Real Time Rendering of Atmospheric Scattering Effects for Flight Simulators

Ralf Stokholm Nielsen

LYNGBY 2003
EKSAMENSPROJEKT
NR.

IMM

Trykt af IMM, DTU

Abstract

A method to simulate the effects of atmospheric scattering in a real time rendering system is proposed. The system is intended for use in pc based combat flight simulators where the aim is to create a perceptually realistic rendering of the sky and terrain.

Atmospheric scattering is responsible for the color of the sky and for the gradual extinction and attenuation of distant objects. For flight simulators, a realistic simulation of these effects play a central role both to the emersion and to the tactical environment.

Realistic radiometric simulation of atmospheric scattering is computationally expensive in such a way that it prohibits any chance of doing it in real time. It is, however, possible to use the radiometric physics of the atmosphere as the basis of a set of simplifications that allows real time visual simulation of atmospheric scattering.

Existing methods for real time visual simulation of atmospheric scattering assumes a constant density atmosphere. This assumption will not provide realistic results in a flight simulator environment.

The proposed system expands an existing method developed by Hoffman and Preetham [10] to consider the density change in the atmosphere. In addition, parts of the model is modified to compensate for shortcomings in previous methods.

The resulting system is capable of producing visually convincing scattering effects for a range of observer altitudes and environments, and will add minimal overhead to an existing rendering system.

Contents

List of Figures **11**

1 Introduction **17**

1.1 Background 18

1.2 Problem 19

1.3 Limitations 19

1.4 Outline 20

2 Atmospheric Scattering **21**

2.1 Rayleigh Scattering 24

2.2 Mie Scattering 25

2.2.1 Henyey-Greenstein Phase Function 26

2.3 Optical Depth 27

2.4 Sky Color 28

2.4.1 Variations in Color over the Sky Dome 28

2.5 Areal Perspective 33

2.6 Visibility 34

2.7 Summary 35

3 Rendering Scattering Effects 37

3.1 Basic Problem 37

3.1.1 Sky Color 38

3.1.2 Areal Perspective 40

3.1.3 RGB Color from Spectral Distribution 41

3.2 Previous Work 44

3.2.1 Real Time Approaches 48

3.2.2 Classical Real Time Methods 52

4 Graphics Hardware and API 55

4.1 3D Graphics API 55

4.1.1 DirectX and Direct3D 56

4.2 Programmable Rendering Pipeline 56

4.2.1 Vertex Shaders 57

4.2.2 Pixel Shaders 58

4.2.3 HLSL High Level Shading Language 59

4.2.4 DirectX Effect Framework 59

5 Hypothesis 61

5.1 Summary of Previous Methods 61

5.2 Scattering Models 62

5.2.1 Rayleigh Scattering 64

5.2.2 Mie scattering 65

5.3 Areal Perspective 67

5.3.1 Proposed Method for Calculating Areal Perspective 67

5.4 Sky Color 69

5.4.1 Proposed Method for Calculating Sky Color 69

6 Implementation 73

6.1 Code Structure 73

6.2 Optical Depth Estimations 75

6.2.1 Areal Perspective 75

6.2.2 Sky Color 75

6.3 Terrain Rendering 76

6.4 Sky Rendering 78

6.4.1 Sky Dome Tessellation 79

6.5 Vertex and Pixel Shader Implementation 81

6.5.1 Constant Registers 82

6.5.2 Vertex Shader 83

6.5.3 Pixel Shader 84

7 Results 87

7.1 Sky Color 88

7.1.1 Rayleigh Scattering Intensity 88

7.1.2 Effect of Observer Altitude on Sky Color 88

7.1.3 Change in Sky Color Close to the Sun 90

7.1.4 Sky Color with Changing Sun Position 91

7.2 Areal Perspective 91

7.2.1 Blue Color of Distant Mountains 92

7.2.2 Variation in Visibility with Terrain Altitude 92

7.2.3 Variation in Visibility With Observer Altitude . . . 94

7.2.4 Visibility as a Result of Aerosol Concentration . . . 95

7.2.5 Variations in Areal Perspective with Sun Direction . 96

7.3 Comparison with Previous Methods 97

7.3.1 Density Effect on Sky Color 97

7.3.2 Density Effect on Terrain Visibility 98
7.3.3 The Modified Rayleigh Phase Function 98
7.4 Artifacts and Limitations 98
7.4.1 Tessellation Artifacts 99
7.4.2 Mie factor artifact 100
7.4.3 Clamping Artifact 100

8 Discussion 103

8.1 Scattering Models 103
8.1.1 Rayleigh Scattering 104
8.1.2 Mie Scattering 104
8.2 Sky Color 105
8.3 Areal Perspective 106
8.4 Optical Depth Estimates 107
8.5 Problems and Future Improvements 107

9 Conclusion 109

9.1 Summary of the Proposed System 110
9.2 Fulfillment of the Hypothesis 111

Bibliography 113

A Instructions for the Demo Application 117

B Content of the CD-ROM 119

C Atmospheric Density 121

D Effect Files 123

D.1 Sky Effect 123
D.2 Terrain Effect 125

E Source Code Snippets **129**

E.1 Roam.cpp 129

E.2 Sky.cpp 131

E.3 Skydome.cpp 133

List of Figures

2.1 Single scattering event 22
2.2 Extinction of distant objects 23
2.3 Rayleigh phase function 25
2.4 Mie scattering phase function 26
2.5 Henyey-Greenstein phase function 27
2.6 Angles on the skydome 29
2.7 Sky color effects 30
2.8 Why the horizon is white. 31
2.9 Optical depth evens colors 32
2.10 Effect of aerosols concentration 33

3.1 Single Scattering 38
3.2 Areal perspective 39
3.3 CIE XYZ Color matching functions 42
3.4 Klassen's model of the atmosphere 44
3.5 Klassen's model of the fog layer 45
3.6 Previous work non real time rendering 46
3.7 Nishita's coordinate system 47
3.8 Rayleigh artifact 49

3.9 Volumetric Scattering effects 51
3.10 Classic hardware fog 52

4.1 Programmable graphics pipeline 57

5.1 Modified Rayleigh phase function 65
5.2 Sky optical depth estimates 68
5.3 Sky color interpolation function 70
5.4 Skydome optical depth profiles. 72

6.1 Implementation class diagram 74
6.2 Wireframe ROAM Terrain 77
6.3 Sky dome horizon 78
6.4 Old style dome tessellation 79
6.5 New dome tessellation 80

7.1 Sunset Sky 87
7.2 Rayleigh Directionality 88
7.3 Sky color at different altitudes 89
7.4 Mie scatter directionality images 90
7.5 Sunset sky color rendering 91
7.6 Areal perspective rendering 92
7.7 Average density estimation methods 93
7.8 Observer altitude effects 94
7.9 Variation in haziness 95
7.10 Directional effects on areal perspective 96
7.11 Comparison to a constant density system 97
7.12 Skydome rendering using previous method 98

7.13 Effect of poor tessellation 99
7.14 Horizon blending artifact 100
7.15 Clamping artifacts 101

C.1 Standard atmosphere density 122

Preface

This thesis is written by Ralf Stokholm Nielsen in partial fulfillment of the requirements for the degree of Master of Science at the Technical University of Denmark. The work was conducted from February 2003 to September 2003 and advised by Niels Jørgen Christensen, Associate Professor, Department of Informatics and Mathematical Modelling (IMM), DTU.

The thesis is related to the areas of computer graphics, real time rendering and atmospheric scattering. The reader is expected to be familiar with engineering mathematics and the fundamental principles of computer graphics and real time rendering.

Acknowledgements

I take this opportunity to thank the people who have helped me during the creation of this thesis.

First I would like to thank my advisor Niels Jørgen Christensen for his encouragement and support throughout the project, and for helping me focus when too many ideas were present.

I would also like to thank Bent Dalgaard Larsen and Andreas Bærentsen for insightful comments and ideas during development.

Special thank to my friend Tomas ”RIK” Eilsøe For his support and critique, and for providing me with insight and a wealth of reference photos through his work as a F-16 pilot in the Royal Danish Air Force.

Thank to Christian Vejlbo for reading and commenting on the thesis on several occasions during development.

Also thank to my sister Susanne Stokholm Nielsen for grammatical revision.

Finally a big thank to my girlfriend Charlotte Højer Nielsen for her support, encouragement and for putting up with me throughout the project.

Technical University of Denmark DTU

September 2003

Ralf Stokholm Nielsen

Chapter 1

Introduction

In the past, real time rendering of outdoor scenes in flight simulator applications has primarily dealt with the problem of increasing the geometric detail of the part of the terrain visible to the user.

During the last five years of the 20'th century, advances in consumer graphics hardware have enabled amazing improvements in the raster capabilities of pc's. With the change of the millennia, consumer graphics hardware capable of transforming and lighting vertices began appearing. This has, over a few years, lead to graphics processors GPU's that are becoming increasingly flexible and capable of performing small programs, both on the pixel and vertex level.

The adversity and raw processing power of modern GPU's are currently shifting the focus of real time rendering towards some of the areas traditionally associated with (non real time) image synthesis. This has opened the opportunity for rendering much more realistic images.

Realism can in this context be interpreted as either physical realism, meaning that the rendering systems use models that closely resemble the physics of lights interaction with the environment, or visual realism, meaning that the generated images display a convincing similarity with the real world. In the following, realism will, unless explicitly stated, refer to the latter definition.

1.1 Background

Flight simulators have always represented an area where the separation between professional software and ”games” has been blurry. Producers of PC flight simulators have competed to supply the most realistic simulation of the flight dynamics and avionics of the aircraft. The focus on realism has also meant that simulators are presented with a challenge when it comes to rivaling this realism in the rendering of the environment.

The rendering of the environment in a flight simulator lacks many of the shortcuts present in other games. A flight in a flight simulator spans a potential long time period and a vast area. In addition, flight simulators are often set in a known environment, requiring the landscape to be recognizable. The time covered prohibits the use of static coloring of the sky and landscape, the free nature of flight prohibits the use of a textured skydome to display cloud effects, and the large area covered during flight requires the processing of enormous amounts of data in order to render the terrain.

Although algorithms for rendering huge terrain datasets have been available for some time now [15, 11, 5], there are still plenty of areas that can be improved tremendously. Recent works have begun investigating the use of modern shader technology for simulating atmospheric scattering effects [10, 3], while others are investigating the use of similar technologies to transfer techniques such as HDR (High Dynamic Range) imagery to realtime applications [6, 7].

Even though the processing power has increased enormously since the first flight simulators were designed, it is still unrealistic to imagine a physically based rendering system, calculating the radiative transfer throughout the scene. Consequently, the rendering system still needs to be designed to imitate realism rather than to be realistic.

Two different approaches exist for rendering atmospheric scattering effects in real time. Dobashi proposed a volumetric rendering method capable of capturing things like shafts of light and shadows cast by mountains [3]. Hoffman and Preetham proposed a simpler approach where atmospheric scattering is calculated at the individual objects based on distance from the viewpoint and angle to the sun, the system was implemented using vertex shaders. Both methods have advantages and disadvantages, and none of them can directly fulfil the requirements for implementation in a PC flight simulator.

1.2 Problem

The purpose of this work is to develop a system of shades to simulate the effects of atmospheric scattering in a flight simulator environment. The problem is divided into two sub problems; rendering of the sky and rendering of the landscape.

For the sky, the system must be capable of capturing the change in color when the sun travels over the skydome during the day. It must also be capable of capturing the change in color that results when the observer changes altitude. Both effects are important to the sensation of immersion.

For the terrain, the purpose is to create a system capable of simulating the effects of *areal perspective*, the effect of observing a distant object through the atmosphere, see section 2.5. The system must be able to simulate the change in color of distant objects. It must also be capable of simulating the change in visibility for different view angles compared to the direction of the sun. Last, it must be capable of capturing the change in areal perspective when changing observer altitude. Areal perspective is central for the human ability to determine distance and, as a result, plays a central role to the realism of a flight simulator rendering system.

Both areal perspective and sky color change with the altitude of the sun above the horizon and with the amount of aerosols in the atmosphere. The system has to be able to cope with these changes fluently.

Flight simulators are some of the most processor intensive real time rendering systems and, consequently, the developed system must be able to work at real time rates while leaving recourses for other tasks like flight modelling, AI, avionics etc.

1.3 Limitations

The main focus of this work is on creating a system suitable for inclusion in a flight simulator. The aim is to be able to convincingly recreate the effects of atmospheric scattering on a subjective level.

The physics of atmospheric scattering are crucial to understanding the problem but the physical accuracy of the system is irrelevant. It is not intended to accurately simulate a given physical environment but to present the virtual pilot with some of the phenomenons he would expect to observe in the real world.

To demonstrate the system, a small application capable of rendering a height field and a skydome is developed. These are developed to demonstrate the shaders and they will not be highly optimized.

The system is targeted at PC systems equipped with a recent graphics card.

1.4 Outline

In chapter two, the theory of atmospheric scattering is presented. It is a rough overview of the system, intended to provide the reader with a basic understanding of the theory required for understanding the proposed simplifications.

Chapter three explains the basic problem of rendering scattering effects and outlines the work of previous researchers in the area of rendering scattering effects.

In chapter four, the technology behind modern rendering hardware and APIs is described to provide the reader with a basic understanding of the technology, required for the understanding of the proposed implementation.

In chapter five, the hypothesis is developed, based on the work of previous researchers and the theory described in chapter two.

Chapter six describes the implementation of the rendering system proposed in the hypothesis.

In chapter seven, the renderings produced by the implemented system are shown.

Chapter eight discusses the results and provides ideas for improvements and future development.

Chapter nine presents the conclusion and is followed by appendices.

Chapter 2

Atmospheric Scattering

As sunlight penetrates the atmosphere it may be absorbed, scattered, reflected or refracted before reaching the surface. Humans perceive light with the help of antenna-like nerve endings located in our eyes (rods and cones). Our eyes detect different intensities (light and dark) and different colors depending on the wavelengths of visible radiation.

White light is a combination of all wavelengths from $400-700$ nm in nearly equal intensities. The sun radiates almost half of its energy as visible light. The peak intensity of the sun's electromagnetic radiation corresponds to the color yellow. Because all visible wavelengths from the sun reach the cones in nearly equal intensities when the sun is close to zenith (with a slight peak at yellow), the sun appears yellowish-white during the middle of the day.

The attenuation of light in the atmosphere is caused by absorbtion and scattering, and can be divided into effects that remove and add light to a given viewing ray. Absorbtion of visible light is negligible except for absorbtion in the ozone layer [8]. As a result, the atmosphere below the ozone layer can be treated as a scattering media only. Scattering is a result of the interaction between the electromagnetic field of the incoming light with the electric field of the atmospheric molecules and aerosols [16]. This interaction is synchronized and, as a result, the scattered light has the same frequency and wavelength as the incoming light. Scattering differs with particle size and varies with wavelength. For this reason, the spectral composition of the scattered light differs from that of the incoming light.

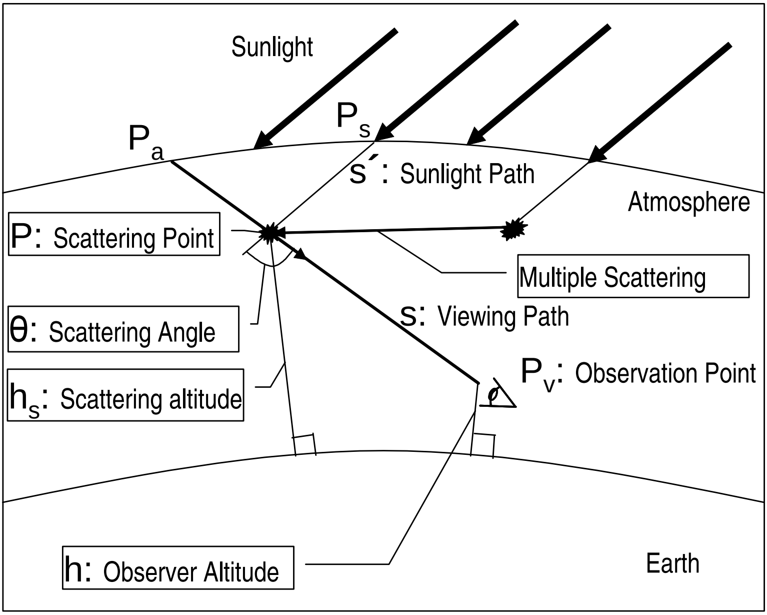

Sunlight
P_s
P_a
s´: Sunlight Path
Atmosphere
P: Scattering Point
Multiple Scattering
θ: Scattering Angle
s: Viewing Path
P_v: Observation Point
h_s: Scattering altitude
h: Observer Altitude
Earth

Figure 2.1: *The sizes and parameters involved in a single scattering event in the atmosphere.*

Figure 2.1 demonstrates a single scattering event. At a point P light from the sun is scattered into the viewing path (single scattering). Light already scattered one or more times also arrives and is scattered at P (multiple scattering). The angle between the viewing path and the direction of the sunlight θ is called the scattering angle. This angle is the independent variable in the scattering phase function described later in the sections on Mie and Rayleigh scattering. The total amount of light arriving at the viewpoint P_v is the combined effect of scattering effects along the entire viewing path s. The light incident on the viewdata from the single scattering event at P is attenuated by scattering before reaching the viewpoint P_v.

Atmospheric scattering will both add and remove light from a viewing ray. This is the effect that causes the brightening of distant dark object along with a loss of contrast (see figure 2.2). This effect is very important to the human ability to assess distances even at surprisingly short ranges [17].

Any type of electromagnetic wave propagating through the atmosphere is

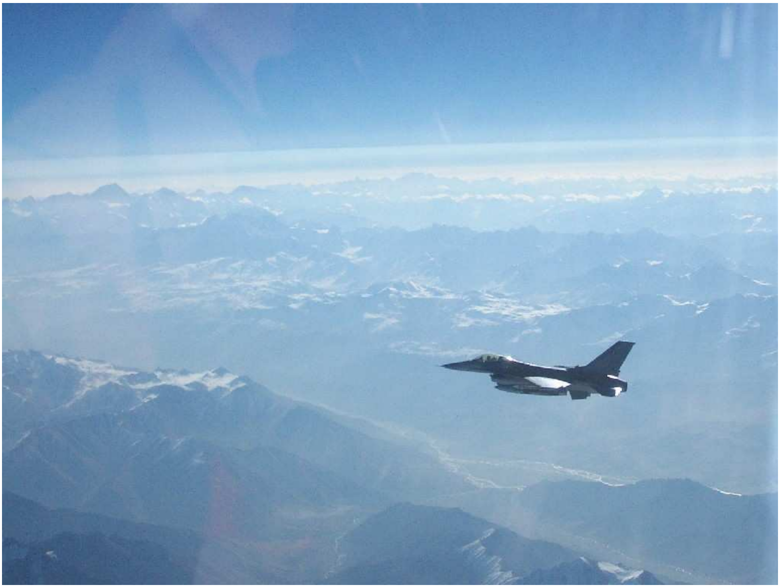


Figure 2.2: *The extinction of the mountains far from the viewpoint caused by atmospheric scattering.*

affected by scattering. The amount of scattered energy depends strongly on the ratio of particle size to the wavelength of the incident wave. When scatterers are small relative to the wavelength of incident radiation ($r < \lambda/10$), the scattered intensity on both forward and backward directions is the same. This type of scattering is referred to as Rayleigh scattering. For larger particles ($r \geq \lambda/10$), the angular distribution of scattered intensity becomes more complex with more energy scattered in the forward direction. This type of scattering is described by Mie scattering theory.

When the scattering particles are considered to be isotropic, the shape of the scattering phase function is uniform with respect to the direction of the incoming beam. This is not always the case for atmospheric particles, but any error introduced by making the assumption that they are, is evened out by the large number of randomly oriented particles [16] and, consequently for this thesis, particles are considered isotropic.

When the relative distance between the scattering particles is large compared to the particle size, the scattering pattern of each function is unaffected by the presence of other particles. This is the case in the atmosphere and, as a result, the scattering from a path in the the atmosphere can be approximated, using a single function for each of the two types of scattering; scattering by particles and scattering by molecules [16].

A small fraction of the scattered light is scattered again one or more times before leaving the scattering media. This is referred to as multiple scattering. In most situations, the effects of multiple scattering on the intensity of the direct beam are barely noticeable. This is the case for scattering in the atmosphere on a clear day [16, 17]. However, ignoring multiple scattering when trying to describe the color of clouds will create noticeable artifacts [9]. In general, multiple scattering influences are more significant in turbid or polluted atmospheres.

2.1 Rayleigh Scattering

Rayleigh scattering refers to the model of atmospheric scattering caused by molecules (clean air). In this context, only scattering of visible light is relevant. The volume angular scattering coefficient, or phase function $\beta^{\lambda}(\theta)$, describes the amount of light at a given wavelength λ scattered in a given direction θ. For Rayleigh scattering, this is given by [13].

$$\beta(\theta) = \frac{\pi^2 \left[n^2 - 1\right]^2}{2N\lambda^4} \left[1 + cos^2\theta\right] \tag{2.1}$$

where θ is the angle between the view angle and the sun direction, n is the refractive index of air, N is the molecular density and λ the wavelength of light. The important property of the Rayleigh scattering phase function (2.1) is the $\frac{1}{\lambda^4}$ dependency of wave length. The net result of this property is that shorter wavelengths are scattered much more than longer wavelengths, approximately an order of magnitude for the visible spectrum $(400 - 700nm)$. This is the main reason why the sky is blue.

The total Rayleigh scattering coefficient β_R can be derived from equation (2.1) by integrating over the total solid angle 4π

$$\beta_R = \int_0^{4\pi} \beta(\theta) d\Omega = \frac{8\pi^3 \left[n^2 - 1\right]^2}{3N\lambda^4} \tag{2.2}$$

The total scattering coefficient describes the total amount of light removed from a light beam by scattering.

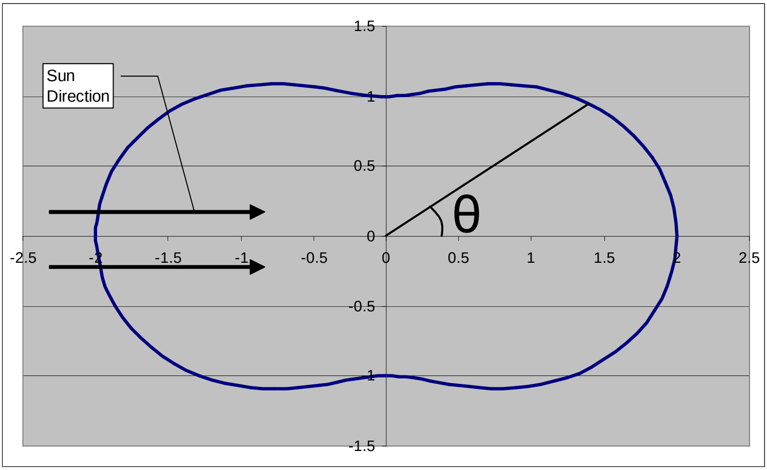


Figure 2.3: *Rayleigh phase function, the intensity of light scattered into the viewing path, as a function of the angle from the sun.*

2.2 Mie Scattering

Mie scattering theory is a general theory applicable to scattering caused by particles of any size. In this thesis, Mie scattering theory is used exclusively for describing the scattering caused by atmospheric particles (aerosols) with sizes equal to or larger than the wavelength of the scattered light.

Rayleigh scattering is a subset of Mie scattering. Consequently, Mie scattering theory will yield the same results as Rayleigh scattering when applied to small particles. If assuming a certain average size of the scattering particles, the Mie scattering function can be written as [20].

$$\beta(\theta) = 0.434c\frac{4\pi^2}{\lambda^2}\,0.5\beta_M(\theta) \tag{2.3}$$

Where c is the concentration factor which varies with turbidity T and is given by [20].

$$c = (0.6544T - 0.6510) \cdot 10^{-16} \tag{2.4}$$

and β_M describes the angular dependency (phase function)[20]. β_M varies with the size of the scattering particles and gives the shape of the angular

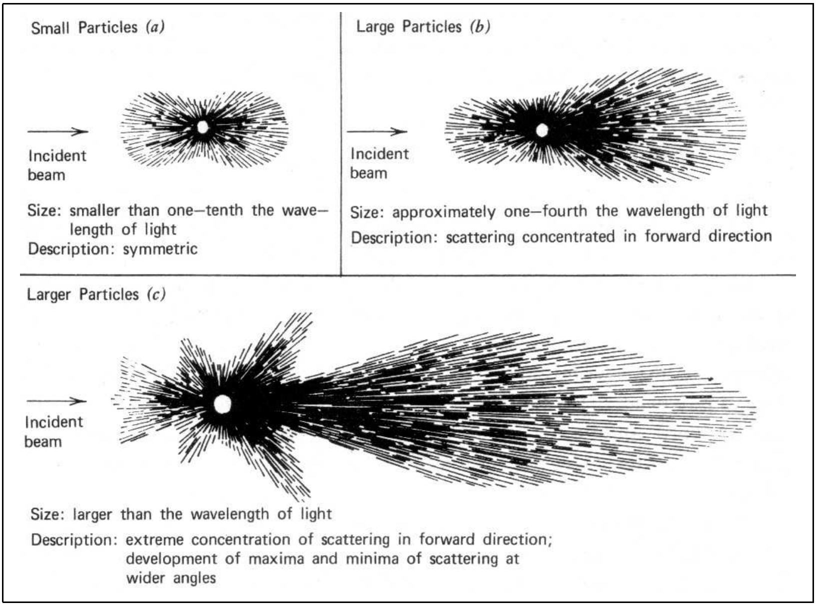

Small Particles (a)
Incident beam
Size: smaller than one–tenth the wave–length of light
Description: symmetric
Large Particles (b)
Incident beam
Size: approximately one–fourth the wavelength of light
Description: scattering concentrated in forward direction
Larger Particles (c)
Incident beam
Size: larger than the wavelength of light
Description: extreme concentration of scattering in forward direction; development of maxima and minima of scattering at wider angles

Figure 2.4: *Mie angular scattering functions. The top left image is identical to the Rayleigh phase function, demonstrating that Rayleigh scattering theory is a subset of Mie scattering for small particles* $r < \frac{\lambda}{10}$

Mie scattering function (figure 2.4). The total Mie scattering factor is determined by.

$$\beta_M = 0.434 c\pi \frac{4\pi^2}{\lambda^2} K \qquad (2.5)$$

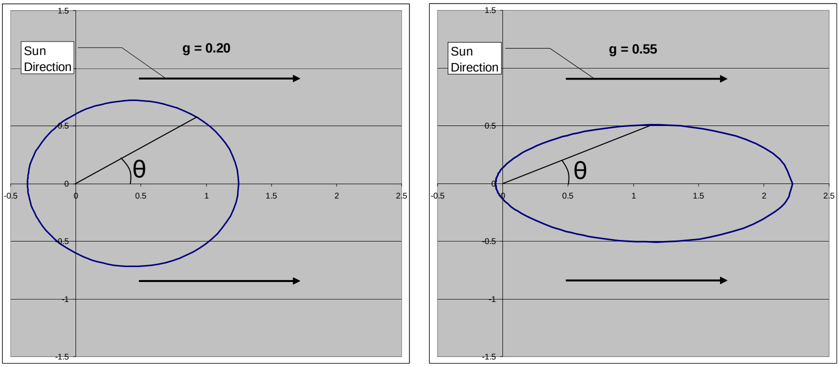
Where K varies with λ and are ~ 0.67[1].

2.2.1 Henyey-Greenstein Phase Function

Mie theory is in general far more complicated than Rayleigh theory. However, for the application to real time rendering, the angular scattering func-

---

[1] Both (2.3) and (2.5) are slightly incorrect $\frac{4\pi^2}{\lambda^2}$ should be $\left(\frac{2\pi}{\lambda}\right)^{v-2}$ where v is Junge's exponent. However a value of 4 is used for the Mie scattering model in this thesis. This is taken from [20, 10]



(a) Shape of the HG phase function for $g = 0.20$

(b) Shape of the HG phase function for $g = 0.55$

Figure 2.5: *The shape of the Henyey-Greenstein phase function with varying g. The Henyey-Greenstein phase function is a simplification of the general Mie scattering phase function as shown in figure 2.4.*

tion can be approximated using the Henyey-Greenstein phase function [10].

$$\Phi_{HG}(\Theta) = \frac{1 - g^2}{4\pi \left(1 + g^2 - 2cos(\theta)\right)^{\frac{3}{2}}} \tag{2.6}$$

Where g is the directionality factor. Figure 2.5 shows how the shape of the phase function varies with the value of g.

The Henyey-Greenstein (HG) phase function belongs to a class of functions used primarily for their mathematical simplicity than for their theoretical accuracy. The HG function is simply the equation of an ellipse in polar coordinates centered at one focus. It can be used to simulate scattereing with the primary scattering is in the backward direction ($g > 0$) or the forward direction ($g < 0$) [2]

2.3 Optical Depth

A term widely used in atmospheric optics is *optical depth*, which is applicable to any path characterized by an exponential law.

Optical depth for a given path can be derived as the integral of the scattering coefficient of all subelements ds of the given path.

$$T = \int_S \beta(s) ds \tag{2.7}$$

Where $\beta(s)$ is the scattering coefficient (combined Mie and Rayleigh total scattering coefficients) that varies from day to day and with altitude.

Optical depth can be used directly to calculate the attenuation over a path. Given an incident spectral distribution I_0, the attenuated spectral distribution I arriving at the observer after passing the atmosphere with the optical depth T is given by.

$$I = I_0 \cdot e^{-T} \tag{2.8}$$

Optical depth is a measure of the amount of atmosphere penetrated along a given path. This means that optical depth is a result of the length of the path and the average atmospheric density along the path.

Optical depth can be separated into the molecular (Rayleigh) and aerosol (Mie) optical depths.

2.4 Sky Color

The color of the clear sky has always amazed people. The blue color of the sky is a result of inscattering of light from the sun. The molecules of the atmosphere scatter light according to the Rayleigh theory and, as a result, show a strong tendency to scatter light in the purple and blue spectrum and less in the yellow and red spectrum. The changed spectral distribution, combined with the fact that our eyes are less sensitive to purple light [17], results in the clear blue color of the sky. The color of the sky changes with the amount of dust and water dissolved in the air. These aerosols scatter light according to Mie scattering theory and affects the total scattering of the sunlight to change the color of the sky.

2.4.1 Variations in Color over the Sky Dome

The change in intensity and color of the sky is complex. It is a result of scattering as described by Rayleigh and Mie scattering theory, but is

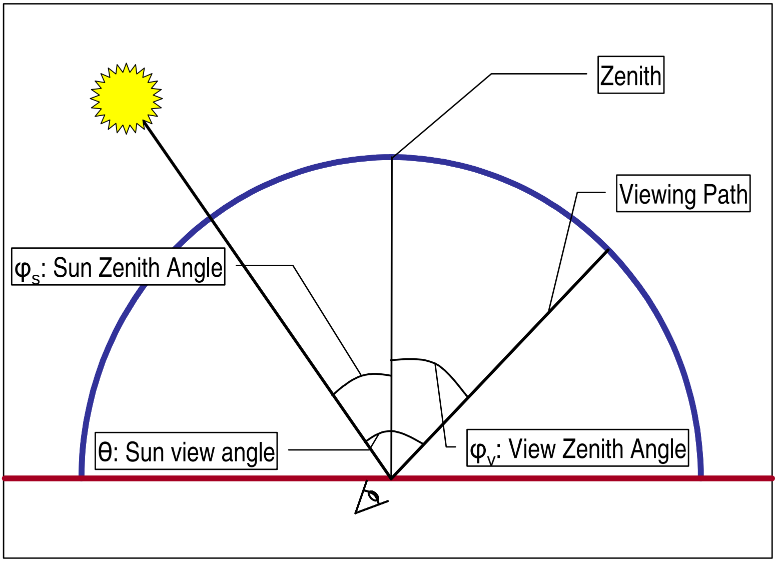


Figure 2.6: *The angles on the skydome as they are used in this thesis. A vertical angle refers to the angle from one point to another through the zenith point.*

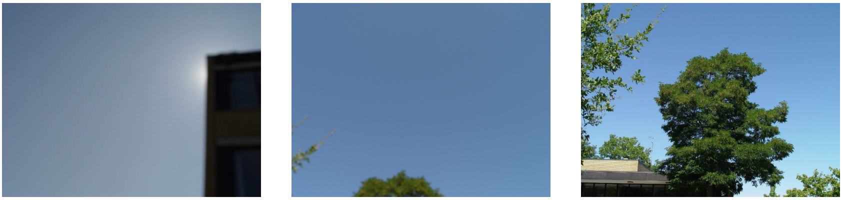
complicated by the fact that the atmosphere is lit, not only by the sun, but also by self illumination (multiple scattering see figure 2.1).

The darkest part of the sky is always found at a point on the vertical circle (through zenith figure 2.6) from the sun at an angle (θ figure 2.6) of 95° from the sun at sunrise and sunset and at 65° when the sun is high in the sky [17]. The dark part divides the sky into the bright region surrounding the sun and another bright region opposing it. This is a result of the Rayleigh phase function (2.1)[2]. The intensity of the bright region surrounding the sun are much brighter than the opposing region and can be dazzling. The distribution and definition of these regions vary with the position of the sun and with the amounts of Mie scatters in the atmosphere. It can be described as an interchange of 3 effects.

1. The intensity of the sky increases rapidly towards the sun while, at the same time, becoming whiter (figure 2.7(a)).

---

[2] This effect is only noticeable on very clear days, and even then, few people notice it unless they know what to look for.



(a) Bright regions surrounding the sun (b) Darkest and ”bluest” part of the sky (c) Sky gets brighter and less blue close to the horizon

Figure 2.7: *The color of the sky can bedescribed as an interchange of three major effects.*

2. At an angle 90° from the sun, the sky is usually darkest while the blue color is richest (figure 2.7(b)).
3. The intensity of the skylight increases toward the horizon and the deep blue color changes and becomes whiter (figure 2.7(c)).

All three effects combine to give the color of the sky. In addition, the amount of aerosols in the atmosphere influences the result, making it impossible to find two days with identical color distribution of the sky.

The first effect is a result of scattering by aerosols. The strong directional dependency of Mie scattering causes the intensity to increase rapidly towards the sun while the relative weak dependency on wavelength causes the whitening of the sky.

The low intensity of the sky color 90° from the sun is explained by the shape of the phase function for Rayleigh scattering. At an angle of 90°, the scattering is about half the scattering in the forward and backward direction. In addition, larger Mie scatterers hardly scatter any light at such a large angle.

The whitening of the sky towards the horizon is explained by the thickness of the atmosphere when the viewing direction approaches the horizon. The atmosphere scatters blue and violet a lot more than red and yellow. This could lead to the assumption that the blue and violet colors should dominate the color of the sky even more when looking through a thicker atmosphere.

This is clearly not the case, as anyone can see the sky color whitens when approaching the atmosphere. This happens because the inscattered blue

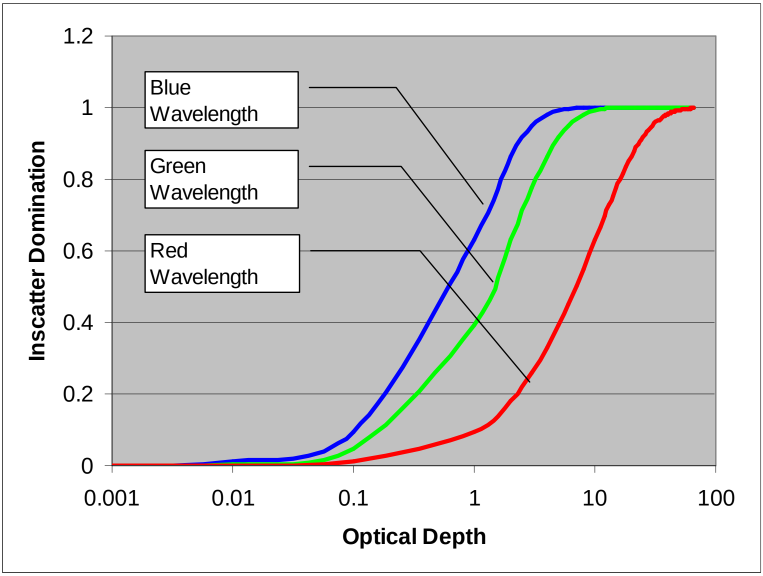


Figure 2.8: *Even though the inscattering coefficient for blue light is approximately ten times that of red, the extinction is equally larger. Consequently the total inscattering coefficient of all wavelength approaches one, or full inscatter domination, as the optical depth approaches infinity. Because the optical depth close to the horizon is big enough to resemble infinity, the color of the sky close to the horizon is the same as the color of sunlight.*

light has a much larger probability of being scattered out again (figure 2.8). As a result, the sky color will converge towards the color of a white sheet of paper (the color of sunlight) when the optical depth is large enough [17]. This explains why the horizon becomes yellow, or even orange, at sunset and sunrise.

Figure 2.9 demonstrates how the blue, shorter wavelengths dominate the inscattered spectrum at smaller values of optical depth, but that, as the optical depth approaches infinity, the spectral distribution will approach that of the sun. If the optical depth of a given path corresponds to the distance from plane A to B, the strong scattering of blue light dominates, but if the depth is increased to cover the distance from A to E, the scattering of blue light from plane D and E never reaches the observation plane and

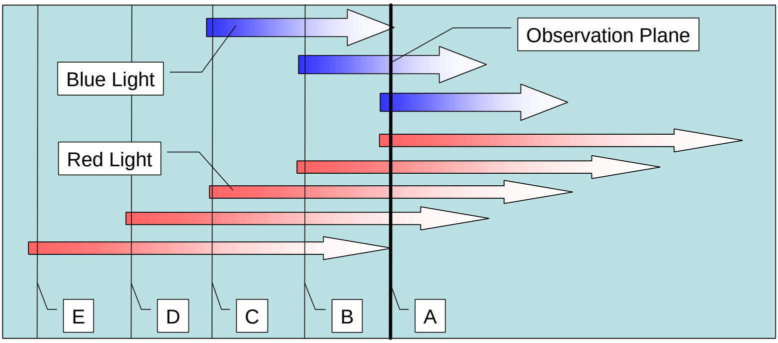
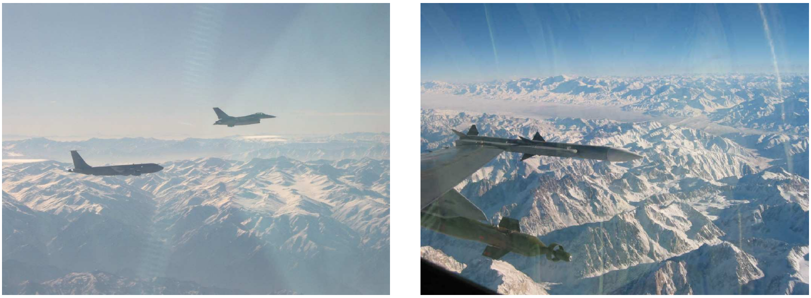


Figure 2.9: *Blue light dominates the individual scattering events, but red light penetrates deeper. Consequently, for an observer at A, the strong blue light scattered at A,B and C is seen, but because the red light penetrates deeper, contributions of red are received from the same points as the blue but also from D and E.*

make no contribution. This evens the amount of blue and red light reaching the observer and, when expanded to cover the entire spectrum, the result is that the spectral distribution will resemble that of the scattered light (sunlight).

In addition to the scattering colors, the ozone layer is important in understanding the color of the sky. Ozone has a true blue color caused by absorbtion, not scattering [17]. The faint blue color of the ozone layer is especially important when explaining the color of the sky after sunset. If only scattering was responsible for the color of the sky, the area around zenith would become gray or even yellow at sunset. This blue contribution of the ozone color is less important during daytime because the intensity of the scattered light dominates [17].

The color of the sky changes from day to day and is a result of the change in composition of the atmosphere. Aerosols make the sky whiter and increase the intensity of the light scattered from the sky. This is a result of Mie scattering, as indicated by the white as opposed to blue color. The blue color of the sky is richest when seen between rain clouds. This is because the rain cleans the atmosphere, thus minimizing Mie scattering. In addition, the blue color is richest at sunrise and sunset when zenith is at a vertical angle 90° from the sun.



(a) Image of the environment on a hazy day (b) Image of the environment on a clear day

Figure 2.10: *The effect of the aerosole concentration on the inscattered color and visibility*

2.5 Areal Perspective

Areal perspective is the effect that blurs distant objects and makes them blend in with the background. In addition to this extinction effect, the color of objects far away is attenuated and becomes faintly blue. This is the result of inscattered light, and the color varies the same way as the color of the sky. On days with few aerosols the color is blue, and on hazy days the color is more white or even yellowish (see figure 2.10).

The shift toward blue is most noticeable on dark or shadowed objects. This is so, because the scattering effects both add and remove light, and because these two effects in principle counter each other. This is the same effect that causes the horizon to become white. So the white light leaving the top of a snow covered mountain will have some of the blue light removed, but the light added by inscattering will also be mostly blue. In contrast, a dark surface such as the side of a cliff unlit by the sun emits very little light, so what we primarily perceive is the contrast (the lack of light) plus the inscattered light. This inscattered light will be mostly blue and the cliff will seem blue when seen from a distance. In principle we are observing the color of the atmosphere on a dark background, which is essentially the same as the color of the sky seen against the black background of space.

The color of bright objects, like cumulus clouds and snowclad mountains, is also attenuated by areal perspective. However, the effect is much more limited due to the reasons described above and because the change in brightness is much less noticeable. For bright objects, the shift is not towards blue but towards yellow. This is because the outscattering or extinction of the blue light is stronger than the inscattering of blue. The net result is that the blue part of the white light is weakened and the color shifts towards yellow. On hazy days, when the amount of aerosols in the air is high, objects seam to lose color and take on a more grey tint.

Areal perspective is logically divided into an extinction part and an addition part.

$$L(s,\theta) = L_0 F_{ex}(s) + L_{in}(s,\theta) \tag{2.9}$$

Equation (2.9) is a formal description of the principle areal perspective where, L_0 represents the light leaving an object, F_{ex} the extinction factor, L_{in} the inscattered light, θ the angle between the viewing ray and the sun and s is the optical depth between the object and the eye point.

2.6 Visibility

The conditions of the atmosphere have a pronounced effect on our ability to distinguish distant objects. Visibility varies daily with the amount of dust and moisture in the air.

Moisture condensates on the dust particles and the resulting droplets scatter light [17]. From this, it is clear that both the amount of dust and the humidity are important to the visibility. Atmospheric dust comes from a variety of sources; from volcanoes to polluting exhaust gasses, from industry and transport. In maritime regions salt particles dispensed into the atmosphere by the surf are often the primary contributor.

Variation in visibility is traditionally described using the parameter turbidity. Turbidity is a measure of the clearness of the atmosphere and describes the haziness of a given day. It relates the amount of scattering due to aerosols to the amount of scattering due to molecules, or, the amount of Rayleigh scattering to the amount of Mie scattering. More formally, turbidity is the ratio of the optical thickness of the atmosphere on a given day (molecules and aerosols) to the optical depth of a theoretical unpolluted atmosphere, consisting exclusively of molecules. This relationship is expressed as.

$$T = \frac{t_m + t_a}{t_m} \tag{2.10}$$

Where T is the turbidity and t_m is the vertical optical thickness of a pure molecular atmosphere and $t_m + t_a$ is the vertical optical thickness of the combined atmosphere of molecules and aerosols [20].

Since scattering varies with wavelength, it follows that turbidity varies with wavelength as well. For optical applications turbidity is measured at 555 nm [20].

Turbidity can be estimated using meteorological range. Meteorological range is the distance under daylight conditions at which a black object is visible against the background. It is roughly the same as the distance to the most distant visible geographic feature. Although meteorological range is somewhat a simplification of turbidity, it is very useful because it is easy to determine. With respect to graphics, it is even more useful because it is inherently related to the visual impression of the atmosphere, and because local data is available in airfield meteorological observations METAR's, which could be used to pull real time weather information for the simulated environment.

2.7 Summary

In the preceding, the theory of atmospheric scattering has been outlined briefly and its influence on the color of the sky and areal perspective has been described.

The most important of these are:

- The blue color of the sky is caused by the wavelength dependency of Rayleigh scattering that favorites the shorter blue wavelength.
- When the optical depth approaches infinity, the inscattered color approaches the color of sunlight. This is why the horizon is white during the day and red/orange during sunset and sunrise.
- Atmospheric scattering is divided into Mie and Rayleigh scattering, governing the scattering of particles(aerosols) and molecules respectively.
- Rayleigh scattering, with equal scattering in the forward and backward directions, is a subset of the far more complicated Mie scattering.
- The Mie scattering phase function can be approximated, using the Henyey-Greenstein phase function.

- Areal perspective is the attenuation and extinction of distant objects and is important for the human ability to asses distances.

Chapter 3

Rendering Scattering Effects

Atmospheric scattering effects result in spectacular visual phenomenons, ranging from the deep blue color of the sky on clear days over the sometimes amazing colors of the sunset to the coloring of distant objects.

These effects have long been the target of computer graphics researchers' attention. Many people have formulated solutions to the problem of rendering these effects. All of these methods are based on solving the scattering equation, and thereby determining the spectral irradiance at the observation point.

3.1 Basic Problem

Solving the scattering equation for a path in the atmosphere is a complicated problem. The problem is divided into rendering of the sky color (figure 3.1) and simulating areal perspective (figure 3.2).

In the following, the basic problem of visualizing the effects of atmospheric scattering and its application to sky color and areal perspective is described.

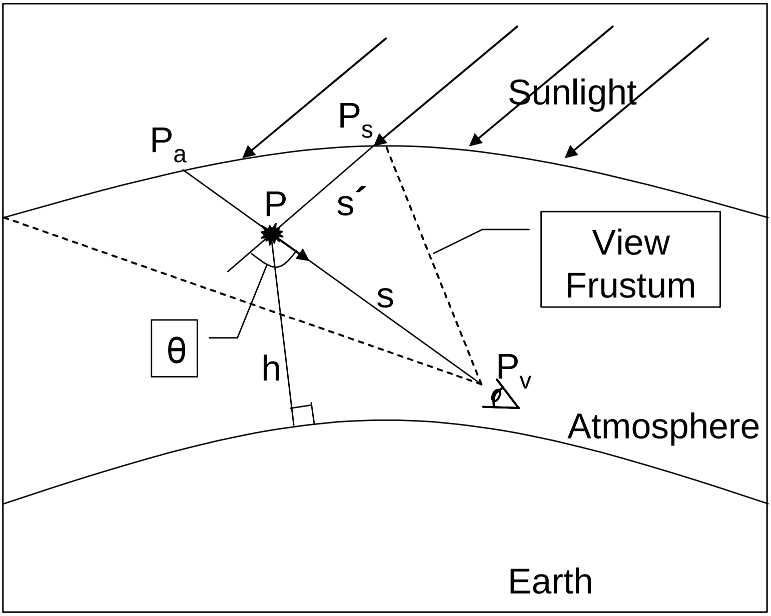


Figure 3.1: *Single scattering of sunlight in the atmosphere*

3.1.1 Sky Color

Determining the spectral distribution of light (the sky color) incident on the eyepoint of an observer positioned at P_v (figure 3.1) requires integrating along the viewing path $P_v - P_a$.

For each point P along the path the single scattering equation need to be evaluated resulting in the following integral [18].

$$I_v(\lambda) = \int_{P_v}^{P_a} I_{sun}(\lambda) \cdot F(\lambda, s, \theta) \cdot e^{(-t(s,\lambda)-t(s',\lambda))} ds \tag{3.1}$$

Where $I_{sun}(\lambda)$ is the incident intensity of sunlight at the given wavelength on the atmosphere, and $F(\lambda, s, \theta)$ is given by.

$$F(\lambda, s, \theta) = \beta_R(\lambda) \cdot \rho_R(s) \cdot \beta_R(\theta) + \beta_M(\lambda) \cdot \rho_M(s) \cdot \beta_M(\lambda, \theta) \tag{3.2}$$

Where $\beta_R(\lambda)$ and $\beta_M(\lambda)$ are the Rayleigh and Mie total scattering coefficients (see sections 2.1 and 2.2), ρ_R and ρ_M are the density of Rayleigh and

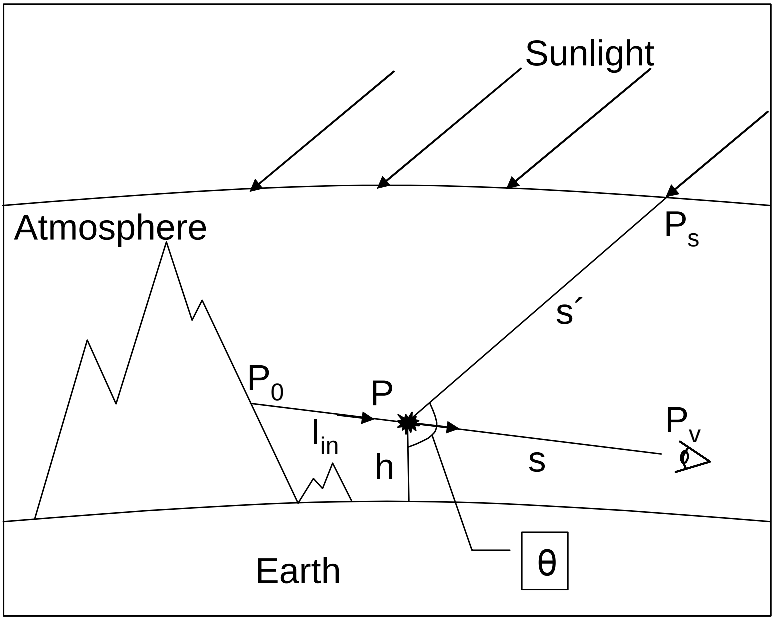


Figure 3.2: *Attenuation of distant object by atmospheric scattering (Aral Perspective)*

Mie scatterers. The value for the molecular density distribution can be accurately approximated[1] using an exponential function. $\rho_R(h) = \rho_0 \cdot e^{-\frac{h}{H_{scale}}}$ with a scale height H_{scale} of roughly 8300 m. $\beta_R(\theta)$ and $\beta_M(\lambda, \theta)$ are the scattering phase functions. The Mie phase function varies with the ratio of particle size to wavelength. Consequently, for a a given particle size, the shape of the phase function will vary for differen wavelengths.

The last part of (3.1), $-t(s, \lambda)$ are the optical depth of the paths s and s'. As described in section 2.6, it is found by integrating the total extinction coefficient over the path. It can be written as.

$$t(s, \lambda) = \beta_R(\lambda) \int_0^s \rho_R(l) dl + \beta_M(\lambda) \int_0^s \rho_M(l) dl \tag{3.3}$$

---

[1] The actual density vary slightly from the pure exponential due to temperature dependency, see figure C.1 page 122

When substituting equation (3.3) into (3.1), the result is a double nested integral. This problem can not be solved analytically and a numerical solution is needed.

Equation (3.1) is only valid when multiple scattering and scattering of light reflected from the earth are ignored. Both second order scattering and inscattering of light from the ground have measurable influences on the final result and can not be completely ignored if the solution have to be physically correct. Adding second order scattering requires solving equation (3.1) integrated over the total solid angle at every point P along the viewing path.

3.1.2 Areal Perspective

The problem of areal perspective is quite similar to the problem of sky color. The difference is that the radiance at the initial point P_0 (figure 3.2) is different from zero.

This changes equation (3.1) to the following form.

$$I_v(\lambda) = I_0(\lambda) \cdot \int_{P_v}^{P_0} e^{-t(s,\lambda)} ds + \int_{P_v}^{P_0} I_{sun}(\lambda) \cdot F(\lambda, s, \theta) \cdot e^{(-t(s,\lambda)-t(s',\lambda))} ds \tag{3.4}$$

Where $\int_{P_v}^{P_0} e^{-t(s,\lambda)} ds$ is the extinction coefficient along the path $P_0 - P_v$ and the second integral is equivalent to equation (3.1).

$I_0(\lambda)$ is the spectral distribution of light leaving P_0. I_0 is a result of the irradiance at P_0 which is a combination of direct sunlight, skylight and light reflected from the earth. The irradiance is multiplied by the BRDF[2] of the material at P_0 giving the radiance.

Establishing the spectral distribution of the incident light from the sky color involves integrating equation (3.1) over the hemisphere, limited by the object surface normal plane at P_0. Contributions from earth reflection could be included by using equation (3.4) in the cases where the path from the integration intersects the earth.

---

[2] The term BRDF stands for *Bidirectional Reflectance Distribution Function.* It is a function that describes how light is reflected of a surface. The result from a BRDF is a unitless value that is the relative amount of energy reflected in the outgoing direction, given the incoming direction [1]

Range Based Fog

Traditionally, areal perspective has been simulated using a linear range based interpolation between the object color and a predetermined fog color.

This approach is unable to realistically simulate areal perspective. Both inscattering and outscattering are interpolated using the same factor. As a result, the attenuation is independent of initial color and the initial shift toward blue, and then white[3], of Rayleigh inscattering is missed. Since the method is based on the range between the viewpoint and the individual vertices it is impossible to simulate the effects of observing terrain with varying height. This is because the interpolation factor can not be adjusted on a per vertex basis.

Last, the method is unable to capture the strong directional dependency of Mie scattering, which is clearly observable when the sun is low in the sky. It would be possible to adjust the interpolation factor on a per frame basis, but to capture this effect, it is necessary to adjust the factor within a single frame or scene.

3.1.3 RGB Color from Spectral Distribution

The eye works by having three different receptors in the eye. Each type of receptor reacts to different wavelengths, sending its signal to the brain. This means that the brain only receives three different signals for any color. This is why three colors can be used to simulate any color the eye can see.

To determine the intensity of each of the three lights, some sort of matching function is needed. The CIE XYZ functions (figure 3.3) are an example of such functions.[4] The XYZ functions have been constructed so that any color can be matched by a linear combination of the three functions. The graphs can be used to calculate the XYZ tristimulus values, based on a spectral distribution $C(\lambda)$. By multiplying the spectral distribution with the color matching functions, and numerically integrating the result

---

[3] As optical depth approaches infinity, the inscattered color approaches the color of a white piece of paper illuminated by sunlight [16].

[4] Color matching functions are designed by conducting experiments on a large number of people, having them match a spectral distribution color by adjusting the intensity of three monochromatic lights blended together [12].

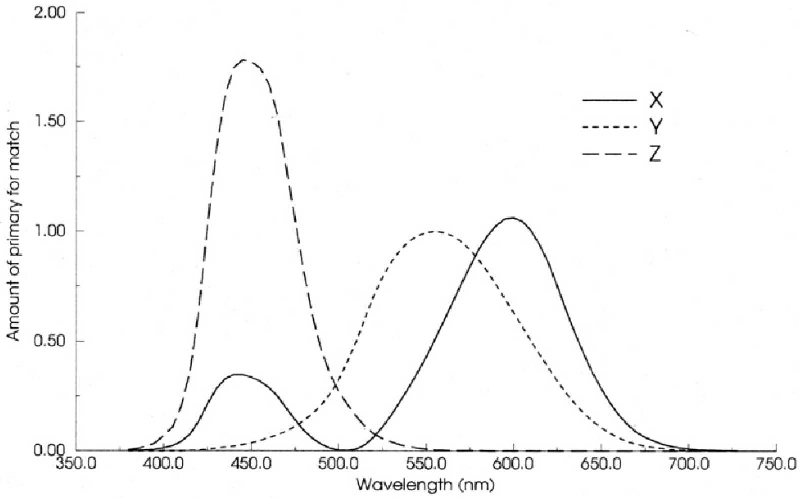


Figure 3.3: *CIE XYZ Color matching functions*

for the sampled wavelengths, a value for each of the three graphs can be determined.

$$
\begin{aligned}
X &= \int_{380}^{780} C(\lambda)\overline{x}(\lambda)d\lambda \\
Y &= \int_{380}^{780} C(\lambda)\overline{y}(\lambda)d\lambda \\
Z &= \int_{380}^{780} C(\lambda)\overline{z}(\lambda)d\lambda
\end{aligned} \tag{3.5}
$$

Where $\overline{x}(\lambda)$, $\overline{y}(\lambda)$ and $\overline{z}(\lambda)$ are the three functions X, Y and Z (figure 3.3).

In practice, both the color matching functions and the spectral distribution of light are sampled at different wavelengths and stored in data tables. For most applications, it is necessary to store the matching functions at 5 or 10 *nm* intervals.[12]

Using these tables the resulting XYZ values can be evaluated, using numerical integration over the finite $5 - 10nm$ intervals.

$$
\begin{aligned}
X &= \sum_i C_i \overline{x}_i \Delta\lambda \\
Y &= \sum_i C_i \overline{y}_i \Delta\lambda \\
Z &= \sum_i C_i \overline{z}_i \Delta\lambda
\end{aligned}
\tag{3.6}
$$

Where i depends on the number of intervals, but should cover the visible spectrum from $380nm$ to $780nm$.

The XYZ and RGB color systems are simply related by a linear transformation which can be written in matrix form as

$$
\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}
\tag{3.7}
$$

Where the values with the $_r$, $_g$ and $_b$ subscripts are based on the tristimulus values of the red, green and blue phosphorous respectively. This equation assumes a linear response of the monitor phosphors to input voltage. This is in general not true, but is corrected by gamma correction.

There is a physical relationship between the input voltage and the resulting brightness I of a pixel on a CRT (cathode ray tube) monitor. This is expressed as [1].

$$
I = a(V + \epsilon)^\gamma
\tag{3.8}
$$

Where V is the input voltage, a and γ are constants that depend on the monitor. ϵ is the black level setting for the monitor. A gamma correction value γ of 2.5 is usually considered adequate, but individual monitors will produce different results when using the same gamma value.

In computer graphics, lighting equations compute intensity values that have a linear relationship to each other. This means that a value of 0.5 should be perceived as half as bright as a value of 1.0. Achieving this requires gamma correction. If gamma correction is not applied, a value of 0.5 will be perceived as considerably less than half the intensity of 1.0.

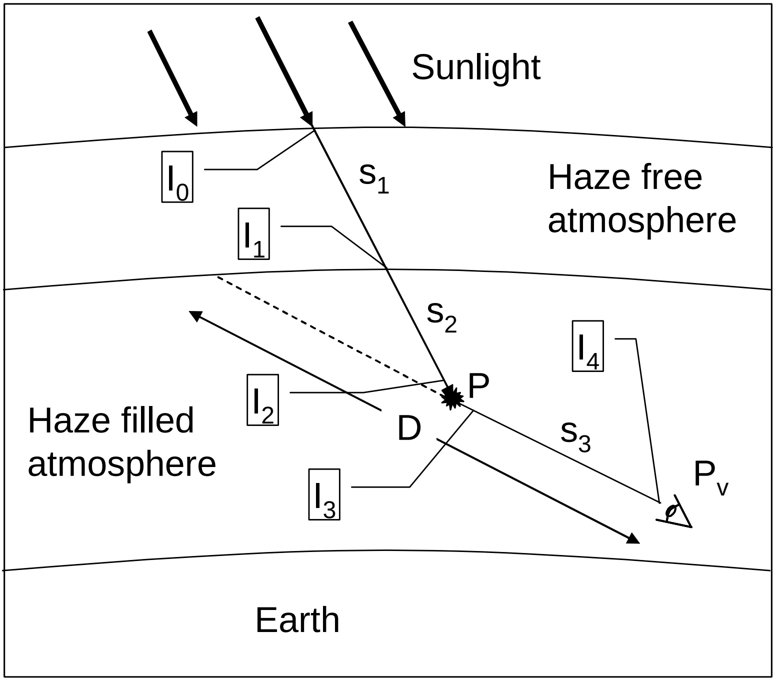


Figure 3.4: *Klassen's model of the atmosphere. He divides the atmosphere into two constant density layers; one consisting only of molecules and a second with molecules and aerosols. All scattering events are assumed to happen in the composite layer.*

3.2 Previous Work

Klassen [14] was one of the first to propose a solution to the basic problem of solving the scattering equations. In his solution, the atmosphere is modelled as two layers (figure 3.4). The top layer is treated as a pure molecular layer and the bottom layer is composed of molecules and aerosols.

Klassen assumes that single scattering events are unlikely to occur before entering the Haze filled part of the atmosphere. Using this assumption, he divides the problem by considering the intensity distributions ($I_0 - I_4$) at the transitions.

I_0 is the intensity vector of sunlight when it hits the atmosphere. This light

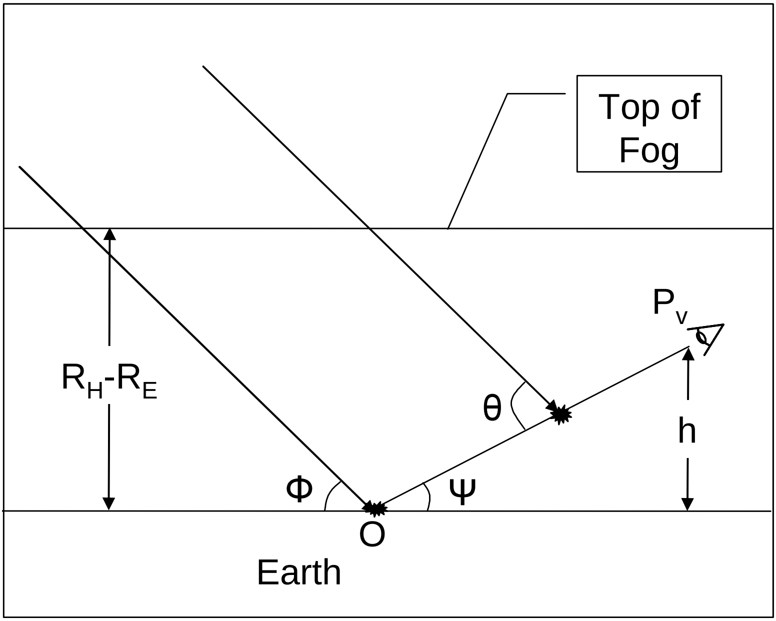
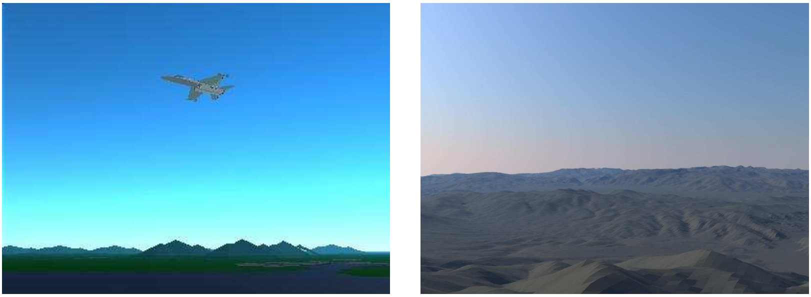


Figure 3.5: *Klassen's model of the fog layer*

is attenuated by pure rayleigh scattering over the distance s_1, resulting in the intensity vector I_1 when entering the haze filled atmosphere. I_1 is then attenuated by Rayleigh and Mie scattering over the distance s_2 resulting in the intensity I_2 before it is scattered at P. After scattering the distribution is I_3, which is again attenuated by Mie and Rayleigh scattering before reaching the eye with the intensity I_4.

Along each of the paths s_i, Klassen assumes a constant density, arguing that since the density is multiplied, a correct result can be approximated by reducing the radius of the atmospheric layers and adjusting the density accordingly. This simplifies the problem by allowing the direct calculation of atmospheric thickness $-t(s, \lambda)$. To determine the correct depth of each part of the ray, he uses geometric computations that account for the curvature of the atmosphere. Using these approximations, the scattering problem is reduced to integrating an analytical equation along the path D.

Klassen proposes modelling a fog layer as a vertical thin layer over a flat



(a) Nishitas method (b) Preethams method

Figure 3.6: *Examples of renderings produced using the methods of Nishita (a) and Preetham (b).*

earth. The problem is then again divided into a number of phases.

First, he calculates the spectral distribution of the light leaving an object. This is done essentially using the same method as for calculating the light leaving a single scattering event in his model for sky color, except that the angular scattering coefficient is replaced by an angular reflectivity coefficient, with the addition of a ambient contribution to account for indirect illumination by scattered light.

When calculating the thickness of light along the viewing ray, the fog layer is assumed to be sufficiently thin to alow multiple scattering to be neglected and to be sufficiently thick compared to the height h of the observer, in order to alow the distance light travels through the fog layer to be considered constant along the viewing ray.

For sun angles less than 15 degrees above the horizon, the thin fog layer idea is no longer valid. Instead it is assumed that the direct sunlight can be neglected and that all light has been scattered at least once. This leads to a uniform light source simulated by an ambient value.

Nishita [18] proposes a method based on exponential distributions of molecules and aerosols in the atmosphere.

The proposed system does a full simulation of the physics of light in the atmosphere, including second order scattering. The cost of such a simulation is huge, and to speed up the rendering, Nishita first precalculate the intensity distribution at a large number of voxels and store these in a data

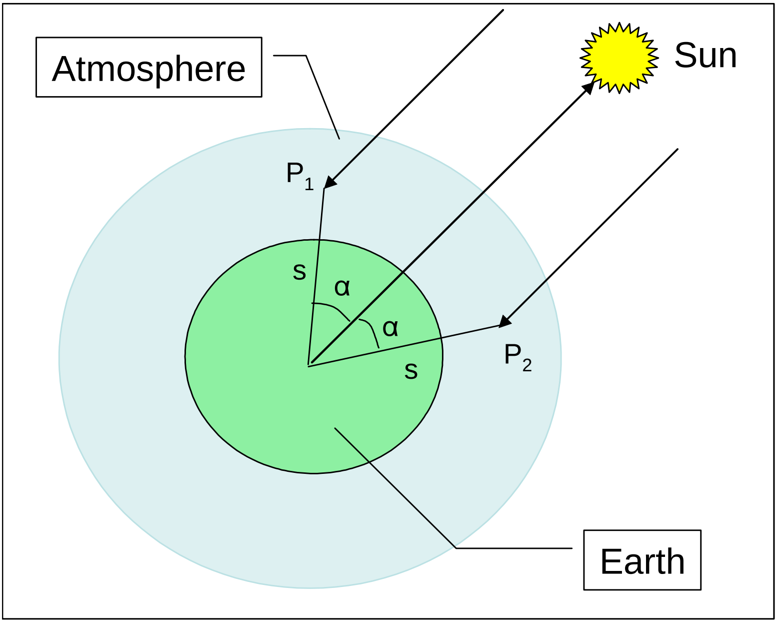


Figure 3.7: *Nishita's coordinate system. The atmospheric depth is axis symmetric. This means that in the shown reference system, the scattering at points* P_1 *and* P_2 *is identical.*

table. This is done by storing the irradiance for a number of directions (buckets). These intensities are then used when gathering the second order reflected light for a single scattering event.

To speed up the calculations of the irradiance stored in the buckets and of the optical depth along a ray from the sun to some point in the atmosphere s (figure 3.1), Nishita uses a cylindrical coordinate system (figure 3.7), where the axis is the axis from the center of the earth towards the sun.

In this system, points with the same α and distance s have equal atmospheric depth from the sun, and the results of equation (3.3) for this path is the same. Nishita uses this relation to precalculate the factors in what he refers to as a *summed shadow table*. This is later used to perform lookup, using bilinear interpolation. This, and the use of adaptive sampling for the integration along the viewing ray, results in a more effective method than those previously used.

Irwin [13] uses an exponential model for molecular density similar to that of Nishita [18]. His model only simulates Rayleigh scattering and primarily focuses on the conversion from spectral distribution to RGB colors. To get good results, he samples the spectrum at 14 different wavelengths and uses the sampled values to reconstruct the spectrum before discretizing it again at 5 nm intervals, which is suitable for conversion into the XYZ tristimulus values.

Preetham, Shirley and Smits [20] use Nishita's method to calculate skylight, ignoring ground reflection and third and higher order scattering. This was done for a variety of different sky conditions, sun positions and for 343 directions in a skydome. The data obtained from the simulation is subsequently fitted, using an analytical function originally constructed to fit the sky luminance. To account for spectral variations, a set of chromaticity variables are fitted as well.

Once the analytical functions are established, they can be used to determine the color of the sky in a given direction, but also to determine the irradiance of skylight when calculating the reflected color of an objet in the scene.

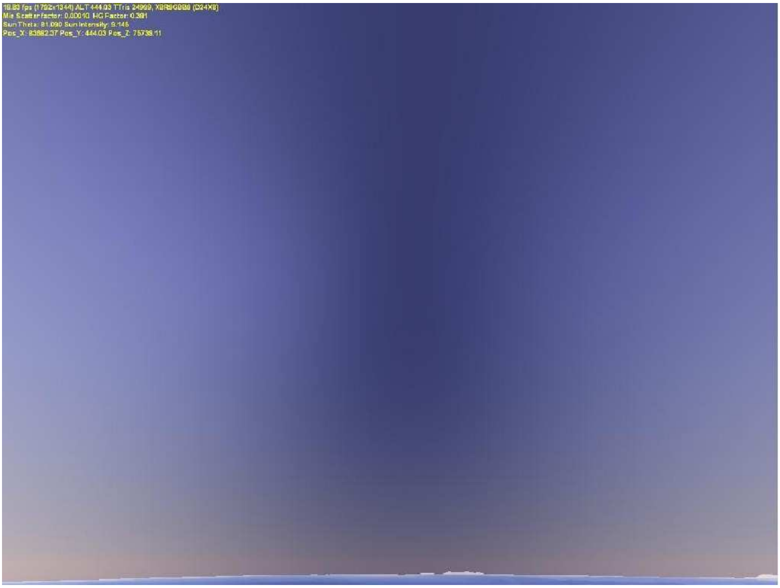
Compared to Nishita, Preetham uses a more realistic model for the density distribution in the atmosphere. A comparison of images generated by using the two systems can be seen in figure 3.6.

For their areal perspective model, they both use a more simple exponential density distribution, and they assume that the earth is flat. Using these assumptions, the integrals are simplified. Further simplification is made possible by splitting the solution into two; one for viewing rays that are almost parallel to the earth and another for viewing rays with larger altitude difference. In the first part, the atmospheric depth can be solved analytically and for the latter the solution of the ray optical depth is fitted, using a hermite cubic polynomial.

3.2.1 Real Time Approaches

Hoffman and Preetham's method [10] for simulating areal perspective is capable of compensating for many of the shortcomings exposed by traditional range based fog.

The method is based on a simplification of atmospheric scattering theory, but is limited to a constant density atmosphere. Their method is capable



```
19.83 fps (1792x1344) ALT 444.03 TTris 24999, X8R8G8B8 (D24X8)
Mie Scatterfactor: 0.00010 HG Factor: 0.391
Sun Theta: 81.090 Sun Intensity: 9.146
Pos_X: 83882.37 Pos_Y: 444.03 Pos_Z: 75738.11
```

Figure 3.8: *180 degree view of the skydome using a direct mapping of Rayleigh angular intensity to RGB. The dark band is too dark, showing that a naive direct mapping from intensity values to RGB colors results in artifacts.*

of capturing the directional dependency on the angle between the sun direction and the viewing ray. It is also capable of capturing the wavelength dependency of atmospheric scattering and will consequently capture the attenuation of Rayleigh scattering correctly.

The proposed simplification developed by Hoffman and Preetham results in the following equation which is evaluated for all vertices in the scene.

$$L(s,\theta) = L_0 F_{ex}(s) + L_{in}(s,\theta) \tag{3.9}$$

Where L_0 is the initial color of the viewing path, which is black for the atmosphere. $F_{ex}(s)$ is the extinction coefficient for the path and $L_{in}(s,\theta)$ is the inscattering, these two terms are given by.

$$F_{ex}(s) = e^{-(\beta_R+\beta_M)s} \tag{3.10}$$

$$L_{in}(s,\theta) = \frac{\beta_R(\theta)+\beta_M(\theta)}{\beta_R+\beta_M} E_{sun}(1-e^{-(\beta_R+\beta_M)s}) \tag{3.11}$$

Where E_{sun} is the RGB color of sunlight and β_R and β_M are the Rayleigh and Mie total scattering constants respectively. $\beta_R(\theta)$ and $\beta_M(\theta)$ are given by.

$$\beta_R(\theta) = \frac{3}{16\pi}\beta_R(1+\cos^2\theta) \tag{3.12}$$

$$\beta_M(\theta) = \frac{1}{4\pi}\beta_M\frac{1-g^2}{(1+g^2-2g\cos\theta)^{3/2}} \tag{3.13}$$

Their method is capable of capturing both the directional effects and the wavelength dependency, but their direct use of simplified theoretical models for the mapping of intensities to RGB colors causes some problems. The factor two reduction in Rayleigh scattering perpendicular to the sun direction causes a dark band in the atmosphere (figure 3.8). This does not resemble the real world, where the difference in intensity would usually be minimized by multiple scattering and the logarithmic mapping of intensities to display values done by the human vision system.

In addition, their method assumes an observer positioned at the ground. It does not simulate the change in sky color and intensity that appears when climbing up through the atmosphere. This allows them to use the distance between points s directly as optical depth. This would not be possible in a flight simulator system, where optical depth depends on both the length of the viewing path and the average density of the penetrated atmosphere.

The method exposes the same fundamental problem of dealing with terrain that contains significant differences in height where the average density of the viewing path cannot be considered constant, and is not directly capable of handling changes in observer altitude[5].

The main advantage of the method proposed by Preetham and Hoffman is speed. Because the scattering effects are applied using relatively simple vertex shaders, the performance penalty imposed by the system is very small and in some circumstance, for instance if the system is fill limited, no performance penalty will be present.

Dobashi, Yamamoto and Nishita [3] render atmospheric scattering effects by drawing a set of sampling planes in front of the screen. Each plane is divided into a mesh and on each vertex of the mesh the scattering contribution is found by using lookup tables sampled as textures.

---

[5] A rough solution to the problem of changing observer altitude could be contained by changing the scattering factors globally when the observer changes altitude. This would still not handle the problem of changes in terrain altitude.

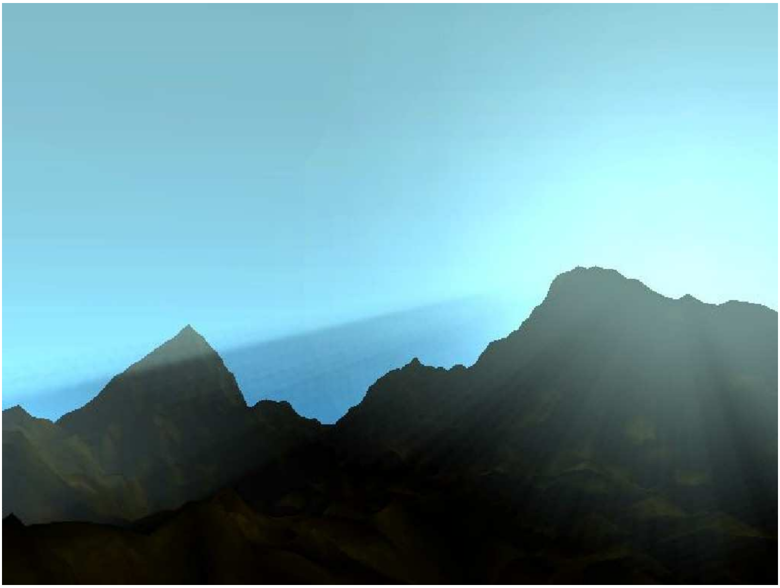


Figure 3.9: *Dobashi, Yamamoto and Nishita have developed a method for rendering atmospheric scattering effect using volumetric methods. The method is capable of capturing effects like shafts of light.*

Because the scattering equation depends on a lot of parameters, the textures need to be high 4-5 dimensional textures, which requires large amount of memory.

The earth's atmosphere is assumed to be a spherical shell, where the density of atmospheric particles and molecules decreases exponentially with altitude. To simulate multiple scattering, Dobashi, Yamamoto and Nishita include an ambient contribution.

To obtain the precise sampling of object shadows and the intensity distribution of the light source, they introduce a set of sub planes. The sub planes are needed because the final color is calculated by adding the contributions of the individual sampling planes and are, as such, subject to quantization errors. This is avoided by sub sampling only those parts of the calculation that require this, specifically the shadow boundaries and intensity distribution.

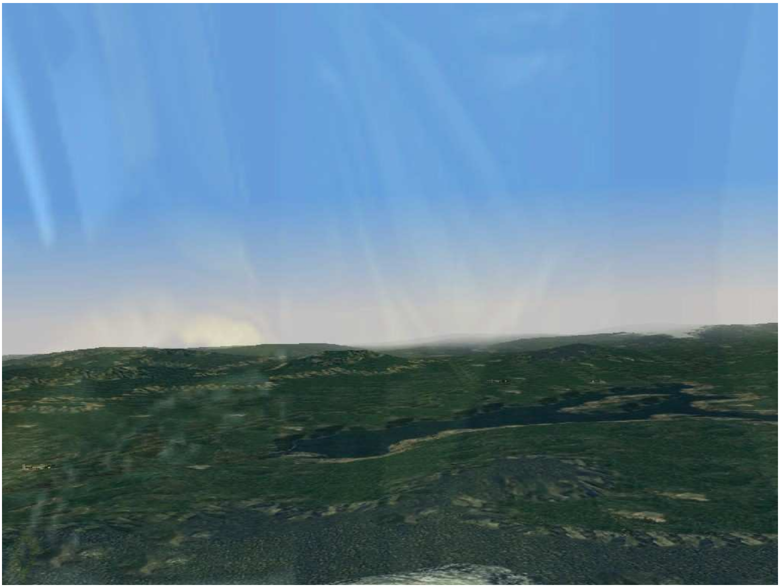


Figure 3.10: *Sky color and Areal perspective in* Falcon 4.0

The work by Dobashi, Yamamoto and Nishita is able to realistically capture volumetric effect in the atmosphere, and produces some very nice images (figure 3.9), but the method is only just able to obtain interactive framerates of around 6 fps. on modern hardware for flight simulator environments (Athlon 1.7 GHz and a Nvidia GeForce3 [3]. This is for a sample application where the system can devote all resources to the rendering of the scattering effect.

3.2.2 Classical Real Time Methods

Flight simulators previously used scripted sky color that would change at certain time intervals. E.g. in *Falcon 4.0* the changes are quite noticeable, and the color of the sky is not dependent on the direction of the sun.(figure 3.10)

The traditional method of simulation areal perspective can also be seen in that (figure 3.10) the terrain in the distance slowly fades into the sky color, using range based fog. Range based fog is based on interpolating between the background color and some preset fog color, based on the distance to the object. This interpolation can be linear or a power function.

The problem is that all three color components are attenuated an equal amount at the same distance. This is clearly wrong, compared to the atmospheric scattering functions, which clearly shows that both extinction and inscattering are wavelength dependant.

Chapter 4

Graphics Hardware and API

In this chapter, the various parts of a real time rendering system is described, specifically the advances in graphics hardware and the API's that accompany these.

I have chosen to use the DirectX 9.0 API, which was released in December 2002. DirectX is limited to the Windows platform, but has great support in the hardware community. In addition, version 9.0 of the API comes with a shader programming language named HLSL (High Level Shading Language)that eases shader development and shader debugging.

4.1 3D Graphics API

A 3D graphics API is a programming interface that acts as an interface between 3D applications and hardware drivers. An API provides a unified interface to the underlying rendering hardware and makes it easy to develop applications capable of running on hardware from different vendors.

Modern graphics API's like OpenGL and DirectX follow a vertex stream pipeline model. The application passes geometry to the API in the form of streams of vertices, containing various attributes such as position, color, normal and texture coordinates. The API is then responsible for transforming, clipping, texturing and presenting the final image of the geometry on the screen. This process is preferred to as the pipeline. To allow the application to control the stages of the pipeline, the API presents a set of render states that the application can change. Traditionally, these stages included changing the way the vertex positions are transformed, the way lighting is calculated and how textures were applied to the transformed vertices. Lately, the use of render states has been combined with the use of small configuration scripts (shaders) that allows the user to control in more detail the transformation and rastering of primitives.

Since the API is basically a frontend for the hardware, it is important to understand that if the hardware does not support a specific feature of the API, it can mean that a program trying to use those features will have a significantly reduced performance or may not run at all on that specific hardware.

4.1.1 DirectX and Direct3D

DirectX is actually not a 3D graphics API. It is more a game programming- or multimedia API, which is composed of a series of independent API's for audio, input, networking and 3D Rendering. The 3D Rendering API in DirectX is called Direct3D.

Direct3D offers objects for manipulating rendering devices, surfaces, textures, vertex and index buffers that store information about geometry data, and various helper objects in the accompanying utility library, called D3DX.

Direct3D8 was the first graphics API to introduce a programmable pipeline and an assembly-like shading language for writing vertex and pixel shaders. In Direct3D9 these programable capabilities have been expanded, providing more registers and the possibility for longer programs.

4.2 Programmable Rendering Pipeline

Two stages of the pipeline can be configured using shaders; the vertex transform stage and the pixel stage. These two parts are configured using small programs known as vertex shaders and pixel shaders.

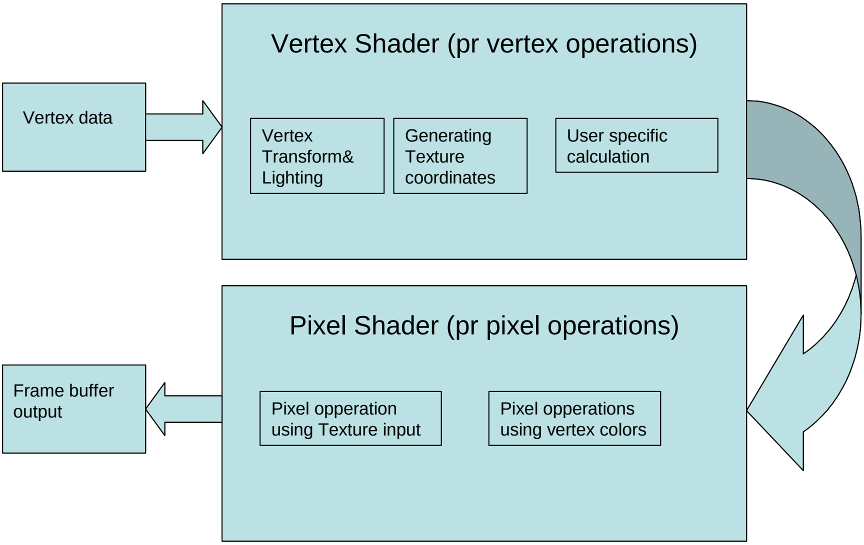

Vertex Shader (pr vertex operations)
Vertex data
Vertex Transform& Lighting
Generating Texture coordinates
User specific calculation
Pixel Shader (pr pixel operations)
Frame buffer output
Pixel opperation using Texture input
Pixel oppearations using vertex colors

Figure 4.1: *Schematics of the programmable graphics pipeline*

4.2.1 Vertex Shaders

Programmable vertex shaders replace the fixed-function transformation and lighting module in the Direct3D rendering pipeline. Vertex processing performed by vertex shaders encompasses only operations applied to a single vertex at a time.

The output of the vertex processing step is defined as individual, ready-to-rasterize vertices, each of which consists of a clip-space position (x, y, z, and w) plus color, texture coordinate, fog intensity, and point-size information. A subsequent processing stage projects and maps these positions to the viewport, assembles multiple vertices into primitives, and clips primitives. The vertex shader does not control those operations.

Direct3D includes the notion of a stream to bind data to input registers for use by shaders. A stream is a uniform array of component data, where each component consists of one or more elements that represent a single entity, such as position, normal, color etc. Streams enable graphics chips to perform a DMA (Direct Memory Access) from multiple vertex buffers in parallel and also provide a more natural mapping from application data.

The interpretation of the input vertex elements is programmed using the shader instructions. The Vertex shader function is defined by an array of instructions to apply to each vertex. The vertex outputs for vertex shaders are explicitly written by instructions in the shader. The shader functions are defined in a typeless vector language. This language is designed for 3D graphics and contains common operations such as dot product, as well as the ability to arbitrarily swizzle components i.e. reordering xyz into zyx.

Since shader functions are small programs, it is important to understand the execution environment. Shaders use input registers to access vertex data; constant registers for stored constants such as lights and matrices, temp registers to save intermediate calculations, output registers, and the array of instructions.

For the first generation of vertex shaders there were no branching operations in the vertex shader instruction set. In second generation vertex shaders, simple branching has been added and the trend of expanding programmability is likely to move ahead. Third generation vertex shaders are already defined and will bring access to more instructions as well as texture lookup in the vertex shader. Currently, no hardware exists that support third generation vertex shaders.

4.2.2 Pixel Shaders

Pixel Shaders perform color blending operations, including sampling of textures, blending of colors and textures and possibly per pixel lighting effects like normal map lighting and bump mapping.

Pixel shaders control the color and alpha-blending operations and the texture addressing operations. The result emitted by the pixel shader is the contents of the output pixel color. Whatever it contains when the shader completes processing is sent to the fog stage and render target blender for further processing.

Pixel shaders contain two types of instructions: color/alpha blending and texture addressing instructions. Pixel color and alpha blending operations modify color data, while texture addressing operations affect and process texture coordinate data.

Like Vertex Shaders, first generation Pixel Shaders contained no branching operations. And like vertex shaders the programmability of pixel shaders increase with each new generation. First generation pixel shaders had no division operator. This was added to second generation etc.

4.2.3 HLSL High Level Shading Language

With the increased programmability of vertex and pixel shaders, and the accompanying increase in the complexity of the shader programs written to control them, it would soon become cumbersome to maintain and reuse shaders. This is why Microsoft and several others[1] decided to develop a high level programming languages for shader programming. This resulted in HLSL.

The syntax of HLSL is loosely based on the C programming language with various changes or enhancements necessary due to the fundamental difference of CPU and GPU programming. The statements and operators supported in HLSL are largely the same as in C, except that most of the operators are capable of operation on vector and matrix data types in addition to the normal scalar types.

The main feature of HLSL is the improved readability of the shader programs. A detailed description of the functions and syntax of HLSL can be found in [21].

4.2.4 DirectX Effect Framework

With the release of DirectX 8.0, Microsoft introduced the effect framework. The effect framework is centered around the idea that a graphics system is composed of two different parts; programming and artwork. Because of the constraints of compilers and API, part of what should be done by artists was often done by programmers, simply because artistic parameters had to be hard coded.

With effect files, developers are given an option to allow artist to modify render states and shader code in effect scripts. These scripts are then compiled at runtime, implementing any changes. This runtime compilation allows the artist to modify artistic parameters without knowledge of programming or compilers.

---

[1] HLSL are developed in co-oporation with Nvidia. Both companies released separate languages Microsoft's HLSL and Nvidia's CG (C for Graphics)

A typical effect is composed of a set of global variables, some sampler structures, a set of vertex and pixel shaders and some techniques containing passes.

The global variables can be assigned by the effect as constants or they can be assigned by the application. For the application to assign global variables, it needs a handle to the parameter. Such handles are acquired by assigning a semantic name to the variable, which the application can inquire. Global variables could be elements like transformation matrices and light vectors.

A technique describes a single rendering of an object. A technique can be composed of one or more passes if multi pass rendering is needed. An effect file can contain multiple techniques to accommodate different levels of detail or to match the capabilities of different hardware.

The DirectX API contains function for interfacing and compiling the effect files, and correct use of these options makes it possible to simplify the structure of the underlying render engine of the rendering system, while, at the same time, increasing the freedom of the content artists.

Chapter 5

Hypothesis

The hypothesis is meant to outline the ideas for solving the problem described in chapter one. The proposed ideas must fulfill the requirements described there, regarding both rendering quality and speed.

The problems of simulating sky color and areal perspective in a real time flight simulator environment are tightly connected. This connection becomes even more clear when the models used are based on the theory of atmospheric scattering, which is the cause of both phenomenon.

The proposed method can be logically divided into rendering of the skydome and rendering of the terrain. The method has to be appropriate for real time application, where GPU recourses have to be available for rendering the rest of a complex environment, as well as for the rendering of the terrain and sky.

5.1 Summary of Previous Methods

Three basic approaches exist when developing a real time rendering system to simulate the effects of atmospheric scattering. They are described in the sections 3.2.1 and 3.2.2.

It was determined in the problem description that the classical methods are insufficient and lack the possibility to simulate important effects, like the wavelength dependency of atmospheric scattering and the directionality of Mie scattering. Consequently, two realtime methods exist, which are candidates for the use in a flight simulator rendering system.

The method developed by Dobashi, Yamamoto and Nishita is capable of rendering volumetric effects, like shafts of light and cast shadows. Their Method is superior to the method proposed by Preetham and Hoffman in terms of rendering quality, because it essentially uses a simplified numerical integration technique. However, the method is very demanding in terms of rendering recourses, and it is not really capable of achieving real time rendering speeds on modern hardware.

Given the rendering quality of Dobashis method and the fact that it is almost capable of real time frame rates in stand alone systems, it could be a candidate for an extremely realistic rendering systems in a few years, when rendering hardware has evolved over a generation or two, but for the problem described in section 1.2 it is still too slow.

This leaves the method proposed by Hoffman and Preetham [10]. Their use of a simple analytical model is capable of capturing both the directional effects and the wavelength dependency of atmospheric scattering, without compromising rendering speeds. The method is, however, not directly portable to a flight simulator environment, as it does not take the decrease in atmospheric density with altitude into account, and contains some problems because of direct mapping from radiometric quantities to RGB colors.

While the method of Hoffman and Preetham is not directly usable in a flight simulator environment, it still presents the best compromise between rendering quality and speed. In the following, improvements to their system is proposed, which will compensate for the shortcomings and make the system usable in a flight simulator environment.

5.2 Scattering Models

All the atmospheric effects included in the system originates from atmospheric scattering. Atmospheric scattering deals with radiometric intensities and can not be directly mapped to RGB colors (see section 3.1.3). This is well suited for non real time rendering systems like photon mapping and radiosity. In order to convert the calculated radiometric distribution into display colors, a tone mapping operator is applied. Tone mapping operators are often non linear, especially for the high dynamic range (HDR) scenes of an outdoor environment.

Such an approach introduces a significant overhead, and are not well suited for a real time environment. Some researchers [7] have developed interactive rendering systems, using (HDR) tone mapping, but they are only slowly approaching real time performance and would not leave resources for the rest of the simulation in a flight simulator environment.

In this work, the focus has been on capturing the perceived effects of atmospheric scattering rather than on simulating the physics. Therefore the atmospheric scattering equations are used as the basis of a system that is modified to produce the desired effect.

The proposed system will be based on the tree RGB colors and all calculations will be done using RGB vectors. This means that the system is not based on a full spectrum and that some adjustments might be needed to account for the missing information.

In order to use the method proposed by Hoffman and Preetham [10], the equations (3.11) have to be modified to compensate for the variation in atmospheric density with altitude. To do this, the distance parameter s in the extinction terms is replaced by a optical depth parameter, which varies with distance, average density and Mie scattering concentration. This optical depth is calculated using the assumption that the optical depth of a given path can be estimated as the length of the path multiplied by the average atmospheric density along the path.

Assuming different variations in the density of molecules and particles, the modification results in the following equation.

$$L(s,\theta) = L_0 \cdot e^{-(\beta_R S_R + \beta_M S_M)} + \frac{\beta_r(\theta) + \beta_M(\theta)}{\beta_R + \beta_M} E_{sun}(1 - e^{-(\beta_R S_R + \beta_M S_M)}) \quad (5.1)$$

Where S_R and S_M indicates the optical depth of the viewing ray resulting from Rayleigh scattering and Mie scattering respectively. These factors are calculated on a per vertex basis, using simplified models for atmospheric density, which will be described in detail in the chapters on sky color and areal perspective.

5.2.1 Rayleigh Scattering

If the intensity distribution of the Rayleigh angular scattering equation is used directly, the resulting hue of the sky is greatly exaggerated (figure 3.8) resulting in a dark band perpendicular to the sun.

To compensate for this effect, the Rayleigh scattering equation (2.1) is altered slightly, reducing the directional dependency of the rayleigh scattering equation. This is needed to compensate for the lack of tone mapping.

Tone mapping operators for HDR imagery are logarithmic [6, 7]. This corresponds to the logarithmic mapping of intensities happening in the human vision system. It reduces the contrast between bright and dark areas to values displayable on a computer screen. The idea is now that to avoid artifacts as those displayed in figure 3.8, the contrast of the Rayleigh phase function should be reduced.

The new relationship used is determined by carefully considering the directional part of the rayleigh scattering phase function. To reduce the directionality of this function, it is necessary to reduce the effect of the $\cos^2(theta)$ part. This is done by multiplying a reducing factor $\frac{1}{2}$ to the $\cos^2(theta)$ part. The size of this reductional part is estimated and then adjusted when the effect of the estimate is seen in the final result. This is in no way physically correct, but allows for an improved visual realism.

The adjustment is simple and results in the following expression for the rayleigh scattering phase function.

$$\beta_R(\theta) = \frac{3}{16\pi}\beta_R\left[2 + \frac{1}{2}\cdot\cos^2\theta\right] \tag{5.2}$$

This reduces the two to one relationship between parallel and perpendicular scattering (figure 2.3), to a 1.25 to one relationship (figure 5.1). This is much more in line with the perceivable difference between the intensity of the sky looking away from the sun, opposed to perpendicular on the sun direction.

β_R is a vector containing the total scattering factors of the 3 RGB colors. It is sampled at 650 nm(R), 610 nm(G) and 475 nm(B)[1] and calculated using (2.2), the $\frac{1}{\lambda^4}$ dependency on wavelength results in the factor of the blue component being approximately ten times greater than the red component.

---

[1] These values are from [10]. however the value of the green component has been adjusted, based on empirical observations.

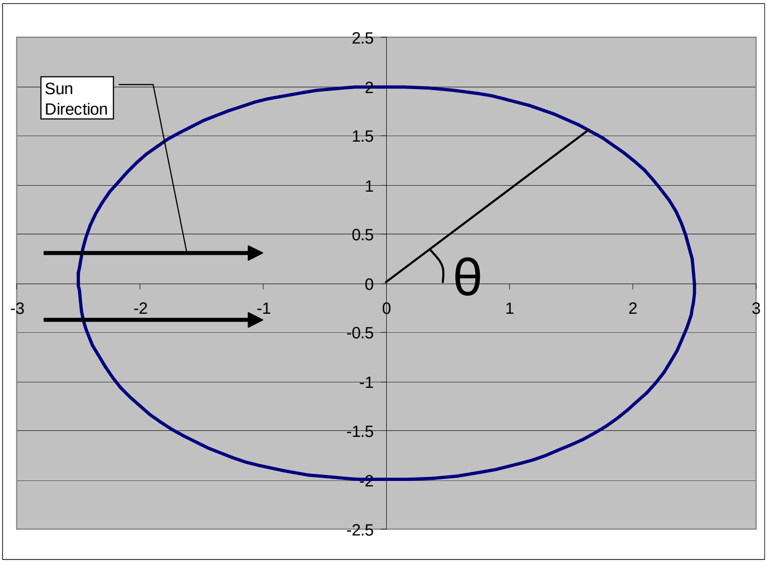


Figure 5.1: *Modified Rayleigh phase function. The directional dependency is greatly reduced compared to the real rayleigh phase function.*

5.2.2 Mie scattering

Mie scattering, like Rayleigh scattering, is calculated for each of the three RGB colors. The Henyey-Greenstein phase function (2.6) is used to approximate the phase function of Mie scattering, resulting in the following Mie angular scattering equation.

$$\beta_M(\theta) = \frac{1}{4\pi}\beta_M \frac{1-g^2}{(1+g^2-2g\cos(\theta))^{\frac{3}{2}}} \tag{5.3}$$

Where β_M is given by (2.5)

Mie scattering[2] is used to model haze, this means that the optical depth of a given path changes with the change in the amount of particles suspended in the atmosphere. When the amount of particles changes, it results in a change in visibility and the color of the sky. The color and intensity of incident sunlight are altered as well. A model is developed that adjusts these values based on turbidity[3], observer altitude and sun altitude.

---

[2] Mie scattering covers scattering of light by both particles and molecules, but are in the proposed system used exclusively to describe scattering by haze particles.

To keep the model usable in a real time environment, Multiple scattering is not included in the calculation. This is no problem in clear conditions, but might cause problems in hazy conditions, where multiple scattering becomes more significant. To compensate for this, the shape of the Mie scattering phase function (2.6) is adjusted when the amount of Mie scattering is increased. At the same time, the intensity of the sunlight is adjusted to compensate for the increased extinction of the sunlight that results from increased haziness. These two adjustments combine to give a rough simulation of the effects of multiple scattering.

Automatic Adjustment of Scattering Parameters

In the previous method by Preetham and Hoffman, the scattering parameters[4] had to be adjusted by the user for individual scenes and individual viewpoints. This conflicts with the idea of interactivity. The problem is that the parameters has to be adjusted for varying observer positions and sun positions. The proposed solution is to make this adjustments automatically, using a linear fit to interpolate between a set of predetermined values.

Because the directional parameter g has to be adjusted based on both observer altitude h and sun position θ, the linear equation is given by.

$$g = a + bh + c\theta + dh\theta \tag{5.4}$$

where a, b, c and d are determined experimentally.

This takes care of the dependency on observer altitude and sun position. However, the amount of Mie scattering also plays an important role for the required value of g. This dependency is covered by calculating g using equation (5.4) for the maximum and minimum Mie scatter factors used. Subsequently, the current value is determined from a linear interpolation between the two extremes, based on the current Mie scatter factors.

---

[3] In this model, Turbidity is not used directly. Instead, a Mie scattering factor is used. This would be proportional to the turbidity (2.4).

[4] The directional factor in the HG function and the intensity of sunlight.

The intensity of sunlight is adjusted based on the Mie scatter factor, using a simple linear equation, which, like the directional factor, is determined experimentally.

It is important to emphasize that the parameters used are determined by subjectively choosing the best parameters based on the visual appearance of the test scenes.

The process is described in the following enumeration.

1. Calculate the g parameter for max and min Mie scatter factor.
2. Calculate the final Mie scatter directional parameter g by interpolating the parameters calculated in 1. based on the current Mie scatter factor (haze factor).
3. Adjust the intensity of the incident sunlight based on the Mie scatter factor. (More haze means less sunlight gets through).

5.3 Areal Perspective

5.3.1 Proposed Method for Calculating Areal Perspective

The proposed method builds on the work of Preetham and Hoffman. To expand the method to non constant density atmospheres, the depth parameter is modified depending on observer altitude and the altitude of the individual vertices.

The scattering function is used to replace the traditional hardware fog. The attenuation of a vertex is calculated, using (5.1), where L_0 is the reflected RGB color of the vertex. To be physically correct, the optical depth should be integrated along the viewing ray for both Rayleigh and Mie scattering, but this is too complicated to allow real time performance and would additionally require tone mapping of the results.

To calculate the optical depth between a point in the terrain and the viewpoint accurately, it is necessary to solve the integral from equation (3.3). This is not feasible for a real time approach.

The two simplest ways of estimating the average density are to either calculate the average altitude of the viewing path and then use the density at the average altitude, or to calculate the density at both ends of the viewing

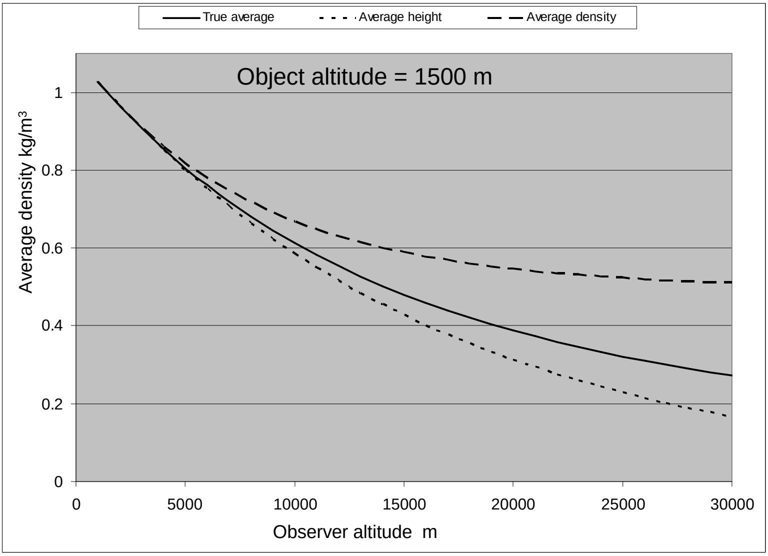


Figure 5.2: *Comparing the average density of a viewing path, calculated as the density at average altitude, to the average density of the start and endpoint. The solid path is the correct average calculated by integration.*

path and use the average of these two values. Figure 5.2 shows a comparison of the two methods for a terrain point at 1500 meters and an observer altitude between 1000 and 30000 meters with a vertical viewing path.

As can be seen from figure 5.2, using the average altitude provides the most precise results. However, both methods have been implemented to determine which method results in the visually most convincing results.

Regardless of which method is chosen, it is mathematically far from correct, but it does retain the basic intuitive quality that the air is clearer(thinner) at high altitude and that a mountain peak will seam clearer than the base.

To hide the fact that our world ends rather abruptly at the far plane of the view frustum, it is assured that the horizon color blends with the sky color at the horizon. This results in a blurry horizon which will always be the case when the horizon is far away[5].

The color of the horizon is a result of complete domination by inscattering. As a result, the color of vertices at the far end of our viewing frustum should be the same as the horizon color of the skydome. The factors used when calculating the inscattered colors of the skydome and the terrain are identical. As a result, assuming that the initial color is completely extinct, variations in two vertices at the same screen position are a result of difference in atmospheric depth s. To ensure that the colors of the terrain at the visibility limit and the color of the skydome at the horizon are identical, it is necessary to ensure that they have the same optical depth.

This is easily done for a fixed density atmosphere, but in this case the problem becomes more complicated. The system is designed around a maximum visibility of 200 kilometers, and to assure that the depth is big enough to blend with the sky color, the visibility is restricted for ground level observers.

5.4 Sky Color

5.4.1 Proposed Method for Calculating Sky Color

The color of the sky is calculated as vertex colors on a skydome, using equation (5.2), combined with the expression for the Mie scattering term.

The color is calculated as a result of inscattering. Since the background/initial color is the black color of space (figure 3.1) the extinction term can be ignored.

$$L(s,\theta) = L_{in}(s,\theta) \tag{5.5}$$

Where L_{in} is given by.

$$L_{in} = \frac{\beta_r(\theta) + \beta_M(\theta)}{\beta_R + \beta_M} E_{sun}(1 - e^{-(\beta_R \cdot s_R + \beta_M \cdot s_M)}) \tag{5.6}$$

β_R is given by (2.2) and β_M is given by (2.5). s_R and s_M is the molecule and particle optical depth of the viewing ray. The optical depth of a viewing

---

[5] The sharp horizon seen eg. at the beach looking over the ocean exist because the horizon line are only two kilometers away when seen from the height of a standing human.

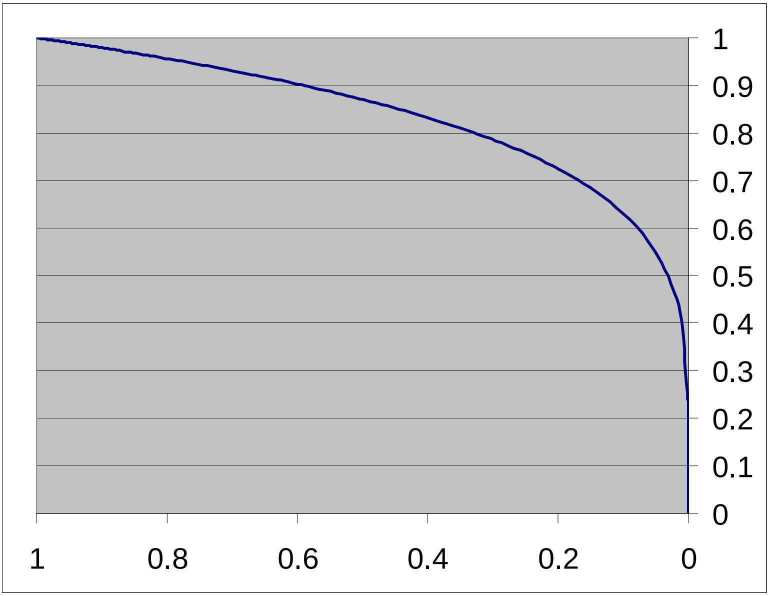


Figure 5.3: *Parameter used to interpolate between variable and static sky function. x-axis values are the length of the y component of the vertex normal, which is 1 at zenith and 0 at the horizon.*

ray varies with the altitude of the observer and with the zenith angle α of the viewing ray.

The optical depth increases fast over the last few angles towards the horizon. This is why the sky turns white close to the horizon. To simulate this, a function with similar characteristics is developed. This value is not based on physical calculations, but is determined on the basis that we need a function with a shape that expresses the same characteristics as the color of the sky.

To get a function with these characteristics, it is proposed to use a root function, based on the vertical y component of the normals of the sky dome. The function used is given by.

$$f = \sqrt[5]{vertex_{normal.y}} \tag{5.7}$$

Choosing a higher root value will cause the falloff to start slower and then fall steeper at the end. The shape of this function for a root value of 5 is shown in figure 5.3.

Given a function with the right characteristics, a static value for the optical depth of the skydome can be calculated, which will result in a gradual transition from the deep blue zenith color to a bright white horizon color. The most important constrain on this constant value is that the resulting color at the horizon has to be completely dominated by inscattering to get a smooth blend with the color of the terrain at the horizon.

This value is chosen to be around 200 km to match the visibility set for the viewing frustum. The equation used is given by.

$$T_{skydome} = (1.05 - f) * 190000 \tag{5.8}$$

Where f is given by equation (5.7), the resulting optical depth is shown as the dark blue function shown in figure 5.4

This static value can only be used for a ground based observer and in order to expand its use to observers with variable altitude it is assumed that the optical depth at the horizon is constant (infinite), following the description presented in section 2.4.1. Using this assumption, the optical depth of a viewing ray over the skydome is calculated as an interpolation between a dynamically calculated zenith value and the static value given by (5.8).

The vertical (zenith) optical depth can be calculated analytically. The (optical) density of molecules and aerosols has to be determined individually, but if exponential falloffs are used for both, the process is identical, assuming that the density of the atmosphere is given by.

$$\rho_{molecules} = 1.2 \cdot e^{\frac{-h}{8400}} \tag{5.9}$$

Where 8400 is the scale value.

The integrated density (optical depth)[6] along a vertical path from an altitude h to $h_{infinity}$ is given by.

$$T_{zenith} = c \cdot 10000 \cdot e^{\frac{-h}{10000}} \tag{5.10}$$

---

[6] integrated density and optical depth are proportional. Consequently, they can be used interchangeably. In this thesis, most values are given as nondimensional values because the significance is on the characteristics of the functions, not the exact values.

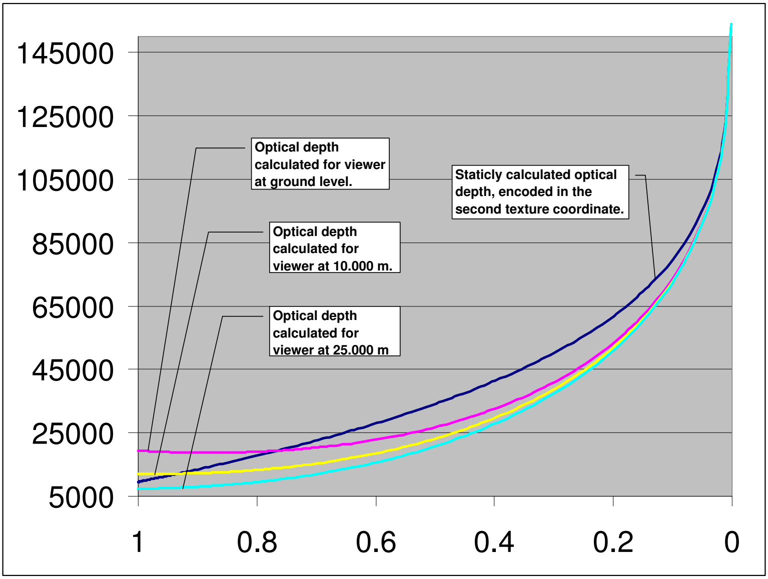


Figure 5.4: *Profiles of optical depth of the skydome for different observer altitudes. On the baseline, a value of 1 corresponds to zenith and a value of 0 corresponds to the horizon.*

Where c is a constant used to adjust the function to give the desired visual results. The scale value is changed to 10000 in order to provide a more visually convincing result.

Using equation (5.10), it is possible to adjust the vertical optical depth each frame based on observer altitude. To expand this to the rest of the skydome, it is proposed to simulate the optical depth of the skydome by interpolating between the dynamic zenith optical depth and the static optical depth $T_{skydome}$. The interpolation is done using the parameter f given by (5.7), in such a way that the dynamic value completely dominates the final value at zenith and the static value completely dominates at the horizon. The resulting functions for optical depth for a few different observer altitudes can be seen in figure 5.4.

The most important result of this function is that it is possible to capture the darkening of the sky that results when the observer is high in the atmosphere.

Chapter 6

Implementation

In the following the details of the implementation are described. The implementation is based on the DirectX sample framework. This means that most of the basic structures of the program are based on classes provided in the DirectX SDK. This includes things like initialization of devices and windows and the basic application structure.

On top of this basic application structure a system is implemented for rendering a world consisting of a sky and terrain. This system is used to demonstrate the workings of the scattering shaders.

The basic structure of the application is shown in (figure 6.1). The CAppForm class is derived from the basic DirectX application class provided by the SDK.

The most important components of the application are the Terrain renderer, the CROAM class and the CSky class, which are responsible for rendering the skydome. The CROAM class handles simplification and rendering of the height map while CSky gives access to various atmospheric and sun data and renders the skydome.

6.1 Code Structure

The structural layout of the implementation is shown in figure 6.1. The CAppForm class is the base class provided by the DirectX sample frame-

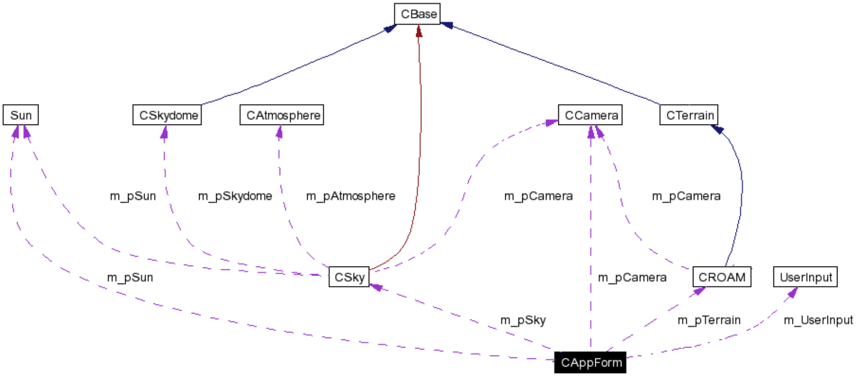

CBase
Sun
CSkydome
CAtmosphere
CCamera
CTerrain
m_pSun
m_pSkydome
m_pAtmosphere
m_pCamera
m_pCamera
m_pSun
CSky
m_pCamera
CROAM
UserInput
m_pSky
m_pTerrain
m_UserInput
CAppForm

Figure 6.1: *Class structure diagram for the Outdoor Rendering application.*

work. It provides the basis of the implementation and includes the main starting point.

The two classes responsible for rendering of the sky dome and the terrain are the CSky and the CROAM classes. Several related parameters needed when rendering both the sky dome and the terrain are shared between the CSky and the CROAM classes. These include the atmospheric conditions stored in the CAtmosphere class, the sun intensity and direction stored in the CSun class and the position of the observer and view parameters stored in the CCamera class.

In addition to being the main access class for atmospheric information, the CSky class is responsible for rendering the sky dome geometry stored in the CSkydome class.

The CBase class contains various debugging functions, and the CTerrain class provides basic functions like the loading of height fields to the CROAM class.

The system was designed to be easily transferable to other systems. Only four classes need to be touched by the application; the CSun, CSky, CCamera and the CROAM classes.

6.2 Optical Depth Estimations

As described in section 5.2, the scattering is calculated on basis of an estimated optical depth. This corresponds to estimating the integral equation (3.3).

The integral can be interpreted as the average optical density of the path timed the distance. $t = \beta^{\lambda}(\rho_{avg} \cdot s)$ where β varies with wavelength and separate values of ρ have to be used for particles and aerosols.

6.2.1 Areal Perspective

To determine the average optical depth of a path between the observer and an object in the terrain, two different methods are implemented as described in section 5.3.1. Both methods can be calculated using roughly the same amount of operations in the vertex shader. The following is a short explanation on the implementation of the two methods.

In the average altitude method, the average altitude of the observer and the vertex is calculated. $h_{avg} = \frac{Vertex.y + Eye.y}{2}$. This is then used to calculate the density at the average altitude, using an exponential equation as described in section 5.4.1 $\rho_{avg} = T_{avg} = e^{\frac{-h_{avg}}{scale}}$ where scale varies for aerosols and molecules.

In the average density method, the density at the eye point is calculated before rendering and stored in a constant register. This is done for both molecules and aerosols. In the shader, the density at the vertex is calculated. Both densities are calculated using the same exponential equations used to determine the density at the average altitude in the previous paragraph. When the two densities are determined, the average density is calculated as $\rho_{avg} = T_{avg} = \frac{\rho_{vertex} + \rho_{eye}}{2}$

From the average optical thickness, calculated using one of the described methods, the optical depth is obtained by multiplying with the distance from the viewpoint to the vertex.

6.2.2 Sky Color

The optical depth of a view path through the atmosphere is determined by the zenith angle of the viewing ray and the altitude of the observer.

As described in section 5.4.1, a static optical depth is calculated. To speed up the rendering, this is encoded into the second texture coordinate of the skydome. The parameter calculated by equation (5.7) is stored in the first texture coordinate.

The vertical or zenith optical depth T_z is precalculated each frame and stored in a constant register, where it is used to determine the final optical depth to a given vertex by interpolation, as described in section 5.4.1.

To determine the optical depth of a given vertex in the skydome the following equation is used.

$$T_{vertex} = Tex_v + Tex_u(T_z - Tex_v) \tag{6.1}$$

Where Tex_v is the static optical depth calculated using equation (5.8) Tex_v is the parameter value and T_z is the zenith optical depth given by (5.10). All of these parameters are precalculated and the final depth computation (equation 6.1) is done in a single instruction for each vertex.

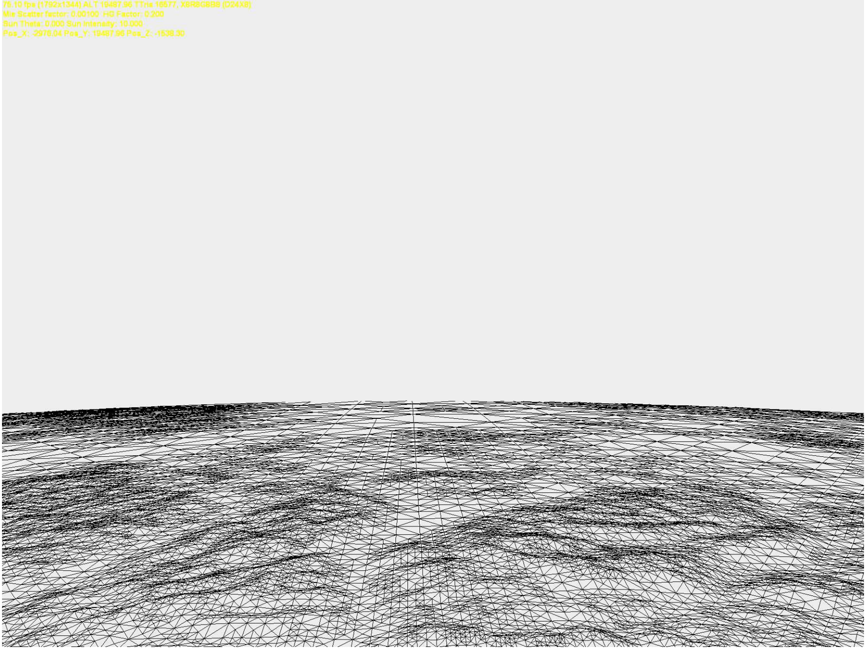
6.3 Terrain Rendering

In flight simulators, the terrain rendering algorithm needs to be able to handle large datasets at a vide range of viewing distances and angles. This requires the terrain rendering algorithm to be able to fluently adapt the level of detail for the visible part of the terrain.

Several algorithms capable of this have been developed. They all have different pros and cons and a thorough discussion of them is beyond the scope of this thesis. I have chosen to use an algorithm called ROAM (Real-time Optimally Adaptive Meshes).

I have chosen to use version 2 of the algorithm [4]. The main difference between the two versions is a change of the underlying data structure from triangles to diamonds, where each diamond is characterized by a single vertex in its center. The actual triangles of the diamond is composed of the center vertices of its parents and ancestors in the diamond tree structure. This leads to a faster and more memory conserving implementation [19].

When a optimal set of terrain triangles has been generated, the change from one frame to the next is usually relatively small. This can be used to optimize the algorithm by modifying the (almost optimal) triangle set from



```
76.10 fps (1792x1344) ALT 19487.96 TTris 16577, X8R8G8B8 (D24X8)
Mie Scatter factor: 0.00100 HG Factor: 0.200
Sun Theta: 0.000 Sun Intensity: 10.000
Pos_X: -2976.04 Pos_Y: 19487.96 Pos_Z: -1538.30
```

Figure 6.2: *Adaptive tessellation of terrain data using ROAM.*

the previous frame, instead of rebuilding the triangle tree from scratch for each frame. ROAM does this by adding the diamonds associated with newly created diamonds to a priority queue, and then, for each frame, subdividing or merging these triangles based on their priority. This priority is based on the distance from the camera and how rough the terrain is. If the terrain areas are rough, diamonds need to be split earlier than they do in flat terrains areas.

Because the proposed scattering algorithm is based on a per vertex calculation, it can produce artifacts if the tessellation of the terrain is unevenly distributed. The tessellation produced by the ROAM algorithm is controlled by an error metric, which calculates the priority of a given diamond, based on the distance from the viewpoint and the change in contour that a split of the diamond would create.

This allows the error metric to be tuned, so that the terrain is tesselated

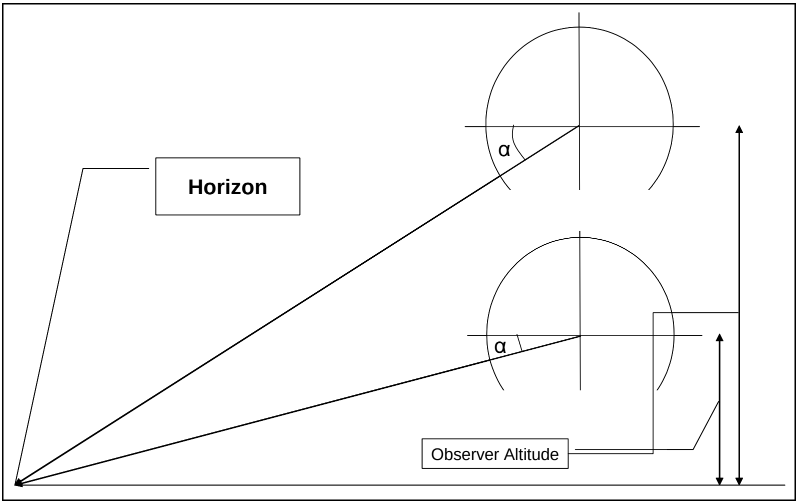


Figure 6.3: *The viewing path crosses the skydome at different angles for different observer altitudes, if the dome is positioned static with respect to the observer.*

primarily based on the distance from the camera, removing any large screen space triangles (figure 6.2).

6.4 Sky Rendering

The sky is modelled as a spherical dome with the optical depth and a interpolation parameter encoded in the texture coordinates, as described in section 6.2.2.

Since the skydome constitutes the background of the image, it can be rendered first without writing to the z-buffer. This assures that anything rendered after the skydome is rendered in front of it. The advantage is that this allows us to render the skydome as a small dome. In this case it is rendered as a hemisphere with a one kilometer radius. If the skydome has been rendered at the edge of the viewing range, effects resulting from z-buffer fighting and numerical inaccuracies might occur.

The small dome does, however, present a problem if the observer is positioned statically with respect to the dome. When the observer changes

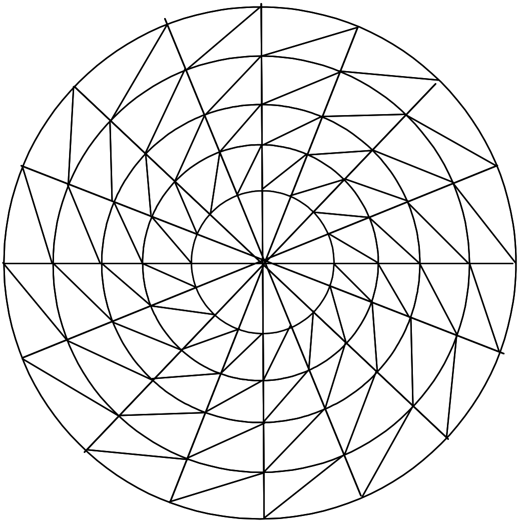


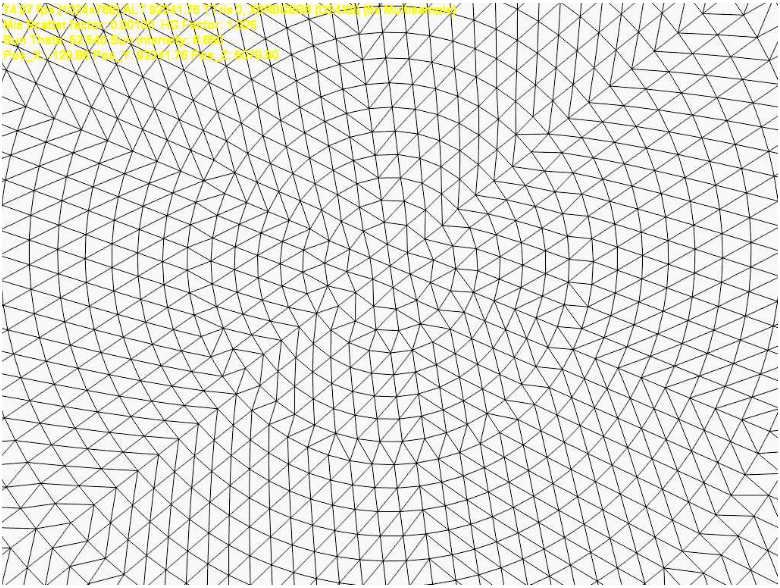
Figure 6.4: *When a skydome is tessellated using a latitude longitude approach the tessellation is considerably denser at zenith compared to the equator.*

altitude, the angle of the viewing path towards the horizon is changed (figure 6.3). This will make the gradual change in sky color close to the horizon climb and descend with the observer. To compensate for this effect, the position of the observer along the zenith axis of the sky dome is altered slightly when the observer changes altitude, assuring that the horizon is kept roughly at a constant angle α.

Because the scattering calculations are performed on the vertex level, the method is somewhat dependent on tessellation. This means that it requires a lot of triangles to render the skydome.

6.4.1 Sky Dome Tessellation

A skydome is usually created similar to the earths geographic system, using a set of latitude and longitude lines. This results in a situation, where the skydome is finely tesselated at the zenith, while triangles close to the horizon are large (figure 6.4). To get a good tessellation at the horizon using a system like this, there will be a considerable waists of triangles close to zenith.



```
74.37 fps (1024x768) ALT 92241.76 Tris 0, X8R8G8B8 (D24X8) (6x Multisample)
Mie Scatter factor: 0.00100 HG Factor: 1.026
Sun Theta: 52.540 Sun Intensity: 8.850
Pos_X: -129.86 Pos_Y: 92241.76 Pos_Z: 5079.90
```

Figure 6.5: *Screen shot of dome evenly tessellated to minimize the amount of vertices needed.*

To improve this, an algorithm to calculate a uniformly tesselated dome has been constructed. The basic idea is to keep the distance between longitude bands constant, and then tessellate each band using roughly equal size triangles. The result is demonstrated in figure 6.5.

The basis of the algorithm is to divide the latitude bands into equal pieces and then create the triangles at these points. From time to time this will result in an extra triangle being added or removed depending on weather the band is expanding or contracting compared to the previous band. The areas where this happens can be seen as bands radiating out from the center in figure 6.5.

The algorithm uses a series of conditional expressions to determine when to insert an extra triangle. A listing of the section of code responsible for generating the triangles can be seen in section E.3.

6.5 Vertex and Pixel Shader Implementation

The main part of the scattering simulation is done using vertex shaders. Part of the scattering implementation could have been implemented using pixel shaders.

The extinction factor could have been found using a texture lookup, where the texture coordinate is a function of the optical depth and would have to be calculated for Mie and Rayleigh scattering separately. The inscatter factor could be found in similar ways, using a 2d texture, where one texture coordinate is a function of of the optical depth and the other a function of the angle from the sun θ. Because both parts would require separate texture lookups for Mie and Rayleigh scattering factors, the pixel shader implementation requires 4 textures; two for extinction and two for inscattering. In addition, most of the calculations would still need to be done in the vertex shader to get the correct optical depths and to adjust the g parameter in the Henyey-Greenstein phase function.

Because of these difficulties, it was chosen to do all scattering calculations using vertex shaders and apply the inscatter and extinction factors as vertex colors. This leaves the pixel shader to apply the extinction factor and add the inscattering.

As described in section 4.2.1, vertex shaders operate on individual vertices received from a vertex buffer, using a combination of constant registers and mathematical instructions.

Solving the scattering problem involves solving equation (5.1). Doing this per vertex involves precalculating some values and storing those in constant registers, combined with a calculation using the available instructions. The code responsible for setting the constant registers can be seen in appendix E.1 and E.2.

The result of the vertex shader operations is passed to the pixel shader, where final color computation is conducted.

The pixel shaders for rendering the skydome and terrain are rather different. The pixels shader of the skydome is simply an interpolation of the inscattering calculated at the vertices. The coloring of the terrain involves lighting calculation and texturing, as well as the extinction and inscattering contributions.

Constant	Description
WorldViewProj	Combined World View And Projection matrix for positioning and projecting the vertices into screen space.
LightDir	Direction of the sunlight in world space.
vSunColorIntensity	The RGB color of sunlight and the intensity of the sun. E_{sun}
vBetaRayleigh	The total Rayleigh scattering term. β_R
vBetaDashRayleigh	The total Rayleigh scattering term multiplied by the term used in the Rayleigh scattering phase function. $\frac{3}{16\pi}\beta_R$
vBetaMie	The total Mie scattering term. β_M
vBetaDashMie	The total Mie scattering term multiplied by the term used in the Mie scattering phase function. $\frac{1}{4\pi}\beta_M$
vOneOverBetaRM	Combined Mie and Rayleigh total scattering constant, *optimization to avoid having to do the addition in the vertex shader.*
vHG	Vector containing recalculated elements of the Henyey-Greenstein phase function equation (5.3). $[\ 1-g^2 \quad 1+g^2 \quad 2g \quad \cdot\ ]$
vEyePos	The position of the observer in world space.

Table 6.1: *Constants used in the terrain and sky shaders.*

Because all shader programs are written using HLSL, the compiler controls which constants to occupy which registers, and any direct reference to register numbers in the following is purely for illustrative purposes.

6.5.1 Constant Registers

The precalculated values are stored in constant variables, of which many are common between the sky shader and the terrain shader. The common values are given in table 6.1

In addition to the common constant registers, the sky shader and the terrain shader each has a few individual constants. These are mainly caused by the different ways of estimating the average optical depth for areal perspective

Constant	Description
vDensityAlt	The density of the atmosphere at the observer altitude, used to calculate average density.
texDetailFactor	Constant variable used to calculate the texture coordinate for the detail texture. *Controls the number of times the detail texture is repeated across the terrain data set.*
LightDirDot	Light direction modified to be used in normal map lighting. *The normal map considers z to be up not y.*

Table 6.2: *Constants used only in the terrain shader.*

and sky color. Individual shader constants for the terrain and sky shaders are given in tables 6.2 and 6.3 respectively.

Constant	Description
vDensityAlt	The integrated vertical density from the observation point to the edge of the atmosphere.

Table 6.3: *Constants used only in the sky shader.*

When using the effect file framework (section 4.2.4), it is possible to define the constants directly in the effect file. In the current implementation, most parameters are set for each frame by the application. This makes it possible to couple the parameters to user input and change them at runtime, which is a great help during development.

6.5.2 Vertex Shader

The vertex shaders for the skydome and the terrain are almost identical. This is especially true for the part that relates to the calculation of atmospheric scattering.

Both shaders are implemented using HLSL programs in the effect framework. The full effect files can be seen in appendix D on page 123.

In table 6.4, a description of the processing of vertices in the terrain vertex shader is outlined. The only difference between this and the sky vertex shader is that the optical depth is estimated differently, see section 6.2.2, and that the extinction term is not transferred to the sky pixel shader.

6.5.3 Pixel Shader

The pixels shader for the skydome simply interpolates the vertex colors. The pixel shader for the terrain is a little more interesting.

For the terrain pixel shader, the extinction and inscattering terms are used to perform an interpolation between the texture color of the terrain and an inscattered color of the atmosphere. The color of the terrain is calculated by multiplying the basic texture with a normal map lighting calculation. This result is multiplied by the extinction term and the inscattered color is added.

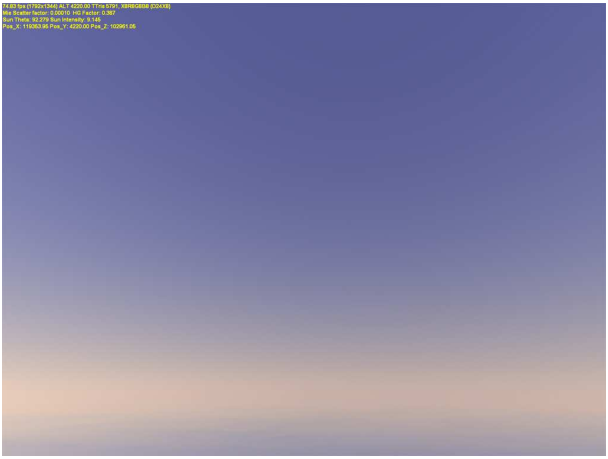
Description	Function
Calculate molecular and aerosol density at the vertex position.	$\rho_m = e^{-vert.y/15000}$ $\rho_a = e^{-vert.y/5000}$
Calculate average molecular and aerosol density.	$\rho_{m,a} = \frac{\rho_{m,a}+\rho_{m,a}^{eye}}{2}$
Calculate vector from eye point to vertex.	$\vec{v} = vert.xyz - eye.xyz$
Calculate length of $\vec{v}$	$l = \|\vec{v}\|$
Normalize $\vec{v}$	$\vec{v} = \frac{\vec{v}}{l}$
Multiply $\rho_{m,a}$ by l to get optical depths.	$t_{m,a} = \rho_{m,a} \cdot l$
Calculate extinction term.	$x = e^{\beta_M t_a + \beta_R t_m}$
Calculate the cosine of the angle between the sunlight and the view vector $\vec{v}$	$\cos(\theta) = \vec{v} \cdot \vec{sun}$
Calculate the modified rayleigh phase function term.	$\beta_R(\theta) = 2 + \frac{1}{2}\cos\theta$
Calculate the Henyey-Greenstein phase function term. (*vHG is the constant described in table 6.1*)	$hg = \frac{vHG.x}{(vHG.y - vHG.z \cdot \cos(\theta))^{3/2}}$
Calculate inscatter terms. *c1 and c2 are the constants vBetaDash – Rayleigh and –Mie (table 6.1)*	$I_{ray} = c1 \cdot \beta_R(\theta)$ $I_{mie} = c2 \cdot hg$
Calculate inscattering. c_{sun} *is the constant vSunColorIntensity*	$temp = \frac{I_{ray}+I_{mie}}{vOneOverBetaRM}$ $temp = temp \cdot (1-x)$ $I_{tot} = temp \cdot c_{sun}$
Extinction x and inscatter I_{tot} is output to the pixel shader.	

Table 6.4: *Description of the work done in the terrain vertex shader.*

Chapter 7

Results

To compare the proposed method to previous methods, and to evaluate the solution of the described problem, a series of test scenes are shown. Each scene demonstrates a specific property of the system, and where possible scenes are presented alongside reference images from real life.



```
74.83 fps (1792x1344) ALT 4220.00 TTris 5791, X8R8G8B8 (D24X8)
Mie Scatter factor: 0.00010  HG Factor: 0.387
Sun Theta: 92.279 Sun Intensity: 9.145
Pos_X: 119353.95 Pos_Y: 4220.00 Pos_Z: 102961.05
```

Figure 7.1: *View of the sky 90 degree angle to the sun at sunset. The directional effect of the Rayleigh scattering phase function is only just visible.*

Depending on the described problem, a mixture of images from previous solutions and real life has been used as references.

7.1 Sky Color

To present the capability of the method to capture the color of the sky at different atmospheric conditions and for various positions of the sun and observer, a series of specific requirements and the accompanying results of the developed system is presented.

7.1.1 Rayleigh Scattering Intensity

The relative strong directional dependency of rayleigh scattering caused visible artifacts for the method proposed by Hoffman and Preetham [10]. This is clearly visible in the image shown in figure 3.8 page 49.

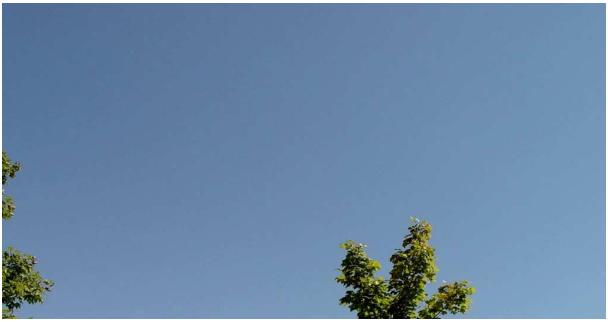
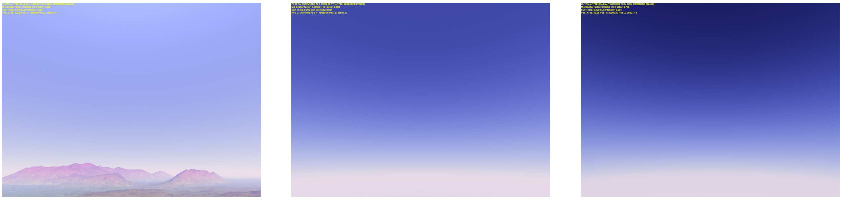


Figure 7.2: *Image of the sky at a clear day with the sun approximately 10 degree above the horizon. The directional dependency of Rayleigh scattering is hardly visible.*

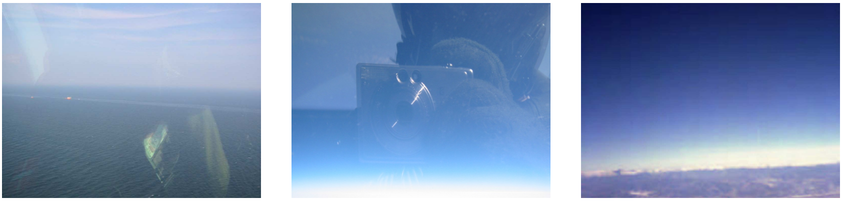
Using the modified Rayleigh scattering phase function, a result similar to the one seen in figure 7.1 is obtained. This is much more in line with the results observed in nature.

7.1.2 Effect of Observer Altitude on Sky Color

The reduction in the optical depth above the observer results in a darkening of the sky above when flying at high altitude. Figure 7.3(a) - 7.3(c) shows



(a) Rendering at ground level (b) Rendering at altitude 16 km (c) Rendering at altitude 30 km



(d) Picture at altitude 500 feet (e) Picture at altitude 50000 feet (f) Picture at altitude 100000 feet

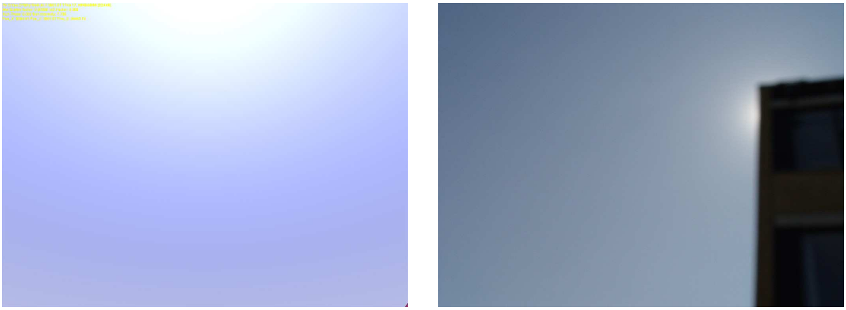
Figure 7.3: *The change in sky color that happens when the observer climbs up through the atmosphere. Shots taken at ground level, 16 km and 30 km*

the result obtained using the proposed system.

For comparison, a series of pictures taken from the cockpit of a Danish F-16 fighter aircraft and a picture taken from a research balloon are shown in figure 7.3(d) - 7.3(f)

In both the images rendered by the proposed system and the images taken of the real atmosphere at different altitudes, it can be seen that the color of the horizon stays white and bright regardless of the observer altitude. It can also be seen that the sky closer to zenith gets darker with increased observer altitude.

This ability to capture the effect of change in atmospheric depth was one of the important improvements that was presented as requirements in the problem description.



(a) Sun direction rendering (b) Sun direction picture

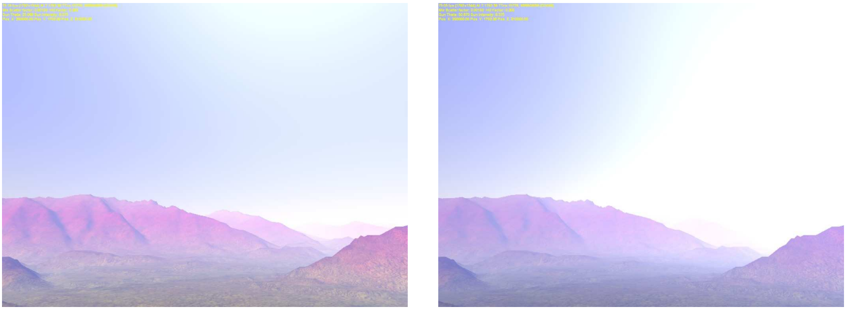
Figure 7.4: *The color of the sky brightens and intensifies fast as the view direction approaches the sun.*

7.1.3 Change in Sky Color Close to the Sun

The color of the sky changes fast when the view approaches the sun. This is the result of the Mie scattering phase function. This effect can be seen in figure 7.4.

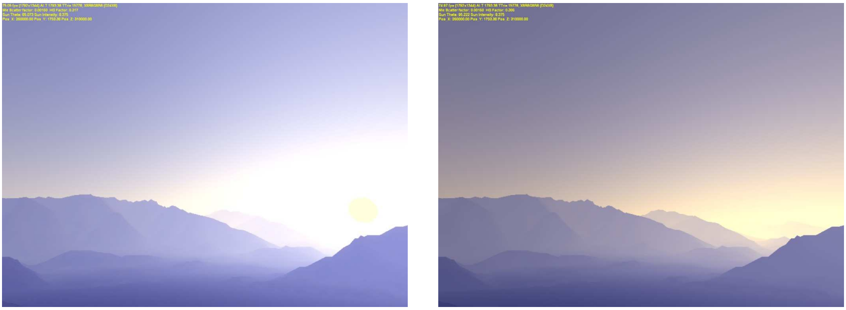
7.1.4 Sky Color with Changing Sun Position

The color and intensity of the sky change during the day. This effect is most noticeable at sunrise and sunset. Figure 7.5 shows the result of a rendering of the sky color during sunset.



(a) Zenith angle 25°

(b) Zenith angle 50°



(c) Zenith angle 85°

(d) Zenith angle 93°

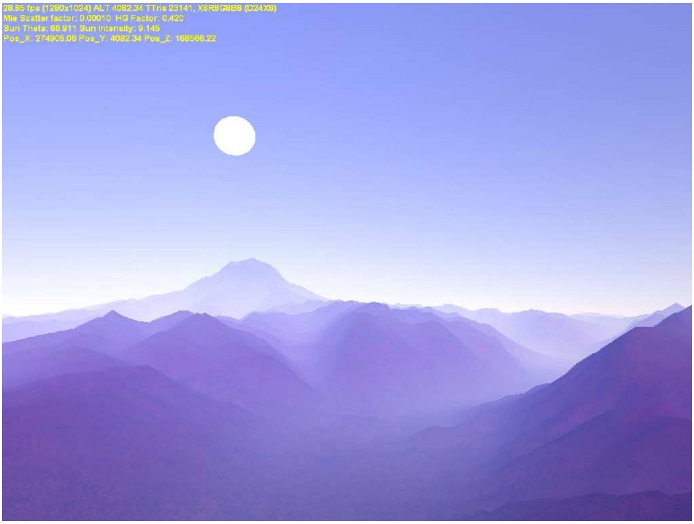
Figure 7.5: *The color of the sky during sunset and sunrise, demonstrating the increased intensity close to the sun and the reddening of the sky close to the horizon.*

7.2 Areal Perspective

The capability of the method to capture the effects of areal perspective is presented, using a series of examples.

7.2.1 Blue Color of Distant Mountains

The color of distant objects is attenuated by extinction and inscattering. Because the wavelength dependency of areal perspective is not captured correctly by traditional hardware fog, it will, in some situations, especially on clear days, make the environment look a bit boring.



```
28.85 fps (1280x1024) ALT 4082.34 TTris 23141, X8R8G8B8 (D24X8)
Mie Scatter factor: 0.00010  HG Factor: 0.420
Sun Theta: 66.911 Sun Intensity: 9.145
Pos_X: 274905.06 Pos_Y: 4082.34 Pos_Z: 168566.22
```

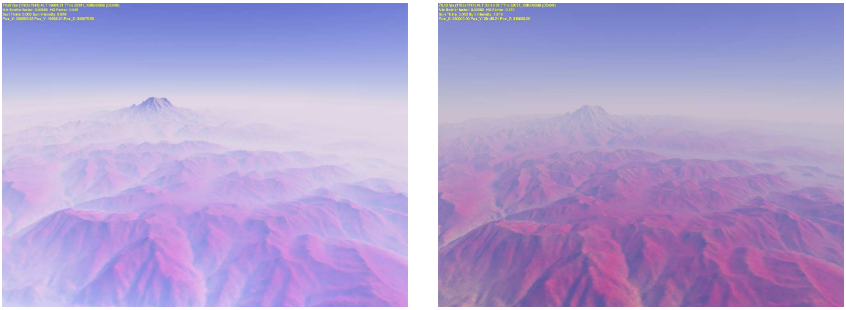
Figure 7.6: *The color of distant objects is attenuated.*

Figure 7.6 demonstrates this effect. Notice that the mountains in the foreground are colored blue by the inscattered light, while the color of the more distant peaks and especially the valleys is more yellow or white.

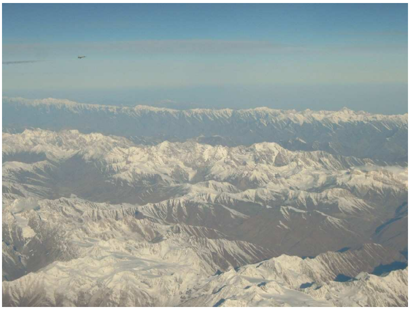
7.2.2 Variation in Visibility with Terrain Altitude

When looking at distant terrain features, the peak of mountains will often be less attenuated than the objects at a lower sea level altitude. This is caused by the thinning of the atmosphere with altitude, which results in a relatively shorter optical depth to the mountain peak.

The ability to capture these effects is one of the main problems to be solved by the proposed system.



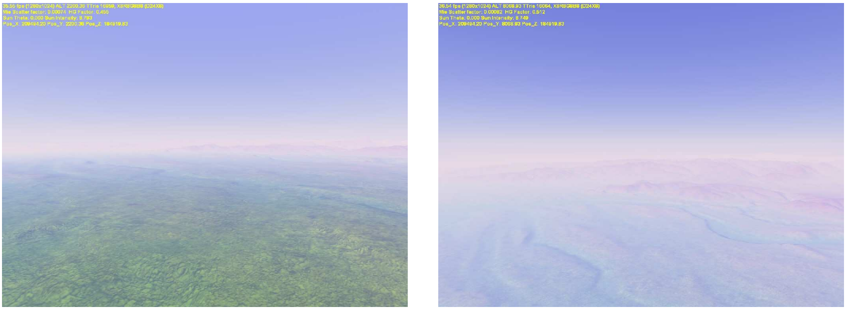
(a) Average density (b) Average altitude



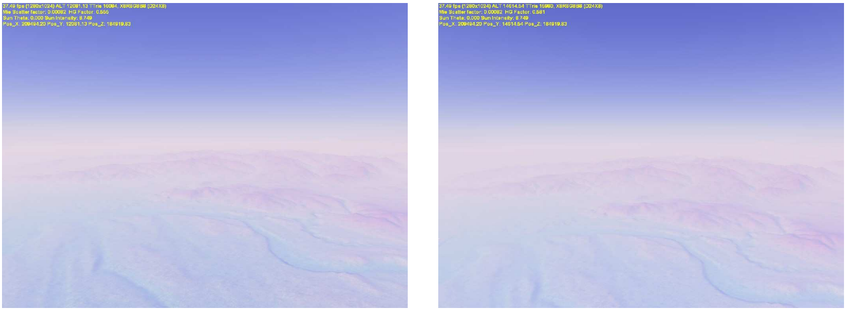
(c) Real world

Figure 7.7: *The optical depth varies with the altitude of the observed terrain. The mountains are seen emerging from the denser lower atmosphere. Image (a) shows the effect of using the average density, image (b) shows the effect of using density at average altitude, image (c) shows the effect in real. Notice that the mountain range in the background seams to emerge from the atmosphere much clearer than the valley in front of it.*

Figure 7.7 demonstrates the effect of terrain altitude on the the optical depth of the viewing ray. The mountains are shown emerging from the denser atmosphere at the valleys. The vertical scale of the terrain has been increased to make the effect more visible. Consequently, the mountain in the foreground is approximately 13 km high.



(a) Observer altitude 2 km (b) Observer altitude 8 km



(c) Observer altitude 12 km (d) Observer altitude 16 km

Figure 7.8: *The visibility increases when the observer altitude is increased. This is a result of the lower average density of the atmosphere penetrated by the view path. The images are generated at the same horizontal position, and it is clear that distant mountains emerge as the viewpoint is raised.*

7.2.3 Variation in Visibility With Observer Altitude

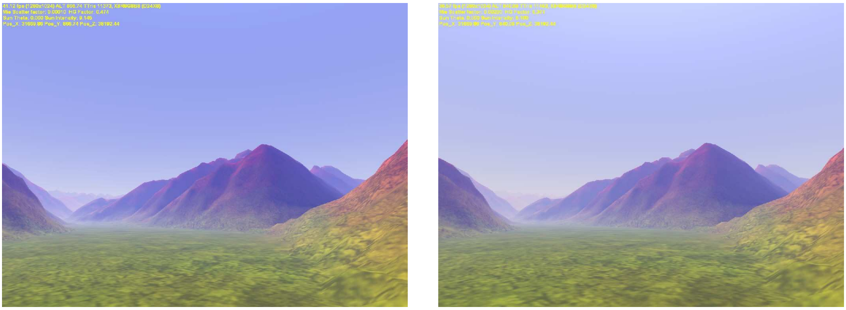
Objects which might appear extinct and barely visible at ground level will often become clear and display improved contrast when climbing up through the atmosphere. This happens because the average density of the atmosphere through which the observer sees falls when the observer altitude is increased.[1] This is basically the same effect described in section

---

[1] The curvature of the earth creates a natural limitation on the visibility at low altitudes, but this is only a factor when considering relatively low objects in really flat terrain.

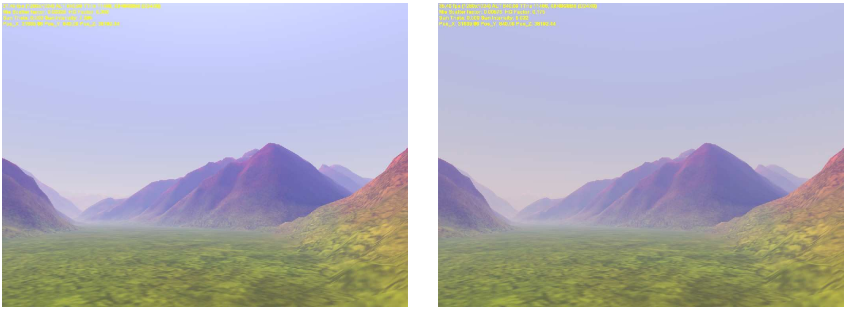
7.2.2 and they are both simulated in the same way.

Figure 7.8 demonstrates this. In image 7.8(a) at 2 km. the distant hills are barely visible. On subsequent images (7.8(b), 7.8(c), 7.8(d)) at 8, 12 and 16 km. the hills in the background become more and more visible.



(a) Clear day

(b) Light haze



(c) Medium haze

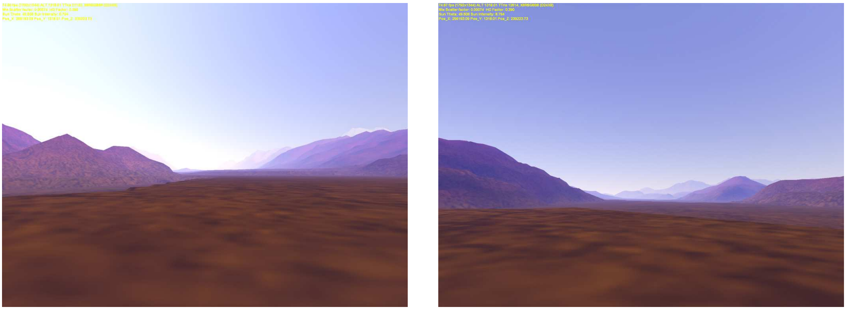
(d) Strong haze

Figure 7.9: *Variations in areal perspective as a result of variations in the amount of aerosols in the atmosphere.*

7.2.4 Visibility as a Result of Aerosol Concentration

When the amount of aerosols in the atmosphere changes, the color and intensity of scattered light change as well. This is known as haze.

Figure 7.9 shows the effect of change in the amount of aerosols in the atmosphere. In the first picture, the mountain range at the end of the valley is clearly visible and only a slight inscattering of blue light is visible. In the last picture, the mountain range is completely extinct and the inscattered light is white or yellowish.



(a) Looking in the direction of the sun

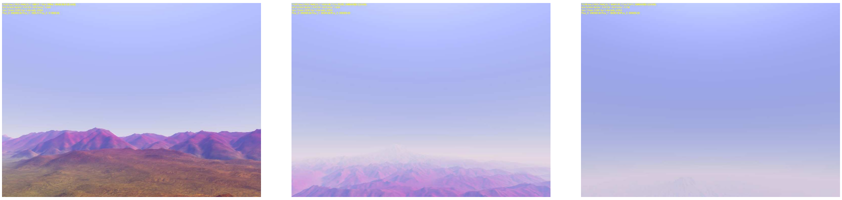
(b) Looking away from the sun

Figure 7.10: *Variation in visibility when looking in the direction of the sun and away from the sun at sunset and sunrise. Image (a) shows the reduced visibility when looking in the direction of the sun, compared to the visibility when looking away from the sun image (b).*

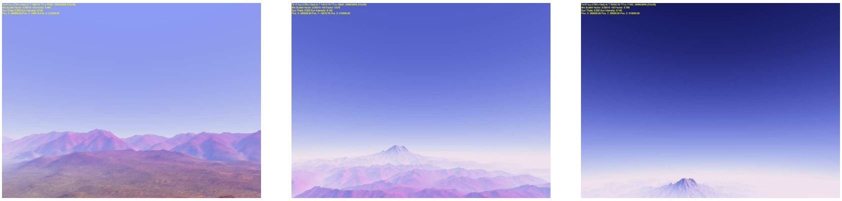
7.2.5 Variations in Areal Perspective with Sun Direction

When the sun is close to the horizon, visibility varies with the view direction. Looking close to the sun, the inscattering is higher than when looking away from the sun, and, as a result, the visibility is greater.

Figure 7.10 demonstrates the effect of the directional dependency of atmospheric scattering on areal perspective and visibility. The first image 7.10(a) shows the strong inscattering when looking close to the sun. The color of the inscattered light is the bright yellowish color of the sunlight, indicating that the Mie scattering dominates. This is in contrast to the second image 7.10(b) from the same location and sun altitude, but now looking in the opposite direction. The visibility is much improved and the inscattered light turns the mountains blue, indicating that Rayleigh scattering dominates.



(a) Old system "constant density" 2000 m (b) Old system "constant density" 10000 m (c) Old system "constant density" 30000 m



(d) New method "variable density" 2000 m (e) New method "variable density" 10000 m (f) New method "variable density" 30000 m

Figure 7.11: *Demonstration of the differences of a system that includes density change to one that don't.*

7.3 Comparison with Previous Methods

In this section, the implemented system is compared to the system developed by Hoffman and Preetham. Some of the images are created using the original demo application developed for Siggraph 2002, while others are created by implementing their methods in the application developed to demonstrate the expansions proposed in this thesis.

7.3.1 Density Effect on Sky Color

It was established in section 1.2 that the change in sky color with altitude is important for flight simulator environments. The system proposed by Hoffman and Preetham is not developed to cover these effects. This is the main issue requiring the system to be expanded. Figure 7.11 demonstrates the difference between a static sky color and a sky color that changes with altitude.

7.3.2 Density Effect on Terrain Visibility

When the density falloff in the atmosphere is modelled, a higher altitude will make it possible to see farther. This capability will be missing in a constant density system. Figure 7.11 demonstrates this. It can be seen that visibility is slightly better at the constant density model when at 2000 m (7.11(a), 7.11(d)), while at an altitude of 10000 m (7.11(b), 7.11(e)) this has changed, making visibility slightly better for the proposed (variable) density system.

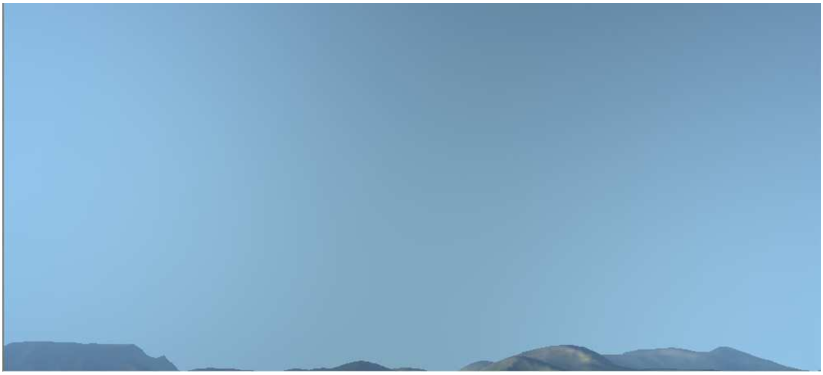


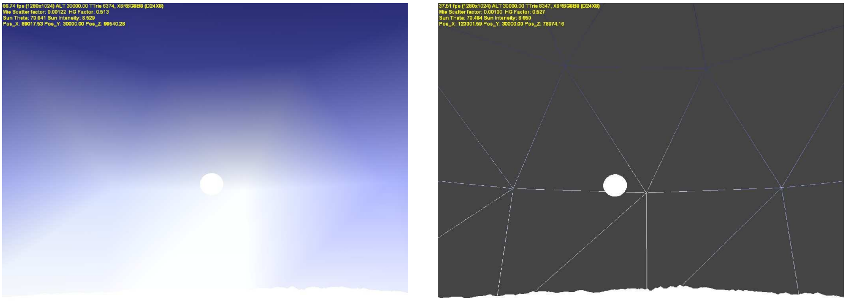
Figure 7.12: *Rendering of the skydome using the demo application supplied by Hoffman and Preetham. Notice the darkening of the sky in the center.*

7.3.3 The Modified Rayleigh Phase Function

As described in section 3.2.1, the direct mapping of intensities to RGB colors cause artifacts. This is also present in the application by Preetham and Hoffman, as can be seen in figure 7.3.2. Another thing left out in their application is the change in optical depth from zenith towards the horizon. However, this must be considered an implementation issue.

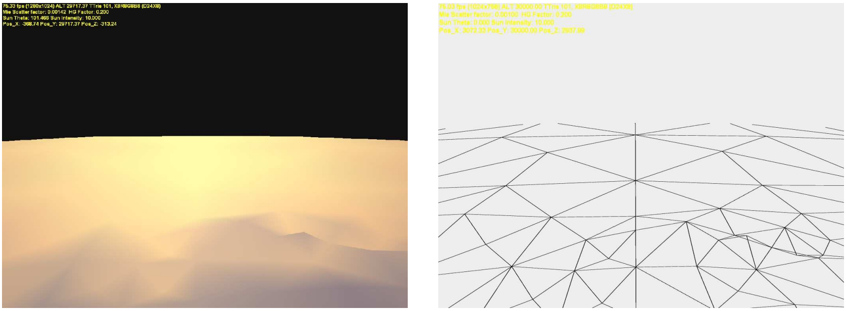
7.4 Artifacts and Limitations

In this section, problems with the current method are described. Some of the issues are issues that can possibly be avoided but present certain



(a) Artifacts as a result of poor tessellation on sky dome

(b) Wire frame demonstrating the tessellation



(c) Artifacts as a result of poor tessellation on terrain

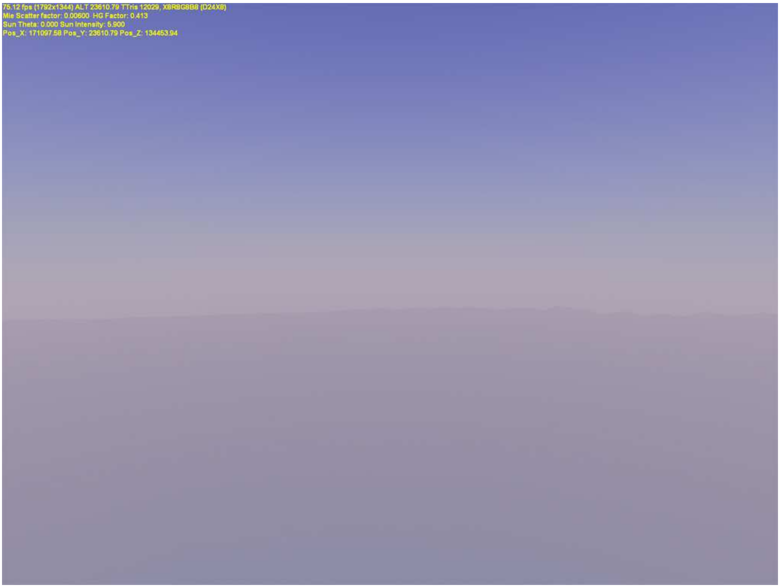
(d) Wire frame demonstrating the tessellation

Figure 7.13: *If the tessellation is insufficient artifacts will occur as a result of the interpolation of the vertex calculated scattering parameters.*

requirements to the implementation.

7.4.1 Tessellation Artifacts

Because the scattering effects are calculated on the vertex level, the method is somewhat dependent on tessellation. Figure 7.13 demonstrates the result of insufficient tessellation of the skydome and terrain.



```
75.12 fps (1792x1344) ALT 23610.79 TTris 12029, X8R8G8B8 (D24X8)
Mie Scatter factor: 0.00600  HG Factor: 0.413
Sun Theta: 0.000 Sun Intensity: 5.900
Pos_X: 171097.58 Pos_Y: 23610.79 Pos_Z: 134453.94
```

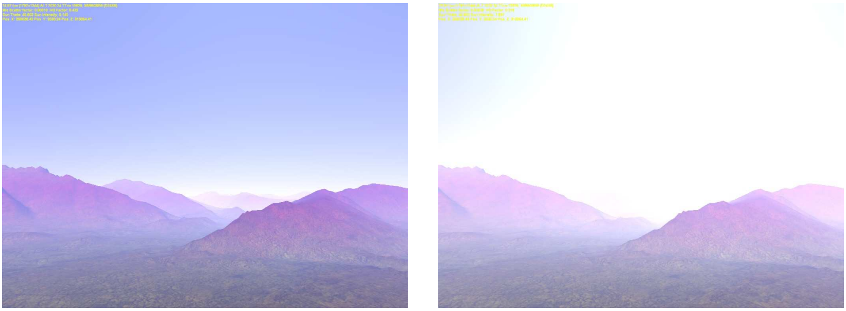
Figure 7.14: *Artifact in the areal perspective rendering prevents the color of the terrain and skydome to blend smoothly, revealing the end of the visible world. This only happens at high altitude when aerosol concentration is high.*

7.4.2 Mie factor artifact

At some points the horizon becomes visible as the perfect blend with the sky color is lost, this happens when the optical depth calculated for the terrain at the horizon is too small resulting in a contribution from the original terrain color.

7.4.3 Clamping Artifact

At some points the inscattering contribution exceeds the maximum value displayable on a computer screen. This leads to clamping artifacts as can be seen in figure 7.15



(a) Within the displayable range (b) Values out of displayable range

Figure 7.15: *When the calculated color values exceed the values displayable on the monitor clamping occurs. Notice the sudden change from blue to white in (b).*

Chapter 8

Discussion

In this section, the pros and cons of the implemented solution are evaluated. First, the different parts of the solution will be analyzed separately, followed by a general analysis of the ability of the whole system to fulfill the requirements, as described in section 1.2. Finally, any possible future improvements to the method are discussed.

8.1 Scattering Models

In section 5.2 it was decided to ignore the full spectral radiometric equation. Instead it was proposed that the effects of atmospheric scattering could be simulated using simplified RGB vector calculations.

Using RGB vectors has shown to be a good compromise between speed and accuracy. Using simple equations based on RGB vectors makes it possible to implement the scattering calculations in vertex shaders.

Because the method only samples the three RGB colors and not the full spectrum, the chosen wavelengths of the red, green and blue components can be seen as a weight given to that specific color. This happens because of the wavelength dependency of both Mie and especially Rayleigh scattering.

This weight can be used to modify the amount of red, green and blue produced by the system and, as such, tune the system to produce as visually pleasing results as possible. However, this possibility will not guarantee correct results in all circumstances, only a full spectrum calculation can do that, but, in practise, the proposed method has proven to produce convincing results with few artifacts and must be considered a clear improvement compared to previous methods.

8.1.1 Rayleigh Scattering

The modified Rayleigh scattering phase function equation (5.2) has proven to provide a good compromise. Because of the relative week angular dependency, it can be argued that it could be abandoned all together and that Rayleigh scattering should be modelled as having no directional dependency.

While most users would probably never notice the missing directional component in the Rayleigh scattering calculations, it has been included for accuracy and because the computational requirements are small.

The primary results of Rayleigh scattering, the blue color of the sky and the blue attenuation of distant objects, are captured well by the system. The white or yellow color of the inscattered light as the optical depth of the viewing path approaches infinity (see section 2.4.1) is also handled by the system.

The scattering equation (5.1) is derived from the work done by Hoffman and Preetham [10] and their system was able to capture the effects of Rayleigh scattering as well. However, their method does produce some artifacts not present in the proposed system. Because they use the Rayleigh scattering phase function directly (equation 2.1), while at the same time treating the RGB calculation as true radiometric calculations, the directional dependency of Rayleigh scattering is exaggerated, as can be seen from figure 3.8.

8.1.2 Mie Scattering

The use of the Henyey-Greenstein phase function makes it possible to capture the strong directional dependencies of Mie scattering. This increases the perceived realism of the sky color, especially close to the sun.

The proposed system is capable of rendering the bright region surrounding the sun when the sun is high in the sky, as well as the impressive coloring of the sky during sunset and sunrise (figure 7.5).

When the concentration of atmospheric aerosols is changed and the amount of Mie scattering is increased, it creates problems because of the limited dynamic range of the display system. One consequence is that the bright white area surrounding the sun grows and the clamping of the color values becomes visible.

Because second and higher order scattering is ignored, the inscattered light from directions far from the sun hardly increases when the Mie scattering factor is increased. Because the increased extinction is not balanced by an increased inscattering, the environment ends up looking dull and washed out.

Both of these problems are greatly reduced by altering the shape of the Henyey-Greenstein phase function by altering the g parameter, see equation (2.6).

To simplify the problem of adjusting the g parameter, a simple linear equation calculating the g parameter as a function of sun zenith angle, observer altitude and Mie scattering concentration was proposed (equation 5.4).

This equation is a clear improvement compared to the method developed by Hoffman and Preetham, where 3-5 parameters had to be adjusted for each scene to look convincing. However, although a clear improvement, the proposed linear fit still contains some artifacts and could be improved.

In the current implementation, similar parameters are used to calculate the g parameter for the terrain and the skydome. Some improvements might be possible by optimizing the linear equations for the ground and sky separately. It may also be possible to use higher order functions to calculate the g parameter and, in that way, get a better result.

8.2 Sky Color

Using the scattering equations to color the sky has proven to be a useful option. Compared to the method of Hoffman and Preetham, the proposed method is, while based on the same basic principle, able to produce much more convincing color, requiring a lot less user interaction.

The modification of the Rayleigh scattering phase function, combined with a linear equation to adjust the Henyey-Greenstein phase function, results in visually convincing images that allow the user to move around in the environment without needing to modify the parameters of the system.

The use of both a constant depth at the horizon and a variable depth towards zenith makes it possible to simulate the effects of the reduced inscattering and consequently much darker sky above the observer at high altitudes, producing results very similar to those seen in the real world (see figure 7.3).

8.3 Areal Perspective

Compared to traditional hardware range based fog, the proposed system creates a much more convincing simulation of the effects of areal perspective. This clearly demonstrates the advantages of the method developed by Hoffman and Preetham.

The method proposed by Hoffman and Preetham was, however, unable to cope with the effects of change in atmospheric density with altitude. For a flight simulator this is clearly necessary and their method is insufficient in that respect.

The proposed expansion of the method, where atmospheric density is taken into account, functions well, especially in a flight simulator environment. It makes it capable of displaying effects that depend on the altitude of the terrain and observer, like the ones described in sections 7.2.2 and 7.2.3.

The simulation of areal perspective is important for the sense of distance in the scene. At the same time, it is used to hide the fact that the visible world ends abruptly at the far clipping plane.

Because the method accounts for varying atmospheric density the average atmospheric density from the observation point to the horizon changes as well. This leads to a visual artifact in some areas, where the horizon of the terrain becomes visible against the color of the skydome, see figure 7.14.

8.4 Optical Depth Estimates

The idea of using an estimated average optical depth in the calculation of atmospheric scattering has proven capable of producing visually convincing results.

The basic idea is somewhat similar to the idea proposed by Klassen in [14]. He proposed using two layers of constant density atmosphere, and then adjust the thickness of the layers to account for varying optical density along the path. In the proposed method, the average atmospheric density is estimated, each frame using a set of simple equations that can be evaluated in real time.

The proposed method demonstrates that it is possible to achieve good results using simple models for the determination of the average optical depth. This follows a general observation that the human vision system has a much higher tolerance for small errors associated with inaccuracies, compared to the tolerance of an effect missing completely.

Considering the simplicity of the proposed model and the relatively good results, one might argue that it would be possible to determine the average atmospheric density based only on observer altitude while ignoring elevation differences in the terrain. This approach would be feasible for scenes containing flat terrain, where all visible terrain features were at roughly the same altitude. But, as can be seen from figure 7.7, for rough mountainous terrain, the difference in atmospheric density in valleys and at mountain peaks results in visible difference in the effect of areal perspective. This shows that the vertex altitude can not be ignored when determining the optical depth.

8.5 Problems and Future Improvements

Two main issues remain with the proposed method. One is adjusting the inscattered sunlight to prevent clamping issues as those presented in figure 7.15. The other issue is assuring that the horizon and the terrain blends together smoothly, failing to do this will cause artifacts as the one shown on figure 7.14.

Clamping issues are a result of the inscattering values exceeding displayable values. The most obvious source of clamping problems arises when the sun sets. As the sun approaches the horizon, the strong directional inscattering resulting from Mie scattering combines with the already bright Rayleigh inscattering through the thick atmospheric path close to the horizon.

Because the problem is caused by the Mie scattering part $\beta_M(\theta)$ from equation (5.1), which is enforced by the large optical depth if the viewing path is closer to the horizon, it is hard to totally eliminate the problem. The sky should be bright close to the sun even when the sun is high in the sky as demonstrated on figure 7.4, and, at the same time, The directional factor should remain at sunset and sundown without half of the sky turning white.

The problem is partly solved by reducing the intensity of the sunlight as the sun descents and partly by reducing the directionality of the Henyey-Greenstein phase function as described in section 5.2.2 equation (5.4).

Unlike the method proposed by Dobashi, Yamamoto and Nishita [3], the proposed system is unable to capture volumetric effects like shafts of light and cast shadows. However, rendering these effects requires an amount of resources, especially from the graphics card, that prohibits the simultaneous running of a flight simulator.

Their method is able to render environments similar in structure to those presented in this work at around 5 frames per second, with a screen resolution of 720x480. With modern graphics hardware this might be 10 - 15 today, but for this method to be used in a flight simulator it is estimated that it should be running at more than 100 frames per second in a stand alone application. It is not unlikely that this will be possible in a few years and, by that time, their method would be capable of increasing the visual realism beyond the capabilities of the proposed system.

In the near future (first half of 2004), hardware capable of doing texture lookups in vertex shaders will emerge. Such a texture could be used to determine the optical depth, using the altitude of the vertex and the eye point as texture coordinates. It could also be possible to use textures for phase functions and, as such, they could present a potential speedup of the proposed system.

Chapter 9

Conclusion

Computer hardware has reached a level, where new levels of visual realism are becoming possible. For demanding real time rendering systems like flight simulators, one such improvement is the realistic rendering of atmospheric scattering effects.

In this thesis, a system is proposed that is capable of rendering effects, which, so far, have been limited to non real time rendering or, at best, interactive rendering systems. The developed system builds on the work of previous researchers, particularly on the work of Hoffman and Preetham [10].

These effects represent important visual clues like the change in the color of the sky for different positions of the sun, varying observer altitude and different concentrations of atmospheric aerosols.

In addition to the improved simulation of sky color, the method contains a clear improvement to the simulation of areal perspective, which has traditionally been simulated using range based fog. The proposed method is capable of capturing the change in areal perspective that results from changes in observer altitude and variation of terrain height.

These effects are important in flight simulator environments. They play a role both for the immersion and for the tactical environment. Some improvements are possible, specifically to the functions that adjust the directionality of the Henyey-Greenstein parameter and the intensity of sunlight.

In the future rendering of scattering effects, could be simulated using volumetric rendering techniques. Such techniques have been introduced by Dobashi [3] but are not yet efficient enough for inclusion in a flight simulator rendering system.

9.1 Summary of the Proposed System

The main new idea in the proposed system is that a relatively simple set of equations can be used to calculate the optical depth between a point in the scene and the eyepoint. The main problem when calculating the optical depth is the variation in atmospheric density, which requires the solution of an integral to determine the optical depth.

In this thesis, it is proposed to determine the optical depth as the average density along the view path multiplied by the length. Two different methods have been proposed to calculate the average density.

1. To calculate the density at the average altitude (the midpoint of the view path).
2. To calculate the average density as the average of the density at the two endpoints.

It was determined that the last method provided the most visually convincing results, even though the first method provided the most physically correct results (figure 5.2).

Another important improvement is a way of adding a variable optical depth to the skydome. The variable optical depth is dependent on the altitude of the observer, and this dependency is strongest close to zenith and then gradually falls, resulting in a constant optical depth at the horizon. This allows the horizon to be seen as having a constant infinite optical depth, which is coherent with what is seen in the real world.

The proposed method of determining the average density, combined with improvements to the optical depth of the skydome, adjustments to the Rayleigh Phase function and the calculation of the Mie scatter factor g, has expanded the system proposed by Hoffman and Preetham to the point, where it is suitable for inclusion in a pc based flight simulator.

9.2 Fulfillment of the Hypothesis

The ideas and propositions presented in the hypothesis have all shown to be possible. Thus, they represent a solution to the problem described in section 1.2.

The proposed method represents a clear improvement compared to previous methods, when viewed both in terms of speed and visual realism. The proposed system include effects, caused by the variation in atmospheric density with altitude. In addition several other parts of the rendering system, have been modified, to give better results when mapping directly from calculated intensities to RGB colors.

The idea of using the scattering theory as a guideline for the development of a system aimed at producing visually pleasing results rather than physically correct ones. Has proven successful.

Some effects are still beyond the capability of the proposed method. These include volumetric effects like shafts of light and the corresponding shadow shafts as shown in figure 3.9.

Bibliography

[1] Tomas Akenine-Moller and Eric Haines. *Real-Time Rendering.* A K Peters Ltd, 2nd edition, July 2002.

[2] James F. Blinn. Light reflection functions for simulation of clouds and dusty surfaces. In *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, pages 21–29. ACM Press, 1982.

[3] Yoshinori Dobashi, Tsuyoshi Yamamoto, and Tomoyuki Nishita. Interactive rendering of atmospheric scattering effects using graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 99–107. Eurographics Association, 2002.

[4] Mark Duchaineau. The roam 2 homepage. Web page, 2003. http://www.cognigraph.com/ROAM_homepage/ROAM2.

[5] Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain: real-time optimally adapting meshes. In *IEEE Visualization*, pages 81–88, 1997.

[6] Fredo Durand and Julie Dorsey. Interactive tone mapping. In *Proceedings of the 11th Eurographics Workshop on Rendering*, pages 219–230, 2000.

[7] Nolan Goodnight, Rui Wang, Cliff Woolley, and Greg Humphreys. Interactive time-dependent tone mapping using programmable graphics hardware. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 26–37. Eurographics Association, 2003.

[8] Vladimir I. Haltrin. A real-time algorithm for atmospheric corrections of airborne remote optical measurements above the ocean. In *Proceedings of the Second International Airborne Remote Sensing Conference and Exhibition*, volume III, pages 63–72. Environmental Research Institute of Michigan, June 1996.

[9] M. J. Harris and A. Lastra. Real-time cloud rendering for games. In *Proceedings Game Developer Conference 2002*. Game Developers Conference, March 2002.

[10] Naty Hoffman and Arcot. J. Preetham. Rendering outdoor light scattering in real time. In *Proceedings Game Developer Conference 2002*. Game Developers Conference, March 2002.

[11] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 99–108. ACM Press, 1996.

[12] R. W. G. Hunt. *Measuring Color*. Ellis Horwood Limited, 1987.

[13] John Irwin. Full-spectral rendering of the earth's atmosphere using a physical model of rayleigh scattering. In *Proceedings of the 1996 Eurographics UK Conference*.

[14] R. Victor Klassen. Modeling the effect of the atmosphere on light. *ACM Transactions on Graphics (TOG)*, 6(3):215–237, 1987.

[15] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 109–118. ACM Press, 1996.

[16] Earl J. McCartney. *Optics of the Atmosphere.* Wiley 1976, 1976.

[17] M. G. J. Minnaert. *Light and Color int the Outdoors.* Springer-Verlag, 1993.

[18] Tomoyuki Nishita, Yoshinori Dobashi, Kazufumi Kaneda, and Hideo Yamashita. Display method of the sky color taking into account multiple scattering. In *Proceedings of Pacific Graphics 1996*, pages 117–132, August 1996.

[19] Trent Polack. *Focus on 3D Terrain Programming.* Premier Press, December 2002.

[20] A. J. Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM Press/Addison-Wesley Publishing Co., 1999.

[21] Microsoft DirectX Team. Directx dokumentation. Online dokumentation. http://msdn.microsoft.com/library/en-us/directx9_c/directx/directx9cpp.asp.

Appendix A

Instructions for the Demo Application

For moving around in the landscape the arrow keys are used, the *Home* and *End* keys are used for vertical rotation.

The *Z* and *X* keys are used to control the sun zenith angle, and the *C* and *V* keys are used to control the amount of aerosols (Haze) in the atmosphere.

Key	Action
UP ARROW	Move forward
DOWN ARROW	Move back
LEFT ARROW	Turn left
RIGHT ARROW	Turn right
END KEY	Pitch up
HOME KEY	Pitch down
NUM+ KEY	Move vertical up
NUM- KEY	Move vertical down
SHIFT KEY	Increase speed of movement
Z KEY	Increase sun zenith angle
X KEY	Decrease sun zenith angle
C KEY	Increase aerosol concentration
V KEY	Decrease aerosol concentration

The demo application is set to use the effect files called *TerrainEffect4.fx* and *SkyEffect2.fx*. They contain the proposed solution.

The other files can be used to replace the current files to see the effect of previous systems. The other effect files contain the following solutions.

- *SkyEffect.fx* Constant density rendering of the skydome, roughly corresponding to [10].
- *TerrainEffect2.fx* Constant density rendering of the terrain, corresponding to [10].
- *TerrainEffect3.fx* Rendering of the terrain using the density at average altitude instead of the average density at the endpoints.

Appendix B

Content of the CD-ROM

- **Articles** The articles referenced in the thesis, which are available for download. Numbered as they appear in the bibliography.
- **DirectX9 SDK** Installer for the DirectX9 software developer kit. Necessary for compiling the demo application.
- **DirectX9 Runtime** Installer for the DirectX9 runtime. Necessary for running the demo application. (This is installed as part of the SDK)
- **Pictures** The screen shots and pictures used in the thesis.
- **Source** The source code for the demo application.
- **Source Documentation** Doxygen html source code browser.

- *Shortcut to Demo application exe file* Shortcut to the demo application.
- *Shortcut to demo application visual studio solution file*Shortcut to the visual studio .NET solution file.
- *Shortcut to sourcebrowser.html* Shortcut to the doxygen source code browser index file.

Appendix C

Atmospheric Density

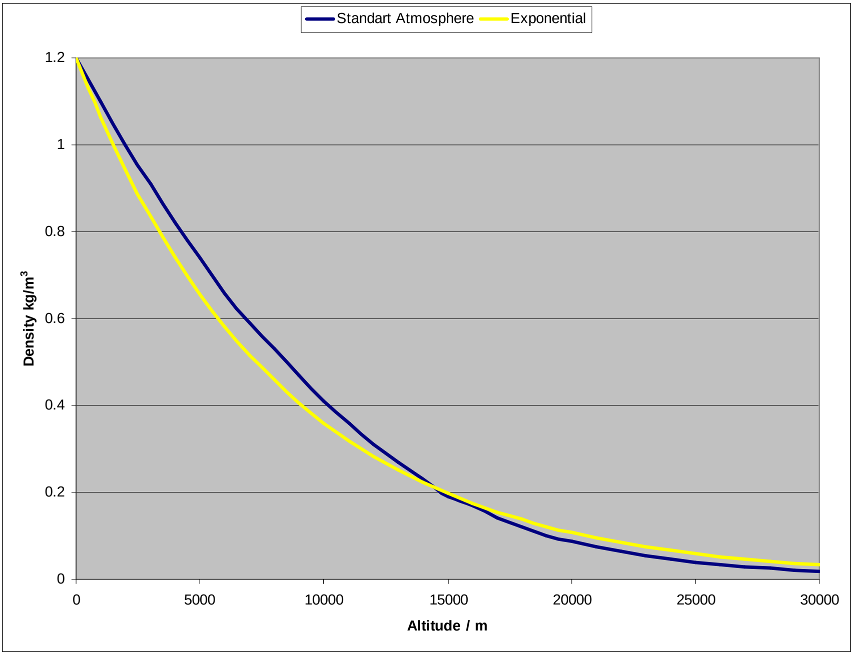


Figure C.1: *The density of the standard atmosphere compared to a exponential model*

Appendix D

Effect Files

D.1 Sky Effect

```
//this is the effect file for the skydome
matrix WorldViewProj          : WORLDVIEWPROJECTION;
float3 LightDir               : LIGHTDIRECTION;
vector vSunColorIntensity     : SUNCOLORINTENSITY;
float3 vBetaRayleigh          : BETARAYLEIGH;
float3 vBetaDashRayleigh      : BETADASHRAYLEIGH;
float3 vBetaMie               : BETAMIE;
float3 vBetaDashMie           : BETADASHMIE;
float3 vBetaRayleighMie       : BETARAYLEIGHMIE;
float3 vOneOverBetaRM         : ONEOVERBETARAYLEIGHMIE;
float4 vHG                    : HENYEYGG;
float3 vEyePos                : EYEPOSITION;
float4 vDensityAlt            : DENSDIST;
float3 vToneMap               : TONEMAP;

float4 vConstants = {1.0f,-1.4426950f,0.01f,1000.0f}; // constants

struct VS_OUTPUT
{
    float4 Pos  : POSITION;
    float4 Diff : COLOR0;
};
VS_OUTPUT VS(float3 vPos: POSITION,float3 Norm: NORMAL, float2 Tex : TEXCOORD)
{
    VS_OUTPUT Out = (VS_OUTPUT)0;

    //transform
    Out.Pos =   mul(float4(vPos,1), WorldViewProj);

    //determine angle between sun and view direction
    float4 viewAngle;
    viewAngle.x = dot(LightDir,Norm);
    viewAngle.y = (viewAngle.x * viewAngle.x) / 2 + 2;
    //viewAngle.y = (viewAngle.x * viewAngle.x)  + 1;

    viewAngle.z = lerp(Tex.y,vDensityAlt.z,Tex.x);
    viewAngle.w = Tex.y;
    viewAngle.w = lerp(viewAngle.w,vDensityAlt.w,Tex.x);

    //calculate extinction terms for inscatered extinction
    float3 extinction;
    extinction = vBetaRayleigh * viewAngle.z + vBetaMie * viewAngle.w;
    extinction = exp(-extinction);

    //calculate mie scatering term
    //Phase2(theta) = (1-g^2)/(1+g^2-2g*cos(theta))^(3/2)
    //vHG = [1-g^2, 1+g^2, -2g, insc]
    float4 phaseThetaMie;
    phaseThetaMie.x = vHG.z * viewAngle.x + vHG.y;
    phaseThetaMie.y = rsqrt(phaseThetaMie.x);
    phaseThetaMie.z = pow(phaseThetaMie.y,3);
    phaseThetaMie.w = phaseThetaMie.z * vHG.x;

    //Inscattering(I) = (Beta_R * Phase_R(theta) + Beta_M * Phase_M(theta)) *
    //                  [1-exp(-Beta_R*s).exp(-Beta_M*s)] / (Beta_R + Beta_M)
    float3 rayleigh;
    rayleigh = vBetaDashRayleigh * viewAngle.y;
    float3 mie;
    mie = vBetaDashMie * phaseThetaMie.w;
    float3 temp;
    temp = vConstants.x - extinction;

    float3 inscatter;   //I (inscattering)
    inscatter = (mie + rayleigh) * vOneOverBetaRM;
    inscatter *= temp;
    inscatter *= vSunColorIntensity.xyz;
    inscatter *= vSunColorIntensity.w;

    //color
    Out.Diff.xyz = inscatter;
    Out.Diff.z += 0.15f;
    Out.Diff.w = 1.0;

    return Out;
}

// PS
float4 PS(VS_OUTPUT In) : COLOR
{
    float4 Color = (float4)0;
    float4 Temp = (float4)0;
    Color = In.Diff;
    return Color;
}

//Technique T0, rendering the skydome with a constant color
//in a single pass, no scatering, no shaders
technique T0
{
    pass P0
    {
        fvf              = XYZ | Normal | Tex1;
        //FillMode       = WIREFRAME;
        VertexShader     = compile vs_1_1 VS();
        PixelShader      = compile ps_1_1 PS();
        Lighting         = FALSE;
        ZwriteEnable     = FALSE;
        CullMode         = NONE;
    }
}
```

D.2 Terrain Effect

```
//this is the terrain effect file, here we will apply the rendering of the terrain

matrix WorldViewProj            : WORLDVIEWPROJECTION;
matrix WorldView                : WORLDVIEW;
float3 LightDirDot              : LIGHTDIRECTIONDOT;
float3 LightDir                 : LIGHTDIRECTION;
vector vSunColorIntensity       : SUNCOLORINTENSITY;
float3 vBetaRayleigh            : BETARAYLEIGH;
float3 vBetaDashRayleigh        : BETADASHRAYLEIGH;
float3 vBetaMie                 : BETAMIE;
float3 vBetaDashMie             : BETADASHMIE;
float3 vBetaRayleighMie         : BETARAYLEIGHMIE;
float3 vOneOverBetaRM           : ONEOVERBETARAYLEIGHMIE;
float4 vHG                      : HENYEYGG;
float3 vEyePos                  : EYEPOSITION;
float4 vDensityAlt              : DENSDIST;

// light states
vector SkyDirection = {0.000, 0.000, 1.000f, 1.000f};

//textures
texture TerrainTexture  : TEXTURE0 <string name = ”gcanyon_texture2_2k.dds”;>;
texture DetailTexture   : TEXTURE1 <string name = ”detailMap.dds”;>;
texture NormalTexture   : TEXTURE2 <string name = ”NormalMap.bmp”;>;

// constants set by the app
float texDetailFactor < string TexFactor = ”Detail repeat count”; > = 100.0f;

sampler Sampler0 = sampler_state
{
    MinFilter   = LINEAR;
    MagFilter   = LINEAR;
    MipFilter   = LINEAR;
    AddressU    = CLAMP;
    AddressV    = CLAMP;
};

sampler Sampler1 = sampler_state
{
    MinFilter = LINEAR;
    MagFilter = LINEAR;
    MipFilter = LINEAR;
};

float4 vConstants = {1.0f,-1.4426950f,-0.00005f,0.1f}; // constants

struct VS_OUTPUT
{
    float4 Pos          : POSITION;
    float4 Inscatter    : COLOR0;
    float4 Extinction   : COLOR1;
    float2 Texcoord0    : TEXCOORD0;
    float2 Texcoord1    : TEXCOORD1;
    float2 Texcoord2    : TEXCOORD2;
};
VS_OUTPUT VS(float3 vPos: POSITION, float2 Tex : TEXCOORD)
{
    VS_OUTPUT Out = (VS_OUTPUT)0;
    Out.Texcoord0 = Tex;
    Out.Texcoord1 = Tex * texDetailFactor;
    Out.Texcoord2 = Tex;

    //position
    //Out.Pos   = mul(float4(vPos,1), WorldViewProj);

    float4 vEyeVert = (float4)0;
    float4 viewAngle = (float4)0;
    float3 extinction = (float3)0;

    //determine rayleigh and Mie optical depth(approximation) ;-)
    viewAngle.z = exp(-vPos.y / 15000.0f);
    viewAngle.w = exp(-vPos.y / 5000.0f);
    viewAngle.zw += vDensityAlt.zw / 2;
    vEyeVert.xyz = vPos - vEyePos;
    vEyeVert.w   = length(vEyeVert.xyz);
    vEyeVert.xyz /= vEyeVert.w;

    viewAngle.zw *= vEyeVert.w;

    //calculate extinction terms
    extinction = vBetaRayleigh * viewAngle.w + vBetaMie * viewAngle.z;
    extinction = exp(-extinction);       //= e^(-beta * depth)

    //determine angle between sun and view direction
    viewAngle.x = dot(LightDir,vEyeVert.xyz);
    viewAngle.y = (viewAngle.x * viewAngle.x) / 2 + 2;

    //calculate mie scatering term
    //Phase2(theta) = (1-g^2)/(1+g-2g*cos(theta))^(3/2)
    //vHG = [1-g^2, 1+g, -2g, insc]
    float4 phaseThetaMie;
    phaseThetaMie.x = -vHG.z * viewAngle.x + vHG.y;
    phaseThetaMie.y = rsqrt(phaseThetaMie.x);
    phaseThetaMie.z = pow(phaseThetaMie.y,3);
    phaseThetaMie.w = phaseThetaMie.z * vHG.x;

    //Inscattering(I)
    float3 rayleigh = (float3)0;
    rayleigh = vBetaDashRayleigh * viewAngle.y;
    float3 mie = (float3)0;
    mie = vBetaDashMie * phaseThetaMie.w;
    float3 temp = (float3)0;
    temp = vConstants.x - extinction;

    float3 inscatter = (float3)0;   //I (inscattering)
    inscatter = (mie + rayleigh) * vOneOverBetaRM;
    //inscatter = rayleigh / vBetaRayleigh;
    //inscatter = mie / vBetaMie;
    inscatter *= temp;
    //inscatter *= vHG.w;            //inscatter intensity
    inscatter *= vSunColorIntensity.xyz;
    inscatter *= vSunColorIntensity.w;

    //extinction *= vSunColorIntensity.xyz;
    //extinction *= vSunColorIntensity.w;

    //color
    //Out.Inscatter.xyz = extinction;
    Out.Inscatter.xyz   = inscatter;
    Out.Inscatter.z     += 0.15f;
    Out.Inscatter.a     = 1;
    Out.Extinction.xyz  = extinction;
    Out.Extinction.a    = 1;

    //position
    viewAngle = float4(vPos,1);
    viewAngle.y -= (vEyeVert.w * vEyeVert.w) / 5000000.0f;
    Out.Pos = mul(viewAngle, WorldViewProj);

    return Out;
}

// PS
float4 PS(VS_OUTPUT In) : COLOR
{
    float4 Color = (float4)0;
    Color.rgb = 2 * tex2D(Sampler0,In.Texcoord2)-1;      //normal map lighting
    Color.rgb = clamp(dot(Color.rgb,LightDirDot),0,1);   //normal map lighting
    Color *= tex2D(Sampler0, In.Texcoord0);              //Apply texture
    Color.rgb *= tex2D(Sampler1, In.Texcoord1)*2;        //apply detail texture
    Color.rgb *= vSunColorIntensity.rgb;                 //apply sun color;

    Color.rgb *= In.Extinction.rgb; //Apply extinction
    Color.rgb += In.Inscatter.rgb;  //Add inscatter
    Color.a = 1.0f;
    return Color;
}
```

```
technique SHADER11
{
    pass P0
    {
        fvf             = XYZ | Tex1;
        Lighting        = FALSE;
        //FillMode      = WIREFRAME;
        Sampler[0]      = (Sampler0);
        Sampler[1]      = (Sampler1);
        Sampler[2]      = (Sampler0);
        Texture[0]      = (TerrainTexture);
        Texture[1]      = (DetailTexture);
        Texture[2]      = (NormalTexture);
        VertexShader    = compile vs_1_1 VS();
        PixelShader     = compile ps_1_1 PS();
    }
}//Technique SHADER11, Render the terrain with shaders version 1.1

technique SHADER20
{
    pass P0
    {
        fvf             = XYZ | Tex1;
        Lighting        = FALSE;
        //FillMode      = WIREFRAME;
        Sampler[0]      = (Sampler0);
        Sampler[1]      = (Sampler1);
        Sampler[2]      = (Sampler0);
        Texture[0]      = (TerrainTexture);
        Texture[1]      = (DetailTexture);
        Texture[2]      = (NormalTexture);
        PixelShader     = compile ps_1_1 PS();
        VertexShader    = compile vs_2_0 VS();
    }
}//Technique SHADER20, Render the terrain with shaders version 2.0
```

Appendix E

Source Code Snippets

In this section snippets of source code that are considered important for understanding the functionality of the program is listed.

E.1 Roam.cpp

```
//-----------------------------------------------------------------
// Name:           CROAM::PreRender - private
// Description:    Inits variables needed by the effect
// Return Value: HRESULT: posible error
//-----------------------------------------------------------------
HRESULT CROAM::PreRender(CSky* pSky)
{
    HRESULT hr = S_OK;
    // Update vertex shader constants from view projection matrix data.
    D3DXMATRIX mat, matView, matProj;
    m_pd3ddevice->GetTransform(D3DTS_VIEW,&matView);
    m_pd3ddevice->GetTransform(D3DTS_PROJECTION,&matProj);
    D3DXMatrixMultiply(&mat,&matView,&matProj);

    //set the transform, currently world transfor for terrain is identity
    hr = m_pEffect->SetMatrix(m_hTransForms,&mat);

    //set the viewTransform, currently world transfor for terrain is identity
    hr = m_pEffect->SetMatrix(m_hWorldView,&matView);

    CAtmosphere* pAtmosphere = pSky->GetAtmosphere();

    // Scattering multipliers.
    float fRayMult = pAtmosphere->GetParam(eAtmBetaRayMultiplier);
    float fMieMult = pAtmosphere->GetParam(eAtmBetaMieMultiplier);

    D3DXVECTOR3 vSunDir = pSky->GetSunDir();
    hr = m_pEffect->SetValue(m_hLightDir,&vSunDir,12);
    D3DXVECTOR3 vSunDirDot(vSunDir.x,-vSunDir.z,vSunDir.y);
    hr = m_pEffect->SetValue(m_hLightDirDot,&vSunDirDot,12);

    D3DXVECTOR4 vSunColorIntensity = pSky->GetColorAndIntensity();
    //vSunColorIntensity.w = 9.677f - 352.95 * fMieMult;
    vSunColorIntensity.w = 9.2f - 550.0f * fMieMult;
    hr = m_pEffect->SetVector(m_hSunColorInt,&vSunColorIntensity);

    D3DXVECTOR3 vEye = m_pCamera->GetEyePt();
    // Eye Position
    m_pEffect->SetValue(m_hEyePosition,&vEye,12);

    //calculate base height for density distance for a athmosphere
    //asuming a exponential density distribution dens = 1.2 * e^(ALT/8000)
    //densbase = intggrate[alt to top] top dens defined = 0.0f
    float fDensityAltBase[4];
    fDensityAltBase[0] = vEye.y / 1000.0f;
    fDensityAltBase[1] = 9.06418430158f * powf(0.95f,fDensityAltBase[0]);
    fDensityAltBase[2] = expf(-vEye.y / 15000.0f);
    fDensityAltBase[3] = expf(-vEye.y / 5000.0f);
    hr = m_pEffect->SetValue(m_hDensityDist,fDensityAltBase,16);

    D3DXVECTOR3 vBetaR, vBetaDashR, vBetaM, vBetaDashM, vBetaRM, vOneOverBetaRM;

    // Rayleigh
    vBetaR = pAtmosphere->GetBetaRayleigh();
    vBetaR *= fRayMult;
    m_pEffect->SetValue(m_hBetaRayleigh,&vBetaR,12);
    vBetaDashR = pAtmosphere->GetBetaDashRayleigh();
    vBetaDashR *= fRayMult;
    hr = m_pEffect->SetValue(m_hBetaDashRayleigh,&vBetaDashR,12);

    // Mie
    vBetaM = pAtmosphere->GetBetaMie();
    vBetaM *= fMieMult;
    m_pEffect->SetValue(m_hBetaMie,&vBetaM,12);
    vBetaDashM = pAtmosphere->GetBetaDashMie();
    vBetaDashM *= fMieMult;
    m_pEffect->SetValue(m_hBetaDashMie,&vBetaDashM,12);

    // Rayleigh + Mie
    vBetaRM = vBetaR + vBetaM;
    m_pEffect->SetValue(m_hBetaRayleighMie,&vBetaRM,12);
    vOneOverBetaRM[0] = 1.0f/vBetaRM[0];
    vOneOverBetaRM[1] = 1.0f/vBetaRM[1];
    vOneOverBetaRM[2] = 1.0f/vBetaRM[2];
    hr = m_pEffect->SetValue(m_hOneOverBetaRayleighMie,&vOneOverBetaRM,12);

    // each term (extinction, inscattering multiplier)
    float fIns = pAtmosphere->GetParam(eAtmInscatteringMultiplier);

    // Henyey Greenstein's G value. and inscatering
    float g = pAtmosphere->GetParam(eAtmHGg);
    //g = 0.163f + vEye.y * 1.393e-5;
    g = 0.444f + vEye.y * 1.053e-5 - 5.984e-2 * pSky->GetSunTheta() - 3.521e-6 * pSky->
         GetSunTheta() * vEye.y;
    float g2 = 0.100f + vEye.y * 1.11e-5 - 3.18e-2 * pSky->GetSunTheta() - 6.01e-6 *
         pSky->GetSunTheta() * vEye.y;
    //g = 0.25f + vEye.y * 1.3e-5 - 0.00e-2 * pSky->GetSunTheta() - 9.2e-6 * pSky->
         GetSunTheta() * vEye.y;
    float c = -0.076923f + 153.846f * fMieMult;
    g = g + c*(g2 - g);
    D3DXVECTOR4 vG(1-g*g, 1+g*g, 2*g,fIns);
    m_pEffect->SetVector(m_hHenyeyGG,&vG);

    return hr;
}
```

E.2 Sky.cpp

```
HRESULT CSky::Prerender(LPDIRECT3DDEVICE9 pd3ddevice)
{
    HRESULT hr = S_OK;

    D3DXMATRIX tempWV, tempWVP;

    D3DXMATRIX matWorld;
    const D3DXVECTOR3* vEye = m_pCamera->GetPosition();
    D3DXMatrixIdentity( &matWorld );
    D3DXMatrixTranslation( &matWorld, vEye->x, vEye->y - (vEye->y/2000.0f), vEye->z);

    D3DXMATRIX matView;
    pd3ddevice->GetTransform(D3DTS_VIEW,&matView);
    D3DXMatrixMultiply(&tempWV, &matWorld, &matView);

    D3DXMATRIX matProj;
    pd3ddevice->GetTransform(D3DTS_PROJECTION,&matProj);
    D3DXMatrixMultiply(&tempWVP,&tempWV,&matProj);

    //Set effect transfor matrix
    hr = m_pEffect->SetMatrix(m_hTransForms,&tempWVP);

    // Scattering multipliers.
    float fRayMult = m_pAtmosphere->GetParam(eAtmBetaRayMultiplier);
    float fMieMult = m_pAtmosphere->GetParam(eAtmBetaMieMultiplier);

    D3DXVECTOR3 vSunDir = m_pSun->GetDirection();
    hr = m_pEffect->SetValue(m_hLightDir,&vSunDir,12);

    D3DXVECTOR4 vSunColorIntensity = m_pSun->GetColorAndIntensity();
    //vSunColorIntensity.w = 9.677f - 352.95 * fMieMult;
    vSunColorIntensity.w = 9.2f - 550.0f * fMieMult;
    m_pSun->SetSunIntensity(vSunColorIntensity.w);
    hr = m_pEffect->SetVector(m_hSunColorInt,&vSunColorIntensity);
    m_mtlSunClr.Emissive.r = vSunColorIntensity.x * 1.5f;
    m_mtlSunClr.Emissive.g = vSunColorIntensity.y * 1.5f;
    m_mtlSunClr.Emissive.b = vSunColorIntensity.z * 1.5f;
    m_mtlSunClr.Ambient = m_mtlSunClr.Diffuse = m_mtlSunClr.Emissive;

    //calculate base height for density distance for a athmosphere
    //asuming a exponential density distribution dens = 1.2 * e^(h/8000)
    //h/8000 does not quite work, the eye is sensitive to log(intencity)
    //densbase = intggrate[alt to top] top dens defined = 0.0f
    float fDensityAltBase[4];
    fDensityAltBase[0] = vEye->y / 1000.0f;
    fDensityAltBase[1] = powf(0.95f,fDensityAltBase[0]);
    fDensityAltBase[2] = 20000.0f * expf(-vEye->y/10000.0f);
    fDensityAltBase[3] = 8000.0f * expf(-vEye->y/20000.0f);
    hr = m_pEffect->SetValue(m_hDensityDist,fDensityAltBase,16);

    D3DXVECTOR3 vBetaR, vBetaDashR, vBetaM, vBetaDashM, vBetaRM, vOneOverBetaRM;

    // Rayleigh
    vBetaR = m_pAtmosphere->GetBetaRayleigh();
    vBetaR *= fRayMult;
    m_pEffect->SetValue(m_hBetaRayleigh,&vBetaR,12);
    vBetaDashR = m_pAtmosphere->GetBetaDashRayleigh();
    vBetaDashR *= fRayMult;
    hr = m_pEffect->SetValue(m_hBetaDashRayleigh,&vBetaDashR,12);

    // Mie
    vBetaM = m_pAtmosphere->GetBetaMie();
    vBetaM *= fMieMult;
    m_pEffect->SetValue(m_hBetaMie,&vBetaM,12);
    vBetaDashM = m_pAtmosphere->GetBetaDashMie();
    vBetaDashM *= fMieMult;
    m_pEffect->SetValue(m_hBetaDashMie,&vBetaDashM,12);

    // Rayleigh + Mie
    vBetaRM = vBetaR + vBetaM;
    m_pEffect->SetValue(m_hBetaRayleighMie,&vBetaRM,12);
    vOneOverBetaRM[0] = 1.0f/vBetaRM[0];
    vOneOverBetaRM[1] = 1.0f/vBetaRM[1];
    vOneOverBetaRM[2] = 1.0f/vBetaRM[2];
    hr = m_pEffect->SetValue(m_hOneOverBetaRayleighMie,&vOneOverBetaRM,12);

    // each term (extinction, inscattering multiplier)
    float fIns = m_pAtmosphere->GetParam(eAtmInscatteringMultiplier);

    // Henyey Greenstein's G value. and inscatering
    float g = m_pAtmosphere->GetParam(eAtmHGg);
    //g = 0.293f + vEye->y * 1.18333e-5;
    //g = 0.193f + vEye->y * 1.03e-5;
    //g = 0.100f + vEye->y * 8.333e-6;
    //g = 0.163f + vEye->y * 1.393e-5;
    g = 0.444f + vEye->y * 1.053e-5 - 5.984e-2 * m_pSun->GetSunTheta() - 3.521e-6 *
        m_pSun->GetSunTheta() * vEye->y;
    float g2 = 0.100f + vEye->y * 1.11e-5 - 3.18e-2 * m_pSun->GetSunTheta() - 6.01e
        -6 * m_pSun->GetSunTheta() * vEye->y;
    float c = -0.076923f + 153.846f * fMieMult;
    g = g + c*(g2 - g);
    //g = 0.250f + vEye->y * 1.3e-5 - 0.00e-2 * m_pSun->GetSunTheta() - 9.2e-6 * m_pSun
        ->GetSunTheta() * vEye->y;
    D3DXVECTOR4 vG(1-g*g, 1+g*g, 2*g,fIns);
    m_pEffect->SetVector(m_hHenyeyGG,&vG);
    m_pAtmosphere->SetParam(eAtmHGg,g);

    // Eye Position
    m_pEffect->SetValue(m_hEyePosition,&vEye,12);

    // calculate the max light intensity
    float Ny = cosf(m_pSun->GetSunTheta());
    float k = powf(Ny,0.3f);
    float a = (1.05f - powf(Ny,0.3f))*190000;
    float Sr = a + k*(fDensityAltBase[2]-a);
    float Sm = a + k*(fDensityAltBase[3]-a);
    float betaRay0 = 2.0f;
    float betaMie0 = vG[0] / powf((vG.y - vG.z),(3.0f/2.0f));
    D3DXVECTOR3 vInMaxSun;
    for (int i=0; i<3; i++)
    {
        vInMaxSun[i] =  (vBetaDashR[i] * betaRay0 + vBetaDashM[i] * betaMie0) *
                        vOneOverBetaRM[i] * vSunColorIntensity[i] *
                        (1 - expf(-(vBetaR[i]*Sr + vBetaM[i]*Sm)));
    }

    Sr = Sm = 215000.0f;
    betaRay0 = 1.0f + Ny;
    betaMie0 = vG[0] / powf((vG.y - vG.z*Ny),(3.0f/2.0f));
    D3DXVECTOR3 vInMaxHoriz;
    for (int i=0; i<3; i++)
    {
        vInMaxHoriz[i] =    (vBetaDashR[i] * betaRay0 + vBetaDashM[i] * betaMie0) *
                            vOneOverBetaRM[i] * vSunColorIntensity[i] *
                            (1 - expf(-(vBetaR[i]*Sr + vBetaM[i]*Sm)));
    }
    //D3DXVec3Maximize(&vInMaxSun,&vInMaxSun,&vInMaxHoriz);
    //float maxval = max(vInMaxSun.x,max(vInMaxSun.y,vInMaxSun.z));
//  maxval = 1.0f / logf(maxval + 1.0f);
//  vG.w = maxval;

//  m_pEffect->SetVector(m_hHenyeyGG,&vG);
    for (int i=0; i<3; i++)
    {
        vInMaxSun[i] = 1.0f / logf(1.0f + vInMaxHoriz[i]);
    }
    m_pEffect->SetValue(m_hToneMap,&vInMaxSun,12);


    return hr;
}
```

E.3 Skydome.cpp

```
//now we need to fill the indexbufffer to actually draw the triangles
WORD* pIndex;
hr = m_pSystemMesh->LockIndexBuffer(0,(void**)&pIndex);
int count1 = 0;      //index of the internal vertex
int count2 = 1;      //index of the external vertex
int triIndex = 0;    //current index in the triangle buffer
for(int i=1; i < nLatBands; i++)
{
    //calculate the free vertex spacing
    float fSpacing = fabsf(bandinfo[i].nVertexCount / (float)bandinfo[i].
         nVertexDelta);
    int nSpaces = 0;
    int count1Start = count1;
    int count2Start = count2;
    int nextspecial = 0;
    for(int j=0; j < bandinfo[i].nVertexCount; j++)
    {
        //first we treat the speial case of the first point
        if(j == 0 && bandinfo[i].nVertexDelta > 0)
        {
            pIndex[triIndex]     = (WORD)count2;
            pIndex[triIndex+1]   = (WORD)count1;
            pIndex[triIndex+2]   = (WORD)count2+1;
            count2++;
            triIndex += 3;
            nSpaces++;
            nextspecial = (int)(nSpaces * fSpacing + 0.5);
        }
        else if(j == 0 && bandinfo[i].nVertexDelta < 0)
        {
            pIndex[triIndex]     = (WORD)count2;
            pIndex[triIndex+1]   = (WORD)count1;
            pIndex[triIndex+2]   = (WORD)count1+1;
            pIndex[triIndex+3]   = (WORD)count2;
            pIndex[triIndex+4]   = (WORD)count1+1;
            pIndex[triIndex+5]   = (WORD)count2+1;
            count1++;
            count2++;
            triIndex += 6;
            nSpaces++;
            nextspecial = (int)(nSpaces * fSpacing + 0.5);
        }
        else if (j == bandinfo[i].nVertexCount-1 && count1 == 0)
        {
                pIndex[triIndex]     = (WORD)count2;
                pIndex[triIndex+1]   = (WORD)count1;
                pIndex[triIndex+2]   = (WORD)count2Start;
                count2++;
                count1++;
                triIndex += 3;
        }
        else if (j == bandinfo[i].nVertexCount-1 && bandinfo[i].nVertexDelta
             >= 0 && count1 > 0)
        {
            pIndex[triIndex]     = (WORD)count2;
            pIndex[triIndex+1]   = (WORD)count1;
            pIndex[triIndex+2]   = (WORD)count1Start;
            pIndex[triIndex+3]   = (WORD)count2;
            pIndex[triIndex+4]   = (WORD)count1Start;
            pIndex[triIndex+5]   = (WORD)count2Start;
            count1++;
            count2++;
            triIndex += 6;
        }
        else if (j == bandinfo[i].nVertexCount-1 && bandinfo[i].nVertexDelta < 0)
        {
            pIndex[triIndex]     = (WORD)count2;
            pIndex[triIndex+1]   = (WORD)count1;
            pIndex[triIndex+2]   = (WORD)count1+1;
            pIndex[triIndex+3]   = (WORD)count2;
            pIndex[triIndex+4]   = (WORD)count1+1;
            pIndex[triIndex+5]   = (WORD)count2Start;
            pIndex[triIndex+6]   = (WORD)count2Start;
            pIndex[triIndex+7]   = (WORD)count1+1;
```

```
                pIndex[triIndex+8]    = (WORD)count1Start;
                count1+=2;
                count2++;
                triIndex += 9;
            }
            else if(j == nextspecial && bandinfo[i].nVertexDelta > 0)
            {
                    pIndex[triIndex]      = (WORD)count2;
                    pIndex[triIndex+1]    = (WORD)count1;
                    pIndex[triIndex+2]    = (WORD)count2+1;
                    count2++;
                    triIndex += 3;
                    nSpaces++;
                    nextspecial = (int)(nSpaces * fSpacing + 0.5);
            }
            else if(j == nextspecial && bandinfo[i].nVertexDelta < 0)
            {
                    pIndex[triIndex]      = (WORD)count2;
                    pIndex[triIndex+1]    = (WORD)count1;
                    pIndex[triIndex+2]    = (WORD)count1+1;
                    count1++;
                    j--;
                    triIndex += 3;
                    nSpaces++;
                    nextspecial = (int)(nSpaces * fSpacing + 0.5);
            }
            else
            {
                pIndex[triIndex]      = (WORD)count2;
                pIndex[triIndex+1]    = (WORD)count1;
                pIndex[triIndex+2]    = (WORD)count1+1;
                pIndex[triIndex+3]    = (WORD)count2;
                pIndex[triIndex+4]    = (WORD)count1+1;
                pIndex[triIndex+5]    = (WORD)count2+1;
                count1++;
                count2++;
                triIndex += 6;
            }
        }
    }
```