A Simple Model for Real Time Sky Rendering

Henry Braun and Marcelo Cohen

Graduate Programme in Computer Science - PUCRS
Av. Ipiranga, 6681 - Building 32 - Porto Alegre/RS - Brazil

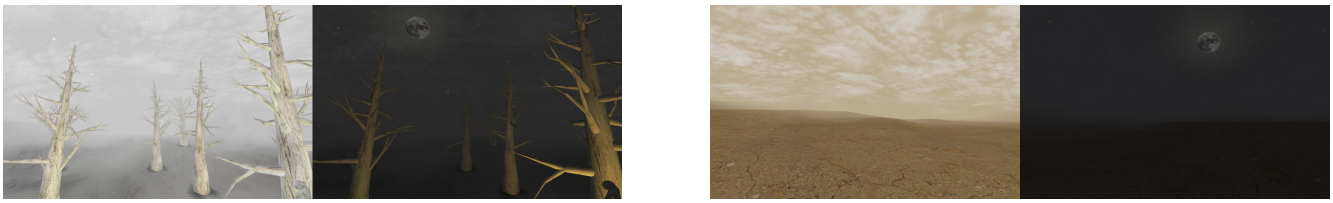

Figure 1: *Pictures from our sky rendering prototype.*

Abstract

This paper presents a model for sky rendering in real time providing day and night cycle animation with a good visual acceptance. Digital games present a set of various genres, some of these games requires outdoor experiences, including day and night cycles. The model proposed in this paper aims to provide a simple technique for sky rendering with minimal graphic processing unit (GPU) and computer processing unit (CPU) costs. This model could be employed in several types of games and applications. Results indicate that our real time animation are visually accepted and have minimal impact on application performance.

Keywords:: Sky, Rendering, Day And Night, Shader, Cycle, Animation, Real Time.

Author's Contact:

henry.braun@acad.pucrs.br,
marcelo.cohen@pucrs.br

1 Introduction

The technological advances in hardware architecture and software engineering allowed the creation of even more realistic scenarios. In order to create more attractive and entertaining games, game developer companies can benefit from the usage of day and night cycle experience. Besides the rendering, these cycles are employed in order to create different gameplay aspects. For example, Warcraft III [1] explores different Non Player Characters (NPC) behaviors according the time of day, such as world visibility and "sleep time" for some enemy troops.

Another set of digital games employ realistic sky rendering techniques, such as Red Dead Redemption[2] and Witcher II[3]. Those games provide good looking outdoor scenes, illustrated in Figure 2 and 3, respectively. However obtaining the accurate sky color can be computationally expensive[Preetham et al. 1999], since the atmosphere consists of air molecules density, sunlight intensity, camera and sun positioning and many other attributes.

This paper presents a model for real time day and night cycles animation, with minimal impact on application performance, allowing the GPU and CPU to be available for other purposes such as Artificial Intelligence (AI), network updates and etc. Our method is easy to implement, as we use a dynamic color based skybox, further detailed in Section 3.3, combined with several layers to render stars, clouds and any other important sky features such as the sun.

The paper is organized as follows: related work are described in the next Section, followed by the description of our model in Section 3.

Figure 2: *Example of light conditions presented in the Red Dead Redemption game.*

Figure 3: *Example of day and night cycle presented in the Witcher II game.*

Some results are shown in Section 4 while in Section 5 some future work is mentioned as well as final considerations.

2 Related Work

Dobashi et al. [Dobashi et al. 1997] developed a method for sky rendering using basis functions. Their method express the sky color intensity distribution by previously calculating each sun altitude sampled at a certain interval. This approach requires a small amount of memory usage and enables the sun and camera positions to be altered. Figure 4 illustrates their results with to different light conditions, morning (left) and evening (right).

Preetham et al. [Preetham et al. 1999] created an analytic model for daylight. Their main goal is to approximate the full spectrum

[1] http://classic.battle.net/war3/basics/daynight.shtml
[2] http://www.rockstargames.com/reddeadredemption
[3] http://www.en.thewitcher.com/

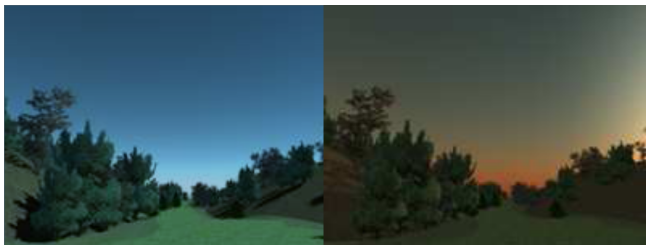

Figure 4: *Dobashi et al. results illustrating to different light conditions: morning (left) and evening (right).*

daylight for a set o atmospheric conditions achieving much accuracy as possible without sacrificing usability. They also verify their results against standard literature from atmospheric science. Figure 5 illustrates their results for morning and evening.

Figure 5: *Preetham et al. results for morning (left) and evening (right).*

Jensen et al. [Jensen et al. 2001] present a physically-based model of the night sky for realistic image synthesis. Their work calculates the illumination coming from the Moon, the stars, zodiacal light and the atmosphere. Their model shows a variety of night scenes based on an accurately predict of the appearance of night scenes using physically-based astronomical data for position and radiometry. They also include position, magnitude and temperature for the visible stars. Figure 6 illustrates their night sky.

Figure 6: *Jensen et al. result for their night sky.*

Another important work in sky rendering was developed by Habert et al. [Haber et al. 2005] Their main goal is to realistically simulate the twilight phenomena for a wide range of different climate conditions. They present a physically-based simulation of the twilight phenomena computing the sky colors before sunrise and after sunset. Figure 7 illustrates the twilight phenomena as their result.

Yang et al. [Yang et al. 2009] explain about several visual effects in computer games. Their article overviews the fixed pipeline main issues and also the programmable pipeline, including topics as global illumination, environmental and sensor effects. For each technique is provided classification regarding it flexibility, complexity and computation stage.

Many of the sky rendering techniques described are not able to be computed in real time. In contrast to previous work, our method presents a good looking and fast technique for sky rendering, easy to implement with minimal GPU and CPU impact.

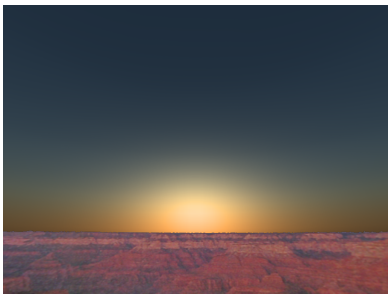

Figure 7: *Habert et al. twilight phenomena result illustration.*

3 Our Model For Sky Rendering

A challenging task in sky rendering is to provide good visual results when combining the sky with other 3D elements, i.e. terrain, trees and buildings. The technique illustrated in our model uses the classic linear fog rendering technique[Akenine-Möller et al. 2008] to envelop the elements and enhance the distance perception, providing a feeling of a long view distance. By adjusting the fog color we are able to provide good scene luminance regarding visual effects, including sunrise and sunset, described in Section 3.1.

The first step in our model consists in obtaining a few parameters defined by the user. These parameters are the minimum and maximum factor for each RGB component. These parameters are processed by the **Luminance Manager** module, further detailed in Section 3.1, responsible in obtaining the light intensity for each given time of our day. For this reason it is necessary a CPU timer. After calculating the light intensity these values are sent to other modules.

The **Fog Manager**, further detailed in Section 3.2, is responsible in painting the scene fog accordingly the color provided by the **Luminance Manager** module. The **Skybox Manager**, explained in Section 3.3, is only for coloring the skybox with the given color, if the user decides to do not use a skybox, the same color should be used as clear color. At last there is the **Layer Manager**, detailed in Section 3.4. This module is in charge of controlling the layers that define the moon, stars, clouds and any other sky feature. Figure 8 illustrates an overview of our Model for Real Time Sky Rendering.

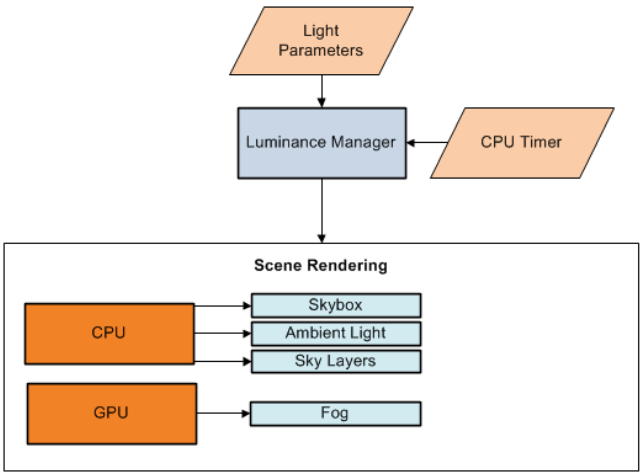

Figure 8: *An Overview of our Model for Real Time Sky Rendering.*

3.1 Luminance Manager - Sunrise and Sunset

The sunrise and sunset effects are computed only in two moments, both defined by the user. These periods are defined by δhf. This approach allows the CPU and GPU to be available to perform other tasks along the other hours of the day.

In order to create the sunrise and sunset effect, we isolate each component from the RGB. For each component we decide the minimum $lmin$ and maximum $lmax$ amounts of strength each should have.

We also must specify the interval δhf we desire the sunrise effect to occur. The light factor l for the sunrise effect is illustrated in equation 1, where cht describes the current hour time (24 hour format), cmt describes the current minute time and st the start time.

$$l = lmin + \delta lf \cdot \frac{(cht - st) \cdot 60 + cmt}{\delta hf \cdot 60}, \quad (1)$$

The dusk effect is very similar, instead we need to provide the maximum amount of light strength in the equation. The light factor for the sunset effect is illustrated in equation 2.

$$l = lmax - \delta lf \cdot \frac{(cht - st) \cdot 60 + cmt}{\delta hf \cdot 60}. \quad (2)$$

The morning curve, described by equation 1, is currently a linear segment representing the crescent sunlight strength amount. Figure 9 illustrates the morning curve during the period from 5:00am to 9:00am, where the $y - axis$ describes the light factor from 0 to 250 and the $x - axis$ express, in minutes, the morning period.

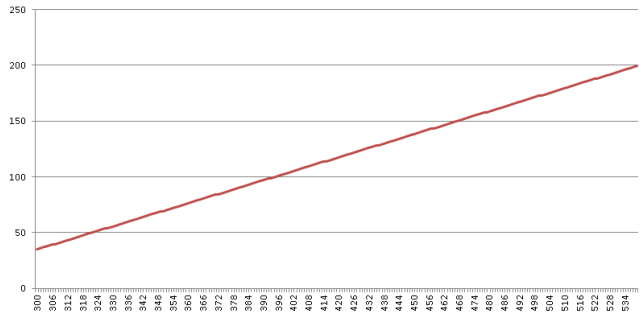

Figure 9: *Illustration of the morning curve during the period from 5:00am to 9:00am.*

The evening curve, described by equation 2, is also a linear segment representing the decreasing sunlight strength amount. Figure 10 illustrates the evening curve during the period from 5:00pm to 9:00pm, where the y-axis shows the computed light factor from 250 to 0 and the x-axis express, in minutes, the evening period.

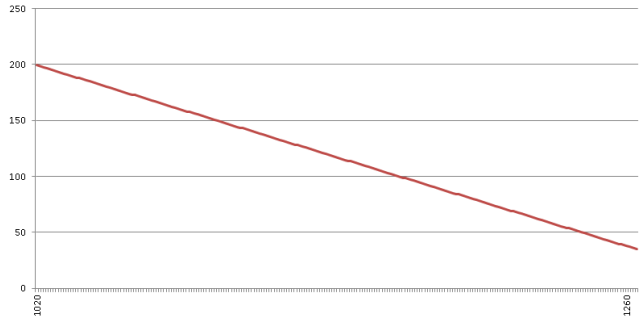

Figure 10: *Illustration of the evening curve during the period from 5:00pm to 9:00pm.*

With those values of light we specify the final color for each frame. This color is used as the ambient light, a uniform light for the whole scene. Besides that we use the same color for coloring the sky and the fog.

3.2 Fog Manager

Fog is a simple atmospheric effect that is performed at the end of the rendering pipeline[Akenine-Möller et al. 2008]. There are several purposes for using this fog effect, in our model, the fog is used to hide the objects sliced by the far plane[4]. We create the fog effect in the GPU, hence it is necessary to provide the near plane and far plane distances. In our prototype we use the linear fog technique combined with the original colors of the objects.

The pixel shader is used only to create the fog for the scene 3D elements. A pseudo-algorithm is presented in Listing 1, illustrating the pixel shader steps used for fog rendering.

```
1  //Constants
2  const float fFogFarDist = 40.0f;
3  const float fFogNearDist = 30.0f;
4
5  //Computing the fog factor based on the far plane and the near plane
6  const float fDeltaFog = fFogFarDist - fFogNearDist;
7  float fFogFactor = clamp((fFogFarDist - VertexPosition.z)/fDeltaFog,0.0,1.0);
8
9  //Collecting the original color
10 vec4 vTexColor = (texture2D(ColoredTextureSampler, TexCoords);
11
12 //Multiply the fog color for better visualization
13 vTexColor *= FogColor;
14
15 //Computing the final color with the fog color and fog factor
16 vec4 vFinalColor = fFogFactor * vTexColor + (1.0 - fFogFactor) * FogColor;
17 return vFinalColor;
```

Listing 1: *Pseudo-algorithm illustrating the pixel shader steps used for the fog effect.*

3.3 Single Color Based Skybox Manager

Our skybox is a cube involving the main camera and this cube is never tested in the Z-Buffer. It is important to mention that our skybox cube contains only background color information. In our prototype, the values obtained for representing the sunlight strength are used as the ambient color and fog color, as explained in Sections 3.1 and 3.2. To tint our skybox we use this same color, this approach creates the combining effect between all scene elements. Painting operations are usually slow when using the CPU, but since our sky contains only one color information, the skybox can be represented using only 4 pixels, one for the left, right, front and back. The top and bottom part can be painted with any other color that the user may like. This approach reflects in a small impact on the application performance. In case the user decide to paint all the skybox with the same color, then this coloring technique may be perform using the clear color. On the other hand clouds and other sky features cannot be represented.

3.4 Layer Manager - Stars, Moon and Clouds

The sky is composed by several elements, but as explained in Section 3.3, our skybox presents only color information, making impossible to create common sky elements, such as clouds and stars. Some of these elements must be represented in order to increase the scene realism. We create these features using a layer for each component, assuring that each layer is never tested in Z-Buffer, and the texture is repeated along its geometry surface. This creates a result similar to the skybox, that these elements cannot be hidden by fog or the far plane. For more realism the texture coordinates can be displaced along the time in order to create a moving illusion i.e. clouds.

All the layers are always drawn, but they must contain alpha information. The layers alpha value can be simply adjusted accordingly the light factor l, for daylight components (i.e. clouds and the sun) the alpha factor af is illustrated in equation 3 and for nighttime components (i.e stars and the moon) the alpha factor is illustrated in equation 4.

$$af = l - lmin, \quad (3)$$

$$af = 255 - (l + lmin). \quad (4)$$

4 Results

The results are obtained using an Intel Xeon E405 equipped with NVidia Quadro FX 4800. For rendering, our prototype uses Irrlicht Engine[5]. Figure 11 illustrates the sunrise and sunset effect from 8 o'clock in the morning to 9 o'clock in the night.

[4] A plane representing the camera farthest viewing distance, any objects beyond the far plane are not drawn.

[5] http://irrlicht.sourceforge.net/

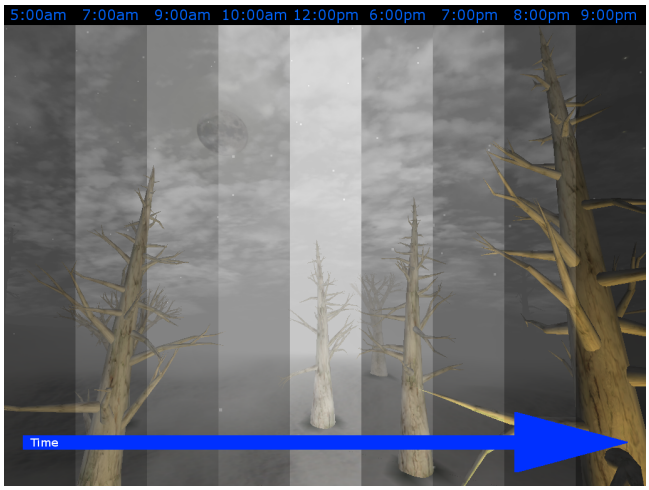

Figure 11: *Illustration of the sunrise and sunset effect from 8 o'clock in the morning to 9 o'clock in the night.*

Figure 12 illustrates the sunrise effect from 8 o'clock in the morning to noon.

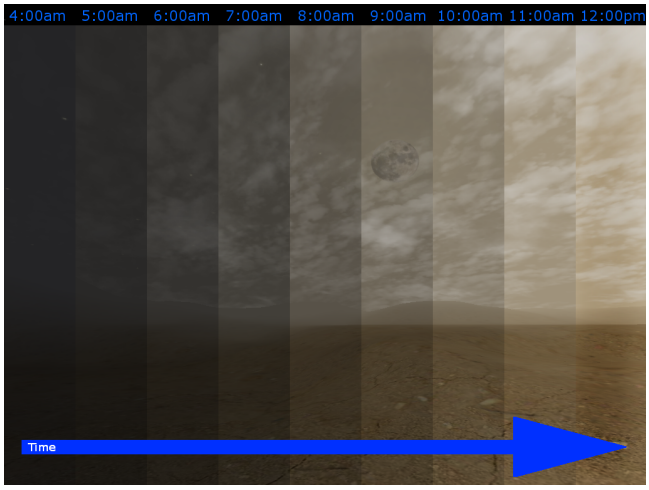

Figure 12: *Illustration of the sunrise effect from 8 o'clock in the morning to noon.*

By changing the Light Parameters we can achieve different atmospheric visual results. Figure 13 illustrates a snow scene with the brightest(left) and the darkest(right) moment of the day using the following set of parameters presented in Table 1.

Channel	*Min*	*Max*
R	35	200
G	35	200
B	35	200

Table 1: *Light Parameters for the snow environment.*

Figure 13: *Illustration of a snow scene with the brightest(left) and the darkest(right) moment of the day.*

Figure 14 illustrates a simple desert atmosphere: in order to create this scene we used the following set of parameters presented in Table 2.

Channel	*Min*	*Max*
R	35	188
G	35	172
B	35	146

Table 2: *Light Parameters for the desert environment.*

Our prototype runs at 70 fps with all the modules enabled, disabling the sky rendering technique the frame rate increases up to 73 fps.

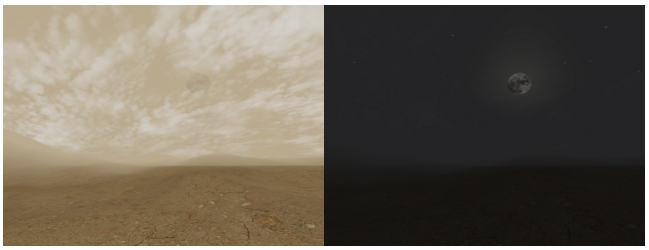

Figure 14: *Illustration of a simple desert atmosphere, with the brightest(left) and the darkest(right) moment of the day.*

Figure 15 illustrates a green forest using the following set of parameters presented in Table 3.

Channel	*Min*	*Max*
R	35	150
G	35	158
B	35	085

Table 3: *Table containing the Light Parameters for the forest atmosphere.*

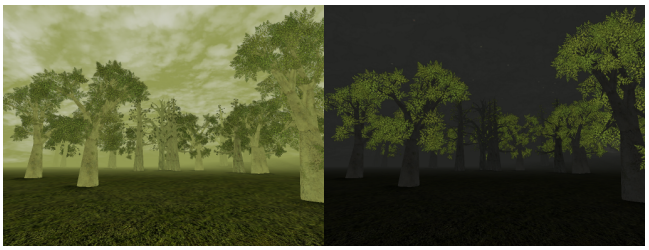

Figure 15: *Illustration of our green forest in two different moments of the day.*

5 Final Considerations

This paper presented a model for sky rendering under different light conditions. In order to achieve better visual results the sky should be improved, creating a system of texture layers to provide the subtle sky color variations. It is also important that the sunlight color factor variation is represented exponentially and not linear [Dobashi et al. 1997], as such approach would represent the sky color variation with a higher degree of realism.

References

AKENINE-MÖLLER, T., HAINES, E., AND HOFFMAN, N. 2008. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA.

DOBASHI, Y., NISHITA, T., KANEDA, K., AND YAMASHITA, H. 1997. A fast display method of sky color using basis functions. In *The Journal of Visualization and Computer Graphics*, 115–127.

HABER, J., MAGNOR, M., AND SEIDEL, H.-P. 2005. Physically-based simulation of twilight phenomena. *ACM Trans. Graph. 24* (October), 1353–1373.

JENSEN, H. W., DURAND, F., DORSEY, J., STARK, M. M., SHIRLEY, P., AND PREMOŽE, S. 2001. A physically-based night sky model. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 399–408.

PREETHAM, A. J., SHIRLEY, P., AND SMITS, B. 1999. A practical analytic model for daylight. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '99, 91–100.

YANG, X., YIP, M., AND XU, X. 2009. Visual effects in computer games. *Computer 42*, 48–56.