



Timothy R. Kol

REAL-TIME CLOUD RENDERING  
ON THE GPU

Master thesis

UTRECHT UNIVERSITY

ICA-3810275

November 2013



# Universiteit Utrecht

GAME AND MEDIA TECHNOLOGY  
DEPARTMENT OF INFORMATION AND COMPUTING SCIENCES

Master thesis by Timothy R. Kol  
Student number 3810275

Supervised by dr. Robby T. Tan, Utrecht University

16<sup>th</sup> November, 2013

# Abstract

In the quest for realistic rendering of outdoor scenes, clouds form a crucial component. The sky is a prominent feature for many such scenes, and seldom is the air devoid of clouds. Their great variety in shape, size and color poses a difficult problem in the pursuit of convincing visualization. Additionally, the complexity of light behavior in participating media like clouds, as well as their sheer size, often cause photo-realistic approaches to have exceedingly high computational costs.

This thesis presents a method for rendering realistic, animated cumuliform clouds through a phenomenological approach. Taking advantage of the parallel processing power of the GPU, the cloud appearance is computed per pixel, resulting in real-time full-screen visualization even on low-end GPU's.

The pixel color is determined by three components. First, the opacity is computed based on a cloud mesh consisting of a set of ellipsoids, upon which a fractal noise texture is superimposed to generate detailed cloud features. Second, the single scattering component is analytically calculated using the same triangle mesh and a discretized Mie phase function. Finally, the multiple scattering component is obtained from a phenomenological model using trigonometric and logarithmic formulas with coefficients based on the view and sun angles and the cloud thickness at the current pixel. The parameters for this model are based on the work of Bouthors et al., who studied light transport in plane-parallel slabs. We apply this model in a more robust manner to find the best-fitting plane-parallel slab, providing for a better and faster alternative to the original approach.

The clouds are animated at no extra cost by translating them across the scene, which creates a simulation of turbulence, due to the fact that the noise texture is sampled by the world coordinates of a pixel. Additionally, cloud formation and dissipation is simulated by modifying the water droplet density.

**Keywords:** Computer graphics, cloud rendering, light scattering, GPU programming.

# Acknowledgments

This endeavor has been a voyage through peaks and dales, and it is only right to give gratitude where it is due, thus I would like to extend my thanks to my family, who pulled me through several setbacks.

Naturally, I am also much obliged to my supervisor dr. Robby T. Tan, who, despite the fact that his own research lies more within the field of computer *vision* rather than graphics, was a tremendous help in cutting through the implementation obstacles that occupied my mind during our weekly meetings to see the bigger picture and separate the important from the trivial matters. His guidance provided for a warm welcome into academic thinking, and his flexibility was beneficial for catering to a high degree of independence.

Furthermore, I would like to thank my fellow students Mauro van de Vlasakker and Michel Sussenbach for their comments during the weekly meetings with Robby Tan, as well as several fruitful discussions.

In addition, I am grateful to prof. dr. Elmar Eisemann of Delft University of Technology, who allowed me to present my work before his research group. Moreover, our conversations cleared up several things for me, as he is familiar with the original Bouthors paper.

Many thanks go to Antoine Bouthors and Fabrice Neyret as well, whose paper – the main reference of this thesis – not only sparked my interest in cloud rendering, but who also communicated with me on several occasions to extensively answer my questions while brainstorming on possible improvements I could implement.

The title page background is based on a photograph provided by Santhosh Kumar on his website <http://www.freeimagescollection.com/>.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Goal . . . . .	1
1.2.1	Cloud genera . . . . .	1
1.2.2	Scope . . . . .	2
1.2.3	Goal . . . . .	4
1.2.4	Applications . . . . .	4
1.2.5	Approach . . . . .	4
1.3	Overview . . . . .	4
<b>2</b>	<b>Background and related work</b>	<b>5</b>
2.1	Preliminaries . . . . .	5
2.1.1	Spherical coordinates . . . . .	5
2.1.2	Intersection . . . . .	7
2.1.3	Solid angles . . . . .	8
2.1.4	Radiometry and photometry . . . . .	9
2.1.5	BRDF . . . . .	14
2.1.6	BSSRDF . . . . .	15
2.1.7	Phase function . . . . .	16
2.1.8	Extinction . . . . .	17
2.1.9	Radiative transfer equation . . . . .	18
2.2	Cloud features . . . . .	19
2.2.1	Fractal shape . . . . .	19
2.2.2	Cloud edges . . . . .	20
2.2.3	Multiple scattering . . . . .	21
2.2.4	Silver lining and other contrast . . . . .	21
2.2.5	Glory and fogbow . . . . .	22
2.2.6	Cloud lighting . . . . .	22
2.2.7	Cloud movement . . . . .	23
2.3	Related work . . . . .	24
2.3.1	Modeling cloud shapes . . . . .	25
2.3.2	Rendering the colors of a cloud . . . . .	31
2.3.3	Animating cloud movement . . . . .	35
2.3.4	Conclusion . . . . .	36
<b>3</b>	<b>Cloud rendering</b>	<b>37</b>
3.1	Methods . . . . .	37
3.1.1	The cloud model . . . . .	37
3.1.2	Opacity and single scattering . . . . .	39
3.1.3	A study of light transport in plane-parallel slabs . . . . .	44
3.1.4	An efficient phenomenological model of light transport in clouds . . . . .	48
3.1.5	Combining the components . . . . .	56

3.2	Drawbacks . . . . .	57
3.2.1	Model . . . . .	57
3.2.2	Opacity and single scattering . . . . .	57
3.2.3	Multiple scattering . . . . .	59
3.3	Hypotheses . . . . .	61
3.3.1	Model . . . . .	61
3.3.2	Opacity and single scattering . . . . .	62
3.3.3	Multiple scattering . . . . .	63
3.4	Implementation notes . . . . .	63
3.4.1	OpenGL and GPU programming . . . . .	63
3.4.2	Blending and depth testing . . . . .	64
3.4.3	Depth maps and mipmapping . . . . .	65
3.4.4	Ellipsoids . . . . .	66
3.4.5	Phenomenological model . . . . .	69
3.4.6	Noise evaluation . . . . .	70
3.4.7	Animation . . . . .	70
<b>4</b>	<b>Results and analysis</b>	<b>72</b>
4.1	Experiment conditions . . . . .	72
4.1.1	Performance . . . . .	72
4.1.2	Accuracy . . . . .	72
4.2	Performance . . . . .	72
4.2.1	Hypotheses . . . . .	73
4.3	Accuracy . . . . .	75
4.3.1	Fractal shape . . . . .	75
4.3.2	Cloud edges . . . . .	75
4.3.3	Multiple scattering . . . . .	77
4.3.4	Silver lining and other contrast . . . . .	79
4.3.5	Glory and fogbow . . . . .	79
4.3.6	Cloud lighting . . . . .	79
4.3.7	Cloud movement . . . . .	79
4.3.8	Hypotheses . . . . .	80
<b>5</b>	<b>Conclusions</b>	<b>82</b>
5.1	Contributions . . . . .	82
5.2	Conclusion . . . . .	82
5.3	Future work . . . . .	83
5.4	Reflections . . . . .	84

# Chapter 1

## Introduction

### 1.1 Motivation

With the technological advancements in hardware, the game and film industries are continually on the lookout for new methods in order to further develop the realism of their respective products. For both, creating an engrossing and convincing environment is of critical importance as customers' expectations grow higher every year. Computer graphics researchers unceasingly attempt catering to this end, often with an eye on the future, as computational power, especially with respect to the graphical processing unit (GPU), continues to increase. Considering the possibility that we may have arrived at the point where people's assumptions of development in the computer hardware industry surpass the actual advancements, researching more efficient solutions is more important than ever, especially with the possible failing of Moore's law [Moo65] – stating that microchip transistor count doubles every two years – in the near future [EBA<sup>+</sup>11] [Mac11] [RS11].

Among common visualization challenges, natural outdoor scenes remain one of the most problematic tasks. One difficulty of such scenes consists in the behavior of light, i.e., its interaction with matter. In particular, clouds form a troublesome subject, and their importance is indisputable. For outdoor environments, the sky often makes up a significant part of the scene, and across the globe, clouds are more often present than not, as illustrated by Figure 1.1. Rendering convincing clouds is especially relevant in special effects pertaining to aerial scenes and flight games or simulations, yet it is also beneficial for other open-air applications, like panoramic outdoor film scenes or any games making use of open world level design.

### 1.2 Goal

#### 1.2.1 Cloud genera

Before being able to concretely define the goal for this thesis, we need to consider the different types of clouds. There are ten main genera to which a cloud can be assigned, based on its altitude, shape and precipitation [Org75]. Latin nomenclature regarding these three characteristics is defined as follows.

High altitude clouds get the prefix *cirro-*, which means *lock of hair*, signifying a wispy texture. They appear between five and twelve kilometers above the earth's surface, and generally consist of ice crystals of varying shape due to the freezing temperatures. Medium-height clouds are preceded by *alto-*, and can be found at an altitude between two and seven kilometers. They largely contain water droplets. Remarkably, their prefix literally translates to *high*. Low-altitude clouds have no height-related prefix. They consist of water droplets and appear below an altitude of three kilometers.

Besides height, the main genera can be grouped based on their shape as well. Layer-like clouds are preceded by *strato-*, meaning *sheet*, with the subgenus described as stratiform. The well-known, puffy, heap-like clouds on the other hand, get a prefix *cumulo-*, translating to *heap*, and are collectively denoted

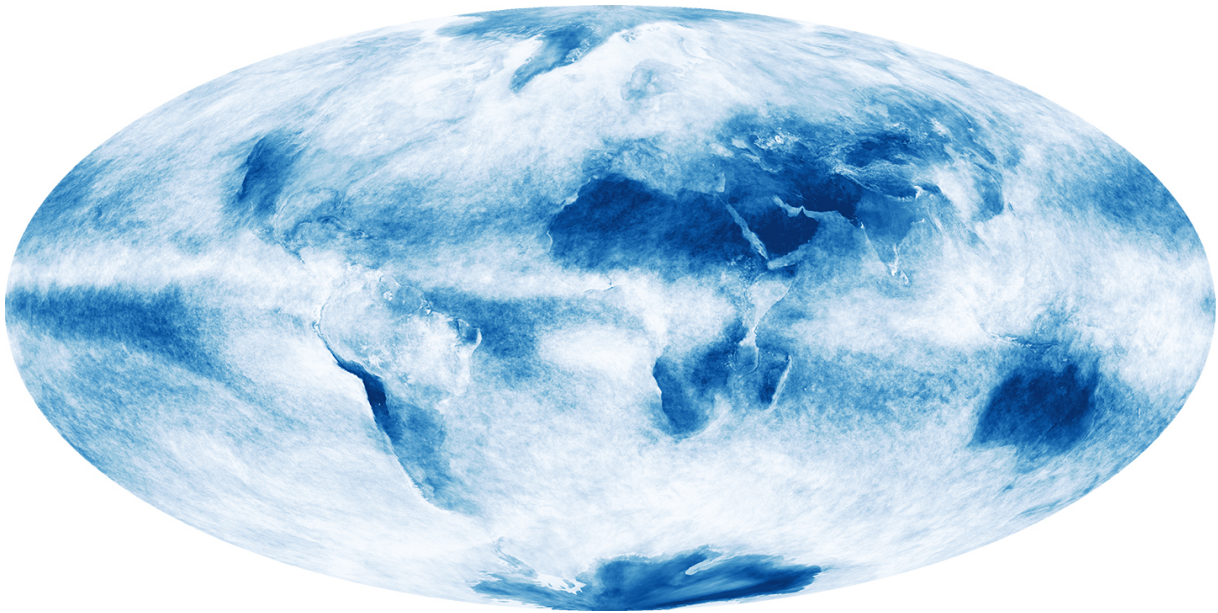


Figure 1.1: Global cloud cover averaged over October 2009, with dark patches corresponding to 0% coverage, and white areas to 100%. It is interesting to note that the contours of the continents are clearly distinguishable for many locations, indicating a sharp difference between cloud coverage over land and sea in those areas [Lin09].

as cumuliform clouds. Finally, clouds that produce precipitation are denoted by a *nimbo-* prefix, meaning *storm cloud*.

Combining these three characteristics, we arrive at the following ten genera. Cirrus, cirrocumulus and cirrostratus are high-altitude clouds consisting of ice crystals. Altocumulus and altostratus are medium-height clouds. Finally, cumulus, stratocumulus, stratus, nimbostratus and cumulonimbus clouds dominate the lower regions of the atmosphere, where it should be noted that the latter, despite having a low-altitude base, has an immense vertical extent so that its peak reaches well into the medium-altitude region, and sometimes even surpasses the height of cirrus clouds. All of these cloud types are displayed in Figure 1.2.

### 1.2.2 Scope

The goal of this thesis is to introduce a real-time rendering method capable of realistically visualizing cumuliform clouds. We leave stratiform clouds out of the scope, as they require a quite different approach, taking into account their layer-like structure. Moreover, stratiform clouds contain less features than cumuliform clouds, as shown in Figure 1.2. Stratus clouds, for instance, often simply consist of a uniform gray layer. Despite being common – they are dominating the sky at the moment of writing – we will not consider them, focusing on cumuliform clouds instead, which provide for the most interesting challenge due to their vertical extent.

Narrowing our scope down some more, we assume that the cumuliform clouds consist of water droplets only, thus ignoring cirrocumulus clouds. The reason for this is that ice crystals interact with light differently, partly due to their asymmetrical shape. This causes several atmospheric phenomena [LL01], which, however interesting, will not be considered throughout this thesis, as several of the assumptions in ensuring correctness of the visualization rely on the fact that the cloud consists of water droplets only.



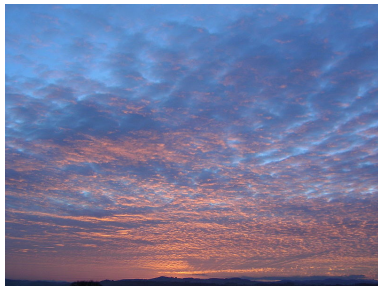
(a) Cirrus. ©📷📷 PiccoloNamek / Wikimedia Commons.



(b) Cirrocumulus. ©📷📷 Typhoon chaser / Wikimedia Commons.



(c) Cirrostratus. ©📷📷 Simon Eugster / Wikimedia Commons.



(d) Altostratus. ©📷📷 Simon Eugster / Wikimedia Commons.



(e) Altostratus. ©📷📷 The Great Cloudwatcher / Wikimedia Commons.



(f) Nimbostratus. ©📷📷 Simon Eugster / Wikimedia Commons.



(g) Cumulus. ©📷📷 PiccoloNamek / Wikimedia Commons.



(h) Stratus. ©📷📷 PiccoloNamek / Wikimedia Commons.



(i) Stratocumulus. ©📷📷 Nicholas A. Tonelli.



(j) Cumulonimbus. ©📷📷 JezFabi / Wikimedia Commons.

Figure 1.2: The ten main cloud genera. The first row shows high-altitude clouds, the first two images of the second row are medium-altitude, and the remaining show low-altitude clouds, or in the case of the cumulonimbus, with a low-altitude base. Cumuliform clouds consist of cirrocumulus, altostratus, cumulus and cumulonimbus; while cirrus, cirrostratus, altostratus, nimbostratus and stratus are stratiform. Stratocumulus, as the name suggests, is a hybrid type of cloud.



### 1.2.3 Goal

To elaborate, we want to be able to render a cumuliform cloud during daylight with at least fifteen frames per second on low-end GPU's at a full-screen resolution, while allowing for a moving camera and sun, preferably animating the cloud as well. The results must stay true to all visual features of cumuliform water droplet clouds, and meet or even surpass current state-of-the-art visualizations that attain interactive frame rates.

Current methods fail to achieve this goal, often by considering highly complex models or complicated light interaction rules which take an excessive amount of computational power. Approaches that have managed to obtain frame rates of over 15 FPS have not accomplished convincing enough visualizations due to inaccurate approximations. A detailed breakdown of current methods and their shortcomings can be found in chapter 2.

The two main problems to overcome are the construction of a realistic, high-detailed cloud model while still attaining real-time frame rates, and a convincing approximation of light behavior inside a cloud – i.e., with relation to water droplets.

### 1.2.4 Applications

We acknowledge the fact that striving for real-time performance does not suffice in being justified to suggest games as an application for this method, however, on high-end GPU's, especially considering the technological advancements in store for the future, we certainly believe this approach could attain acceptable frame rates to be incorporated in a game engine. Flight simulators and other aerial games in particular are well-suited for the implementation of the presented method to greatly enhance immersion.

Other than games, the method would be suitable for low-budget special effects, with preview visualizations in particular, where there are no resources for heavy rendering and time is of the essence.

### 1.2.5 Approach

The problems are approached by considering the state-of-the-art method that best manages to achieve the aforementioned goal, described by Bouthors et al. in [BNM<sup>+</sup>08]. Their method will be implemented and improved upon, with special consideration for the bottlenecks of their technique, which are, according to them, the implementation of Perlin noise on the GPU in order to generate a high-detail cloud model, and an iterative algorithm to approximate light behavior in a cloud. Their results will be evaluated both performance- and accuracy-wise to highlight the opportunities for improvement. Additionally, we will investigate the possibility of animating the cloud to add more realism.

All of this will be integrated in a framework consisting of an implementation [Kol12] of a realistic skylight model [HW12] so that the authenticity of the cloud is more fairly judged.

## 1.3 Overview

Chapter 2 will discuss the background and related work in this field, first elucidating some preliminaries with special attention to light behavior in participating media, then discussing the cloud features that we wish to reproduce, and finally giving an overview of related work.

Chapter 3 discusses the theory behind cloud rendering, first explaining the method presented by Bouthors et al., subsequently illuminating its drawbacks and presenting the corresponding hypotheses for solving them. Finally, a section is devoted to implementation notes to aid future researchers.

Chapter 4 contains an analysis of the results of the method proposed in this thesis, consisting of a comparison with existing techniques, and a discussion of how well the cloud features are reproduced. The results of this evaluation are discussed, along with a justification of the advantages and disadvantages of this approach.

Finally, chapter 5 concludes the thesis by reiterating the contributions as well as discussing their drawbacks while proposing hypotheses to overcome these as future work. Finally, the whole project is reflected upon.

## Chapter 2

# Background and related work

### 2.1 Preliminaries

#### 2.1.1 Spherical coordinates

The sky model that serves as a framework for cloud rendering makes use of both *Cartesian* and *spherical* coordinates. Figure 2.1 shows a vector  $\vec{s}$ , which can be defined by Cartesian coordinates  $(x, y, z)$  on one hand, and spherical coordinates  $(\theta, \phi)$  and length  $r = \|\vec{s}\|$  on the other.

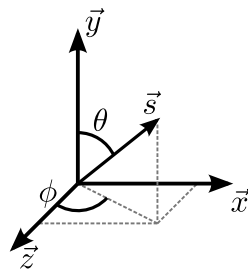


Figure 2.1: Cartesian and spherical coordinates.

Conversion from one coordinate system to the other is frequently applied in these circumstances, and is done as follows. Given Figure 2.1 and vector  $\vec{s} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$ , the radius  $r$ , altitude  $\theta$  and azimuth  $\phi$  are given by equations (2.1), (2.2) and (2.3).

$$r = \|\vec{s}\| = \sqrt{x^2 + y^2 + z^2} \quad (2.1)$$

$$\theta = \arccos\left(\frac{y}{r}\right) \quad (2.2)$$

$$\phi = \arctan\left(\frac{x}{z}\right) \quad (2.3)$$

Note that for equation (2.3), the quadrant as well as the case where  $z = 0$  need to be taken into account. For this, many programming libraries supply an additional function `atan2()` besides the common `atan()`, which, taking two arguments instead of one, correctly handles all cases.

On the other hand, given the spherical coordinates and the radius, the Cartesian coordinates may be retrieved using equations (2.4), (2.5) and (2.6).

$$x = r \sin \phi \sin \theta \quad (2.4)$$

$$y = r \cos \theta \quad (2.5)$$

$$z = r \cos \phi \sin \theta \quad (2.6)$$

Note that these equations are dependent on the notational convention regarding the spherical coordinates and the Cartesian axes, and as such may be found to differ for other sources. We make use of the coordinate system that is used by OpenGL, the API in which the methods of this thesis have been implemented.

In the case of ellipsoids, the equations differ somewhat, especially with respect to the spherical, or rather, ellipsoidal coordinates. As illustrated by Figure 2.2, an interesting phenomenon occurs due to the distinct radii of an ellipsoid, which are also called the semi-principal axes.

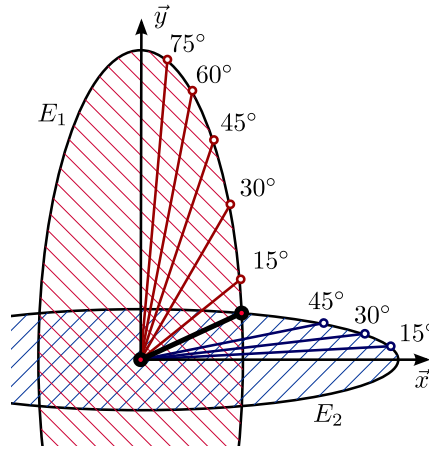


Figure 2.2: The ellipsoidal coordinates can vary tremendously for different ellipsoids or ellipses. Consider the bold black line denoting a certain angle. For red ellipse  $E_1$  the ellipsoidal coordinate will be lower than  $15^\circ$ , while for blue  $E_2$ , it will be in the vicinity of  $60^\circ$  or  $70^\circ$ .

Equations (2.7), (2.8) and (2.9) need the semi-principal axes  $(r_x, r_y, r_z)$  as input to produce the correct ellipsoidal coordinates.

$$r_e = \|\vec{s}\| = \sqrt{x^2 + y^2 + z^2} \quad (2.7)$$

$$\theta_e = \arccos\left(\frac{\frac{y}{r_e}}{\frac{r_y}{r_e}}\right) = \arccos\left(\frac{y}{r_y}\right) \quad (2.8)$$

$$\phi_e = \arctan\left(\frac{\frac{x}{r_x}}{\frac{z}{r_z}}\right) \quad (2.9)$$

There are few applications where these equations are of any use, due to the fact that the semi-principal axes need to be supplied. However, an adaptation of them is used in the implementation described in chapter 3.

The conversion from ellipsoidal to Cartesian coordinates is more commonly used, and given by equations (2.10), (2.11) and (2.12).

$$x = r_x \sin \phi_e \sin \theta_e \quad (2.10)$$

$$y = r_y \cos \theta_e \quad (2.11)$$

$$z = r_z \cos \phi_e \sin \theta_e \quad (2.12)$$



### 2.1.2 Intersection

Computing intersection points is not as trivial as it may seem, and it is an important part of the implementation of the cloud rendering model discussed in chapter 3. To obtain the intersection points between a line and a sphere, we consider the definitions of the two, given by equation (2.13) for a line and (2.14) for a sphere.

$$\vec{p} = \vec{p}_0 + t\vec{v} \quad (2.13)$$

$$1 = \left( \frac{\vec{p} - \vec{c}_0}{r} \right)^2 \quad (2.14)$$

$$1 = \left( \frac{\vec{p} - \vec{c}_0}{r} \right) \cdot \left( \frac{\vec{p} - \vec{c}_0}{r} \right)$$

For the line,  $\vec{p}$  denotes a point on the line,  $\vec{p}_0$  is the origin of the line,  $t$  is the distance along the line from  $\vec{p}$  to  $\vec{p}_0$ , and  $\vec{v}$  is a unit vector denoting the direction of the line. For the sphere,  $\vec{p}$  denotes a point on the surface of the sphere,  $\vec{c}_0$  is the sphere's origin and  $r$  is the radius. We can now solve for  $t$  using the quadratic formula. First replace  $\vec{p}$  in (2.14) with (2.13), as shown in (2.15). Then rewrite (2.15) to a quadratic equation of  $t$  to obtain (2.16).

$$1 = \left( \frac{\vec{p}_0 + t\vec{v} - \vec{c}_0}{r} \right) \cdot \left( \frac{\vec{p}_0 + t\vec{v} - \vec{c}_0}{r} \right) \quad (2.15)$$

$$0 = t^2 \left( \frac{\vec{v}}{r} \cdot \frac{\vec{v}}{r} \right) + 2t \left( \frac{\vec{v}}{r} \cdot \left( \frac{\vec{p}_0}{r} - \frac{\vec{c}_0}{r} \right) \right) + \left( \frac{\vec{p}_0}{r} - \frac{\vec{c}_0}{r} \right) \cdot \left( \frac{\vec{p}_0}{r} - \frac{\vec{c}_0}{r} \right) - 1 \quad (2.16)$$

Subsequently identifying the constants  $a$ ,  $b$  and  $c$ , we get (2.17), (2.18) and (2.19), which we can plug into the quadratic formula to obtain  $t$ , as in (2.20).

$$a = \frac{\vec{v}}{r} \cdot \frac{\vec{v}}{r} \quad (2.17)$$

$$b = 2 \left( \frac{\vec{v}}{r} \cdot \left( \frac{\vec{p}_0}{r} - \frac{\vec{c}_0}{r} \right) \right) \quad (2.18)$$

$$c = \left( \frac{\vec{p}_0}{r} - \frac{\vec{c}_0}{r} \right) \cdot \left( \frac{\vec{p}_0}{r} - \frac{\vec{c}_0}{r} \right) - 1 \quad (2.19)$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (2.20)$$

Note that when  $\sqrt{b^2 - 4ac} < 0$ , there is no solution, thus, no intersection either, and when  $\sqrt{b^2 - 4ac} = 0$ , the line intersects the sphere just once, so that it is a tangent line. To obtain the intersection points, we fill in the two values of  $t$ , which we call  $t_1$  and  $t_2$ , in equation (2.13). We now have the two intersection points,  $\vec{p}_1$  and  $\vec{p}_2$ , as per (2.21) and (2.22).

$$\vec{p}_1 = \vec{p}_0 + t_1\vec{v} \quad (2.21)$$

$$\vec{p}_2 = \vec{p}_0 + t_2\vec{v} \quad (2.22)$$

An ellipsoid has a slightly different form, as shown by equation(2.23). The vectors denoting a point on the ellipsoid boundary and the origin are fully written out because they need to be divided by the radii of the ellipsoid.

$$\left( \left[ \begin{array}{c} \frac{p_x}{r_x} \\ \frac{p_y}{r_y} \\ \frac{p_z}{r_z} \end{array} \right] - \left[ \begin{array}{c} \frac{c_{0x}}{r_x} \\ \frac{c_{0y}}{r_y} \\ \frac{c_{0z}}{r_z} \end{array} \right] \right)^2 = 1 \quad (2.23)$$

Solving for  $t$  is done similarly as for a sphere, resulting in the quadratic equation constants (2.24), (2.25) and (2.26).

$$a = \begin{bmatrix} \frac{v_x}{r_x} \\ \frac{v_y}{r_y} \\ \frac{v_z}{r_z} \end{bmatrix} \cdot \begin{bmatrix} \frac{u_x}{r_x} \\ \frac{u_y}{r_y} \\ \frac{u_z}{r_z} \end{bmatrix} \quad (2.24)$$

$$b = 2 \left( \begin{bmatrix} \frac{v_x}{r_x} \\ \frac{v_y}{r_y} \\ \frac{v_z}{r_z} \end{bmatrix} \cdot \left( \begin{bmatrix} \frac{p_{0x}}{r_x} \\ \frac{p_{0y}}{r_y} \\ \frac{p_{0z}}{r_z} \end{bmatrix} - \begin{bmatrix} \frac{c_{0x}}{r_x} \\ \frac{c_{0y}}{r_y} \\ \frac{c_{0z}}{r_z} \end{bmatrix} \right) \right) \quad (2.25)$$

$$c = \left( \begin{bmatrix} \frac{p_{0x}}{r_x} \\ \frac{p_{0y}}{r_y} \\ \frac{p_{0z}}{r_z} \end{bmatrix} - \begin{bmatrix} \frac{c_{0x}}{r_x} \\ \frac{c_{0y}}{r_y} \\ \frac{c_{0z}}{r_z} \end{bmatrix} \right) \cdot \left( \begin{bmatrix} \frac{p_{0x}}{r_x} \\ \frac{p_{0y}}{r_y} \\ \frac{p_{0z}}{r_z} \end{bmatrix} - \begin{bmatrix} \frac{c_{0x}}{r_x} \\ \frac{c_{0y}}{r_y} \\ \frac{c_{0z}}{r_z} \end{bmatrix} \right) - 1 \quad (2.26)$$

As before, we fill in the two values of  $t$  in equation (2.13) to obtain the intersection points  $\vec{p}_1$  and  $\vec{p}_2$ .

### 2.1.3 Solid angles

A *solid angle* denotes a collection of directions, as an area defines a set of points. When observing an object in three-dimensional space from a certain point, called the vertex, all directions originating from this point that intersect the object, form the solid angle that is *subtended* by the object. Figure 2.3 shows vertex  $\vec{p}$  and the solid angle  $\Omega$ , denoted by the dashed lines, subtended by object  $C$ .

Thus, the solid angle can be viewed as a measurement of how large an object appears when observed from the vertex. For example, the sun and the moon both appear to be of a size when beheld from the earth, as the sun's vastly greater size is negated by the moon's proximity to our planet.

The solid angle's unit is the dimensionless steradian, with its quantity being equal to the area on a sphere centered at the vertex, that is covered by the object, divided by the squared radius of this sphere, as seen below in equation (2.27).

$$\Omega = \frac{A}{r^2} \quad (2.27)$$

In the special case of the unit sphere (i.e.,  $r = 1$ ), this means that the projected area simply equals the amount of steradians in the solid angle. In Figure 2.3, the shaded area, a projection of the object on the unit sphere  $U$ , is denoted by  $A$ .

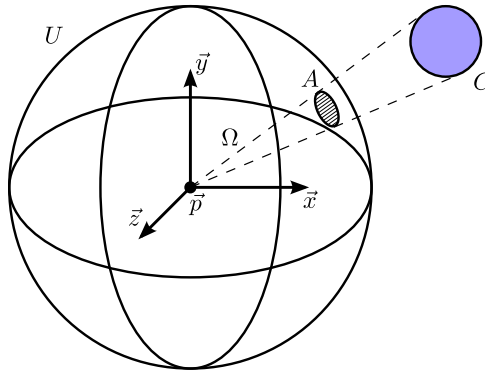


Figure 2.3: Solid angle  $\Omega$  subtended by object  $C$ .

With this knowledge we can derive the solid angle of a complete sphere. The area of a sphere is given by  $A(r) = 4\pi r^2$ , thus, this becomes  $\Omega = \frac{A(r)}{r^2} = \frac{4\pi r^2}{r^2} = 4\pi$  sr.

### 2.1.4 Radiometry and photometry

*Radiometry* is used for measuring all electromagnetic radiation – photons carrying energy, traveling through space – of which humanly visible light forms but a small portion, as illustrated by Figure 2.4. *Photometry*, on the other hand, solely focuses on the measurement of visible light with respect to the brightness sensitivity of the human eye.

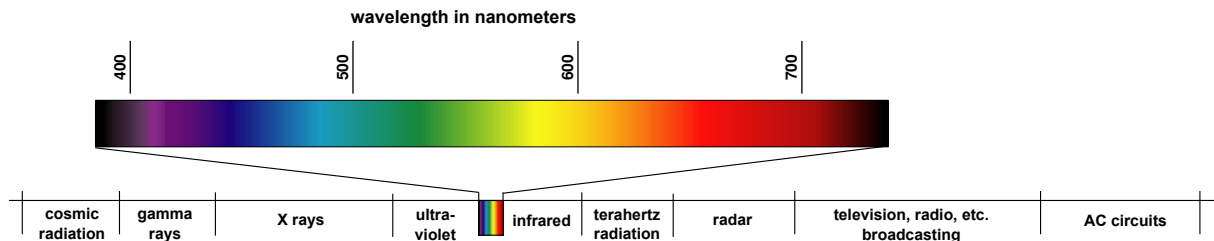


Figure 2.4: The humanly visible light spectrum compared to the full electromagnetic radiation spectrum. © Jonas Tullus.

#### Radiant and luminous energy

The energy of a single photon in joules is characterized by its wavelength, and can simply be computed using equation (2.28). It can easily be observed from this formula that the energy is inversely proportional to the wavelength of a photon. We take  $c$  the speed of light in vacuum, and assume this to be valid for all cases considered –  $h$  is a constant.

$$E = \frac{hc}{\lambda} \quad (2.28)$$

$$h \approx 6.626 \cdot 10^{-34} \text{ J}\cdot\text{s}$$

$$c \approx 2.998 \cdot 10^8 \text{ m}\cdot\text{s}^{-1}$$

The radiometric quantity of *radiant energy*,  $Q_e$ , measured in joules, denotes the energy that electromagnetic radiation transports through space, and can be seen as the combined energy of all the photons that constitute this radiation.

We can also define the *spectral radiant energy*,  $Q_{e\lambda}$ , which is the radiant energy per unit wavelength, measured in  $\text{J}\cdot\text{m}^{-1}$ . It allows us to generate a plot of radiant energy against the wavelength. Its formal definition is given in equation (2.29).

Note that for radiation consisting of an unchanging amount of photons, the radiant energy is inversely proportional to the wavelength, i.e., gamma rays relatively carry vast amounts of energy when compared to radio broadcasting waves, as per Figure 2.4.

$$Q_{e\lambda} = \frac{dQ_e}{d\lambda} \quad (2.29)$$

To reiterate, the radiant energy denotes the total amount of energy present over all wavelengths, while the spectral radiant energy denotes the amount of energy present at a certain wavelength interval. The spectral component and its definition is equivalently applicable to all radiometric quantities that are discussed in this section and thus will henceforth be omitted.

In photometry, we are interested in the energy of a light ray as perceived by the human eye. This means that this *luminous energy*,  $Q_v$ , depends on our eyes' sensitivity to certain wavelengths. The unit of luminous energy is the lumen second ( $\text{lm}\cdot\text{s}$ ). Figure 2.5 shows a plot of the average human eye response against the wavelength of light, called the standard luminosity function  $\bar{y}(\lambda)$  [SG31]. It is clear that for electromagnetic radiation outside the visible spectrum, this function will return zero.

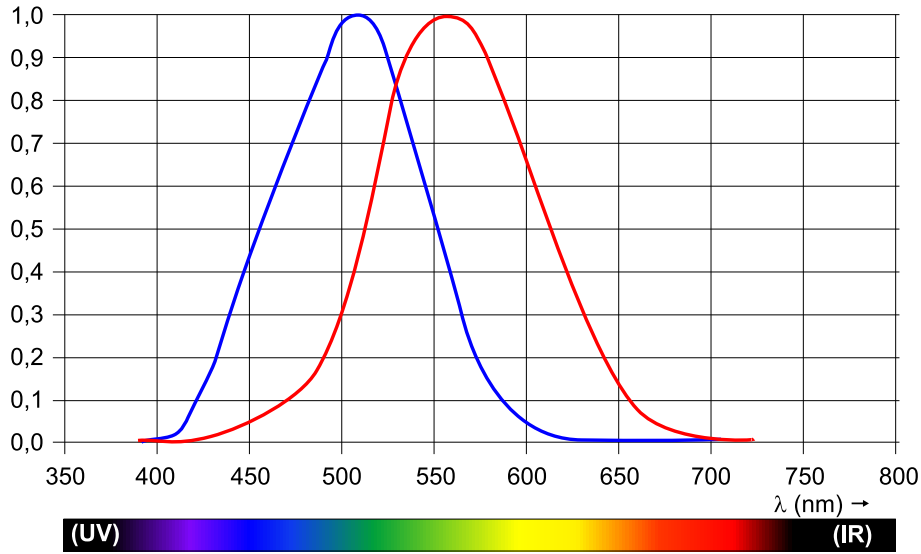


Figure 2.5: The luminosity function  $\bar{y}(\lambda)$  showing eye response for photopic (red line) and scotopic vision (blue line). Note that the function is experimentally constructed and prone to small errors, yet still a good representation. © H. Hahn / Wikimedia Commons.

To make matters more complicated, this function is different for bright light – photopic vision, perceived by cone cells in our eyes – and dim light – scotopic vision, perceived by rod cells, which can not distinguish colors. However, since we are focusing on daytime cloud rendering, the scotopic luminosity function may be ignored.

To obtain the luminous energy, the spectral radiant energy of radiation is weighted with the luminosity function. This results in a relatively high  $Q_v$  for wavelengths near  $\lambda = 555$  nm, while radiation outside the visible spectrum will have no luminous energy at all, no matter how potent.

Note that spectral radiometric quantities are required in order to obtain the corresponding photometric quantity. After all, the luminosity function is weighted per wavelength interval, which requires the radiometric value specific to this interval. For the luminous energy, for example, the spectral radiant energy is weighted with  $\bar{y}(\lambda)$  as it is integrated over the spectrum of visible light to produce  $Q_v$ .

### Radiant and luminous flux

The *radiant flux*,  $\Phi_e$ , also called radiant power, is defined as the amount of energy transported by electromagnetic radiation per unit time. It is measured in joules per second, i.e., watts (W). The radiant flux is given by equation (2.30).

$$\Phi_e = \frac{dQ_e}{dt} \quad (2.30)$$

The photometric version is called the *luminous flux*,  $\Phi_v$ , and is defined as the energy of visible light perceived by the human eye per unit time, measured in lumens. One lumen is defined as the luminous flux of a light source emitting radiation with a wavelength  $\lambda = 555$  nm and a radiant flux of  $\Phi_e = \frac{1}{683}$  W.

To obtain the luminous flux of a source that emits radiation non-uniformly over a broad interval of wavelengths, we need to integrate the spectral radiant flux over the spectrum of visible light while weighting it with the luminosity function, and multiply the result by 683, as shown in equation (2.31).

$$\Phi_v = 683 \int_0^\infty \bar{y}(\lambda) \Phi_{e\lambda} d\lambda \quad (2.31)$$

### Radiant and luminous flux density

The *radiant flux density* is the amount of radiant flux that is arriving at, passing through, or emitted from a surface area. It can also be expressed as the amount of energy transferred by electromagnetic radiation per unit time per unit area, and is measured in watts per square meter. The radiant flux density can take two forms; when flux is incident on the surface, we speak of *irradiance*,  $E_e$ , while when radiation is passing through or is emitted by a surface, we consider *radiant exitance*,  $M_e$ . The formal definitions are given in equations (2.32) and (2.33), with Figure 2.6 illustrating both forms of flux density.

$$E_e = \frac{d\Phi_e}{dA} \quad (2.32)$$

$$M_e = \frac{d\Phi_e}{dA} \quad (2.33)$$

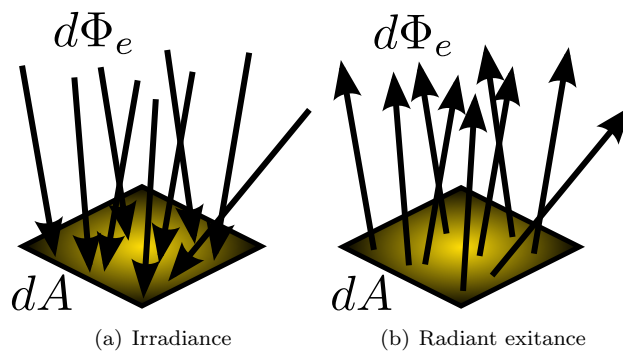


Figure 2.6: Radiant flux density denotes either irradiance ( $E_e$ ) or radiant exitance ( $M_e$ ).

Irradiance and spectral irradiance are units that are often used to quantify sources of radiation. For instance, the spectral solar irradiance is plotted in Figure 2.7.

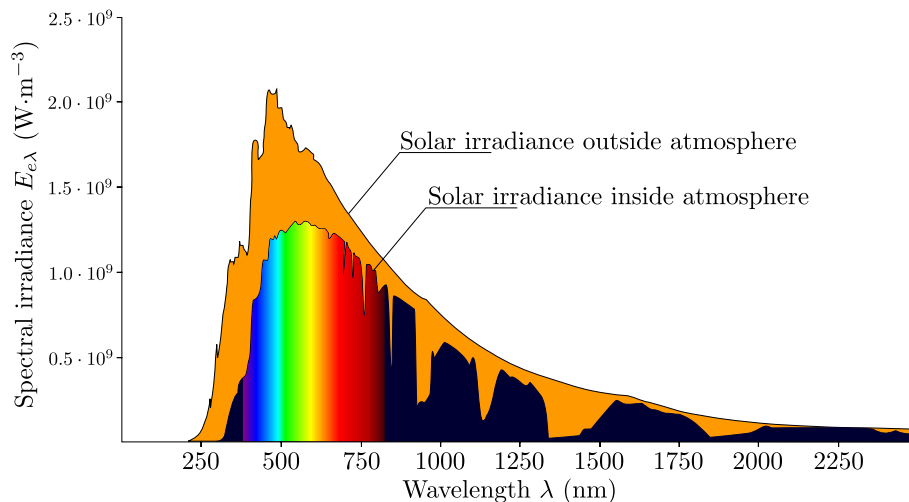


Figure 2.7: The spectral solar irradiance, as measured from outer space as well as the earth's surface, plotted against the wavelength. Note how the spectral irradiance is strongest around the visible spectrum.

©️🌐🌐 Degreen / Wikimedia Commons.

In photometry, this unit is called *luminous flux density*, better known for the names of its two distinct forms, *illuminance* ( $E_v$ ) and *luminous exitance* ( $M_v$ ). Both are measured in lux (lx), which is the amount of lumens per square meter.

Like irradiance, illuminance is an important measure in quantifying light levels. For instance, required lighting levels specified in working condition guides are described by the illuminance. The illuminance produced by the sun when directly overhead is approximately 130000 lx, while skylight emits an illuminance between 10000 lx and 25000 lx [Sch09].

### Radiant and luminous intensity

The *radiant intensity*  $I_e$  is defined as the amount of radiant flux that is emitted from a point source per unit solid angle in some direction  $\vec{s}$ , as defined in equation (2.34). The unit of radiant intensity is watt per steradian ( $\text{W}\cdot\text{sr}^{-1}$ ). Figure 2.8 illustrates this concept.

$$I_e = \frac{d\Phi_e}{d\Omega} \quad (2.34)$$

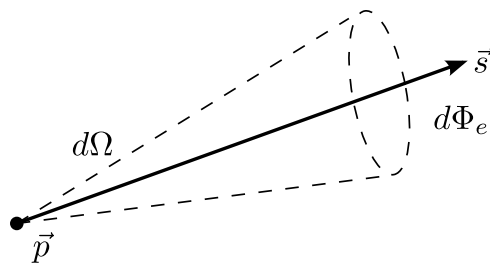


Figure 2.8: For a point source at  $\vec{p}$ , the radiant intensity  $I_e$  is the amount of radiant flux  $d\Phi_e$  per  $d\Omega$ .

The photometric equivalent is the *luminous intensity*  $I_v$ , which is measured in candelas (cd). One candela is defined to equal one lumen per steradian, and originates from roughly being the luminous intensity of an average candle light.

For example, consider a candle with a luminous intensity of 1 cd in all directions. This means that  $\Phi_v = \int I_v d\Omega = 4\pi$  lm, the solid angle of a full sphere. Likewise, if a light bulb emits 1 lm of luminous flux, uniformly distributed along a hemisphere,  $I_v = \frac{\Phi_v}{\Omega} = \frac{1}{2\pi}$  cd for any direction within this hemisphere.

### Radiance and luminance

*Radiance*,  $L_e$ , has the same relationship with radiant intensity as radiant flux density has with radiant flux. It denotes the radiant flux per unit solid angle in direction  $\vec{s}$ , per unit surface area. It is irrelevant whether the radiation is arriving at, coming through, or emitted by the surface.

One alteration has to be made with respect to the aforementioned analogy; that is, the inclination of the surface needs to be taken into account by considering the angle  $\theta$  between the normal  $\vec{n}$  of the surface area  $dA$ , and the viewing direction  $\vec{s}$ . This was neither required for the radiant flux density, as it considers radiance coming in from all directions, nor for radiant intensity, as it denotes a point source, for which tilting has no effect due to the lack of a surface. The concept of radiance is illustrated in Figure 2.9.

The radiance is defined in equation (2.35), and is measured in  $\text{W}\cdot\text{sr}^{-1}\cdot\text{m}^{-2}$ . It should be noted that the surface  $dA$  may be imaginary, e.g., a small portion of the sky.

$$L_e = \frac{d^2\Phi_e}{dAd\Omega \cos\theta} \quad (2.35)$$

Note the introduction of the cosine, which modulates the aforementioned inclination. It follows from the relationship between radiance and irradiance; to obtain irradiance  $E_e$ , we must integrate the differential irradiance  $dE_e$  over the solid angle, which is equivalent to integrating the radiance  $L_e$  over the solid angle. However, the differential irradiance becomes smaller as the angle  $\theta$  grows larger, due to the fact that the area upon which the radiant flux is incident, increases in size. The introduction of the

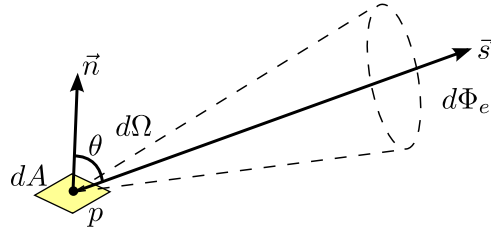


Figure 2.9: The radiance  $L_e$  is the amount of radiant flux  $d\Phi_e$  per area  $dA$ , per solid angle  $d\Omega$  in direction  $\vec{s}$ .

cosine term to keep the differential irradiance and radiance proportional is known as Lambert's cosine law, illustrated in equation (2.36).

$$E_e = \int_{\Omega} dE_e = \int_{\Omega} L_e \cos \theta d\Omega \quad (2.36)$$

An interesting property of radiance is its invariance with respect to the distance between the surface  $dA$  and the observer. As this distance gets larger, the cross-sectional area  $dA$  grows, but the solid angle  $d\Omega$  decreases proportionally. This phenomenon is illustrated in Figure 2.10. Given a radiating surface  $C$ , and a camera with a certain solid view angle  $d\Omega_V$ , the cross-sectional area that the camera captures grows larger proportionally to the squared distance between the camera and the surface. Since  $x_2$  is twice as big as  $x_1$ , and with  $d\Omega_V$  constant, the cross-section  $dA_{L2}$  becomes four times as large as  $dA_{L1}$ . Simultaneously, the solid angle  $d\Omega_L$  – which is dependent on  $x$  and the constant cross-sectional area  $dA_V$  – i.e., the size of the camera's entrance pupil – *decreases* proportionally to the squared distance. Thus,  $d\Omega_{L2}$  is four times as small as  $d\Omega_{L1}$ , resulting from equation (2.27), substituting  $r$  with  $x$ . This way, the changes in  $d\Omega_{L2}$  and  $dA_{L2}$  cancel each other out, and the radiance remains constant.

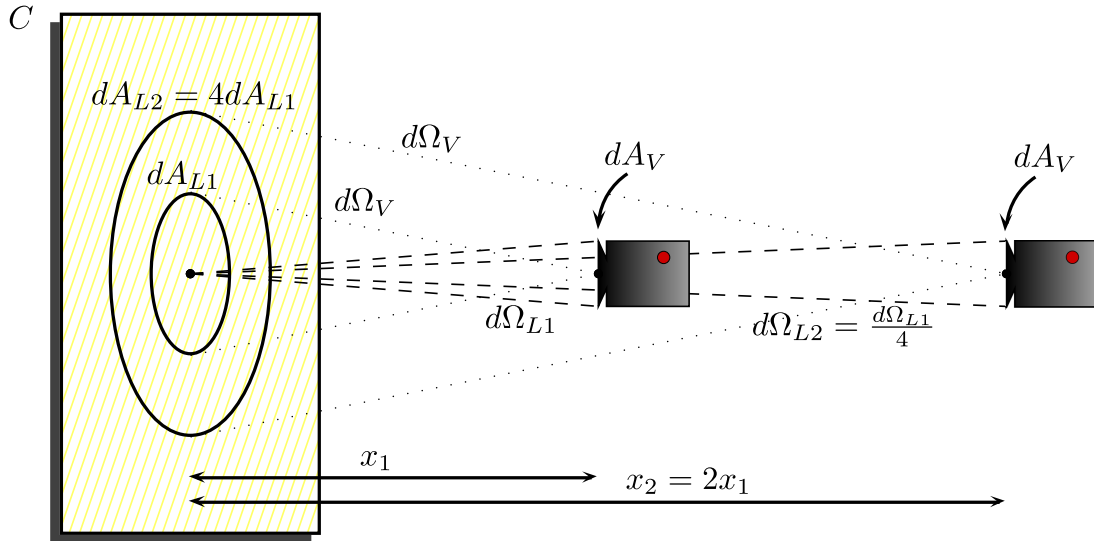


Figure 2.10: The invariance of radiance with respect to the distance between surface and observer.

Moreover, radiance is invariant with respect to the angle between the surface normal  $\vec{n}$  and view direction  $\vec{s}$ . Equation (2.35) signifies that as this angle  $\theta$  increases, the term  $\cos \theta$ , and in turn  $L_e$ , decrease. Note that  $\theta < \frac{\pi}{2}$ , or the radiance flowing through the surface could never reach the observer. On the other hand,  $dA$  *increases* with  $\theta$ , inversely proportional to  $\cos \theta$ , as illustrated in Figure 2.11. As with the distance  $x$ , any variation in  $\theta$  does not alter the resulting value of the radiance  $L_e$ .

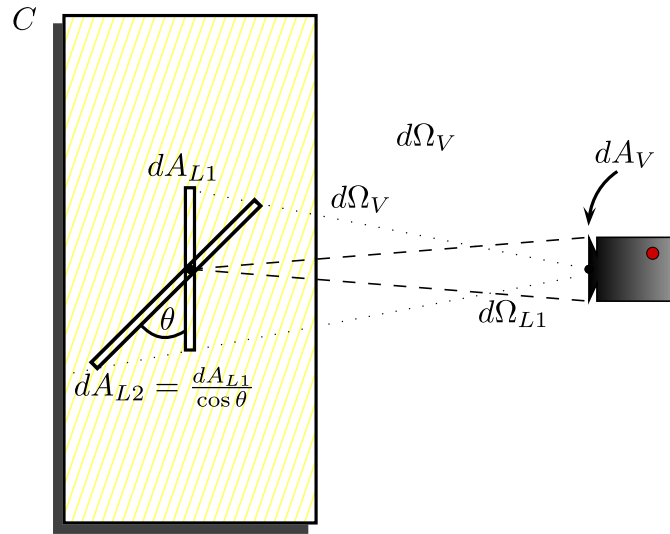


Figure 2.11: The invariance of radiance with respect to the  $\theta$ . The area  $dA_{L2}$  corresponds to an inclination of  $\theta = 45^\circ$ .

The photometric variant of radiance is called *luminance*, and is measured in candela per meter, its definition shown in equation (2.37).

$$L_v = \frac{d^2\Phi_v}{dAd\Omega \cos \theta} \quad (2.37)$$

Luminance is a very important unit, as it represents the brightness that is perceived by a camera or the human eye, as becomes apparent by the scenes in Figures 2.10 and 2.11.

### 2.1.5 BRDF

The *bidirectional reflection distribution function*, often called BRDF, is the ratio of radiance reflected by a surface in a certain view direction  $\vec{s}_V$ , to the irradiance incident on this surface, coming from a certain direction  $\vec{s}_L$ . As the vectors, which are defined relative to the surface normal, consist of two spherical coordinates each, the BRDF is a four-dimensional function, making it difficult to visualize. Usually, a single incoming radiation direction  $\vec{s}_L$  is considered, and the result can be visualized as a two-dimensional function, with variable view direction, as illustrated in Figure 2.12.

The formal definition of the BRDF  $R$ , given radiation direction  $(\theta_L, \phi_L)$  and view direction  $(\theta_V, \phi_V)$  is given below in equation (2.38) [Nic65], and it is measured in  $\text{sr}^{-1}$ . Note the insertion of Lambert's cosine law.

$$R(\theta_V, \phi_V, \theta_L, \phi_L) = \frac{dL(\theta_V, \phi_V)}{dE(\theta_L, \phi_L)} = \frac{dL(\theta_V, \phi_V)}{L(\theta_L, \phi_L) \cos \theta_L d\theta_L d\phi_L} \quad (2.38)$$

The BRDF may also be used to denote the ratio between the photometric quantities of luminance and illuminance, thus the subscript denoting a radiometric or photometric unit is omitted in equation (2.38).

Similar to the BRDF, the *BTDF* (*bidirectional transmittance distribution function*) denotes the ratio between *transmitted* radiance and irradiance and is often used as a measure to describe the opacity of an object.



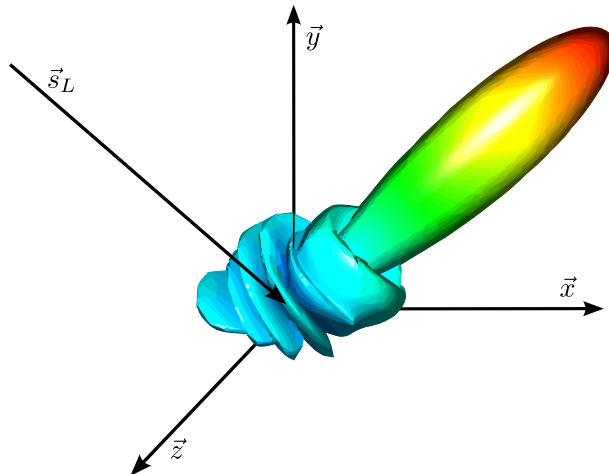


Figure 2.12: A complex, specular BRDF given a radiation direction  $\vec{s}_L$ . The result of the function is denoted by the distance from the origin and the color, plotted for all outgoing view directions. Image created with BRDFLab [FPBP09].

### 2.1.6 BSSRDF

Clouds however, are a participating medium, which means that radiation can also enter the object and *scatter* in different directions – often multiple times – before re-emerging. The amount of times a photon bounces off in a different direction is called its scattering order. Zero-order scattering simply modulates the transmittance of an object, and will be more thoroughly discussed in section 2.1.8. First-order scattering, often called *single scattering*, consists of the photons that are scattered only once. Higher orders of scattering are often grouped together under the name *multiple scattering*.

In any case, this subsurface scattering is where a *BSSRDF* comes in. The *bidirectional surface scattering reflectance distribution function* is a generalization of the BRDF. The BRDF assumes that all reflected radiance comes from irradiance that is incident on the same point on the surface. However, with participating media such as clouds, we need to take into account the aforementioned scattering. The importance of capturing subsurface scattering accurately becomes apparent from images such as Figure 2.13.



(a) BRDF

(b) BSSRDF

Figure 2.13: Comparison of rendering human skin – a participating medium – using a BRDF and BSSRDF. Images by Jensen et al. [JMLH01].

The BSSRDF denotes the ratio of radiance exiting from a surface at some point  $(x_V, z_V)$  in view

direction  $\vec{s}_V$ , to the incident irradiance on the surface at another point  $(x_L, z_L)$  coming from direction  $\vec{s}_L$ . From this we can see that the BRDF simply assumes that  $(x_V, z_V) = (x_L, z_L)$ . The idea of the BSSRDF is illustrated by Figure 2.14, and the formal definition is given in equation (2.39).

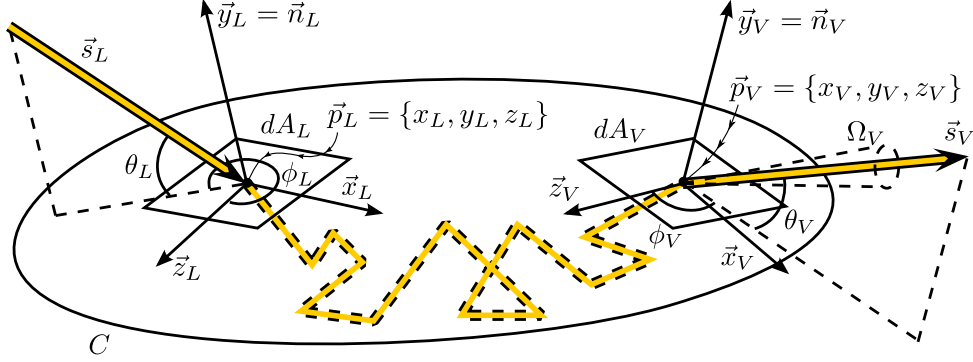


Figure 2.14: Illustration of the BSSRDF model, with irradiance coming in from direction  $\vec{s}_L$  scattering beneath the surface of  $C$  before re-emerging in direction  $\vec{s}_V$  as radiance.

$$S(\theta_V, \phi_V, x_V, z_V, \theta_L, \phi_L, x_L, z_L) = \frac{dL(\theta_V, \phi_V, x_V, z_V)}{dE(\theta_L, \phi_L, x_L, z_L)dA_L} = \frac{dL(\theta_V, \phi_V, x_V, z_V)}{L(\theta_L, \phi_L) \cos \theta_L d\theta_L d\phi_L dA_L} \quad (2.39)$$

This means we can now write the final outgoing radiance as a function of the BSSRDF and incoming radiance, as shown in equation (2.40).

$$L(\theta_V, \phi_V, x_V, z_V) = \int_C \int_0^{2\pi} \int_0^{\frac{\pi}{2}} S(\theta_V, \phi_V, x_V, z_V, \theta_L, \phi_L, x_L, z_L) L(\theta_L, \phi_L) \cos \theta_L d\theta_L d\phi_L dA_L \quad (2.40)$$

Substituting radiance with luminance, we could theoretically render a cloud, solving (2.40) for every pixel. However, the BSSRDF is an eight-dimensional function, which makes it an unfeasible option for online rendering, given the current computational resources.

### 2.1.7 Phase function

So far, we have defined a scattering event as a photon bouncing off into a different direction while inside a participating medium. However, we have left out in *which* direction the scattering occurs. This is defined by a random event, modulated by a probability density function known as the *phase function*,  $P(\vec{s}_L, \vec{s}_V)$ , with  $\vec{s}_L = (\theta_L, \phi_L)$  and  $\vec{s}_V = (\theta_V, \phi_V)$ . It is defined such that its integral over all scattering directions equals one, as shown in equation (2.41). As the phase function for water droplets is radially symmetric, we can reduce it to  $P(\Theta)$  with  $\cos \Theta = \vec{s}_L \cdot \vec{s}_V$  the *scattering angle*. If the probability of radiation being scattered is uniformly distributed over all directions, we consider the phase function to be *isotropic*. In real life, this is generally not the case, and we call the phase function *anisotropic*, i.e., view dependent.

$$\int_0^{2\pi} \int_0^\pi P(\vec{s}_L, \vec{s}_V) d\theta_V d\phi_V = 1 \quad (2.41)$$

The phase function is often very complicated, depending on the shape of the particle that causes the radiation to scatter, the wavelength of the photon, and several external parameters like the temperature. For the spherical water droplets that constitute the cumuliform clouds we are trying to render, the phase function is described by Mie scattering [Mie08]. For clouds, the most important parameters of the phase function are the wavelength of a photon and the radius of the water droplet. Figure 2.15 shows a logarithmic plot of the phase function that is used throughout this thesis against the scattering

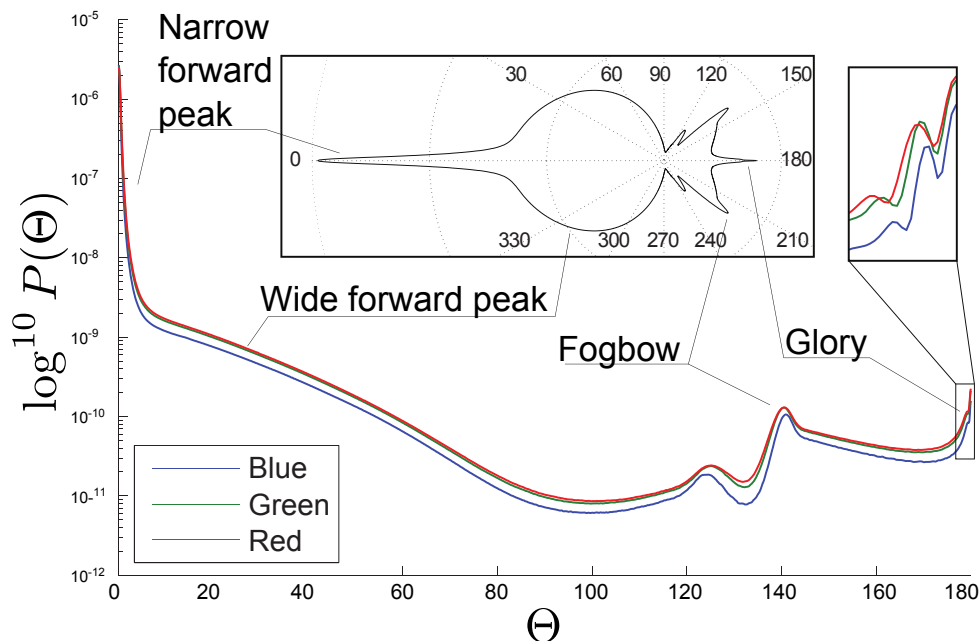


Figure 2.15: Mie phase function of the red, green and blue channels that is used throughout this thesis, based on a schematic by Bouthors [Bou08]. The inset displays a logarithmic polar plot for the phase function over the whole spectrum. A discretized version can be found on Bouthors’ thesis website at <http://evasion.imag.fr/~Antoine.Bouthors/research/phd/>.

angle. Instead of plotting against the wavelength too, the phase function is displayed separately for the standardized red, green and blue channels used for computer displays.

It is clear that the phase function for clouds is highly anisotropic, as an isotropic phase function would result in a flat line when plotted against the scattering angle. Also, it is interesting to note that the great majority of incoming radiation will be scattered in the forward direction, with a narrow peak around  $\Theta = 0$ . The fogbow and glory will be discussed in section 2.2.5.

### 2.1.8 Extinction

We have discussed what happens when photons interact with cloud droplets and are scattered. However, it is also of critical importance to know *when* such an interaction takes place, which is modulated by the probability of a droplet being in the path of the photon. This is called the *extinction function*, which classically depends on the absorption and scattering functions. However, pure water droplets absorb virtually no visible light [CRC84], and we approximate extinction to be solely dependent on the amount of scattering events. Logically, this is determined by the radius of the droplets and their density. However, both these parameters vary throughout a cloud, making it difficult to compute the extinction function for the whole cloud.

The radius of the water droplets is often described by a *droplet size distribution* (DSD). As mentioned in [PK97], it appears from real-life measurements that this distribution often takes a very characteristic shape, which can be approximated using a gamma or log-normal distribution. Using the DSD, we can obtain the *effective radius*  $r_e$  in meters through integrating over all radii. In real clouds, the effective radius usually varies between  $5 \mu\text{m}$  and  $15 \mu\text{m}$  [Bou08].

With the effective radius, we can define the *extinction cross section*  $\sigma_e$  in  $\text{m}^2$ , which is the effective area of the region where interaction between the photons and droplets occurs. With the spherical water droplets in a cloud, this boils down to double the geometric cross section, as shown in equation (2.42) [BC06] [BH08].

$$\sigma_e = 2\pi r_e^2 \quad (2.42)$$

With the effective cross section of water droplets in a cloud, we can continue to compute the *extinction coefficient*,  $k_e(\vec{p})$ , which is measured in  $\text{m}^{-1}$  and varies throughout the cloud. It is dependent on the droplet density  $\rho(\vec{p})$  in  $\text{m}^{-3}$  at the considered point  $\vec{p}$ , as shown in equation (2.43). Note that we do not use the classical definition of density in  $\text{kg}\cdot\text{m}^{-3}$ , as we are not interested in the mass of the droplets, but only in their number. For the remainder of this thesis we will refer to the density as number per unit volume, which is formally called the *number density*.

$$k_e(\vec{p}) = \rho(\vec{p})\sigma_e \quad (2.43)$$

With the extinction coefficient, we can obtain the dimensionless *optical thickness* or optical depth  $\tau(\vec{p}_a, \vec{p}_b)$  of a certain path  $(\vec{p}_a, \vec{p}_b)$ , the physical meaning of which is how much extinction takes place when radiation passes through the object along this path. It is obtained by integrating the extinction coefficient over the considered path, as shown in equation (2.44).

$$\tau(\vec{p}_a, \vec{p}_b) = \int_{\vec{p}_a}^{\vec{p}_b} k_e(\vec{p})d\vec{p} \quad (2.44)$$

Finally, with the optical thickness, we can obtain the extinction function  $\beta(\vec{p}_a, \vec{p}_b)$  as follows.

$$\beta(\vec{p}_a, \vec{p}_b) = e^{-\tau(\vec{p}_a, \vec{p}_b)} \quad (2.45)$$

The extinction function is equal to the *transparency* of the cloud, which means the *opacity* is given by its complementary, as in equation (2.46).

$$\alpha(\vec{p}_a, \vec{p}_b) = 1 - \beta(\vec{p}_a, \vec{p}_b) \quad (2.46)$$

All these quantities along with the phase function are called the basic radiative properties of the cloud, and form the basis of the radiative transfer that occurs within a cloud. Note that the subscript  $e$  denotes extinction, and differs from the subscript for radiometric quantities used throughout 2.1.4.

### 2.1.9 Radiative transfer equation

Now that we have the basic radiative properties, we are theoretically able to compute the luminance  $L(\vec{p}, \vec{s}_V)$  at a given point  $\vec{p}$  on the cloud surface, exiting in view direction  $\vec{s}_V$ . The physical formulation that enables us to do so, is called the *radiative transfer equation*.

Consider an infinitesimal cylindrical volume  $dV$  of height  $ds$  and base area  $dA$  located at  $\vec{p}$ . The luminance that arrives at this cylinder in direction  $\vec{s}_V$ , gains due to in-scattering and extinguishes due to out-scattering before leaving  $dV$  and arriving at our eye (or, in the virtual case, a pixel). The combination of in- and out-scattering is the differential luminance  $dL(\vec{p}, \vec{s}_V)$  of the cylinder  $dV$ . In-scattering is modulated by the optical thickness – i.e., the extinction coefficient  $k_e(\vec{p})$  multiplied by the cylinder length  $ds$  – and the so-called emission coefficient  $j(\vec{p}, \vec{s}_V)$ . For the case of clouds, the emission coefficient simply boils down to an integration of the phase function multiplied by the luminance coming from the light direction, over all light directions, with the result divided by  $4\pi$  for normalization, resulting in equation (2.47).

$$j(\vec{p}, \vec{s}_V) = \frac{1}{4\pi} \int_0^{2\pi} \int_0^\pi P(\Theta)L(\vec{p}, \vec{s}_L)d\theta_L d\phi_L \quad (2.47)$$

The out-scattering is much simpler, and reduces to the same optical thickness multiplied by the luminance arriving from the view direction,  $L(\vec{p}, \vec{s}_V)$ . Combining the two results in equation (2.48).

$$dL(\vec{p}, \vec{s}_V) = k_e(\vec{p})j(\vec{p}, \vec{s}_V)ds - k_e(\vec{p})L(\vec{p}, \vec{s}_V)ds \quad (2.48)$$

Which can be rewritten to equation (2.49), which is called the radiative transfer equation. Note that this is a simplified version where absorption is omitted.

$$\frac{dL(\vec{p}, \vec{s}_V)}{k_e(\vec{p})ds} = \frac{1}{4\pi} \int_0^{2\pi} \int_0^\pi P(\Theta)L(\vec{p}, \vec{s}_L)d\theta_L d\phi_L - L(\vec{p}, \vec{s}_V) \quad (2.49)$$

Solving this equation for the luminance would give us the necessary data to render a cloud. However, due to the fact that in this equation the luminance is dependent on its own derivative and integral, it makes for an unfeasible approach for real-time visualization. Nonetheless, it has been used for several applications that pertain to the subject of cloud rendering, as will be discussed later on in this chapter.

## 2.2 Cloud features

Note that from here on, we delve into the particular subject of clouds, thus considering photometric units only, leaving behind the world of radiometry. Therefore, instead of talking of radiation, we will discuss only visible light.


In order to visualize convincing clouds, we need to investigate and reproduce the visual features that persuade us when we are indeed looking at clouds in real life. As mentioned in chapter 1, the scope of this thesis is to render cumuliform clouds in real-time. Therefore, the features of this particular category of cloud will be considered.

### 2.2.1 Fractal shape

Cumuliform clouds are defined as having a shape consisting of heaps, creating the typical puffy look. When looking at them more closely, we can see that all cumuliform clouds consist of several main heaps. Investigating these, we can distinguish sub-heaps making up these main heaps, which in turn consist of heaps themselves. This fractal behavior is well illustrated in Figure 2.16.

As with many things in nature, the fractal property of clouds adds to their complexity, however, luckily, there are methods for efficiently simulating this behavior into tricking the viewer they are looking at subtle irregularities – for instance, fractal sums of noise.



Figure 2.16: The fractal property of cumuliform clouds is very apparent in this cumulonimbus.  Nicholas A. Tonelli.

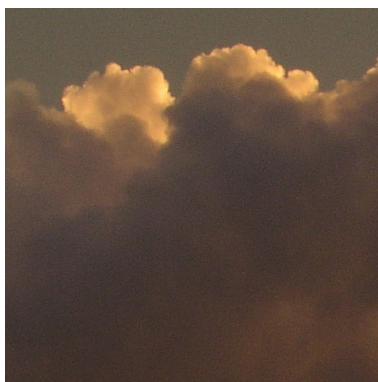


### 2.2.2 Cloud edges

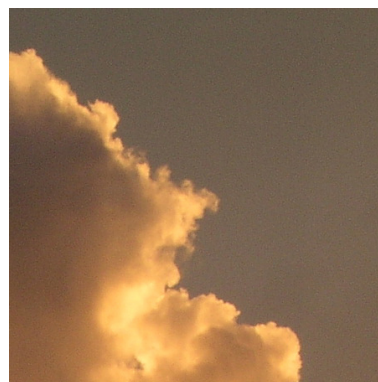
Another important aspect of cumuliform cloud shapes is the appearance of their edges. Their texture varies between well-defined and very wispy or smudged boundaries, as shown in Figure 2.17. This is due to the highly fluctuating droplet density throughout the cloud, with hard edges caused by sudden transitions of high density to zero density, and soft edges due to a more gradual transition. The different types of edges in cumuliform clouds have one thing in common, and that is their detailed appearance. Unlike some stratiform clouds, which can have a certain blurred look, cumuliform clouds have very advanced features which can prove to be difficult to visualize. It should be taken into account that the edges are three-dimensional, and their importance reaches beyond the two-dimensional silhouette of the cloud. Indeed, oftentimes a cumuliform heap is visible in front of the main body of the cloud, as in Figure 2.16, and the high detail of the edges of this heap is just as crucial as when we would view it set against a background consisting of the sky.



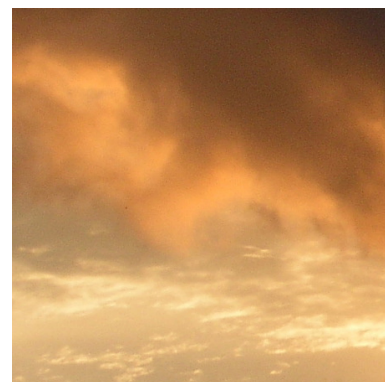
(a) Cumulus clouds during sunset. The top cloud's edges are scrutinized below.



(b) Puffy yet hard, well-defined cloud boundary.



(c) Somewhat wispy, yet still clear boundary.



(d) Very smudged and wispy edge, hard to define cloud boundary.

Figure 2.17: Cloud edges can vary greatly, even for the same cloud, as shown by the bottom images. © Mayang Murni Adnin.

### 2.2.3 Multiple scattering

Focusing on the color, the appearance of a cloud is predominantly determined by multiple scattering. As mentioned in section 2.1.8, the odds of a scattering event occurring are proportional to the length of the path through the object and its density, which can both be relatively high for clouds. On average, photons get scattered every ten to thirty meters they travel in cumuliform clouds, while the clouds themselves can be up to hundreds or thousands of meters in size [Bou08]. Correspondingly, except for the boundaries, the opacity of cumuliform clouds generally approaches one – it is virtually impossible to see through them.

This means that the macroscopic behavior of light in clouds is modulated by a great number of scattering events, which offsets the original Mie phase function that is used for a single bounce. Consider for example Figure 2.18, which shows a great cumulonimbus cloud. Despite the Mie phase function containing a very strong forward peak, the cloud top is actually much brighter than its base. This is due to the great amount of scattering events that can occur in such a massive cloud, making cumulative *backscattering* more dominant than forward scattering. In thin regions of the cloud, however, where single scattering is more common, the strong forward peak of the phase function will dominate the macroscopic behavior, resulting in bright bases and darker tops. We need to somehow mimic this behavior correctly in order to render convincing results.



Figure 2.18: Cumulonimbus with a blinding white peak compared to an ominously dark brown and gray base. © Evan Blaser.

### 2.2.4 Silver lining and other contrast

A common visual feature of clouds that is of utmost importance when pursuing realistic visualization, is the well-known silver lining, as seen in Figure 2.19. This is mainly caused by single scattering in the thin regions mentioned in section 2.2.3, which are naturally prevalent in the cloud boundaries. Due to this, the forward scattering along the edges of a cloud decreases when transitioning to the core, resulting in the bright outline when the sun is roughly behind a cloud.

Aside from the edges, the contrast between bright and dark patches in cumulus clouds can be significant in other areas too, as becomes apparent from images such as those shown in Figures 2.16, 2.18 and 2.19. The great variation in water droplet density is the main culprit, along with the self-shadowing



Figure 2.19: The cloud on the left displays a particularly bright silver lining. © Richard Carlson.

property of large clouds. Moreover, light can become *trapped* in concavities [Ney00], causing relatively bright creases.

### 2.2.5 Glory and fogbow

A particularly beautiful feature of any cloud is the glory. Due to the conditions under which it is visible, it is common knowledge amongst pilots, hot air balloonists, or keen-eyed airline passengers. The glory only shows itself when you are viewing a cloud looking at the antipodal point of the sun, i.e., the antisolar point. For this, the observer needs to be between the cloud and the sun, which can normally be achieved by air travel alone. The glory is mainly caused by single scattering in accordance with the Mie phase function, with a strong dependence on wavelength, as shown in the inset in Figure 2.15. The real-life result is beautifully captured in Figure 2.20.

Very similar to the glory is a phenomenon called a fogbow, which appears as a bright white arc around the antisolar point, at a scattering angle of around  $140^\circ$ , caused by backscattering according to the increased intensity of the Mie phase function at that angle, as shown in Figure 2.15. Note that sometimes, the name cloud bow is used to signify the event occurring in clouds, while fogbow is reserved for describing the manifestation in fog. The phenomenon is similar to a rainbow, which is caused by raindrops under a similar scattering angle, yet it lacks its cousin's vivid colors due to the fact that water *droplets* are significantly smaller than raindrops, which causes diffraction to smear out the colors. A photograph of a fogbow is shown in Figure 2.21.

Despite being relatively rare in observing clouds, these features are part of the cloud's visual properties, and should be taken into account for rendering. Although many related phenomena exist, like rainbows, sun dogs, halos, and crepuscular rays, these are not caused directly by cumuliform water droplet clouds, thus lie outside the scope of this thesis.

### 2.2.6 Cloud lighting

We can conclude from Figure 2.7 that overhead sunlight reaching the earth's surface is more or less white, or green-yellowish if anything. However, when over-saturating cloud photographs, we often obtain blue clouds, as exemplified in Figure 2.22. We conclude from this that skylight – sunlight reflected off particles





Figure 2.20: The glory as photographed from a hot air balloon. The vivid colors are clearly visible. © Michael J. Slezak.

in the atmosphere – also constitutes a significant light source for clouds. Remembering the illuminance values for sunlight and skylight from section 2.1.4, we can say that about one fifth of a cloud’s illuminance is due to skylight when the sun is overhead, a number which becomes even larger as the sun lowers [DK02].

Besides skylight, [Bou08] and others claim that light reflected by the ground also plays a big role in a cloud’s appearance. We certainly acknowledge that the albedo of the ground can have a significant influence on the brightness of the cloud, as illustrated by Figure 2.23, however, we are less certain of the color of the ground having any effect worth mentioning, given the fact that very high albedo terrains have a white color (snow and ice). It is true that for some photographs, over-saturation brings out certain hues that correspond to the coloring of the ground, yet for every image we found this to be the case, there were ten images where the ground color did not seem to have an effect at all. Due to this, going through the trouble of considering terrain coloring in rendering a cloud may not be worth it, especially with an eye on the required performance for real-time rendering. We do think including an albedo value to somehow modulate the cloud brightness would be feasible.

### 2.2.7 Cloud movement

Cloud movement is governed by the laws of fluid mechanics, of which the most common form is a simple translation across the sky, roughly parallel to the earth’s surface, due to wind. At times, however, a more complicated pattern emerges upon closer inspection, which is especially noticeable when playing back recorded clouds in a time-lapse fashion, which shows behavior quite similar to the motion of smoke, but at a slower rate. Furthermore, formation and dissolution of clouds or cloud parts adds to the cloud movement. A beautiful clip showing all three types of movement can be found at <http://www.temponaut.com/mediaplayer.swf?file=product/Sonstige/test.mov>. Due to the complexity of the underlying physics, correctly animating clouds is no easy feat, yet methods exist for real-time smoke animation already [FSJ01] [ZRL<sup>+</sup>08], which may be applicable to clouds as well.



Figure 2.21: A fogbow occurring in fog photographed from a plane taking off. ©(i) Mila Zinkova.



(a) Original photograph of a cumulus cloud.

(b) Over-saturated version bringing out the blue tinge.

Figure 2.22: Over-saturating cloud photographs often exposes the blue tints that are hard to distinguish with the naked eye. ©(i) Nanimo.

## 2.3 Related work

There are three main challenges that pertain to cloud visualization. First of all, the complex shape of a cumulus cloud needs to be represented. Second, a rendering method needs to be devised, making use of



Figure 2.23: The high albedo of a snowy landscape has an obvious effect on the brightness of the clouds, as can be seen by comparing the white clouds over land to the gray ones over water, which has a low albedo. ©️🌐📷 Liam Quinn.

some lighting model to obtain the correct colors of the cloud. Finally, the cloud movement is simulated, ideally according to the laws of fluid dynamics. These three challenges will be discussed with respect to the existing work in the field of cloud rendering, considering their performance as well as their accuracy by comparing the results to the cloud features described in section 2.2.

A recent, more general survey on cloud rendering techniques describes some of these methods in more detail [HH12], however, we approach the current body of work attempting to define the best method in realizing the aforementioned cloud features through real-time visualization.

### 2.3.1 Modeling cloud shapes

#### Grid approaches

In representing the shape of a cloud, there are several main approaches. One technique relies on a three-dimensional grid of cells or voxels, where each voxel typically contains the cloud droplet density at that location. The grid method is advantageous in that it allows for an intuitive solution to the animation problem as well. The three dimensions of the grid however cause for a poor scaling with regard to computational performance. Oftentimes, to reach at least interactive frame rates, the amount of grid cells has to be reduced to such a coarse description that the results lose their detail, which makes it difficult to succeed in visualizing the detailed edges that are described in section 2.2.2.

The grid approach was first applied in the form of cellular automata to model cloud shapes in 2000 by Dobashi et al. [DKY<sup>+</sup>00] – note that [NDN96] used a grid as early as 1996, but for the rendering step only, which will be discussed in section 2.3.2. As a cellular automaton is a discrete model, meaning each cell can only have a finite number of states, they simply denoted a zero or one for cloud presence in every voxel. This allowed for more efficient computation, which was required for real-time performance on the hardware available at that time. To obtain a more realistic droplet distribution, they applied a smoothing operation on the resulting voxel grid. Still, the results lack detail due to the relatively low resolution of the grid, which becomes apparent in Figure 2.24.

An extension to cellular automata is the coupled map lattice, which uses smooth, real-value variables, unlike the discrete states of a cellular automaton, while subdividing the simulation space into lattices.



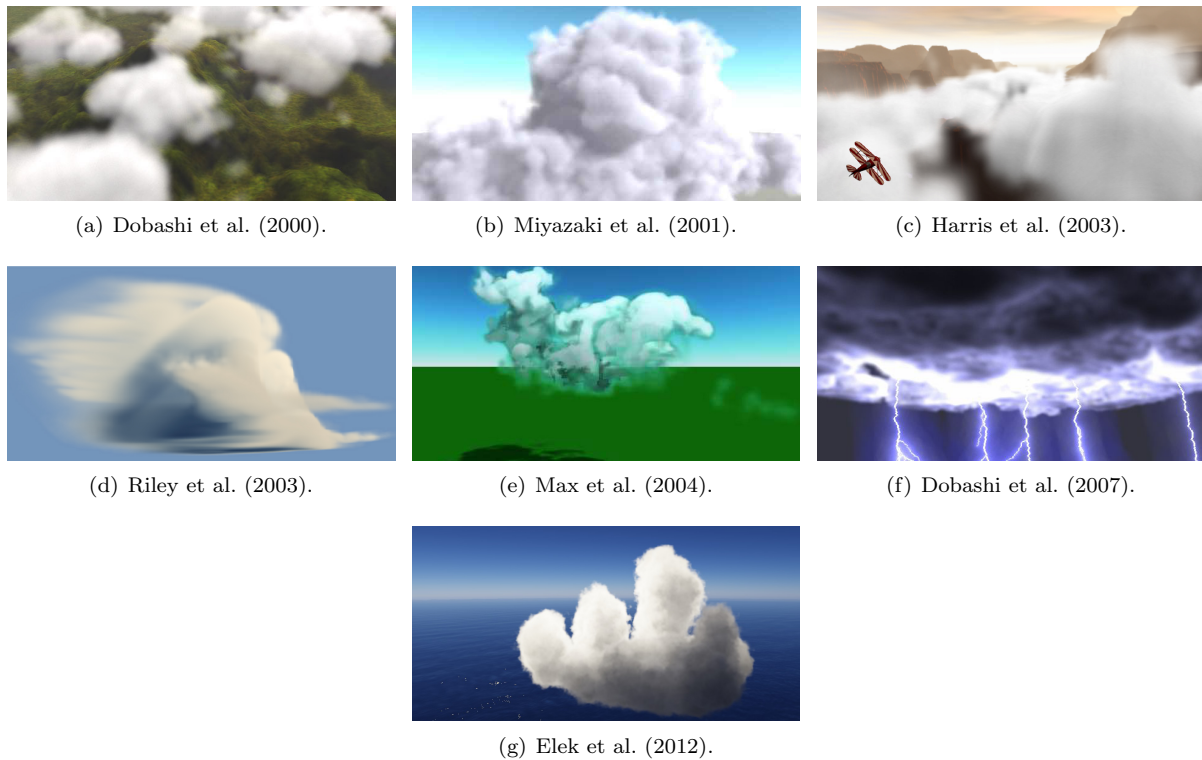


Figure 2.24: Overview of the results achieved by the methods that use a grid for the cloud's shape representation.

This approach was applied for clouds by Miyazaki et al. [MYDN01], and as such does not require the smoothing post-process, yet once again fails to capture the detailed edges for the same reason, as shown in Figure 2.24.

A more classical grid approach was proposed by Harris et al. in 2003 [HBSL03], who utilized the staggered grid to represent the cloud shape – i.e. a grid where the density is stored at the cell centers while the forces (for animation) are located at the cell faces, a method that has been used for smoke simulation in the past [FSJ01]. This approach caused them to use an even lower resolution than Dobashi et al., resulting in the blurry cloud edges in Figure 2.24. To overcome this problem, in the same year, Riley et al. discussed adding noise to a grid of meteorological data as a post-processing step [REHL03]. Max et al. on the other hand, used a multigrid solver in 2004 to characterize the density distribution [MSMI04]. Both these techniques attained rather disappointing results, as shown in Figure 2.24.

In 2007, Dobashi et al. built on their previous work in cloud rendering by devising a method for visualizing clouds solely illuminated by lightning, for which they subdivided the simulation space in a grid, creating virtual point light sources to simulate the lightning both from inside and outside the cloud [DEYN07]. Again, the results were too low-detailed.

A recent implementation however finally managed to overcome this issue by upsampling a low resolution grid every time step, counting on the temporal coherence of clouds to produce a high resolution image. For this, a preprocessing step is required to obtain the initial high resolution grid. Presented by Elek et al. in 2012 [ERWS12], this method achieves impressive results, yet suffers from performance issues, which will be discussed in section 2.3.2.

Figure 2.24 clearly shows the aforementioned lack of detailed edges due to low resolution grids, with the exception of the upsampled grid method presented by Elek et al. The other shape-related feature of cumuliform clouds, the fractal pattern, may also pose a problem in grid approaches for the same reason. Even when using meteorological data [REHL03], the lack of detail makes it difficult to discern

the characteristic fractal appearance.

An interesting development is the use of sparse voxel octrees to represent three-dimensional models, as shown in Figure 2.25. Gobbetti et al. proposed using such a structure on the GPU to take advantage of its parallel processing [GMG08], which was improved by Crassin et al. in 2009 [CNLE09]. Their technique was applied by Bouthors et al. [BNM<sup>+</sup>08] as part of their cloud shape representation, and may prove to be useful for future research in cloud visualization as well, as sufficiently high resolutions can be achieved without the extreme computational requirements of regular grids. However, with sparse voxel octrees, you lose the ability to easily animate the cloud through intuitive fluid simulation.

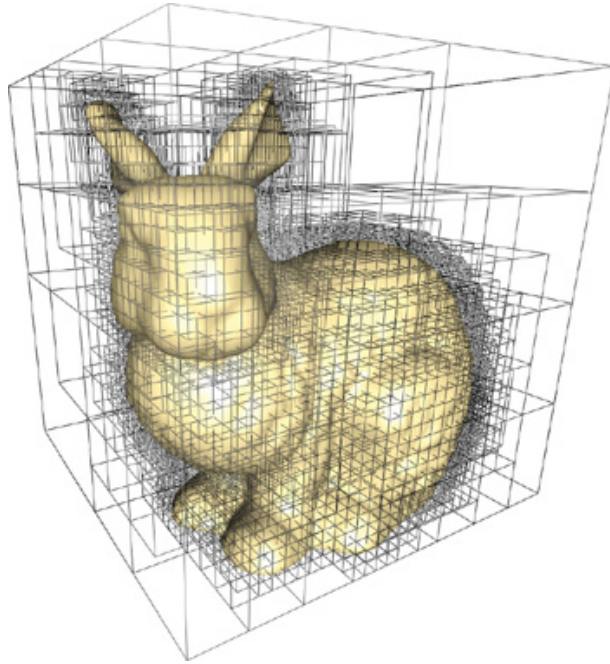


Figure 2.25: Stanford bunny represented by a sparse voxel octree. Image by Sylvain Lefebvre et al. [LHN05].

### Particle and ellipsoid approaches

An intuitive approach that comes to mind for the fuzzy cloud shape is the use of particle systems [Ree83]. One method consists in using one particle to represent one ellipsoid-like heap that is part of a cumuliform cloud. However, this causes the particles to be significantly larger than how they are classically applied. Indeed, many researchers prefer describing their model as a set of spheres or ellipsoids, yet the underlying theory often remains largely similar to particle systems with small particles, which is why we group these two approaches together.

This method scales much better computationally than a three-dimensional grid, yet realistically animating the particles may be difficult, especially when they are relatively large. Generally, noise is added to the particles to add detail, which works well in simulating the cloud edges. However, this is no distinct advantage over grid-based approaches, as Elek et al. have shown. The fractal shape on the other hand is more easily implemented using particles, by iteratively adding them in decreasing size but increasing number along the edges of the previous particle.

The pioneering work in cloud rendering was presented by Gardner in 1985, who made use of ellipsoids to represent the cloud shape [Gar85]. Gardner rightfully stated that clouds rarely have the general shape of an ellipsoid, and even when augmented with noise this holds, as is obvious from Figures 2.16 through 2.18, for instance. Therefore, Gardner proposed the use of clusters of ellipsoids, enhanced with noise to realistically simulate the cloud silhouette, for which the results are visible in Figure 2.26.

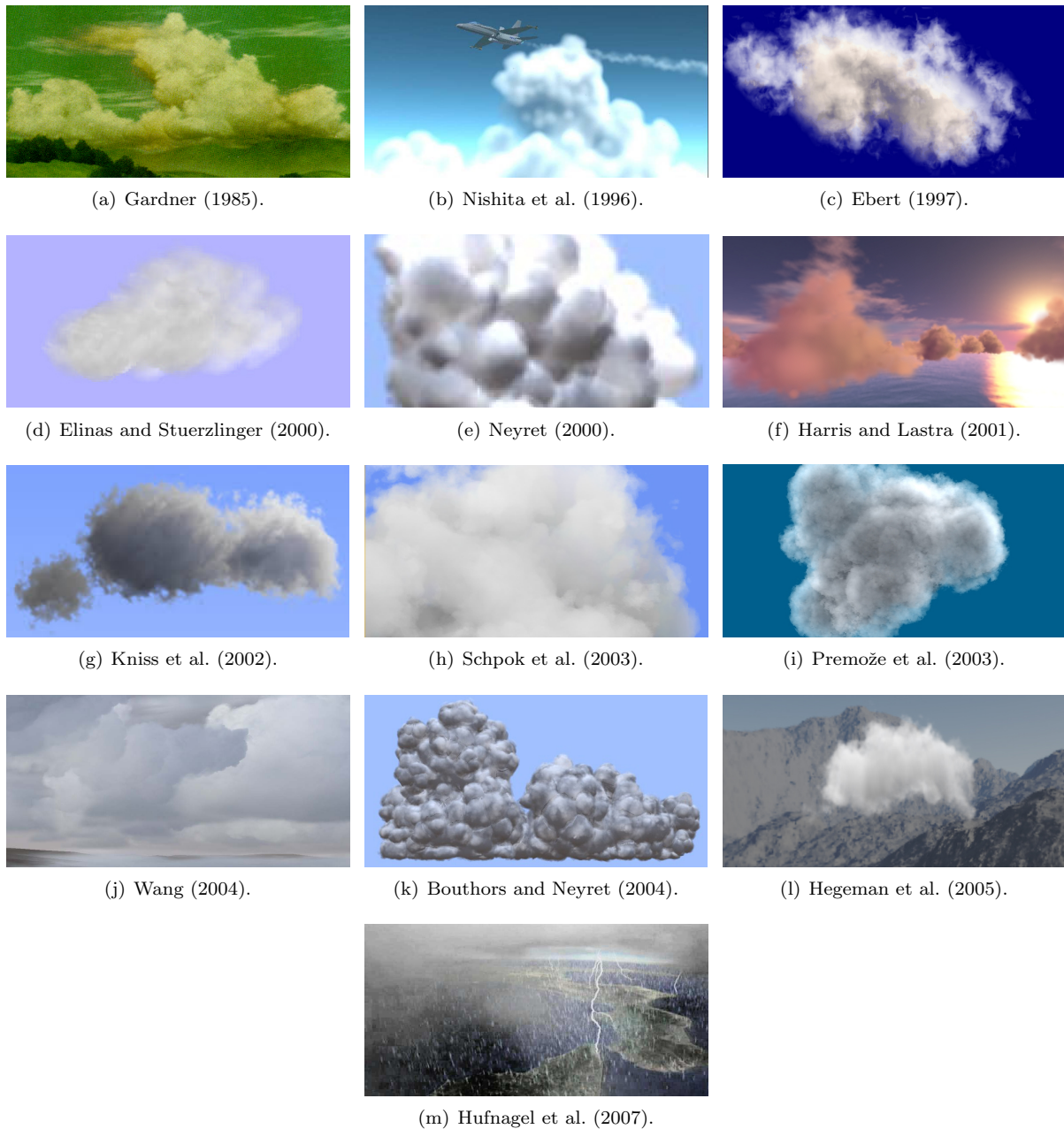


Figure 2.26: Overview of the results achieved by the methods that use a particle or ellipsoid approach.

Ellipsoids are capable of approximating triangle meshes quite well, even reducing the amount of data [LJWH07]. For fuzzy objects like clouds, we need noise to generate realistic results, yet the general shape can be taken care of by ellipsoids alone. The general shape is what allows us to recognize a cloud as, e.g., a certain creature, as in Figure 2.27.

A different approach was taken by Nishita et al. in 1996, who advocated the use of metaballs [NDN96], an implicit model dependent on the distance between neighbors so that nearby metaballs connect to form one object, as illustrated in Figure 2.28. First devised by Blinn in 1982 [Bli82], Nishita et al. built on their previous work on metaballs [NN94] to use them to represent the density field of their clouds. They used an adaptation of the fractal method to simulate the cumuliform heaps, adding metaballs of



Figure 2.27: With a little fantasy, we can often discern a creature in the general shape of a cloud.

decreasing size at the top of the clouds, while restricting attachment at the cloud base, based on their claim that cumuliform clouds usually have a flat base. However, due to the lack of noise addition and their grid approach in the rendering step, the results suffer from the same low level of detail as the methods discussed in the previous section. Nonetheless, their metaballs approach is quite useful still, especially with respect to new techniques for efficiently rendering a large number of them [GPP<sup>+</sup>10].

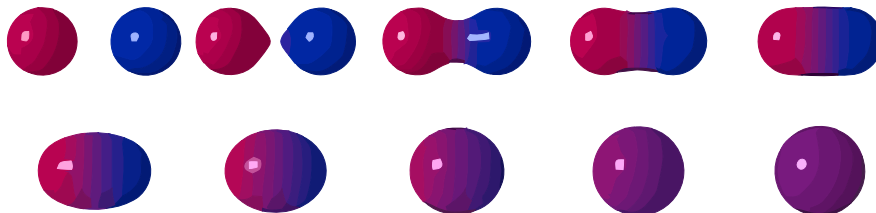


Figure 2.28: Influence of metaballs on their neighbors. ©️🇩🇪 SharkD / Wikimedia Commons.

In 1997, Ebert too proposed using implicit functions, this time combined with volumetric procedural modeling to easily obtain realistic cloud shapes [Ebe97]. The added level of detail compared to Nishita et al., due to the inclusion of noise, is clearly visible in Figure 2.26.

An extension to Gardner’s method using fractal Perlin noise [PH89] instead of sine waves was presented by Elinas et al. in 2000, achieving quite similar results to Gardner, but in real-time [ES00]. In the same year, Neyret presented a phenomenological cloud shading model using ellipsoids as well [Ney00], yet his method is more focused on the rendering of the clouds and will be discussed in section 2.3.2.

A more classical interpretation of particle systems with respect to cloud modeling was given by Harris and Lastra in their 2001 paper [HL01], using hundreds of small particles to simulate a cloud shape, resulting in hundreds of thousands in the complete scene. Figure 2.26 shows that shape-wise, their results are quite convincing.

In 2002, Kniss et al. on the other hand again used a set of ellipsoids to test their volume rendering



method in the context of cloud visualization [KPHE02] [KPH<sup>+</sup>03], augmenting them with a noise texture. A similar approach was taken by Schpok et al. in 2003 [SSEH03] to characterize cloud shapes, except they applied a noise texture to an artist-driven cloud modeling system using Nishita’s metaballs. In the same year, a similar model was used by Premože et al. to demonstrate their offline path integration rendering approach [PAS03], which will be discussed in section 2.3.2.

Conversely, in 2004, Wang presented an approach based on clusters of boxes to easily model the clouds throughout the atmosphere before applying a particle rendering technique [Wan04]. Her method was implemented in the release of that year’s Microsoft Flight Simulator, which was possible thanks to high frame rates.

In the same year, Bouthors and Neyret proposed a method for easily constructing cumuliform clouds using an adaptation of metaballs [BN04], iteratively placing particles of decreasing size, creating a clearly discernible hierarchy as shown in Figure 2.26.

The next year, Hegeman et al. created a real-time implementation of [PAS03], as presented in [HAP05]. Finally, in 2007, Hufnagel et al. built further upon the metaballs approach with special attention to large-scale cloud visualizations [HHS07].

Due to the inclusion of noise enhancement for many of these approaches, most of the results in Figure 2.26 produce the required level of detail at the edges, with the exception of Nishita et al., who did not use noise, Neyret, who focused mainly on the rendering process and used too little noise to realistically simulate a cloud’s wispy edges, and finally, Bouthors and Neyret, who focused solely on a method for outlining the base shape of a cloud, ignoring detailed features.

The fractal shape is explicitly simulated by Nishita et al. and Bouthors and Neyret by iteratively placing smaller metaballs on the cloud interface. However, many of the ellipsoid-based approaches also attain this feature due to the authors, possibly even subconsciously, placing ellipsoids in a similar fashion, with the inclusion of fractal noise taking care of the rest.

### Other approaches

In 2002, Trembilski and Broßler devised a method to visualize clouds for animation purposes [TB02]. They computed isosurfaces from meteorological data, subsequently partitioning it into a fine triangulation. They smoothed the mesh and eventually deformed it to give it a cumuliform look. The results are visible in Figure 2.29, and it is quite obvious that the depicted cloud is a triangle mesh, as it lacks for wispy and fuzzy regions.



(a) Trembilski and Broßler (2002).

(b) Bouthors et al. (2006).

(c) Bouthors et al. (2008).

Figure 2.29: Overview of the results achieved by the other methods.

Despite our focus on cumuliform clouds, for the integrity of this chapter, we will discuss the method presented by Bouthors et al. in 2006 [BNL06], which is specifically tailored to render stratiform clouds. Much of the ideas they proposed in this paper have found their way to their work on cumuliform clouds [BNM<sup>+</sup>08], however, this mainly pertains to rendering methods. For the shape, they simply used a height field, taking advantage of the typical sheet-like aspect of most stratiform clouds.

Finally, serving as the main reference is the method proposed by Bouthors et al. in 2008 [BNM<sup>+</sup>08], where they allowed for an arbitrary triangle mesh to represent the cloud shape, defining the inside of



this mesh to have a constant droplet density. To get realistic results, they introduced a heterogeneous boundary on this mesh, with the density decreasing when getting close to the mesh surface, and increasing when approaching the homogeneous core. To realize this idea, they stored the distance to the cloud boundary in the sparse voxel octree described by [CNLE09], essentially creating two shape representations. They further enabled wispy edges and a general fuzzy appearance by augmenting the distance value in the octree with fractal Perlin noise [PH89] during rendering.

Figure 2.29 shows little of the wispy features that are prevalent in most cumuliform clouds. Trembilski and Broßler attempted to generate a realistic triangulation through vertex deformation, which works quite well, yet can not fully capture the required level of detail. They managed to generate quite realistic edges through their transparency texturing, however, they are unable to extend this when an edge is located in front of the cloud silhouette – a very common occurrence given the heaps that constitute a cumulus cloud – which causes the triangulation to become very obvious. The stratiform clouds technique by Bouthors et al. fails to generate a high level of detail due to the use of a height field texture where each texel is mapped to multiple pixels in the result, even when viewing the cloud layer from a distance. Their 2008 take on cumuliform clouds does manage to generate sufficiently detailed noise, however, due to what seems to be a relatively small boundary region, the edges are very well-defined and the noise is not as prominent as it perhaps should be.

### 2.3.2 Rendering the colors of a cloud

In overcoming the rendering problems of clouds, a great variety of solutions have been proposed. Originally, Gardner introduced the use of a mathematical texture function with phenomenologically established parameters to calculate the transparency of their ellipsoids [Gar85]. A phenomenological model such as this does not necessarily follow the established laws of physics, but rather aims at a certain outcome – a visually convincing cloud – and tries to attain this by varying the parameters of the model. This process is generally repeated for a host of input variables to obtain a model that can achieve the best conceivable result for as many cases as possible. Such approaches can often be used to greatly enhance the computational performance of a method, however, it is often difficult or even impossible to obtain a sufficiently accurate model, depending on the amount of parameters and input variables. Moreover, the criteria of a good approximation are often not clear; for clouds, we can compare the results to real-life images and consider the visual features that we have discussed in section 2.2. Despite these disadvantages, impressive results have been achieved using phenomenological models in more recent years as well [DLR<sup>+</sup>09].

As for Gardner, additionally, he demonstrated the use of an early form of fractal noise, as displayed in Figure 2.30, by summing randomly offset three-dimensional sine waves of increasing frequency and decreasing amplitude to give the clouds a more realistic appearance; a technique that currently is often combined with Perlin noise [PH89] instead of sine waves, which, unfortunately, was not yet available in 1985. Gardner’s results were obtained during an offline rendering stage. However, this was mainly due to the significantly lower computational power available at the time.

Nishita et al. took a more physically based path [NDN96]. They subdivided the simulation space in a grid of parallelepiped (parallelogram-based) voxels in order to solve the radiative transfer equation (2.49). They achieved this by only taking into account up to the third order of scattering, justifying this decision with the notion that for water droplets, light is principally scattered in the forward direction, thus quickly leaving the cloud volume. However, as we have shown in section 2.2, several visual features depend on a large number of scattering events, which can even cause backscattering to become dominant in large cloud formations.

To solve the radiative transfer equation, they made use of a technique called Monte Carlo integration, a stochastic method for approximating an integral through a large number of experiments, as illustrated in equation (2.50).

$$\int_a^b f(x)dx \approx \frac{1}{N} \sum_{i=1}^N f(x_i) \quad (2.50)$$

Given function  $f(x)$ , its integral can be approximated by considering  $N$  randomly distributed points in the integration interval, i.e.  $x_1, x_2, \dots, x_N \in (a, b)$ . The law of large numbers dictates that as  $N$  grows

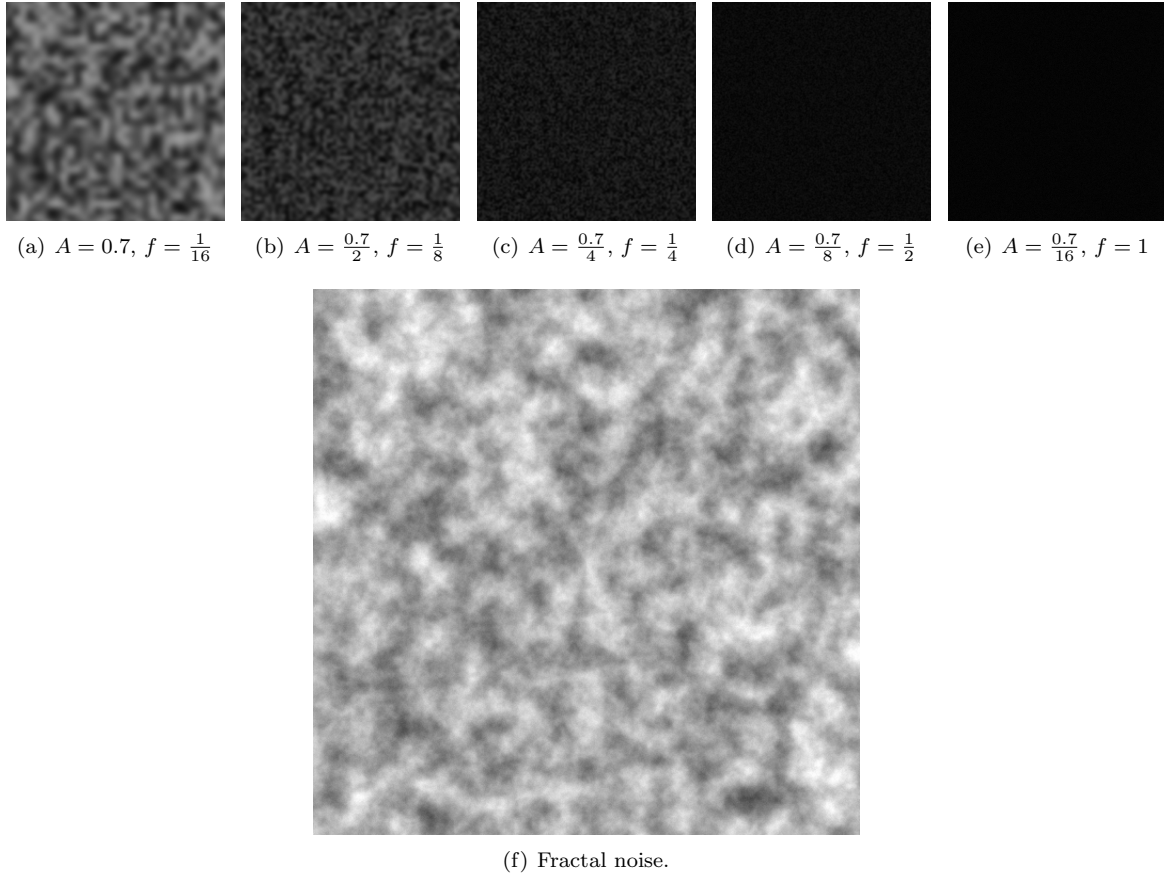


Figure 2.30: The first five images show two-dimensional noise of decreasing amplitude ( $A$ ) and increasing frequency ( $f$ ). Each of these images is a layer or octave. By adding them together, the result on the bottom is obtained, which is called two-dimensional fractal noise consisting of five octaves, and simulates a cloud texture quite nicely.

larger, the approximation becomes better, with the error margin being proportional to  $\frac{1}{\sqrt{N}}$ . For the phase function, Nishita et al. made use of the Henyey-Greenstein function [HG41], a procedural phase function shown in equation (2.52) that takes the anisotropy parameter  $g$  as input, where  $g = 1$  corresponds to full forward scattering,  $g = 0$  denotes an isotropic phase function, and  $g = -1$  defines full backward scattering. Note that it can be obtained from the phase function through integrating over the scattering angle as in equation (2.51).

$$g = 2\pi \int_{-\pi}^{\pi} P(\Theta) \cos \Theta \sin \Theta d\Theta \quad (2.51)$$

As per Figure 2.15, for clouds, the anisotropy parameter is quite high, and turns out to be  $g \approx 0.9$  [Bou08]. Nishita et al. used an anisotropy parameter that is a function [NSTN93] of atmospheric conditions as described in [CS92].

$$P(\Theta) = \frac{1 - g^2}{(1 + g^2 - 2g \cos \Theta)^{\frac{3}{2}}} \quad (2.52)$$

The disadvantage of the Henyey-Greenstein function is that it fails to capture the glory and fogbow, as it is but a rough approximation of the actual Mie phase function. This becomes apparent from Figure 2.31 as well. Furthermore, the results of Nishita et al. lack detail due to the absence of noise. Moreover,

their method could not achieve real-time performance, and despite the fact that recent Monte Carlo acceleration techniques [YIC<sup>+</sup>10] [SKTM11] may be capable of ensuring online rendering, the exclusion of high orders of scattering causes incorrect macroscopic light behavior, and considering a significant number of scattering orders is still computationally unfeasible.

Dobashi et al. [DKY<sup>+</sup>00] modified the cells of their cellular automata approach to simulate their density distribution using metaballs, subsequently rendering them through the use of splatting [Wes90] [Wes91]. The idea of this technique is to *splat* a volume element onto a viewing surface like a snowball, varying the properties – density in the case of clouds – of the element based on a Gaussian distribution. Dobashi et al. however used a density distribution that works especially well for metaballs [WT90] to splat their grid cells onto the viewing screen in a back-to-front order, correctly blending them for transparent areas.

To simulate light behavior in real-time, they only took into account single scattering using a simple isotropic phase function, first rendering the scene to a texture from the light source (the sun), and subsequently using this *shadow map* to analytically solve the single scattering. This is an adaption of a well-known method for enabling shadow casting in three-dimensional scenes [Wil78]. Finally, they added an ambient term for the multiple scattering component and the inclusion of skylight to ensure a correct level of overall brightness, yet it is no solution for the persisting low contrast and lack of backscattering.

Like Gardner, Neyret used a phenomenological model to implement a shader model for rendering cumuliform clouds [Ney00]. Neyret took into account three components for his phenomenological shader: an ambient term based on the environment, i.e., the ground and sky colors; single scattering with respect to the *corona* of the cloud, which is the imaginary heterogeneous boundary of the cloud as discussed in section 2.3.1; and inter-reflections that occur in the concave areas of cumuliform clouds, which Neyret called *light traps*. He applied his model separately for every ellipsoid in the cluster representing the cloud, blending the results to obtain a seamless final visualization.

Harris and Lastra strongly based their approach [HL01] on the method presented by Dobashi et al. Despite their use of small particles instead of a grid, they too visualized them through the splatting technique. Also, they used a similar way to solve single scattering analytically. However, instead of using Nishita’s Henyey-Greenstein function or Dobashi’s isotropic approach, Harris and Lastra opted for the anisotropic Rayleigh phase function, which denotes the scattering probability distribution for very small particles [Str71] and whose high level of dependence on the wavelength of light causes the sky – which consist of such small particles – to be blue. Harris and Lastra however only took into account the scattering angle in computing the phase function, according to equation (2.53).

$$P(\Theta) = \frac{3}{4} (1 + \cos^2 \Theta) \quad (2.53)$$

Despite knowing that cloud droplets are not at all small enough to be approximated well by the Rayleigh phase function, they still used it. A comparison of this phase function against Henyey-Greenstein and an isotropic function is given in Figure 2.31.

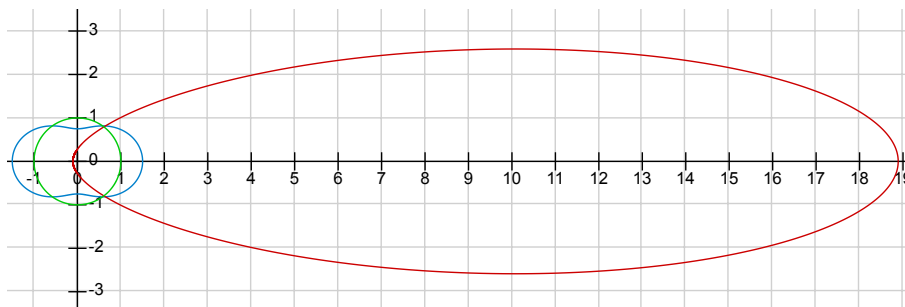


Figure 2.31: Plot of the Henyey-Greenstein function with anisotropy parameter  $g = 0.7$  (red), Rayleigh phase function (blue) and isotropic function (green). Note the significant difference between the Rayleigh and more correct Henyey-Greenstein function, even with a conservative anisotropy parameter.

They further improved upon Dobashi’s method by including *multiple forward scattering*, a technique for approximating multiple scattering by taking advantage of the fact that droplets mainly scatter light in the forward direction while assuming that backscattered light is negligible. The idea is to introduce an additional pass before solving the single scattering in the view direction. In this pass, the forward scattering is obtained in the light direction by accumulating all incident light. The downside of this approach is that it fails to capture the backscattering and high contrast that occurs in clouds.

To improve their performance, Harris and Lastra proposed the use of impostors [Sch95], which replace particles with a polygon with a texture to which these particles were rendered in a previous pass. The polygon and its texture are only updated when necessary, i.e., when a certain error margin is surpassed. In slow-changing systems such as clouds, this technique can result in a considerable speed-up, as Harris and Lastra proved in their paper.

Miyazaki et al. [MYDN01] used the same approach as Dobashi et al. for the rendering of clouds, focusing solely on the modeling aspect. Trembilski and Broßler [TB02] presented a phenomenological lighting model based on four components: ambient light; sunlight reflected like ambient light; diffusely reflected sunlight; and strongly forward scattered sunlight, using a home-brew phase function that takes a parameter similar to Henyey-Greenstein’s anisotropy parameter into account.

Kniss et al. showed that their volume rendering model can be applied to clouds as well [KPHE02], using an empirical model to approximate multiple scattering. They managed to maintain the translucency, color bleeding and diffusion of light that are due to multiple scattering, yet failed to capture backscattering and contrast adequately, suffering from the same problem as Harris and Lastra.

Premože et al. [PAS03] presented a Monte Carlo path tracing approach, i.e., integrating over all the illuminance arriving at a pixel, that they applied to clouds as well. They increased performance by using the *most probable paths* method, a mathematical formulation of multiple forward scattering, which they further extended in [PARN04]. Despite impressive visualizations of participating media, their methods take several minutes even on current hardware.

Schpok et al. [SSEH03] proposed a phenomenological, user-friendly rendering model, where an artist can define parameters such as shadow magnitude, shadow color and cloud fuzziness to create real-time visualizations that *look good*, yet lack physical correctness, which causes the absence of many visual features such as backscattering, silver lining and high contrast. They did not take into account the sun angle and let the whole lighting computation depend on the parameter input of the artist. Nonetheless, they achieved some impressive results, which seem quite convincing at first glance, especially due to their animation model, which will be discussed in section 2.3.3.

Harris et al. [HBSL03] re-applied their multiple forward scattering lighting model [HL01], this time implementing it to work with their three-dimensional textures instead of a particle system.

Riley et al. [REHL03] were the first to apply a more physically-based phase function to cloud rendering, based on values calculated by [WWW79], even taking into account a separate phase function for clouds containing ice crystals. They finally managed to reproduce the glory and fogbow, while simulating multiple scattering using a similar approach as Kniss et al. [KPH<sup>+</sup>03]. However, Riley et al. did not manage to overcome Kniss’ method’s lack of cloud features, and even produced rather smudged, and definitely less convincing images.

Wang, like Schpok et al., proposed the use of an artist-driven lighting model, letting the user associate a cloud pixel’s color to its height in the cloud [Wan04]. She let an artist define a set of boxes to simulate the general cloud shape, subsequently making use of the splatting technique to render particles based on this set of boxes. Like Harris and Lastra [HL01], she advocated the use of impostors to increase performance. Her method too suffers from the same issues as Schpok’s model.

Max et al. [MSMI04] employed the diffusion process to approximate multiple scattering, based on the notion that for a significant number of scattering events, the phase function becomes isotropic, which is valid for very dense media. To overcome the problem that arises in sparse areas, such as the cloud boundaries, where this assumption does not hold, Max et al. corrected their diffusion approximation where necessary. Additionally, they solved single scattering analytically to produce the final renderings, resulting in rather unrealistic images, which may be due to the input data that they used to represent the cloud shape.

Bouthors et al. [BNL06] rendered their stratiform clouds by analytically solving zero and first order

scattering using a discretized Mie phase function generated on software specifically engineered to plot solutions of the Mie function. Second order scattering was solved by using an approximation presented by Riley et al. in 2004 [REK<sup>+</sup>04]. For the remaining scattering orders, they used a simple phenomenological model that was obtained by fitting it to the results of a Monte Carlo path tracing preprocessing step. They also accounted for inter-reflections between the stratus layers and the ground through a radiosity model.

Dobashi et al. [DEYN07] rendered only the effect of lightning on clouds, which lies outside the scope of this thesis. Despite being an interesting approach, it can not be easily adapted to apply to daytime cloud rendering.

The lighting model presented by Bouthors et al. [BNM<sup>+</sup>08] consists of three components. Opacity and single scattering are solved analytically by integrating along the view direction using the same Mie phase function as [BNL06], while multiple scattering is computed using an advanced phenomenological model. Their methods will be more thoroughly discussed in chapter 3.

Finally, Elek et al. [ERWS12] recently proposed to make use of photon mapping, which consists of a first pass tracing photons from the light source, using a Monte Carlo technique to determine whether they are absorbed, scattered or transmitted when intersecting a surface, subsequently storing the result in a cache called the photon map; and a second pass ray tracing from the camera, calculating the radiance in the view direction for surface intersections by reading back the photon map. They implemented this using the Henyey-Greenstein function. While they attained remarkable results, their method is based on temporal coherency. When the view or sun angle changes drastically – admitted, this normally does not happen for the latter – the precomputation step needs to be repeated due to their grid upsampling technique. This means that real-time rendering can not be achieved unless the view angle is only allowed to move slightly every frame.

### 2.3.3 Animating cloud movement

Dobashi et al. were the first to implement real-time cloud animation [DKY<sup>+</sup>00]. Extending an extreme simplification of fluid dynamics [NR92], they managed to simulate their cellular automata to approximate cloud formation and dissipation and wind effects.

Miyazaki et al. [MYDN01] took a physically based approach, simulating cloud movement on their coupled map lattice according to the laws of atmospheric fluid dynamics. Harris et al. [HBSL03] too animated their clouds as governed by fluid dynamics, which was possible in real-time thanks to a very coarse grid description.

Trembilski and Broßler [TB02] allowed for animated clouds by processing meteorological data; as this data changes, the resulting triangulations change as well. Based on the fact that cloud movement generally is a slow process, showing the resulting renderings in order will produce smoothly animated images.

Schpok et al. [SSEH03] proposed to make use of the noise augmentation to realistically animate clouds at a very low cost. They combine simple translation of the cloud model, increasing or decreasing ellipsoid radii for formation and dissipation, and translation and rotation of texture coordinates that are used to sample a three-dimensional noise texture for the more subtle motion patterns discussed in section 2.2.7. They allowed for key-framing all these parameters, to enable artists to easily save and export animations. The artist-driven approach presented by Wang, only contains a simple feature enabling cloud formation and dissipation by modulating the opacity of the splatted particles [Wan04].

Bouthors et al. [BNL06] made use of a height field texture to represent the cloud shape. By animating this texture, they managed to realize a convincing animation of stratiform clouds passing by overhead. Their paper focusing on cumuliform clouds [BNM<sup>+</sup>08] did not include animation, yet they did mention that it would be possible by animating the triangle mesh that is used for the shape representation, updating the octree as described in [CNLE09]. Neither however is an easy feat, and the already precarious frame rates would suffer from this as well.

Elek et al. [ERWS12] allowed for animation as shown by their use of an animated smoke dataset, yet they did not present a method for animating existing data, despite using a grid approach, which is well fit for fluid dynamics.

### 2.3.4 Conclusion

We can conclude that despite all the research that has been done on the subject of cloud rendering, there is no method yet that is able to completely fulfill our goal as stated in chapter 1. From here on, chapter 3 will discuss Bouthors' methods, with the phenomenological model in particular. This is the approach that comes closest to achieving our stated goal, forming the starting point for our thesis. After touching upon its drawbacks, we will formulate solutions and improvements, drawing from the large discussion that is presented in this chapter. Finally, we discuss our implementation of these solutions.



# Chapter 3

## Cloud rendering

### 3.1 Methods

This section discusses the methods proposed by Bouthors et al. [BNM<sup>+</sup>08] to render cumuliform clouds on the GPU, giving an overview of their modeling and rendering techniques.

#### 3.1.1 The cloud model

##### General shape representation

As mentioned in chapter 2, a triangle mesh was used to represent the general cloud shape. In their paper, they characterized a cumuliform cloud with a mesh consisting of 5000 triangles, yet they showed that any arbitrary mesh may be used in conjunction with their method. Creating such a model however, is an arduous task for artists, which is why they made use of an automated modeling technique as presented in [BN04]. Furthermore, they made mention of *jittering* vertices – that is, randomly offsetting them slightly – of existing models to make the meshes look more like clouds. They applied this to the Stanford bunny, the results of which can be seen in the right image of Figure 3.2.

##### Droplet density representation

To enable a higher level of detail at the silhouette of the cloud in order to simulate detailed edges, Bouthors et al. defined a hypertexture [PH89] – i.e., a surface texture that is superimposed upon an object to produce small deformations – on the boundary of the triangle mesh. They assumed their cloud to have a homogeneous core, i.e., a constant droplet concentration, justified by the notion that significant variance in density occurs mainly in a cloud’s boundary, and even when the core of the cloud does contain significant changes in droplet concentration, its effect is negligible due to the blurring of details caused by the high scattering orders that are dominant in this area.

For the boundary, they did allow for heterogeneity, i.e., a varying density, resulting in the concentration characterization shown in equation (3.1). Here,  $d(\vec{p})$  is the signed closest distance between a point  $\vec{p}$  and the surface of the triangle mesh that represents the cloud shape, while the thickness of the heterogeneous boundary is given by  $h$ .

$$\rho(\vec{p}) = \begin{cases} \rho_0 & \text{if } d(\vec{p}) \geq h \\ \rho_0 s\left(\frac{d(\vec{p})}{h} + p(\vec{p})\right) & \text{if } 0 \leq d(\vec{p}) < h \\ 0 & \text{if } d(\vec{p}) < 0 \end{cases} \quad (3.1)$$

At any point  $\vec{p}$ , the water droplet density at that point,  $\rho(\vec{p})$ , simply equals the density predefined for the homogeneous core,  $\rho_0$ , when  $d(\vec{p})$  is larger than  $h$  – that is, the point  $\vec{p}$  lies inside the homogeneous core. Bouthors et al. chose  $\rho_0 = 400 \text{ cm}^{-3}$  based on values presented in [Mas71].

If  $\vec{p}$  lies in the heterogeneous boundary, i.e.,  $d(\vec{p})$  is smaller than  $h$  but larger than zero, the density is given by multiplying a sigmoid function  $s(x)$  by  $\rho_0$ . The input of the sigmoid function is the signed closest distance divided by the boundary thickness, augmented with three-dimensional fractal Perlin noise  $p(\vec{p})$ . Bouthors et al. claimed that this way, on average, the density varies smoothly from  $\rho_0$  at the border between the homogeneous core and heterogeneous boundary, to zero at the mesh surface, while detail is added through the use of Perlin noise, which is generated during rendering on the GPU and is in the range  $[-1, 1]$ . They did not mention which sigmoid function they used, yet we assume it maps values in the range  $[0, 1]$  to  $[0, 1]$  smoothly. For our implementation, we use the sigmoid function  $s(x)$  described in equation (3.2), which satisfies the criteria  $s(0) = 0$  and  $s(1) = 1$ , and for which both first and second derivative at  $x = 0$  as well as  $x = 1$  equal 0, that is,  $f'(0) = f'(1) = f''(0) = f''(1) = 0$ .

$$s(x) = 6x^5 - 15x^4 + 10x^3 \quad (3.2)$$

For points where the signed distance is negative, which happens when they lie outside the surface-bounded mesh, the density is simply zero.

The difficulty with this approach lies in the distance computation. For simple shapes like boxes or spheres, the distance from a point inside the object to its surface can easily be obtained through geometric calculations. However, for an arbitrary triangle mesh as proposed by Bouthors et al. – a mesh of 5000 triangles, no less – this is no trivial task, and can not be achieved in real-time. They did not mention exactly how they computed the distance, only that it was done in a preprocessing step, as is necessary to realize online visualization. Having implemented our own method of computing the distance, we found this step to be very time consuming – after all, with a naive approach, for every cloud volume position, all mesh triangles need to be taken into account. It took several minutes even for relatively low-detailed triangle meshes, a small number of cloud positions, and a simplified computation algorithm only considering the vertex positions of the mesh.

### Distance storage

Besides the computation, another issue is the storage of the signed distance, which is required since it is computed in a preprocessing step. To allow for a high level of detail – i.e., significant variance between neighboring points – a resolution of at least  $512^3$  is desirable, which, although possible, is unfeasible for direct storage on the GPU as a three-dimensional texture due to high memory requirements. More importantly, however, precomputing the distance for  $512^3$  points takes a tremendous amount of time.

To overcome this problem, Bouthors et al. proposed the use of a sparse voxel octree as presented by Crassin et al. [CNLE09] to store the distance. Only subdividing octree nodes in the heterogeneous boundary, they were able to store the distance with a detail level of  $512^3$  on the GPU efficiently. To obtain the distance for some point  $\vec{p}$ , the octree is sampled on the GPU as described in [LHN05]. The model is illustrated for the two-dimensional case in Figure 3.1.

Bouthors et al. did not quantify  $h$ , merely mentioning that *only a thin layer of voxels is necessary*. The schematic illustrations that they used, like Figure 3.1, suggest a significant boundary thickness. However, from their results it seems that a thin layer was used instead. Consider for instance the left image in Figure 3.2, which has very sharp edges, suggesting only a tiny fraction of the cloud was considered as part of the heterogeneous boundary.

We believe that they used a very thin layer indeed, based on the fact that for boundaries that span around a third of the cloud volume, such as in Figure 3.1, a sparse voxel octree would lose much of its merit, as its original application consists in giving a detailed characterization for only a very small fraction of the whole volume. Also, if a significant boundary thickness is chosen, a  $512^3$  octree would not be feasible due to the same computation- and memory-related issues of a complete grid. Our assumption is further strengthened by the fact that they used a mesh consisting of 5000 triangles, which suggests much care was taken in creating the model, probably since it served as the main influence on the shape, with the hypertexture just being there to add a little bit of detail. Indeed, when comparing the hypertexture result (left image in Figure 3.2) to the Stanford bunny cloud that was generated without it (right image in Figure 3.2), we can see only a small difference in the cloud edges. This is an interesting observation, given the fact that the hypertexture slows their approach down by a factor five.

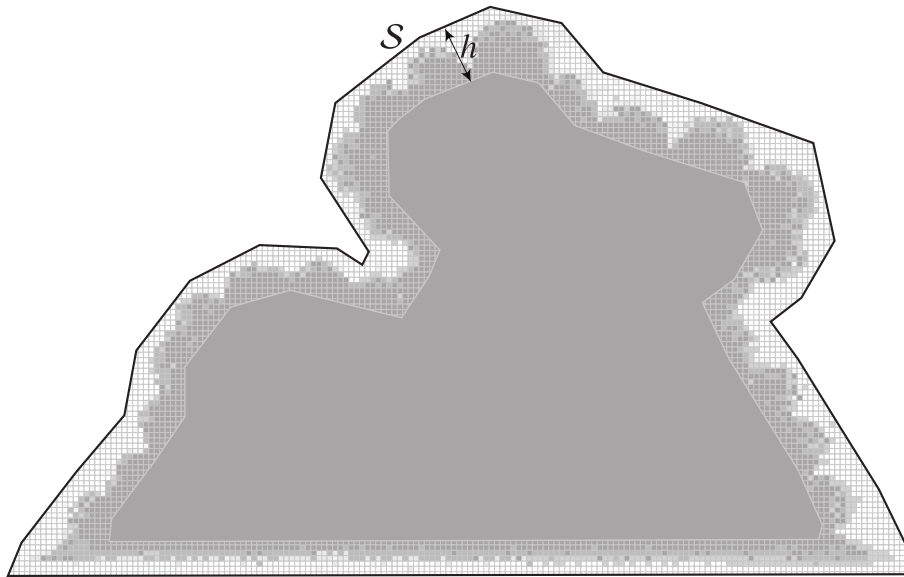


Figure 3.1: The triangle mesh  $S$  contains a heterogeneous boundary of thickness  $h$ , which is represented by a sparse voxel octree. Here, the density has been obtained through the sigmoid function coupled with the Perlin noise and signed distance. It clearly transitions from gray ( $\rho = \rho_0$ ) at the homogeneous core to white ( $\rho = 0$ ) at the mesh surface (shape silhouette in this case). Schematic by Bouthors [Bou08].



(a) Cumulus cloud with hypertexture.



(b) Stanford bunny cloud without hypertexture.

Figure 3.2: The edges of both these clouds are very well-defined, while only the left image makes use of the hypertexture. Images by Bouthors [Bou08].

### 3.1.2 Opacity and single scattering

Bouthors et al. proposed to compute the appearance of every pixel on the cloud's view surface using three components: opacity, and *single scattering* and *multiple scattering* as two separate components to regulate the *light transport*. They defined the light transport similar to the BSSRDF that we have investigated in section 2.1.6.

The three components are calculated through the use of a shader – a computer program that runs on the GPU and can efficiently perform computations on a per-pixel basis. There are some limitations on what a shader can do, however, which are examined in section 3.4.1.

### Opacity through extinction

Bouthors et al. proposed to solve the opacity using the basic radiative properties of a cloud, coupled with their droplet density representation. Recalling section 2.1.8, we can compute the opacity as described in equation (3.4), using the extinction coefficient of equation (3.3).

$$k_e(\vec{p}) = \rho(\vec{p})2\pi r_e^2 \quad (3.3)$$

$$\alpha(\vec{p}_a, \vec{p}_b) = 1 - e^{-\int_{\vec{p}_a}^{\vec{p}_b} k_e(\vec{p})d\vec{p}} \quad (3.4)$$

Bouthors et al. used  $r_e = 6 \mu\text{m}$  for the effective radius, and the density can be obtained by sampling the sparse voxel octree as explained in section 3.1.1, resulting in the extinction coefficient  $k_e(\vec{p})$ .

To find the opacity, however, we need to integrate the extinction coefficient along the view direction between points  $\vec{p}_a$  and  $\vec{p}_b$ , which are the intersection points of the ray (line) describing the view direction, with the surface mesh. More precisely, to cover the case of a concave mesh, they are the intersections that lie closest ( $\vec{p}_a$ ) and furthest ( $\vec{p}_b$ ) from the camera.

### Deferred shading

In order to find these intersections efficiently, Bouthors et al. resorted to the use of depth maps – textures to which the depth buffer is written, with the value in the depth buffer being the distance between the camera and the corresponding point on the mesh. Implementation details of this will be discussed in section 3.4.3. Two depth maps were generated; one containing the points on the viewing surface closest to the camera, also called the front depth map, and one with the points furthest from the camera, called the back depth map. These depth maps correspond to the collection of closest and furthest intersections of the ray and the volume with respect to the camera position.

To obtain the depth maps, they applied *deferred shading*, a technique in which initially, information is gathered in the first passes, before the image is rendered through shading in the final pass. In this case, the cloud shape is rendered three times per frame. The first two passes are used to generate the depth maps, with shading disabled, and the third and final pass computes the resulting pixel color through an advanced shader, making use of the two depth maps.

Sampling them in the final pass, we can obtain the distance to the camera, and use that to get the world coordinates of  $\vec{p}_a$  and  $\vec{p}_b$ .

### Volume ray casting

Now that we have the interval, we can integrate the extinction coefficient. To do this, Bouthors et al. applied the trapezoidal rule; a technique in numerical integration to approximate an integral, based on the premise of equation (3.5).

$$\int_a^b f(x)dx \approx (b-a) \frac{f(a) + f(b)}{2} \quad (3.5)$$

To provide for a more accurate approximation, the interval  $(a, b)$  can be subdivided in several segments, so that we can apply the trapezoidal rule to each segment, as illustrated in Figure 3.3.

To perform this approximation, Bouthors et al. applied a technique called *volume ray casting*, sometimes referred to as volume ray tracing or volume ray marching, which consists in sampling points along the ray (in this case, the ray is *cast* from the camera in the view direction). By obtaining the droplet density at every sampling point, we can calculate the corresponding extinction coefficient and apply the trapezoidal rule to obtain the optical thickness  $\tau$ , which can subsequently be used to find the opacity, resulting in equation (3.6) for  $N$  segments.

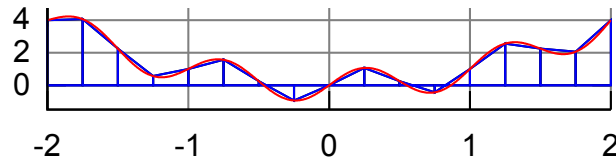


Figure 3.3: The trapezoidal rule can be used to numerically approximate the integral of a function through sampling the function at certain segmentation points. Each segment forms a trapezoid; summing the areas of all trapezoids results in an approximation of the whole integral.

$$\alpha(\vec{p}_a, \vec{p}_b) = 1 - e^{-\tau(\vec{p}_a, \vec{p}_b)} \quad (3.6)$$

$$\tau(\vec{p}_a, \vec{p}_b) = \int_{\vec{p}_a}^{\vec{p}_b} k_e(\vec{p}) d\vec{p} = \sum_{i=0}^{N-1} \tau(\vec{p}_i, \vec{p}_{i+1}) \approx \sum_{i=0}^{N-1} \|\vec{p}_{i+1} - \vec{p}_i\| \frac{k_e(\vec{p}_i) + k_e(\vec{p}_{i+1})}{2}$$

A schematic overview of the method is given in Figure 3.4, where  $x = \|\vec{p}_b - \vec{p}_a\|$  and  $x_i = \|\vec{p}_{i+1} - \vec{p}_i\|$ . The sampling points  $\vec{p}_i$  are chosen so that they lie in the heterogeneous boundary to best approximate the integral; after all, the integral of the constant extinction coefficient of the homogeneous core can be obtained by just using one segment.

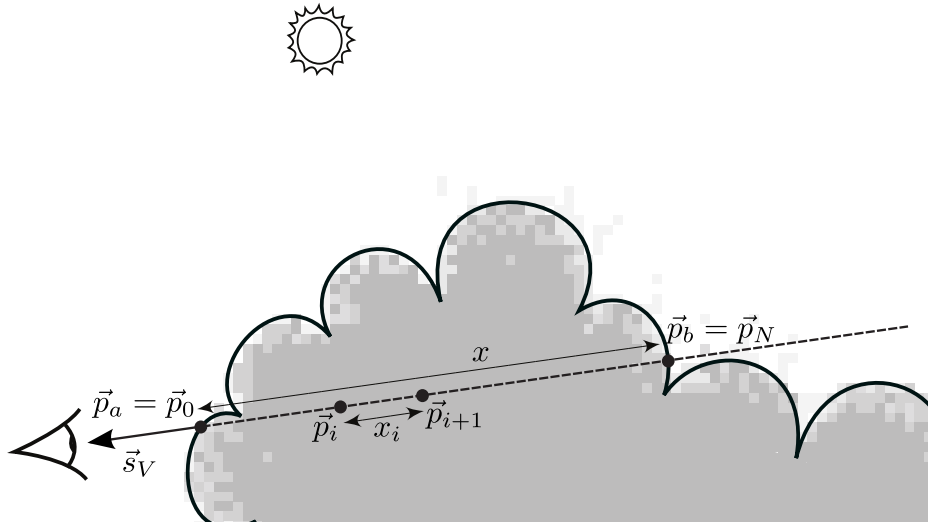


Figure 3.4: Opacity calculation through ray marching. For every segment  $x_i$ , the octree texture is sampled at  $\vec{p}_i$  and  $\vec{p}_{i+1}$  to obtain the density. The trapezoidal rule is used to compute the optical thickness  $\tau(\vec{p}_i, \vec{p}_{i+1})$  of the segment, and the summed optical thickness over all segments,  $\tau(\vec{p}_a, \vec{p}_b)$ , is used to compute the opacity as per equation (3.6). Based on a schematic by Bouthors [Bou08].

To reiterate, we compute the opacity as per equation (3.7). Bouthors et al. made use of the octree structure by taking segments of large  $x_i$  when the homogeneous concentration  $\rho_0$  is obtained from the density calculation. They used a total of around twenty segments for opacity calculation, i.e.,  $N = 20$ .

$$\alpha(\vec{p}_a, \vec{p}_b) = 1 - e^{-\tau(\vec{p}_a, \vec{p}_b)} \quad (3.7)$$

$$\tau(\vec{p}_a, \vec{p}_b) \approx \sum_{i=0}^{N-1} x_i \frac{k_e(\vec{p}_i) + k_e(\vec{p}_{i+1})}{2}$$



It should be noted that Bouthors et al. also used ray tracing for the generation of the depth maps, considering intersection points only when the sampled density exceeds some predefined value  $\rho_1$ . To obtain the intersections, they ray marched through the volume, setting the octree voxel location as the intersection point when the droplet concentration surpasses  $\rho_1$ .

### Single scattering

Single scattering is solved similarly to opacity. However, we are now interested in the luminance  $L(\vec{p}_a, \vec{s}_V)$  at point  $\vec{p}_a$  in view direction  $\vec{s}_V$ . Classically, in accordance with the radiative transfer equation (2.49), this would be done as in equation (3.8). As with opacity, we integrate over all points that lie between  $\vec{p}_a$  and  $\vec{p}_b$ . This time however, for every point  $\vec{p}$ , we multiply the transparency  $\beta(\vec{p}_a, \vec{p})$  – i.e., the extinction function – of the path  $(\vec{p}_a, \vec{p})$  with the extinction coefficient  $k_e(\vec{p})$  and the luminance arriving at point  $\vec{p}$ ,  $L(\vec{p}, \vec{s}_L)$ , multiplied by the phase function  $P(\Theta)$ , integrated over all directions. Note that in equation (3.8),  $\vec{s}_L$  is described by  $(d\theta_L, d\phi_L)$ , and  $\cos \Theta = \vec{s}_V \cdot \vec{s}_L$ .

However, only the sun is considered as a light source for single scattering, thus it is not necessary to integrate the incoming luminance at  $\vec{p}$  over all directions. The luminance at point  $\vec{p}$  in direction  $\vec{s}_V$  is given by the incident luminance from direction  $\vec{s}_L$ , multiplied by the phase function, so that we obtain equation (3.9).

The incident luminance at  $\vec{p}$  in direction  $\vec{s}_V$  can then be obtained by multiplying the luminance arriving at the cloud surface with the transparency of the path between  $\vec{p}$  and the point on the cloud surface in direction  $-\vec{s}_L$ , which we call  $\vec{p}_l$ . This results in equation (3.10).

Writing out the two extinction functions in equation (3.11), we can see a problem arising. Due to the two inner integrals, this equation can no longer be efficiently approximated using the trapezoidal rule. Instead, Bouthors et al. opted for considering the extinction coefficient to be constant over the path  $(\vec{p}, \vec{p}_l)$ , and equal to  $k_e(\vec{p})$ , simplifying it to equation (3.12), where  $l = \|\vec{p}_l - \vec{p}\|$  is the distance from  $\vec{p}$  to the cloud surface in direction  $-\vec{s}_L$ .

$$L(\vec{p}_a, \vec{s}_V) = \int_{\vec{p}_a}^{\vec{p}_b} \beta(\vec{p}_a, \vec{p}) k_e(\vec{p}) \int_0^{2\pi} \int_0^\pi P(\Theta) L(\vec{p}, \vec{s}_L) d\theta_L d\phi_L d\vec{p} \quad (3.8)$$

$$= \int_{\vec{p}_a}^{\vec{p}_b} \beta(\vec{p}_a, \vec{p}) k_e(\vec{p}) P(\Theta) L(\vec{p}, \vec{s}_L) d\vec{p} \quad (3.9)$$

$$= \int_{\vec{p}_a}^{\vec{p}_b} \beta(\vec{p}_a, \vec{p}) k_e(\vec{p}) P(\Theta) \beta(\vec{p}, \vec{p}_l) L(\vec{p}_l, \vec{s}_L) d\vec{p} \quad (3.10)$$

$$= \int_{\vec{p}_a}^{\vec{p}_b} e^{-\int_{\vec{p}_a}^{\vec{p}} k_e(\vec{p}') d\vec{p}'} k_e(\vec{p}) P(\Theta) e^{-\int_{\vec{p}}^{\vec{p}_l} k_e(\vec{p}') d\vec{p}'} L(\vec{p}_l, \vec{s}_L) d\vec{p} \quad (3.11)$$

$$= \int_{\vec{p}_a}^{\vec{p}_b} e^{-\int_{\vec{p}_a}^{\vec{p}} k_e(\vec{p}') d\vec{p}'} k_e(\vec{p}) P(\Theta) e^{-lk_e(\vec{p})} L(\vec{p}_l, \vec{s}_L) d\vec{p} \quad (3.12)$$

Note that we can bring the phase function out of the integral, since the scattering angle does not change. This is due to the fact that  $\vec{s}_L$  can be approximated to be constant. In real life, the vast distance from the earth to the sun renders the distance between objects on earth negligible, allowing us to consider rays of sunlight to be parallel. This means that no matter which point in a cloud is inspected, the corresponding direction of sunlight remains the same. Bouthors et al. did not mention how they determined this vector precisely; yet for our implementation we simply consider the normalized vector pointing from the sun position to the cloud center as  $\vec{s}_L$ , where the cloud center  $\vec{C}$  is defined as the averaged extremities in the  $\vec{x}$ ,  $\vec{y}$  and  $\vec{z}$  direction, as in equation (3.13).

$$C_x = \frac{x_{\min} + x_{\max}}{2} \quad (3.13)$$

$$C_y = \frac{y_{\min} + y_{\max}}{2} \quad (3.14)$$

$$C_z = \frac{z_{\min} + z_{\max}}{2} \quad (3.15)$$

For the same reason, we can bring out the luminance incident on the cloud surface, since it is constant as well for any point on the cloud surface. Additionally, recalling that we are actually interested in the light transport  $T(\vec{p}_a, \vec{s}_V)$  – that is, the ratio between the luminance incident on the cloud surface and the luminance coming from point  $\vec{p}_a$  on the cloud surface in direction  $\vec{s}_V$  – rather than the luminance  $L(\vec{p}_a, \vec{s}_V)$ , we can set  $L(\vec{p}_i, \vec{s}_L)$  to one to simplify equation (3.12). More precisely, we consider only the light transport due to single scattering here, i.e.,  $T_1(\vec{p}_a, \vec{s}_V)$ . All of this results in equation (3.16).

$$T_1(\vec{p}_a, \vec{s}_V) = P(\Theta) \int_{\vec{p}_a}^{\vec{p}_b} k_e(\vec{p}) e^{-\int_{\vec{p}_a}^{\vec{p}} k_e(\vec{p}') d\vec{p}'} e^{-lk_e(\vec{p})} d\vec{p} \quad (3.16)$$

It should be noted that Bouthors et al. used a slightly different Mie phase function for each of the standardized red, green and blue channels, thus equations (3.17), (3.18) and (3.19) would be more precise. However, the integral remains the same for all channels and only has to be computed once.

$$T_{1R}(\vec{p}_a, \vec{s}_V) = P(\Theta)_R \int_{\vec{p}_a}^{\vec{p}_b} k_e(\vec{p}) e^{-\int_{\vec{p}_a}^{\vec{p}} k_e(\vec{p}') d\vec{p}'} e^{-lk_e(\vec{p})} d\vec{p} \quad (3.17)$$

$$T_{1G}(\vec{p}_a, \vec{s}_V) = P(\Theta)_G \int_{\vec{p}_a}^{\vec{p}_b} k_e(\vec{p}) e^{-\int_{\vec{p}_a}^{\vec{p}} k_e(\vec{p}') d\vec{p}'} e^{-lk_e(\vec{p})} d\vec{p} \quad (3.18)$$

$$T_{1B}(\vec{p}_a, \vec{s}_V) = P(\Theta)_B \int_{\vec{p}_a}^{\vec{p}_b} k_e(\vec{p}) e^{-\int_{\vec{p}_a}^{\vec{p}} k_e(\vec{p}') d\vec{p}'} e^{-lk_e(\vec{p})} d\vec{p} \quad (3.19)$$

The problem of the inner integral persists. However, upon closer inspection, we can conclude that we are doing double work here. After all, we are integrating over the points between  $\vec{p}_a$  and  $\vec{p}$  for *every* point  $\vec{p}$  between  $\vec{p}_a$  and  $\vec{p}_b$ . This means that we consider several points multiple times. With this in mind, Bouthors et al. optimized the equation to enable a numerical approximation, resulting in equation (3.20) for  $N$  segments. The trapezoidal rule is applied again, and we store the outer integral in the function  $F(\vec{p})$ . Equation 3.21 shows the inner integral, for which the optical thickness  $\tau(\vec{p}_a, \vec{p}_i)$  between  $\vec{p}_a$  and  $\vec{p}_i$  is used. The most important approximation, that allows us to perform numerical integration efficiently, is shown in equation 3.22. For every segment, the optical thickness is saved in a variable that is used for the next segment. Note that we also need to save the extinction coefficient of the previous segment,  $k_e(\vec{p}_{i-1})$ , as well as its length  $x_{i-1}$ . Using this method, the single scattering can be adequately approximated with the same efficiency as the opacity.

$$T_1(\vec{p}_a, \vec{s}_V) \approx P(\Theta) \sum_{i=0}^{N-1} x_i \frac{F(\vec{p}_i) + F(\vec{p}_{i+1})}{2} \quad (3.20)$$

$$F(\vec{p}_i) = k_e(\vec{p}_i) e^{-l_i k_e(\vec{p}_i)} e^{-\tau(\vec{p}_a, \vec{p}_i)} \quad (3.21)$$

$$\tau(\vec{p}_a, \vec{p}_i) = \begin{cases} \tau(\vec{p}_a, \vec{p}_{i-1}) + x_{i-1} \frac{k_e(\vec{p}_{i-1}) + k_e(\vec{p}_i)}{2} & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{cases} \quad (3.22)$$

The whole concept is illustrated in Figure 3.5. Note that the sun direction  $\vec{s}_L$  as well as the line segments between any  $\vec{p}_i$  and  $\vec{p}_{i+1}$  are parallel, as explained before. The distance between  $\vec{p}_i$  and  $\vec{p}_{i+1}$  is given by  $l_i$ , which is used as input for equation (3.21).

However, we have not yet discussed how to obtain this value. To this end, Bouthors et al. proposed the use of another depth map, this time rendered from the viewpoint of the sun. In this additional pass, the camera is set to the sun position, and the cloud is rendered using orthographic projection to simulate the parallel nature of solar rays. Normally, scenes are rendered using a perspective projection matrix to imitate real-life three-dimensional perspective, yet we want a parallel projection matrix for this application to simulate the vast distance to the sun. The distance to the closest point on the cloud surface is stored similarly to the previously discussed depth maps. During the final pass, besides reading the depth maps generated from the camera viewpoint, we now also sample the depth map that is generated from the point of view of the sun. With this distance value, we can obtain  $l_i$  and solve the single scattering equation to retrieve the light transport.

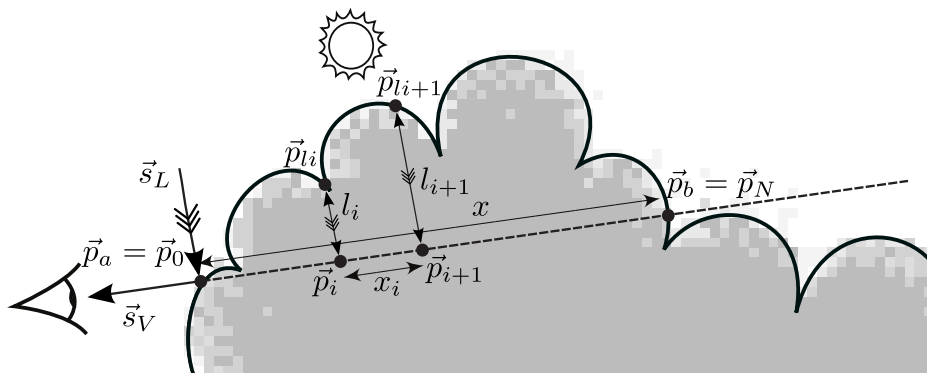


Figure 3.5: Single scattering calculation through ray marching. For every segment  $x_i$ , the octree texture is sampled at  $\vec{p}_i$  and  $\vec{p}_{i+1}$  to obtain the density. Then, the sun depth map is sampled to find  $l_i$  and  $l_{i+1}$ . Finally, the optical thickness  $\tau(\vec{p}_a, \vec{p}_i)$  is computed using equation (3.22), and we can obtain the light transport with equation (3.20). Note that prior to considering the next segment,  $x_i$ ,  $k_e(\vec{p}_i)$  and  $\tau(\vec{p}_a, \vec{p}_i)$  are saved in a variable for future reference. Based on a schematic by Bouthors [Bou08].

Bouthors et al. opted for  $N = 10$  segments for the single scattering approximation, spacing them logarithmically along the distance  $x$ . They chose for this since  $F(\vec{p}_i)$  decreases exponentially as the cloud volume is marched, due to the growing optical thickness being situated in a negative exponent.

For this logarithmic spacing, we apply the following method. Given a starting number  $q_1$ , final number  $q_n$  and  $n$  the number of intervals, we first compute the logarithmic multiplier  $t$  as in equation (3.23). Then we can obtain the  $i^{\text{th}}$  number using equation (3.24). With this, we can define  $\vec{p}_i$  when ray marching through the cloud volume.

$$t = \frac{q_n^{\frac{1}{n}}}{q_1} \quad (3.23)$$

$$q_i = q_1 t^i \quad (3.24)$$

### 3.1.3 A study of light transport in plane-parallel slabs

#### Orders of scattering

To obtain the light transport due to multiple scattering, Bouthors et al. proposed the use of a phenomenological model. Recalling the discussion of several of such methods in chapter 2, we can say it is no easy feat to construct a model that simulates all effects of multiple scattering as mentioned in section 2.2. Bouthors et al. proposed to define this model for several sets of orders of scattering separately, based on the notion that the orders in a set display similar behavior. Indeed, scattering orders of above thirty, for instance, all display isotropic behavior, causing diffuse lighting. However, the lower the scattering order, the more distinct the behavior, due to the anisotropy largely being preserved. To this end, Bouthors et al.

chose to treat the following sets of orders of scattering separately:  $\{2, 3, 4, 5 - 6, 7 - 9, 10 - 15, 16 - \infty\}$ . Note that in their paper they mentioned different sets, yet in their thesis as well as in the data they supplied, the aforementioned numbers were used.

### Plane-parallel slab

In order to devise a phenomenological model for multiple scattering, Bouthors et al. first set out to conduct a study of light transport in plane-parallel slabs. In this model, two infinite planes  $P_t$  and  $P_b$  are placed parallel to one another at a distance  $t$  apart – the thickness of the slab. They are arranged so that the light source is above the top plane  $P_t$ . They studied the light transport caused by photons emitted by the light source, entering  $P_t$ , and subsequently arriving in direction  $\vec{s}_V$  at some point  $\vec{p}$  that lies between the two planes – note that it may also lie *on* either  $P_t$  (which reduces to reflectance) or  $P_b$  (transmittance). We denote the distance between  $\vec{p}$  and  $P_t$  as  $d$ , the viewpoint depth, and correspondingly the distance between  $\vec{p}$  and  $P_b$  as  $d'$ , so that  $t = d + d'$ .

The coordinate system is determined as follows. The  $\vec{y}$  axis is chosen to be perpendicular to the planes, i.e., equal to the normal so that  $\vec{y} = \vec{n}$ . The  $\vec{z}$  axis is selected so that the vector  $\vec{s}_L$  denoting the light direction lies in the plane formed by  $\vec{y}$  and  $\vec{z}$ , or, perhaps more precisely,  $\vec{z}$  is picked so that it lies in the plane defined by  $\vec{y}$  and  $\vec{s}_L$ . Naturally,  $\vec{x}$  is perpendicular to both  $\vec{y}$  and  $\vec{z}$  and is obtained through the cross product.

We furthermore define the angle between  $\vec{y}$  and  $-\vec{s}_L$  as  $\theta_L$ . Additionally, the angle between  $-\vec{y}$  and  $\vec{s}_V$  is denoted  $\theta_V$ , while  $\phi_V$  is the angle between  $-\vec{z}$  and the projection of  $\vec{s}_V$  onto the plane defined by  $\vec{x}$  and  $\vec{z}$ . Bouthors et al. opted for these definitions, and despite the fact that taking the negative axes to define the angles seems illogical, we are inclined to respect their system to avoid confusion. The whole model is illustrated in Figure 3.6 for clarity.

Bouthors et al. hypothesized that if they simulated enough offline experiments with different parameters in this plane-parallel slab, they would be able to make a valid approximation of the light transport given any five input parameters –  $t$ ,  $d$ ,  $\theta_V$ ,  $\phi_V$  and  $\theta_L$ . Besides the advantage of an opportunity to make some useful observations about light behavior, they posed that if a cloud could be approximated as a plane-parallel slab, they would be able to obtain the multiple scattering component at an accurate enough level to retain all cloud features.

They conducted these experiments for the previously mentioned sets of orders of scattering separately, to allow the final phenomenological model to simulate the cloud features of section 2.2. This introduced an additional input parameter  $n$  defining the considered set of scattering orders, with  $n \in \{1, 7\}$ , since there are seven sets.

The function describing the light transport is then defined as a *bidirectional scattering distribution function* (BSDF), as it denotes the ratio between the luminance at point  $\vec{p}$  and the illuminance incident on  $P_t$ . When  $\vec{p}$  lies on  $P_t$ , this BSDF reduces to a BRDF, while if  $\vec{p}$  is located on  $P_b$ , it reduces to a BTDF. The BSDF  $C$  is defined in equation 3.25, and the BRDF  $R$  and BTDF  $T$  in 3.26 and 3.27 respectively, with the viewpoint depth  $d$  not being of importance, as  $d = 0$  for  $R$  and  $d = t$  for  $T$ .

$$C(n, t, d, \theta_V, \phi_V, \theta_L) \quad (3.25)$$

$$R(n, t, \theta_V, \phi_V, \theta_L) \quad (3.26)$$

$$T(n, t, \theta_V, \phi_V, \theta_L) \quad (3.27)$$

### Experiment implementation

To perform their simulations, they implemented a Monte Carlo path tracer as follows. Starting at point  $\vec{p}$ , they randomly traced a very large amount of paths back to the light source using the extinction function  $\beta$  and the Mie phase function  $P$  with the previously mentioned effective radius  $r_e = 6 \mu\text{m}$  and density  $\rho = 400 \text{ cm}^{-3}$  kept constant throughout the simulation.

Keeping track of the resulting light transport due to all these light paths, they were able to define  $C$  (3.25) for millions of sets of varying slab thickness  $t$ , viewpoint depth  $d$ , view angles  $\theta_V$  and  $\phi_V$ , and

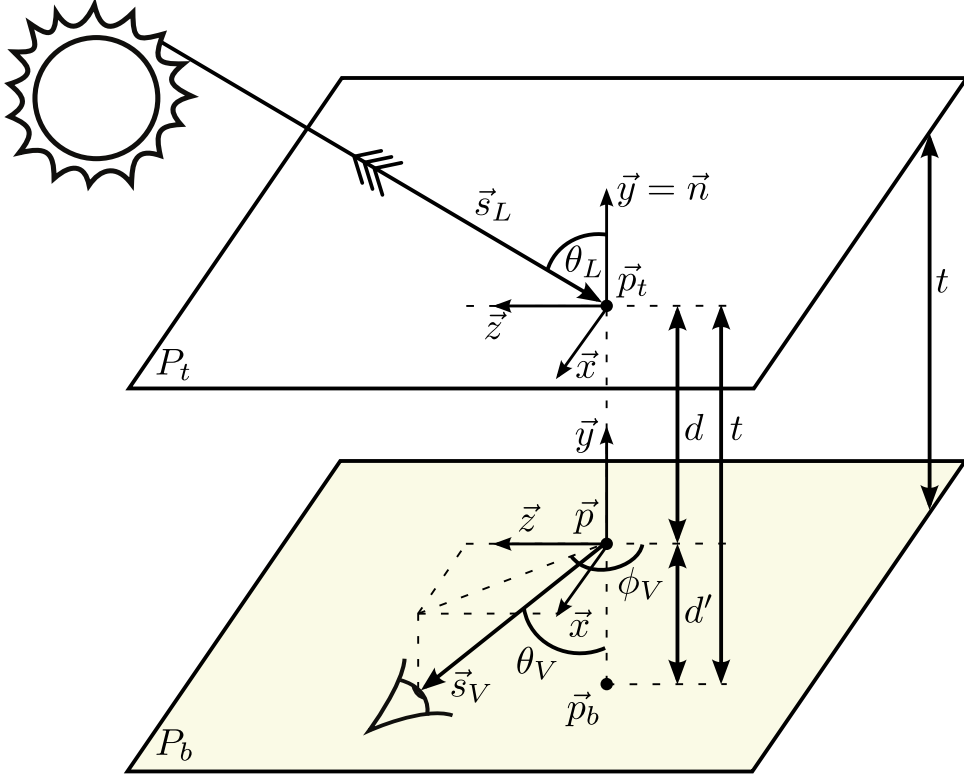


Figure 3.6: The plane-parallel slab model in which light transport was extensively studied by Bouthors et al. Note that in this case,  $\vec{p}$  lies between  $P_t$  and  $P_b$ . Additionally, in reality,  $P_t$  and  $P_b$  are infinite planes.

light angle  $\theta_L$ . Taking into account the number of scattering events when tracing the light paths, they attributed the light transport to the correct set number  $n$  to generate  $C$  for all seven sets separately.

Note that due to the fact that  $P_t$  and  $P_b$  are infinite, it is irrelevant where  $\vec{p}$  lies to obtain  $C$ , except for its depth in the slab, given by  $d$ .

### Collector area

Besides investigating the measure of light transport,  $C$ , Bouthors et al. were also interested in the distribution of the light paths. That is, during simulation, they also kept track of where on  $P_t$  the traced light path entered the slab, relative to  $\vec{p}$ , to obtain a description of the surface distribution  $D$  of the light entrance points, given a slab model.

To further elucidate this idea, let us re-examine the BSSRDF described in section 2.1.6 first, its definition given in equation (2.39). In the case of the plane-parallel slab, we can reduce the eight dimensions, first by omitting  $\phi_L$ , which is allowed due to the fact we align the coordinate system so that  $\phi_L = 0$  at all times. We can reduce it further by taking advantage of the infinite planes that we use, by letting  $x' = x_L - x_V$  and  $z' = z_L - z_V$ , only defining the *relative* location of the view and light points. This reduces the original BSSRDF  $S$  to a function  $D$ , which is shown in equation (3.28) and makes use of the input parameters as illustrated in Figure 3.6.

$$D(n, t, d, x', z', \theta_V, \phi_V, \theta_L) \quad (3.28)$$

To reiterate,  $C$  denotes the *bidirectional* distribution of light transport in a slab, while  $D$  denotes the *surface* distribution of light transport in a slab, that is, the density distribution of the light entrance



points on  $P_t$ .

Instead of defining this function through its current eight parameters, Bouthors et al. opted to define  $D$  through its mean and standard deviation. The mean of  $D$ , also called the first moment, is obtained by integrating it over all points, with the considered point  $\vec{p}' = \{x', 0, z'\}$ , as in equation (3.29), resulting in the mean  $\vec{c} = \{c_x, 0, c_z\}$ .

The standard deviation or second moment is obtained by integrating the squared distance between the considered point  $\vec{p}'$  and the mean  $\vec{c}$ , over all points. This is done separately for the standard deviation over  $x$ ,  $\sigma_x^2$  (3.30), and the one over  $z$ ,  $\sigma_z^2$  (3.31).

$$\vec{c}(n, t, d, \theta_V, \phi_V, \theta_L) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} D(n, t, d, x', z', \theta_V, \phi_V, \theta_L) \vec{p}' dx' dz' \quad (3.29)$$

$$\sigma_x^2(n, t, d, \theta_V, \phi_V, \theta_L) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} D(n, t, d, x', z', \theta_V, \phi_V, \theta_L) (x' - c_x)^2 dx' dz' \quad (3.30)$$

$$\sigma_z^2(n, t, d, \theta_V, \phi_V, \theta_L) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} D(n, t, d, x', z', \theta_V, \phi_V, \theta_L) (z' - c_z)^2 dx' dz' \quad (3.31)$$

Together,  $\vec{c}$ ,  $\sigma_x^2$  and  $\sigma_z^2$  define the *collector area*, which is defined as the area on  $P_t$  on which 95% of the light paths' entrance points lie. This idea is inspired by the most probable paths method [PAS03] [PARN04] that has been mentioned in chapter 2, yet we now consider an area instead of a single most likely entrance point. Bouthors et al. simplified the standard deviation by assuming the distribution to be symmetric, so that  $\sigma = \sigma_x = \sigma_z$ . The idea is illustrated in Figure 3.7. Having noted the distribution to display Gaussian behavior, they defined the collector area as a flat disk described by its center,  $\vec{c}$ , and its radius  $\sigma$ .

Their motivation for keeping track of the collector area during the experiments was founded in the idea of being able to find a plane-parallel slab fitting an arbitrary cloud model. Its importance will be further discussed in section 3.1.4.

## Results of the study

From their experiments, Bouthors et al. were not only left with a huge database containing the outcome for millions of sets of input parameters, but also with some interesting insights in the behavior of light in plane-parallel slabs that translate to arbitrary clouds as well. We briefly reiterate the most important conclusions with respect to the cloud features here, and refer to [Bou08] for the experiment results and reasoning justifying these statements.

Once the slab thickness  $t$  exceeds approximately 500m, it becomes more reflective than transmissive, causing the cloud top to be brighter than its base. Roughly dividing the scattering orders in low orders, with fifteen or less scattering events, and high orders with more than fifteen of these, they concluded that the former attribute to anisotropic cloud features, with the corresponding light paths being short with most entry points localized in a small area. The latter on the other hand consist of long light paths with very spread out entry points, causing isotropic cloud features.

Bouthors et al. made several conclusions pertaining to three important parts of a cloud, namely its edges, base, and top. The appearance of cloud edges corresponds to light behavior in very thin slabs, with low orders of scattering dominating. This causes most light to exit in the forward direction, and the anisotropy of the Mie phase function is kept largely intact. Due to this, cloud edges are much brighter when viewed from the antisolar point. Furthermore, the spatial spreading of the entrance points, i.e., the surface distribution, is very small. The result is that heterogeneities are clearly visible on the cloud boundaries.

Cloud bases are characterized similarly to the bottom of relatively thick slabs, with high orders of scattering dictating light transport. This means that transmittance is low, and the angular distribution is more uniformly dispersed than for cloud edges, resulting in almost isotropic behavior. Additionally, the spatial spreading is exceedingly high depending on the thickness of the slab. All of this causes the heterogeneities to be flattened, resulting in a smoothed look of low contrast.

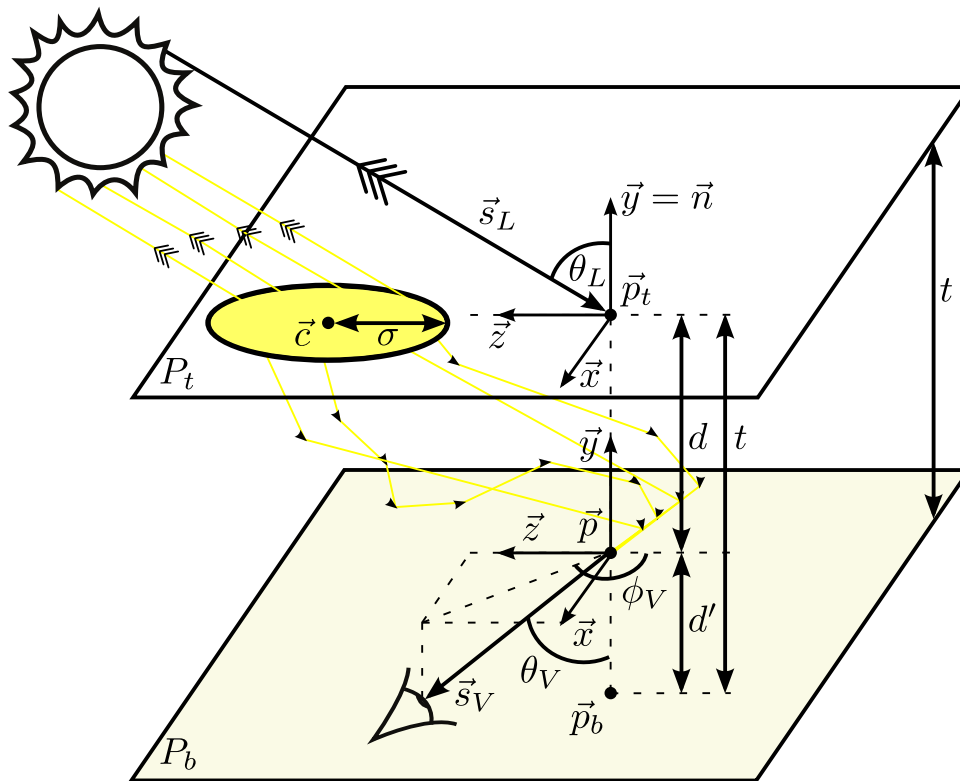


Figure 3.7: The plane-parallel slab model with the collector area as defined by its center  $\vec{c}$  and size  $\sigma$ . Almost all light (95%) that reaches point  $\vec{p}$  in direction  $\vec{s}_V$  entered the slab at a point that lies within the collector area.

The top of clouds on the other hand corresponds to the appearance of the top of the same thick slabs, which is made up of light paths consisting of all orders of scattering. The reflectance is high, and the angular distribution is a mixture of anisotropic features (glory, fogbow, etc.) due to low orders of scattering, and isotropic features (diffuse look) caused by high scattering orders. The same goes for the spatial spreading, which is a blend of small and large due to the different scattering orders. Details are visible due to low orders of scattering, yet not as much as in the cloud edges, as per the diffuse effect of high scattering orders.

### 3.1.4 An efficient phenomenological model of light transport in clouds

Being left with several gigabytes of data resulting from their study of light transport in plane-parallel slabs, Bouthors et al. posed that if an arbitrary cloud could be approximated by such a slab, the corresponding multiple scattering component could be easily interpolated from this database. However, this already poses problems for real-time performance on the CPU, let alone on the GPU, where memory is even more limited. To reduce the amount of data, they set out to formulate a parameter function to approximate the light transport as faithfully as possible.

#### Model for light transport

For the light transport, the parameter function was found through experimentation with trigonometric and logarithmic functions, using non-linear least-square fitting in Matlab. The function that they deduced is given in equation (3.32).

All variables denoted by \* are actually functions themselves, with  $P^*$  (3.33) modulating the anisotropy,

taking slab thickness  $t$  and the scattering angle  $\Theta$  as input. Note that when  $n = 7$ , we are looking at scattering orders  $(16, \infty)$ , which have an isotropic phase function, thus  $P^*(7, t, \Theta) = 1$  for any  $t$  and  $\Theta$ .

$A^*$  (3.34) on the other hand consists of two other functions, and it encodes the maximum value of  $C$ . The parameter  $B^*$  (3.35) defines the viewpoint depth at which  $C$  starts to drastically decrease, and is composed of two functions as well. The broadness of the peak that appears when plotting  $C$  against slab thickness  $t$  is modulated by  $C^*$  (3.36), while the logarithmic behavior of this same plot corresponds to the parameter  $D^*$  (3.37).

$$C(n, t, d, \theta_V, \phi_V, \theta_L) = P^* A^* \cos \theta_L \frac{\log^2(d + D^*)}{\log^2(B^* + D^*)} e^{-\frac{(d-B^*)^2}{2(C^*)^2}} \quad (3.32)$$

$$P^* = P^*(n, t, \Theta) \quad (3.33)$$

$$A^* = A1^*(n, t, \theta_V) \cdot A2^*(n, t, \theta_L) \quad (3.34)$$

$$B^* = B1^*(n, t, \theta_V) - B2^*(n, t, \theta_L) \quad (3.35)$$

$$C^* = C^*(n, t, \theta_V) \quad (3.36)$$

$$D^* = D^*(n, t, \theta_V) \quad (3.37)$$

The functions in equations (3.33) through (3.37) –  $P^*$ ,  $A1^*$ ,  $A2^*$ ,  $B1^*$ ,  $B2^*$ ,  $C^*$  and  $D^*$  – are stored in precomputed two-dimensional tables for every set of scattering order, allowing for linear interpolation to retrieve approximate results for arbitrary  $t$ ,  $\theta_V$ ,  $\phi_V$  and  $\theta_L$ . This approach vastly reduces the amount of data that is required, allowing for an implementation on the GPU, storing the aforementioned tables in textures. The tables that are used in this thesis corresponding to these seven functions can be found on Bouthors' thesis website at <http://evasion.imag.fr/~Antoine.Bouthors/research/phd/>.

Like many phenomenological models, it is difficult to grasp the physical meaning of the parameters, if there is any, yet as described in chapter 2, we are only interested in the *results* of these kind of models. However, Bouthors et al. do provide for convincing justification for their model, yet their analysis of simulation results is too lengthy to fully delve into here. Chapters 4 and 5 of their thesis [Bou08] contain more details on the construction of this model.

### Model for collector area

Besides the light transport, Bouthors et al. devised a phenomenological model for determining the collector area as well, i.e.,  $\vec{c} = \{c_x, 0, c_z\}$  and  $\sigma$ . The functions for the first moment or the mean, that is, the coordinates  $c_x$  and  $c_z$ , are given in equations (3.38) and (3.41), respectively. Again, parameters that are functions themselves are denoted with \*, resulting in the ten functions of equations (3.44) through (3.53).

$$c_x(n, d, \theta_V, \phi_V, \theta_L) = A_x^* \log^2(1 + E^* d) + B_x^* \quad (3.38)$$

$$A_x^* = F^* \sin \phi_V \sin(G^* \theta_L) \quad (3.39)$$

$$B_x^* = H \sin \phi_V \sin \theta_L \quad (3.40)$$

$$c_z(n, d, \theta_V, \phi_V, \theta_L) = A_z^* \log^2(1 + E^* d) + B_z^* \quad (3.41)$$

$$A_z^* = I^* + J^* (\cos \phi_V \sin(K^* \theta_L) + L^* \theta_L) \quad (3.42)$$

$$B_z^* = M^* + N^* \cos \phi_V \quad (3.43)$$

$$E^* = E^*(n, \cos \theta_V) \quad (3.44)$$

$$F^* = F^*(n, \cos \theta_V) \quad (3.45)$$

$$G^* = G^*(n, \cos \theta_V) \quad (3.46)$$

$$H^* = H^*(n, \cos \theta_V) \quad (3.47)$$

$$I^* = I^*(n, \cos \theta_V) \quad (3.48)$$

$$J^* = J^*(n, \cos \theta_V) \quad (3.49)$$

$$K^* = K^*(n, \cos \theta_V) \quad (3.50)$$

$$L^* = L^*(n, \cos \theta_V) \quad (3.51)$$

$$M^* = M^*(n, \cos \theta_V) \quad (3.52)$$

$$N^* = N^*(n, \cos \theta_V, \cos \theta_L) \quad (3.53)$$

As with the light transport, these functions were discretized for every set of scattering orders  $n$ , and subsequently converted to textures to be able to access them on the GPU. Again, the discretized values for these ten functions are available on Bouthors' website.

This model seems to make even less sense than the one for light transport, yet it was the best approximation Bouthors et al. were able to find. For the second moment or standard deviation  $\sigma$ , they devised a much simpler model, as seen in equation (3.54), where the variables  $O$ ,  $Q$ ,  $R$ ,  $S$  and  $T$  are simply scalars. Note that for low orders with less than sixteen scattering events, i.e.,  $n < 7$ , another function is used, where the parameters  $S_a^*$ ,  $S_b^*$  and  $S_c^*$  are once again functions. Luckily, however, they just depend on the set of scattering order, so seven scalars suffice to define them. All these parameters are available on Bouthors' website as well.

$$\sigma(n, t, d) = \begin{cases} O + Qt \log^2(1 + Rd) + S \log^2(1 + Tt) & \text{if } n = 7 \\ S_a^* + S_b^* \log^2(1 + S_c^* d) & \text{if } n < 7 \end{cases} \quad (3.54)$$

### Fitting a plane-parallel slab to an arbitrary cloud

So far, only the light transport in a plane-parallel slab has been investigated. To extend this to arbitrary cloud models, the cloud needs to be approximated by a slab in such a way that an adequate multiple scattering component can be found using the models discussed in the previous paragraphs. In order to fit a slab on a cloud, Bouthors et al. made use of the collector area.

They posed that given a viewpoint  $\vec{p}$  and a corresponding, known correct collector area defined by  $\vec{c}$  and  $\sigma$ , the corresponding slab can be deduced. For this, we need one additional depth map. Recalling section 3.1.2, we already have a front depth map rendered from the sun position. Additionally, we now obtain a back depth map, by considering the *furthest* points from the sun's point of view. Furthermore, we render a *normal map* from this viewpoint, so that every pixel in this map contains the normal of the closest point on the cloud surface when viewed from the position of the sun. This means that we now have a total of six passes; two for depth maps from the camera position, two for depth maps from the sun position, one for a normal map from the sun position, and one final pass from the camera position to compute the cloud appearance.

Sampling the three sun maps at the collector center location  $\vec{c}$ , expressed in the sun's screen-space coordinates, we can obtain both the normal at this point on the cloud surface,  $\vec{n}$ , as well as the thickness of the cloud,  $t$ . However, we are considering a collector *area*, and not just one most probable path. Therefore, we need to sample the whole area, weighted with the spatial spreading defined by  $D$ . Bouthors et al. observed that this spreading can be approximated nicely using a Gaussian distribution, setting the standard deviation to  $\sigma$ . Still, integrating the Gaussian over all points on the collector area, especially for large  $\sigma$ , seems unfeasible. However, the GPU is capable of accurately approximating this with tremendous efficiency, making use of mipmapping. Implementing this is more thoroughly discussed in section 3.4.3, yet it boils down to convolution with a box kernel. Using the sum of three of these mipmapped sampling results of different box kernel size, Bouthors et al. assumed to have approximated  $D$  adequately.

Based on the notion that the cloud area under the collector is of the highest importance, Bouthors et al. claimed that the corresponding best-fitting plane-parallel slab is defined by a point  $\vec{m}$ , a normal  $\vec{n}$  and a slab thickness  $t$ , where  $\vec{m}$  is obtained by taking the mipmapped sampling result of the front depth map,  $\vec{n}$  equals the mipmapped sampling result of the normal map, and  $t$  is the difference between the mipmapped sampling results of the front and back depth maps.

Figure 3.8 illustrates this approach, showing three distinct, presumably correct collector areas on the same cloud, defined by their centers  $\vec{c}_1$ ,  $\vec{c}_2$  and  $\vec{c}_3$ , that each correspond to some viewpoint that is not shown in the image. Sampling the normal map yields  $\vec{n}_1$ ,  $\vec{n}_2$  and  $\vec{n}_3$ , each denoting the normal of the corresponding points on the collector area according to a Gaussian distribution. The points  $\vec{m}_1$ ,  $\vec{m}_2$  and  $\vec{m}_3$  are obtained by sampling the front depth map in a similar fashion. The resulting distance to the collector area is then used to define  $\vec{m}$ .

The collector area defined by  $\vec{c}_1$  results in  $\vec{m}_1$  lying much deeper in the cloud volume, due to the fact that the average point on the cloud surface under the collector area lies much further from the sun than  $\vec{c}_1$  – admitted,  $\vec{m}_1$  is not the average but rather the integral over all these points weighted with a Gaussian distribution, yet the result is similar. For collector  $\vec{c}_2$  it is the other way around, since it lies in a concavity, yet is large enough for the surrounding cloud surface to influence  $\vec{m}_2$ . Finally,  $\vec{c}_3$  denotes a case where there is no high variance under the collector area, so  $\vec{m}_3$  lies relatively close to  $\vec{c}_3$ .

Similarly, the normal is dependent on all the normals of the cloud surface under the collector area, as reflected in Figure 3.8. Finally, the thickness is obtained by sampling the back depth map in the same way to obtain an unnamed point opposite of  $\vec{m}$  (the other red dot in Figure 3.8), where the distance between the two is used to define  $t$ . Note that the ray passing through this unnamed point,  $\vec{m}_i$  and  $\vec{c}_i$ , lies parallel to the sun direction  $\vec{s}_L$ . With these parameters, the plane-parallel slab can be described.

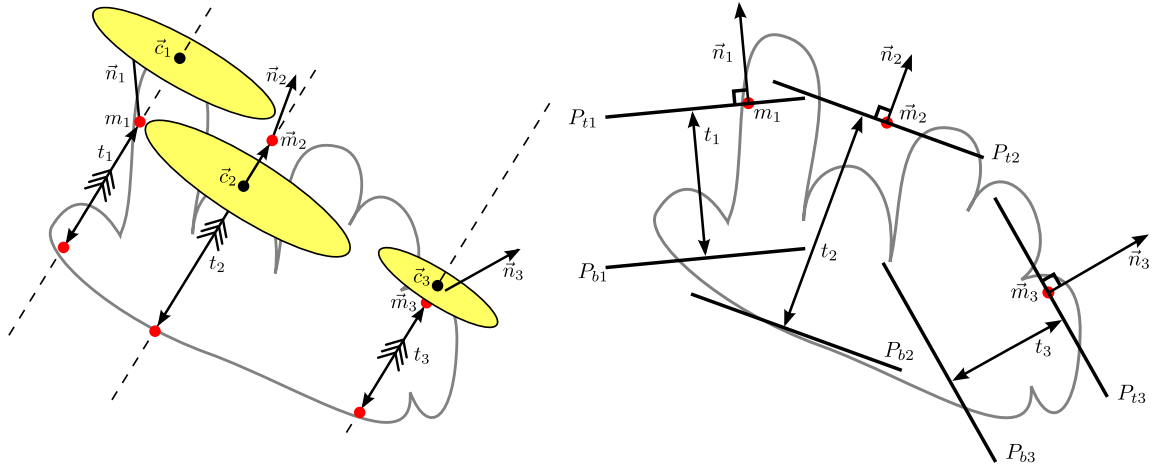
(a) Finding  $\vec{n}$ ,  $\vec{m}$  and  $t$ .(b) Fitting a slab  $(P_t, P_b)$ .

Figure 3.8: Fitting a plane-parallel slab to a collector area for three instances.

Finally, before we can use the light transport model, we still need some input parameters, which can easily be obtained from the information that we already have. The view direction  $\vec{s}_V$  is the vector from the viewpoint  $\vec{p}$  to the camera position, and the light direction  $\vec{s}_L$  is unchangeably the vector pointing from the sun position to the center of the cloud. The viewpoint depth  $d$  is obtained by projecting the difference between  $\vec{p}$  and  $\vec{m}$ , on the normal  $\vec{n}$ , as in equation (3.55). Then, the view altitude  $\theta_V$  can be obtained by using the dot product of the view direction and normal, the inverse cosine of which is the angle between them. Subsequently subtracting this number from  $\pi$  as in equation (3.56), we get  $\theta_V$  as in Figure 3.6. Similarly, we obtain the light angle  $\theta_L$  using the negative  $\vec{s}_L$ , this time without the subtraction, shown in equation (3.57). Finally, the view azimuth  $\phi_V$  can be obtained by projecting the view and inverse light direction vectors on the slab plane, normalizing these projections, and again taking



the inverse cosine of their dot product as in equation (3.58). We subtract this from  $\pi$  and correct the result by using the sign function on the  $\vec{x}$  coordinate axis projected on the view direction, so that we get  $\phi_V$  exactly as in Figure 3.6.

$$d = (\vec{m} - \vec{p}) \cdot \vec{n} \quad (3.55)$$

$$\theta_V = \pi - \arccos(\vec{s}_V \cdot \vec{n}) \quad (3.56)$$

$$\theta_L = \arccos(-\vec{s}_L \cdot \vec{n}) \quad (3.57)$$

$$\phi_V = \left( \pi - \arccos \left( \frac{\vec{s}_V - \vec{n}(\vec{s}_V \cdot \vec{n})}{\|\vec{s}_V - \vec{n}(\vec{s}_V \cdot \vec{n})\|} \cdot \frac{-\vec{s}_L - \vec{n}(-\vec{s}_L \cdot \vec{n})}{\|-\vec{s}_L - \vec{n}(-\vec{s}_L \cdot \vec{n})\|} \right) \right) \cdot \text{sgn} \left( \frac{\vec{n} \times (-\vec{s}_L - \vec{n}(-\vec{s}_L \cdot \vec{n}))}{\|\vec{n} \times (-\vec{s}_L - \vec{n}(-\vec{s}_L \cdot \vec{n}))\|} \cdot \vec{s}_V \right) \quad (3.58)$$

Finally, we can use these parameters to obtain the multiple scattering component. However, one problem still remains.

### Finding the collector area

We have assumed that the correct collector area for a viewpoint  $\vec{p}$  is somehow known to us. However, this is not the case. The problem lies in finding this collector area, which would be easy if we had the best-fitting plane-parallel slab, through the phenomenological collector model. However, to find this slab, we first need the correct collector area.

This dilemma can be reduced to the mathematical problem of finding the *fixed point* of a function  $f(x)$ . A value  $x^*$  denotes a fixed point when  $f(x^*) = x^*$ , that is, the locations where a function intersects the line  $y = x$  are fixed points. As an analogy to our collector-slab problem, we can view  $f(x)$  as the function denoting the phenomenological collector model discussed in section 3.1.4, while  $x$  is the plane-parallel slab that serves as input for this model, and the result  $y$  is the collector area. If we can manage to find the fixed point  $x^*$ , we know that the slab corresponds to the collector area resulting from the model, thus we have found the best-fitting slab, and can continue to compute the multiple scattering component.

Many techniques exist for finding the fixed point. One well-known approach that can be used to obtain an *attractive fixed point* is by iterating the function as in equation (3.59), for some initial  $x_0$ , so that we can define  $x^*$  as equation (3.60).

$$x_{i+1} = f(x_i) \quad (3.59)$$

$$x^* = \lim_{i \rightarrow \infty} x_i \quad (3.60)$$

This iterative process is illustrated in Figure 3.9, which shows a good example of a function containing precisely one fixed point, that is attractive as well, and for which the initial  $x_0$  can be any value in the domain of the function. However, not all functions are so well fit for this approach. Consider for instance  $f(x) = x + 1$ , which has no fixed point at all, or  $f(x) = 2x$ , for which any  $x_0 \neq 0$ , the method diverges, never finding the fixed point  $x^* = 0$ . Also, there is the problem of multiple fixed points, e.g., for  $f(x) = x^2$ , where the fixed point  $x^* = 0$  can only be found if  $x_0 < 1$ , while  $x^* = 1$  requires  $x_0 = 1$ . Finally, it is possible to get stuck in so-called *periodic points* using this technique, that is,  $x_i = x_{i+k}$  with  $k > 1$ . For example, consider  $f(x) = -\frac{1}{x}$ , with some  $x_i = 1$ , which results in  $x_{i+1} = -1$  and  $x_{i+2} = 1$  again.

While techniques exist that can overcome some of these problems, they require information like the derivate of the function  $f(x)$ . These characteristics can not be given for the phenomenological model, or are too expensive to compute for this application. Bouthors et al. thus opted for the iterative approach, justifying their choice with the assumption that the collector area always exists and is unique. For the initial value, they chose the viewpoint for the collector center  $\vec{c}_0 = \vec{p}$  and a collector size corresponding to the size of the whole cloud by taking a very high  $\sigma_0$ . By sampling the normal and depth maps at the collector area, the plane-parallel slab is fitted as described in the previous section, resulting in the input

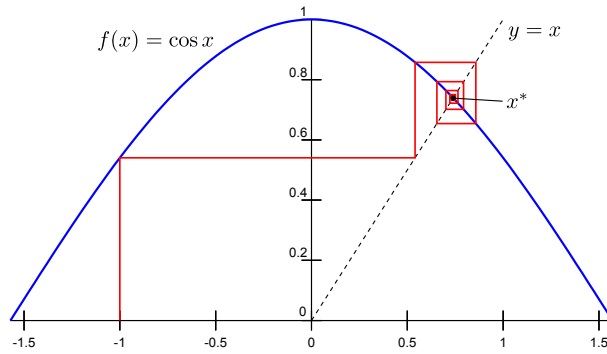



Figure 3.9: Application of the iterative fixed point finding algorithm on a cosine function.  Tinc-torius / Wikimedia Commons.

parameters for the phenomenological collector model. From this, we obtain a new collector area, which we call  $(\vec{c}_1, \sigma_1)$ , and this process is repeated until we converge, analogous to the fixed point method. This iterative process is illustrated in Figures 3.10 and 3.11.

Bouthors et al. noticed that the algorithm could get trapped in periodic points, which is why they introduced constraints on both the collector center  $\vec{c}$  and its radius  $\sigma$  to prevent taking large steps, as per equations (3.61) and (3.62).

$$\|\vec{c}_{i+1} - \vec{c}_i\| < \frac{\sigma_i}{2} \quad (3.61)$$

$$|\sigma_{i+1} - \sigma_i| < \frac{\sigma_i}{2} \quad (3.62)$$

Finally, the convergence criterion was defined so that no unnecessary iterations are considered once the correct collector area is already adequately approximated, so that we have the criterion as in equation (3.63).

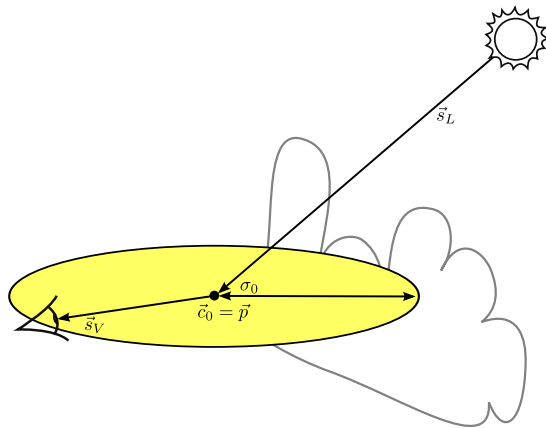
$$\|\vec{c}_{i+1} - \vec{c}_i\| < \frac{\sigma_i}{10} \quad (3.63)$$

Bouthors et al. found that using this criterion, most cases would converge within ten iterations, and those that did not were claimed to have a negligible visual effect on the result. In any case, using this algorithm, the correct collector area can be approximated, and using the corresponding plane-parallel slab as input for the phenomenological light transport model, the multiple scattering component can be obtained in real-time.

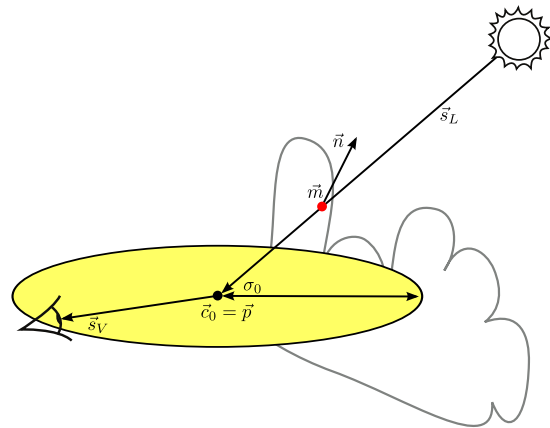
### The chopped-peak model

Before advancing to combining the opacity, single and multiple scattering components that we now have, it should be noted that Bouthors et al. did not use the Mie phase function as described in section 2.1.7. Instead, they made use of the *chopped-peak model*, which does what it says on the tin: the narrow forward peak of the Mie phase function is *chopped*, so that all light scattering within this peak is approximated as in equation (3.64), which is based on the work presented in [Len85]. The chopped phase function  $\tilde{P}(\Theta)$  is equal to the Mie phase function  $P(\Theta)$  for scattering angles  $\Theta \geq 8^\circ$ , while smaller scattering angles are approximated.

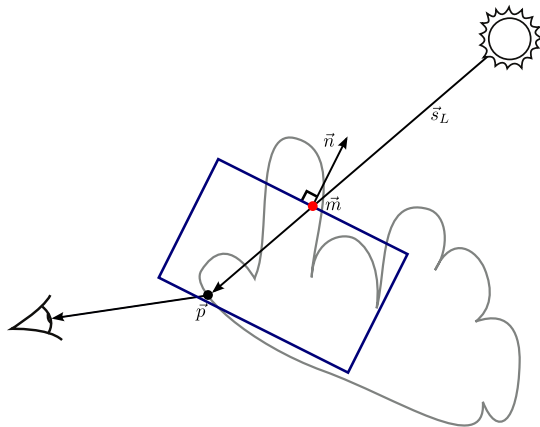
$$\tilde{P}(\Theta) = \begin{cases} P(8^\circ) & \text{if } \Theta < 8^\circ \\ P(\Theta) & \text{if } \Theta \geq 8^\circ \end{cases} \quad (3.64)$$



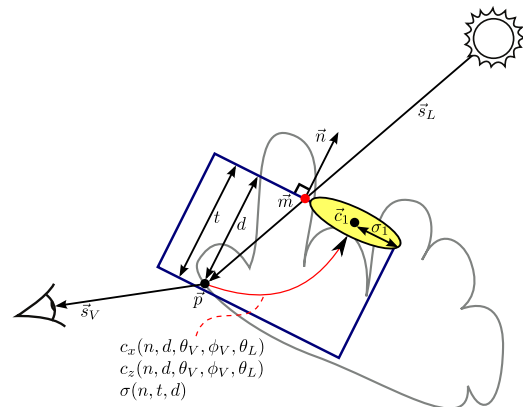
(a) Iteration one. Initial values  $(\vec{c}_0, \sigma_0)$  are chosen.



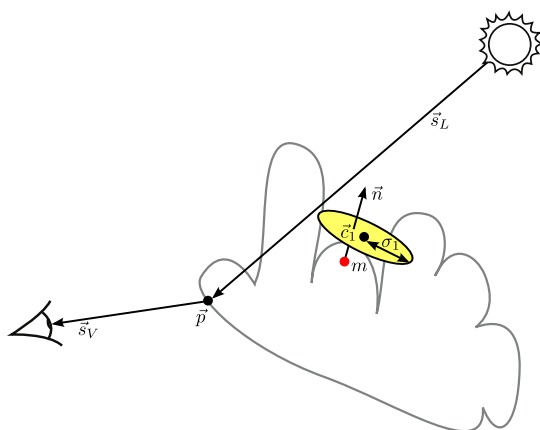
(b) The normal and front depth maps are sampled over the collector area to obtain  $\vec{n}$  and  $\vec{m}$  as in Figure 3.8.



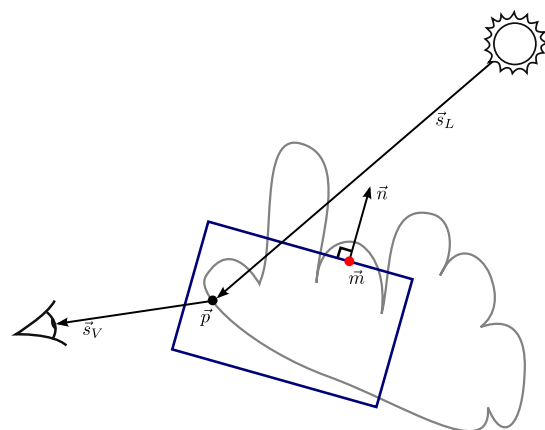
(c) Sampling the back depth map, we can now define the plane-parallel slab.



(d) Iteration two. Slab thickness  $t$  and viewpoint depth  $d$  are shown here. We apply the phenomenological collector model to obtain the new collector  $(\vec{c}_1, \sigma_1)$ .

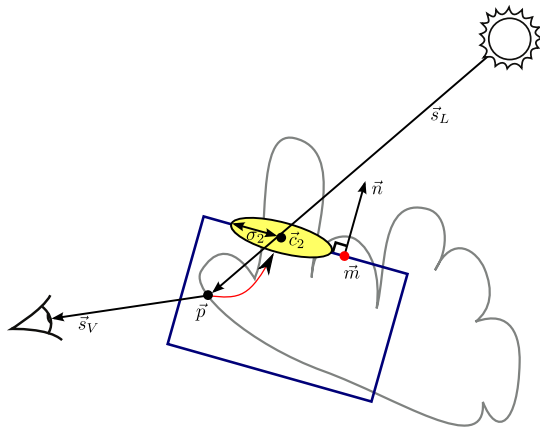


(e) With this new collector area, we repeat the process by finding  $\vec{n}$  and  $\vec{m}$  again.

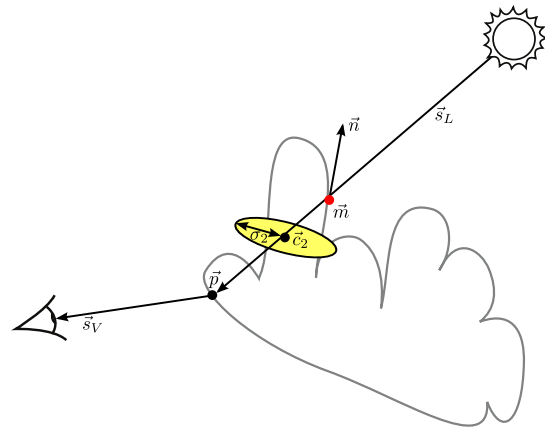


(f) As before, the slab is fitted.

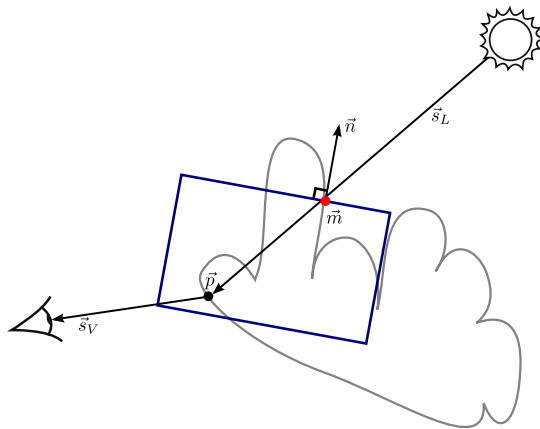
Figure 3.10: The first steps in finding the correct collector area through the iterative algorithm.



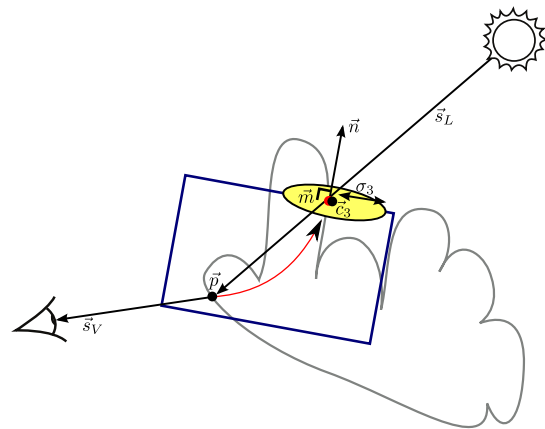
(a) Iteration three. The phenomenological functions are again used to obtain the next collector ( $\vec{c}_2, \sigma_2$ ).



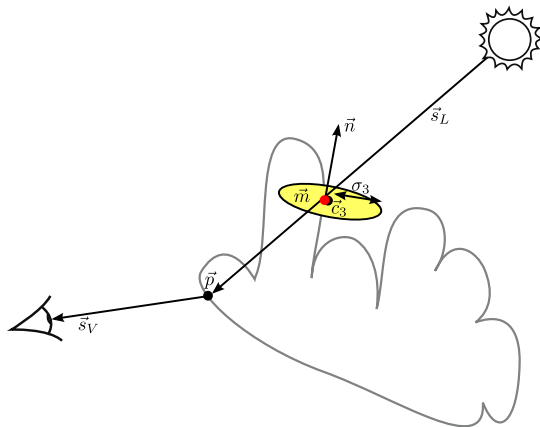
(b) We obtain  $\vec{n}$  and  $\vec{m}$  as before.



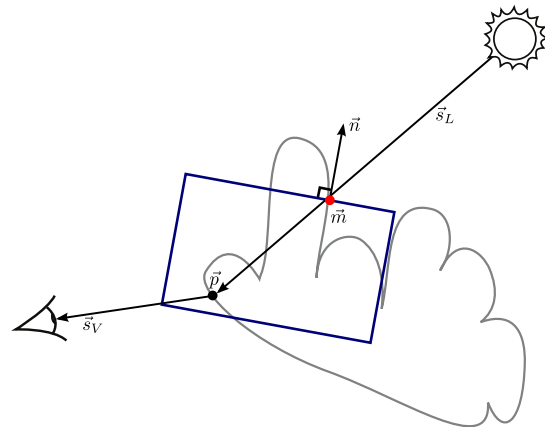
(c) A slab is fitted again.



(d) Iteration four. The collector area resulting from the model is shown to lie very close to  $\vec{m}$ .



(e) Sampling at this collector area again, we find virtually equal values for  $\vec{n}$  and  $\vec{m}$  as in the previous iteration.



(f) Fitting the slab shows that convergence has occurred, and we use the parameters corresponding to this slab as input for the phenomenological light transport model.

Figure 3.11: The last steps in finding the correct collector area through the iterative algorithm.

Bouthors et al. proved that this model works well as an approximation for the multiple scattering component, however, for single scattering, we need to recover from this removed peak. To do so, we consider the removed peak phase function,  $P_p(\Theta)$ , which is given in equation (3.65) and is available on Bouthors' thesis website.

$$P_p(\Theta) = P(\Theta) - \tilde{P}(\Theta) \quad (3.65)$$

Bouthors et al. recovered from the removed peak through an additional light transport function,  $T_p(\vec{p}_a, \vec{s}_V)$  shown in equation (3.66). Here, the optical thickness of the chopped-peak model,  $\tau_c$ , is this model's extinction coefficient  $k_c$  integrated over the view direction  $\vec{s}_V$ . However,  $k_c$ , as shown in (3.67), is simply the weight of the removed peak phase function,  $f$ , multiplied by the regular extinction coefficient. This weight  $f$  is constant, as becomes apparent from its definition in equation (3.68), which enables us to bring it out of the integration, allowing for a rewrite of (3.66), resulting in equation (3.69). Note that Bouthors et al. used  $f = 0.43$  throughout their thesis.

$$T_p(\vec{p}_a, \vec{s}_V) = P_p(\Theta)(e^{-\tau_c} - e^{-\tau}) \quad (3.66)$$

$$k_c(\vec{p}) = f\rho(\vec{p})\sigma_e \quad (3.67)$$

$$f = 2 \int_{-\pi}^{\pi} P_p(\Theta) \sin \Theta d\Theta \quad (3.68)$$

$$T_p(\vec{p}_a, \vec{s}_V) = P_p(\Theta)(e^{-f\tau} - e^{-\tau}) \quad (3.69)$$

Now, we have all components and can continue to incorporate them to determine the final pixel color. Note that the actual reason why Bouthors et al. made use of this model was to speed up the experiments pertaining to their study of light transport in slabs, which took several weeks on a supercomputer. Despite seeming like a rather useless optimization – after all, it is only a nuisance when applying the phenomenological model and perhaps even slightly slows it down, just for a more efficient preprocessing step – we can sympathize due to the difficulty in obtaining free time slots to make use of a supercomputer.

### 3.1.5 Combining the components

Finally, we can obtain the pixel color of the cloud. For every pixel on the screen, we find the corresponding world position  $\vec{p}_a$  by sampling the depth map. The view ( $\vec{s}_V$ ) and light direction ( $\vec{s}_L$ ) are calculated using  $\vec{p}_a$  and the respective camera and sun position and the cloud center. Then, by sampling the back depth map that was rendered from the camera viewpoint at the pixel location, we obtain  $\vec{p}_b$ , the point on the cloud surface that lies furthest from the camera in the view direction.

With these two, we obtain the opacity  $\alpha(\vec{p}_a, \vec{p}_b)$  as per section 3.1.2, and transparent pixels are blended correctly, which will be discussed in section 3.4.2. Next, the single scattering component  $T_1(\vec{p}_a, \vec{s}_V)$  is computed by using  $\vec{p}_a$  and  $\vec{p}_b$  as well, following section 3.1.2. Simultaneously, we recover from the removed peak using  $T_p(\vec{p}_a, \vec{s}_V)$  corresponding to section 3.1.4.

Then, the iterative algorithm of section 3.1.4 is applied, and the results are used as input for the phenomenological model to obtain  $C(n, t, d, \theta_V, \phi_V, \theta_L)$  for every  $n \in (1, 7)$ . Finally, we compute the total light transport  $T(\vec{p}_a, \vec{s}_V)$  by summing the components, as shown in equation (3.70).

$$T(\vec{p}_a, \vec{s}_V) = T_1(\vec{p}_a, \vec{s}_V) + T_p(\vec{p}_a, \vec{s}_V) + \sum_{n=1}^7 C(n, t, d, \theta_V, \phi_V, \theta_L) \quad (3.70)$$

This is then multiplied by the solar illuminance to obtain the final luminance. From a more practical point of view, we simply multiply the color of the sun with  $T(\vec{p}_a, \vec{s}_V)$  to obtain the pixel color. The opacity of the pixel is set equal to  $\alpha(\vec{p}_a, \vec{p}_b)$ , which allows for correct blending with the background's pixel colors as well as other cloud pixels. To obtain the sun color, Bouthors et al. made use of a well-known real-time model for sky color, devised in 1999 by Preetham et al. [PSS99]. As mentioned in chapter 1, we employ

an improved model proposed by Hosek and Wilkie in 2012 [HW12] in our implementation, which has been slightly enhanced as per [Kol12].

Bouthors et al. further included a blue light source above the cloud, denoting sky light, for which only  $C(7, t, d, \theta_V, \phi_V, \theta_L)$  was computed to attribute to the light transport. Naturally, in reality, the sky lights a cloud from all directions, which is unfeasible for real-time implementation. Due to this, they simply used a point light source as with the sun, and only considered diffuse light transport, as modeled by  $n = 7$ . We do not use this approach, as we found that the color of the sun resulting from Hosek and Wilkie’s skylight model was already slightly blue. However, the method of Bouthors et al. could easily be included in our implementation as well.

Furthermore, they added a brown light source below the cloud to model reflected light from the ground, again only computing diffuse light transport through their phenomenological model. They simply added the colors caused by these two light sources to the contribution of the sun to obtain the final pixel color.

## 3.2 Drawbacks

### 3.2.1 Model

The triangle mesh model combined with the sparse voxel octree approach to represent the cloud shape that was used by Bouthors et al. has some drawbacks. For instance, despite using the cloud modeling approach they discussed in [BN04], generating these models is still a relatively time-consuming process, even when fully automated. Additionally, to generate arbitrary cloud shapes, artists would need to take great care to model a highly detailed three-dimensional mesh.

Related to this is the issue of the construction of the sparse voxel octree. As mentioned in section 3.1.1, we found that this preprocessing step puts significant strain on the computational resources, which results in a slow initialization process every time a different cloud shape is loaded.

Moreover, while Bouthors et al. claimed that their model allows for animated clouds, this is by no means an easy feat. It is true that their octree can be efficiently updated during runtime as in [CNLE09], yet creating an animated triangle mesh that realistically simulates atmospheric fluid dynamics is an arduous task for any animator. On top of that, the distance values that are used to update the octree given the next animation frame of the mesh, are not easily computed in real-time.

Furthermore, the evaluation of Perlin noise on the GPU as applied by Bouthors et al. is rather costly. Using five octaves, they needed to perform sixty texture reads every time Perlin noise was evaluated. They made use of *simplex noise*, designed by Perlin in 2001 as an optimization of classic Perlin noise, a description of which can be found in [Gus05]. Still, it forms a bottleneck in their approach, as the noise evaluation is invoked thousands of times per pixel.

Bouthors et al. showed that if they did not use the hypertexture to augment the triangle mesh with details on the edges, the frame rates went up from two to ten frames per second. A factor five slowdown seems rather harsh given how little it adds to the realism of the cloud. Recalling Figure 3.2, we can see that *both* images seem to have rather overly well-defined boundaries. We believe that with a thicker heterogeneous boundary, this problem could be solved, yet this would slow down the frame rate even more. At this rate, however, the hypertexture does not have a large enough effect to justify its impact factor on the frame rate in our opinion.

To come back to the issue of the thin boundary, it seems to us that the cloud renderings produced by Bouthors et al. contain excessively sharp boundaries. It is certainly true that these boundaries exist on cumuliform clouds, however, they are mixed with much wispy cloud edges as well. Consider for instance the cumuliform cloud shown in Figure 3.12, which has wispy edges that can never be convincingly visualized with a heterogeneous layer as thin as the one used by Bouthors et al.

### 3.2.2 Opacity and single scattering

The proposition made by Bouthors et al. to compute the opacity and single scattering seems like a sound method. However, there is one main drawback pertaining to the use of depth maps for the volume ray marching in single scattering computation: concavities are not taken into account correctly.





Figure 3.12: Cumiform cloud with very wispy edges. © Kate Haskell.

Non-convex meshes cause the algorithm to march through empty air, which normally would not be a problem, as the density that is sampled at these locations would logically equal to zero. However, Bouthors et al. used segments to march the cloud volume, which can cause problems with concavities for the single scattering computation, where segmentation points are distributed logarithmically instead of based on the sparse voxel octree, like for opacity. Three instances of this problem are shown in the first three images of Figure 3.13.

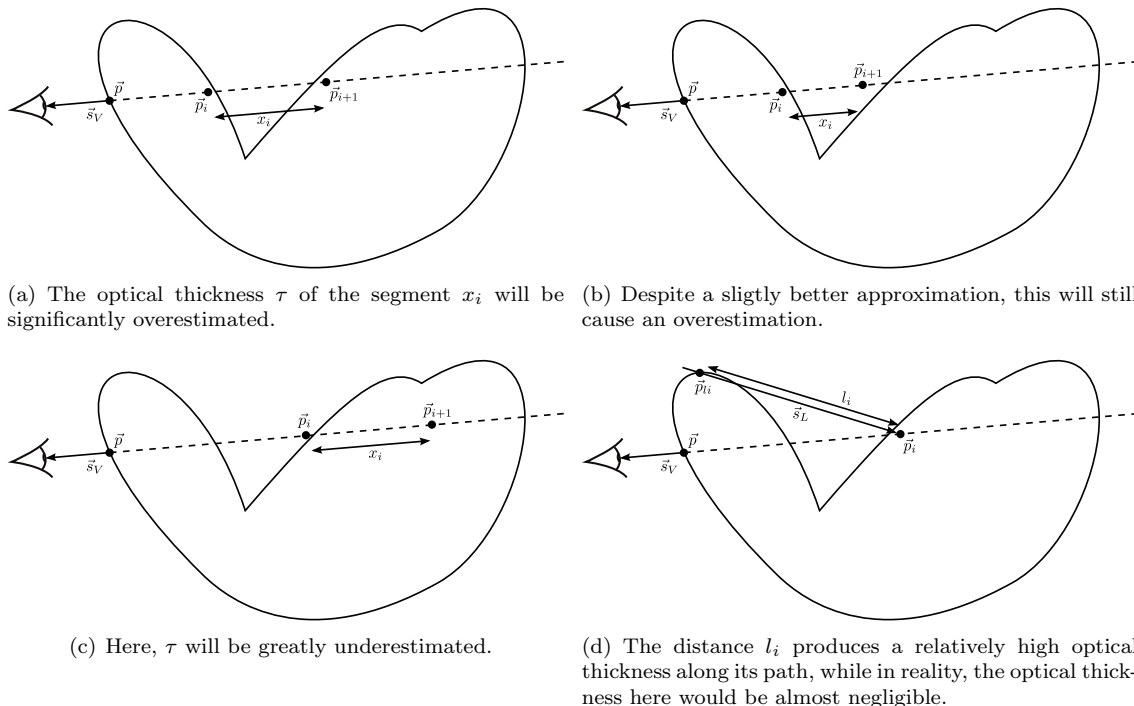


Figure 3.13: Approximation issues caused by concavities and the use of depth maps combined with segment-based volume marching.

An additional issue appears when sampling the sun depth map, as shown in the last image of Figure 3.13. Bouthors et al. proposed the use of depth peeling for future work to overcome this problem, yet

this technique, while viable, will slow the method down considerably, since multiple passes need to be performed, which is computationally expensive due to the highly detailed triangle mesh.

Besides the depth map issues, the renderings produced by Bouthors et al. sometimes contain visual artifacts in the form of dark pixels along the edges on the cloud, as highlighted in Figure 3.14. They claimed these to be purely implementation-related, and it is true that we did not encounter this problem at all in our implementation. Despite a lack of explanation by the authors about what caused these artifacts, we suspect it was due to either bugs in the opacity calculation, or an incorrect blending method.



Figure 3.14: Dark pixels are visible on the cloud edges in the highlighted areas. These visual artifacts are probably caused by incorrect opacity calculation or blending issues. Image by Bouthors [Bou08].

### 3.2.3 Multiple scattering

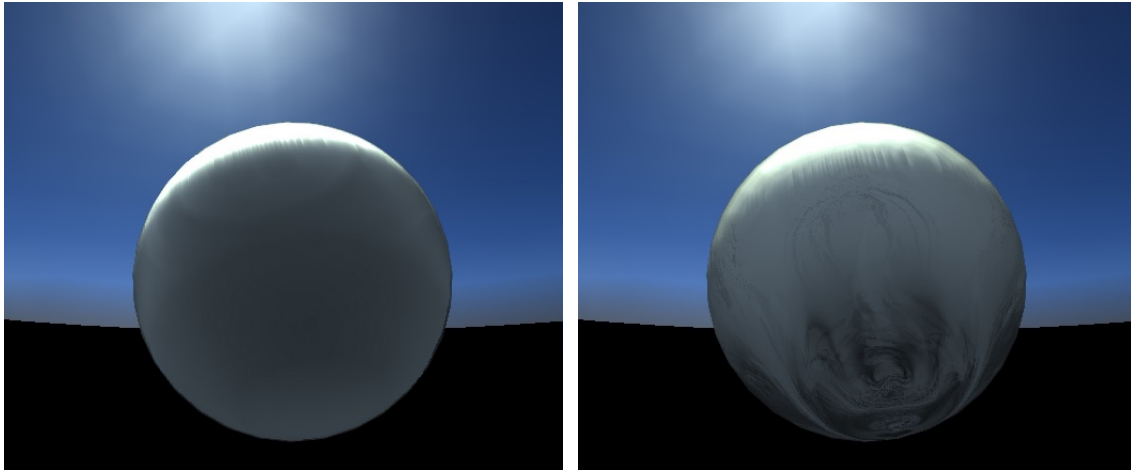
The issue with depth maps plays a role in multiple scattering as well. When fitting a plane-parallel slab to a cloud volume, the thickness of this slab may be overestimated due to concavities, depending on the location of the collector area relative to the sun position, resulting in an understated transmittance and exaggerated reflectance.

However, the main problems with multiple scattering lie in the iterative algorithm described in section 3.1.4. For one, it does not take into account the size of the cloud, which can result in collector areas lying completely beyond the cloud surface. When sampling the depth and normal maps with such a collector area, the results equal zero. To overcome this problem, Bouthors et al. chose to increase the collector area until a result was found. Additionally, to prevent taking into account sample values that do not lie on the cloud surface, they stored an alpha value in the maps as well, where  $\alpha = 1$  denotes cloud surface, and  $\alpha = 0$  defines background pixels. When sampling a mipmap, this value is averaged, and dividing the color by the alpha value produces the color value as if only the cloud surface pixels were taken into account, as described in [DS03].

Still, this is not an entirely correct approach. After all, the area of the cloud surface that is considered could be smaller or greater than the actual collector area that was obtained from the phenomenological model, depending how the area is increased. Even when the collector area only partly lies outside the cloud surface, the considered area will not equal the actual area. It is difficult to determine how this visually influences the outcome, yet it will play a big role if, for instance, the actual collector area is very

large, lying so far away from the cloud surface that only a tiny portion of the surface is sampled, resulting in a localized approximation while in reality, the spatial spreading was significant.

A problem that definitely has a high visual impact is the appearance of discontinuities. Neighboring points on the cloud model may converge to completely different slab solutions due to the phenomenological model for finding the collector area. This is not much of a problem after one iteration, yet every additional repetition causes more noise due to nearby points producing distinct collector locations. According to Bouthors et al., the discontinuities are especially apparent on spheres, yet we found it to be noticeable for any cloud shape. Figure 3.15 shows how badly noise degrades the results. Surprisingly, the results of one iteration seem correct already.



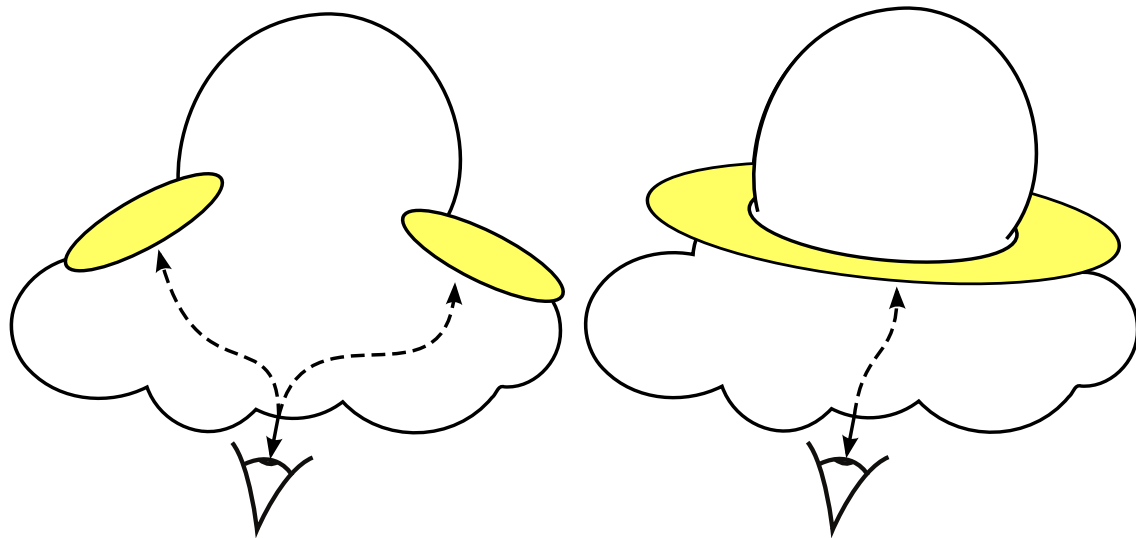
(a) The first iteration using a large collector area produces a noise-free image. This is before the phenomenological collector model is invoked.

(b) The result after up to ten iterations (depending on whether the convergence criterion was satisfied) contains a significant amount of discontinuities, resulting in noisy, swirling patterns.

Figure 3.15: Noise is caused by neighboring points converging to very distinct results, which adds up over several iterations of the collector finding algorithm.

Closely related to this are the wrongful assumptions that the collector is always unique and has the shape of a disc. It may very well be possible that multiple disc-shaped collector areas are present on the cloud surface given a certain viewpoint, as shown in the left image of Figure 3.16. The algorithm as presented by Bouthors et al. will probably return one of either, which can cause further discontinuities due to neighboring points or view angles returning a different collector area. It is true that the *actual* collector area always exists and is unique, yet since Bouthors et al. approximated it by a disc-shaped area, the uniqueness is not guaranteed.

Furthermore, in the case of Figure 3.16, it seems more likely that the collector area will have the shape of a ring, as displayed in the right image. For arbitrary clouds, it seems unlikely indeed that the assumption of a disc-shaped collector area is generally correct, even though it is an excellent approximation for a plane-parallel slab. However, like Bouthors et al., we lacked time to perform a study of light transport in arbitrary clouds to investigate the actual shape of the collector areas. The visual effects of this issue are hard to determine, as a whole new model would have to be devised making use of differently shaped collector areas for comparison.



(a) Multiple disconnected collector areas may be present given a certain view point. (b) The collector area may actually take forms other than a disc, such as a ring around a protruding cloud top.

Figure 3.16: Issues with the assumption that there always is a unique disc-shaped collector area.

## 3.3 Hypotheses

### 3.3.1 Model

The main drawbacks pertaining to the cloud model are the time-consuming preprocessing step consisting of modeling the cloud shape and constructing a corresponding sparse voxel octree, the overly well-defined edges caused by the limited memory in which the octree is stored, the relative low frame rates brought about by the highly detailed triangle mesh as well as the octree, and finally, the difficulty in animating the cloud. Additionally, the Perlin noise evaluation forms a bottleneck of Bouthors' approach.

We propose a different model that solves most of these issues and present another method of noise evaluation. We represent the cloud shape through the use of a set of ellipsoids, as originally proposed by Gardner [Gar85]. We show later that only a small number of ellipsoids is needed to faithfully represent a cloud, thanks to noise augmentation. In most of our experiments, we worked with only eight ellipsoids, for which it is easy and quick to define their center coordinates and semi-principal axes (radii). This way, artists can still control the general shape of the cloud by using some well-placed ellipsoids. Although we did not have time for this, a user-friendly system could easily be implemented in which the location and radii of the ellipsoids can be adapted in real-time to create a cloud of any shape in a matter of seconds.

An additional advantage of this approach is that it does not require the signed distance to be stored on an octree. In fact, the distance from any point within an ellipsoid to its boundary can be efficiently computed on the fly, as will be shown in section 3.4.4. Omitting the octree frees up a large chunk of memory, removes the need of a substantial preprocessing step, and significantly ups the frame rates.

Thanks to the ease with which the distance to the edges can be obtained, we can now define a much thicker heterogeneous boundary size to better simulate the wispy behavior in clouds.

Furthermore, this model allows for a much easier way to animate the clouds. As we are using a thicker boundary, the noise plays a much bigger role in the appearance of the cloud, so that we can apply the technique proposed by Schpok et al. in [SSEH03] to modify noise texture coordinates to simulate fluid dynamics. Additionally, heterogeneous boundary thickness and ellipsoid position and size can be varied over time to create movement. Allowing artists to key-frame such animations, realistic results could be created and saved, using a negligible amount of memory.

Finally, we propose the use of a three-dimensional noise texture that is generated as a preprocessing

step instead of evaluating Perlin noise on the fly. Using mipmapping, we produce fractal noise to better simulate a cloud texture. The advantage of this is that we only need one texture read for every octave, instead of the twelve reads that are required when using simplex noise. Furthermore, the evaluation is much faster, as the sampling results simply need to be multiplied by a scalar and summed.

We believe that this revised cloud model eases the artists' jobs in generating clouds, while also speeding up the frame rates significantly. Moreover, this new model allows for easy animation, adding to the realism of the visualization.

### 3.3.2 Opacity and single scattering

Using ellipsoids, we can also overcome most of the problems of depth maps and concavities. Instead of marching over the entire volume, we march over each ellipsoid separately, using the intersection points between the ellipsoid and the ray in the view direction as per section 2.1.2. As ellipsoids are by definition convex, there are no concavities. This way, we ensure that the spaces between ellipsoids are not taken into account, which solves the issues presented in the first three images of Figure 3.13. However, the problem with sampling the sun depth map still persist, yet we assume its visual effects to be negligible. This is related to the assumption that the extinction coefficient is constant along the sun direction discussed in section 3.1.2, and similarly would require significant computational resources to solve. Note that using intersection points means we can omit the depth maps that are rendered from the camera position as well, reducing the rendering pipeline with two passes.

Furthermore, we propose to compute the opacity and single scattering component in one go, unlike Bouthors et al., who performed the volume marching separately. While this speeds up the process, we do need to use the same segmentation for both components. As we prefer to stay true to the logarithmically spaced segments for single scattering, opacity is computed in the same way. Luckily, logarithmic spacing does not have an important visual effect on the opacity when compared to uniform spacing, as shown in Figure 3.17. Additionally, we believe that as little as five segments suffice for marching the ellipsoid. The heterogeneous boundary looks unrealistic when too few segments are used and the camera rotates around the cloud, yet we found five segments to produce acceptable results, given our aim for real-time performance.

Our hypothesis is that this new method provides for a significantly faster and more accurate opacity and single scattering calculation. We compute both in one volume march and account for concavities, preventing single scattering artifacts to emerge due to scenarios such as in Figure 3.13.

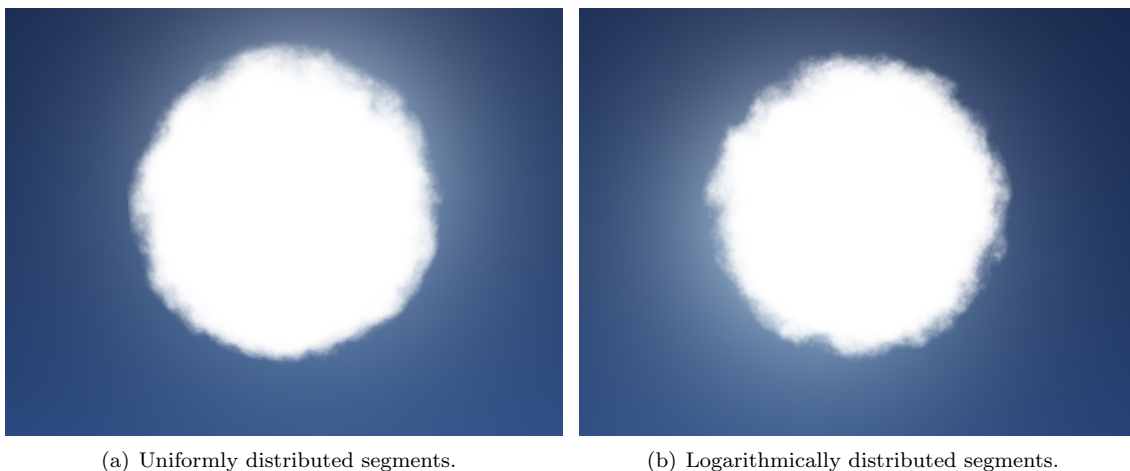


Figure 3.17: Difference between uniformly and logarithmically spaced segments regarding opacity computation. The only distinction – besides the randomness of the noise – is that uniformly distributed segments result in a slightly higher average opacity around the edges, due to the fact that logarithmic sampling favors the relatively sparse areas near the first intersection point.

### 3.3.3 Multiple scattering

The main problem with multiple scattering lies within the rather slow iterative algorithm and the discontinuities caused by the phenomenological collector model. The light transport model on the other hand seems to work quite well. For this reason, we opt to simply stick to the plane-parallel slab that is fitted on the cloud volume in the first iteration, and immediately compute the light transport. This absence of the collector model solves most of the problems that we have discussed, i.e., locations outside the cloud surface will no longer be sampled, neighboring points on the cloud surface result in similar plane-parallel slabs to create smooth transitions, and we no longer have the problem of arbitrarily-shaped collector areas.

However, for this approach to be valid, we need the initial fitted slab to be a good approximation of the cloud surface, at least in comparison to the slab found by the iterative algorithm. We hypothesize that this method is a significantly faster and more robust option than the iterative algorithm, while still producing adequate results.

We test our hypotheses in chapter 4, and prove them as far as possible.

## 3.4 Implementation notes

This section features a more detailed description of our methods, with implementation-related information to aid future researchers. Only our model will be discussed; that is, not all components of the original work by Bouthors et al. are examined, such as the octree implementation and the collector-finding algorithm. Our model, like the one by Bouthors et al., is implemented in OpenGL using GLSL for the shaders.

### 3.4.1 OpenGL and GPU programming

The current standard for OpenGL implementations relies on programmable hardware functionality. In the past, programmers would have very little influence on the actions performed by the GPU, except for configuring parts of the hardware. Despite the fact that GLSL, the current principle shading language for OpenGL, is in existence since 2004, most online sources are still based on this old model, called the fixed function pipeline, which may confuse programmers. These days, programmers are expected to fully handle the functionality of the GPU, including world-space to screen-space transformation – for which several useful libraries such as GLFW exist.

One advantage of programmable functionality is that we can now efficiently invoke customized, complicated logic on a per-pixel basis, which is crucial for our method of cloud rendering as well. We make use of a vertex shader to perform operations on vertices, and a fragment or pixel shader to execute computations on pixels.

#### Vertex shader

The primary task of the vertex shader is to convert the world position of all vertices in the scene to the corresponding screen space coordinates. This is achieved by defining the view and projection matrices on the CPU, subsequently sending them to the vertex shader as uniforms – global variables that are equal for all invocations of the shader program, e.g., the projection matrix will be the same for *all* vertices until it is modified on the CPU and sent to the GPU again.

Note that when we are in a pass for the depth and normal map computation from the sun's point of view, an orthogonal projection matrix is sent, while otherwise, we make use of a perspective projection matrix.

Besides all this, we also store the world position and the normal of the vertex in a variable to send on to the pixel shader. Note that both of these values can be obtained in the vertex shader using the `gl_Vertex` and `gl_Normal` identifiers.



### Pixel shader

The pixel shader is tasked with writing the color of a fragment – a segment corresponding to the area of a pixel – to the frame buffer. Multiple fragments may be considered for the same pixel, depending on the scene. For instance, when one object is behind the other, yet rendered first, at least two fragments are considered for a pixel for which the ray intersects both objects. The colors written to the frame buffer by a pixel shader are blended, or one fragment is discarded, depending on the depth testing and the opacity of the objects.

Note that variables carried over from the vertex shader are linearly interpolated so that for fragments lying between vertices these variables are influenced by the neighboring vertices.

The pixel shader is where our cloud visualization method is implemented. First, however, we check the rendering mode – also a uniform variable set on the CPU – to see if we are constructing a depth map, normal map, or are performing the final cloud rendering. In the first case, we write the depth value to the frame buffer, which is obtained by reading the  $z$  value of the screen space coordinates generated by the vertex shader. Since it is defined in the range of  $[-1, 1]$ , and the frame buffer values are clamped to  $[0, 1]$ , we multiply by 0.5 and add 0.5. If we are generating a normal map, we obtain the normal from the variable created by the vertex shader and again convert it to the range of  $[0, 1]$ , where the three coordinates are written to the red, green and blue color channels of the frame buffer. Finally, in the case of cloud rendering, we invoke our model as implemented on the shader.

One limitation of the pixel shader is that for every fragment it is invoked for, it knows nothing about the neighboring fragments, which impedes some applications.

### Branching

The power of GPU programming lies in that the computation is spread over a large number of processors. However, this does require all invocations to finish concurrently; that is, the computation for every pixel should take the same amount of time. When there are branches in the code, e.g., caused by an `if` statement, depending on the branches, chances are several fragment programs need to wait for other invocations to finish, causing the corresponding processors to be idle. For this reason, branching should be avoided as much as possible, especially in the bottlenecks of an algorithm.

## 3.4.2 Blending and depth testing

### Depth testing

Besides the color buffer, which contains the colors of pixels that are rendered to the screen, there also is a depth buffer, which stores the depth of fragments in the scene. Depth testing is enabled by invoking the `glEnable(GL_DEPTH_TEST)` function, which causes all considered fragments to be tested against the value currently in the depth buffer, according to the depth function. Typically, this test consists in checking if the current fragment lies closer to the camera than the depth buffer value – if so, this new value is written to the depth buffer and the fragment color is blended with the frame buffer; if not, the fragment is discarded. However, for some applications, we are interested in the values that lie furthest away from the camera, in which case we want to use a different depth function. The depth function is set using `glDepthFunc(func)`, where, among other options, `func = GL_LESS` results in the closest points being considered, `func = GL_EQUAL` only considers fragments with equal depth to the depth buffer values, and `func = GL_GREATER` considers the points with the greatest depth, which is used, for instance, for constructing the back depth map from the sun's point of view.

Finally, it should be noted that besides enabling or disabling depth testing, we can also enable or disable *writing* to the depth buffer, by using `glDepthMask(GL_TRUE)` or `glDepthMask(GL_FALSE)`, respectively. This is especially useful for rendering transparent objects.

### Transparency

In OpenGL, transparency is enabled by making use of blending through calling the `glEnable(GL_BLEND)` function. That is, when fragment colors are written to the frame buffer, they are multiplied by the so-

called source factor, and added to the value that is already present in the frame buffer, which is multiplied by the destination factor. These factors are set using the OpenGL function `glBlendFunc(sourceFactor, destFactor)`. Typically, to allow for transparency, the source factor is set to the source alpha value, while the destination factor is set to one minus this value. Subsequently, opaque objects are rendered first with depth testing and writing enabled, so that they fill the frame buffer. Then, transparent objects are rendered in back-to-front order with depth testing enabled, yet writing disabled, causing them to blend with the opaque objects. Consider for instance a blue background, i.e. color  $(0, 0, 1)$ , that is written to the frame buffer. Then, a red object, i.e. color  $(1, 0, 0)$ , with opacity  $\alpha = 0.25$  is rendered, causing the resulting pixels to have a color of  $0.25 \cdot (1, 0, 0) + 0.75 \cdot (0, 0, 1) = (0.25, 0, 0.75)$ , resembling indigo.

### Our method

The problem with this is that we can not easily render the partly transparent ellipsoids in a back-to-front order. It is true that we know their positions, yet they often intersect, which causes this order to change depending on the pixel that is being considered. Due to this, we opt for handling the blending between ellipsoids in the pixel shader, where we have access to the ellipsoids and can compute the correct order on a per-pixel basis. The algorithm for this will be discussed in section 3.4.4. Note that the blending with the background is handled by OpenGL as mentioned in the previous paragraph.

Furthermore, we enable a custom depth testing method to *only* consider pixels that lie closest to the camera, without performing the computationally heavy color calculation for fragments that we know will be overwritten in the future. For this, we introduce an additional pass, first rendering the ellipsoids in arbitrary order with depth testing enabled, yet using a dummy pixel shader so that we obtain the depth buffer containing the closest points very fast. Then, we set the depth function to `func = GL_EQUAL`, so that all fragments that are not corresponding to the values in the depth buffer are discarded before going through the computations on the pixel shader. We then render the ellipsoids again, this time using the advanced pixel shader for cloud visualization.

## 3.4.3 Depth maps and mipmapping

### Render to texture

In order to obtain depth and normal maps that are required by the pixel shader, we need to write the depth and color buffers to a texture, i.e., we have to render to a texture instead of the screen. For this, a new frame buffer is generated using `glGenFramebuffers(1, &_frameBuffer)` where `_frameBuffer` is its integer identifier.

We define three textures for the normal and depth maps with `glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA32F, _width, _height, 0, GL_RGBA, GL_FLOAT, NULL)` to allow for maximal precision. Additionally, we need to generate a texture to write the depth buffer to, as we want to use depth testing, yet have no access to the regular depth buffer. Thus, we use `glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, _width, _height, 0, GL_DEPTH_COMPONENT, GL_FLOAT, NULL)`.

At rendering time, we bind the custom frame buffer with `glBindFramebuffer(GL_FRAMEBUFFER, _frameBuffer)`. We first bind the depth buffer texture and attach it to the frame buffer by calling the function `glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D, _depthTexture, 0)`. Subsequently, we bind either of the normal or depth textures and attach it to the frame buffer to render the results to it, using, e.g., `glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, _normalTexture, 0)`. Then, the ellipsoids are simply rendered, causing their normals to be written to `_normalTexture`. We repeat this texture binding and attachment with the depth textures `_sunDepthTextureFront` and `_sunDepthTextureBack`, using `GL_GREATER` as the depth function for the latter.

Finally, we switch back to the standard frame buffer using `glBindFramebuffer(GL_FRAMEBUFFER, 0)`, and send the textures to the GPU as uniform variables before continuing with the actual cloud rendering.

## Mipmaps

As we have mentioned, the phenomenological light transport model makes use of mipmapped textures, originally described in 1983 [Wil83]. These are textures that, besides the original image, contain multiple pre-filtered, downsized copies of this image, where every next mipmap is decreased in size, generally by half. This is one of the reasons that a power of two is preferred for texture resolutions. We define the level of detail, or LOD, as a description of which mipmap is to be considered. If the texture is sampled at a level of detail of zero, we simply consider the original image. For every additional LOD, the next mipmap is sampled. This model is illustrated by Figure 3.18.



Figure 3.18: Mipmap pyramid of a cloud texture, from  $512 \times 512$  all the way down to  $1 \times 1$ . The LOD is shown for the original image (LOD0) and the first three mipmaps (LOD1 through LOD4). © Ilya Khamushkin.

The main use of mipmapping is to achieve a higher quality of rendering by generating these texture pyramids as a preprocessing step. Figure 3.19 shows one of its applications. However, we use it to simulate the spatial spreading as described in section 3.1.4. To enable mipmapping for the three normal and depth maps in OpenGL, we first set the filtering accordingly after the texture generation using `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR)`.

When rendering to textures as previously described, we generate the mipmaps of the texture after drawing the ellipsoids by invoking `glGenerateMipmap(GL_TEXTURE_2D)`. Note that on some ATI GPU's, for mipmapping, it is necessary to first explicitly enable textures using `glEnable(GL_TEXTURE_2D)` before generating the mipmaps. Subsequently, we can access the mipmapped textures using the `textureLod()` function on the pixel shader, supplying the required level of detail.

### 3.4.4 Ellipsoids

#### Rendering

Ellipsoids are rendered by looping through the spherical coordinates, converting to Cartesian coordinates to define the vertex positions and to draw the corresponding quads. How many spherical coordinates are considered is modulated by the precision. Throughout the thesis, we use a precision of ten, i.e., we consider ten distinct values for both the altitude and azimuth.

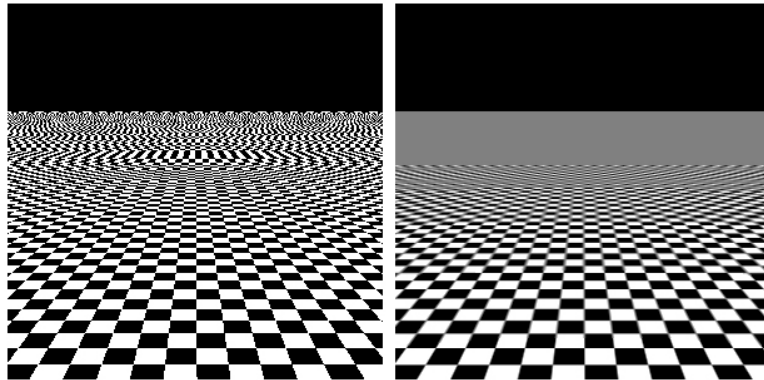


Figure 3.19: Use of mipmapping to improve the rendering quality of objects that lie far away. © ⓘ ⓘ Wojciech Muła / Wikimedia Commons.

### Ordering

On the CPU, we send a list of the ellipsoids to the pixel shader, subsequently looping through this array. For every ellipsoid, we compute the intersection points of the ray that is cast from the considered pixel, as described in section 2.1.2. If the intersection points are found, we use them to perform volume ray marching to compute the opacity and single scattering component of the ellipsoid, while also saving the first intersection point – with a slight offset in the ray direction for more robust sampling – to later use as input for computing the multiple scattering component. Furthermore, the optical thickness that is obtained through ray marching is added to a variable that contains the summed optical thickness of all ellipsoids.

We store three variables – transparency, single scattering, and this *multiple scattering point* – in three arrays, each holding up to six variables. They are stored in ascending order of distance between the camera and the multiple scattering point through comparing the points already in the array with the new point, which means that if the ray intersects more than six ellipsoids, the furthest ones are not taken into account. However, in the rare case that this occurs, these ellipsoids will normally have no visual effect on the cloud anyway, due to being obscured from view. This approach is a bottleneck due to the high degree of branching that is necessary for comparing the multiple scattering point with the values in the arrays, yet we were unable to devise a way that avoids this excessive branching. Still, our method has a significantly better performance than Bouthors et al., as will be shown in chapter 4.

After performing volume marching on all ellipsoids, we compute the opacity of the cloud pixel by using the summed optical thickness of all ellipsoids as in equation (3.7), which ensures a correct result. We initially wanted to apply a similar approach to single scattering; however, the true value for the single scattering component is influenced by the optical thickness of the ellipsoids that lie closer to the camera, and we can not render the ellipsoids in such an order that this optical thickness is already known, for the reasons explained in section 3.4.2. Due to this, we have to correct the single scattering components. Recalling equation (3.21), we have the term  $e^{-\tau}$ , where  $\tau$  is the optical thickness between the first intersection point of all ellipsoids and the considered point in the current ellipsoid. This can be replaced by a term  $e^{-\tau_1}e^{-\tau_0}$ , where  $\tau_1$  is the optical thickness between the *current* ellipsoid’s first intersection point and the considered point, while  $\tau_0$  is the optical thickness between the first intersection point of all ellipsoids and the one of the current ellipsoid, so that  $\tau = \tau_1 + \tau_0$ . We compute the single scattering component that is saved in the array by using only  $\tau_1$ , which means that in order to correct this value, we simply need to multiply it by  $e^{-\tau_0}$ . This value is obtained by reading the array containing the transparencies,  $tt$ , which allows us to compute the single scattering component  $ss_i$  of any  $i^{\text{th}}$  ellipsoid in the single scattering array  $ss$  by multiplying the value with the transparencies of all preceding ellipsoids, and summing the results for all following ellipsoids, as in equation (3.71).

$$ss_i = \sum_{j=i}^5 \left( ss[j] \prod_{k=0}^{j-1} tt[k] \right) \quad (3.71)$$

Having obtained the corrected single scattering, we now need to blend the results of the ellipsoids, as we opted to do this in the pixel shader. It is done in exactly the same way as the standard OpenGL method; we consider the ellipsoids in a back-to-front order, i.e., starting with the sixth ellipsoid, we multiply its single scattering component (analogous to the source) with its opacity (one minus the transparency, equivalent to the source factor) and save it in some variable. For every next ellipsoid, we do the same and add this to the value currently in the variable (the destination) multiplied by the transparency of the currently considered ellipsoid (analogous to the destination factor). This way, the blending is correctly performed, and the value of the variable after all ellipsoids have been considered is used as the final single scattering component, and multiplied by the phase function  $P(\Theta)$  that is available on the GPU in discretized texture form.

Next, the removed peak is computed by simply using the summed optical thickness and the removed peak phase function  $P_p(\Theta)$  in discretized texture form and its weight  $f$ , as in equation (3.65).

We compute the multiple scattering component by using the multiple scattering point that is saved in the array as input for the phenomenological model, the implementation of which is described in section 3.4.5, and blend it similarly to single scattering.

The results are then simply summed together, and subsequently multiplied by the sun color as obtained from the sky model by Hosek and Wilkie, so that the pixel shader can return the final color.

### Distance to boundary

Instead of using a value saved in an octree, we compute the distance from a considered point  $\vec{p}$  to the boundary of an ellipsoid on the fly, making use of the theory discussed in section 2.1.1. We translate  $\vec{p}$  so that it is expressed in Cartesian coordinates relative to the ellipsoid origin  $c_0$ , and then compute the ellipsoidal coordinates corresponding to these relative Cartesian coordinates of  $\vec{p}$ , which we call  $\vec{p}_e$ . The azimuth  $\phi_e$  is simply computed as per equation (2.9), yet we can not apply equation (2.8) for the altitude, as the point  $\vec{p}_e$  does not necessarily lie on the boundary of the ellipsoid. Instead, we consider the help vector  $\vec{q}$  and its length  $q = \|\vec{q}\|$  as shown in Figure 3.20, which are simply computed using the  $x$  and  $z$  coordinates.

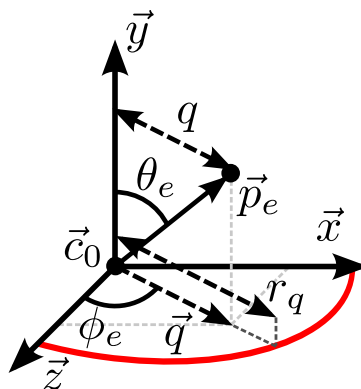


Figure 3.20: The red line describes part of an ellipsoid surface. By computing  $q$  and  $r_q$ , the ellipsoidal coordinate  $\theta_e$  can be calculated.

The corresponding ellipsoid radius in direction  $\vec{q}$ , i.e., the distance between the origin  $c_0$  and the point on the ellipsoid boundary to which  $\vec{q}$  points, which we call  $r_q$ , can be calculated by using the radii of the ellipsoid in the  $\vec{x}$  and  $\vec{z}$  direction and the azimuth, as in equation (3.72). Finally, we can use  $q$  and  $r_q$  to obtain the ellipsoidal altitude  $\theta_e$ , as shown in equation (3.73).

$$r_q = \left\| \begin{bmatrix} r_z \cos \phi_e \\ r_x \sin \phi_e \end{bmatrix} \right\| \quad (3.72)$$

$$\theta_e = \arctan \frac{\frac{q}{r_q}}{\frac{y}{r_y}} \quad (3.73)$$

We then convert these ellipsoidal coordinates back to Cartesian coordinates to find the point on the ellipsoid surface that lies in the direction of  $\vec{p}_e$ , which we call  $\vec{p}_E$ , using equations (2.10) through (2.11). The distance from  $\vec{p}$  to the ellipsoid boundary is then given by the distance between  $\vec{p}_e$  and  $\vec{p}_E$ . Note that due to intersecting ellipsoids, we need to compute the signed distance to every ellipsoid, and take the maximum value (points outside an ellipsoid result in a negative distance).

With the distance to the boundary, we have the input necessary to invoke the sigmoid function and we can subsequently determine the density  $\rho(\vec{p})$  at the point  $\vec{p}$ . However, we found that sometimes, the transition between the heterogeneous boundary and both the homogeneous core as well as the empty air was quite noticeable. To counter this, we invoke the sigmoid function twice around these problematic areas to ensure a smooth transition.

### Rendering from inside ellipsoid

We allow for rendering from inside the cloud by checking if the camera position lies within an ellipsoid. If this is the case, the volume marching is performed between the camera position and the second intersection point. Note that backface culling is *disabled*. Otherwise, an ellipsoid is not rendered when the camera is inside it due to viewing a backface.

### 3.4.5 Phenomenological model

Given the multiple scattering point for which we want to invoke the phenomenological model, the screen-space coordinates of this point from the sun's viewpoint are obtained, which are used for sampling the normal and depth maps. Bouthors et al. sampled these using a level of detail dependent on the collector size of the current iteration, yet we opt for a static level of detail of LOD6, which produced the best results, as shown by Figure 3.21. Due to the fact that we render the normal and depth maps without noise, low LOD results in discontinuities introduced by discrepancies between the noise-free textures and noise-augmented renderings.



(a) LOD3 has details but results in discontinuities. (b) LOD6 retains details and high contrast, yet produces smooth results. (c) LOD7 has smooth results – without any details, however.

Figure 3.21: Different results for certain levels of detail.

While the implementation of the phenomenological model is quite straightforward, one issue occurs due to the mipmapping. When sampling at point  $\vec{p}$  on a protruding part of the cloud volume as in Figure 3.22, we obtain a viewpoint depth of  $d = 0$ . However, when the camera position is below this viewpoint



with regard to the normal  $\vec{n}$ , which commonly occurs, this means that we are looking at the slab from the bottom, which, coupled with  $d = 0$ , results in no light transport from multiple scattering at all. To overcome this, we clamp the minimum value of  $d$  to  $\frac{t}{10}$  to ensure some light transport for these scenarios.

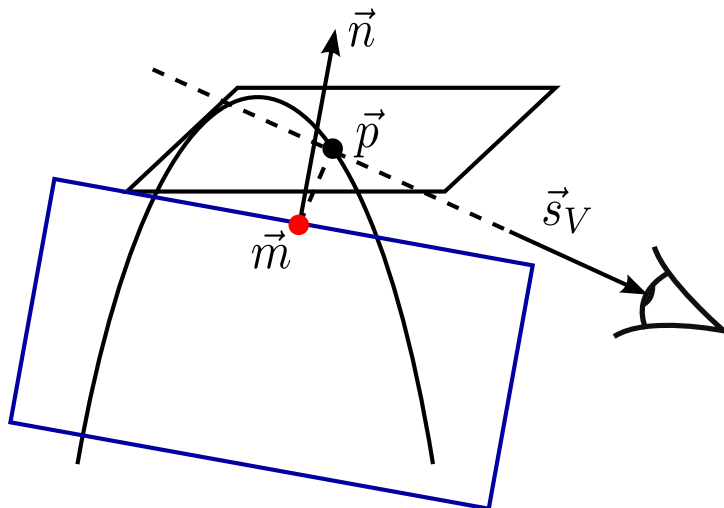


Figure 3.22: This scenario results in no light transport, while in reality, there can be a significant amount of light exiting the cloud at viewpoint  $\vec{p}$ .

### 3.4.6 Noise evaluation

For our noise evaluation, we make use of a texture with a resolution of  $64^3$ , which can be generated quickly during the preprocessing step, using only  $4 \frac{64^3}{1024^2} = 1\text{MB}$  of memory when using 4-byte precision. To obtain a level of detail corresponding to a texture of size  $512^3$ , we multiply the texture coordinates  $\vec{p}$  that are used for sampling by eight. Due to the fact that this causes repetition, we also sample the texture with the original coordinates, and add the results. This creates a sufficiently distinct noise texture, with high details and relatively low memory requirements.

To produce the fractal shape, the noise texture is sampled at five mipmaps levels, where the results of each octave are multiplied by a scalar. Through experimentation, we found the best results using the values shown in equation (3.74), where  $N(\vec{p})$  is the resulting noise and  $N_i(\vec{p})$  is the noise texture sampled at  $\vec{p}$  with LOD  $i$ . Finally,  $N(\vec{p})$  is clamped to the range  $[0, 1]$  and subsequently converted to the range  $[-1, 1]$

$$\begin{aligned} N(\vec{p}) = & 1.5N_4(\vec{p}) + 0.35N_3(\vec{p}) + 0.1N_2(\vec{p}) + 0.035N_1(\vec{p}) + 0.015N_0(\vec{p}) - 0.75 \\ & + 1.5N_4(8\vec{p}) + 0.35N_3(8\vec{p}) + 0.1N_2(8\vec{p}) + 0.035N_1(8\vec{p}) + 0.015N_0(8\vec{p}) - 0.75 \end{aligned} \quad (3.74)$$

### 3.4.7 Animation

We animate the cloud at virtually no additional cost by translating the model along the scene, and artificially lowering the distance to the ellipsoid boundary as the cloud lies further from the center of the scene.

The former is modulated by a two-dimensional vector denoting the wind direction, and every frame, the cloud is moved in this direction until it lies completely outside of the sky dome, by which time it is reset to the opposite position outside the sky dome, so that it can slowly translate back into view again to return to its original position. We multiply the distance to the ellipsoid boundary with one minus the length of the vector describing the world position of a pixel, divided by the radius of the sky dome. This

means that this factor transitions smoothly from one at the scene origin (which also is the center of the sky dome), to zero at the sky dome boundary, in turn resulting in  $\rho = 0$ .

Translating the cloud has a significant advantage; due to the fact that the noise evaluation takes the pixel's world position as input, the noise is animated as well. This causes the cloud to display movements that look quite realistic, without using complex fluid dynamics calculations.

## Chapter 4

# Results and analysis

### 4.1 Experiment conditions

#### 4.1.1 Performance

We can not replicate the exact implementation that Bouthors et al. used to achieve their frame rates, due to a lack of information and resources, such as the heterogeneous boundary thickness and the complex triangle mesh of the cumulonimbus that they used. As a consequence, we can not make a scientific comparison with regards to the performance of our method. Instead, we compare our FPS directly with the frame rates of Bouthors et al. in 2008, taking into account the respective GPU's that were used to obtain the frame rates. Using this information, we give a guesstimate of the speedup of our method compared to Bouthors et al.

If we can show to achieve real-time frame rates (at least 15 FPS) in full-screen mode on relatively low-end GPU's, we have fulfilled the part of the goal related to performance. We measure the frame rate by using FRAPS, a benchmarking utility for Windows.

Furthermore, we consider the hypotheses of section 3.3, and discuss the performance-related statements, quantifying them as best as possible.

#### 4.1.2 Accuracy

In computer graphics, it is often difficult to quantify the accuracy of results, as the ground truth is not always known. It is true that Monte Carlo path tracing could be used to compare real-time results against the offline Monte Carlo renderings, yet we lacked the time to implement this.

We could say that graphics are accurate when they look realistic, yet the definition of realism is quite subjective; what looks indistinguishable from nature to some, may look completely wrong for others, while some people regard computer generated images as realistic, yet when faced with real-life footage they must admit that it looked nowhere near as realistic as they thought. Finally, many natural phenomena exist that some may find highly unrealistic – if unaware of their authenticity – let alone the computer generated renderings that simulate such phenomena.

Still, to analyze the accuracy, we have no other choice but to resort to visual inspection, comparing our results with those of Bouthors et al. To provide for a more quantifiable judgment, we also consider all visual cloud features discussed in section 2.2 and determine how well we simulate them. If our results model all these cloud features adequately, we have attained the accuracy-related part of our goal as well.

### 4.2 Performance

Bouthors et al. noted frame rates of around 10 FPS for their detailed cumuliform cloud, using no hypertexture to augment the triangle mesh, while only 2 FPS were achieved with the hypertexture enabled. They obtained these frame rates at a resolution of  $800 \times 600$ , on an NVIDIA GeForce 8800 GTS. The

specifications of this GPU consist of a core clock of 500 MHz and a video memory of 320 or 640 MB [NVI06]. However, benchmarks are generally considered to say a lot more about a GPU’s capabilities than its specifications. One of the industry standards for video card benchmarking is PassMark software, for which the GeForce obtains a score of 602 [Pas13]. The GeForce was a high-end card at the time of its release in 2006, and could still be considered as such when used by Bouthors et al. in 2008. Currently, in 2013, it is but a mid-level GPU, however.

Considering the fact that the present top-of-the-line video card, the GeForce GTX Titan, achieves a PassMark score of 8055, we are confident in saying that current hardware allows for the Bouthors method to reach real-time performance.

Running our method at a  $800 \times 600$  resolution, we obtain frame rates of 48 FPS with animated clouds, run on a notebook with an AMD ATI Mobility Radeon HD 4650. This video card has a clock speed of 500 MHz and 512 MB of memory [GPU09], quite comparable to the GeForce. The PassMark results for this GPU however, result in a score of only 379 [Pas13], indicating significantly inferior performance. At its release in 2008, it was already a mid-range GPU, specifically made for laptop computers. Now, it inclines toward being a low-end video card, as shown by the relatively low PassMark performance. Based on the benchmarks, we feel confident in saying that the GPU used by Bouthors et al. could achieve at least 50% higher frame rates for most applications than our video card. This means that our method consists of a performance improvement of at least factor 36.

In full-screen mode, which corresponds to a resolution of  $1600 \times 900$  for the laptop on which the tests were run, an average FPS of 25 is attained. With this, we achieve our goal of performing the full-screen cloud visualization with at least 15 frames per second on a low-end GPU.

The main bottleneck in our approach is the way in which the ellipsoids are considered. For every ellipsoid, the distance to the boundary needs to be computed for every segment, and this distance computation requires checking all ellipsoids in turn. For  $E$  ellipsoids and  $N$  segments, this results in a time complexity of  $O(E^2N)$ , which is rather bad. We still attain high frame rates due to using only eight ellipsoids and five segments, which show to be enough for realistic cloud rendering. However, if multiple clouds need to be rendered, this scales rather poorly. Luckily, all clouds can be visualized separately, with OpenGL taking care of blending by rendering them in a back-to-front order. By not letting clouds intersect – if they did, they would not be distinct clouds – this can be done efficiently by considering the cloud centers. This means that while the time complexity is  $O(E^2N)$  for one cloud, it scales proportionally to the amount of clouds, and the method is still efficient due to the low number of ellipsoids per cloud and segments per ellipsoid.

As for using our method in games, we do believe it to be possible for low resolutions on high-end GPU’s. It would be interesting to benchmark our method on such a video card to get a better idea of the possibilities. Our current implementation is not fully optimized, thus we are confident in saying it can be applied for high-end games – if not now, at least in the near future.

### 4.2.1 Hypotheses

The hypothesis in section 3.3.1, pertaining to our ellipsoid model, states that it allows for a significant increase in performance, yet it is difficult to provide for a comparison with Bouthors’ model, as we lack information on the boundary thickness, and have no access to the triangle mesh that they have used. To this end, we first discuss the improved frame rates due to our other contributions.

In the mean-time, we can state that thanks to our model, the preprocessing step is a lot faster than that of Bouthors et al. After starting the executable, it takes less than two seconds before the first rendering shows up on screen, compared to less than a second for a dummy executable, rendering our preprocessing almost negligible. Considering the amount of initialization required for the triangle mesh and octree, we can safely say our method is significantly faster.

The main performance improvement related to opacity and single scattering, as mentioned in section 3.3.2, is the notion of combining the two to compute them in one go. Interestingly, after experimentation we found that this only has a very small effect on the frame rate, yet it does make the code more readable.

The comparison of the multiple scattering point with values in the array containing previous points, and the subsequent storage of the transparency and single scattering in arrays, only has to be done once, even with opacity and single scattering computed separately. The fact that the remainder of the

computation has a small effect on the amount of frames per second shows that this comparison is quite expensive. Indeed, when removing it, the frame rate is increased by approximately 20%, yet it is crucial to our ellipsoid model, to allow for blending and correcting the single scattering.

In section 3.3.2, we considered five segments to be sufficient in obtaining adequate results, while retaining good performance. Figure 4.1 shows a plot of the amount of frames per second against the number of segments  $N$  that were used for volume marching, for resolutions of  $1600 \times 900$  and  $800 \times 600$ . Bouthors et al. used twenty segments for opacity computation, and ten for single scattering; when using thirty segments with our method, we obtain frame rates of 6 and 10 FPS for resolutions  $1600 \times 900$  and  $800 \times 600$ , respectively. However, we define the number of segments per *ellipsoid*, while Bouthors et al. defined the amount of segments per *pixel*. This means that when six ellipsoids intersect the ray cast from one pixel, using five segments still results in thirty segments per pixel. As the amount of ellipsoids vary per pixel, especially given the unpredictable branching that follows from this, it is impossible to estimate if our choice of segments allows for an increased performance when compared to Bouthors et al.

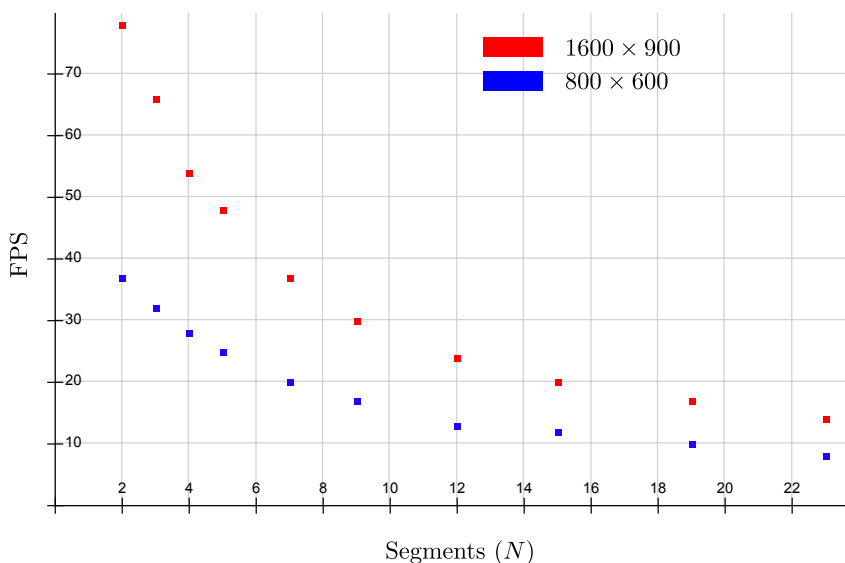


Figure 4.1: Frame rates plotted against the amount of segments  $N$  seem to show negative exponential behavior.

In our improvement of the multiple scattering computation, discussed in section 3.3.3, we stated that taking the initially fitted slab obtained through sampling the normal and depth maps at a static level of detail would significantly speed up the frame rates. At  $800 \times 600$ , the original approach of using the iterative collector finding algorithm and the triple sampling of the normal and depth maps to simulate a Gaussian, results in an average of 6 FPS, which means our approach achieves a speedup of factor 8. Additionally, at a resolution of  $1600 \times 900$ , an average frame rate of 3 FPS is noted, further confirming the frame rate increase factor of around 8 on multiple scattering alone.

Having discussed the increased performance for all contributions but the ellipsoid model, we can conclude, given the total speedup factor of approximately 36, that reducing the shape representation from a high-detail triangle mesh and a sparse voxel octree to a set of ellipsoids, plays a very significant role in the improved frame rates. To what extent exactly is difficult to say, since the frame rate does not scale predictably.

Finally, having experimented with cloud animation as described in section 3.4.7, we found that it had no noticeable effect on the performance whatsoever. This is thanks to our animation technique, which simply consists of translation, letting the noise evaluation efficiently imitate complex movement. Indeed, the only component of our framework slowing down the rendering speed when animated, is the sky color model of Hosek and Wilkie.

### 4.3 Accuracy

The results attained by Bouthors et al. are shown in a collage in Figure 4.2. Videos containing a moving camera or sun position are available on their project website at <http://www-evasion.imag.fr/Publications/2008/BNMBC08/>. They display quite realistic visualizations, although the issue of overly sharp boundaries persists. In our opinion, this gives the clouds just a slightly cartoon-like look.



Figure 4.2: Results achieved by Bouthors et al. [Bou08].

A collage of our results is shown in Figure 4.3, which displays a cloud consisting of just eight ellipsoids, with a maximum cloud diameter of about 400m and a heterogeneous boundary thickness of approximately 62m. While our results do simulate the wispy boundaries of a cloud nicely, they lack the sharp edges that can occur as well. This diminishes the realism of the clouds, causing less convincing results than those of Bouthors et al. Video captures of our animated clouds can be viewed on YouTube at <http://www.youtube.com/watch?v=C05PWpJpfo> and <http://www.youtube.com/watch?v=uXXV8kz-1wY>.

Decreasing the boundary size does not solve the problematic edges; due to the fact that we use simple ellipsoids to define the general shape, thin edges produce very unrealistic results, as can be seen in Figure 4.4, for instance, which displays a cloud of the same size, rendered with a boundary thickness of around 18m.

We now discuss all visual features as observed in section 2.2, and how well our method manages to simulate them.

#### 4.3.1 Fractal shape

The fractal shape of the cloud is mainly simulated through the augmentation of fractal noise. Figure 4.5 shows that this is done quite well, yet attaining fractal results with sharp boundaries is unfeasible for our method, due to the fact that only a relatively small set of ellipsoids can be used to maintain high frame rates.

#### 4.3.2 Cloud edges

Cloud edges form the main weakness of our method. As discussed, wispy cloud edges are realistically simulated, yet the sharp boundaries that are common in cumuliform clouds as well, can not be adequately visualized with our model. One way to artificially increase how well-defined the cloud boundaries appear



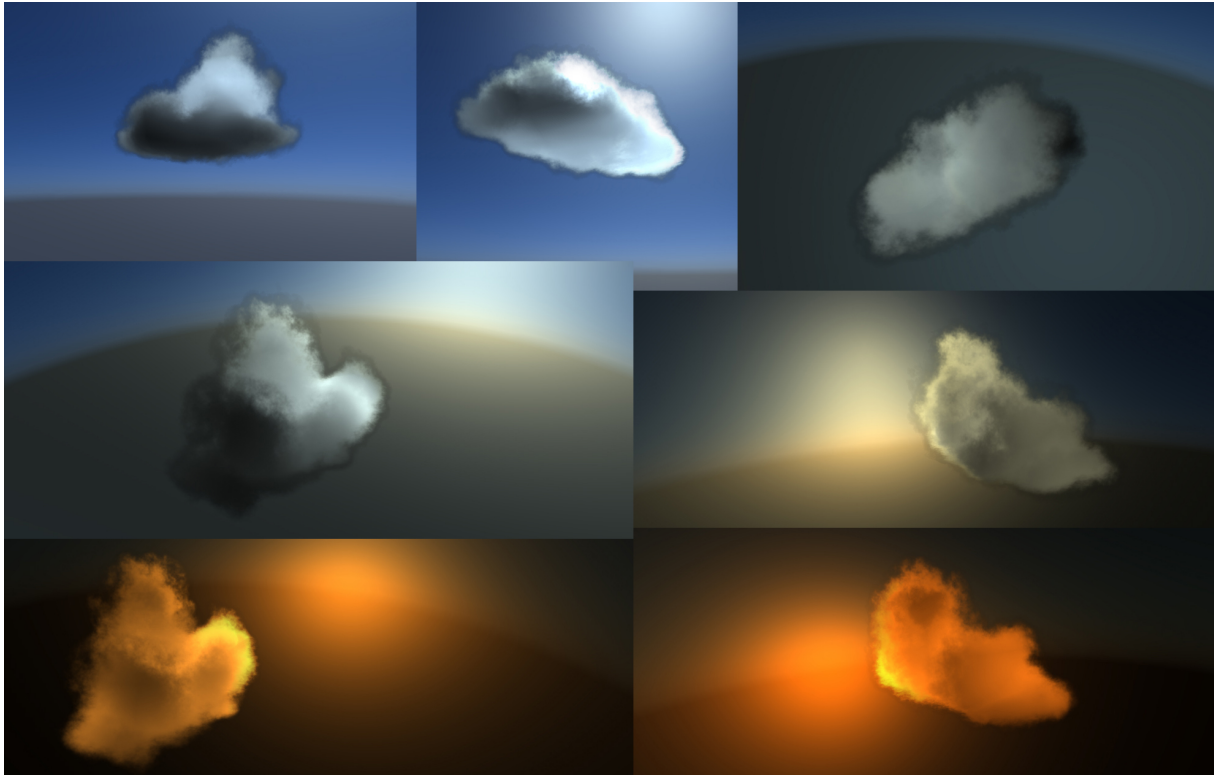


Figure 4.3: Our results.

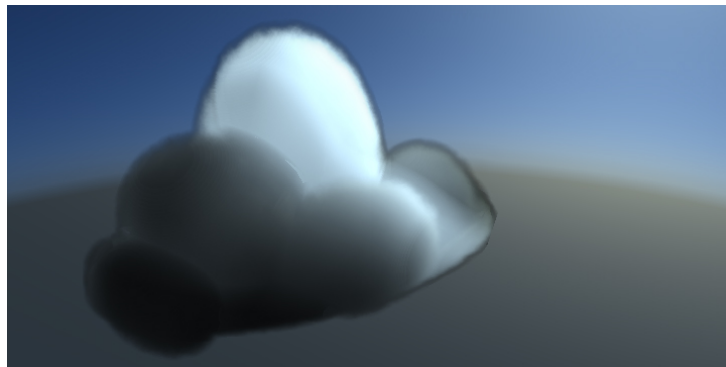


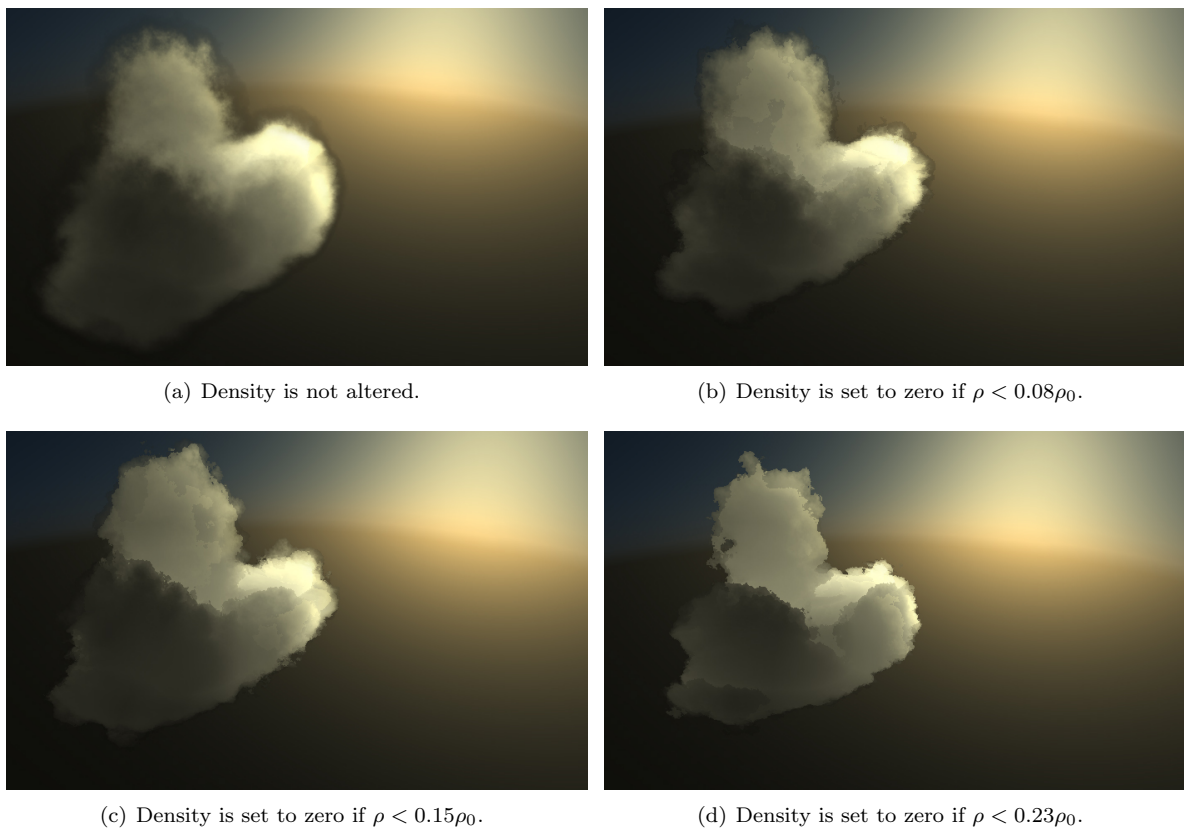
Figure 4.4: Using a thin boundary causes the underlying shape of the ellipsoids to emerge, producing very unrealistic results.

is setting the density  $\rho$  to zero when it is exceeded by a percentage of the homogeneous density  $\rho_0$ . Figure 4.6 shows the results of experimenting with this approach. However, it can be observed that this does not improve the realism of the cloud very much.

All in all, we can say that the cloud edges are partly simulated adequately, yet we lack the means for a convincing visualization of sharp boundaries. The three-dimensional nature of the edges is taken into account nicely, as our noise evaluation does not merely consider the silhouette of the cloud, and the distance to the ellipsoid boundary is used to properly vary the droplet density.



Figure 4.5: Fractal shape of our cloud rendering.

Figure 4.6: Setting the density value to zero at sampling points that produce a  $\rho$  below a certain threshold, results in sharper boundaries, yet does not add much realism.

### 4.3.3 Multiple scattering

As we use the phenomenological light transport model, the macroscopic light behavior is faithfully approximated. Figure 4.7 shows that for relatively thin clouds, like the top images, with a cloud thickness of 225m – i.e., the maximum cloud diameter in the  $\vec{y}$  direction – the transmittance is significantly stronger than the reflectance. Additionally, the images in the middle show that with a thickness of 500m, the

transmitted light is about equal to the backscattered light. Finally, the bottom images display clouds with a considerable thickness of 900m, where backscattering is obviously dominant, in accordance with our observations of section 2.2.3 and Bouthors' study of light transport. Furthermore, we can see that the anisotropy is correctly preserved in the optically thin areas of the cloud boundaries.

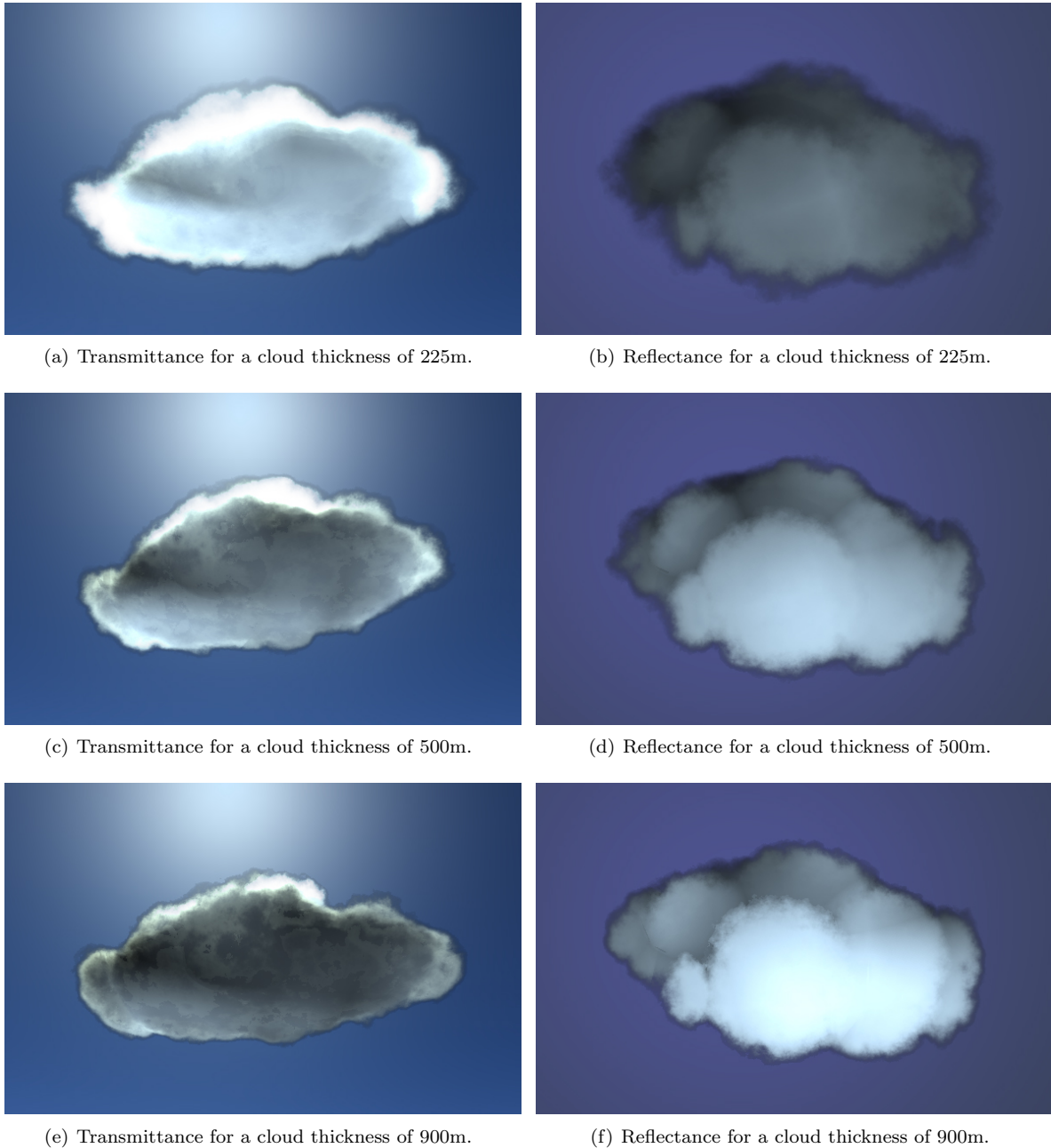


Figure 4.7: The macroscopic behavior of light for transmittance and reflectance given a varying cloud thickness.

### 4.3.4 Silver lining and other contrast

The silver lining is simulated nicely by the single scattering computation, as testified by the images of Figure 4.8. The high contrast has been shown in many images already, such as Figures 4.3, 4.5, 4.6 and the right image in Figure 4.8. Self-shadowing is accounted for thanks to sampling the sun front depth map for finding the single scattering component. High variety in droplet density is ensured by augmenting the model with high-detailed noise, combined with using ellipsoids, with empty spaces in between allowing for high contrast.

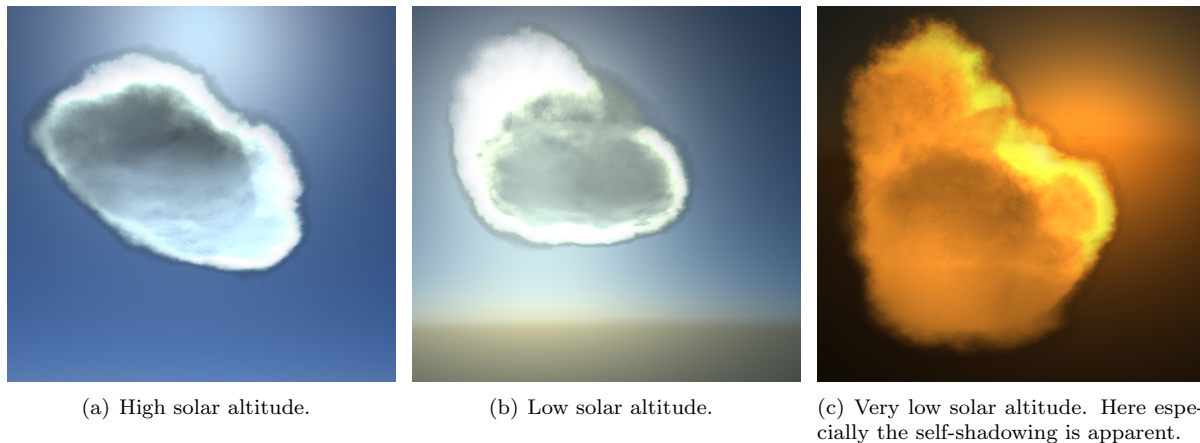


Figure 4.8: Images displaying silver lining and high contrast due to self-shadowing and varying droplet density.

### 4.3.5 Glory and fogbow

Thanks to the Mie phase function we maintain the glory and fogbow in our model, as seen in Figures 4.9 and 4.10. Especially the fogbow is rather subtle, which is why saturated images are provided as well to better distinguish them. This subtlety is the same in nature, however, as shown by the photographs in Figures 2.20 – which is slightly saturated itself – and 2.21. As per real life, the size of the glory and fogbow remains constant with respect to the distance from the camera to the cloud.

### 4.3.6 Cloud lighting

Using Hosek and Wilkie’s skylight model, the color of the sky is inherently taken into account when computing the color of the sun. Indeed, we can easily see that our clouds contain blue hues – possibly even too much of it. We do not take the color or the albedo of the ground into account, yet doing so would not have a huge impact on the cloud appearance, as per section 2.2.6.

### 4.3.7 Cloud movement

To judge the realism of our cloud animation, the videos on YouTube at <http://www.youtube.com/watch?v=C05PWpJpfo> and <http://www.youtube.com/watch?v=uXXV8kz-1wY> need to be investigated. We acknowledge that the results do not look exactly like fluid dynamics, yet due to the slow-moving nature of clouds, this noise animation produces, in our eyes, quite convincing results. The formation and dissipation is also simulated nicely through artificially lowering the distance to the boundary as the cloud moves further from the center of the scene.

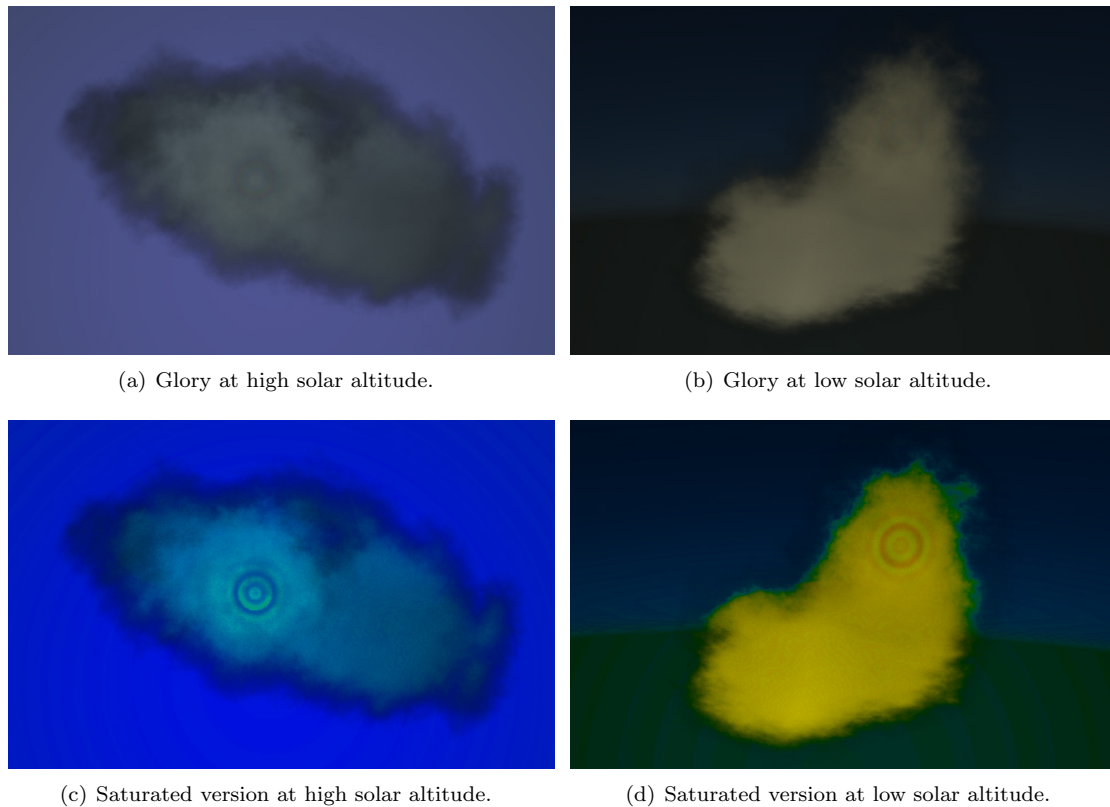


Figure 4.9: The glory is visible in the antisolar point, as cleared up by the saturated images. The cloud has a low optical thickness so that the glory is not drowned out by significant backscattering.

#### 4.3.8 Hypotheses

We have claimed that five segments suffice for generating convincing images, while still retaining high frame rates. Figure 4.11 shows that this is true; there is no visible difference between using five or eight segments, at least for static images. Using too few segments, on the other hand, causes discontinuities. It should be noted that when the camera is moving, the use of a low number of segments becomes more noticeable, as the fractal noise seems less consistent. With five segments, we found the resulting inconsistencies to be acceptable. When the cloud is moving, they are hardly noticeable due to the animated noise.

In section 3.3.3, we stated that using the initially fitted slab produces adequate results. Having shown a variety of results, we can conclude this to be the case; our model simulates the macroscopic behavior correctly and produces high contrast as in real clouds.



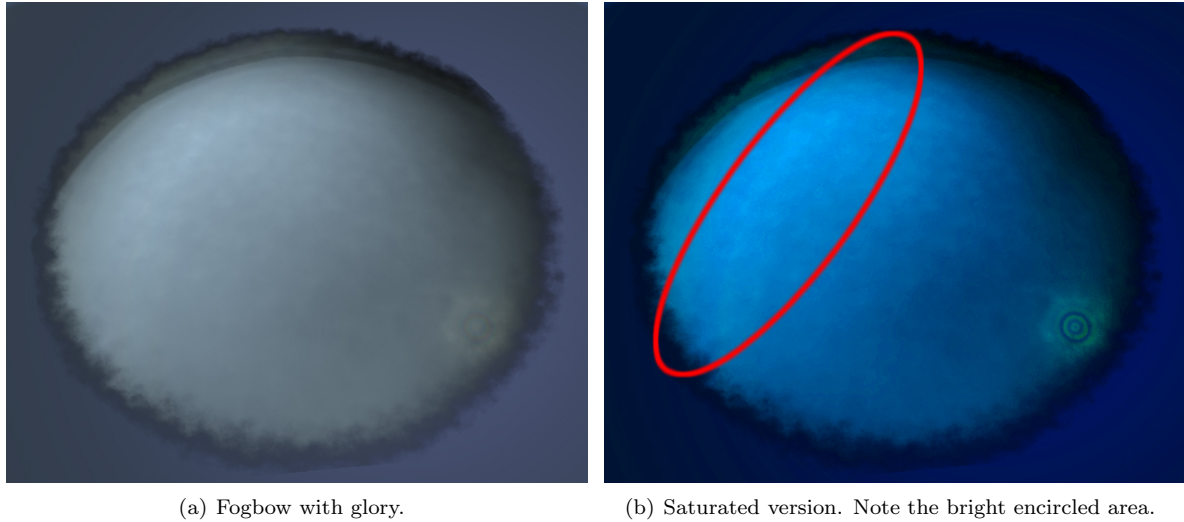


Figure 4.10: The fogbow is visible as a ring around the antisolar point, as cleared up by the saturated image. The cloud has a low optical thickness so that the fogbow is not drowned out by significant backscattering. A single ellipsoid is used to make the fogbow more noticeable. While it can be distinguished in our regular cloud model as well, the camera needs to move around to notice it.

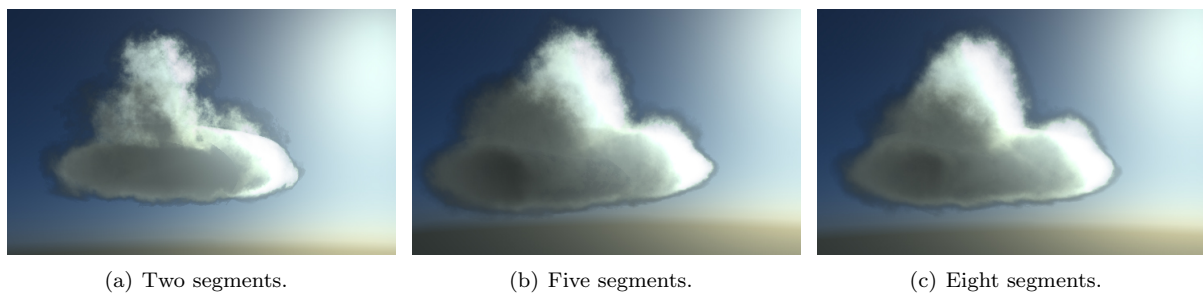


Figure 4.11: Using too few segments causes discontinuities, yet five seem sufficient to produce good results.



# Chapter 5

## Conclusions

### 5.1 Contributions

We have proposed a new way of implementing ellipsoids to represent the shape of a cloud, combining this model with a discretized Mie phase function to correctly simulate opacity and single scattering, and the phenomenological model of light transport in a plane-parallel slab as described by Bouthors et al.

Moreover, we have shown that the preprocessing step of our method is negligible, while existing techniques, especially that of Bouthors et al., have a significant initialization time.

Also, we have eased the job for artists, due to the fact that our model requires only the definition of a few ellipsoids to be fully capable of rendering a realistic cloud with the specified general shape. This is opposite to the advanced triangle mesh that is required to generate realistic results using Bouthors' method.

Directly related to the model is our solution for concavities; using ellipsoids and considering them separately for ray marching, we ensure convex volumes, thus preventing artifacts in the single scattering component.

Using ellipsoids, we are able to define a much thicker heterogeneous boundary, as is the case with real-life clouds. This way, we have managed to simulate the wispy edges that occur in cumuliform clouds realistically, using our algorithm for computing the distance to the boundary of an ellipsoid.

Furthermore, we have presented a method for very fast noise evaluation on the GPU, requiring a relatively small amount of memory (1MB at most) yet still producing high-detailed, non-repetitive results.

The ellipsoid model instead of Bouthors' triangle mesh combined with an octree, as well as our significant performance enhancement pertaining to the multiple scattering computation, results in much better frame rates, with a speedup factor of approximately 36.

Additionally, we have combined the opacity and single scattering computation to obtain both in one volume march, making the algorithm more natural and elegant.

Moreover, our modifications of the multiple scattering algorithm make it more robust, with no discontinuities occurring for certain shapes. The removal of the collector finding algorithm ensures that neighboring points on the cloud surface produce similar light transport results.

Finally, we have implemented animation, translating the cloud across the scene while simulating fluid dynamics through the animated noise texture, and dissipation and formation through modifying the droplet density by artificially lowering the distance to the ellipsoid boundary as the cloud travels further from the origin of the scene.

### 5.2 Conclusion

The goal of the thesis as described in section 1.2.3, was *to render a cumuliform cloud during daylight with at least fifteen frames per second on low-end GPU's at a full-screen resolution, while allowing for a moving camera and sun, preferably animating the cloud as well. The results must stay true to all visual features*

*of cumuliiform water droplet clouds, and meet or even surpass current state-of-the-art visualizations that attain interactive frame rates.*

We have done quite well to achieve this, easily surpassing the set minimum of 15 FPS, and implementing adequate animation too. The sun and camera can be freely moved in the framework, causing no computational overhead whatsoever. While the results model most cloud features nicely, there is one issue with cloud edges. Our method is unable to realistically visualize sharp boundaries that are quite common on cumuliiform clouds.

Still, in our opinion, our results surpass most existing work in terms of realism, yet we fail to reach the level of authenticity as achieved by Bouthors et al. [BNM<sup>+</sup>08] and Elek et al. [ERWS12]. Both these methods however have a considerable amount of drawbacks, as thoroughly discussed in chapters 2 and 3. Our method produces one of the most realistic animated real-time results, and is easy to use for artists as well, even if we have not managed to fully achieve the goal as set in section 1.2.3.

### 5.3 Future work

There is still much that can be done to improve our model. For instance, it would be very advantageous to artists to implement an integrated ellipsoid modeling system, in which ellipsoids can easily be constructed, moved, and scaled (through modifying the radii) during the real-time visualization. This way, they can immediately see what the resulting cloud looks like. Additionally, it would be ideal if certain parameters could be tweaked during runtime, such as the cloud scale, boundary thickness, homogeneous core density and translation vector – i.e., wind direction. Implementing such a system would be quite straightforward; unfortunately, we lacked the time to do so.

Furthermore, it would be interesting to investigate the possibilities of augmenting the current animation by rotation of the noise texture sampling coordinates. Schpok et al. [SSEH03] use this to produce quite impressive movement simulations, and it may prove to be a fruitful endeavor for our model as well. Related to this, the ellipsoids themselves could be animated, e.g., through translation relative to the other ellipsoids that constitute the cloud or by modifying the radii describing them.

Another addition that might prove to be beneficial is the inclusion of the albedo of the terrain. The sky model of Hosek and Wilkie that is used in our framework already makes use of this, and implementing an algorithm that slightly modulates the cloud brightness depending on the albedo would not be difficult.

Perhaps more importantly, the current sun color seems to be a tad too blue at high solar altitudes, which reflects on the cloud color directly. However, this is part of the sky color model and has little to do with our cloud rendering method. Still, looking for a more correct color computation may produce better results.

Despite the fact that these extensions of our model are quite interesting and may improve the results, the most crucial parts that require further research are the three main drawbacks.

One of these pertains to the single scattering component. Due to the approximation of the optical thickness between a sampling point and the corresponding point on the cloud surface in the negative sun direction, we can observe quite steep variance in color at the base of our clouds, being caused by the noise. In real life, this is canceled out due to the significant optical thickness between the first sampling points and the corresponding point on the cloud surface. Due to the approximation, however, this optical thickness is estimated to be negligible, as the extinction coefficient at these first sampling points is negligible as well. This results in high variance when looking at a cloud in the sun direction, which does not occur in real life. This may be solved by using a better approximation – for instance, introducing an additional sampling point along the line between the cloud surface and the volume marching sampling point.

The second drawback is related to the phenomenological light transport model. It sometimes causes very sudden transitions to appear, which is usually only noticeable when either the camera, sun, or cloud itself is moved. Overcoming this problem is very difficult; we have experimented a long time with the model and have found the method described in this thesis to produce the most robust results. Solving this drawback would probably mean having to define a new phenomenological model, which would be an interesting endeavor in any case, as this time, perhaps a model for spheres or ellipsoids instead of

plane-parallel slabs could be investigated. These primitives may prove to allow for a better fit to an arbitrary cloud, thus resulting in more robust and realistic images.

Probably the most crucial drawback is the lack of sharp boundaries, and the inability to faithfully reproduce them. As we have discussed in chapter 4, several options that have been considered – lowering the heterogeneous boundary thickness or artificially setting the density to zero for a certain threshold – proved to be unable to overcome this problem. However, going back to Bouthors’ model would not solve the issue either; as that method is incapable of rendering convincing wispy edges. A model would have to be devised that is able to realistically simulate both. In our opinion, future research should be focused on this issue.

## 5.4 Reflections

This project has been an extremely educational undertaking for me. I have learned a great deal about computer graphics, both in the theoretical sense – reading through numerous of papers on cloud rendering – and the practical sense – spending days on OpenGL and GPU programming websites and implementing techniques myself – and I liked it so much that I now aim to make a career in this field. Already having had an interest for the graphical side of computer science, I have become even more fascinated during this project. For me, the field of computer graphics transcends regular science, almost becoming like art, while still retaining the magnificent challenges that only science can provide. After all, the goal is to produce realistic imagery, not unlike traditional painting styles, yet it is often required to do this as efficiently as possible as well. Additionally, its applications are forms of art themselves: films and games, huge industries that play important roles in the lives of many people.

The process of crafting this thesis has not always been easy, with several setbacks such as being stuck for months on correctly implementing the multiple scattering algorithm. The fact that prior to this project, I had never worked with GPU programming also cost a lot of time. However, I am still quite content with the results, and this journey has definitely sparked my interest in pursuing more knowledge.



Figure 5.1: One last photograph emphasizing the importance of the sky and clouds in natural scenes.

© ⓘ Ⓞ Ⓜ Rawhead Rex.

# Bibliography

- [BC06] C.F. Bohren and E.E. Clothiaux. *Fundamentals of Atmospheric Radiation: An Introduction with 400 Problems*. John Wiley & Sons, 2006.
- [BH08] C.F. Bohren and D.R. Huffman. *Absorption and Scattering of Light by Small Particles*. John Wiley & Sons, 2008.
- [Bli82] James F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, 1982.
- [BN04] Antoine Bouthors and Fabrice Neyret. Modeling clouds shape. In *Short papers from Conference of the European Association for Computer Graphics*. Eurographics Association, 2004.
- [BNL06] Antoine Bouthors, Fabrice Neyret, and Sylvain Lefebvre. Real-time realistic illumination and shading of stratiform clouds. In *Proceedings of Conference on Natural Phenomena*, pages 41–50. Eurographics Association, 2006.
- [BNM<sup>+</sup>08] Antoine Bouthors, Fabrice Neyret, Nelson Max, Eric Bruneton, and Cyril Crassin. Interactive multiple anisotropic scattering in clouds. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, pages 173–182. ACM, 2008.
- [Bou08] Antoine Bouthors. *Realistic rendering of clouds in real-time*. PhD thesis, Université Joseph Fourier, 2008.
- [CNLE09] Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, and Elmar Eisemann. Gigavoxels: Ray-guided streaming for efficient and detailed voxel rendering. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, pages 15–22. ACM, 2009.
- [CRC84] Petr Chýlek, V. Ramaswamy, and Roger J. Cheng. Effect of graphitic carbon on the albedo of clouds. *AMS Journal of the Atmospheric Sciences*, 41(21):3076–3084, 1984.
- [CS92] William M. Cornette and Joseph G. Shanks. Physically reasonable analytic expression for the single-scattering phase function. *OSA Applied Optics*, 31(16):3152–3160, 1992.
- [DEYN07] Yoshinori Dobashi, Yoshihiro Enjyo, Tsuyoshi Yamamoto, and Tomoyuki Nishita. A fast rendering method for clouds illuminated by lightning taking into account multiple scattering. *The Visual Computer: International Journal of Computer Graphics*, 23(9):697–705, 2007.
- [DK02] Stanislav Darula and Richard Kittler. CIE general sky standard defining luminance distributions. In *Proceedings of eSim*, pages 7–14. IBPSA-Canada, 2002.
- [DKY<sup>+</sup>00] Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of SIGGRAPH*, pages 19–28. ACM, 2000.
- [DLR<sup>+</sup>09] Craig Donner, Jason Lawrence, Ravi Ramamoorthi, Toshiya Hachisuka, Henrik Wann Jensen, and Shree Nayar. An empirical BSSRDF model. In *Proceedings of SIGGRAPH*, pages 30:1–30:10. ACM, 2009.

- [DS03] Carsten Dachsbacher and Marc Stamminger. Translucent shadow maps. In *Proceedings of Workshop on Rendering*, pages 197–201. Eurographics Association, 2003.
- [EBA<sup>+</sup>11] Hadi Esmailzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of International Symposium on Computer Architecture*, pages 365–376. IEEE, 2011.
- [Ebe97] David S. Ebert. Volumetric modeling with implicit functions: A cloud is born. In *Visual Proceedings of SIGGRAPH*, page 147. ACM, 1997.
- [ERWS12] Oskar Elek, Tobias Ritschel, Alexander Wilkie, and Hans-Peter Seidel. Interactive cloud rendering using temporally-coherent photon mapping. In *Proceedings of Graphics Interface*, pages 141–148. Canadian Information Processing Society, 2012.
- [ES00] Pantelis Elinas and Wolfgang Stürzlinger. Real-time rendering of 3D clouds. *Journal of Graphics Tools*, 5(4):33–45, 2000.
- [FPBP09] Adrià Forés, Sumanta N. Pattanaik, Carles Bosch, and Xavier Pueyo. BRDFLab: A general system for designing BRDFs. In *Proceedings of Congreso Español de Informática Gráfica*, pages 153–160. Eurographics Association, 2009.
- [FSJ01] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of SIGGRAPH*, pages 15–22. ACM, 2001.
- [Gar85] Geoffrey Y Gardner. Visual simulation of clouds. In *Proceedings of SIGGRAPH*, pages 297–304. ACM, 1985.
- [GMG08] Enrico Gobbetti, Fabio Marton, and José Antonio Iglesias Guitián. A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets. *The Visual Computer: International Journal of Computer Graphics*, 24(7):797–806, 2008.
- [GPP<sup>+</sup>10] Olivier Gourmel, Anthony Pajot, Mathias Paulin, Loïc Barthe, and Pierre Poulin. Fitted BVH for fast raytracing of metaballs. *Eurographics Computer Graphics Forum*, 29(2):281–288, 2010.
- [GPU09] GPU Boss. Mobility Radeon HD 4650. Web, January 2009. Consulted October 2013.
- [Gus05] Stefan Gustavson. Simplex noise demystified. Technical report, Linköping University, Sweden, 2005.
- [HAP05] Kyle Hegeman, Michael Ashikhmin, and Simon Premoze. A lighting model for general participating media. In *Proceedings of Symposium on Interactive 3D Graphics and Games*, pages 117–124. ACM, 2005.
- [HBSL03] Mark J. Harris, William V. Baxter, Thorsten Scheuermann, and Anselmo Lastra. Simulation of cloud dynamics on graphics hardware. In *Proceedings of Conference on Graphics Hardware*, pages 92–101. ACM / Eurographics Association, 2003.
- [HG41] Louis G. Henyey and Jesse L. Greenstein. Diffuse radiation in the galaxy. *The Astrophysical Journal*, 93:70–83, 1941.
- [HH12] Roland Hufnagel and Martin Held. A survey of cloud lighting and rendering techniques. In *Proceedings of WSCG International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 205–216. Václav Skala-UNION Agency, 2012.
- [HHS07] Roland Hufnagel, Martin Held, and Florian Schröder. Large-scale, realistic cloud visualization based on weather forecast data. In *Proceedings of International Conference on Computer Graphics and Imaging*, pages 54–59. IASTED, 2007.

- [HL01] Mark J. Harris and Anselmo Lastra. Real-time cloud rendering. *Eurographics Computer Graphics Forum*, 20(3):76–85, 2001.
- [HW12] Lukas Hosek and Alexander Wilkie. An analytic model for full spectral sky-dome radiance. In *Proceedings of SIGGRAPH*, pages 95:1–95:9. ACM, 2012.
- [JMLH01] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. A practical model for subsurface light transport. In *Proceedings of SIGGRAPH*, pages 511–518. ACM, 2001.
- [Kol12] Timothy Kol. Analytical sky simulation. Technical report, Utrecht University, 2012.
- [KPH<sup>+</sup>03] Joe Kniss, Simon Premože, Charles Hansen, Peter Shirley, and Allen McPherson. A model for volume lighting and modeling. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):150–162, 2003.
- [KPHE02] Joe Kniss, Simon Premože, Charles Hansen, and David S. Ebert. Interactive translucent volume rendering and procedural modeling. In *Proceedings of Conference on Visualization*, pages 109–116. IEEE, 2002.
- [Len85] J. Lenoble. *Radiative Transfer in Scattering and Absorbing Atmospheres: Standard Computational Procedures*. A. Deepak Publishing, 1985.
- [LHN05] Sylvain Lefebvre, Samuel Hornus, and Fabrice Neyret. Octree Textures on the GPU. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*, chapter 37. Addison-Wesley Professional, 2005.
- [Lin09] Rebecca Lindsey. Clouds can reveal shape of continents: Image of the day. Web, November 2009. Consulted September 2013.
- [LJWH07] Shengjun Liu, Xiaogang Jin, Charlie CL Wang, and Kin-chuen Hui. Ellipsoidal-blob approximation of 3D models and its applications. *Computers & Graphics*, 31(2):243–251, 2007.
- [LL01] David K. Lynch and William Livingston. *Color and Light in Nature*. Cambridge University Press, 2001.
- [Mac11] Chris A Mack. Fifty years of Moore’s law. *IEEE Transactions on Semiconductor Manufacturing*, 24(2):202–207, 2011.
- [Mas71] B.J. Mason. *The physics of clouds*. Clarendon Press, 1971.
- [Mie08] Gustav Mie. Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen. *Annalen der Physik*, 330(3):337–445, 1908.
- [Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.
- [MSMI04] Nelson Max, Greg Schussman, Ryo Miyazaki, and Kei Iwasaki. Diffusion and multiple anisotropic scattering for global illumination in clouds. In *Proceedings of WSCG International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 277–284. UNION Agency - Science Press, 2004.
- [MYDN01] Ryo Miyazaki, Satoru Yoshida, Yoshinori Dobashi, and Tomoyula Nishita. A method for modeling clouds based on atmospheric fluid dynamics. In *Proceedings of Pacific Conference on Computer Graphics and Applications*, pages 363–372. IEEE, 2001.
- [NDN96] Tomoyuki Nishita, Yoshinori Dobashi, and Eihachiro Nakamae. Display of clouds taking into account multiple anisotropic scattering and sky light. In *Proceedings of SIGGRAPH*, pages 379–386. ACM, 1996.



- [Ney00] Fabrice Neyret. A phenomenological shader for the rendering of cumulus clouds. Technical Report RR-3947, INRIA, 2000.
- [Nic65] Fred E. Nicodemus. Directional reflectance and emissivity of an opaque surface. *OSA Applied Optics*, 4(7):767–773, 1965.
- [NN94] Tomoyuki Nishita and Eihachiro Nakamae. A method for displaying metaballs by using Bézier clipping. *Eurographics Computer Graphics Forum*, 13(3):271–280, 1994.
- [NR92] Kai Nagel and Ehrhard Raschke. Self-organizing criticality in cloud formation? *Physica A: Statistical Mechanics and its Applications*, 182(4):519–531, 1992.
- [NSTN93] Tomoyuki Nishita, Takao Sirai, Katsumi Tadamura, and Eihachiro Nakamae. Display of the earth taking into account atmospheric scattering. In *Proceedings of SIGGRAPH*, pages 175–182. ACM, 1993.
- [NVI06] NVIDIA. GeForce 8800. Web, November 2006. Consulted October 2013.
- [Org75] World Meteorological Organization. *International Cloud Atlas - Volume 1: Manual on the Observation of Clouds and Other Meteors*. World Meteorological Organization, 1975.
- [PARN04] Simon Premože, Michael Ashikhmin, Ravi Ramamoorthi, and Shree Nayar. Practical rendering of multiple scattering effects in participating media. In *Proceedings of Conference on Rendering Techniques*, pages 363–374. Eurographics Association, 2004.
- [PAS03] Simon Premože, Michael Ashikhmin, and Peter Shirley. Path integration for light transport in volumes. In *Proceedings of Workshop on Rendering*, pages 52–63. Eurographics Association, 2003.
- [Pas13] PassMark Software. PassMark Software - Video Card Benchmarks. Web, October 2013. Consulted October 2013.
- [PH89] Ken Perlin and Eric M. Hoffert. Hypertexture. In *Proceedings of SIGGRAPH*, pages 253–262. ACM, 1989.
- [PK97] H.R. Pruppacher and J.D. Klett. *Microphysics of Clouds and Precipitation*. Springer, 1997.
- [PSS99] Arcot J. Preetham, Peter Shirley, and Brian Smits. A practical analytic model for daylight. In *Proceedings of SIGGRAPH*, pages 91–100. ACM, 1999.
- [Ree83] William T. Reeves. Particle systems - a technique for modeling a class of fuzzy objects. In *Proceedings of SIGGRAPH*, pages 359–375. ACM, 1983.
- [REHL03] Kirk Riley, David S. Ebert, Charles Hansen, and Jason Levit. Visually accurate multi-field weather visualization. In *Proceedings of Conference on Visualization*, pages 279–286. IEEE, 2003.
- [REK<sup>+</sup>04] Kirk Riley, David S. Ebert, Martin Kraus, Jerry Tessendorf, and Charles Hansen. Efficient rendering of atmospheric phenomena. In *Proceedings of Conference on Rendering Techniques*, pages 375–386. Eurographics Association, 2004.
- [RS11] Karl Rupp and Siegfried Selberherr. The economic limit to Moore’s law. *IEEE Transactions on Semiconductor Manufacturing*, 24(1):1–4, 2011.
- [Sch95] Gernot Schaufler. Dynamically generated imposters. In *Proceedings of Workshop on Modeling - Virtual Worlds - Distributed Graphics*, pages 129–136. GI, 1995.
- [Sch09] Paul Schlyter. Radiometry and photometry in astronomy. Web, March 2009. Consulted September 2013.

- [SG31] T. Smith and J. Guild. The C.I.E. colorimetric standards and their use. *Transactions of the Optical Society*, 33(3):73, 1931.
- [SKTM11] László Szirmay-Kalos, Balázs Tóth, and Milán Magdics. Free path sampling in high resolution inhomogeneous participating media. *Eurographics Computer Graphics Forum*, 30(1):85–97, 2011.
- [SSEH03] Joshua Schpok, Joseph Simons, David S. Ebert, and Charles Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of Symposium on Computer Animation*, pages 160–166. ACM / Eurographics Association, 2003.
- [Str71] J.W. Strutt. On the scattering of light by small particles. *Philosophical Magazine*, 41(275):447–454, 1871.
- [TB02] Andrzej Trembilski and Andreas Broßler. Surface-based efficient cloud visualisation for animation applications. In *Proceedings of WSCG International Conferences in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 453–460, 2002.
- [Wan04] Niniane Wang. Realistic and fast cloud rendering. *Journal of Graphics Tools*, 9(3):21–40, 2004.
- [Wes90] Lee Westover. Footprint evaluation for volume rendering. In *Proceedings of SIGGRAPH*, pages 367–376. ACM, 1990.
- [Wes91] Lee Alan Westover. *Splatting: a parallel, feed-forward volume rendering algorithm*. PhD thesis, University of North Carolina at Chapel Hill, 1991.
- [Wil78] Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH*, pages 270–274. ACM, 1978.
- [Wil83] Lance Williams. Pyramidal parametrics. In *Proceedings of SIGGRAPH*, pages 1–11. ACM, 1983.
- [WT90] Geoff Wyvill and Andrew Trotman. *Ray-tracing soft objects*. Springer, 1990.
- [WWW79] Peter Wendling, Renate Wendling, and Helmut K. Weickmann. Scattering of solar radiation by hexagonal ice crystals. *OSA Applied Optics*, 18(15):2663–2671, 1979.
- [YIC<sup>+</sup>10] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. Unbiased, adaptive stochastic sampling for rendering inhomogeneous participating media. In *Proceedings of SIGGRAPH Asia*, pages 177:1–177:8. ACM, 2010.
- [ZRL<sup>+</sup>08] Kun Zhou, Zhong Ren, Stephen Lin, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Real-time smoke rendering using compensated ray marching. In *Proceedings of SIGGRAPH*, pages 36:1–36:12. ACM, 2008.