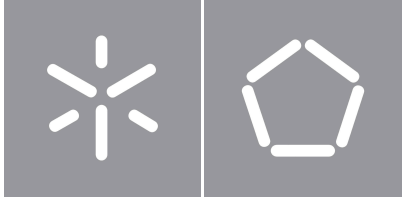
University of Minho
School of Engineering

Mariana Lino Lopes Costa

Cloud Modeling and Rendering

july 2023



University of Minho
School of Engineering

Mariana Lino Lopes Costa

Cloud Modeling and Rendering

Masters Dissertation
Integrated Master's in Informatics Engineering

Dissertation supervised by
Professor António José Borba Ramires Fernandes

july 2023

Copyright and Terms of Use for Third Party Work

License granted to users of this work:

Acknowledgements

I would like to sincerely and gratefully thank everyone who contributed directly and indirectly to this journey.

I would like to start by thanking Professor António Ramires, for showing me the world of Computer Graphics, challenging me on this path, and always being available to clarify my doubts about the dissertation.

I would like to thank my family, my father, mother, brother, and sister, for supporting me unconditionally, being there for me in the most difficult moments of this journey, and being my greatest strength throughout these years. For always believing in me and motivating me not to give up on this dream. I would also like to thank my maternal grandmother, paternal grandfather, and godfather for being there whenever I needed them. To my maternal grandfather and paternal grandmother, who although did not see the end of my academic journey, would be very proud to know that I managed to get here. A special thank you to my entire family. Thank you to my cats for being my company throughout these academic years and also during the completion of this dissertation.

I would like to thank Francisco, my true friends, and all my friends that the university provided me with. I appreciate the support, phone calls, video calls, study and work nights, for all the relaxing moments that were essential in the midst of so much anxiety and stress. For being there to celebrate small victories, cheering me up in my failures, and bringing me so much inspiration and motivation throughout these years of university. To everyone who in some way made this academic experience something unforgettable and that I will always remember with much love and joy. I am truly grateful to have had you by my side. A big thank you from the bottom of my heart.

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

University of Minho, Braga, july 2023

Mariana Lino Lopes Costa

"*If you can dream it, you can do it.*" (Walt Disney)

Abstract

Cloud Modeling and Rendering

Clouds play an important role in enhancing the realism of ambient and outdoor scenes in Computer Graphics (CG). However, rendering clouds and certain atmospheric elements remains a major challenge in this field. Their representation involves processes such as modeling, photorealistic rendering, and animation.

The main objective of this project was to explore techniques for modeling and rendering clouds. To this end, code was developed in GLSL shader language to implement clouds, using the NAU3D application for code compilation.

Different types of cloud modeling were studied, based on textures, geometric shapes, noise, real and/or satellite images. Volumetric clouds were implemented, following the approach of Schneider and Vos [2015], Hillaire [2016a], and Högfeldt [2016]. A volume was used in which the voxels were filled with information stored in a Weather texture.

Cloud rendering was performed using the Ray marching algorithm. For cloud lighting, a study of different light events was conducted independently of the participating media. Lighting in this implementation was adapted from formulas presented in Jarosz [2008] and Scratchapixel [2022b], taking into account that the participating media is the cloud. Various light phenomena were studied, including in-scattering. To address this effect, several phase functions were tested.

Tests were also conducted to evaluate computational costs, such as the time spent on GPU rendering of the implemented cloud shader and the required FPS for rendering the atmosphere with clouds. These tests were performed to understand the balance between computational cost and visual results, allowing for customization of parameters based on this balance. The visual results obtained did not reach the level of state-of-the-art realistic clouds in CG, but there is room for improvement in the implemented clouds.

Keywords Clouds, Computer Graphics, Cloud rendering, Atmosphere, Volumetric, Cloud modeling

Resumo

Modelação e Renderização de Nuvens

As nuvens representam um papel importante no aumento do realismo de ambientes e cenas externas em computação gráfica (CG). A renderização de nuvens e alguns elementos da atmosfera ainda representam um grande desafio nesta área. A sua representação envolve processos como modelação, renderização fotorrealista e animação. O principal objetivo deste projeto foi explorar as técnicas de modelação e renderização de nuvens. Neste sentido foi desenvolvido código em linguagem de *shaders* GLSL para se implementar nuvens. A aplicação usada para compilar o código foi a NAU3D. Estudou-se os diferentes tipos de modelação de nuvem baseados em texturas, figuras geométricas, ruídos, imagens reais e/ou de satélite. Optou-se por se implementar nuvens volumétricas, seguindo a abordagem de Schneider and Vos [2015], Hillaire [2016a] e Högfeldt [2016]. Utilizou-se um volume no qual os *voxels* foram preenchidos com informação armazenada numa *Weather texture*. A renderização das nuvens foi feita através do algoritmo de *Ray marching*. Para a iluminação da nuvem, foi realizado um estudo dos diferentes eventos de luz independentemente do *participating media*. Para elaboração da iluminação nesta implementação, foram adaptadas as fórmulas apresentadas em Jarosz [2008] e Scratchapixel [2022b], tendo em conta que o *participating media* é a nuvem. Vários fenómenos da luz foram estudados, incluindo o *in-scattering*. Para tratar esse efeito foram testadas várias funções de fase. Foram também realizados testes para se avaliar o custo computacional, como o tempo em GPU gasto apenas na renderização do *shader* das nuvens implementadas e os FPS requeridos para a renderização da atmosfera com as nuvens. Estes testes foram feitos de forma a se perceber o balanço entre o custo computacional e os resultados visuais, permitindo assim personalizar os parâmetros tendo em conta este equilíbrio. Os resultados visuais obtidos não atingiram o nível do estado de arte de nuvens realistas em CG, mas existe margem para as nuvens implementadas serem melhoradas.

Palavras-chave Nuvens, Computação Gráfica, Renderização de nuvens, Atmosfera, Volumes, Modelação de nuvens

Contents

List of Figures

List of Tables

List of Acronyms

2D Two-dimensional. x, 20, 23, 24, 27, 28, 29, 32, 36, 37, 38, 42, 50, 65

3D Three-dimensional. ix, 20, 22, 27, 28, 29, 40, 44

CG Computer Graphics. v, vi, 1

CPU Central Processing Unit. 53

FBM Fractal Brownian Motion. 26, 41, 42

FHD Full high-definition. 60, 61, 62

FPS Frames Per Second. v, vi, xi, 53, 55, 57, 58, 59, 61, 62

GHz Gigahertz. 53

GLSL OpenGL Shading Language. v, vi, 52

GPU Graphics Processing Unit. v, vi, 25, 53, 54, 55, 56, 57, 58, 61, 62, 67

HD High Definition. 60

HD+ High Definition Plus. 60

HG Henyey-Greenstein. 14, 62

km Kilometre. 5, 7, 8, 33

nHD Ninth High Definition. 60

PC Personal Computer. 52, 66

qHD Quad High Definition. 60

QVGA Quarter Video Graphics Array. 60, 62

RGB Red Green Blue. x, xii, 27, 29, 32, 38, 42, 52

RGBA Red Green Blue Alpha. 31, 52

RGBA32F Red Green Blue Alpha 32 Floating-point. 52

RPNN Reduced Probabilistic Neural Network. 25

RTE Radiative Transfer Function. 18, 33, 35

RTX Ray Tracing Texel eXtreme. 53

Super VGA Super Video Graphics Array. 60

VGA Video Graphics Array. 60

WVGA Wide Video Graphics Array. 60

WXGA Wide Extended Graphics Array. 60, 62

XGA Extended Graphics Array. 60

Chapter 1

Introduction

1.1 Contextualization

Computer Graphics (CG) is the area of computing focused on the generation of images to represent data and information or to recreate the real world. Nowadays, computer graphics is a core technology in several areas like digital photography, film, video games, animation films, cell phones and computer displays, and many specialized applications.

In the last two decades, there has been a huge advance in technology and consequently an extraordinary development in this area. Today, some images are almost indistinguishable from reality. But despite this, there are still plenty of challenges in this area, among them, the rendering of realistic clouds and all the weather environment. The clouds have an important place when it comes to creating a realistic outdoor scene. The real-time rendering of realistic and convincing cloud scenes has always been a desired feature in CG.

The cloud implementation covers three important areas: modeling, rendering, and animation of clouds. This master's dissertation only addresses the first two phases: modeling, and rendering. Modeling is the process of making the object shape so that it approximates the appearance of the real. There are several modeling techniques, such as mesh-based and volumetric methods, generating shapes from texture or noise and even from terrestrial or satellite images. The rendering of the cloud is divided into two essential phases: the lighting and the rendering techniques. The lighting captures the illumination or interaction of cloud particles with the surrounding impinging light and the rendering techniques employs ray casting, rasterization, or a variant to convert the existing shape representation and light setting to display the final cloud on the screen.

1.2 Objectives

Several techniques and algorithms can be used for implementing clouds. There are different techniques for the different cloud implementation phases: modeling, rendering, and animation of clouds.

The main objective of this master's dissertation was to explore cloud modeling and rendering. The cloud implementation follows the approach of Hillaire [2016a], Högfeldt [2016], and Schneider and Vos [2015] for the modeling and Jarosz [2008] and Scratchapixel [2022b] for the illumination.

To achieve the main objective, it is necessary to understand and study all the nature, the sky, and the physiology of the clouds. Then, the techniques and algorithms were studied and compared to understand their advantages and disadvantages. It is intended to implement clouds based on one technique and create realistic clouds.

1.3 Document Structure

From a structural point of view, this dissertation work includes, in addition to the present chapter, four main chapters. A brief description of the content of each of these chapters is presented below.

- Chapter 2: This chapter focuses on introducing the general theme of clouds, presenting some characteristics and the main types of clouds. This chapter also describes general lighting events in participating media, giving special attention to cloud lighting. The evolution of different cloud modeling and rendering techniques is mentioned and briefly explained, to end the chapter.

- Chapter 3: This chapter is related to the previous one, given that through this it was possible to take a set of necessary decisions to implement clouds. It shows all the necessary steps for the idealization and implementation of realistic clouds. This chapter includes the explanation of the cloud modeling process, cloud rendering, and lighting.

- Chapter 4: In this chapter, the final visual cloud implementation results are presented. In addition, some required resources and run-time performance tests are analyzed.

- Chapter 5: In the last stage of the dissertation work, the final conclusions are presented. This includes the problems that were solved, their results, and the challenges that can be addressed in future work.

Chapter 2

State of the Art

In Chapter 2 we present some of the recent developments, studies, and techniques used to generate volumetric realistic clouds. Section 2.1 starts by showing the types of cloud and their phenomenology to familiarize some terms used in the following sections. Section 2.2 presents the cloud illumination.

Cloud generation techniques are an extensively researched area that is constantly evolving. Over the last few years, innovative ideas and advances have emerged on how to render clouds almost indistinguishable from reality. Several techniques have been developed and proposed by the Computer Graphics community. The growth of modern hardware has also enabled development of techniques that can achieve cloud display and animation at interactive frame rates. So, different techniques were being improved, evolved, and/or conjugated differently, with the aim of spending as few resources as possible and at the same time presenting the most realistic visual result.

Section 2.3 presents some resources and techniques used in the last years to model shapes and render clouds.

2.1 Cloud Types

This section offers an introduction to clouds. It is important to present certain cloud-related terms, such as the type of clouds, their lightning and their physical properties.

The planet is enveloped by the atmosphere, a variety of gases. The heat of the sun reaches the surface of planet Earth, causing water to evaporate. As the hot air is lighter, it rises into the atmosphere and the cloud is formed by transforming steam into small drops of water or ice crystals. Water vapor is invisible to the human eye until water condenses and consequently, the formation of the cloud occurs. The wind is also an indispensable factor in cloud generation and it is a force that can put different pressures on the cloud depending on the part of the atmosphere where they are located.

The clouds can appear in various shapes and display a large variety of types. Some cloud examples are

shown in Figure 1. In Hufnagel and Held [2012] are refereed and described parameters that characterize clouds and how they can help to classify cloud types.

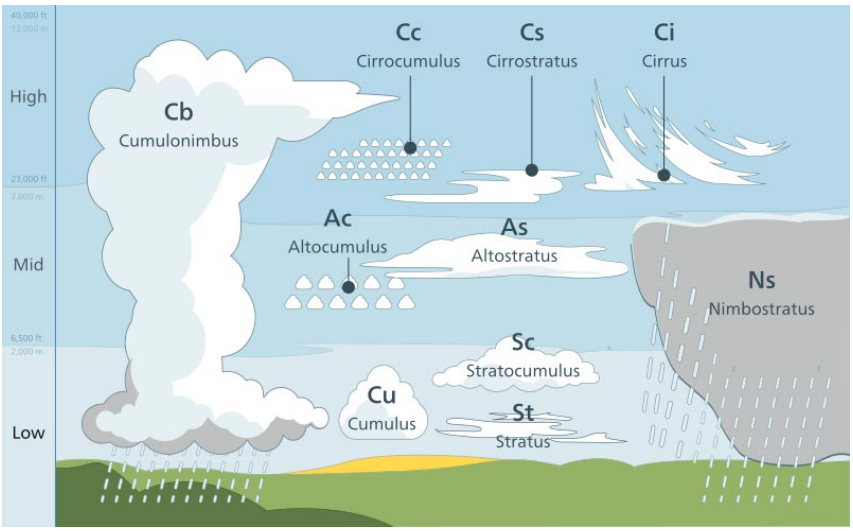


Figure 1: Representation of the main types of clouds. Illustration by *Valentin de Bruyn*.

Clouds can have different **geometry**. Often clouds can appear with a cauliflower-like texture above a sharp cloud base, so-called Cumulus. Clouds can form different layers. Low clouds can mix forms of convective and layered clouds.

The **boundary appearance** of clouds depends mainly on the constituents of the clouds' volume: droplets of different sizes or ice particles. While large droplets of water produce a limit-determined surface, the smaller ones create a diffuse boundary or fractal. Ice particles usually form clouds similar to hair or fibers. Depending on the observer's distance, the appearance of a cloud can differ. The distant clouds usually seem to have a distinct surface with sharp contours. However, a closer look shows diffuse or fractal structures.

Optical phenomena due to the water droplets and ice crystals that scatter light occurs. All clouds are white. Gray tones can appear because of cloud self-shadowing. Moreover, depending on the cloud **density**, clouds can appear darker if they have bigger water particle sizes (rainy clouds). The inter-reflection of light can brighten or darken some parts of the cloud.

When the light is behind the clouds, a **corona** effect can emerge. The clouds' edges produce a bright silhouette (silver-lining).

Clouds usually have vivid colors, because of the scattering of light outside the cloud volume on air molecules and aerosols, the **atmospheric scattering**. The sun and skylight color can influence the final color of the cloud. For example, at sunrise and sunset, clouds can appear in yellowish and reddish

colors. The ground can create an inter-reflection in some low-level clouds.

Clouds reside in a layer above the earth's surface in the atmosphere, called the troposphere. They are encountered over a range of **altitudes** varying from sea level to the top of the troposphere. The cloud **altitude** is the vertical distance from mean sea level to the point being measured (base of cloud). Clouds can differ in **size**, they can have different **vertical extents**, vertical distance from a cloud's base to its top, and different **heights**, the vertical distance from the point of observation on the Earth's surface to the point being measured. The horizontal extension can vary from a few hundred meters to thousands of kilometers (Hufnagel and Held [2012]).

The cloud can appear in several forms and variations depending on a combination of the aforementioned attributes. A characteristic that differentiates the various types of clouds is the altitude at which they form (low, mid, and high altitude). It is important to note that this altitude varies according to the geographical (latitudinal) position of the region under consideration (Buthiran [2004], World Meteorological Organization (WMO) [2022]).

There are four main designations for clouds where its base is found in the low stage, 2 km from the Earth's surface, represented in Figure 2:

- **Cumulus** (Figure 2a): isolated clouds which have a horizontal base, well-defined contours, white color when illuminated by the sun, consisting mainly of droplets of water, may contain ice crystals at the top, usually have a swollen appearance.

- **Cumulonimbus** (Figure 2b): Clouds with great vertical development present the shape of a mountain and its shape can only be seen from afar due to its enormous size. Their base is in the low level, but these clouds can extend to the other two levels. Their top may reach high levels. On the top, it usually has the characteristic shape of an anvil. It's a darker cloud formed by great lakes of water and hail, which may contain ice crystals at the top. It usually arises from the development of cumulus clouds, due to the wind.

- **Stratocumulus** (Figure 2c): low clouds with rounded masses but with related flat top and base. They can be gray or whitish clouds, they are formed by water droplets and are associated with rainfall weak.

- **Stratus** (Figure 2d): very low clouds that usually cover up the whole sky. Strong gray cloud with a uniform base, like a "carpet" in the sky.

The three denominations for mid-stage clouds exist, 2 to 8 km in tropical latitude, 2 to 7 km in a temperate region, and 2 to 4 km in the polar region (Figure 3):

(a) *Cumulus*. Source:*Jim Corwin*. (b) *Cumuloninbus*. Source:*nabaseshop.com*.

(c) *Stratocumulus*. Source:*WhatsThisCloud*. (d) *Stratus*. Source:*WhatsThisCloud*.

Figure 2: Three simple graphs

- **Altocumulus** (Figure 3a): gray cloud (sometimes white) that presents own shadows and takes the form of fibrous rolls or diffuse sheets.
- **Altostratus** (Figure 3b): they resemble a gray sheet. They always have thin parts that allow seeing the sun. Altosatrus is usually found in the middle level, but it often extends to higher levels.
- **Nimbostratus** (Figure 3c): a cloud of very dark color and large diffuse extension and base formed by raindrops. Nimbostratus is almost always found in the middle level, but it usually extends into the other two levels.

(a) *Altocumulus*. Source: *flickr.com*.

(b) *Altostratus*. Source: *Famartin*

(c) *Nimbostratus*. Source: *WhatsThisCloud*.

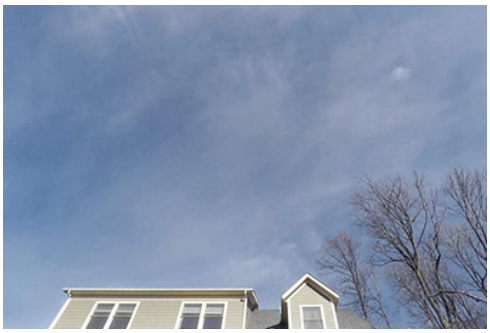
Figure 3: Mid-stage clouds.

Finally, there are also three denominations for cloud formations in high stages, that is, 6 to 18 km in the tropical region, 5 to 13 km in the temperate region, and 3 to 8 km in the polar region (Figure 4):

- **Cirrus** (Figure 4a): clouds with a silky shine, isolated and formed by ice crystals appearing to converge to the horizon.
- **Cirrocumulus** (Figure 4b): fathom clouds composed almost exclusively of ice crystals grouped into granules semi-transparent.
- **Cirrostratus** (Figure 4c): clouds that look like a transparent veil.

(a) *Cirrus*. Source: *Simon P.*.

(b) *Cirrocumulus*. Source: *Ken Varnum*



(c) *Cirrostratus*. Source: *WhatsThisCloud*

Figure 4: High-stage clouds.

Table 1 shows approximate cloud altitudes of low, middle and high level in troposphere relative to the Earth's surface, and the cloud type occurring in each one.

Table 1: Cloud types altitudes.

Level	**Type**	**Tropical region**	**Temperate region**	**Polar region**
Low	Stratus, Stratocumulus, Cumulus, Cumulonimbus	0 - 2 km	0 - 2 km	0 - 2 km.
Middle	Nimbostratus, Altostratus, Altocumulus	2 – 8 km	2 – 7 km	2 – 4 km
High	Cirrus, Cirrocumulus, Cirrostratus	6 – 18 km	5 – 13 km	3 – 8 km

2.2 Light Transport in Participating Media

Participating media is the term used in computer graphics to describe volumes filled with particles. It is a material property that affects the transport of light through its volume (for example, glass, water, fog, **clouds**). When a photon travels through particles, it may interact with some of them. When an interaction occurs, two events may happen: the photon may be absorbed by the particle or may be scattered in another direction.

Materials and objects may absorb and reflect light. For instance, when a material absorbs all frequencies of light equally, it looks black. When a material reflects all the light, it seems white (Scratchapixel [2022a]).

Particles can have various sizes: they can be large impurities (e.g., dust, air pollution, water droplets, sand, aerosols) or they can be small (e.g., air molecules). The light scattering depends on the properties of the particles, their size, and their volume density. Accordingly, the participating media interacts differently with light traveling through it and bouncing on particles.

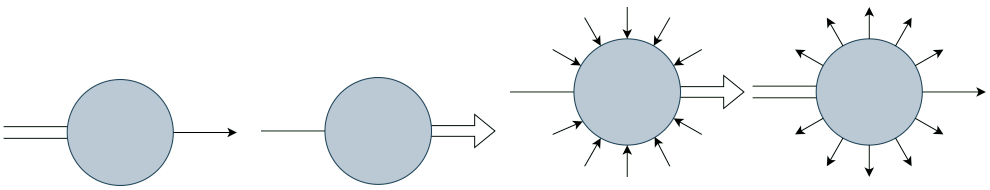
Clouds are filled with particles and receive lighting from several sources: the light of the sun, sky, ground, and other particles. Thus, cloud illumination is related to light from the entire atmosphere. The atmosphere is also made up of various small particles which sometimes mix with larger ones from dust, sand raised by the wind, or even due to air pollution. The larger particles influence and hinder light scattering.

The study of light scattering is divided into two categories (Nishita et al. [1996]):

- Rayleigh scattering (air molecules): it depends on the wavelength and it is responsible for the blue color of the sky and the orange colors of sunrise and sunset;
- Mie scattering (aerosols): it does not differentiate individual wavelength colors, resulting in equally scattered light from the sun, creating the white color.

The white/gray color of the clouds is caused by Mie scattering in water droplets, but clouds do not always appear white. Due to haze and dust in the atmosphere, clouds may appear more yellow, orange, or red. If clouds become denser (higher concentration of water droplets), sunlight does not pass through the clouds as easily, making them darker (gray color). When there is no direct sunlight striking the cloud, it may reflect the color of the sky and appear bluish.

The next subsections present different light events that can happen when light travels through a participating media: absorption, scattering, extinction, transmittance, emission, and radiative transfer. Some of



(a) Absorption (b) Emission (c) In-Scatter (d) Out-Scatter

Figure 5: Light transport events.

these events are related to cloud lighting. Figure 5 represents the main four light transport events (Jarosz [2008]): absorption, emission, in-scatter and out-scatter.

For the next subsections and to understand the formulas, it is necessary to consider Equation 2.1:

$$\delta_n(x, \vec{\omega}) = (\vec{\omega} \cdot \nabla_n) L(x, \vec{\omega}) \tag{2.1}$$

where $\delta_n(x, \vec{\omega})$ is the change in radiance due to the light event n, ∇_n denotes the gradient due to the light event (which is analogous to the concept of derivative), $(\vec{\omega} \cdot \nabla_n)$ is the directional derivative along the direction $\vec{\omega}$ and, $L(x, \vec{\omega})$ describes the radiance that starts in position x and follows the direction $\vec{\omega}$.

2.2.1 Absorption

Photons that travel through participating media can be absorbed. The relative probability that a photon is absorbed is determined by the absorption coefficient represented by σ_a. The absorption can be illustrated in Figure 5a, where an object receives (input) more light than it sends (output).

Equation 2.2 gives the reduced radiance due to absorption:

$$\delta_a(x, \vec{\omega}) = -\sigma_a(x) L(x, \vec{\omega}) \tag{2.2}$$

where $\delta_a(x, \vec{\omega})$ is the change in radiance due to absorption, σ_a is the absorption coefficient, and $L(x, \vec{\omega})$ describes the radiance that starts in position x and follows the direction $\vec{\omega}$.

Clouds with rain have more and larger water particles, so they absorb more light. Thus, rain clouds are darker because the light is more absorbed (Figure 6).

(a) White clouds with a lower absorption coefficient. Source: *Eva Mijalkovic*.

(b) Rain clouds with a larger absorption coefficient. Source: *Alim Mohammed*.

Figure 6: Clouds with different absorption coefficient.

2.2.2 Scattering

Scattering describes the event where light travels through a volume in a certain direction and the photons are deflected in other directions when they collide with particles (Figure 5c, Figure 5d).

The scattering coefficient, σ_s, is related to the probability that a photon is scattered when traveling through a volume.

Clouds are, in general, white because they scatter light independently of the wavelength. The sky is blue since atmospheric scatter blue wavelengths more than others. Two examples of scattering phenomena in clouds are Out-Scatter and In-Scatter effects (Figure 7).

(a) In-Scatter cloud effect. Source: *Flowermanjoe*.

(b) Out-Scatter cloud effect. Source: *André Moraes*.

Figure 7: In-scatter and Out-scatter cloud effect.

Out-Scatter When the light is traveling towards the eye with a direction $\vec{\omega}$, it can be scattered in different directions (Figure 5d). The amount of light that enters the cloud is reduced along a ray and scattered in

various directions, creating a cauliflower effect, white clouds with darker edges highlighting the billowy shapes (Figure 7b).

Equation 2.3 expresses the out-scatter event, the change in radiance due to out-scattering:

$$\delta_o(x, \vec{\omega}) = -\sigma_s(x)L(x, \vec{\omega}) \tag{2.3}$$

where $\delta_o(x, \vec{\omega})$ is the change in radiance due to out-scattering, σ_s is the scatter coefficient, $L(x, \vec{\omega})$ describes the radiance that starts in position x and follows the direction $\vec{\omega}$.

In-Scatter Some photons are deflected away from the viewing direction, and some others are deflected along the viewing direction, this event is known as In-scatter (Figure 5c).

In-scattering accounts for increased radiance due to scattering from other directions. This scattering causes some of the light to be deflected back along the viewer's direction. If the light source is behind the cloud (relative to the eye), it can pass through it towards the eye, and the effect of in-scattering is better observed. The edges of the cloud get more light and the cloud gets darker. This effect is often known as a silver-lining effect (Figure 7a).

Equation 2.4 expresses the in-scatter event, the change in radiance due to in-scattering:

$$\delta_i(x, \vec{\omega}) = \sigma_s(x)L_i(x, \vec{\omega}) \tag{2.4}$$

where $\delta_i(x, \vec{\omega})$ is the change in radiance due to in-scattering, σ_s is the scattering coefficient and $L_i(x, \vec{\omega})$ is all the light which is scattered into the thin beam x along direction $\vec{\omega}$.

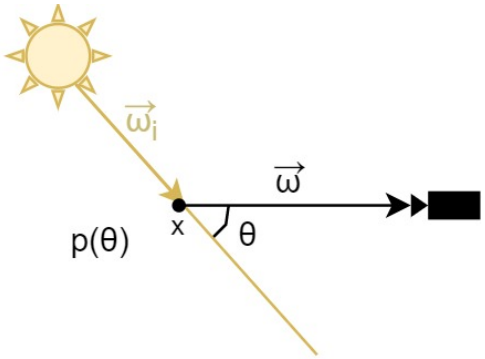


Figure 8: Phase function representation.

In Figure 8, it is possible to better understand the phase function, light direction, and viewer direction. Equation 2.5 represents all the light scattered:

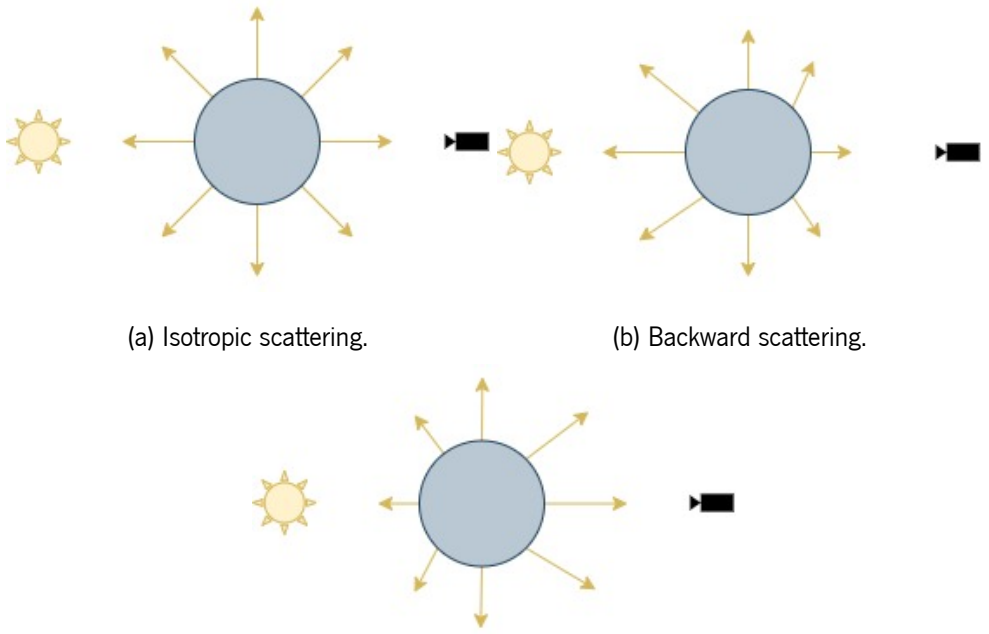
$$L_i(x, \vec{\omega}) = \int_{\Omega_{4\pi}} p(\theta)L(x, \vec{\omega}_i)d\vec{\omega}_i \tag{2.5}$$

where, $L_i(x, \vec{\omega})$ is all the light that is scattered into the thin beam x along direction $\vec{\omega}$, $p(\theta)$ is the phase function, θ is the angle between the light incident direction $\vec{\omega}_i$ and the viewing direction $\vec{\omega}$, $L(x, \vec{\omega}_i)$ is the incident radiance that reaches the beam x from direction $\vec{\omega}_i$. This computation involves integrating incident radiance over the whole sphere of directions $\Omega_{4\pi}$.

In computer graphics, the scattering phenomenon is simplified by using a mathematical model called a phase function. This function is responsible for the angular distribution of scattered light.

Phase function

So, the phase function gives the fraction of incoming light traveling along the direction $\vec{\omega}$. The distribution of scattered light depends on the medium's properties and the angle, θ, between the light direction vector and the view direction vector. There are two types of scattering behavior that a participating medium can exhibit: isotropic or anisotropic (Figure 9). In isotropic scattering, the distribution is equal in different directions. In anisotropic scattering, some directions in the sphere of directions are more likely to be chosen than others, preferentially in the backward or forward directions. The medium is isotropic if $g = 0$ (Figure 9a). When $g < 0$, light is scattered backward (Figure 9b) and, when $g > 0$, light is scattered forward (Figure 9c).



(a) Isotropic scattering.

(b) Backward scattering.

(c) Forward scattering.

Figure 9: Types of scattering behavior.

Isotropic phase function An isotropic phase function describes equal scattering in all directions.

Equation 2.6 represents an isotropic phase function:

$$p_I(\theta) = \frac{1}{4\pi} \tag{2.6}$$

where p_I is the isotropic phase function, θ is the angle between the light incident direction and the viewing direction.

Anisotropic phase function An anisotropic phase function has two input parameters to shape a scattering profile: the mean cosine, also known as g, which describes the degree of anisotropy of the phase function, and the angle between the light incident direction and the viewing direction.

Equation 2.7 represents the Henyey-Greenstein (Henyey and Greenstein [1941]) phase function equation:

$$p_{HG}(\theta, g) = \frac{1 - g^2}{4\pi(1 + g^2 - 2g \times \cos(\theta))^{\frac{3}{2}}} \tag{2.7}$$

where p_{HG} is the phase function of Henyey-Greenstein, θ is the angle between the light incident direction and the viewing direction, and g is the mean cosine that controls the preferential scattering direction.

The HG phase function is represented in the graph of Figure 10 for $g = -0.5$, $g = 0$, and $g = 0.5$.

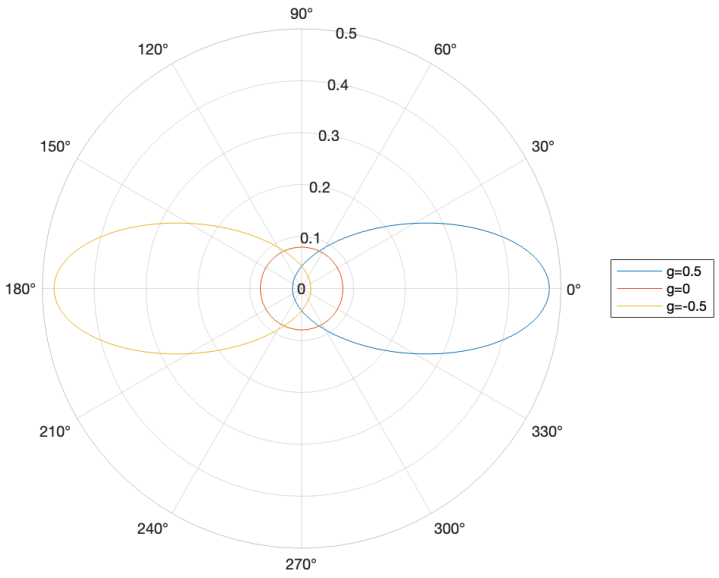


Figure 10: Henyey-Greenstein phase function representation.

Equation 2.8 represents the Cornette-Shank (Cornette and Shanks [1992]) phase function:

$$p_{CS}(\theta, g) = \frac{3(1 - g^2)}{8\pi(2 + g^2)} \frac{1 + \cos^2(\theta)}{(1 + g^2 - 2g \times \cos(\theta))^{\frac{3}{2}}} \tag{2.8}$$

where p_{CS} is the phase function of Cornette-Shank, θ is the angle between the light incident direction and the viewing direction, and g is the mean cosine that controls the preferential scattering direction.

The Cornette-Shank phase function is represented in Figure 11 for $g = -0.5$, $g = 0$, and $g = 0.5$.

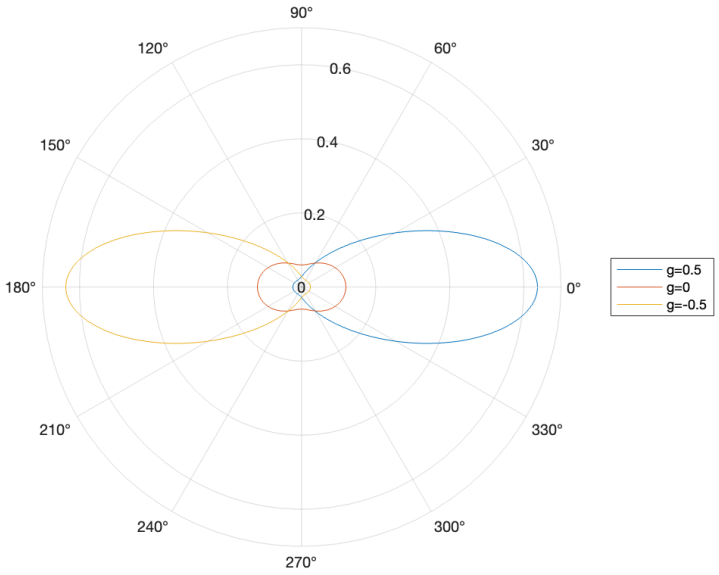


Figure 11: Cornette-Shank phase function representation.

Equation 2.9 represents the Schlick phase function (Blasi et al. [1993]):

$$p_S(\theta, k) = \frac{1 - k^2}{4\pi(1 + k\cos(\theta))^2} \text{ with } k \approx 1.55g - 0.55g^3 \tag{2.9}$$

where p_{CS} is the phase function of Schlick, θ is the angle between the light incident direction and the viewing direction, and g is the mean cosine that controls the preferential scattering direction.

The Schlick phase function is represented in Figure 12 for $g = -0.5$, $g = 0$, and $g = 0.5$.

Equation 2.10 represents the Rayleigh (Rayleigh [1871]) phase function:

$$p_R(\theta) = \frac{3}{4} \times (1 + \cos^2(\theta)) \tag{2.10}$$

where p_R is the phase function of Rayleigh, and θ is the angle between the light incident direction and the viewing direction.

The Rayleigh phase function is represented in the graph of Figure 13.

2.2.3 Extinction

The probability that photons interact with the participating media when they travel through it is related to the extinction coefficient σ_t. The probability that a photon causes the reduction in radiance depends on

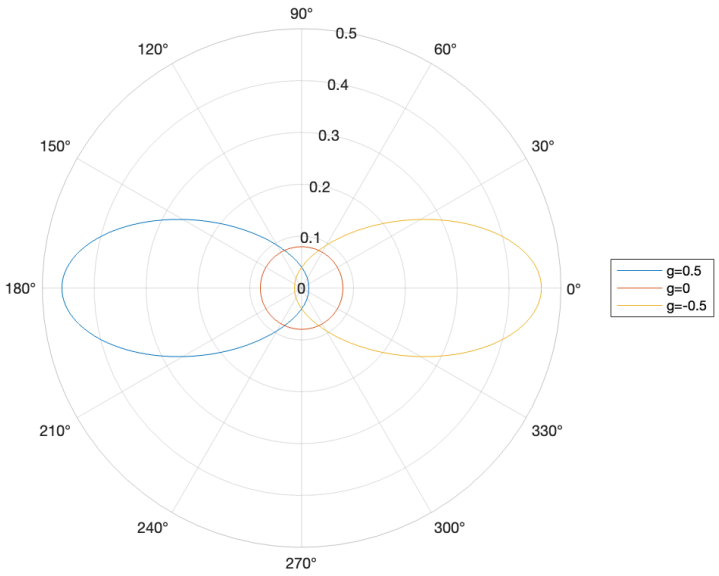


Figure 12: Schlick phase function representation.

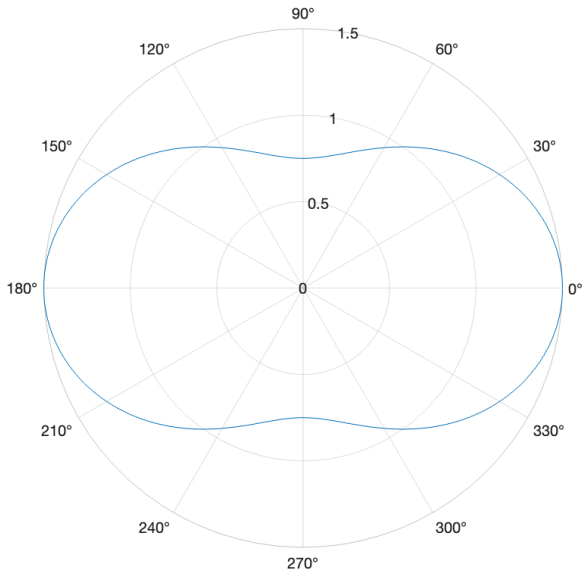


Figure 13: Rayleigh phase function representation.

the sum of the out-scatter and absorption, as described in Equation 2.11:

$$\sigma_t = \sigma_a + \sigma_s \tag{2.11}$$

where σ_t is the extinction coefficient, σ_a is the absorption coefficient, and σ_s is the scatter coefficient.

2.2.4 Transmittance

Transmittance is the ratio between the light received by the viewer and the light emitted by the light source (the incident light), i.e. the fraction of light that is transmitted through the volume. It is possible to consider the transmission as a measure of the volume's opacity, a measure of how much light can pass through a volume. The transmittance can be calculated by Equation 2.12:

$$T_r(d) = e^{-\int_0^d \sigma_t(x+t\vec{\omega})dt} \tag{2.12}$$

where $T_r(d)$ is the transmittance, $x + t\vec{\omega}$ with $t \, \epsilon[0, d]$ parameterizes the line segment between x_0 and x_1, and σ_t is the extinction coefficient.

In a homogeneous participating media, where σ_t is a constant, the transmittance can be computed by Equation 2.13. This equation is known as *Beer–Lambert's law*:

$$T_r(d) = e^{-\sigma_t d} \tag{2.13}$$

where $T_r(d)$ is the light transmittance along the distance d between two points in the volume, and σ_t is the extinction coefficient.

Equation 2.14 expresses how radiance is reduced as it travels from x to y:

$$L(x, \vec{\omega}) = T_r(d)L(y, \vec{\omega}_i) \tag{2.14}$$

where $L(x, \vec{\omega})$ describes the radiance that starts in position x and follows the direction $\vec{\omega}$, $T_r(d)$ is the transmittance, d is the distance between the two points x and y, and $L(y, \vec{\omega}_i)$ describes the radiance that starts from position y with direction $\vec{\omega}_i$.

2.2.5 Emission

The process of increased radiance due to other forms of energy is called emission. In emission, the amount of output light is larger than the input light (Figure 5b). This is because the participating media can emit light when it reaches high heat, e.g. fire. The emission can be expressed by Equation 2.15:

$$\delta_e(x, \vec{\omega}) = \sigma_a(x)L_e(x, \vec{\omega}) \tag{2.15}$$

where $\delta_e(x, \vec{\omega})$ is the change in radiance due to emission, σ_a is the absorption coefficient, and $L_e(x_t, -\vec{\omega})$ is the radiance resulting from the emission at the start of the beam x along direction $\vec{\omega}$.

2.2.6 Radiative Transfer Function

The different events of how light can interact with participating media have been described in the previous sections. It is possible to form a complete model of how light behaves in a participating medium by combining the equations of these events: absorption, scattering, transmittance, and emission (Equation 2.16):

$$\delta(x,\vec{\omega}) = \underbrace{\underbrace{\delta_a(x,\vec{\omega})}_{\text{absorption}} \underbrace{+\delta_o(x,\vec{\omega})}_{\text{out-scattering}}}_{\text{extinction}} \underbrace{+\delta_e(x,\vec{\omega})}_{\text{emission}} \underbrace{+\delta_i(x,\vec{\omega})}_{\text{in-scattering}} \tag{2.16}$$

where $\delta(x,\vec{\omega})$ is the total change in radiance at position x along direction $\vec{\omega}$, $\delta_a(x,\vec{\omega})$ is the change in radiance due to absorption (described in Subsection 2.2.1), $\delta_o(x,\vec{\omega})$ is the change in radiance due to out-scattering (described in Subsection 2.2.2), $\delta_e(x,\vec{\omega})$ is the change in radiance due to emission (described in Subsection 2.2.5), and $\delta_i(x,\vec{\omega})$ is the change in radiance due to in-scattering (described in Subsection 2.2.2).

The total change in radiance along the ray at a position x, also known as the integro-differential form of the radiative transfer or Radiative Transfer Function (RTE) (Chandrasekhar [2013], Jarosz [2008], Hillaire [2016a], Scratchapixel [2022b]), can be expressed by Equation 2.17. This equation shows that rate radiance changes as we take a step in the light flow direction:

$$(\vec{\omega}\cdot\nabla)L(x,\vec{\omega}) = \underbrace{\underbrace{-\sigma_a(x)L(x,\vec{\omega})}_{\text{absorption}} \underbrace{-\sigma_s(x)L(x,\vec{\omega})}_{\text{out-scattering}}}_{\text{extinction}} \underbrace{+\sigma_a(x)L_e(x,\vec{\omega})}_{\text{emission}} \underbrace{+\sigma_s(x)L_i(x,\vec{\omega})}_{\text{in-scattering}} \tag{2.17}$$

where ∇ denotes the gradient due to the light event (which is analogous to the concept of derivative), $(\vec{\omega}\cdot\nabla)$ is the directional derivative along the direction $\vec{\omega}$, $L(x,\vec{\omega})$ describes the radiance that starts in position x follows the direction $\vec{\omega}$, $L_e(x_t,\vec{\omega})$ is the radiance resulting from the emission at the start of the beam x from direction $\vec{\omega}$, $L_i(x,\vec{\omega})$ is all the light which is scattered into the thin beam along direction $\vec{\omega}$, σ_a is the absorption coefficient, and σ_s is the scattering coefficient.

If both sides of Equation 2.17 are integrated and considering the Equation 2.14 as a boundary condition, the transmittance of the medium over distance s that starts at the position x can be express as Equation 2.18 (Jarosz [2008]):

$$L(x,\vec{\omega}_i) = T_r(s)L(x_s,\vec{\omega}_i) + \int_{t=0}^{s} T_r(t)\sigma_a(x)L_e(x,\vec{\omega}_i)dt + \int_{t=0}^{s} T_r(t)\sigma_s(x_t)L_i(x_t,\vec{\omega}_i)dt \tag{2.18}$$

where $L(x,\vec{\omega}_i)$ describes the radiance that reaches position x from direction $\vec{\omega}_i$, s is the depth of medium from position x with direction $\vec{\omega}$, $x_s = x + s\vec{\omega}$, $x_t = x + t\vec{\omega}$ with $t\ \epsilon\ (0,s)$, T_r is the transmittance (described in Subsection 2.2.4), $L_e(x,\vec{\omega}_i)$ is the radiance resulting from the emission that reaches x

from direction $\vec{\omega}_i$, $L_i(x, \vec{\omega})$ is all the light which is scattered into the thin beam along direction $\vec{\omega}$, σ_a is the absorption coefficient, and σ_s is the scattering coefficient.

The first term of Equation 2.18 is the surface radiance that enters at the backside of the medium. The second term is the accumulated emitted radiance along the medium. Finally, the last term is the accumulated in-scattered radiance.

Figure 14 illustrates the radiance reaching the camera. $L(x, \vec{\omega}_i)$ is the sum of the reduced radiance from the nearest visible surface $L(x_s, \vec{\omega}_i)$ and the accumulated in-scattered radiance $L_i(x_t, \vec{\omega}_i)$ along a ray (Jarosz [2008]).

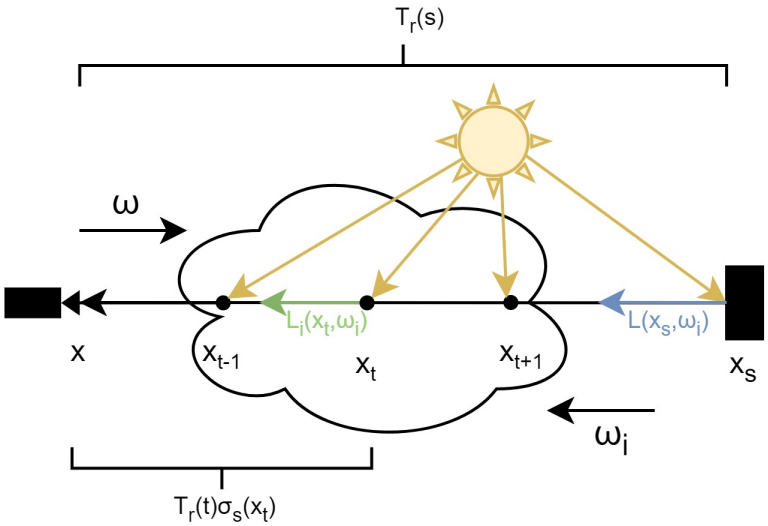


Figure 14: Scheme illustrating the integration of radiance reaching the camera.

The clouds do not emit light. The final per-spectrum radiance integration can be simplified, ignoring the emission part. The final result can be obtained using Equation 2.19:

$$L(x, \vec{\omega}_i) = T_r(s)L(x_s, \vec{\omega}_i) + \int_{t=0}^{s} T_r(t)\sigma_s(x_t)L_i(x_t, \vec{\omega}_i)dt \tag{2.19}$$

where $L(x, \vec{\omega}_i)$ describes the radiance that reaches position x from direction $\vec{\omega}_i$, s is the depth of medium from position x with direction $\vec{\omega}$, $x_s = x + s\vec{\omega}$, $x_t = x + t\vec{\omega}$ with $t \epsilon (0, s)$, T_r is the transmittance (described in Subsection 2.2.4), $L_i(x, \vec{\omega})$ is all the light which is scattered into the thin beam along direction $\vec{\omega}$, σ_a is the absorption coefficient, and σ_s is the scattering coefficient.

2.3 Cloud Modeling and Rendering Techniques

Static cloud implementation covers two important areas: modeling and rendering. Modeling is the process of developing a mathematical representation of any surface of an object (inanimate or living) in three

dimensions. Various types of techniques have been proposed to solve this problem. This includes mesh-based and volumetric methods, generating shapes from texture, noise, or even from images. In this subsection, Table 2 presents some resources used to model clouds. It is important to emphasize that these resources can be used either individually or combined with each other.

Table 2: Modeling Cloud Resources.

Resource	Description
Geometric figures	Using geometric shapes like pyramids or metaballs to shape the cloud.
Contours	Curves and meshes shapes are used to generate the base shape of the clouds. Techniques like using blob hierarchy superimposed on top of each others, sketches or silhouettes to transform the outlines in the shape of the cloud.
Images	Clouds can be modeled by real pictures, images captured from terrestrial angles or even by satellite images.
Noise and textures	Using several type of noise, like Perlin and Worley noise to make the cloud shape. Almost all other techniques can be combined with noise techniques to improve and detail the shape of the cloud.

The rendering step consists of the entire process to visualize the object and the scene on the screen, which is generating a photorealistic or non-photorealistic image from a 2D or 3D model. The end result image is referred to as the rendered image. Cloud rendering actually is divided into two essential phases: the lighting and the rendering techniques. The lighting captures the illumination or interaction of cloud particles with the surrounding impinging light and the rendering techniques employ ray casting, rasterization, or a variant to convert the existing shape representation and light setting to display the final cloud on the screen.

Table 3 shows some pros and cons of different rendering techniques (3D-Ace [2021]).

Table 3: Pros and Cons of different rendering techniques.

Rendering type	Pros	Cons

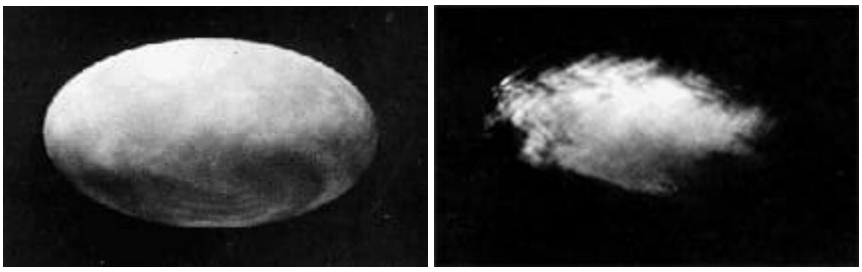
Rasterization (Scanline)	Optimal storage requirements, time-effective process and availability of the necessary tools in standard software	The algorithm is difficult to customize
Ray Casting	Applicable to flat surfaces lighter hardware load and more time-effective (faster than Ray Tracing)	Lower quality and level of detail (less realistic)
Ray Tracing	Realistic results, possible to store on disks and handles specular lighting very well	Complex algorithm and time-consuming (slower than Ray Casting)
Radiosity	More physically accurate, each object's surface is covered, realistic light effects and allows interreflection diffusion	Challenging to visualize and very light shadows

Goswami [2021] reviews existing methods for modeling and rendering clouds and tried to classify and name them according to their implementation.

2.3.1 Geometric

Gardner [1985] was one of the first to create a simple method using textures in geometric figures for modeling the cloud. In his method, ellipsoids were employed as the basic building block to model coarse three-dimensional and then a simplified Fourier series function is applied to do the details of the cloud, like its shading intensity or the transparency relative to the sky. A texturing function is used to simulate the secondary topographical detail by modulating surface shading intensity and translucence. Figure 15 shows the cloud result by Gardner [1985].

Elinas and Stuerzlinger [2000] implementation follows the Gardner [1985] approach. They also built blocks for the clouds using textured ellipsoids, whereas the only difference is that with this approach they used Perlin noise in order to have more flexibility in controlling the clouds' appearance. The texture indicates the density of the ellipsoid that when rendered becomes ellipses. The next step was spreading the edges of the ellipses so that the cloud becomes more transparent at the edges and denser at the center. One cloud is created by assembling several ellipses. So, each ellipsoid is first rendered and then

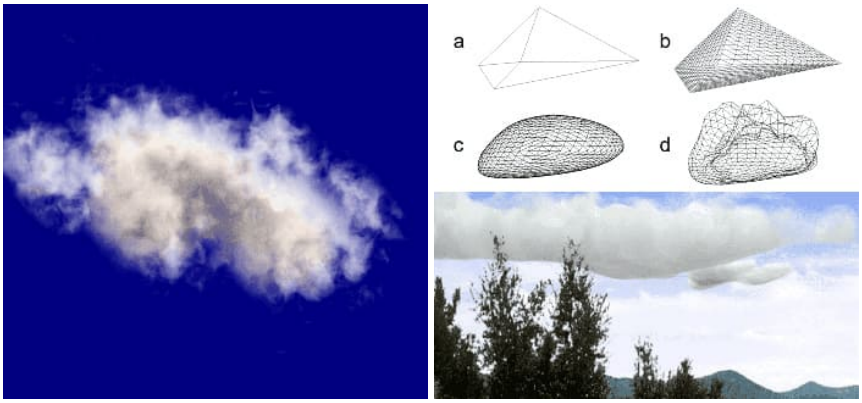


(a) Textured Ellipsoid (b) With 3D Translucence Threshold

Figure 15: Cloud modeling by Gardner [1985].

the transparency is evaluated at the edges of each ellipse result. Finally, ellipses are combined in order to make the final cloud.

Ebert [1997] begins using volumetric modeling to generate cumulus clouds with the aid of a volumetric implicit function. Figure 16a depicts a created cloud with the positioning of 9 spheres with varying diameters and blending amounts to define the entire cloud shape. This technique was improved and optimized later. Trembilski and Broßler [2002] propose a cloud modeled as isosurfaces using marching cubes. They use a geometric figure (a pyramid) and then the geometry is refined: the figure is split into sub-triangles, sharp edges are smooth, and at last, the vertex positions are shifted and moved. There is also a vertex displacement and refining to render the cloud shape (Figure 16b).



(a) Cloud resulting from the method by Ebert [1997].

(b) Cloud resulting from the method by Trembilski and Broßler [2002].

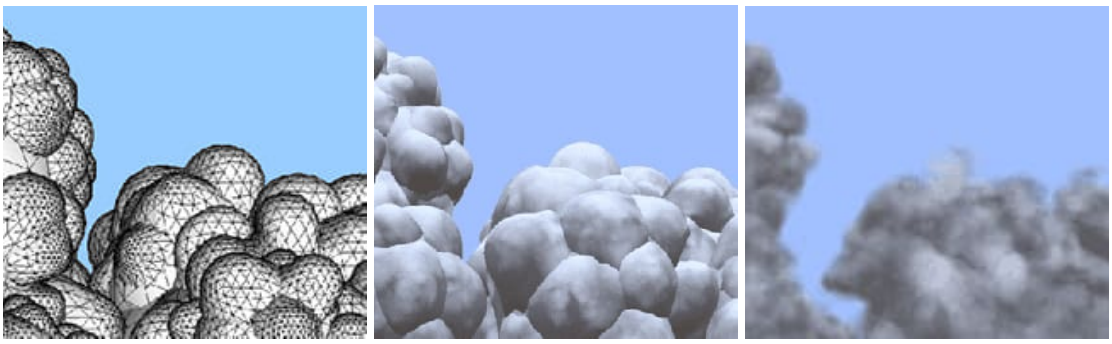
Figure 16: Geometric modeling cloud results.

Schpok et al. [2003] made a program that allows the user to model outlines of the cloud shape interactively using ellipsoids. The cloud modeling in Hufnagel et al. [2007] consists of placing particles ("metaballs") in the scene and personalizing parameters like the size, density, and texture for each metaball.

Blobs are spread out, accumulated in a large space, and distorted using a smoothing function. The light intensity in the blobs to and from the environment is calculated as if it was a radiosity. A ray is shot and the intensity is calculated through a scattering equation integrated along the path, determining which blobs the ray intersects. Bouthors et al. [2006] followed this method but using the Mie model, that is, modifying the phase function and extinctions parameters.

2.3.2 Contours

Bouthors and Neyret [2004] proposed a model to define and build the shape of cumulus clouds. Clouds were modeled with mesh shapes or curves with the help of a hierarchy of overlapping blobs (Figure 17). A combination of field functions controls several factors like cloud size, overlap, and bottom flatness. The result was pretty realistic, as the blobs are deformed to mimic the shape of real clouds. Wither et al. [2008] create clouds from 2D outlines. Spheres are placed along this contour and this is the final mesh. Stiver et al. [2010] create a sketch that is converted into a cloud mesh filled with particles to model different types of clouds. In the "Puss in boots" movie, Miller et al. [2012] were responsible for the model created for the animation film by DreamWorks. They have modeled modular clouds as polygonal meshes which were able to render the movie's set and animate its characters. During this process, the clouds were converted into narrow-band level sets, represented in a compact data structure. This data structure, called VDB (volumetric dynamic grid), forms the basic volume representation for the cloud system, enabling fast and cache-coherent data access of large volumes.



(a) Mesh of generated blobs. (b) Mesh with Perlin texture. (c) With the Gardner-like shader.

Figure 17: Cloud modeled by Bouthors and Neyret [2004].

2.3.3 Image-based

Billboards are 2D images placed on surfaces always facing the camera. In the technique developed by Dobashi et al. [2000], in each cell, a continuous density field is built by interpolation with neighbor cells. Sorted based on the distance to the sun or viewer, the textured map polygons (billboards) are placed in the center of each metaball. The light source projection gives the light intensity and the camera projection mixes the densities of the cloud with the sky (background). This rendering technique was used in clouds modeled by 2D images (image-based modeling). Yu and Wang [2011] improved this method by employing an integral line convolution to create several types of complicated and asymmetrical textures (mapped by billboards). This change has therefore improved the cloud's shape.

Real images are used to estimate the parameters of cloud images in Dobashi et al. [2012] (sky intensity, sun) with the help of genetic algorithms. Multiple interactions are performed for convergence, and during the division between the synthetic image and the target, a set of images with different settings of rendering parameters are precomputed to reduce the execution time. They used multiple scattering of light for their clouds.

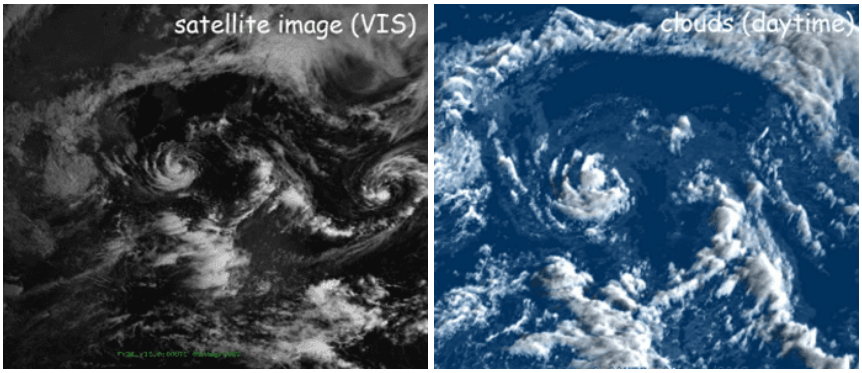
Peng and Chen [2013] aimed to extend the concept of the sky index to a random field indicating the level of cloudiness. By capturing cloud images as a test input, they can train the machine learning model to create the cloud shapes using the cloudiness variable of each sky pixel as a label to that model. They analyze every pixel of the sky, and check if it is a cloud or not, by verifying the color or exposure of that pixel. The model starts learning what it means to be cloudy, and from that, it is able to shape the clouds and create a sky.

Figure 18: The left figure is the real photography and the right one is the clouds generated and rendered by Alldieck et al. [2014] method .

Alldieck et al. [2014] use hemispherical images covering all sky as input images of the machine learning

model. The light has to be filtered, as some of the illuminated pixels can mislead the model since the data from these pixels can be similar to a cloud pixel. Then again, the pixel classification process is carried out, whether it is cloud or sky in the image, and at last, there is the reconstruction of the sky without clouds. In order to generate the cloud mesh, the vertices of the cloud on the hemisphere are created with the intensity and opacity values (Figure 18).

Yuan et al. [2014] have also used real images to model clouds assuming a symmetric cloud structure. The height field is calculated using the pixels labeled as clouds. A pixel propagation procedure is developed to build the cloud geometry. To improve its shape, a refinement is done on the front of the cloud, while on the back a more simplified process is performed. The surface is finished using the best and smallest pixel samples to fill the cloud, by performing an adaptive sampling process. Kallweit et al. [2017] created clouds using Reduced Probabilistic Neural Network (RPNN) along with the integration of the Monte Carlo algorithm into the GPU. This neuronal network learns through real images and predicts the brightness function for the desired shadow settings in the cloud.



(a) Real satellite image. (b) Satellite rendered image.

Figure 19: Cloud modeled by Yuan et al. [2013].

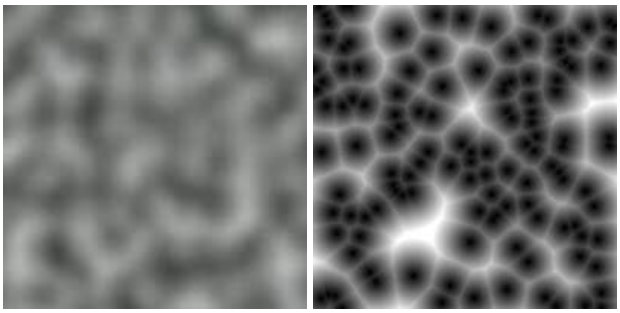
Dobashi et al. [1998] and Nishita and Dobashi [1999] used a satellite image to classify each pixel as having a cloud or not. When the pixel is identified with the maximum intensity (presence of clouds), meta-balls are created in these pixels. Dobashi et al. [2010] created a volumetric data structure that classifies the visible data blocks of the satellite image as surface, point, or volume. The different intensities and opacities are calculated and stored in preprocessing for a better runtime rendering. These are calculated for different viewing directions and cloud lighting. Wenke et al. [2012] generated clouds from the weather forecast data, which allows the possibility of using a satellite image as weather visualization effects. They detect the particles of water, ice, or snow in the satellite image and therefore can calculate the extinction and scattering coefficients.

Yuan et al. [2013] also classified satellite image pixels as cloud or non-cloud pixels. Furthermore, it can estimate the upper height of the cloud, by the amount of infrared and water vapor, thus extracting geometric structures from clouds. The medium wave infrared channel along with the visual one also stimulates the induction of other parameters such as cloud thickness and quenching properties (Figure 19).

2.3.4 Noise and textures

Perlin Noise (Figure 20a) is a type of gradient noise developed by Perlin [1995] as a result of his frustration with the "machine-like" look of computer-generated imagery at the time.

Worley noise (Figure 20b) was introduced by Worley [1996]. It is used to create procedural textures which are automatically created with arbitrary precision and do not have to be drawn by hand. Worley noise is used to simulate textures of stone, water, or biological cells.



(a) Example picture generated with Perlin Noise.

(b) Example picture generated with Worley Noise.

Figure 20: Perlin and Worley noises.

These two noises (Figure 20) are the most used to create clouds. Its application can be made using the Fractal Brownian Motion (FBM) technique (Mandelbrot and Van Ness [1968]). This technique consists of adding different iterations of noise (octaves), where the frequencies are successively increased in regular steps and the noise amplitude is decreased. It is possible to obtain a finer granularity in the noise and get more details with the FBM technique.

Schneider and Vos [2015] created voxel clouds. Their modeling approach uses ray marching to produce clouds. Throughout the ray march, sampler noises and a set of gradients are applied to model the cloud. All the illumination is calculated during the ray marching. The layered noises used in this technique were developed by them. They started by inverting the Worley noise. When the Worley noise is inverted, it made tightly packed billow shapes (Figure 21).

The inverted Worley noise was layered with the FBM technique. This result was used as an offset

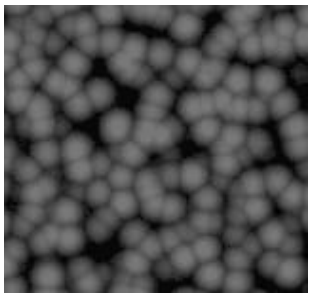


Figure 21: Worley Noise inverted.

for the Perlin noise. This allows for keeping the Perlin Noise while adding some billowy shapes. They combined the two noises and called it the "Perlin-Worley" noise (Figure 22).

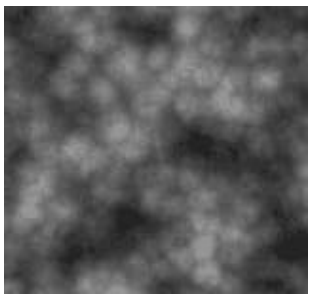


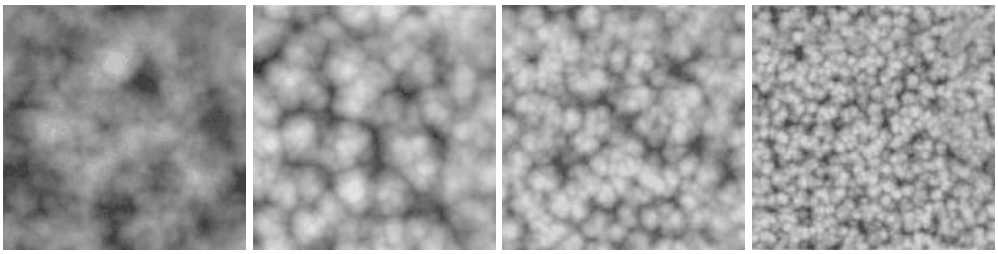
Figure 22: "Perlin-Worley" noise.

They compressed their noise to two 3D textures and one 2D texture (Table 4). The first 3D texture represented in Figure 23 has four channels, the red stores the Perlin noise, and the last three channels store Worley noise at increasing frequencies. Figure 24 is the second 3D texture, it only has three channels with Worley noise at increasing frequencies. It is used to get a more detailed cloud shape. The last texture is a 2D texture and uses curl noise. This noise is used to distort the cloud shape and to add a sense of turbulence.

Textures	Resolution	R	G	B	A
First 3D texture (Figure 23)	128^3	Perlin	Worley	Worley	Worley
Second 3D texture (Figure 24)	32^3	Worley	Worley	Worley	–
2D texture (Figure 25)	128^2	Curl	Curl	Curl	–

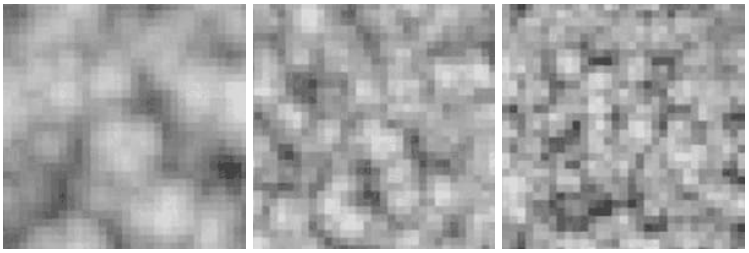
Table 4: Information stored in RGB channels of Schneider and Vos [2015] noise textures.

They used different gradients to change the noise signal over altitude. There are three presets to represent the major low altitudes, which allows creating the different types of clouds at different altitudes,



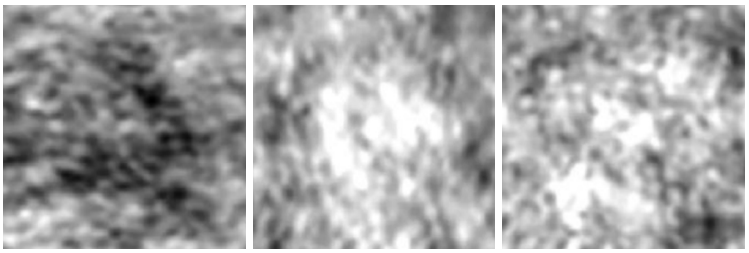
(a) Red Channel. (b) Green Channel. (c) Blue Channel. (d) Alpha Channel.

Figure 23: First 3D noise texture used by Schneider and Vos [2015].



(a) Red Channel (b) Green Channel (c) Blue Channel

Figure 24: Second 3D noise texture used by Schneider and Vos [2015].



(a) Red Channel (b) Green Channel (c) Blue Channel

Figure 25: 2D noise texture created by Schneider and Vos [2015].

blending between the presets at the sample position. They have one value that varies between 0 and 1, indicating how much cloud coverage they want to have in that sample position.

For modeling clouds, it is built a basic cloud shape with a low resolution by sampling the first 3D texture (Figure 23) and multiplying it by the height signal. Then the result is multiplied by the coverage of the signal (between 0 and 1) and reduced density on the bottoms of the clouds. This makes the bottoms wispier and increases the number of clouds in a more natural way.

The second texture is multiplied by the 2D curl texture (Figure 25) to distort the shape of the cloud. This step is important to fake the swirly distortions from atmospheric turbulence. The erosion of the base cloud shape is the final stage, where there is a subtraction between the edges of the cloud and the second distorted 3D texture (Figure 24).

In order to create rainy clouds and get a weather setting, a 2D texture was used to store some information. A sample of the texture is represented in Figure 26a. The information stored in each channel of this texture is specified in Table 5. With this type of image, there is greater control over the weather. They called these images Weather textures. The various pink color is the result of the output from the weather system. The more red areas have less of the blue signal, making them stratus clouds.

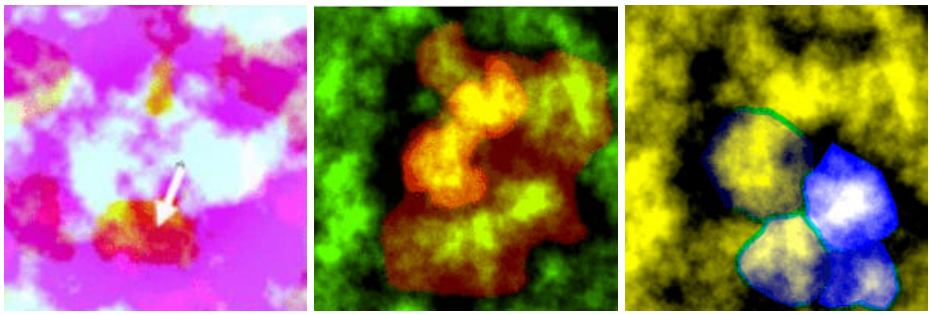
Weather texture	R	G	B
Figure 26a	Coverage	Precipitation	Cloud type
Figure 26b	Density	Figure 27	–
Figure 27	Density in the layer height	Erosion amount applied	–
Figure 26c	Coverage	Height	Altitude

Table 5: Information stored in RGB channels of the different Weather textures.

To render the clouds they used ray marching. There are three main goals in cloud illumination: directional scattering, the silver lining effect, and the dark edges of the cloud, creating the cauliflower effect. To fulfill the first objective, the Beers-law was used to determine the amount of light reaching a point-based (Subsection 2.2.4). The Henyey-Greenstein function was used to create the silver lining effect when we are looking toward the sun through a cloud (Subsection 2.2.2). The dark edges visible on clouds when we look away from the sun appeared when a Powder function was applied (Equation 2.20):

$$T_r(x, \vec{\omega}) = 1 - e^{-\sigma_t d \times 2} \tag{2.20}$$

where $T_r(x, \vec{\omega})$ is the light transmittance along the distance d and σ_t is the extinction coefficient. The Beer's Law and Powder function are represented in Figure 28.



(a) Texture used by Schneider and Vos [2015]. (b) Texture used by Hillaire [2016a]. (c) Texture used by Högfeldt [2016].

Figure 26: Example of cloud weather textures.

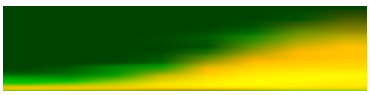


Figure 27: Second weather texture used by Hillaire [2016a].

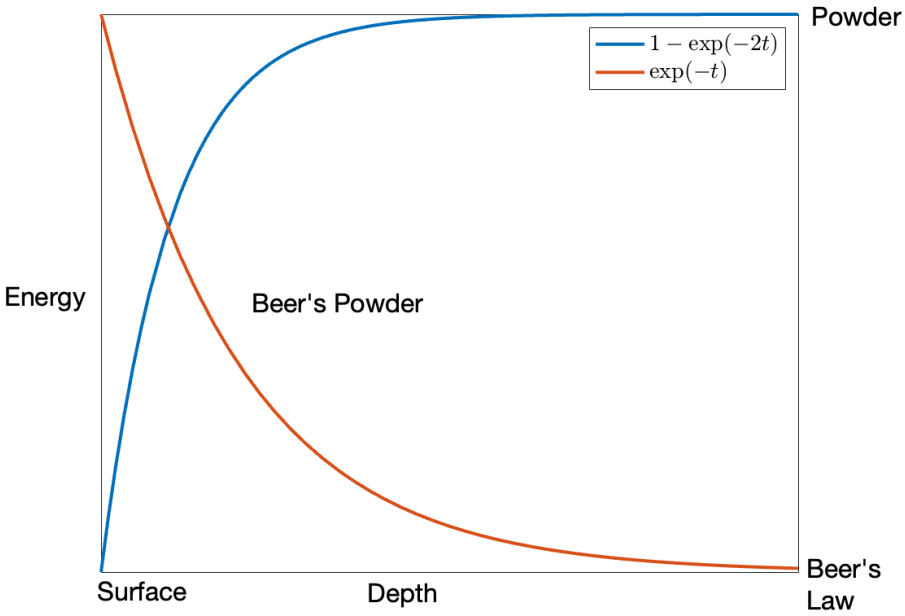


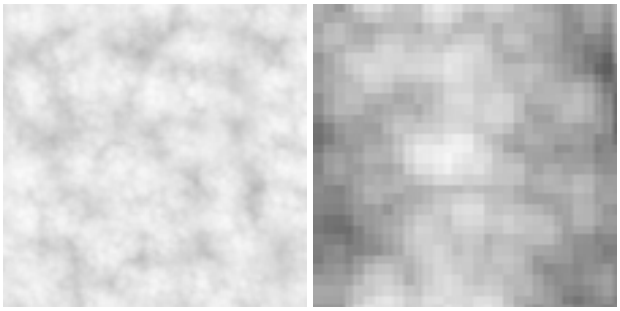
Figure 28: Beer's Law and Powder function representation.

The clouds resulting from this method are represented in Figure 29.

Hillaire [2016a] and Högfeldt [2016] (Frostbite game) decided to follow Schneider and Vos [2015] because that method generated realistic cloud shapes, possibly large-scale clouds, dynamic volumetric lighting and shadowing support, and dynamic clouds where the weather can change.

Hillaire [2016a] used two different noises to make the base shape of the cloud (Figure 30). The first

Figure 29: Resulting clouds of the Schneider and Vos [2015] method.



(a) Noise for the cloud shape. (b) Noise for cloud detail.

Figure 30: Noise Textures used by Hillaire [2016a].

noise volume texture (Figure 30a), with the low frequency, is generated as a combination of Perlin-Worley noise and multiple octaves of Perlin noise. It is used to give a base shape to clouds. The high-frequency noise texture (Figure 30b) is generated as multiple octaves of Worley noise. It is used to erode the base shape and add details at edges.

These textures can be presented as 4-component RGBA textures that are combined using shader math in Schneider and Vos [2015] method. In this case, they have opted to use a single component in order to reduce the required memory and consequently render the cloud faster.

Hillaire [2016a] provided a program that generates these two textures [1]. The information stored in each channel of the noise textures is represented in Table 6.

[1] These two textures were generated with the program Hillaire [2016b]

Textures	Resolution	R	G	B	A
First Texture 2D (Figure 30a)	$128 \times 32 \times 128$	Perlin-Worley	Perlin	Perlin	Perlin
Second Texture 2D (Figure 30b)	32^3	Perlin	Worley	Worley	–

Table 6: Information stored in RGB channels of Hillaire [2016a] noise textures.

Högfeldt [2016] used different textures to create the cloud shape and improve its details (Figure 31). Table 7 shows the information stored in each channel of these noise textures. They used more Worley noise to create clouds than in the Hillaire [2016a] approach.

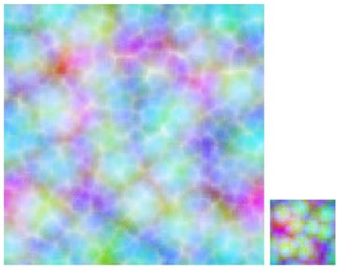


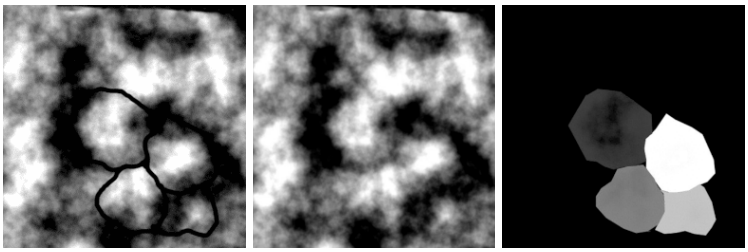
Figure 31: On the left the Shape noise and on the right the Detail noise texture used by Högfeldt [2016].

Textures	Resolution	R	G	B	A
First Texture 2D	$128 \times 32 \times 128$	Perlin	Worley	Worley	Worley
Second Texture 2D	32^3	Perlin	Worley	Worley	–

Table 7: Information stored in RGB channels of Högfeldt [2016] noise textures.

Both used weather textures to store the information needed to model the clouds. This weather texture has been created having a world space size that was scaled and repeated all over that world. Compared to the Schneider and Vos [2015] texture, these weather textures stored the information differently. In Hillaire [2016a], the red channel stores the cloud density, and one of the channels stores information from another texture which saves details about the type of cloud. The more concentrated the red color, the denser the cloud (Figure 26b, Figure 27). The weather texture used by Högfeldt [2016] (Figure 26c) also stores the cloud density in the red channel. In the green and blue channels, it is stored the height and cloud altitude, respectively (Table 5). In order to better understand and observe the information stored in the different channels, the image was separated into the images of each channel (Figure 32).

For cloud rendering, Hillaire [2016a] used the equations presented in Section 2.2 to integrate different lighting components: background transmittance, ambient scattering, not shadowed sunlight scatter-



(a) Red Channel (b) Green Channel (c) Blue Channel

Figure 32: The three channels of the weather texture used by Högfeldt [2016].

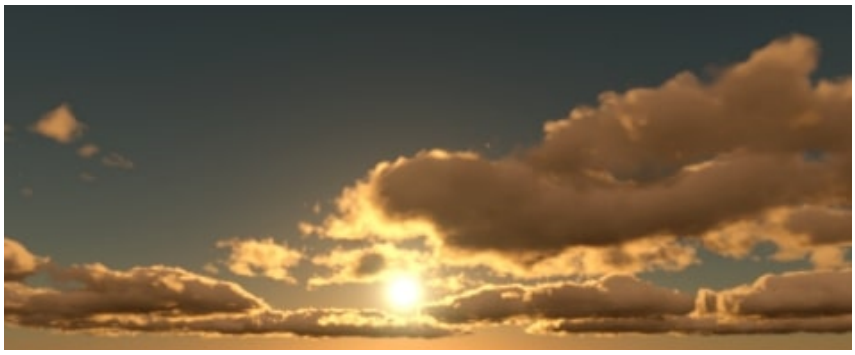
ing, cloud self-shadowed sunlight scattering without and with forward scattering phase function. Hillaire [2016a] and Högfeldt [2016] also used ray marching to create volumetric cloud shadows and handle the cloud light. In both approaches, they used a simplification of the Radiative Transfer Function (RTE) (Equation 2.19). Transmittance follows the properties of Beer–Lambert's law (Equation 2.12). Finally, the Henyey-Greenstein phase function (Equation 2.7) was used for cloud lighting effects such as silver lining.

Hillaire [2016a] and Högfeldt [2016] final cloud results are represented in Figure 33.

2.4 Summary

Chapter 2 presents a brief introduction to the cloud describing the main cloud types, explaining the light transport in clouds, and showing some cloud modeling and rendering techniques. The Earth's planet is enveloped by the atmosphere. Clouds are in the troposphere, a layer residing above the Earth's surface in the atmosphere. Clouds can be formed at various altitudes (low, mid, and high altitudes). Section 2.1 presents the cloud types. The main cloud designations where its base is found in the low stage (2 km from the Earth's surface) are Cumulus, Cumulonimbus, Stratocumulus, and Stratus. The three denominations for mid-stage clouds (2 to 8 km in tropical latitude, 2 to 7 km in a temperate region, and 2 to 4 km in the polar region) are Nimbostratus, Altostratus, Altocumulus. Finally, there are three cloud types at the high stage altitude (6 to 18 km in the tropical region, 5 to 13 km in the temperate region, and 3 to 8 km in the polar region), Cirrus, Cirrocumulus, and Cirrostratus. Section 2.2 describes the light transport in participating media, i.e. volumes filled with particles. There are some events that happen when light travels through a participating media: absorption, scattering (in and out scattering), extinction, transmittance, emission, and radiative transfer. The absorption (Subsection 2.2.1) is the event when photons are absorbed after traveling through participating media. Scattering (Subsection 2.2.2) is the event when light travels through a volume in a certain direction and the photons are deflected in other directions when they collide with particles. The photons can be deflected in the viewer's direction

(a) Hillaire [2016a] clouds result.



(b) Högfeldt [2016] clouds result.

Figure 33: Frostbite game clouds.

(In-scatter), and away from the viewing direction (Out-scatter). In computer graphics, the phenomenon of scattering is simplified by using a mathematical model called a phase function. This is responsible for the angular distribution of scattered light. There are two types of scattering behavior that a participating medium can exhibit: isotropic and anisotropic. An isotropic phase function describes equal scattering in all directions. There are some anisotropic phase functions such as Mie, Henyey-Greenstein, Cornette-Shank, Schlick and Rayleigh. The RTE is a complicated function used in computer graphics to calculate the total change in radiance in a participating medium. This function encompasses the four main light events: absorption, scattering, transmittance, and emission. Since clouds do not emit light, the emission part can be ignored to calculate the radiance in this participating media. Subsection 2.2.6 explains how to get to the final RTE. It is important to understand each light event to implement cloud lighting in computer graphics. Section 2.3 describes the process to implement clouds in computer graphics and refers to the different cloud rendering techniques used during these years. To implement static clouds it is necessary to go through two important phases: modeling and rendering. Modeling is the process of making a cloud shape. The cloud shape can be modeled with geometric figures, outlines, terrestrial or satellite images or even using several types of noise, like Perlin and Worley noise. The rendering step is divided into two essential phases: the lighting and the rendering techniques. There are several rendering techniques used to create clouds. The most used in recent years is ray marching, because, despite its computational cost, it creates very realistic outdoor scenes. Section 2.3 briefly describes the techniques used over time, from Gardner [1985] to Hillaire [2016a].

Chapter 3

Volumetric Clouds

This chapter describes how to create volumetric clouds, providing the required resources and details for their implementation. Cloud implementation requires two main phases: Modeling and Rendering. Section 3.1 addresses cloud modeling in which the shape of the cloud is defined. Section 3.2 is devoted to cloud rendering.

3.1 Cloud Modeling

The first step to create a cloud is to define its shape. Subsection 3.1.1 describes the base representation of the cloud. Subsection 3.1.2 details the post-processing stages using noise and other methods in order to achieve a more organic shape.

3.1.1 Base Cloud Shape

Figure 34 depicts two clouds with a geometrical base shape. However, these shapes are improved later to obtain more realistic clouds. To create geometrical clouds we need to know some information beforehand, such as initial cloud density and data on its position. Cloud vertical location can be computed with two values: height and altitude. The horizontal placement is defined by a grid of values (texture).

The method used to treat this information follows a volumetric approach, as described in Hillaire [2016a].

The volume is placed on the atmosphere, with its altitude and height depending on the types of clouds to represent. The volume is a rectangular box, as can be seen in Figure 35.

The information to generate a model is contained in its voxels. Here each voxel represents the density of the cloud at that particular position.

Although Voxels can be filled with values through algorithms, more intuitive methods exist, like the approach in Högfeldt [2016] using 2D map textures images. The disadvantage of using 2D images is

Figure 34: Clouds with a geometrical base shape.

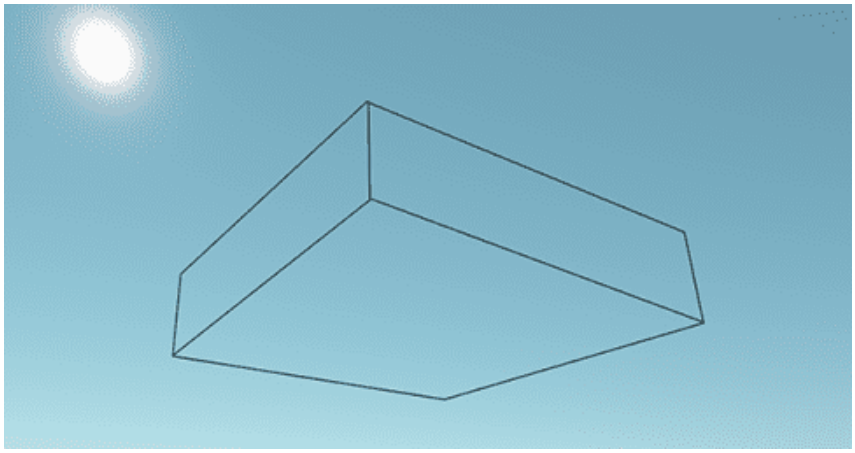


Figure 35: The volume box in the sky.

the fact that it is no longer possible to have overlapping clouds, that is clouds in the same location with different altitudes and heights.

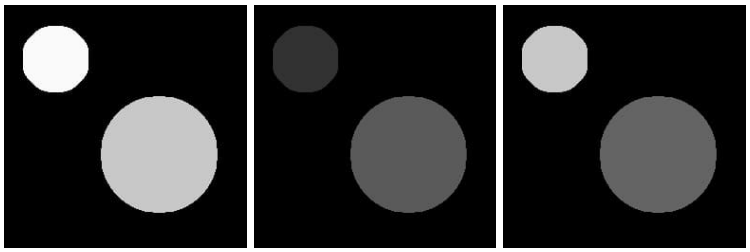
Three 2D image textures were created to map this information, one for the coverage another for the height, and another to store the altitude of the cloud. These images contain values between 0 and 1. These values are not the real height and altitude of the cloud, but a percentage related to the base of the box volume.

As it can be seen in Figure 34 the cloud on the left has a higher density than the one on the right. So, the first texture, represented in Figure 36a stores a greater coverage value, closer to one, in the left circle. Consequently, the left circle is lighter.

The circle on the left is at a higher altitude than the one on the right. Thus, the texture represented in Figure 36c, has stored a greater altitude value in the left circle. So, this circle is lighter.

The cloud on the left has a lower height. So the second texture, represented in Figure 36b, stores a smaller height value, closer to zero, for the left circle. Therefore, the left circle is darker.

It is important to mention that clouds only exist when the height value is greater than zero. Further-



(a) Coverage (b) Height (c) Altitude

Figure 36: 2D textures images created.

more, the addition of height and altitude value must be lower than one.

These three texture images were combined in one image represented in Figure 37. This image is called weather texture.

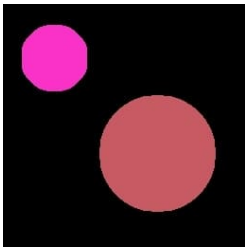


Figure 37: RGB weather texture.

Table 8 shows the information stored in each channel.

R	G	B
Coverage	Height	Altitude

Table 8: Information stored in RGB channels of the Weather texture.

The weather texture does not represent the final shape of the cloud, but its limits and initial densities. It allows us to define the horizontal shape of the cloud. The vertical component is an extrusion of the profile defined by the texture.

As it can be seen in Figure 38, the blue channel stores the altitude of the cloud, i.e., the distance between the bottom of the volume and the base of the cloud, while the green channel stores the height of the cloud, that is, the distance from the base (altitude value) to the top of the cloud.

In order to obtain more realistic cloud shapes, it is important to include noise in the creation process of the weather texture.

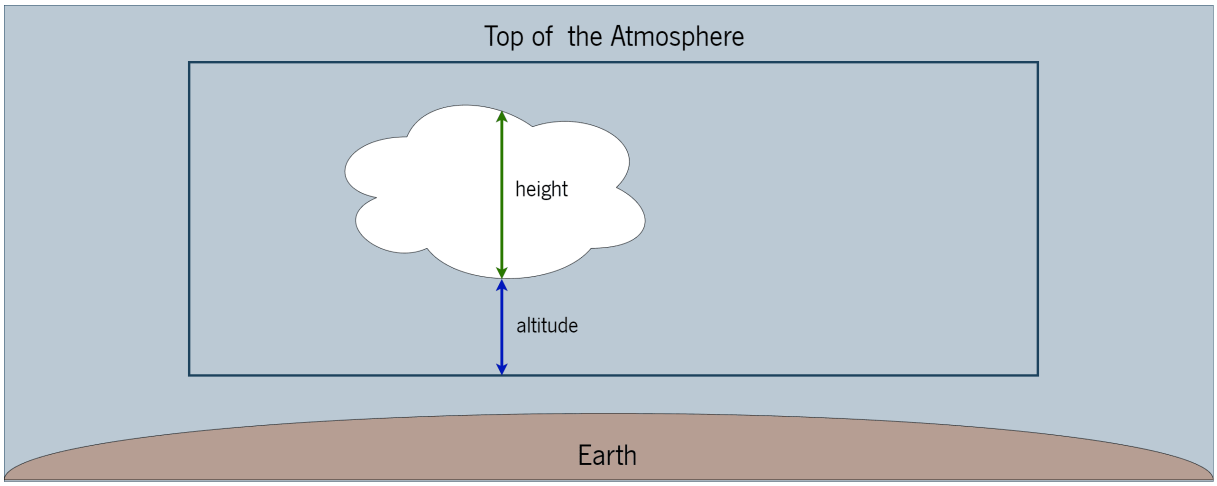


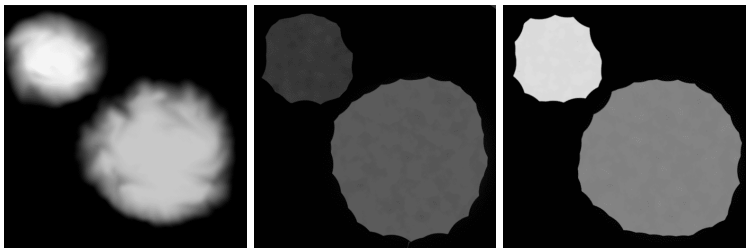
Figure 38: Scheme to represent the information stored in the channels.

To achieve a more organic shape, We have conducted an experiment in which Perlin noise (Perlin [1995]) is incorporated. Adding noise to the coverage channel alters the horizontal profile of the cloud, providing irregular borders. This noise alters the cloud coverage and, hence, the new coverage must be propagated to the remaining channels to ensure that altitude and height information is available wherever the red channel contains a value different from zero. Next noise was applied to both the altitude and height channels. Adding noise to the altitude channel makes the cloud base look less polished, while the new height values provide a more diverse shape for the top of the cloud.

The weather texture represented in Figure 40 was created to obtain the scene in Figure 39.

Figure 39: Weather texture result.

With the purpose of exemplifying the contents of the weather texture, the three channels are also shown in Figure 40. These images were converted to a gray-scale to identify whether the value is closer to 0 (black) or 1 (white) since the values are always treated on a scale between 0 and 1.



(a) Coverage (b) Height (c) Altitude

Figure 40: Example of a weather texture with Perlin noise and its three channels.

In games, usually, weather textures are created by artists, and are meticulously planned to obtain aesthetically pleasing and realistic clouds.

3.1.2 Cloud Post-processing

In order to approximate the shape to a cloud, further post-processing is required. This section describes the height signal algorithm and discusses the application of further noise to improve the cloud shape.

Height Signal

Realistic clouds have rounded edges. A solution to achieve this is to apply the height signal formula. as described in Högfeldt [2016]. This function is implemented as a parabola function (Equation 3.1), where $D(a)$ is the final cloud density, a is the altitude in volume, a_0 is the cloud initial altitude and h is the cloud height.

$$D(a) = (a - a_0) \times (a - a_0 - h) \times -\frac{4}{h^2} \tag{3.1}$$

In Figure 42 it is possible to compare a cloud without height signal and after applying this formula. In Figure 42b the edges have been rounded.

Noise post processing

Combinations of two other noises that are well-known and used by developers in computer graphics were used in this implementation: Perlin noise (Figure 20a) and Worley noise (Figure 20b). [1]

These noises were combined and precomputed into textures to improve the cloud's shape and details. Figure 43 represented the two 3D textures slice.

The noise texture, illustrated in Figure 43a is used to create the shape of the cloud.

[1] These two textures were generated with code found in Hillaire [2016b]

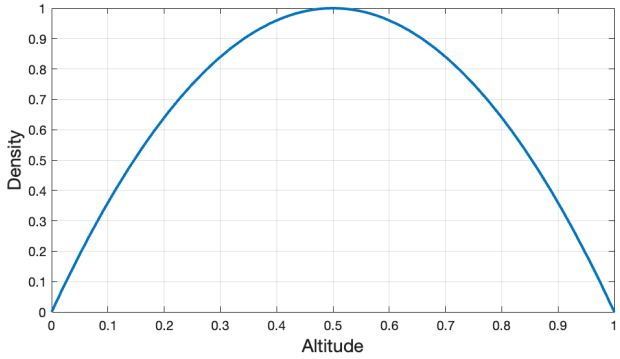
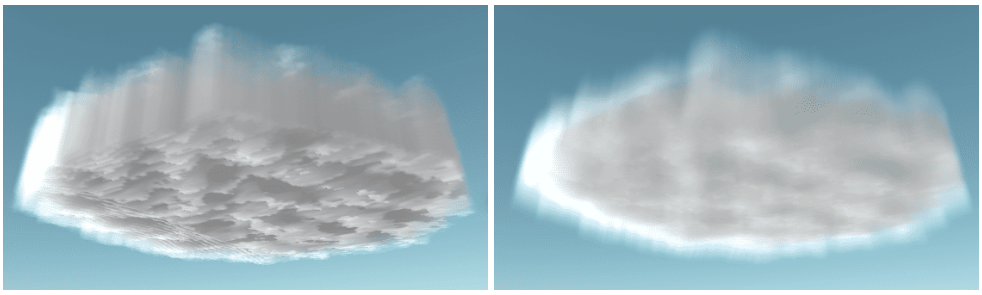


Figure 41: Height signal with $a_0 = 0$ and $h = 1$.



(a) Without Height Signal algorithm. (b) With Height Signal algorithm.

Figure 42: Results of Height Signal algorithm.

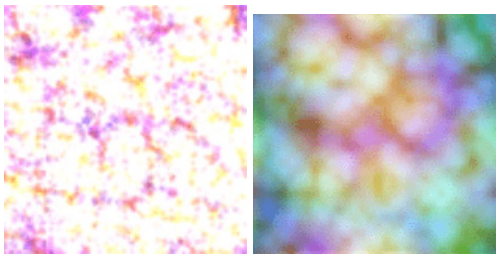
The Perlin noise in green, blue, and alpha channels is combined using the Fractal Brownian Motion (FBM) technique according to Equation 3.2. The result is then multiplied by the red channel noise (the Perlin-Worley noise). Combining Perlin noise stored in Green, Blue, and Alpha channels gives more detail to the cloud. Equation 3.2 expresses the shape noise application:

$$density\times = (SN_g \times 0.5 + SN_b \times 0.25 + SN_a \times 0.125) \times SN_r \tag{3.2}$$

where SN_g is the noise in the Green channel, SN_b is the noise in the Blue channel, SN_a is the noise in the Alpha channel, and SN_r is the noise in the Red channel.

The result of this shape noise can also be multiplied by a user-defined noise shape factor. Figure 44 shows clouds with different noise shape factors. In Figure 44a the cloud is less dense and more transparent when compared to the cloud in Figure 44b. The higher the noise factor value, the denser the cloud becomes.

The detail noise, represented in Figure 43b, serves to erode the cloud at the edges. The way this noise is handled is very similar to shape noise.



(a) Shape texture generated.

(b) Detail texture generated.

Figure 43: Generated textures.

Textures	Resolution	R	G	B	A
First Texture 2D (Figure 43a)	$128 \times 32 \times 128$	Perlin-Worley	Perlin	Perlin	Perlin
Second Texture 2D (Figure 43b)	32^3	Perlin	Worley	Worley	–

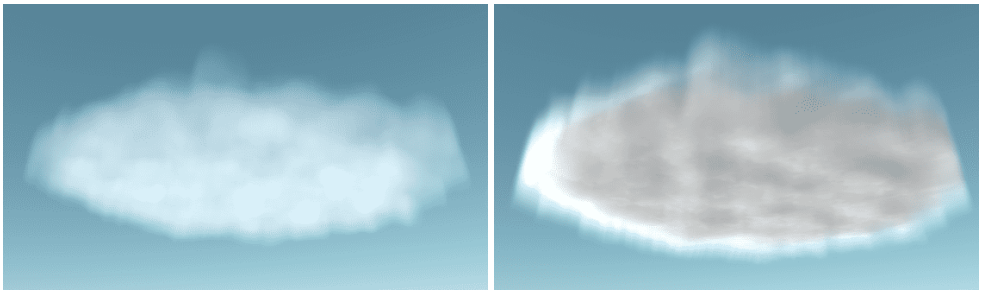
Table 9: Information stored in RGB channels of the noise textures.

As previously, the FBM technique as in Equation 3.3 was applied. This time using Perlin noise and only two Worley noises. The Perlin noise is saved in the red channel, and the two Worley noises are in the green and blue channels. In this detailed noise texture, the alpha channel does not contain any information. Equation 3.3 expresses the shape noise application:

$$density- = SN_r \times 0.5 + SN_g \times 0.25 + SN_b \times 0.125 \tag{3.3}$$

where SN_g is the noise in the Green channel, SN_b is the noise in the Blue channel, and SN_r is the noise in the Red channel.

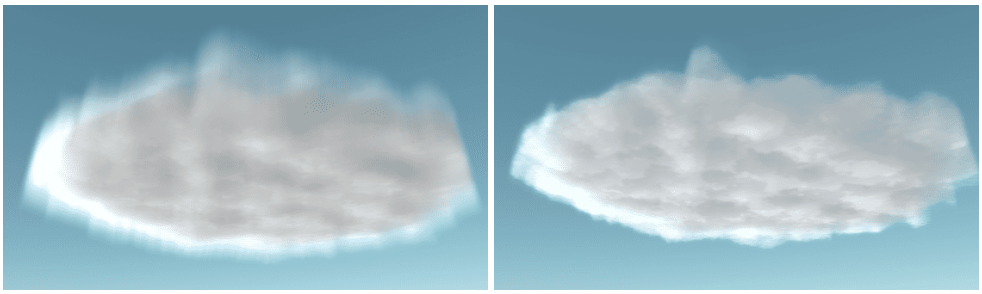
The resulting density value is also multiplied by a percentage of detail noise. The user can decide how much he wants the cloud to be eroded at the edges, on a scale between 0 and 1.



(a) Noise shape factor = 1.

(b) Noise shape factor = 3.

Figure 44: Shape Noise difference.



(a) Percentage Detail = 0. (b) Percentage Detail = 0.9.

Figure 45: Noise Detail difference.

The code in Listing 3.1 shows how the noises were applied to compute the density.

```
//Add noises to cloud
    float density;
    density = weathertexture.r //get the coverage
    density *= heightSignal //remove the edges
    density *= getShapeNoise //improve the shape with noise
    density -= getDetailNoise //improve the details with noise
```

Listing 3.1: Implementation specification of the noises addition to the cloud

3.2 Cloud Rendering

Analyzing the advantages and disadvantages of the rendering methods presented in section 2.3, ray marching was the chosen technique to render clouds. Ray marching is a type of ray tracing. The main advantage, compared to the other rendering methods, is to create more realistic lighting and shadows (3D-Ace [2021]).

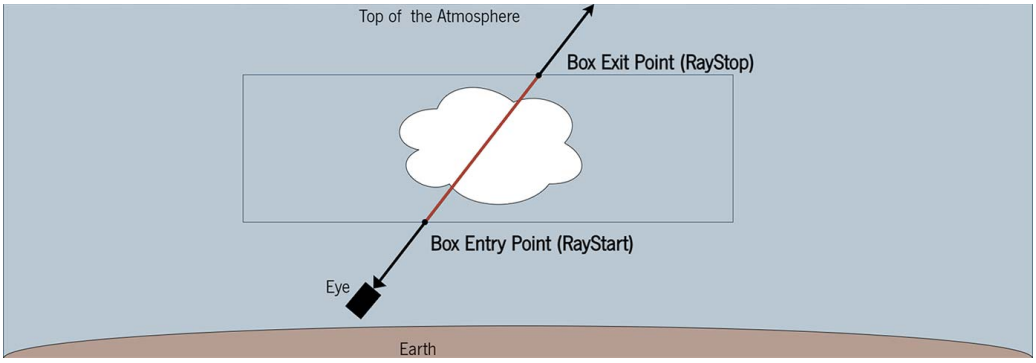


Figure 46: Box Entry and Exit points.

Ray marching over a volume requires a marching start and an endpoint. These points on the 3D box were calculated with an algorithm published by Rideout [2021]. In Figure 46 it is possible to visualize the algorithm to calculate the two different points in the box.

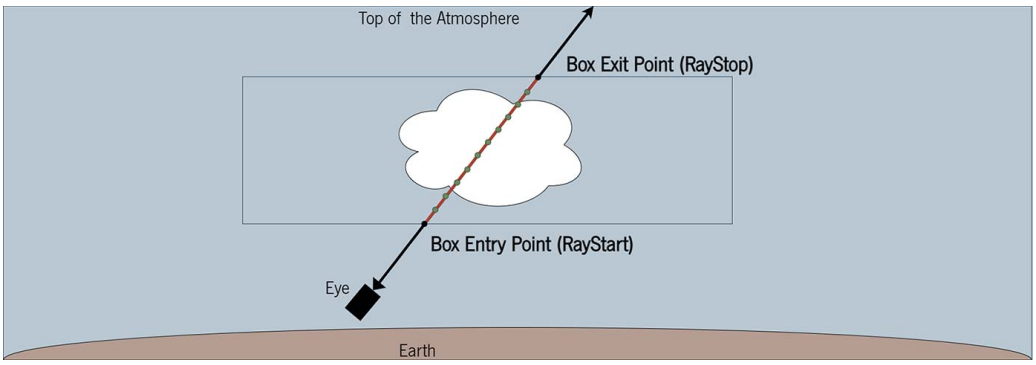


Figure 47: Raymarching with steps.

For each pixel at each step along the ray, its density and illumination are calculated. The more steps, the smaller the step length gets. So the cloud gets a more realistic rendering (with a less "pixelated" effect). In this solution, the step size is the distance from the box, that is, the difference between the point when the ray leaves the box and the entry point of the ray in the box (RayStop and RayStart) divided by the number of steps chosen by the user in the interface of the program (Figure 47).

The light density is calculated for each pixel in the step, through a light marching. A ray comes from the light source, the sun, to the point in the ray that goes from our field of vision to the atmosphere. The

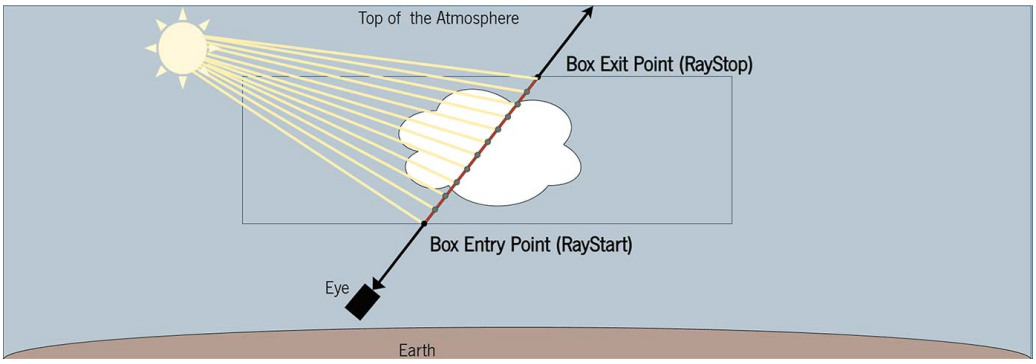


Figure 48: Light Marching.

light step size is also the difference between the two points where the light ray intersects with the box, but is now divided by twice the grid size. The more iterations, the sharper the shadows, thus the effect of absorption and scattering of light in the cloud can be seen more clearly (Figure 48).

The code in Listing 3.2 shows the main cloud rendering algorithm. The function $calculateEnergy()$ is represented in Listing 3.4.

```
1  //return cloud
2  int divisions; //number of steps
3  int lightdivisions; // number of light steps
4  vec4 getCloud(divisions, lightdivisions){
5      vec3 rayStart; //ray march start point
6      vec3 rayStop; //ray march stop point
7      //computes the 2 intersecting box points (rayStart and rayStop) along
       the ray, starting in origin and with direction dir
8      IntersectBox (origin, dir);
9
10     vec3 step = distance(rayStart,rayStop)/divisions; //step lenght
11     vec3 position = rayStart + step *0.5;
12     float lightTransmittance, transmittance;
13     for (i=0; i<divisions; i++){
14         //calculate the density of the cloud in position
15         density = GetDensity(position);
16         //computes the 2 intersecting box points (lightRayStart and
       lightRayStop) along the ray with origin in position along the direction
       sunDir
17         IntersectBox(position, sunDir);
18         vec3 lightStep = distance(lightRayStart, lightRayStop)/
       lightdivisions; // light step lenght
19         lightPosition = lightRayStart + lightStep;
```

```
20          for (j=0; j<lightdivisions; j++){
21              lightDensity = getDensity(lightPosition)
22              lightTransmittance = beerLaw(lightDensity, coefficient,
    lightStep);
23              lightPosition +=lightStep;
24          }
25          transmittance = beerLaw(density, coefficient, step);
26          calculateEnergy(){
27           //see listing 3.4
28          }
29          position+=step;
30      }
31  }
```

Listing 3.2: Cloud rendering algorithm implementation

Along the march, the light is calculated with both the density of the sample and the density between the sample and the sun. To implement the cloud light, some laws and functions were applied: the Beer-Lambert law and the phase function.

3.2.1 Beer-Lambert law

The transmittance is the ratio between the light received by the observer and the light emitted by the light source. The attenuation of the light and the properties of the material through which the light is traveling are related by the Beer-Lambert law. The Beer-Lambert law, represented in Equation 3.4, is used to compute the cloud transmittance. This law uses an absorption and a scattering coefficient. The absorption coefficient is the probability that a material absorbs light and the scattering coefficient is the probability of scattering the light. Absorption and scattering apply to all objects. If an object is black, it is absorbing most of all the light. When an object is white, it absorbs very little light and scatters most of it. Clouds absorb much less light than they scatter. That is the reason clouds, in most situations, are whiter. If both coefficients are zero, the volume is completely transparent. The absorption and scattering coefficients are said to express a probability density. The user can decide the value of these both probabilities in the interface of the program. To have more realistic cloud lighting, the scatter coefficient has to be higher, and the absorb value smaller, this value has to be close to zero. In order to create an effect of rainy clouds, the absorption coefficient has to be increased. When clouds contain denser particles of water, they absorb

more light and automatically become darker. The law is express by Equation 3.4:

$$T = e^{-(\sigma_a+\sigma_s)l} = e^{-\sigma_t l} \tag{3.4}$$

where T is the the transmittance, σ_a is the absorption coefficient, σ_s is the scattering coefficient, σ_t is the extinction or attenuation coefficient and l is the distance the light travels through the material.

This law is applied to both paths, vision, and light path.

```
float beerLaw(){
transmittance = exp(-density * coefficient * stepSize);
return transmittance;
}
```

Listing 3.3: Beer Lambert Law implementation.

3.2.2 Energy-conserving

In order to obtain energy-conserving scattering over a range from 0 to the depth range d, the scattered light is analytically integrated according to extinction σ_t and a scattered light sample S. Equation 3.5 expresses the computation of energy conserving scattering (Hillaire [2016a]):

$$\int_{x=0}^{d} e^{-\sigma_t x} \times S \, dx = \frac{S - S \times e^{-\sigma_t d}}{\sigma_t} \tag{3.5}$$

where σ_t is the extinction or attenuation coefficient, S is the scattered light sample and d is the depth or distance.

If σ_t is zero, the equation is undefined, and the cloud becomes black. To avoid this issue, the σ_t should not be larger than zero.

```
//calculateEnergy
...
  vec3 scatteredLight = calculateS();
  vec3 energy = (scatteredLight-scatteredLight * transmittance)/
 sampleSigmaT;
  cloud += T * energy * sampleSigmaS;
  T *= transmittance;
...
```

Listing 3.4: Energy conserving implementation

3.2.3 Phase function

In computer graphics, the scattering is simplified by using a mathematical model called a phase function. This is responsible for the angular distribution of scattered light, to create the In-Scattering effect, also known as the silver-lining effect. The phase function has two input parameters to shape a scattering profile: the mean cosine, also known as g, which describes the degree of anisotropy of the phase function and the angle between the light incident direction and the viewing direction.

```
//calculateS
 vec3 S = sunColor* lightTransmittance * beersPowder * phaseFunction;
```

Listing 3.5: Scattered light implementation

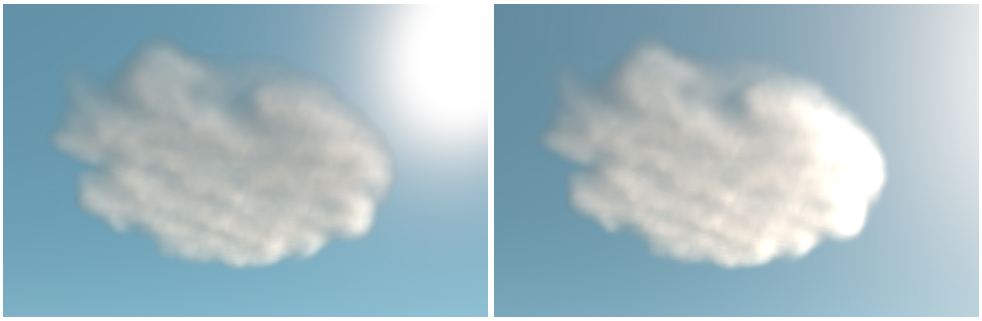
Some functions can be used to make cloud lighting. In theimplementation, several phase functions were tested in order to evaluate which one was most suitable for lighting the clouds: the Henyey-Greenstein (Henyey and Greenstein [1941]), the Cornette-Shank phase function (Cornette and Shanks [1992]) and the Rayleigh phase function.

```
float henyey_greenstein(const float phase, const float g){return (1.0 - g *
    g) / (4.0 * PI * pow(1.0 + g * g - 2.0 * g * phase, 3.0/2.0));}

float cornette(const float phase, const float g){return (3*(1-g*g)/(4.0 *
   PI *2*(2+g*g)))* (1+(phase)*(phase)) / (pow(1+g*g-2*g*(phase), 1.5));}

float schlick(const float phase, const float g){return (1-g*g)/ 4 *PI*(pow
   (1+g*phase, 2));}

float rayleigh_phase_func(float mu){return 3. * (1. + mu*mu)/ (16. * PI);}
```

Listing 3.6: Code for different phase functions

Figure 49 shows the difference between the various phase functions application.

The user can choose between these four phase functions in the program interface. In computational terms, there is not much variation for the different phase functions. In visual terms, Henyey-Greenstein and Cornette-Shank do not make much difference, and the presented graph in Figure 50 manages to show that they are very similar for g = 0.9.



(a) Cloud with no phase function.

(b) Henyey-Greenstein phase function.

(c) Cornette-Shank phase function.

(d) Schlick phase function.

(e) Rayleigh phase function.

Figure 49: Cloud with different phase functions for $g = 0.9$.

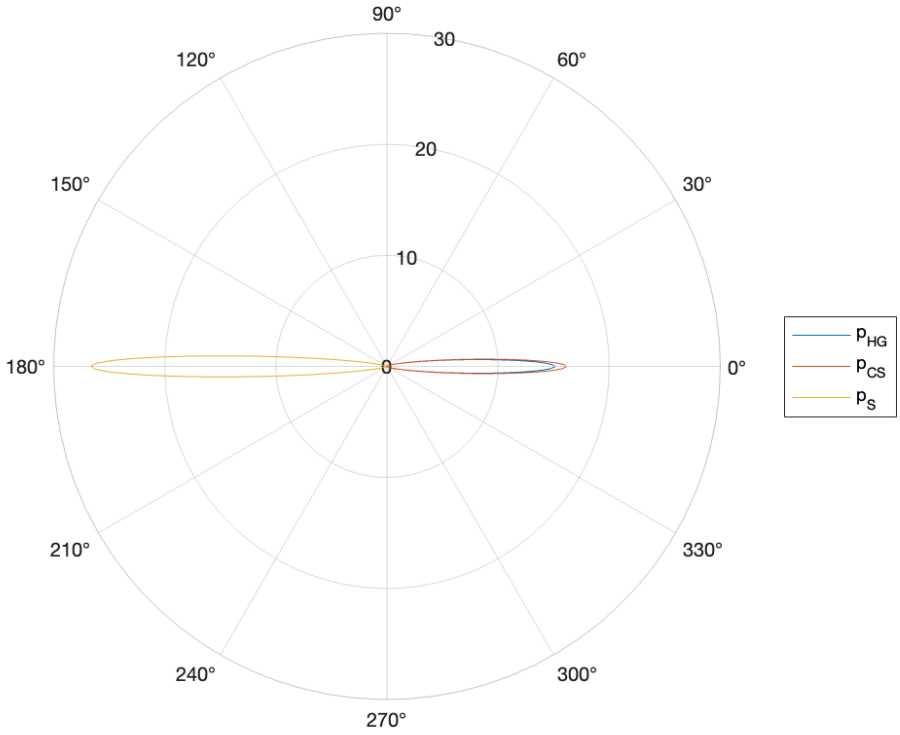


Figure 50: Henyey-Greenstein, Cornette-Shank, and Schlick phase function for $g = 0.9$ representation.

3.3 Summary

Chapter 3 describes how to create volumetric clouds in two steps: modeling and rendering. Section 3.1 covers cloud modeling, including defining its shape and post-processing stages to achieve a more organic shape. Section 3.2 is dedicated to cloud rendering. In Section 3.1, modeling starts with defining the cloud shape, followed by creating a volume of data for the cloud composed of voxels. Values to fill the voxels can be generated by algorithms or through 2D textures. Three 2D textures are created to store information about cloud coverage, height, and altitude. The Weather texture is a combination of these three textures. Adding noise to the textures can improve the cloud's appearance. Textures are created by game artists to achieve aesthetically pleasing and realistic clouds. The process of post-processing clouds improves their shape and creates more realistic edges. A height signal algorithm (parabola function) was used to round the edges. Then the Perlin and Worley noises were combined in two new 2D textures. The first texture was created to improve the cloud base shape and the second texture was used to erode the cloud edges, resulting in a more detailed cloud. The resulting density value was multiplied by a noise shape factor and a percentage of detail noise. The user can manipulate these noise factors to control the density and erosion of the clouds. Section 3.2 describes the process of cloud rendering using ray marching as the chosen

technique, which is a type of ray tracing. The density and illumination of each pixel are calculated for each step along a ray. The light density is calculated for each pixel through light marching. The Beer-Lambert law is used to compute the cloud transmittance, which uses an absorption and a scattering coefficient. The scattering coefficient has to be higher, and the absorb value smaller, to create realistic cloud lighting. The phase function is used to create the In-scattering effect or silver-lining effect, and the user can choose between several phase functions.

Chapter 4

Results

Chapter 4 presents the obtained results. In Section 4.1 the required resources and run-time performance of the described implementation in Chapter 3 are presented. Section 4.2 explores some visual parameters and shows the final results of this implementation. Rendering very realistic clouds is a complicated task that requires considerable computational resources. Thus, it is important to analyze the performance of this cloud implementation.

4.1 Performance

NAU3D (Ramires and Oliveira [2020]) was the application used to implement these clouds. The programming language used was OpenGL Shading Language (GLSL) and it was made in Microsoft Visual Studio 2019. This section describes the measured run-time performance on a PC with the hardware configuration presented in Table 11.

The required resources for this implementation are presented in Table 10. Five textures were used in this implementation. In Table 10 the format and size are presented for five different textures.

Texture	Format	Size
Weather texture	RGB	256 x 256
Shape texture	RGBA	128 x 128 x 128
Detail texture	RGB	32 x 32 x 32
Clouds texture	RGBA32F	Gridsize x Gridsize x Gridsize/2
Sky	RGBA32F	Gridsize x Gridsize x Gridsize

Table 10: Required resources.

Table 11 shows the specific hardware configuration of the PC used during the performance evaluation

of this implementation.

Hardware		Released year
Graphics card	NVIDIA GeForce RTX 2060	2019
Processor	Intel Core i5-9600KF CPU 3.70 GHz	2019

Table 11: Hardware configuration.

The implemented solution allows the user to change some parameters to customize the appearance of the cloud. Two possible parameters that can be adjusted are the number of ray marching steps and the number of light ray marching steps inside the box. These parameters are part of the rendering algorithm explained in Section 3.2. The number of steps and light steps can influence the computational cost and the execution time that the program takes to draw the clouds. Other parameters that can influence the computational time cost are the screen resolution, and the texture size used to draw the clouds (grid size).

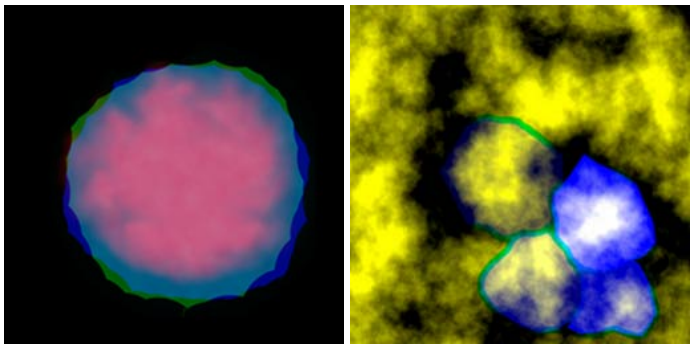
Tests were performed to understand the weight and effect of changing these parameters. The tests also allow for identifying the best parameters configuration. So, there is a balance between the time cost, computational resources, and the visual results of the clouds.

The measured values for each test were the GPU time (ms) and the number of Frames Per Second (FPS). That is, the time it takes to draw only the clouds on the scene and how many frames of all scenery (clouds and sky) it is possible to render per second.

These tests were done for the two different textures represented in Figure 51, one with just one cloud (the stored values in the texture are identical) and another with several clouds (texture with a greater variety of stored values). Thus, it is possible to investigate if the number of pixels with different values influences the computational time cost. The first tests were made with the created texture represented in Figure 51a and the other tests with the texture depicted in Figure 51b.

4.1.1 Number of eye steps and light steps

The fixed screen resolution used for these tests was 800x600 pixels with a texture (grid size) of 256x256 pixels. The bottom of the volume box is at a relative altitude to the earth's surface of 1500 m. The height of the volume is 500 m and both width and length are 2000 m.



(a) Texture with one cloud (texture 1).

(b) Texture by Högfeldt [2016] (texture 2).

Figure 51: Textures used to perform the tests.

Number of eye steps in the ray march

In order to see if the computational cost of the eye steps can get worse when the light step parameter is larger, tests were performed with two different constant light step parameters (light step = 1 and light step = 32).

Figure 52 shows the computational cost evolution for different ray march eye steps with two different light steps. These tests are only for the cloud shader. Figure 52 shows the time spent on GPU for the textures depicted in Figure 51a (texture 1) and Figure 51b (texture 2), respectively.

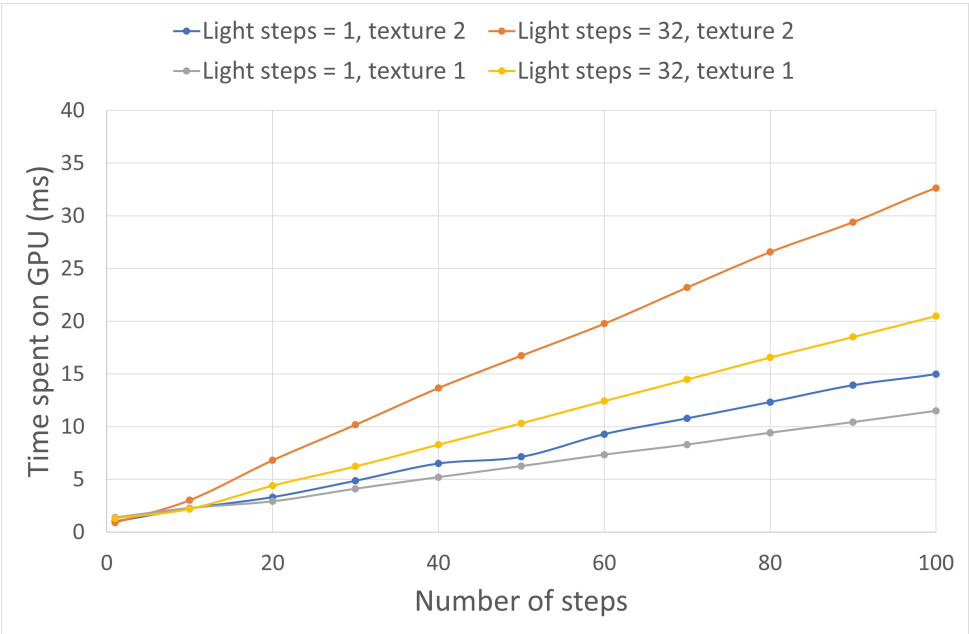


Figure 52: Computational cost evolution for different ray marching eye steps using the different textures.

In the graphic represented in Figure 52, cloud draw time spent on GPU (ms) increases linearly with

the number of eye steps, both for light steps equal to 1 and 32. When the number of light steps is 32, the time spent on the GPU is greater than when the number of light steps is one. It is possible to observe that the time spent on the GPU for texture 2 is higher when compared with the time spent on GPU for texture 1.

Figure 53 shows the evolution of FPS for both, sky and clouds. FPS decreases as the number of eye steps increases. When the number of light steps equals 32, the FPS is lower than when the light steps equal 1. Overall, FPS for Texture 1 is higher than for Texture 2.

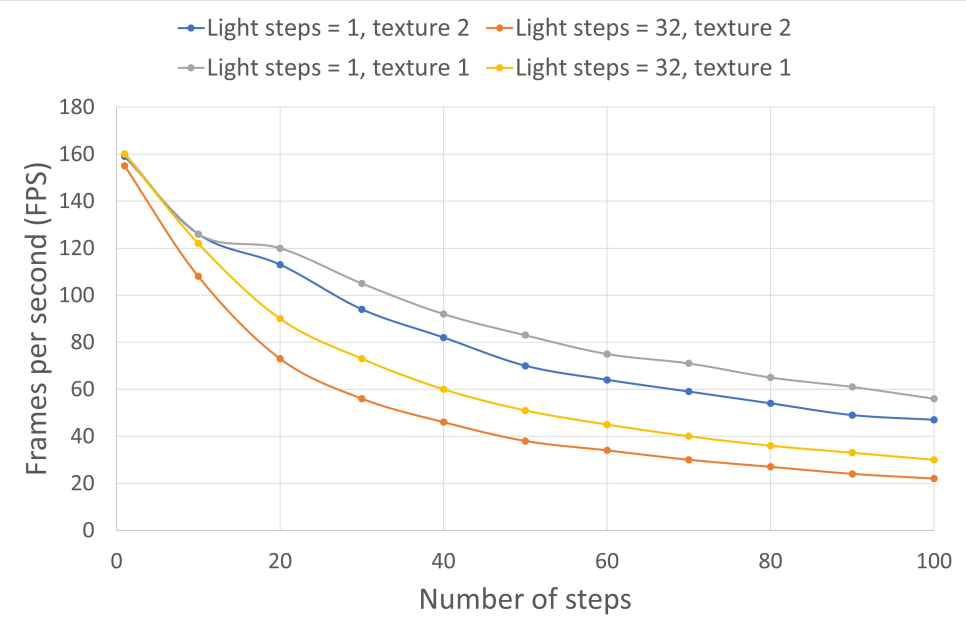
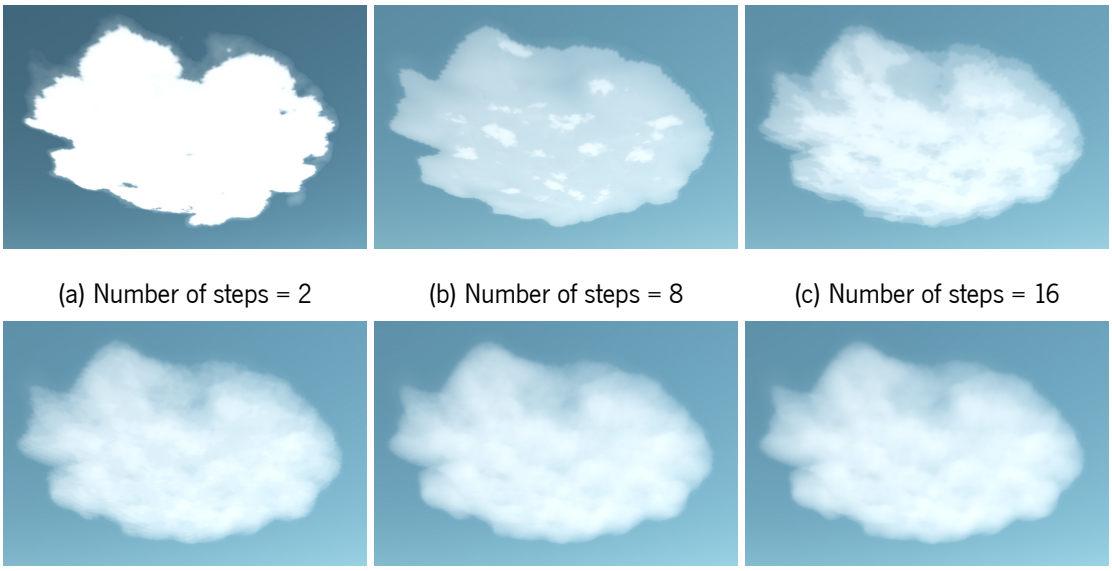


Figure 53: FPS evolution for different ray marching steps using the different textures.

It should be noted that FPS (Figure 53) is not exactly the inverse of the time spent on GPU represented in Figure 52 since this figure just includes the time spent on GPU to draw the clouds. On the other hand, FPS is measured for both clouds and the sky.

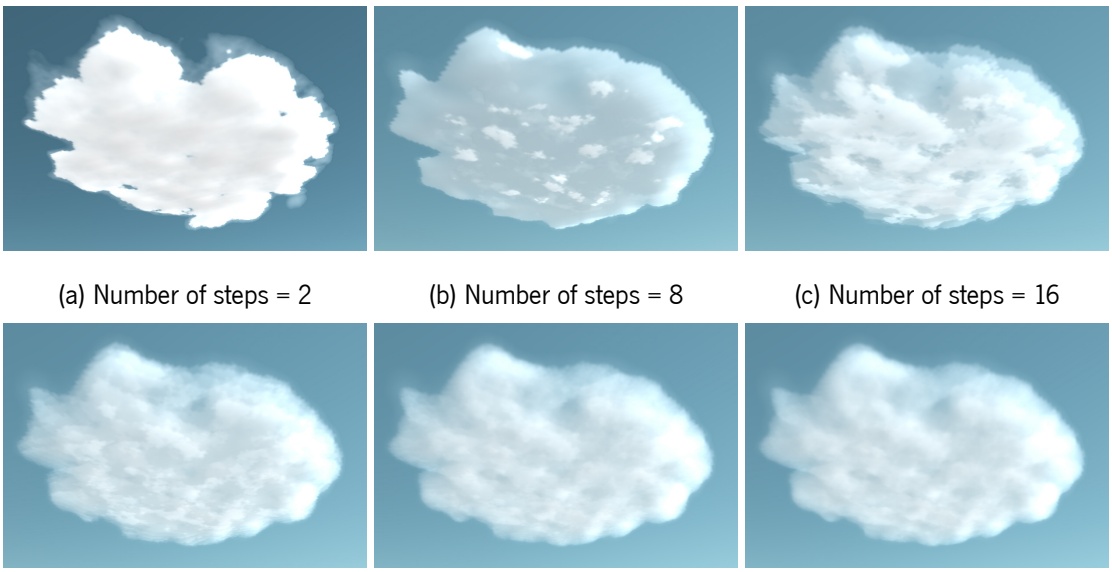
Figure 54 shows the cloud evolution for a different number of eye steps, generated with texture Figure 51a when the light step is equal to one. Figure 55 also shows the cloud generated with texture Figure 51a, but when the light step is equal to 32. As the number of eye steps increases, both clouds become more defined. When the light step is 32, the cloud lightning is already visible, so the cloud is more realistic. Moreover, observing the evolution of the different pictures represented in Figure 54 and in Figure 55, it is possible to see that after 64 eye steps, there is not much visual difference.



(a) Number of steps = 2 (b) Number of steps = 8 (c) Number of steps = 16

(d) Number of steps = 32 (e) Number of steps = 64 (f) Number of steps = 128

Figure 54: Cloud eye step evolution when light step = 1.



(a) Number of steps = 2 (b) Number of steps = 8 (c) Number of steps = 16

(d) Number of steps = 32 (e) Number of steps = 64 (f) Number of steps = 128

Figure 55: Cloud eye step evolution when light step = 32.

Number of light steps in raymarch

The effect of the number of light steps was also studied for the two different textures represented in Figure 51. These tests are only for cloud shader. These tests were performed with two different fixed values for the number of eye steps (step = 32 and step = 64).

In the graphic represented in Figure 56, time increases approximately linearly with the number of light steps. When the number of steps is greater, the time spent on the GPU by increasing the number of light

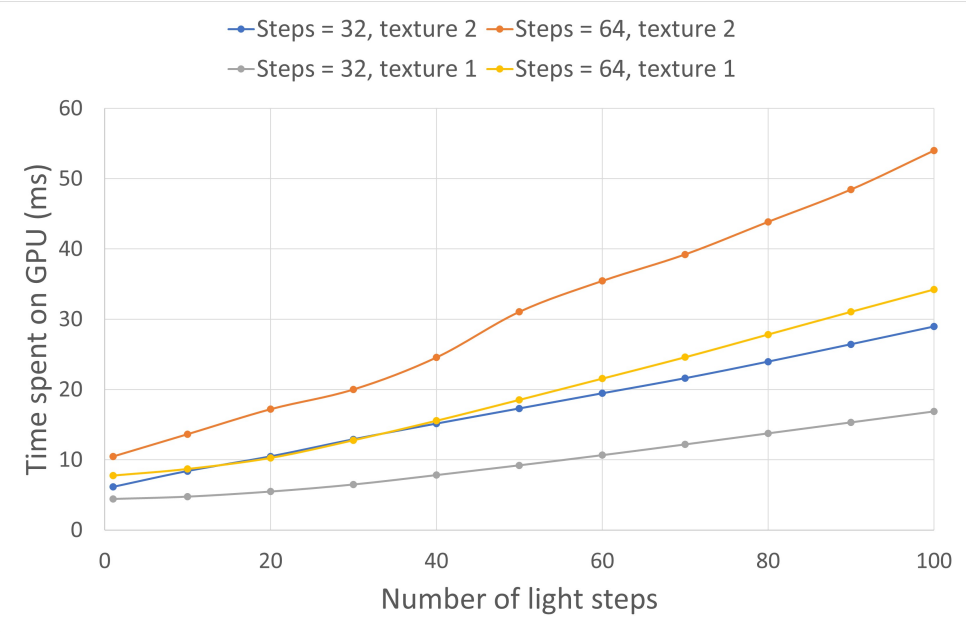


Figure 56: Computational cost evolution for different light ray marching steps using the different textures.

steps is greater too. Furthermore, it is possible to observe that texture 1 requires less computational time on GPU than texture 2.

Figure 57 represents the variation of FPS for both, sky and clouds using two different textures. FPS decreases as the number of light steps increases. When the number of steps is equal to 64, the FPS achieved lower values than when the number of steps is equal to 32. In general, FPS for Texture 1 is higher than for Texture 2.

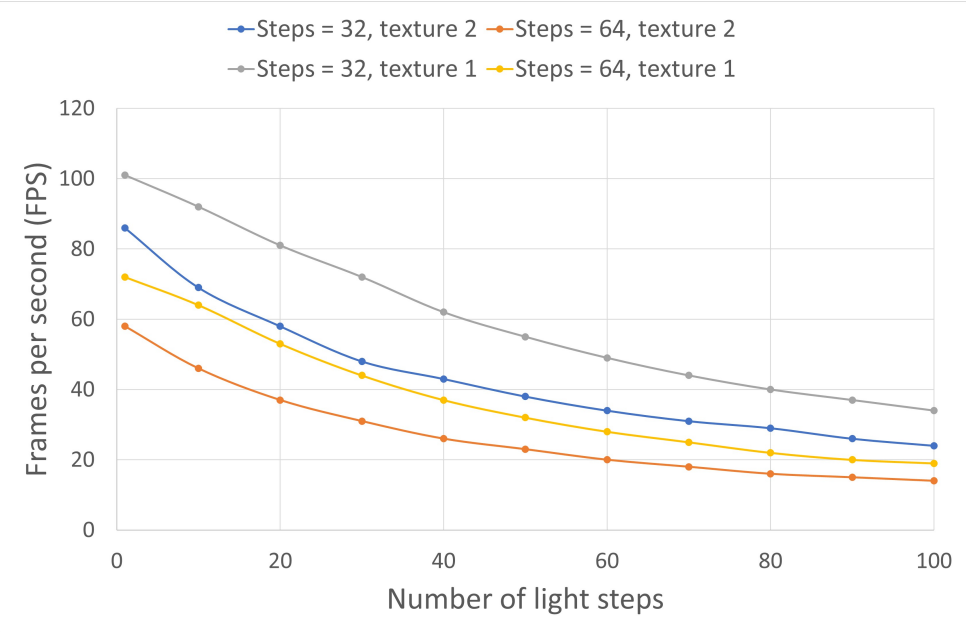
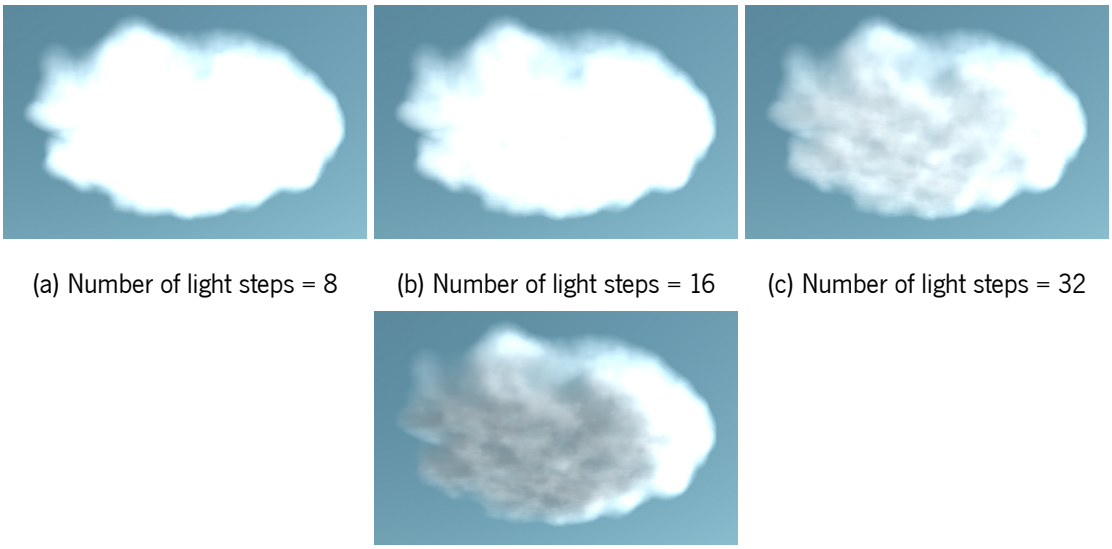


Figure 57: FPS evolution for different light ray marching steps using the different textures.

Figure 58 shows the cloud evolution for a different number of light steps, generated with texture Figure 51a when the eye step is equal to 32. As the number of light steps increases, the cloud illumination is more visible. Moreover, observing the evolution of the different pictures represented in Figure 58, it is possible to see that when the number of light steps is equal to 32, the cloud is more realistic.



(a) Number of light steps = 8 (b) Number of light steps = 16 (c) Number of light steps = 32

(d) Number of light steps = 64

Figure 58: Cloud light step evolution when step = 32.

4.1.2 Grid size

Tests were also performed for different grid sizes: 32x32x16, 64x64x32, 128x128x64, 256x256x128, and for 512x512x256. Figure 59 represents the computational cost (time spent on GPU) for the different grid sizes using texture in Figure 51a.

Analyzing the graphic of Figure 59, the computational cost variation is small for grid sizes inferior to 512x512x256. The computational time does not increase a lot over these grid sizes.

In spite of the computer used for the tests having enough dedicated memory, when the grid size is set to 512x512x256, the time spent on GPU is significantly larger than the other four grid sizes. The number of computational calculations required by using this grid size is larger. However, these values were not expected to be so large. This could be happening due to poor optimization of the program to render the clouds.

Figure 60 represents the FPS (for both clouds and sky) for the different grid sizes using texture in Figure 51a. The number of FPS decreases with the grid size.

It is possible to see a larger visual difference when the grid size and texture size used are changed in

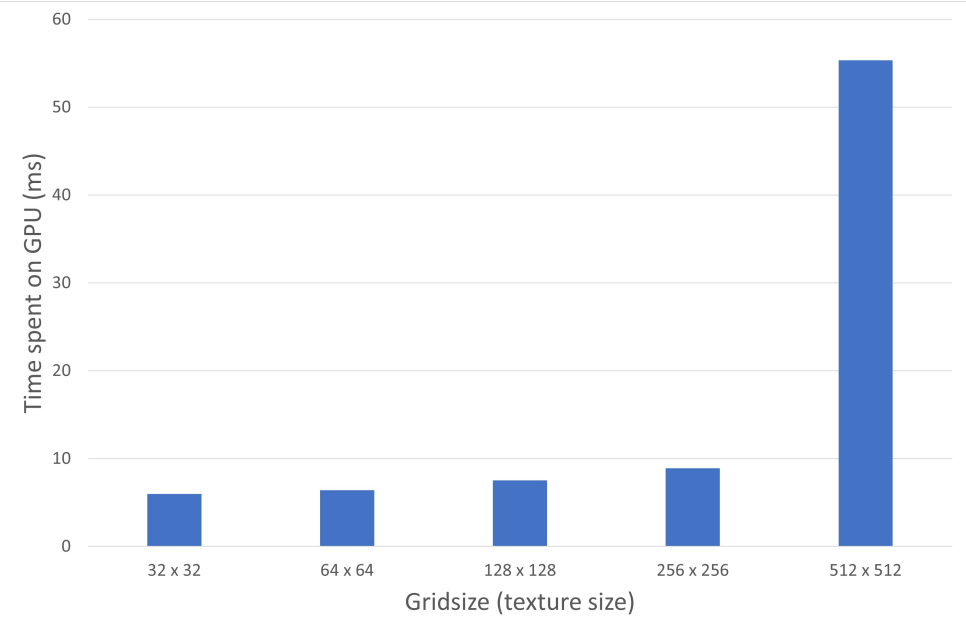


Figure 59: Computational cost for different grid sizes using texture in Figure 51a.

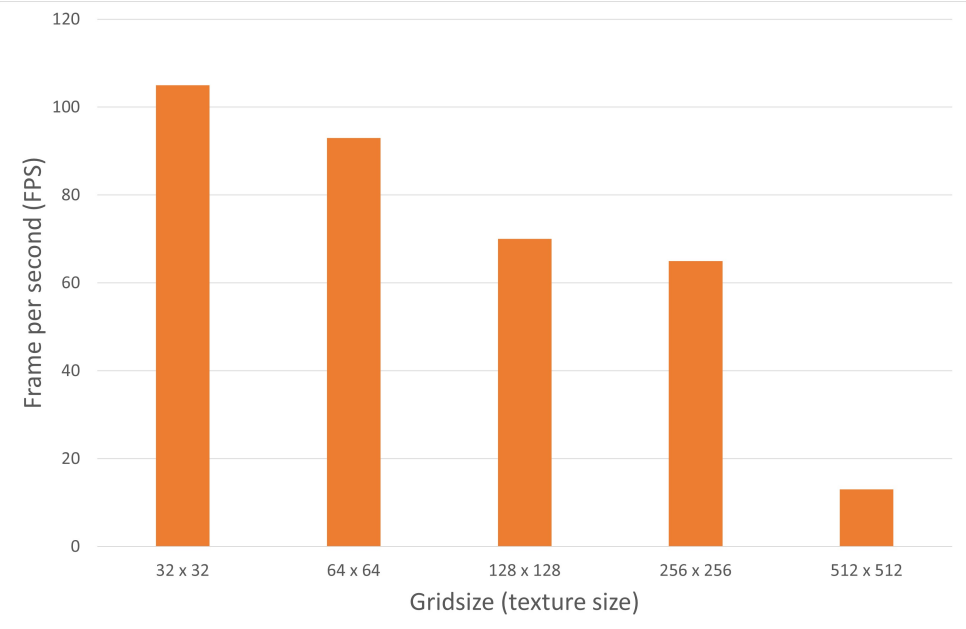
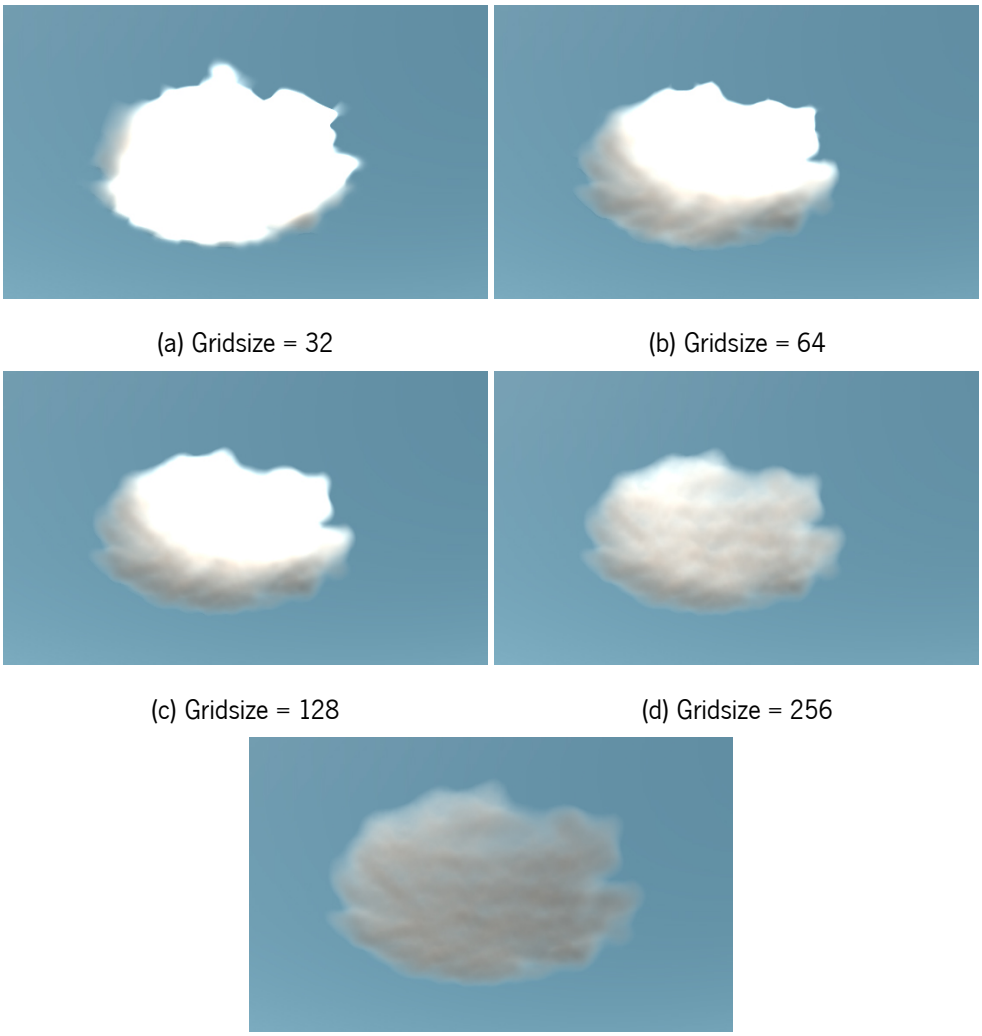


Figure 60: FPS for different grid sizes using texture in Figure 51a.

Figure 61. The smaller the texture size, the less sharp cloud, and the more blurry. It can be concluded that to have a balance between visual results and computational times, the best combination is to use a 256x256x128 grid size on this computer.



(a) Gridsize = 32 (b) Gridsize = 64

(c) Gridsize = 128 (d) Gridsize = 256

(e) Gridsize = 512

Figure 61: Cloud with different gridsize.

4.1.3 Screen Resolution

Tests were also done to see how the program behaves at different screen resolutions. Table 12 presents the considered screen resolution sizes.

QVGA	nHD	VGA	WVGA	Super VGA	qHD	XGA
320×240	640×360	640×480	800×480	800×600	960×540	1024×768
HD	WXGA	HD+	FHD	2k	4k	
1280×720	1366×768	1600×900	1920×1080	2048×1080	3840×2160	

Table 12: Screen resolution sizes.

Figure 62 represents the computational cost for the different screen resolutions using texture in Fig-

ure 51a. This time spent on GPU is only for cloud shader. As expected, the higher the resolution of the screen, the longer the GPU time (Figure 62). This happens because the larger the screen, the more pixels the program has to render. The screen resolution that requires more time on GPU are FHD, 2K, and 4K. For the other screen resolutions, the time cost does not vary significantly.

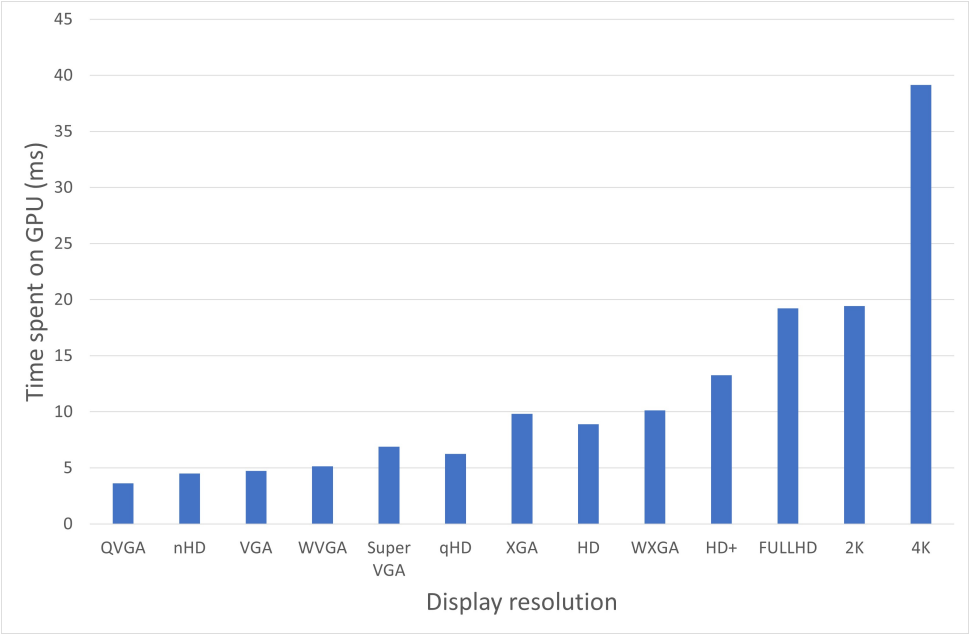


Figure 62: Computational cost for different screen resolutions using texture in Figure 51a.

Figure 63 represents the FPS (for both clouds and sky) for the different screen resolution using texture in Figure 51a. The higher the screen resolution, the lower the number of FPS.

4.1.4 Conclusions

Taking into account the results of the tests, it can be concluded that:

- computational cost depends on the Weather texture: textures with more clouds, and pixels with different values, tend to take longer to render and have lower FPS;
- as the eye steps and light steps parameters increase, the GPU time of the cloud shader increases, and the number of FPS decreases;
- the most balanced option in terms of computational cost and visual result, taking into account Figure 55 and Figure 58, is eye steps = 64 and light step = 32;
- the best option for grid size, analyzing the results presented in Figure 59, Figure 60, and Figure 61, is 256x256x128;

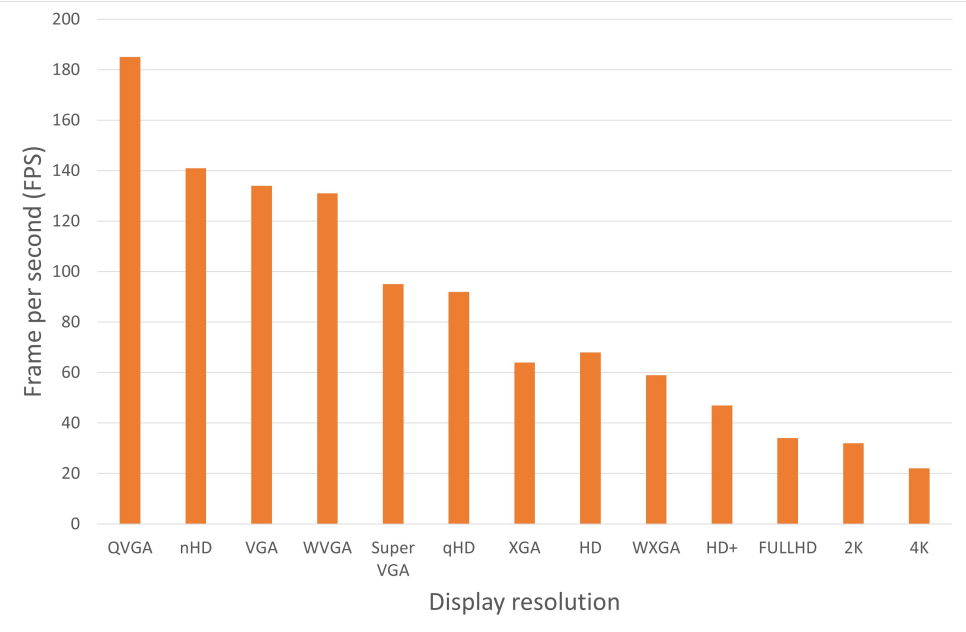


Figure 63: FPS for different screen resolutions using texture in Figure 51a.

- there is no significant variation on time spent on GPU between QVGA and WXGA. The time on GPU of FHD and 2k resolution double to WXGA. The time on 4k screen resolution is almost four times the time spent on GPU for WXGA resolution;
- the higher the screen resolution, the lower the number of FPS. In general, it is considered that the minimum acceptable rate is 30 FPS. For 4K the number of fps achieved is inferior to that value.

4.2 Visual results

This section shows some of the final results of the code implementation made in this dissertation. By changing the Weather texture, noise textures, and some parameters, it is possible to create different types of clouds and different lighting effects. For all cloud examples presented in this section, the eye step parameter is always equal to 64, the light step equal to 32, and the grid size is 256x256x128. The bottom of the volume box is located at a relative altitude to the earth's radius of 1500 m. The height of the volume is 500 m and both width and length are 2000 m.

In Figure 64, it is possible to see *Cumulus* clouds resulting from this implementation. The base noise multiplier factor is equal to 2, the percentage of detail noise parameter is equal to 0.9, the absorption coefficient is equal to 0.2, the scattering coefficient is equal to 0.9, the brightness is equal to 0.49, the phase function used was Henyey-Greenstein (HG), and the phase function multiplier is 4.

Figure 65 shows the same *Cumulus* cloud but at a different day hour, at midday (12 a.m.), the base

Figure 64: *Cumulus* cloud at 5 p.m.

noise multiplier factor is now equal to 4, the scattering coefficient is equal to 0.3, and the absorption coefficient is equal to 0.1. This change made the clouds denser and less yellow from sunlight.

Figure 65: Denser *Cumulus* cloud at 12 a.m.

The cloud's color is related to the light source color. Positioning the sun close to the horizon, the sun's color becomes more orange, creating the effect of a sunset/sunrise in clouds (Figure 66 and Figure 67).

(a) Sunrise clouds (5 a.m).

(b) Sunset clouds (8 p.m)

Figure 66: Sunlight events.

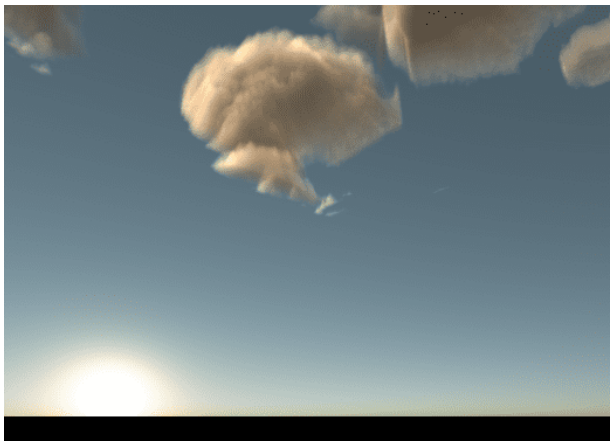


Figure 67: Sunset cloud in another perspective.

In Figure 68, it is possible to observe the *silver-lining* effect (Figure 7a) when the sun is behind the clouds.

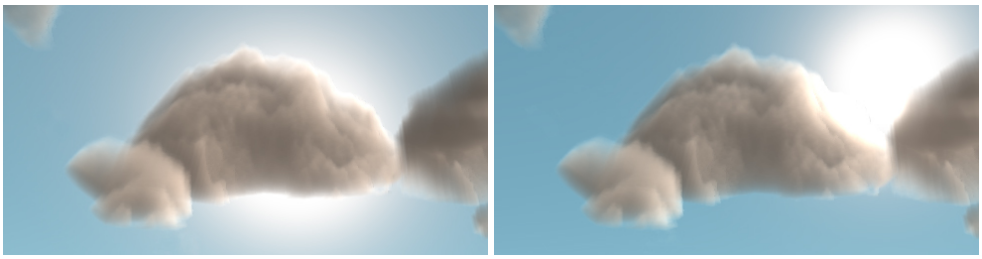


Figure 68: Silver-lining effect with light step = 32.

If the number of light steps is increased to 64, the silver-lining effect becomes even more visible (Figure 69 and Figure 70).

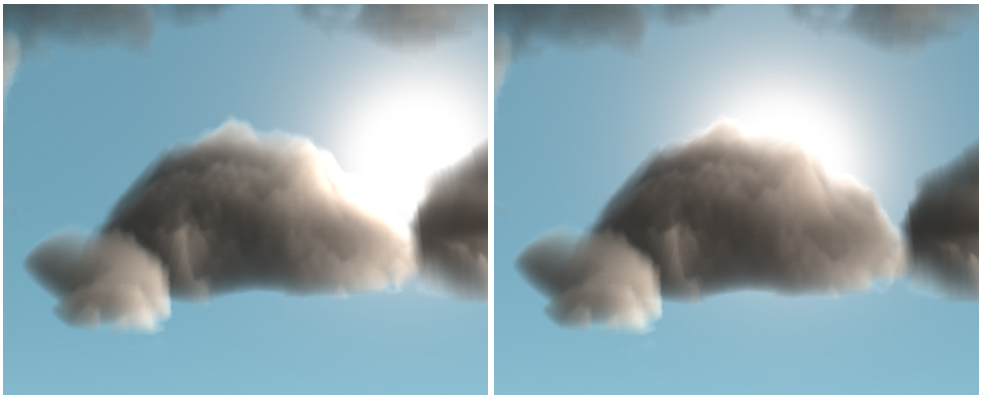


Figure 69: Silver-lining effect with light step = 64.

It is possible to create other types of clouds like *Cirrocumulus* (Figure 4b) and *Altocumulus* (Figure 3a), but a different Weather texture is required. The Weather texture to create this type of cloud has a higher

Figure 70: Silver-lining effect with light step = 80.

coverage value and covers the entire 2D texture. In addition, more Perlin noise is added. The *Altocumulus* clouds (Figure 71) are gray (sometimes white) that present their own shadows and take the form of fibrous rolls or diffuse sheets.

Figure 71: *Altocumulus* clouds.

The *Cirrocumulus* (Figure 72) are very visually identical to *Altocumulus* but the main differences between them are that the *Cirrocumulus* clouds are usually whiter and semi-transparent and, *Altocumulus* is at a mid-stage altitude while *Cirrocumulus* is at a high-level altitude. To differentiate them, it is needed to create two different Weather textures, one with clouds at higher altitudes than the other.

Figure 71 and Figure 72 depict the *Altocumulus* and *Cirrocumulus* clouds generated with this implementation, respectively.

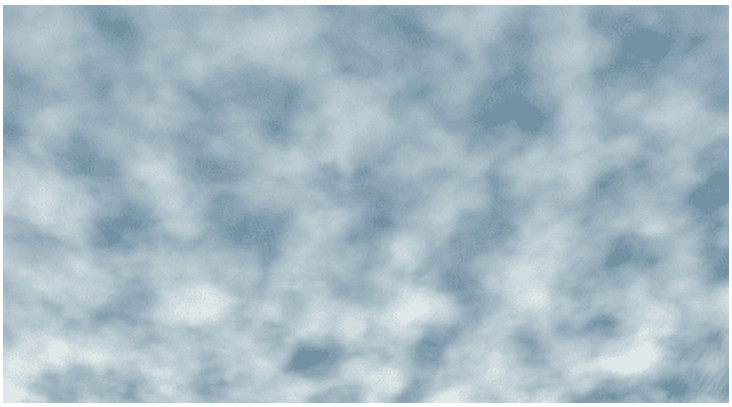


Figure 72: *Cirrocumulus* clouds.

Using a Weather texture with the maximum coverage and height values, and the minimum altitude value, it is possible to create the *Nimbostratus* clouds (Figure 3c), rain clouds (Figure 6b). Other parameters were altered to create this effect, such as increasing the noise multiplier factor to 8 and setting the absorption coefficient to the maximum value. Figure 73 represents the *Nimbostratus* clouds generated with this implementation.

Figure 73: *Nimbostratus* clouds.

4.3 Summary

Chapter 4 presents the results obtained from the implementation of this cloud implementation. The chapter begins with a discussion of the required resources, including the PC hardware configuration, and the analysis of the runtime performance of this implementation. This implementation allows the user to adjust some parameters to customize the appearance of the cloud.

Tests were performed to understand the visual and computational cost effect of changing parameters

(the number of eye steps, light steps, the grid size, and the screen resolution). The measured values for each test were the GPU time and the number of frames per second.

Overall, this chapter discusses the balance between the time cost, computational resources, and visual results of the clouds. The results show that the computational cost of the implementation increases linearly with the number of eye steps and the number of light steps. It is also shown that the computational time cost depends on the Weather texture.

Finally, some cloud results of this implementation are shown, including the different types of cloud and light effects that can be created by changing some parameters in the application.

Chapter 5

Conclusions and future work

In this chapter, conclusions and future work are presented. Section 5.1 addresses the main conclusions of the process of developing this dissertation, from the cloud physics and rendering techniques to the cloud implementation in computer graphics. In Section 5.2, some future improvements are proposed.

5.1 Conclusions

In this dissertation, the techniques and algorithms for realistic cloud rendering in computer graphics are comprehensively studied. The main objective was to explore techniques for cloud modeling and rendering. The most advanced methods and techniques were reviewed in order to identify the key challenges and opportunities for building a code and implementing realistic clouds. The modeling technique adopted follows the Schneider and Vos [2015], Hillaire [2016a], and Högfeldt [2016] approaches. For cloud modeling, it was opted to have a volume, in which voxels were filled with information stored in a texture. A texture, called Weather texture, can be hard to be created when the goal is to generate aesthetic and realistic clouds. Some issues and limitations of this model are that this technique does not support clouds that overlap in altitude and the clouds are bounded by the volume box size. A possible solution was to place several overlapping boxes, but this would greatly increase the time cost. For cloud illumination, the equations from Jarosz [2008] and Scratchapixel [2022b] were adapted. In this work, ray marching was used. This is a rendering algorithm often used to create realistic lighting but costs more in computational terms. A cloud generator was implemented based on these techniques. In this implementation, it is possible to create different types of clouds by changing some customized parameters, such as the base and detail cloud noise multiplier, the brightness, the scattering, and the absorption coefficients. The Weather texture controls the height of the cloud, the cloud altitude from the bottom of the volume box, and the coverage. Creating different Weather textures can generate different types of clouds.

This implementation using volumetric clouds and physically-based lighting achieved satisfactory results

in terms of visual fidelity, computational efficiency, and scalability. However, the final visual results are not at the level of some state-of-the-art results, mainly in terms of realism. The proposed implementation can be improved in order to generate more realistic clouds in the future. This could happen by creating better weather textures, improving the noise application, and fixing some issues and limitations. Despite that, the evaluation of the method on various test cases demonstrated its robustness and flexibility in simulating different types of clouds and lighting conditions.

5.2 Future work

Various corrections and improvements can be proposed to overcome some of the limitations encountered in this dissertation. These future work proposals are aimed at adding value to the created code by producing even more realistic and optimized clouds. As future work, some additions that could further enrich this project, as well as possible tasks to support the implementation of realistic clouds are listed below.

1. **Development of a Weather texture generator application**: 2D textures are used in this cloud implementation and they are responsible for creating the different types of clouds. There are a large number of combinations of noise and values to draw these weather textures. Although a Python code was separately created that generates a Weather texture, it would be interesting to integrate a code in order to create a Weather texture program generator. That application could create textures that automatically generate realistic and beautiful clouds.

2. **Clouds animation**: Extend the implementation to introduce cloud animation in order to transform static clouds into dynamic ones. Throughout this project, there were some tests to animate the cloud, but without very realistic movements and results. It is possible to develop further simulations of clouds to obtain realistic animated results, including interaction with wind, the formation of air currents, and modeling natural cloud movements.

3. **Algorithm optimization**: The need for further research on optimizing the rendering pipeline. To investigate methods to optimize the rendering of clouds on a large scale, including overcoming the box limitation.

4. **Clouds and surrounding environment**: In this implementation, only the interaction of the cloud with the sky and sun was considered. It would be interesting to improve the cloud simulation technique, including the relationship of the cloud with the terrain.

5. **Rain**: Although in this implementation it is possible for the user to create gray color clouds by manipulating the parameters, it would be interesting to simulate rain. Exploring techniques to simulate clouds with the meteorological condition of rain, not just the clouds but the entire atmospheric environment.

Bibliography

3D-Ace. Different rendering techniques: Pros, cons and tips, 2021. URL https://3d-ace.com/blog/different-rendering-techniques/.

T. Alldieck, D. H. Lundtoft, N. Montanari, I. A. Nikolov, I. G. Vlaykov, and C. B. Madsen. Modelling of clouds from a hemispherical image. In *TPCG*, pages 17–24, 2014.

P. Blasi, B. Le Saec, and C. Schlick. A rendering algorithm for discrete volume density objects. In *Computer Graphics Forum*, volume 12, pages 201–210. Wiley Online Library, 1993.

A. Bouthors and F. Neyret. Modeling clouds shape. In *Eurographics (short papers)*. Eurographics Association, 2004.

A. Bouthors, F. Neyret, and S. Lefebvre. Real-time realistic illumination and shading of stratiform clouds. In *Proceedings of the Second Eurographics Conference on Natural Phenomena*, NPH'06, page 41–50, Goslar, DEU, 2006. Eurographics Association. ISBN 390567338X.

V. Buthiran. High-altitude cirrus clouds and climate. *Resonance*, 9:23–32, 03 2004. doi: 10.1007/BF02834985.

S. Chandrasekhar. *Radiative Transfer*. Dover Books on Physics. Dover Publications, 2013. ISBN 9780486318455. URL https://books.google.pt/books?id=1YHCAgAAQBAJ.

W. M. Cornette and J. G. Shanks. Physically reasonable analytic expression for the single-scattering phase function. *Appl. Opt.*, 31(16):3152–3160, Jun 1992. doi: 10.1364/AO.31.003152. URL https://opg.optica.org/ao/abstract.cfm?URI=ao-31-16-3152.

Y. Dobashi, T. Nishita, H. Yamashita, and T. Okita. Modeling of clouds from satellite images using metaballs. In *Proceedings Pacific Graphics' 98. Sixth Pacific Conference on Computer Graphics and Applications (Cat. No. 98EX208)*, pages 53–60. IEEE, 1998.

Y. Dobashi, K. Kaneda, H. Yamashita, T. Okita, and T. Nishita. A simple, efficient method for realistic animation of clouds. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 19–28, 2000.

Y. Dobashi, Y. Shinzo, and T. Yamamoto. Modeling of clouds from a single photograph. In *Computer Graphics Forum*, volume 29, pages 2083–2090. Wiley Online Library, 2010.

Y. Dobashi, W. Iwasaki, A. Ono, T. Yamamoto, Y. Yue, and T. Nishita. An inverse problem approach for automatically adjusting the parameters for rendering clouds using photographs. *ACM Transactions on Graphics (TOG)*, 31(6):1–10, 2012.

D. Ebert. Volumetric modeling with implicit functions: a cloud is born. In *SIGGRAPH Visual Proceedings*, page 147, 1997.

P. Elinas and W. Stuerzlinger. Real-time rendering of 3d clouds. *Journal of Graphics Tools*, 5(4):33–45, 2000.

G. Y. Gardner. Visual simulation of clouds. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 297–304, 1985.

P. Goswami. A survey of modeling, rendering and animation of clouds in computer graphics. *The Visual Computer*, 37(7):1931–1948, 2021.

L. Henyey and J. Greenstein. Diffuse radiation in the galaxy. *Astrophysical Journal*, 93:70–83, jan 1941. doi: 10.1086/144246.

S. Hillaire. Physically based sky, atmosphere and cloud rendering in frostbite. In *ACM SIGGRAPH*, 2016a.

S. Hillaire. Tileable volume noise, 2016b. URL https://github.com/sebh/TileableVolumeNoise.

R. Högfeldt. Convincing cloud rendering–an implementation of real-time dynamic volumetric clouds in frostbite. Master's thesis, University of Gothenburg, 2016.

R. Hufnagel and M. Held. A survey of cloud lighting and rendering techniques. *J. WSCG*, 20, 01 2012.

R. Hufnagel, M. Held, and F. Schröder. Large-scale, realistic cloud visualization based on weather forecast data. In *Proc. IASTED Int. Conf. Comput. Graph. and Imaging (CGIM'07)*, pages 54–59, 2007.

W. Jarosz. *Efficient Monte Carlo methods for light transport in scattering media*. University of California, San Diego, 2008.

S. Kallweit, T. Müller, B. Mcwilliams, M. Gross, and J. Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Transactions on Graphics (TOG)*, 36(6):1–11, 2017.

B. B. Mandelbrot and J. W. Van Ness. Fractional brownian motions, fractional noises and applications. *SIAM review*, 10(4):422–437, 1968.

B. Miller, K. Museth, D. Penney, and N. B. Zafar. Cloud modeling and rendering for "puss in boots". *ACM SIGGRAPH Talks*, 2012.

T. Nishita and Y. Dobashi. Modeling and rendering methods of clouds. In *Proceedings. Seventh Pacific Conference on Computer Graphics and Applications (Cat. No. PR00293)*, pages 218–219. IEEE, 1999.

T. Nishita, Y. Dobashi, K. Kaneda, and H. Yamashita. Display method of the sky color taking into account multiple scattering. In *Pacific Graphics*, volume 96, pages 117–132. Department of Computer Science, National Tsing Hua University Taiwan, 1996.

K.-C. Peng and T. Chen. Incorporating cloud distribution in sky representation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2152–2159, 2013.

K. Perlin. Real time responsive animation with personality. *IEEE transactions on visualization and Computer Graphics*, 1(1):5–15, 1995.

A. Ramires and B. Oliveira. Nau 3d, 2020. URL https://nau3d.di.uminho.pt/nau3d/.

J. Rayleigh. *On the Scattering of Light by Small Particles*, volume 41. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science, 1871. URL https://books.google.pt/books?id=YbNuMwEACAAJ.

P. Rideout. Single-pass raycasting at the little grasshopper, 2021. URL https://prideout.net/blog/old/blog/index.html@p=64.html.

A. Schneider and N. Vos. The real-time volumetric cloudscapes of horizon: Zero dawn. *Advances in Real-time Rendering, SIGGRAPH*, 2015.

J. Schpok, J. Simons, D. S. Ebert, and C. Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 160–166, 2003.

Scratchapixel. Volume rendering for developers: Foundations, 2022a. URL https://www.scratchapixel.com/lessons/3d-basic-rendering/volume-rendering-for-developers/intro-volume-rendering.html.

Scratchapixel. From the radiative transfer equation to the volume rendering equation, 2022b. URL https://www.scratchapixel.com/lessons/3d-basic-rendering/volume-rendering-for-developers/volume-rendering-summary-equations.html.

M. Stiver, A. Baker, A. Runions, and F. Samavati. Sketch based volumetric clouds. In *Smart Graphics: 10th International Symposium on Smart Graphics, Banff, Canada, June 24-26, 2010 Proceedings 10*, pages 1–12. Springer, 2010.

A. Trembilski and A. Broßler. Surface-based efficient cloud visualisation for animation applications. In *WSCG*, pages 453–460, 2002.

W. Wenke, G. Yumeng, X. Min, and L. Sikun. Automatic generation of large scale 3d cloud based on weather forecast data. In *2012 International Conference on Virtual Reality and Visualization*, pages 69–73. IEEE, 2012.

J. Wither, A. Bouthors, and M.-P. Cani. Rapid sketch modeling of clouds. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling (SBIM)*, pages 113–118. Eurographics Association, 2008.

World Meteorological Organization (WMO). Definitions of clouds | international cloud atlas, 2022. URL https://cloudatlas.wmo.int/en/clouds-definitions.html.

S. Worley. A cellular texture basis function. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, page 291–294, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917464. doi: 10.1145/237170.237267. URL https://doi.org/10.1145/237170.237267.

C.-M. Yu and C.-M. Wang. An effective framework for cloud modeling, rendering, and morphing. *Journal of information science and engineering*, 27(3):891–913, 2011.

C. Yuan, X. Liang, S. Hao, and G. Yang. Modeling large scale clouds from satellite images. In B. Lévy, X. Tong, and K. Yin, editors, *21st Pacific Conference on Computer Graphics and Applications, PG 2013 - Short Papers, Singapore, October 7-9, 2013*. Eurographics Association, 2013. doi: 10.2312/PE.PG.PG2013short.047-052. URL https://doi.org/10.2312/PE.PG.PG2013short.047-052.

C. Yuan, X. Liang, S. Hao, Y. Qi, and Q. Zhao. Modelling cumulus cloud shape from a single image. In *Computer Graphics Forum*, volume 33, pages 288–297. Wiley Online Library, 2014.