related stuff) is code written inside the *skylightdome.js* script and executed when the function *drawDomeOnCanvas* is called.

4.5.1 Color mapping

The first thing that the *drawDomeOnCanvas* function does (after it gets the result from the AJAX response – that is position + colors of the dome points and the position + colors of all light sources including the Sun) is to perform a *color mapping* for the colors of the sky dome as well as the RGB spectral colors of the Sun. This is essential because as we discussed in previous sections, the Hosek-Wilkie model did not support the calculation of the complete radiance (sky + Sun) in the RGB format but only in spectral. That's why we used only 3 channels/wavelengths that represent the 3 colors of the RGB schema: Red, Green and Blue. This is a good approximation since the alternative would be (as other researchers have done) to first

convert the spectral color values to CIE XYZ space using the standard XYZ primaries and then to convert the previous result to sRGB space by multiplication with an $XYZ \rightarrow sRGB$ conversion matrix.

So, without the use of a good color mapping, the result of the sky dome for a turbidity equal to 1 (clear blue sky) was dark blue (values < 0.01). Also, the haze around the Sun for high turbidity setups was too large and was having an abnormal effect on that part of the hemisphere (values > 1). Also, for solar elevations smaller than 5°, the Sun's light colors (which could be seen on the white 3D sphere) were not the expected ones. The color tuning we performed was a simple normalization based on the larger color value for the dome points and for the Sun's RGB colors we applied the same technique while adding a necessary term to keep the light values from becoming 1 mainly on low solar elevations setups. The code snippets can be seen below:

```
// color mapping/toning for the Sky Dome (make the results more bright)
var maxim_dome = getMaxOfArray(colors_points);
for (var i = 0; i < dome_points_length; i++) {
        colors_points[i] = colors_points[i] / maxim_dome;
}

if (solar_elevation < 2.0) {
    sun_color[i] = sun_color[i] / (sun_maxim + 0.0005);
} else {
    sun_color[i] = sun_color[i] / sun_maxim;
}
```

4.5.2 Camera movement

The starting view of the camera is defined as a perspective one with a frustum vertical field of view of 65° and position $(posX, posY, posZ) = (0, 5, 30)$. The maximum far plane that can be seen is 2000 (since the dome radius is defined as 1000, this value should be enough). The parameter $posY = 5$ never changes, although the $posX$ and $posZ$ can be changed with the use of the keyboard. The keyboard buttons that can be used to move the camera the standard way (front, back, right and left) are the ASWD keys, the arrows ←↓↑→ (with double the speed from the ASWD) and the O (front – 10x speed) and K button (back – 10x speed). The two last keys are really helpful if we want to go quickly outside the dome (!) and see it from "outside" as can be seen in Figure 35.

For the mouse movement, we have deployed specific functions that check when the mouse cursor is moving while it is pressed down (*dragging* phenomenon) or when the mouse (left) button is up (the

"dragging" stops) and frequently adjust the camera rotation (rotation is done only for the X and Y axes). The transition from movement to immobility when the *mouse dragging* stops, happens smoothly and not abruptly. Also, when rendering different sky dome samples in the canvas element (or playing the demo), the position and rotation of the camera stays the same as the one in the previous rendering in order to have a consistency between the snapshots of the scene that are being watched by the user of the GUI before and after the generation of the new sample.

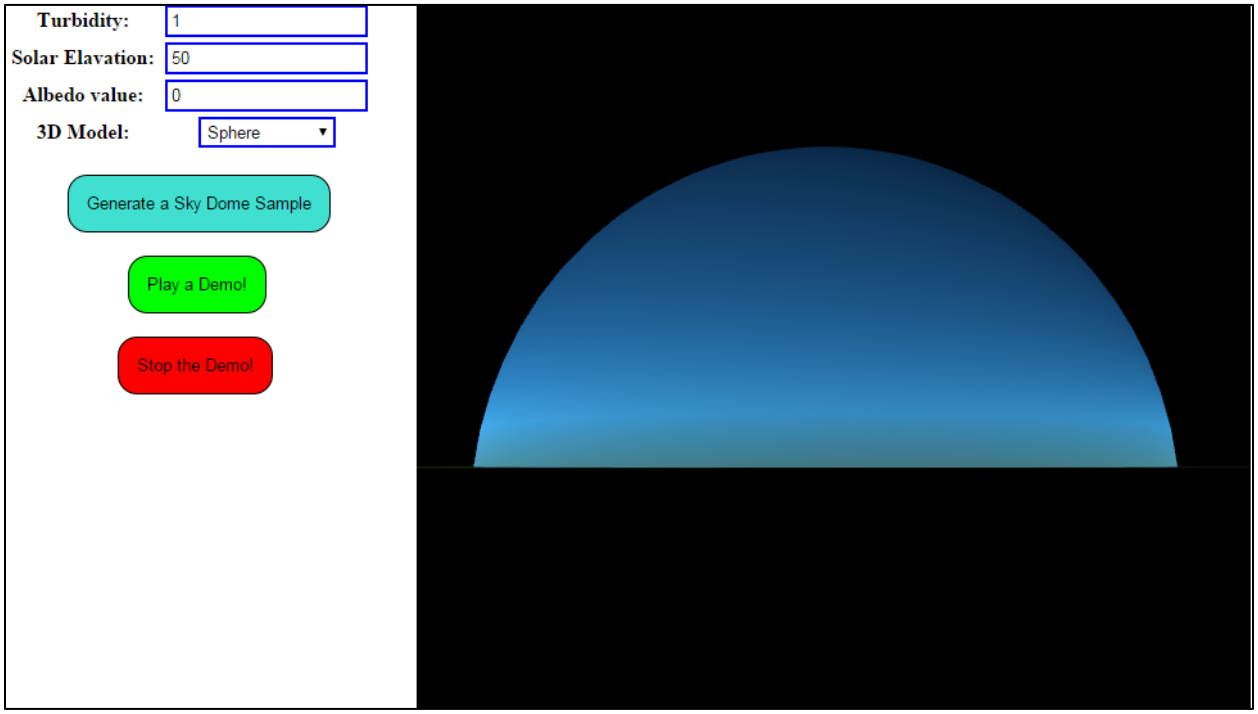


Figure 35: The sky dome on a clear day setup as seen from "outside" the dome

4.5.3 Scene Elements

The sky dome's geometry and colors are defined in the next code snippet, where we can see that all you need to define are the dome's points, their colors and their respective indexes as described in the "Building the sky-dome" section. Note the double-sided drawing feature in the material, without which, the Figure 35 couldn't exist (we would only see black).

```
// the Dome
var domeGeometry = new THREE.BufferGeometry();
domeGeometry.addAttribute('position',new THREE.BufferAttribute(dome_points,3));
domeGeometry.addAttribute('color',new THREE.BufferAttribute(colors_points,3));
domeGeometry.addAttribute('index',new THREE.BufferAttribute(triangle_indices,3));
var material = new THREE.MeshBasicMaterial( { vertexColors: THREE.VertexColors, side:
THREE.DoubleSide } );
var domeMesh = new THREE.Mesh( domeGeometry, material );
SCENE.add(domeMesh);
```

Next, we add the Sun to the scene, which is essential a sphere with radius r as defined in equation (16) – the result of which we can multiply with anything we want if the Sun seems a little too small in our setup (in our case we multiply it by a factor of 2). An image depicting the Sun of our model can be seen in Figure 36. The two "24" values in the *SphereGeometry* function of the code snippet below are the number of horizontal and vertical segments respectively that will be used to draw the sphere: the more the better for drawing smoother spheres but also this results in larger rendering times ("24" is an excellent choice between rendering speed and result for the sphere). The color of the sphere is the Sun's RGB color value (taken from the AJAX response) that we normalized in the "Color mapping" section.

```
// the Sun
var dome_radius = 1000;
var sun_radius = 2 * (dome_radius * 0.004675); // using equation (16)
var sunColor = new THREE.Color(sun_color[0],sun_color[1],sun_color[2]);
var sunGeometry = new THREE.SphereGeometry(sun_radius,24,24);
var sunMaterial = new THREE.MeshBasicMaterial({
        color: sunColor
});
var Sun = new THREE.Mesh( sunGeometry, sunMaterial );
Sun.position.set(sun_position[0],sun_position[1],sun_position[2]);
Sun.castShadow = false;
SCENE.add(Sun);
```

Figure 36: The Sun modeled as a sphere, for turbidity 6, albedo 1 and solar elevation of 50°

The following code snippet adds the ground element to the scene: the lambert material is used to make the ground model (loaded from a texture) as close to a Lambertian surface as it can be. The other characteristics of the plane geometry are self-explanatory. An example of what the ground is like can be seen in the previous Figure 37: it looks like a kitchen tile!

```
//the ground
var planeBufferGeometry = new THREE.PlaneBufferGeometry (2000, 2000); // R=1000 (dome
radius)
var planeColor = new THREE.Color("#8B4513");     // SaddleBrown
var planeMaterial = new THREE.MeshLambertMaterial( {
        color: planeColor,
        map: THREE.ImageUtils.loadTexture('images/funkyGround.jpg')
});
var planeMesh = new THREE.Mesh(planeBufferGeometry, planeMaterial);
planeMaterial.map.wrapS = THREE.RepeatWrapping;
planeMaterial.map.wrapT = THREE.RepeatWrapping;
planeMaterial.map.repeat.set(500,500);
planeMesh.position.set(0,0,0); //center of the ground quad
planeMesh.rotateX(-Math.PI/2); //set it horizontally because planeBufferGeometry is along X,Y
```

```
planeMesh.castShadow = false;
planeMesh.receiveShadow = true;
planeMaterial.map.repeat.x = planeMaterial.map.repeat.y = 50;
SCENE.add(planeMesh);
```

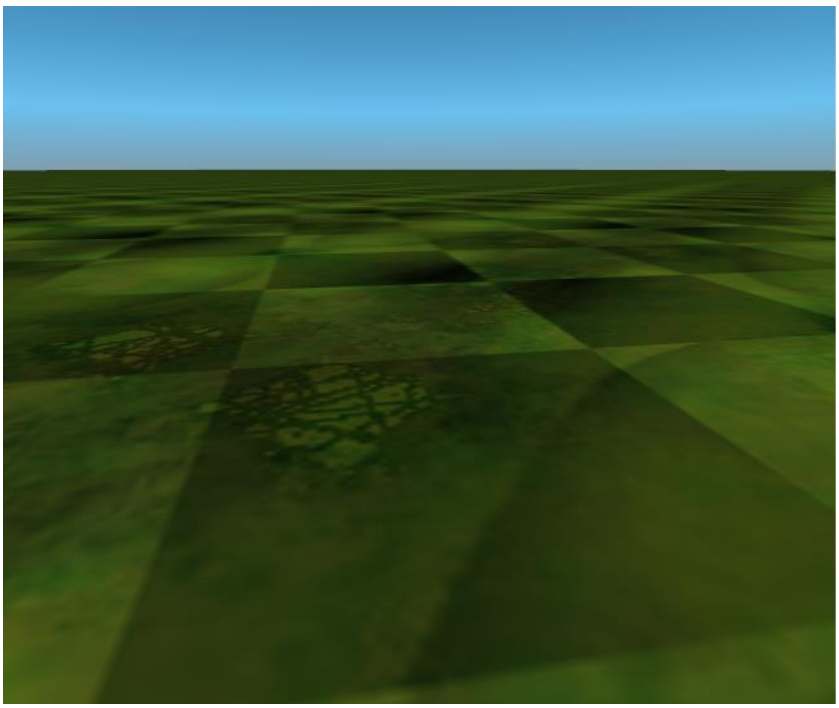


Figure 37: An image showing the ground plane tiles in the scene

Next, we add the 3D-model in the center of our scene based on what the user has chosen from the respective GUI drop-down list. The available models that the user can choose from, are namely:

- a horse
- a dinosaur
- a simple (white) sphere
- a man sitting on a box
- a monster
- an avatar-human
- or no 3D-model (if you don't want to see the effect of the skylight on the model at all or you don't want to lose time waiting for a "heavy" model to load)

The way that a specific model is defined and modeled plays a very important role in the time that it takes to load it into the scene. For example, the white sphere is easy to render since it is described through Three.js code in just a small number of lines (just like the sphere of the Sun which was analyzed above). On the other hand, most of the other models are defined inside a specific file: the man sitting on the box and the horse are 3D models which are defined inside a *.json* file (the loading of which is much optimized in three.js and the function which actually does the loading – called *JSONLoader* – is integrated inside the *three.min.js* file), while all the other models are defined as COLLADA documents (*.dae* files[22]) – which depending on the density of their digital characteristics (polygons, textures, materials, geometries, etc.) that are described inside the .dae file, they can be pretty heavy to load into the scene. This slow-loading behavior may also be a resulting characteristic of the *ColladaLoader.js* script, which does the actual loading for this kind of files – meaning that it is not optimized enough for use by the Three.js on the Web.

Figure 38: Four of our 3D models in a setup of turbidity 5, albedo 1 and solar elevation 30°

[22] **Collada** is an XML-based open standard file format used to store digital assets (such as 3D models for example) in order for those assets to be portable to various graphics software applications that might otherwise store their assets in incompatible file formats.

The dinosaur 3D-model for example, while being the most smooth-curved and free 3D-model we found, it is also the most demanding with regards to the time it takes to load it to the scene: its loading time takes more than 1 sec on an Intel Core 2 Duo processor (usually 4-6 seconds) which means that the demo function won't work well with that model and the browser will get stuck a lot. Four of the aforementioned 3D models can be seen in Figure 38.

Lastly, we add the lights to our scene as seen from the below code snippets. In Three.js there are many kinds of light, the most important of which are:

- Ambient Light (where the light's color gets applied to all the objects in the scene globally)
- Directional Light (this light behaves as though it is infinitely far away and the rays produced from it are all parallel – it can also cast shadows)
- Hemisphere Light (this behaves juts like a light source in the zenith)
- Point Light (this creates a light source which shines in all directions just like a light bulb)
- Spot Light (the de facto light that is used to cast shadows in the objects of a scene)

The 5 dome lights sources as well as the Sun's light source (as seen in Figure 33) are defined as *Point Lights* in our Three.js code: there was a problem defining so many Spot Lights (there is a limit in WebGL rendering to the number of shadow-enabled lights in the scene that can be currently on) and subsequently our scene does not support shadows. The intensity of the Sun's light source is *2x* times larger than the intensity of the dome lights: we wanted the Sun to be the light source that delivers the most significant coloring and hue effect on the 3D model of our scene.

```
// the dome light source that is in the same meridian as the Sun
var color1 = new THREE.Color(lights_colors[0],lights_colors[1],lights_colors[2]);
var domePointLight1 = new THREE.PointLight( color1 );
domePointLight1.position.set(lights_points[0],lights_points[1],lights_points[2]);
domePointLight1.intensity = 1;
SCENE.add(domePointLight1);

// the Sun light source (strongest)
var sunLight = new THREE.PointLight( sunColor );
sunLight.position.set(sun_position[0],sun_position[1],sun_position[2]);
sunLight.intensity = 2;
SCENE.add(sunLight);
```

4.6 Sky Dome Results

In this last section, we will provide some screenshots of the sky dome and the 3D models while we experiment with the 3 basic parameters a user can adjust in the GUI (turbidity, ground albedo and solar elevation). In Figure 39, we can observe the effect of higher albedo values on the luminance of the sky dome for sky profiles with low and high turbidities (we compare albedos of 0 and 1). In Figure 40, we can see a gradual sun rising sequence for $T = 4$: note the change of the colors on the dinosaur model as well as the hazy atmosphere around the Sun. The same sequence is presented in Figure 41, for a turbidity of 8 and different solar elevation snapshots. One can notice the darker atmosphere and the minimized glow around the horizon (all being the result of the higher turbidity) as well as the larger haze around the Sun mainly for the lower solar elevation values. Lastly, in Figure 42 (model used: man sitting in a box) and Figure 43 (model used: white sphere), one can see the effect of turbidity for a fixed solar elevation (high and low respectively) across a large part of the scene's sky dome. In Figure 43, the difference in the sky dome appearance as well as in the sphere's chroma is apparent throughout the change of the turbidity value.

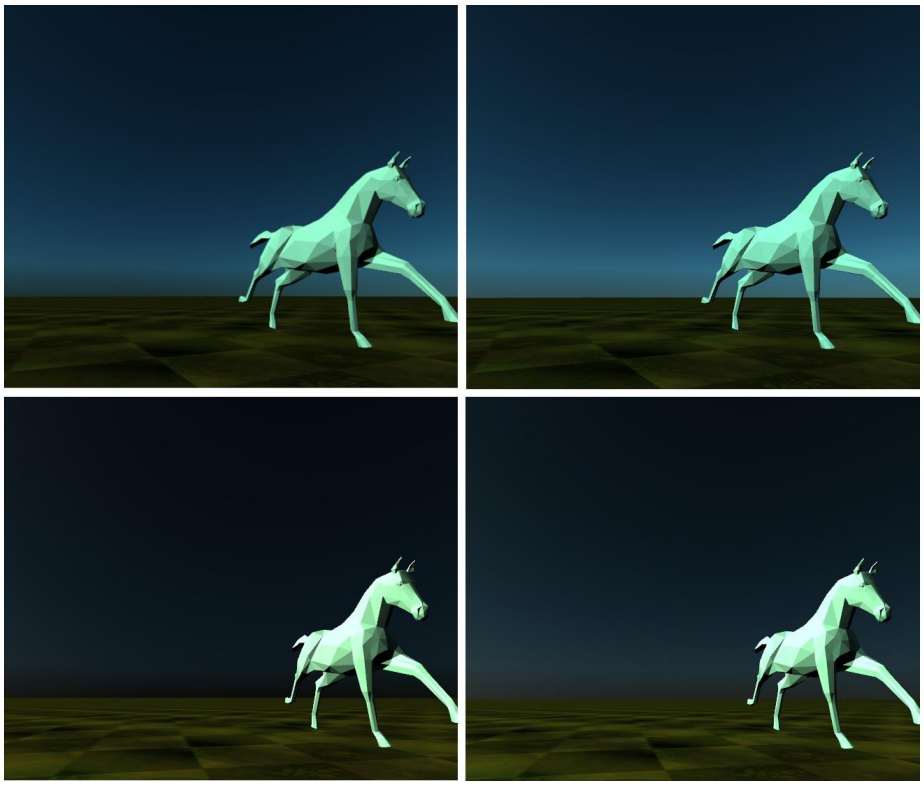


Figure 39: The effect of ground albedo on the sky dome luminance. Up is for turbidity 3 (left: albedo 0, right: albedo 1), below is for turbidity 7 (left: albedo 0, right: albedo 1).

Figure 40: Results of our model for turbidity 4, albedo 0 and solar elevations (from left to right): η = 0.1°, 3°, 6°, 12°, 22°, 34°, 43°

Figure 41: Results of our model for turbidity 8, albedo 0 and solar elevations (from left to right): η = 2°, 4°, 8°, 16°, 25°, 50°

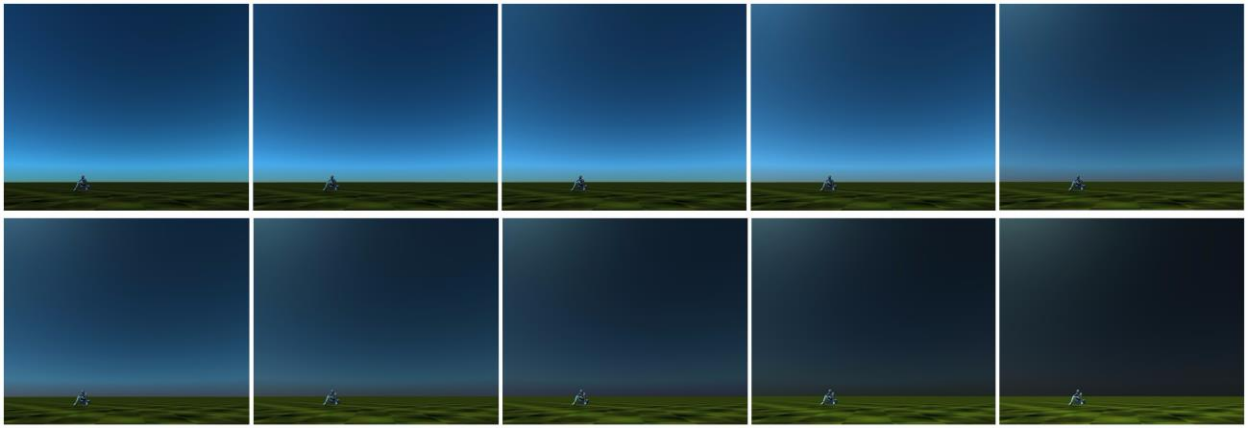


Figure 42: Results of our model for solar elevation η = 64°, albedo 0.5 and turbidities 1 to 10 (left to right)

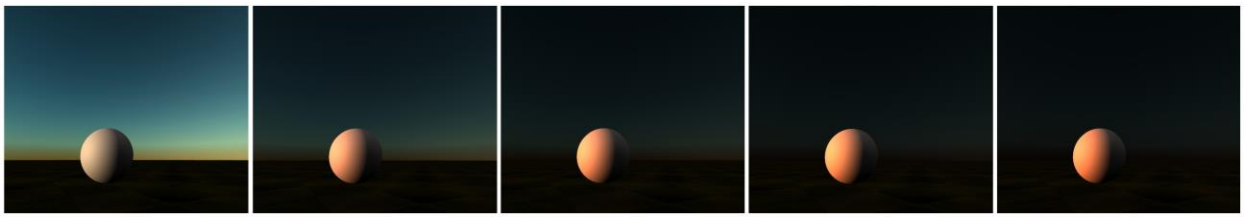


Figure 43: Results of our model for solar elevation η = 3°, albedo 0.5 and turbidities 1 to 5 (left to right)

5 Conclusion

We have presented the most recent skylight models that exist in the relative computer graphics research area. These models include both physical and parametric based models, whose sole purpose is to capture as different sky profiles as possible while keeping the required rendering times to a desired minimum. We have extensively analyzed the Hosek-Wilkie Skylight Model which was the base for building a Web application that shows different sky profiles – while the user changes only 2 or 3 basic parameters. Our work's architecture could be extended/changed in order to update the scene with the new model parameters when the user requests it (or when the demo is playing) and not - as it is now - by starting the rendering process again from the beginning. This would help a lot when loading a "heavy" 3D model, because it would have to be loaded only once and not every time a parameter is changed (e.g. when the solar elevation during the demo is changed every 1 sec).

6 References

Bruneton, Eric, and Fabrice Neyret. 2008. "Precomputed Atmospheric Scattering." *Comput. Graph. Forum Special Issue: Proceedings of the 19th Eurographics Symposium on Rendering.* 1079–1086.

Bruneton, Eric, Fabrice Neyret, and Nicolas Holzschuch. 2010. "Real-time Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF." *Comput. Graph. Forum, Volume 29* 487–496.

CIE. 2002. "Spatial distribution of daylight - CIE standard general sky."

CIE. 1973. "Standardization of Luminance Distribution on Clear Skies."

Elek, Oskar, and Petr Kmoch. 2010. "Real-Time Spectral Scattering in Large-Scale Natural Participating Media." *SCCG '10 Proceedings of the 26th Spring Conference on Computer Graphic.* ACM. 77-84.

Habel, Ralf, Bogdan Mustata, and Michael Wimmer. 2008. "Efficient Spherical Harmonics Lighting with the Preetham Skylight Model." *Eurographics 2008.* Crete, Greece: Eurographics Association. 119-122.

Haber, Jörg, Marcus Magnor, and Hans-Peter Seidel. 2005. "Physically-based simulation of twilight phenomena." *ACM Transactions on Graphics (TOG)* 1353-1373.

Hess, M., P. Koepke, and I. Schult. 1998. "Optical Properties of Aerosols and Clouds: The Software Package OPAC." *Bulletin of the American Meteorological Society,Volume 79, Issue 5* 831-844.

Hosek, Lukas, and Alexander Wilkie. 2013. "Adding a Solar-Radiance Function to the Hosek-Wilkie Skylight Model." *Computer Graphics and Applications.* IEEE. 44-52.

Hosek, Lukas, and Alexander Wilkie. 2012. "An analytic model for full spectral sky-dome radiance." *ACM Transactions on Graphics (TOG), Proceedings of ACM SIGGRAPH 2012.*

Jensen, Henrik Wann, Fredo Durand, Michael M. Stark, Simon Premoze, Julie Dorsey, and Peter Shirley. 2001. "A physically-based night sky model." *SIGGRAPH '01.* ACM New York, NY, USA ©2001. 399-408.

Johnsen, Kjeld, and Richard Watkins. 2010. *Daylight in Buildings.* AECOM Ltd on behalf of the International Energy Agency "Energy Conservation in Buildings and Community Systems Programme" and "Solar Heating and Cooling Programme".

Miguet, Francis, and Dominique Groleau. 2002. "A daylight simulation tool for urban and architectural spaces - application to transmitted direct and diffuse light through glazing." *Building and Environment, Vol. 37* 833 – 843.

Nishita, Tomoyuki, Takao Sirai, Katsumi Tadamura, and Eihachiro Nakamae. 1993. "Display of the earth taking into account atmospheric scattering." *SIGGRAPH '93*. New York. 175-182.

Nishita, Tomoyuki, Yoshinori Dobashi, and Eihachiro Nakamae. 1996. "Display of Clouds Taking into Account Multiple Anisotropic Scattering and Sky Light." *SIGGRAPH '96*. New York. 379-386.

Nishita, Tomoyuki, Yoshinori Dobashi, Kazufumi Kaneda, and Hideo Yamashita. 1996. "Display Method of the Sky Color Taking into Account Multiple Scattering." *SIGGRAPH '96*.

Perez, R., R. Seals, and J. Michalsky. 1993. "All-weather model for Sky Luminance." *Solar Energy* 235-245.

Preetham, A. J., Peter Shirley, and Brian Smits. 1999. "A practical analytic model for daylight." *SIGGRAPH '99*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA ©1999. 91-100.

Zotti, Georg, Alexander Wilkie, and Werner Purgathofer. 2007. "A Critical Review of the Preetham Skylight Model." *WSCG*. Plzen: University of West Bohemia. 23-30.