Real-Time Rendering of Atmospheric Clouds

Parker Ford

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2024

Committee:

Kelvin Sung, Chair

Clark Olson, Committee Member

Yusuf Pisan, Committee Member

Program Authorized to Offer Degree:

Computing and Software Systems

University of Washington

Abstract

Real-Time Rendering of Atmospheric Clouds

Parker Ford

Chair of the Supervisory Committee:

Professor Kelvin Sung

Computing and Software Systems

Rendering realistic clouds is an important aspect of creating believable virtual worlds. The detailed shapes and complex light interactions present in clouds make this a daunting task to complete in a real-time application. Our solution, based on Schneider's cloud modeling and Fong's volumetric rendering frameworks for low-altitude cloudscapes, supports realism and real-time performance. For efficient approximations of radiance measurements, we adopt Hillaire's energy-conserving integration method for light scattering. To simulate the effect of multiple light scattering, we followed Wrenninge's approach for computing the multi-bounce diffusion of light within a volume. To capture the details of light interreflection off microscopic water droplets, the complex behavior of Mie scattering is approximated with Jenderise and d'Eon's phase function modeling technique. To capture the details with nominal computational cost, we introduce a

temporal anti-aliasing strategy that unifies the sampling strategy for the area over a pixel and interval of volumetric participating media.

The resulting system is capable of rendering scenes consisting of expansive cloudscapes well within real-time requirements, achieving frame rates between 2 and 3 milliseconds on a typical machine. Users can adjust parameters to control various types of low-altitude cloud formations and weather conditions, with presets available for easily transitioning between settings. Our unique combination of techniques adopted in the volumetric rendering process enhances both efficiency and visual fidelity where the novel approach to volumetric temporal anti-aliasing efficiently and effectively unifies the sampling of pixel areas and volumetric intervals. Looking forward, this technique could be adapted for real-time applications such as video games or flight simulations. Further improvements could refine the cloud modeling system, incorporating procedural generation for high-altitude clouds, thus broadening the range of cloudscapes that can be represented. Additionally, our volumetric rendering framework could be paired with recent investigations into voxel-based cloud rendering.

TABLE OF CONTENTS

Chapter 1. INTRODUCTION

Clouds are an ever-present and beautiful fixture in both our physical and virtual skies. Their beauty and complexity make rendering them not just visually satisfying, but also a captivating and challenging area of research. The demand for virtually rendered clouds is high, with their applications spanning across several fields. For instance, weather simulations require highly realistic clouds to accurately simulate meteorological events [1], flight simulators rely on realistic cloud depictions to provide immersive and practical training experiences for pilots [2], and the entertainment industry, especially video games and films, utilizes aesthetically pleasing clouds to enhance the beauty and engagement of their outdoor environments [3], [4]. Figure 1.1 showcases the level of realism capable in modern cloud rendering techniques. In the realm of computer graphics, the task of rendering clouds has prompted significant research and continues to be an active area of investigation today.

Figure 1.1: An example of modern cloud rendering capabilities, provided by Novák et al [5].

The specific area of research we focus on is rendering clouds in real-time. Real-time rendering involves generating images quickly enough to provide immediate feedback. This process is challenging in the context of rendering clouds due to their complex nature. Many physical processes affect a cloud's shape, such as temperature, air pressure, altitude, and water density [6]. Additionally, the way light scatters through cloud particles makes accurate illumination modeling difficult [7]. The combination of these physical and optical complexities makes rendering clouds computationally expensive, and challenging to achieve in real-time [8].

Schneider proposes a computationally efficient procedural method for modeling cloud shape [4]. In this method, the shapes of clouds are based on several procedural noise functions. While the resulting cloud shapes are not based on the physical processes that form real-world clouds, they have a natural appearance and contain the characteristic details observed in clouds. Independently, Fong et al. propose a volumetric rendering framework capable of simulating the interaction of light as it travels through volumes of particles [9]. This technique can be applied to the realistic illumination of smoke, fog, and clouds.

Our solution is a novel cloud rendering platform that paired Schneider's modeling of cloud with an extended version of Fong et. al.'s volumetric rendering framework. Schneider's method is chosen due to its balance of computational efficiency and visually appealing results. Fong et al.'s volumetric rendering framework is selected for its rigor in theoretical foundation and flexibility in integrating other more advanced phenomena. Based on the rendering framework, for efficient approximations of radiance measurements, we adopt Hillaire's energy-conserving integration method for light scattering [10]. To simulate the effect of multiple light scattering, we follow Wrenninge's approach for computing the multi-bounce diffusion of light within a volume [11]. To capture the details of light interreflection off microscopic water droplets, the complex behavior of Mie scattering is approximated using Jenderise and d'Eon's phase function modeling technique [12].

Additionally, to alleviate artifacts from discretized sampling and achieve details with nominal computational cost, we introduce a novel temporal anti-aliasing strategy that unifies the sampling of pixel area and volumetric interval. The multi-frame pixel area sampling integrates a blue-noise distribution with an n-rooks offset, while the corresponding volumetric samples follow a coordinated stratification strategy. The resulting system is capable of rendering expansive cloudscapes within real-time requirements, achieving frame rates between 2 and 3 milliseconds on typical systems. Users can adjust parameters to control various types of low-altitude cloud formations and weather conditions. Our clouds also react realistically to light, enhancing overall visual fidelity.

This paper is structured as follows: Chapter 2 reviews the existing literature on cloud rendering, focusing on both the technical approaches and the challenges identified in prior research. Chapter 3 details the design of our solution and the equations used to achieve realistic results. Chapter 4 explains our strategies for implementing the design outlined in Chapter 3. Chapter 5 presents the outcomes of our work, showcasing the level of realism we were able to achieve and analyzing our system's performance in various scenarios. Finally, Chapter 6 concludes the paper, summarizing our contributions to the field of cloud rendering and outlining potential avenues for future investigation.

Chapter 2. RELATED WORK

In this chapter, we review the existing literature related to cloud rendering. We begin by briefly providing some background knowledge on cloud formation. Following this, we introduce techniques for cloud modeling and rendering that are applicable to real-time applications.

2.1 Background on Clouds

Having some background knowledge on clouds is useful to understand why certain decisions are made in the cloud rendering process. The following information is sourced from the book "The Clouds" by Roger Clausse and Lèopold Facy [6].

The cloud formation process begins when water on the Earth's surface is warmed by the sun and turned into water vapor. The water vapor then rises upwards until it reaches a cool pocket of air in the atmosphere. At this point, the water vapor condenses onto dust particles in the air, forming a cluster of small water droplets. As this process continues, the clusters grow into clouds. Depending on the atmospheric conditions, different types of clouds can form. These cloud types vary in appearance and form at different altitude ranges within the atmosphere, as can be seen in Figure 2.1. Clouds can be grouped into two major categories: low-altitude and high-altitude clouds. Low-altitude clouds, such as Stratus and Cumulus clouds, form in slightly warmer pockets of air, consist of water droplets, and have a billowy, rounded appearance. High-altitude clouds, such as Cirrus and Cirrostratus clouds, form in colder pockets of air, consist mostly of ice crystals, and have a flat, sparse appearance.

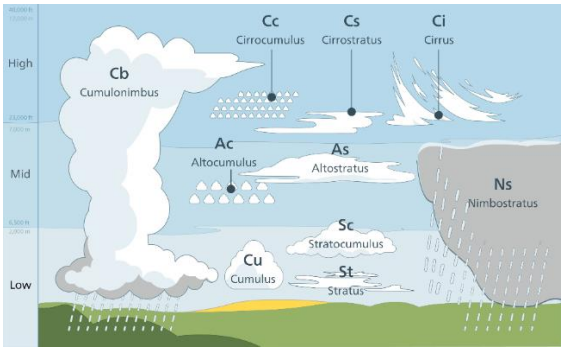


Figure 2.1: Illustration of various cloud types and altitude range in which they form [13].

2.2 Cloud Modeling Methods

Cloud rendering in computer graphics is a well-researched field with many different methodologies and approaches. A survey by Zumari & Sumar in 2022 identified 29 unique categories of atmospheric cloud modeling methods [14]. Below, we introduce several common methods for rendering atmospheric clouds that are most relevant to real-time rendering, each with varying levels of realism and performance.

2.2.1 Particle Systems

A particle system is a technique used to render objects with irregular, non-rigid surfaces such as fire, smoke, or clouds. First introduced by Reeves [15], particle systems differ from traditional mesh-based objects in three main ways. First, rather than representing objects with primitives that define a surface, particle systems use a large collection of much smaller independent objects, called particles, to define a volume. Second, unlike static shapes, particle systems produce dynamic forms, with individual particles changing position within the volume over time. Third, the shapes produced by particle systems are non-deterministic, governed by a predefined set of rules that determine how particles evolve over time.

Hädrich et al. describes a method of implementing a particle system to simulate and render the formation of clouds in the atmosphere [1]. In this method, particles represent parcels of air containing water molecules, and the system's rules are designed to simulate atmospheric conditions. As these parcels of air move through the simulated atmosphere, they converge to form clouds. Particle system-based methods for cloud rendering have the potential for very high levels of realism since the rules are based on the physics of how clouds form in the real world. However, the downside of this method is the large amount of computational resources required to achieve realistic results. Representing a large-scale environment with many clouds filling the sky requires simulating a significant number of particles. Moreover, simulating the interactions of air parcels within the atmosphere involves complex algorithms. While this approach may be suitable for niche use cases, the combination of these factors makes it too expensive for most real-time applications.

2.2.2 Procedural Generation

Procedural generation refers to methods of data generation that occur algorithmically rather than manually. In the context of clouds, this typically involves using procedural noise functions, such as Perlin noise, to approximate cloud shapes. The goal of these methods is to forgo simulating the complicated physics of cloud formation and instead approximate cloud formation with a very inexpensive noise function. This method is much more applicable to real-time applications due to its efficiency. However, if a simple noise function is used, the resulting cloud formations may not look realistic.

Schneider proposes a procedural method capable of producing realistic-looking results [4]. His modeling method uses several detailed procedural noise functions, along with additional supporting texture maps, to create realistic appearing clouds. While not based on physical processes, the results can still appear very convincing. We adopt Schneider's modeling approach in our solution due to its balance of computational efficiency for real-time applications and its realistic appearance. Further details on the design of this method are presented in Section 3.1.

Figure 2.2: A rendering of clouds using Schneider's modeling approach [4].

2.3 Cloud Rendering Methods

2.3.1 Imposter-based Rendering

Imposter-based methods provide an efficient way to render complicated objects [16]. The term "imposter" refers to a 3D object that is pre-rendered as an image. Instead of placing the actual 3D object in the scene, it is replaced with a semi-transparent plane overlaid with the image of the object. This plane is then rotated to always face the camera, creating the illusion that the 3D object is at that position. This technique significantly reduces computational load, as it eliminates the need to re-render the complicated object each frame, requiring only a simple plane with a texture lookup, which is trivially inexpensive.

Harris describes a method of dynamically generating imposters of clouds for use in real-time applications [17]. In this method, particle-based clouds are rendered once from a single viewpoint and then replaced by imposters, as can be seen in Figure 2.3. When the rotation of the imposter based on the current viewpoint exceeds a certain threshold, the imposter is regenerated from the new viewpoint. This method is extremely efficient for clouds that are very distant from the viewer, as little rotation is needed for the imposter to face the viewer. However, this method loses effectiveness as the distance decreases and the viewpoint changes more frequently. Additionally, it does not allow for dynamic clouds that change over time.

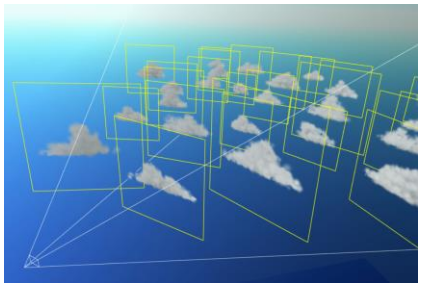


Figure 2.3: Visualization provided by Harris of their imposter-based cloud system [17].

2.3.2 Volumetric Rendering

Volumetric rendering is a technique used to visualize the propagation of light through a participating medium first introduced by Kajiya & Von Herzen [18]. Participating media refer to

volumes that contain particles. Unlike particle systems, which simulate the dynamics of each individual particle, participating media represent the volume of particles as a whole. Additionally, volumetric rendering differs from traditional surface rendering; while surface rendering simulates light reflecting off a surface, volumetric rendering simulates light traveling through a volume.

The goal of this technique is to simulate the transmission of light through the participating media into the camera. Based on the properties of the participating media, light will behave differently as it travels through the volume. A participating medium has two properties that determine how light will interact within it:

- σ_a – absorption coefficient
- σ_s– scattering coefficient

Depending on the value of these properties, light will interact differently with the participating media. Light interactions, known as scattering events, can decrease or increase the amount of light along the ray, as can be seen in Figure 2.4. The scattering events that decrease light energy are:

- Absorption (function of σ_a): Models particles absorbing photons from the light ray.
- Out-scattering (function of σ_s): Models photons colliding with particles and diverting from their original direction.

The scattering events that increase light energy are:

- In-scattering (function of σ_s): Models photons from different light rays bouncing off particles into the direction of the light ray being measured.
- Emission: Involves high-energy particles emitting additional photons into the light ray; however, for the purposes of cloud rendering, emission is typically ignored.

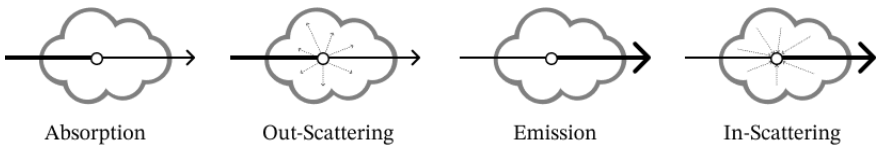


Figure 2.4: Visualization of the different scattering events in volumetric rendering.

Fong et al. provides a framework for volumetric rendering with participating media such as fog, smoke, and clouds [9]. This framework effectively models the complex interactions of light within cloud volumes. We chose this method because of its basis in theoretical foundation that offers flexibility in integrating additional advanced phenomenon resulting in realistic cloud illumination. The specific techniques used in this framework are detailed in Section 3.2.

Chapter 3. SOLUTION DESIGN

This chapter outlines the design of our solution for realistic cloud simulation and rendering. We first describe our cloud modeling approach, based on the method proposed by Schneider [4], which determines the formation and distribution of clouds in the atmosphere. We then describe our cloud rendering method, which follows the general volumetric rendering framework as described by Fong et al [9]. To increase realism, we incorporate several improvements specific to cloud rendering based on others' research, which will be detailed in section 3.2. The focus of this chapter is on the design aspects of our solution, with implementation details to be discussed in the next chapter.

It should be noted that our approach of modeling clouds, described in Section 3.1, is based entirely on Schneider's proposal of representing the 3D shape of clouds as decompressed 2D data, paired with procedural noise functions [4]. The focus and innovation of Schneider's solution design lie in this modeling approach, while the details of their volumetric rendering process are briefly discussed comparatively. As such, our volumetric approach, discussed in Section 3.2, is based on the in-depth framework provided by Fong et al. [9], and extended with cloud-specific improvements.

3.1 Cloud Modeling

Cloud modeling refers to the process of determining the physical form of clouds within the atmosphere. Our approach follows the framework proposed by Schneider and specifies the locations where clouds will form, the types of clouds that will appear, the shapes the clouds will take, and the detailed features that give them their characteristic appearance [4]. In this method, given a 3D position in the atmosphere, we algorithmically determine the density a cloud will have at that position. By applying this algorithm to every position within the atmosphere, we create a model that maps atmospheric position to cloud density. The following subsections describe the algorithms used to create the model.

3.1.1 Dimensional Profile

The algorithm begins by first defining where in the atmosphere a cloud is capable of having density. This density is approximated with the dimensional profile: a value ranging from 0 (no cloud) to 1 (complete coverage), representing the percentage of a cloud’s density that can be present at a given 3D position. This value is a function of two other parameters: cloud coverage and vertical profile.

Cloud coverage is defined as the potential presence of clouds based on the longitude (x-axis) and latitude (z-axis) in the world. This allows us to determine where clouds can exist on a horizontal plane, without considering altitude. This value is derived from a pre-computed 2D cloud

coverage texture map. One can consider this texture map as a bird's eye-view of the atmosphere. Each location on the map corresponds to a specific position in the world, and the value stored at each map location represents the cloud coverage at that position. Figure 3.1 is an example cloud coverage texture map. It should be noted that the cloud coverage process is only dependent on the x and z components of the position. The y component, or altitude, is used in a separate process, which will be discussed next.

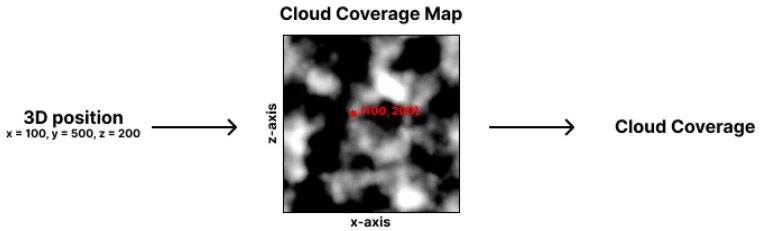


Figure 3.1: Visualization of process from 3D position to Cloud Coverage value. Based on the x,z coordinate of the 3D position, the Cloud Coverage Map is sampled at that position. The sampled value is the cloud coverage value. Details on how the cloud coverage map is generated are presented in section 4.2.1.

The vertical profile is designed to model the variations in cloud density at different altitudes. Different types of clouds form at varying altitude ranges within the atmosphere. As illustrated in Figure 2.1, Stratus clouds form low in the atmosphere and typically form within a narrow altitude range, while Cumulus clouds also form low in the atmosphere but cover a broader altitude range. The vertical profile captures these variations by providing potential density based on the type of cloud forming and the current altitude at any given position. The type of cloud is approximated by a value ranging from 0 (low-altitude clouds like Stratus) to 1 (high-altitude clouds like Cumulus), where values in between represent a blend of these types. The process of determining the value for cloud type is similar to that of cloud coverage—through a pre-defined 2D texture map, as illustrated in the top left texture in Figure 3.2. The resulting cloud type value, together with the current altitude, is then used as input to look up the value of the vertical profile, as illustrated in the bottom right texture of Figure 3.2.

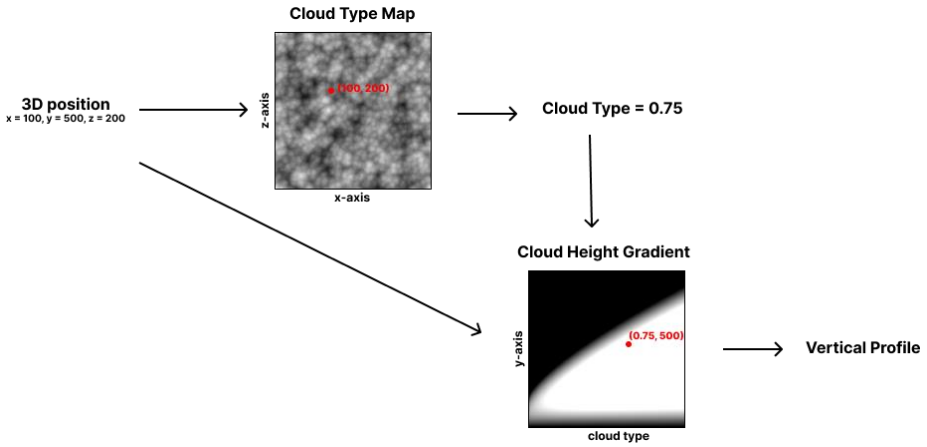


Figure 3.2: Visualization of process from 3D position to vertical profile value. Based on the x,z coordinates of the position, a type value is sample from the Cloud Type Map. That type value is then used in sampling the Cloud Height Gradient along with the y position of the 3D position. The resulting sampled value is the vertical profile.

The dimensional profile is then calculated with the following equation:

$$d_p = c_{cov} c_{vert} \tag{3.1}$$

Where, based on an 3D input position (x, y, z):

- c_{cov} the cloud coverage at (x, z)
- c_{vert} is the vertical profile of the cloud at (x, z) with height of y, and
- d_p is the resulting dimensional profile value at (x, y, z)

The dimensional profile combines both the horizontal potential (cloud coverage) and the vertical potential (vertical profile) to represent the overall potential cloud density at a given 3D position. It is important to note that d_p does not represent the actual cloud density value. Instead, it is an intermediate value used in calculating the final cloud density, as described in the next subsection.

3.1.2 Cloud Density

In the atmosphere, the density of a cloud varies across the space it occupies. This variation leads to complex shapes and features on the cloud surfaces. We follow Schneider and model these surface details with a 3D procedural noise function designed to capture the natural appearance of clouds. With a given position (x, y, z), the cloud density, ρ, is approximated as value:

$$\rho = \left(N - \left(1 - d_p\right)\right) \rho_c \tag{3.2}$$

Where:

- N: the 3D noise value at (x, y, z)
- ρ_c: a global user defined variable, that allows the overall resulting density value to be modulated to account for varying levels of cloud densities

Note that, the values of ρ, N, and d_p, are all functions of the 3D position upon which the values are computed.

By subtracting the inverse of the dimensional profile from the noise value, the areas in which the dimensional profile was found to be low are lost, while areas where the dimensional profile are higher begin to take the shape of the procedural noise. Schneider describes this process as eroding away the dimensional profile to reveal the detailed cloud shape within, as illustrated in Figure 3.3.

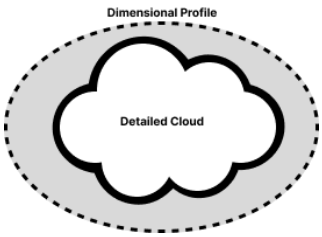


Figure 3.3: Visualization of the detailed cloud shape that is sculpted out of the dimensional profile. The grey oval represents areas where the dimensional profile is non-zero. By performing noise erosion, we are left with the detailed cloud shape within the dimensional profile.

With the density of the clouds determined, the next step is to translate these densities into visually accurate representations. This involves simulating the interaction of light with the cloud particles to generate realistic images. The following section will delve into our cloud rendering design, detailing the methods and techniques used to create the final visual output based on our cloud density model.

3.2 Cloud Rendering

In this section, we detail the solution for rendering an image based on the approximated cloud density. The rendering solution is similar to how a physical camera captures an image: by approximating visible energies as rays from the atmosphere intersecting with the image plane. To simulate this process, we first determine the location that a ray from the light source, or the light ray, arrives and intersects with our image plane, and second, approximate the visible energy arriving along that ray.

To determine the location where a light ray intersects our camera's image plane, we need the ray’s origin and direction. Typically, in an outdoor environment, light rays originate from the sun and radiate outward in all directions. Simulating the inter-reflection of these rays to determine which subset will arrive at the camera’s image plane is inefficient, as the majority of the light rays emitted by the sun will never intersect with our image plane. Instead, we calculate rays originating from the camera, or camera rays, and direct these rays toward specific positions in the image. This method is justified by the Helmholtz reciprocity principle [19], which confirms that a light ray and

its reverse traverse identical paths. Thus, by modeling rays from the camera to the image, we ensure accurate lighting results while avoiding unnecessary computations.

Radiance is a measurement of light energy traveling along a ray. The radiance value at the point where a ray intersects with the image plane determines the color at that specific position. As light travels through the participating media, it interacts with cloud particles within the media resulting in alterations to the radiance value. To calculate the resulting radiance value, we follow the volumetric rendering equation from Fong et al. [9]:

$$L(c, -v) = T_r(c, p)L_o(p, v) + \int_{t=0}^{||p-c||} T_r(c, c - vt)L_{scat}(c - vt, v)\sigma_s dt \tag{3.3}$$

The integral term calculates light scattering along the camera ray while traveling through the volume, and the final term represents the radiance that passes through the entire volume interval arriving at camera:

- c: position of the camera, or origin of a camera ray
- v: inverse direction of the camera ray, or direction from p to the camera
- p: the final visible point along the camera ray
- $T_r(c, c - vt)$: the total accumulated transmittance between positions c and $c - vt$. Note that in the absence of clouds, with no cloud density this term would have a value of one.
- $L_{scat}(c - vt, v)$: the scattered radiance at position $c - vt$ along v. Note that in the absence of cloud, with no media participating in the scattering, this term would have a value of zero
- σ_s: scattering coefficient of cloud media. Refer to Section 2.3.2 for description of this parameter.
- $L(c, -v)$: the total radiance arriving at c along v.
- $T_r(c, p)$: the total transmittance from c to p. This accounts for how much of the original visible light energy that is capable of traveling through the cloud medium.
- $L_o(p, v)$: the radiance at position p along the ray direction v.

The integral captures energy accumulated along the path of the eye ray. This process is illustrated in Figure 3.4.

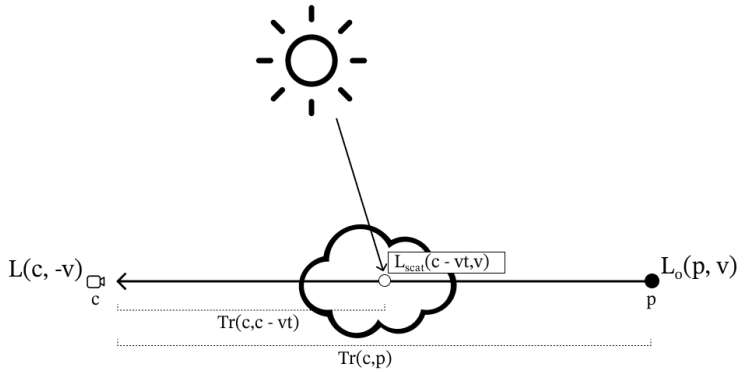


Figure 3.4: An illustration depicting the radiance equation. A ray begins at point p and travels to the camera at point c. $L(p, v)$ is the original radiance value at that point, and $L(c, -v)$ is the radiance value that reaches the camera. $Tr(c, p)$ is the amount of original light from point p that was able to make it to the camera. $L_{scat}(c - vt, v)$ represents the amount of light reaching a single point in our integral. $Tr(c, c - vt)$ represents the how much of that scattered light is capable of continuing out of the cloud and into the camera.

The following sub-sections detail the approximation for the transmittance term (T_r) the scattering term (L_{scat}), as well as our adaptation of the phase function [12] which refines the L_{scat} term with approximated directional scattering probability of light within the medium.

3.2.1 Transmittance

Transmittance represents the ratio of light that manages to pass through a medium, with values ranging from 0 (no light passed through) to 1 (light ray fully traversed the participating media). We approximate the transmittance of light through our participating media based on the Beer-Lambert Law which states that the greater the optical depth, the lower the transmittance. This relationship is captured by the equation provided by Fong et al.[9], where τ denotes the optical depth between the points x_a and x_b:

$$T_r(x_a, x_b) = e^{-\tau} \tag{3.4}$$

Optical depth is the measurement of how much light is attenuated as it travels through a medium, taking into account factors such as the medium's density, extinction coefficient, and the distance of traveled through the medium. The equation for calculating optical depth, as described by Fong et al. [9], where ρ is the density of the cloud as defined in Equation 3.2 and σ_t is the medium's extinction coefficient, is:

$$\tau = \int_{x=x_a}^{x_b} \sigma_t dx \tag{3.5}$$

It can be seen that as the distance of the interval increases, and as the density of the medium increases, the resulting optical depth will increase and in turn the transmittance will decrease. The medium's extinction coefficient can then be seen to be a modifier to the Beer-Lambert equation, whereby a having a lower extinction coefficient, the medium can allow more light to pass through.

3.2.2 Light Scattering

In the real-world, some portion of the rays from the sun will always scatter in the direction of our camera rays. This is known as in-scattering. Following the model proposed by Fong et al. [9], we measure in-scattering at a position x from a direction v with the following equation:

$$L_{scat}(x, v) = p(v, l_{sun})T_r(x, p_{sun}) \tag{3.7}$$

Where $p(v, l_{sun})$ is the phase function based on the light direction from the sun l_{sun} and $T_r(c, p_{sun})$ is the transmittance value between the position x and the position of the sun p_{sun}. In essence, we are measuring the energy from the sun reaching a specific point within the medium. The phase function models the probability that the light from the sun will scatter in the v direction, or the direction of our camera ray. The details of phase function is described in the next sub-section.

The resulting scattering value from Equation 3.7 only accounts for light energy arriving at position x directly from the sun. As this is only accounting for a single ray, this is known as single scattering. This does not capture general in-scattering. As light travels through clouds, there are many chances for light rays from other directions to in-scatter and resulting in them aligning with the direction of camera ray. This is known as multiple scattering. A visualization of multiple scattering versus single scattering can be seen in Figure 3.5. Simulating multiple scattering is computationally expensive, as it requires calculating additional scattering integrals for each light bounce.

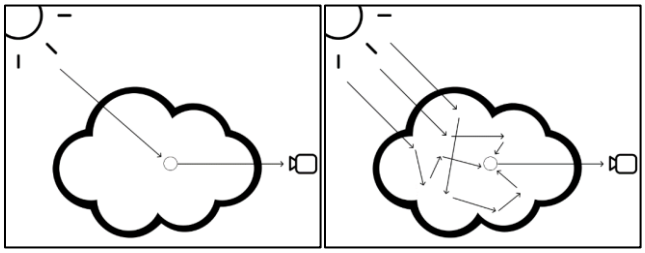


Figure 3.5: Visualization of single scattering (left) compared to multiple scattering (right). Single scattering only accounts for the scattering of the ray directly from the sun. Multiple scattering accounts for additional rays that may have in-scattered into the camera ray.

For efficient approximation of multiple scattering, we follow Wrenninge et al.'s approach [11] where the total scattering is computed as a sum of *N* different scattered light rays:

$$L_{mult}(x, v) = \sum_{i=0}^{N-1} L_i(x, v) \tag{3.8}$$

Each subsequent scattered light ray has its extinction coefficient artificially increased to simulate the decrease in energy as light continues to scatter throughout the cloud:

$$L_i(x, v) = b^i p(v, l)\, e^{-a^i \tau} \tag{3.9}$$

Where:

- b: controls the overall contribution of this octave of light scattering
- a: controls the attenuation of light scattering at each octave of light
- $e^{-a^i \int_0^t \sigma_t(s)ds}$: a modified version of the transmittance equation (Equation 3.4), where the optical depth value is modified by a.

In essence, this equation is recomputing the scattering equation N times, with each subsequent calculation decreasing in its contribution to the sum. This simulates multiple rays of varying intensity in-scattering.

3.2.3 Phase Function

A phase function describes the direction light scatters when it interacts with particles within a medium. The scattering direction depends on the characteristics of the particles and the angle of the incoming light ray. Variations in particle size, shape, and refractive index result in different angular distributions of scattered light. The phase function specifies this distribution. It takes the angle between the incoming light direction and the viewing direction and returns a value between 0 and 1, indicating the ratio of scattered light that will travel in the viewing direction. This process is illustrated by Figure 3.6.

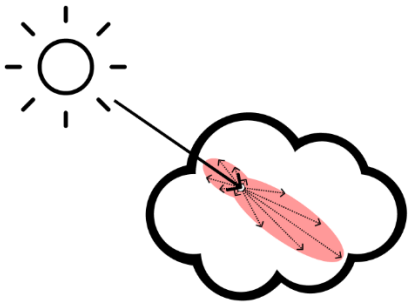


Figure 3.6: A visualization of a phase function. The area in red is representative of the angular distribution of light when scattered. In this example, a greater proportion of light is scattering forward.

The phase function for clouds is determined by the microscopic water droplets that compose clouds. These droplets are comparable in size to the wavelength of light, making Mie scattering applicable for accurately modeling a cloud phase function [20]. However, Mie scattering is complex and computationally expensive, where solutions often involve extensive lookup tables [12].

The approximation phase function we use is based on the work of Jendersie & d'Eon [12], who propose combining the Henyey-Greenstein(HG) and Draine phase functions. The HG phase function is a single-parameter model:

$$p_{hg}(\theta, g) = \frac{1 - g^2}{4\pi(1 + g^2 - 2g\ cos(\theta))^{1.5}} \tag{3.11}$$

Where θ is the angle between the camera ray and the light direction and g is the asymmetry parameter, ranging from -1 to 1. When $g = 0$, light is scattered equally in all directions, when g = 1, it is forward scattering dominant, and when $g = -1$, it is backward scattering dominant. The Draine phase function is a two-parameter model:

$$p_d(\theta, \alpha, g) = \frac{1 - g^2}{4\pi(1 + g^2 - 2g\,cos(\theta))^{1.5}} \frac{1 + \alpha \cos^2(\theta)}{1 + \frac{\alpha(1 + 2g^2)}{3}} \tag{3.12}$$

In which the first term is the HG phase, and the second term is the Generalized Rayleigh phase function. The α term is also used as an asymmetry parameter to control the directionality of light scattering. The phase functions are combined with a weight parameter w:

$$p(\theta, \alpha, g_{hg}, g_d) = (1 - w)p_{hg}(\theta, g_{hg}) + wp_d(\theta, \alpha, g_d) \tag{3.13}$$

Jendersie & d'Eon compared the results of this phase function with several other common phase functions. They found it to match most closely with the complex Mie scattering, as seen in Figure 3.7. We use this phase function due to its high level of accuracy.

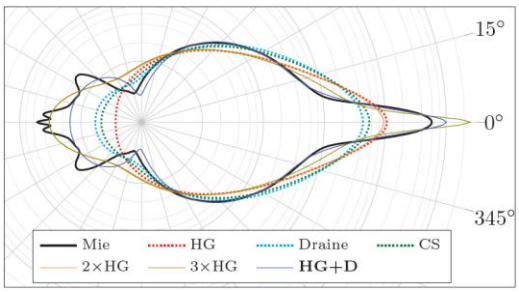


Figure 3.7: A visualization of the angular distributions of different phase functions as presented by Jendersie & d'Eon[12]. Mie scattering is represented as the solid black line. They compared this with many other phase functions and found HG + D to match Mie scattering most closely.

Jendersie & d'Eon derived equations for each of the parameters, assuming d is the diameter in microns of the particles within a medium:

$$g_{hg}(d) = e^{-\frac{0.0990567}{d - 1.67154}} \tag{3.14}$$

$$g_d(d) = e^{-\frac{2.20679}{d + 3.91029} - 0.428934} \tag{3.15}$$

$$\alpha(d) = e^{3.62489 - \frac{8.29288}{d + 5.52825}} \tag{3.16}$$

$$w(d) = e^{-\frac{0.599085}{d - 0.641583} - 0.664888} \tag{3.17}$$

The diameter of cloud particles can range anywhere from 5 – 50 microns, typically with values closer to 10 microns [21]. As such, are the values used in our implementation:

- g_{hg} = 0.9881
- g_d = 0.5567
- α = 21.9955
- w = 0.4824

3.3 Summary

This chapter outlines our solution design for realistic cloud modeling and rendering. While our cloud modeling method is a relatively straightforward adaptation of Schneider's work, our cloud rendering solution is an enhancement to the volumetric rendering framework as proposed by Fong [9]. More specifically:

- We follow Wrenninge's approach for approximating light scattering to improve the radiance calculation by accounting for multiple scattering [11].
- We adopted Jendersie & d'Eon's phase function to account for the angular distribution of light scattering based on the particle size of water droplets [12].

The next chapter discusses the implementation of this design, where we provide additional contributions to optimize performance and achieve real-time rendering.

Chapter 4. IMPLEMENTATION

This chapter discusses how we implement our design as described in Chapter 3. We begin by outlining the necessary system requirements necessary for our cloud rendering system. Next, we detail our implementation of cloud formation, explaining the process for generating the textures and noise used in cloud modeling. We then discuss our ray marching algorithm used to render our clouds. We also describe the anti-aliasing techniques used to improve the visual quality of our renders and the methods we use to reduce artifacts and noise. Finally, we present optimization strategies that help us achieve real-time rendering rates.

4.1 System Requirements

The support required for our cloud rendering system include abilities to:

- execute shading operations at the pixel level,
- access multiple 2D and 3D textures during shading operations.
- save the results of per-pixel shading to a texture, and
- perform rendering operations for each frame in real-time.

We have chosen the Unity game engine [22] as our rendering engine. Unity is a free-to-use game engine primarily designed for game development. This system meets all our system requirements. Its rendering pipeline is designed for real-time image generation, with flexible support for 2D and 3D textures storage and access during shading operations. We perform per-pixel operations using compute shaders and blit the computation results as an image to the display. Unity provides simple compute shader dispatch and streamlines the process of sending data to the GPU. Additionally, it offers GUI components for real-time parameter tweaking. Lastly, we have ample previous experience with the system. It should be noted that our implementation is not Unity-specific and can be adapted to any engine that meets the system requirements.

Unity provides the computer shader to realize the parallelism of the underlying GPU. In our implementation the functionality of our per-pixel operations is implemented as a compute shader. One Unity invocation of our compute shader computes the transmittance and radiance values for all the eye-rays through every pixel of an image. The results of the compute shader are sent to the post processing fragment shader and converted into RGB values to be displayed to the screen.

4.2 Cloud Formation

In this section we will describe the implementation strategy taken to solve section 3.1 of our solution design: Cloud Modeling. We first discuss how we generate the cloud map textures used in the computation of the dimensional profile. We will then discuss our process of generating procedural noise and how it is used to add detail to our clouds.

4.2.1 Cloud Maps

The cloud maps of Figures 4.1, 4.2, & 4.3 are implemented as 1024 x 1024 textures. Each value in the texture corresponds to a single value ranging from 0 to 1. We have three lookup textures: the cloud coverage map, the cloud type map, and the cloud height gradient. The sampled values from these textures correspond to the parameters in Equation 3.1. As illustrated in Figure 3.1 & 3.2, the textures are sampled based on the world coordinate of the positions.

Schneider's work presents the framework for cloud modeling, however, details on the exact methodology used for generating cloud maps are missing. In our implementation, we approximated the maps based on observations of real clouds. For example, one observation is that the base of clouds tends to be circular in shape. Figure 4.1 shows our process of recreating this aspect of cloud formation in our cloud coverage map. To create these shapes, we invert the Worley noise [23], producing randomly placed circles with edges that slowly fade. We then add another higher frequency level of noise to increase detail. Perlin noise is generated, and we map the high areas of the Perlin noise to the high areas of the Worley noise to add intricate details to the shapes. Both our Perlin and Worley noise generation functions are tileable, as our final cloud map will be tiled across the atmosphere.

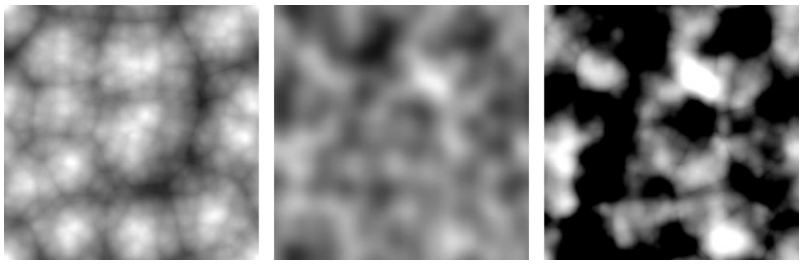


Figure 4.1: Cloud map generation process. Multiple frequencies of inverted Worley noise (left), Perlin noise (center), Perlin noise mapped to Worley noise (right).

Another observation we made is that clouds forming in a given area tend to be of a similar type. Because of this, rather than determining the cloud type entirely via a texture, we use a global variable to represent the type of all clouds in the scene. The value we sample from the cloud type map modulates this global cloud type variable, adding additional details to the vertical profile of our clouds. To recreate the round bulges that form at the top of clouds, our cloud type map is composed of the product of several frequencies of inverted Worley noise, as seen in Figure 4.2.

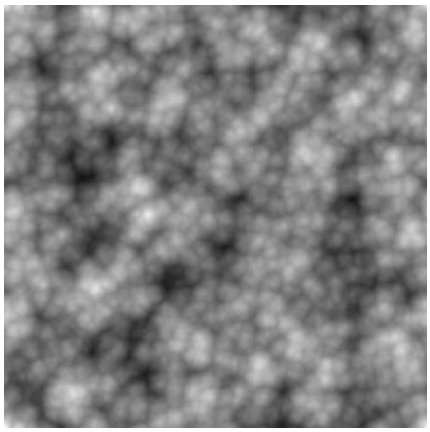


Figure 4.2: An example of a cloud height map generated with multiple levels of Worley noise.

We base our approach for generating our cloud height gradient on Schneider's method for generating height gradients for single cloud types [4]. In this method, values for the top and bottom of the altitude describe the range of altitudes a cloud can form in. The values within this are blended over height to smoothly transition from 0 to 1. Examples of single height gradients for Stratus clouds and Cumulus clouds can be seen in the left and middle of Figure 4.3.

We extend this approach by interpolating our gradients across all possible cloud types. We begin by defining a range for our lowest altitude cloud type: Stratus, and our highest altitude cloud type, Cumulus. We then interpolate these values using a square root function. An example of our height gradient can be seen in the right of Figure 4.3.

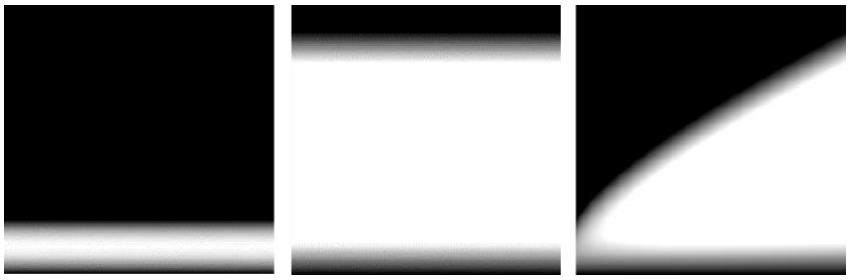


Figure 4.3: Stratus height gradient (left), Cumulus height gradient (right), interpolated height gradient (right).

4.2.2 Cloud Noise

The next step is our implementation of Section 3.1.2, determining cloud density. Density is calculated by eroding the outer edges of the cloud based on a procedural noise value and the current value of the dimensional profile, as seen in Equation 3.2. This process sculpts the detailed cloud shape within the dimensional profile. To achieve a shape that appears cloud-like, our procedural noise function generates values that resemble the details observed in clouds. We adopt Schneider's categorization of cloud details: wispy details where cloud density dissipates and billowy details where cloud density increases, pushing water vapor outward and upward. To achieve procedural

noise that recreates these details, we follow Schneiders approach of using a technique called Fractional Brownian Motion (FBM) [4].

FMB is a mathematical process that generates fractal-like structures through the summation of several octaves of self-similar frequencies [24]. In our detailed noise generation, this is accomplished by sampling and accumulating procedural noise functions with increasing frequency and decreasing amplitude. When applied to Perlin noise, the result resembles wispy details, while using inverted Worley noise produces billowy effects, as can be seen in Figure 4.4.

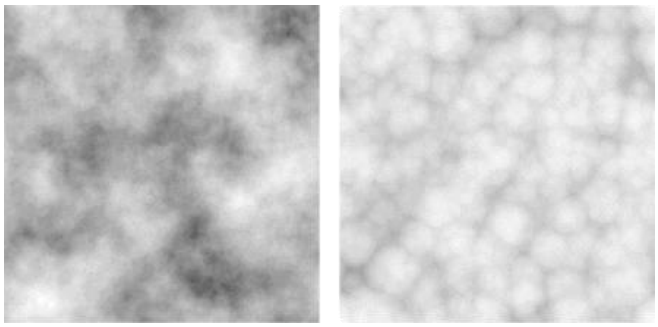


Figure 4.4: Example of Perlin Noise (left), and Worley noise (right)

Noise values are pre-computed and stored in a 256 x 256 x 256 3D texture map. In Schneider's approach, Perlin and Worley noise are combined into a single "Perlin-Worley" noise texture. Our approach differs in that we store each noise type separately. This increases the amount of memory required but allows the modeling of typical variations along the altitude when computing cloud density. Additionally, as each noise value is just a single floating point number, we can store different frequencies of the noise in each RGBA channel of the texture, as seen in Figure 4.5.

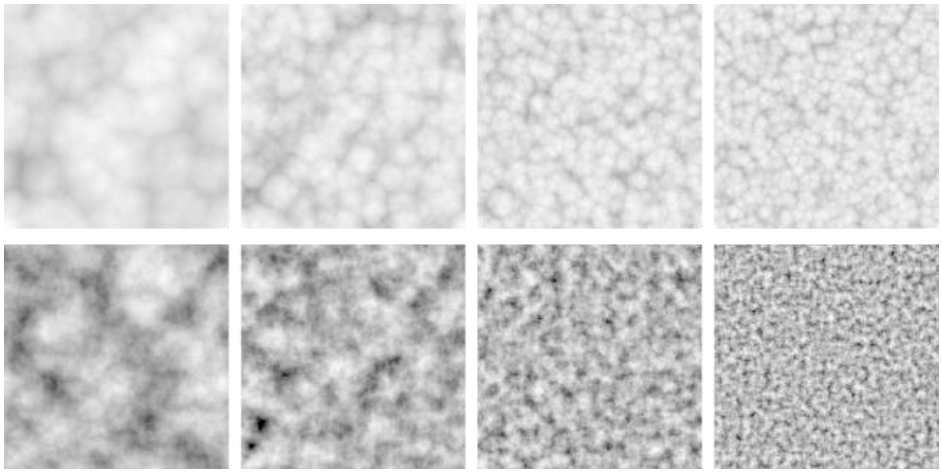


Figure 4.5: Varying frequencies of FBM Worley noise (top row), and of FBM Perlin noise (bottom row).

The noise value is sampled in a similar manner to the cloud coverage map, where the texture coordinate is based on the world coordinate.

Both Perlin and Worley noise are sampled per position. The final noise value is an interpolation of the two sampled noise values based on the height of the position. Positions at lower altitude are characterized by Perlin noise, while higher altitude will be Worley. This gives the bottoms of clouds wispy details and the tops billowy details.

4.3 Ray Marching Algorithm

The Equation 3.3 introduced in Section 3.2 computes the intensity of light along a ray from the camera (or eye) to determine the color of each individual pixel in our image. This equation integrates transmittance and light scattering values over a volume of cloud density. Since cloud density varies spatially and can take complex shapes, this integration must be approximated. The technique we use to approximate this integral is ray marching [25].

Ray marching is a technique commonly used in volumetric rendering where a ray is incrementally advanced through a scene. For each pixel in our image, we generate a ray whose origin is at the camera position and whose direction intersects the image plane within the area of the desired pixel. We then travel along this ray in discrete intervals called steps. Within each step, we determine a position to approximate the terms in the integral. By summing the contributions of each sample along the ray, multiplied by the step size, we can approximate integrals over the ray path.

The bounds of our ray march are defined by the atmospheric volume in which clouds form. As illustrated in Figure 4.6, this volume is defined by two concentric spheres bounding the lower and upper altitude of the atmosphere where clouds are formed. The starting position of our ray march is where the ray intersects the lower atmospheric shell and ends at the outer atmospheric shell. This approach ensures we only take samples in regions where clouds are present, preventing unnecessary calculations.

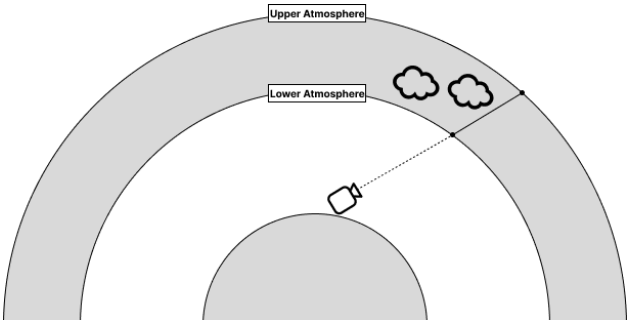


Figure 4.6: Visualization of our camera ray first intersecting with the lower atmosphere sphere and then intersecting with the upper atmosphere. The grey shell between the two spheres represents the atmosphere volume where clouds can form.

Equation 3.3 integrates light transmittance and light scattering over some distance. Both the transmittance and scattering equations depend on cloud density. Figure 4.7 shows that at each

sample along our ray march, we begin by calculating the cloud density at the given position using Equation 3.2. If the density value is zero, this sample will not contribute to our integration, allowing us to skip further calculations for this position. If the density value is non-zero, the sample will contribute to our integral, and transmission and scattering calculations of Equation 3.3 must be performed. When a camera ray exists the cloud density, the distance travelled from the camera is used as a parameter for blending with the background [26]. This final blending step allows distant cloud illumination to merge gradually with the background.

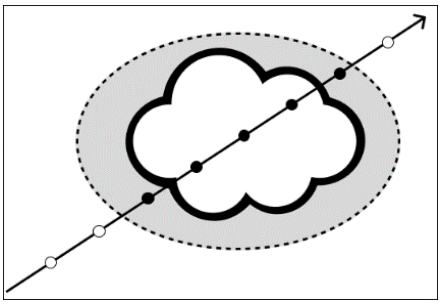


Figure 4.7: Illustration of ray march skipping samples where dimensional profile is zero (white dot) and performing calculations where dimensional profile is non-zero (black dot).

Transmittance, T_r in Equation 3.3, is a function of extinction, which is calculated using our sampled density as described in Equation 3.4. The transmittance is then modulated by the Beer-Lambert equation with the given step size. This contribution is then accumulated to the total transmittance up to the current step. At the end of the march, the accumulated transmittance provides our approximation of the optical depth integral for the ray.

As illustrated in Figure 4.8, to calculate scattering, L_{scat} in Equation 3.3, for a given sample position, we start by determining the transmittance between the current sample position and the light source. This requires performing another ray march, beginning at the sample position and marching towards the light source. The transmittance calculated from this light-ray march is then used in our multiple scattering approximation (Equation 3.8) to obtain a scattering value. This scattering value is multiplied by the step size and accumulated by adding it into our total scattering value for the integration.

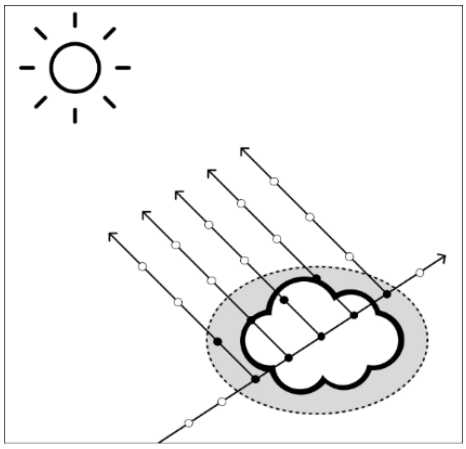


Figure 4.8: Visualization of ray march from sample position and moving towards the light source. This is done for each sample where density is found (black dot). Additionally, no calculations are needed to perform on areas where no density is found (white dot). The resulting Transmission value found for each ray will be used in that position's scattering contribution.

The integral in Equation 3.3 can be approximated with the straightforward trapezoid rule. With such an approach, the volumetric interval would be discretized into segments with the assumption that the transmittance (T_r) and scattered radiance (L_{scat}) are constants over the segment. As discussed in the following, the assumption on transmittance being a constant presents a challenge.

$$T_r(x_a, x_b) = e^{-\tau} = e^{-\int_{x_a}^{x_b} \sigma_t \rho(x) dx} \quad 4.1$$

Equation 4.1 examines the accumulated transmittance over an interval x_a to x_b in detailed by substituting Equation 3.5 into Equation 3.4. Substituting Equation 4.1 into the transmittance (T_r) in the integral of Equation 3.3 results in:

$$T_r(x, x_b) L_{scat}(x, v) \sigma_s = e^{-\int_{x_a}^{x} \sigma_t \rho(x) dx} L_{scat}(x, v) \sigma_s \quad 4.2$$

In Equation 4.2, x is a position between x_a and x_b and is the actual position where the expression is evaluated. As discussed, this equation computes the visible radiance along the camera ray (the v direction) at the position x and assumes this result is a constant along the x_a to x_b interval. Notice that the integration range of the transmittance, from x_a to the current sample position x, does not cover the entire interval from x_a to x_b. Straightforward approaches to rectify this inconsistency typically either compute the transmittance that covers the entire x_a to x_b interval or uses the accumulated transmittance from segments prior to the sampling position [25]. While capable of producing acceptable results, these approaches either over-estimate, with clouds being too dark, or under-estimate the transmittance, with clouds being too bright.

Hillaire [11] proposed an alternate approach where based on the trapezoid rule and the assumption that scattered radiance (L_{scat}) is constant over each segment, the integral of Equation 3.3 can be approximated with:

$$\int_{t=0}^{||p-c||} T_r(c, c-vt) L_{scat}(c-vt, v)\sigma_s dt \; = \sum_{i=0}^{n} \left[\int_0^d \sigma_t \, \rho(w) dw\right] L_{scat}(p_i, v)\sigma_s \qquad 4.3$$

Where:

- n: is the number of intervals along the camera-ray march
- d: is the size of each camera-ray march step
- p_i: is the position being sampled for L_{scat} computation

Hillaire went on to show that the term within summary of Equation 4.3 can be evaluated as:

$$\left[\int_0^d \sigma_t \, \rho(w) dw\right] L_{scat}(p_i, v)\sigma_s = \frac{L_{scat}(p_i, v)\sigma_s(1 - e^{-d})}{\sigma_t} \qquad 4.4$$

In the next chapter, results from both over-estimating the transmittance, and from Equation 4.3 will be presented and discussed.

4.4 Anti-Aliasing

The previous sections detail the core of our implementation: approximating the color of a pixel area with one eye-ray. Such an approximation is not capable of capturing rapid changes of color with the pixel area and thus would result in low quality image with artifacts. This overall lack of quality is referred to as aliasing, and strategies for remedy are known as anti-aliasing. This section describes the anti-aliasing techniques we employ to improve the quality of our cloud images.

The anti-aliasing strategy we adopt for both pixel and ray interval sampling involves introducing randomization into our sampling position. Instead of sampling at the center, we sample at a random position within each pixel and ray interval. This approach reduces aliasing artifacts and improves the visual quality. An example of aliasing-prone sampling compared to our anti-aliasing samples can be seen in Figure 4.9.

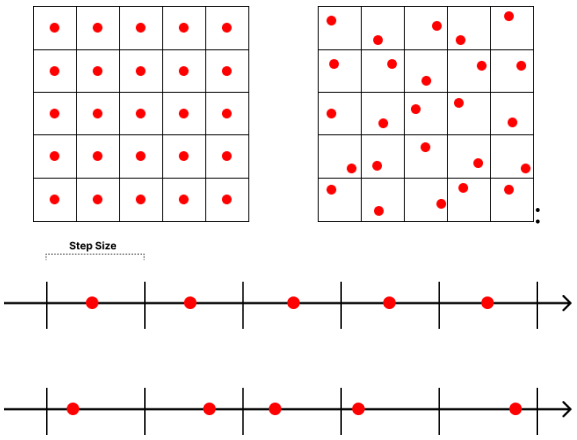


Figure 4.9: Visualization of our randomization strategy for anti-aliasing. The top row shows and example of uniform pixel sampling (left), and randomized pixel sampling (right). The bottom of the figure shows a uniformly sampled ray (top) and a randomly sampled ray (bottom).

We use blue noise to achieve randomization when determining our pixel sample position. Blue noise has the characteristic that neighboring sample positions have a high probability of being far apart resulting in higher image quality than straightforward randomization [27]. An example of blue noise compared to white noise can be seen in Figure 4.10.

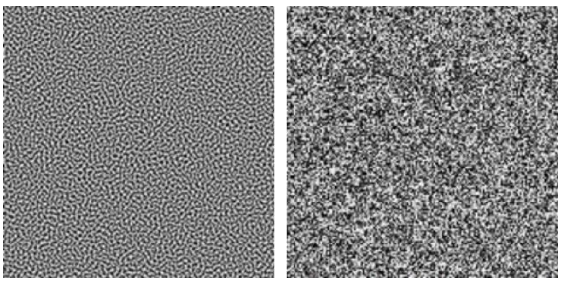


Figure 4.10: A comparison of blue noise (left) and white noise (right).

A set of three blue noise values are pre-computed and stored for each pixel in our image. The first blue two noise values are used to offset the pixel sample position in the x and y and the third is used for offsetting the ray march interval position. This can be seen in Figure 4.11.

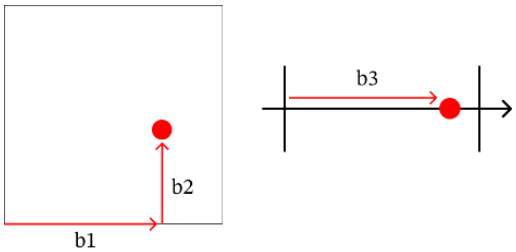


Figure 4.11: Visualization of using blue noise to add randomization. Offset within pixel is determined by the values of $b1$ and $b2$, while offset within ray march segment is $b3$.

4.5 Unified Temporal Anti-Aliasing

This section details our novel solution that unifies the sampling of 2D pixel area and the 1D volumetric depth interval based on the temporal anti-aliasing framework.

4.5.1 Sliding Window

We follow the concept behind Temporal Anti-Alias (TAA) [28] and blend the results of some number of previous frames into the current frame being displayed. We implement TAA by using a sliding window approach. We allow the user to specify the size of our window n, and store n frame buffers to be used in our blended image. As illustrated in Figure 4.12 with n=3, the window consists of the last n rendered frames. Each new frame is modulated by 1/n before storing into the window of frame buffers. Each time a frame leaves the window, its contribution is subtracted from the current frame. The benefit of this approach is that as each frame only requires a single add and subtract operation, the user specified n does not affect the run time. The downside is that n is a function of time, and a larger n, in addition to memory cost, may lead to visible motion blur when there are fast moving objects.

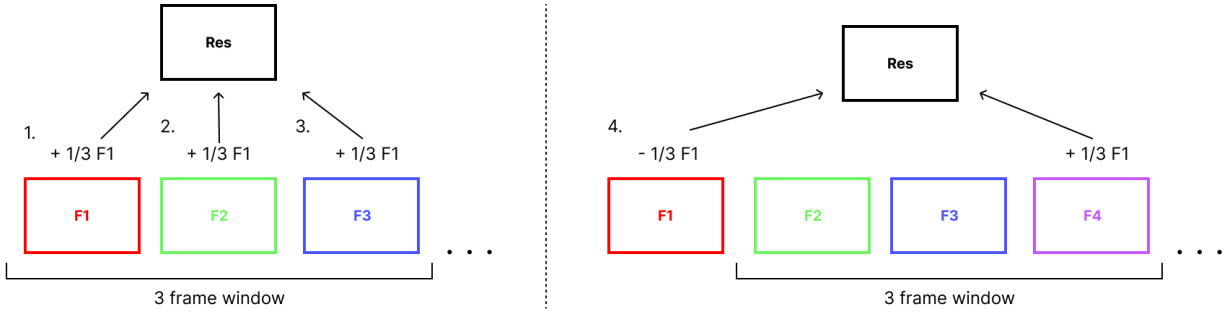


Figure 4.12: A visualization of a sliding window approach to TAA. On the left we see the first 3 frames, each getting 1/3 of their values added into the final result. On the 4th frame, the first frame exits the window and has its contribution to the final result subtracted, while the 4 frame has its contribution added.

Using our sliding window, we unify pixel area sampling and volumetric depth sampling into a unified TAA solution, as described in the following sub-sections.

4.5.2 Pixel Area Temporal Anti-Aliasing

For the 2D pixel area, the sample position of pixels for new frames are shifted modularly from the pre-computed blue noise position based on a pre-computed N-Rooks pattern [29]. A set of 1 to 16 pre-computed N-Rooks pattern supports the corresponding frame window size in temporal anti-aliasing. This is detailed in Figure 4.13 for a window size of n=4. The N-Rooks offset pattern is chosen because of its uniform coverage even when the number of samples, n, is not a perfect square [30].

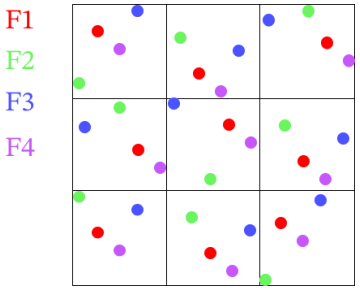


Figure 4.13: A visualization of our pixel sampling method used in conjunction with our TAA strategy. Each colored dot within a pixel represents a sample position at a given frame. The resulting values from these samples are averaged together to increase coverage of image and reduce aliasing artifacts.

4.5.3 Volumetric Temporal Anti-Aliasing

The offset strategy used in our pixel area sampling is generalized into the sampling of the volume interval based on a similar approach. As illustrated in Figure 4.14, the user defined step size segment is split into individual frame segments. A frame segment is the length of the step size divided by the number of frames in the window. Each frame samples within its corresponding frame segment. The distance at which the sample is taken within the frame segment is determined by a separate blue noise value stored in the third channel of the blue noise offset map.

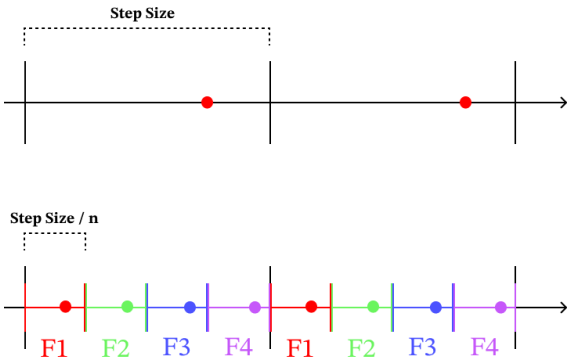


Figure 4.14: A visualization of our volumetric TAA approach. The top shows 2 ray march steps without our TAA applied, while the bottom is the same march with our TAA strategy applied. Each colored segment represents an individual frame segment. It can be seen that this effectively reduces the step size by a factor of n.

This approach unifies the sampling of the 2D pixel area with the volumetric depth interval. It effectively samples the visible volume at a higher rate without increasing the cost of computing each camera ray.

4.6 Optimizations

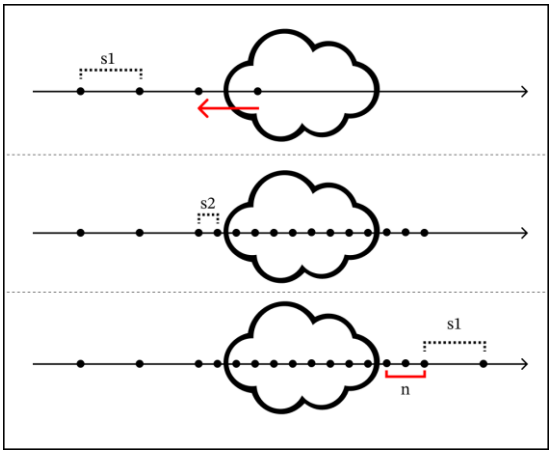
This section describes the optimization strategies implemented into our ray marching algorithm required to achieve real-time rendering rates.

4.6.1 Early Termination

One optimization strategy we use is an early exit in our ray march once we reach a certain threshold on our transmittance. As our approximation of transmittance approaches zero, the contribution of scattering decreases as can be seen in equation 3.3. At a certain point, each sample will contribute little to nothing to the total radiance. At each iteration of our ray march, we check if the transmittance value has reached this point, and if it has, we exit the loop. Additionally, if we sample our density and find that is below a certain threshold, we can skip this iteration of the loop as the resulting Transmission and Scattering computations will have low impact on our final results.

4.6.2 Adaptable Volume Step Size

Another optimization we implement is based off of an approach described by Schneider [4]. This optimization further reduces the number of required samples by performing larger steps in areas where clouds are not present. As illustrated in Figure 4.13, we define two step sizes: a large step to traverse through empty space and a small step when traversing through cloud density. We begin our march using the large step size, continuing until we find a sample position that contains cloud density. At this point, we travel backward by the large step size and switch to the small step size. While using the small step size, if we find that a consecutive number of density samples are zero, we return to the large step size. To prevent missing cloud density regions, there is a limit on how large the large step size can be. In our approach, our large step size is double the size of our small step size. This method minimizes unnecessary sampling in empty regions, improving the efficiency of our ray marching process.



4.13: Illustration of our adaptative volume sampling technique. We begin with a large step size as seen in the top row. Once a sample with cloud density is found, we move back to avoid missing any detail. We then resume our march with a small step size as seen in the middle row. When we have n number of consecutive samples without density, we switch back to our large step size.

4.6.3 Progressive Refinement

We also implement settings to dynamically change the resolution of the compute buffers used to store the radiance data. The options include full resolution, half resolution, and quarter resolution. If the calculations occur at a lower resolution, the final result is upscaled to fit the full size of the window. Lowering the resolution significantly reduces the number of rays that need to be processed per frame, thus improving performance. While this reduction can potentially lower the quality of the results due to data loss, the increased efficiency may outweigh the loss in quality depending on the situation. In our experience, halving the resolution made an imperceptible reduction in quality while providing a significant increase in efficiency, making the trade-off worthwhile.

4.7 Summary

This chapter describes the implementation of each step in our solution design. Our implementation for cloud map and cloud noise generation follows the idea presented by Schneider, though the details in the implementation are affected by our own observations of clouds. Additionally, our novel pixel and volumetric temporal anti-aliasing strategy unifies the support for super sampling across the different dimensions accomplishing a higher volumetric sampling rate without extra computation costs. This anti-aliasing strategy is further improved by adopting Hillaire's improved scattering integration, which allows for increased accuracy in radiance calculations with larger step sizes. In the next section, we will visualize the results that our implementation was able to achieve.

5. RESULTS

This chapter presents the results of our implementation. The success of our solution at both creating realistic clouds and achieving a visually pleasing render that runs in real-time will be demonstrated. The presentation visualizes the solution steps in modeling and rendering clouds, followed by results highlighting the impact of our refinements to the volumetric rendering framework proposed by Fong et al. including runtime optimizations [9]. The results presented are rendered from within the Unity editor, with a window resolution of 1920x1080 running on an NVIDIA RTX 4060 laptop GPU.

5.1 Cloud Modeling

This section visualizes each step of our implementation of Schneider's cloud modeling framework. It sequentially goes through each step, showing the effects on the cloud shape. The volumetric rendering techniques used on the cloud shape will be detailed in the next section.

5.1.1 Cloud Maps

Figure 5.1 visualizes rays from our camera intersecting the lower atmosphere and sampling the Cloud Coverage Map (c_{cov} in Equation 3.1) and the Cloud Type Map as discussed in Section 4.2.1. The color values shown represent the shape of the clouds. Areas in the Cloud Coverage Map that are lighter indicate regions where clouds will form, while areas in the Cloud Type Map that are lighter indicate areas where clouds will reach higher into the atmosphere. The actual shape of the cloud will become more apparent in the next sub section.

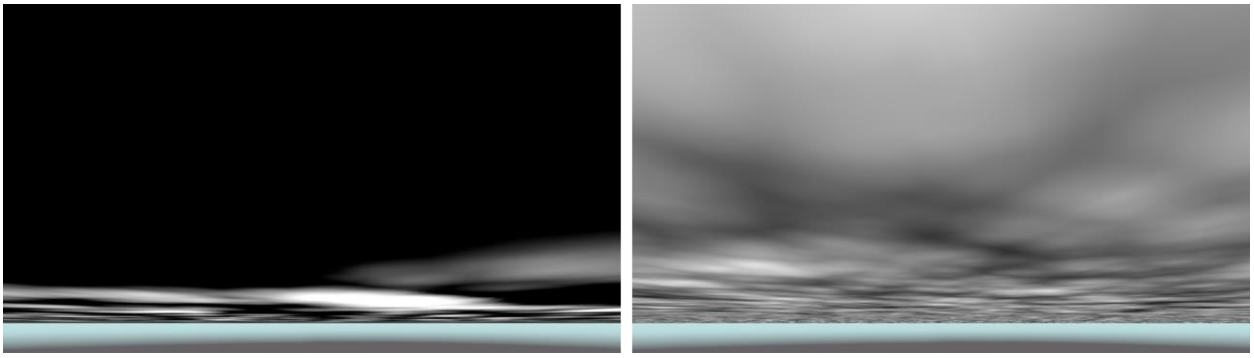


Figure 5.1: Visualization of Cloud Coverage Map (left) and Cloud Type Map (right) sampled at the intersection of the ray and the atmosphere.

5.1.2 Dimensional Profile

The product of the per pixel cloud coverage and the cloud type computes the dimensional profile according to Equation 3.1. The left image of Figure 5.2 illustrates a straightforward visualization of the dimensional profile, notice that the regions with no cloud coverage (top-darker region on the left image in Figure 5.1) are absent of cloud coverage. To better visualize the actual cloud

shape, the right image of Figure 5.2 applies basic illumination assuming a downward directional light. The base of the clouds matches the shapes seen in the dimensional profile, while the peaks at the top of the dimensional profile correspond to the lighter areas of the cloud type map. These dimensional profile values will be used to approximate cloud density in the next section.

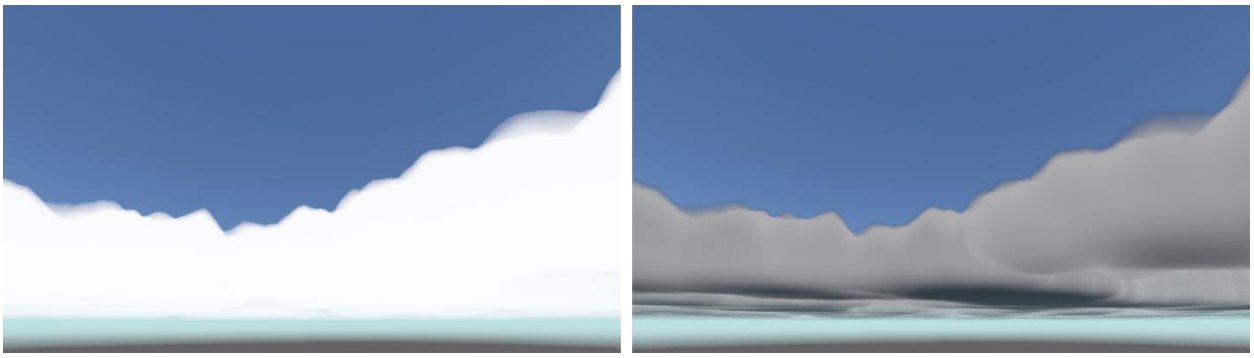


Figure 5.2: Render of dimensional profile (left) and shaded dimensional profile for better visualization(right).

5.1.3 Noise Erosion

To introduce detailed variation to the shape of the dimensional profile, noise erosion of Equation 3.2 is applied. The result is areas of the dimensional profile with values closer to 0 are eroded away by the procedural noise, while areas with values closer to 1 takes on the shape of the procedural noise. As described by Schneider, this erosion procedure leaves behind a detailed 3D volumetric distribution of cloud density, as Visualized in Figure 5.3. Due to the lack of illumination with light sources, the interior region of the cloud remains a constant white color.

Figure 5.3: Render of noise-eroded clouds prior to any illumination.

5.2. Cloud Refinements for Volumetric Rendering

This section presents the results of our implementation of Equation 3.3, focusing on our cloud-specific improvements to the volumetric rendering framework proposed by Fong [9]. We will see

how each improvement contributes to the final image, and the level of realism we are able to achieve.

5.2.1 Single Scattering

Figure 5.4 shows the results a of straightforward implementation of Equation 3.3 by ray marching and evaluating Equations 3.4 and 3.7 at distinct intervals to approximate the transmittance (T_r) and light scattering (L_{scat}). Some details of the cloud's interior structure can now be observed. The now visible details are the results of direct Sun light penetrating the cloud density and revealing internal shadows and highlighting the billowy tops and wispy bottoms. The overall illumination results, however, seem to resemble grey smoke rather than a white cloud. This is due to the absence of multiple scattering, as discussed in Section 3.2.2. The lack of multiple scattering results in less light energy throughout the volume affecting the overall visual details of the cloud.

Figure 5.4: Clouds illuminated by only single scattering. The lack of a multiple scattering approximation leads to a dark grey appearance.

5.2.2 Multiple Scattering

Figure 5.5 shows the results of applying multiple light scattering within a participating medium as proposed by Wrenninge [11]. In this case, 8 octaves of scattered light are approximated, or, N is set to 8 when evaluating Equation 3.8. This results in a significant brightening of the medium, resembling real-world clouds more closely. Areas that were once dull grey have become more vibrant white. However, compared to actual clouds, the interior self-shadowing and undersides remain overly dark. This issue is addressed in the next section.

Figure 5.5: Clouds with 8 octaves of multiple scattering approximation applied. The clouds now appear to have white peaks, but the underside of large dense clouds remain dark.

5.2.3 Ambient Light

Figure 5.6 integrates ambient light, approximating light reflecting off other elements in the environment, such as the ground or surrounding clouds. Ambient light is approximated by artificially increasing the illumination at the edges of the cloud based on a user-defined ambient light color.

Figure 5.6: Clouds illuminated with ambient light. This addition lightens areas of the clouds that were once overly dark.

5.2.4 Phase Function

The images in Figure 5.7 illustrate the results applying the phase function as described in Section 3.2.3. This refinement is most apparent in situations where clouds are between the camera and the sun. The phase function models the angular distribution of scattered light, brightening the edges of the clouds when backlit by the sun. This bright edge, or "silver lining," effect results from increased forward scattering, which directs more light towards the viewer. The right images of

Figure 5.7 illustrates how this effect is most prominent when the sun is directly behind the clouds during a sunset.

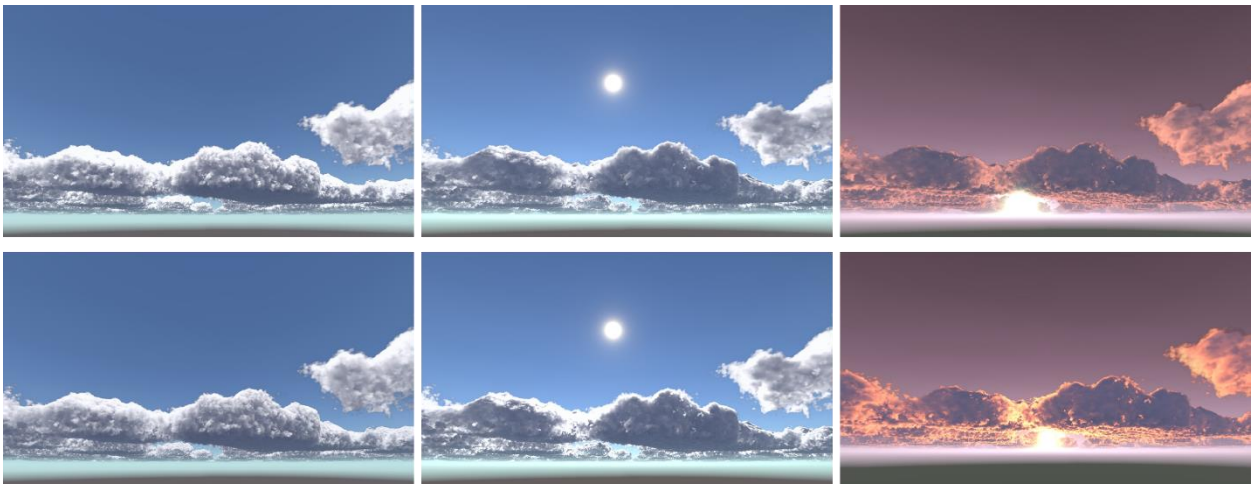


Figure 5.7: Clouds before (top row) and after (bottom row) applying the Phase function. In the bottom row images, the edges of clouds appear brighter as a result of increased forward scattering. These bright edges, or “silver lining”, are more prominent in areas of the cloud that are directly facing the sun.

5.2.5 Parameter Animation

Many of parameters in our implementation are directly accessible to the users. In the Unity editor, these parameters can readily be modified, or animated. For example, Figure 5.1.8 shows the results of changing the Cloud Coverage Map achieving the results of increasing cloud coverage over time. It should be noted that this result is a demonstration of potential capability and not an attempt to simulate the gathering of clouds in the real-world.

Figure 5.8: Cloud coverage levels ranging from 0 (left) to 1 (right), increasing at image by increments of 0.25.

5.3 Anti-Aliasing

This section visualizes the importance of our anti-aliasing strategy in enhancing the visual appearance of our renders. All of the rendered images presented thus far have had our anti-aliasing strategies applied. Figure 5.9 illustrates the aliasing artifacts without any anti-aliasing strategies applied. In this case, the one camera-ray per pixel leads to a hard edge on the outline of the cloud shape. Additionally, distinct layers in the cloud that correspond to the discrete sample points along the camera-ray march can clearly be observed. In this case, improving the quality of rendered images involves increasing the accuracy of the approximation by using more samples on the image plane and smaller ray march steps.

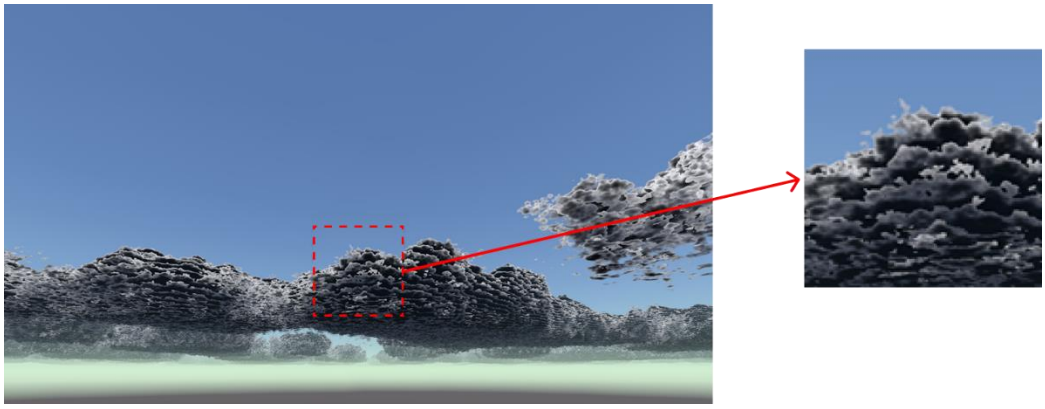


Figure 5.9: A closer view of aliasing effects. The layered appearance of the cloud is caused by the constant sampling rate of our ray march. Additionally, the dark appearance of the cloud is due to a poor scattering approximation that results when a large step size is used.

To address the layering artifacts on the image, we implement an anti-aliasing strategy using blue noise as described in Section 4.4. Figure 5.10 shows that by introducing randomness to our sampling position, the layering artifacts are much less prominent.

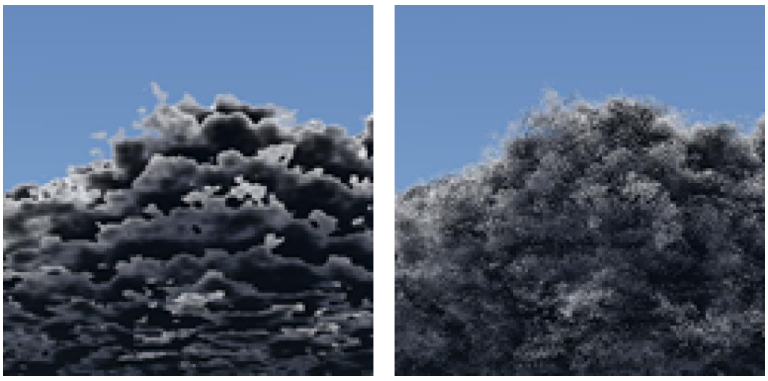


Figure 5.10: Render without anti-aliasing applied (left), and render with anti-aliasing applied. Due to the introduction of randomness in our sampling frequency, the overall shape of the cloud is softened.

To improve the scattering approximation, as detailed in Section 4.3, we adopt Hillaire's method of an energy conserving scattering integration [10]. This approach is capable of approximating the visible radiance based on a smaller number of samples along the camera-ray march. Figure 5.11 illustrates the results of the improved integral approximation. The improvement better approximates the conservation of light energy within the medium and results in more accurate light scattering measurement.

Figure 5.11: Rendering using straightforward radiance integration approximation (left), compared to rendering using Hillaire's improved integration approximation. Both renders use a step size of 80. The render on the left does not properly conserve energy in its integration leading to a much darker appearance, while the render on the right does account for this, leading to more accurate lighting.

To better approximate volumetric details, our temporal anti-aliasing strategy, as described in Section 4.5 is implemented. This strategy allows us to approximate the results of a ray march with a much smaller step size, without actually reducing the ray march step size. The image on the right in Figure 5.12 is the result of a rendering window of 8 frames, where the final image is the average of these 8 frames. The smoother and more refined image has an increased resemblance with real-world clouds.

5.12: Image without Temporal Anti-Aliasing (left), and image with Temporal Anti-Aliasing (right). The addition of our TAA strategy greatly reduces the visible noise in the image.

Our anti-aliasing strategy supports the rendering of high-quality clouds with a smaller number of samples taken along the camera-ray march. By averaging the previous 8 frames in our rendering window, we can achieve the same level of detail in our ray march using a step size that is 8 times larger. Figure 5.13 illustrates this by comparing an image rendered with a step size of 10 to one rendered with a step size of 80. The effect of these methods on efficiency will be discussed in the next section.

Figure 5.13 Render with step size of 8 (left) compared to render with step size of 80 with anti-aliasing techniques applied (right). Although the right has 10x the step size as left, very little difference in detail can be seen in the render.

5.4 Optimizations

This section presents the effects of the different techniques we use to optimize our cloud rendering system. The reported run-time measurement are based on Unity's profiling tool. Typical real-time applications update at a rate of 60 frames per second, or a rendering time of 16.67 ms. Therefore, for our system to be considered real-time, it must render a frame within this time allotment.

In general, even if this metric is met, it is still not sufficient for most real-time applications. This is because clouds are often only a fraction of what needs to be rendered. For example, if the clouds render at 16.67 ms per frame, it leaves no room for other elements to be rendered, such as the terrain. Because of this, our goal is to reduce the render time as much as possible without compromising the final render.

We will use Figure 5.13 as a reference image to compare against the renders with our optimizations applied. This reference render uses no optimization strategies and employs a step size of 9. On average, it takes 1558 ms per frame to render.

Figure 5.13: Render with no optimization methods applied. It has a step size of 9 and a render time of 1558ms per frame.

5.4.1 Early Exit

The first optimization we apply is our early exit strategy. This strategy immediately ends the ray march when the transmittance value falls below a user defined threshold. The two images of Figure 5.14 are results of before (left) and after (right) the early exit strategy with transmittance value set to 0.075. As can be observed, the differences in the images are nearly imperceptible. This optimization improves our render time to 373 ms per frame.

Figure 5.14: Reference image (left) compared to image with early exit strategy with a transmittance threshold value of 0.075 (right).

5.4.2 Skip Low Density Samples

The next optimization is to skip low-density samples where samples with cloud densities of than 0.01 are ignored. The two images of Figure 5.15 are results of before (left) and after (right) this strategy. Once again, it is challenging to observe any differences. This optimization further reduced rendering time to 34.6 ms per frame with a step size of 9.

Figure 5.15: Reference image (left) compared to image where density values less than 0.01 are skipped (right).

5.4.3 Temporal Anti-Aliasing

Through the use of our Temporal Anti-Aliasing strategy, we are able to increase the step size from 9 to 100, with a 8-frame window. The increase step size results in a reduced rendering time of 6.5ms per frame. The two images of Figure 5.16 are results of before (left) and after (right) this

strategy. A slight darkening of the cloud shadows can be seen in the image using our TAA approach, however it is very subtle.

Figure 5.16: Reference image using a step size of 9 (left) compared to image using TAA strategy with a step size of 100 and an 8-frame window (right).

5.4.4 Adaptable Volume Step Size

The images in Figure 5.17 are results of using our dynamic step size optimization described in Section 4.6. This optimization uses a step size of 100 and achieves a render time of 6.0 ms per frame.

Figure 5.17: Reference image (left) compared to image using dynamic step size strategy (right).

5.4.5 Image Resolution

Finally, clouds generally appear fluffy and have blurry, semi-transparent edges. This softness and lack of high-frequency details make it challenging to distinguish or identify features within the interior of cloud volumes. Additionally, it is difficult to identify features when a cloud image is zoomed in. Due to this softness, we have found that when clouds are rendered at a lower resolution and then upscaled to fit the image, there is very little visual difference. Figure 5.17 shows the results of rendering the cloud image at full (left), half (center), and quarter (right) resolutions. When rendered at full resolution, we achieved a frame time of 6.0 ms. At half resolution, the frame time improved to 2.7 ms. At quarter resolution, we further reduced the frame time to 1.8 ms.

Figure 5.18: Image rendered at full resolution (left), half resolution (center), and quarter resolution (right).

6. CONCLUSION

We have presented our implementation of a system for rendering volumetric clouds. Our modeling approach is based on Schneider's method of modeling cloud density as a combination of decompressed 2D data fields and 3D procedural noise[4]. Our rendering approach follows the volumetric framework provided by Fong[9]. We offer a unique set of enhancements to this volumetric rendering approach that improve its ability to realistically render clouds. These improvements include:

- Wrenninge's approach for approximating multiple scattering to improve radiance calculation [11].
- Jendersie & d'Eon's phase function to enhance the accuracy of the angular distribution of light scattering based on the particle size of water droplets within clouds [12].
- Hillaire's improved scattering integration to ensure energy-conserving radiance values when integrating with a large step size [10].

Additionally, to capture additional details within the ray march with minimal additional computational cost, we introduce a temporal anti-aliasing strategy that unifies pixel and volumetric sampling. The pixel area sampling integrates a blue-noise distribution with an n-rooks offset, while volumetric samples follow a stratification strategy, amortizing results over n frames.

The resulting system is capable of rendering expansive cloudscapes well within real-time requirements, achieving frame rates between 2 and 3 milliseconds on a standard laptop. Our clouds have the ability to dynamically change their shape based on user-defined weather conditions. In addition, we are able to simulate light transport through our cloud material to provide realistic illumination. This work can serve as a case study for those interested in creating their own real-time volumetric cloud rendering solution.

There are some limitations to consider in our system. One limitation is that the generation of our dimensional profile is based on observation rather than the physical processes that dictate real-world clouds. Our cloud maps, which define the large-scale shape of the clouds, are generated based on these observations. As a result, the overall shape of the clouds may lack realism. Additionally, our process only accounts for low-altitude clouds, meaning we cannot accurately simulate high-altitude clouds such as Cirrus clouds. Another limitation, related to our temporal anti-aliasing strategy, is the possibility of blur when the camera moves quickly. This issue becomes more apparent as the size of the frame window increases.

Possible avenues for future work include an improvement to the cloud modeling system. Effort could be made to base the cloud map generation system on the physics of cloud formation rather than observation. Additional parameters such as temperature, humidity, time of day, wind conditions, and the terrain that clouds form over could be considered in map generation to achieve more realistic results. Additionally, work could be done to enhance the way in which cloud maps

evolve over time. Webanck et al. describes a method of interpolating between different cloud textures based on keyframes, which could be explored [31]. Schneider has recently described an updated cloud volumetric system that utilizes voxel-based modeling [32]. Applying this work to our system could improve the cloud modeling process and the resulting cloud shapes. Moreover, the cloud renders we present do not interact with the environment. They are rendered as a post-process, meaning this current implementation renders clouds on top of other elements in the scene. Future work could ensure that clouds render behind objects closer to the camera and interact with the scene, such as casting shadows onto other objects. Furthermore, additional improvements can be made to our anti-aliasing system. Yang describes a method of using pixel reprojection based on the camera movement of previous frames to reduce the blurring experienced when the camera is quickly moved [28]. This approach could be incorporated to enhance the visual quality of our renders.

REFERENCES

[1] T. Hädrich, M. Makowski, W. Pałubicki, D. T. Banuti, S. Pirk, and D. L. Michels, "Stormscapes: simulating cloud dynamics in the now," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–16, Dec. 2020, doi: 10.1145/3414685.3417801.

[2] G. Y. Gardner, "Visual simulation of clouds," in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, in SIGGRAPH '85. New York, NY, USA: Association for Computing Machinery, Jul. 1985, pp. 297–304. doi: 10.1145/325334.325248.

[3] S. Hasegawa, J. Iversen, H. Okano, and J. Tessendorf, "I love it when a cloud comes together," in *ACM SIGGRAPH 2010 Talks*, Los Angeles California: ACM, Jul. 2010, pp. 1–1. doi: 10.1145/1837026.1837043.

[4] A. Schneider, "Nubis: authoring real-time volumetric cloudscapes with the Decima Engine," *SIGGRAPH Advances in Real-Time Rendering in Games Course. ACM*, pp. 619–620, 2017.

[5] J. Novák, I. Georgiev, J. Hanika, J. Křivánek, and W. Jarosz, "Monte Carlo methods for physically based volume rendering," in *ACM SIGGRAPH 2018 Courses*, Vancouver British Columbia Canada: ACM, Aug. 2018, pp. 1–1. doi: 10.1145/3214834.3214880.

[6] R. Clausse and L. Facy, *The Clouds*. Grove Press, 1961.

[7] A. Marshak and A. Davis, *3D radiative transfer in cloudy atmospheres*. Springer Science & Business Media, 2005.

[8] P. Kobak and W. Alda, "Modeling and rendering of convective cumulus clouds for real-time graphics purposes," *Computer Science*, vol. 18, pp. 241–268, 2017.

[9] J. Fong, M. Wrenninge, C. Kulla, and R. Habel, "Production volume rendering: SIGGRAPH 2017 course," in *ACM SIGGRAPH 2017 Courses*, Los Angeles California: ACM, Jul. 2017, pp. 1–79. doi: 10.1145/3084873.3084907.

[10] S. Hillaire, "Physically based sky, atmosphere and cloud rendering in frostbite," presented at the ACM SIGGRAPH, 2016, pp. 1–62.

[11] M. Wrenninge, "Art-directable multiple volumetric scattering," in *ACM SIGGRAPH 2015 Talks*, Los Angeles California: ACM, Jul. 2015, pp. 1–1. doi: 10.1145/2775280.2792512.

[12] J. Jendersie and E. d'Eon, "An Approximate Mie Scattering Function for Fog and Cloud Rendering," in *ACM SIGGRAPH 2023 Talks*, Los Angeles CA USA: ACM, Aug. 2023, pp. 1–2. doi: 10.1145/3587421.3595409.

[13] "List of cloud types," *Wikipedia*. Apr. 28, 2024. Accessed: May 21, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_cloud_types&oldid=1221274379

[14] M. N. Zamri and M. S. Sunar, “Atmospheric cloud modeling methods in computer graphics: A review, trends, taxonomy, and future directions,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, Part B, pp. 3468–3488, Jun. 2022, doi: 10.1016/j.jksuci.2020.11.030.

[15] W. T. Reeves, “Particle systems—a technique for modeling a class of fuzzy objects,” in *Seminal graphics: pioneering efforts that shaped the field, Volume 1*, vol. Volume 1, New York, NY, USA: Association for Computing Machinery, 1998, pp. 203–220. Accessed: May 06, 2024. [Online]. Available: https://doi.org/10.1145/280811.280996

[16] K. R. Christiansen, “The use of Imposters in Interactive 3D Graphics Systems,” *Department of Mathematicsand Computing Science Rijksuniversiteit Groningen Blauwborgje*, vol. 3, 2005.

[17] M. J. Harris and A. Lastra, “Real-time cloud rendering for games,” presented at the Proceedings of Game Developers Conference, 2002, pp. 21–29.

[18] J. T. Kajiya and B. P. Von Herzen, “Ray tracing volume densities,” *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 165–174, 1984.

[19] B. Hapke, *Theory of Reflectance and Emittance Spectroscopy*, 2nd ed. Cambridge: Cambridge University Press, 2012. doi: 10.1017/CBO9781139025683.

[20] A. Bouthors, “Realtime realistic rendering of clouds,” Jun. 2008.

[21] P. K. Wang, Ed., “The shape and size of cloud and precipitation particles,” in *Physics and Dynamics of Clouds and Precipitation*, Cambridge: Cambridge University Press, 2013, pp. 27–67. doi: 10.1017/CBO9780511794285.003.

[22] “Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine,” Unity. Accessed: May 20, 2024. [Online]. Available: https://unity.com/

[23] S. Worley, “A cellular texture basis function,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, Aug. 1996, pp. 291–294. doi: 10.1145/237170.237267.

[24] B. B. Mandelbrot and J. W. Van Ness, “Fractional Brownian Motions, Fractional Noises and Applications,” *SIAM Review*, vol. 10, no. 4, pp. 422–437, 1968.

[25] T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire, *Real-time rendering*, Fourth edition. in An A K Peters book. Boca Raton London New York: CRC Press, Taylor & Francis Group, 2018.

[26] M. Nuebel, “Introduction to Different Fog Effects,” *ShaderX2: Introductions & Tutorials with DirectX 9*, p. 151.

[27] D. Heck, T. Schlömer, and O. Deussen, “Blue noise sampling with controlled aliasing,” *ACM Trans. Graph.*, vol. 32, no. 3, pp. 1–12, Jun. 2013, doi: 10.1145/2487228.2487233.

[28] L. Yang, S. Liu, and M. Salvi, “A Survey of Temporal Antialiasing Techniques,” *Computer Graphics Forum*, vol. 39, no. 2, pp. 607–621, May 2020, doi: 10.1111/cgf.14018.

[29] K. Chiu, P. Shirley, C. Wang, and M. Sampling, “Graphics gems iv,” *by Paul S. Heckbert. Chap. Multi-jittered Sampling*, pp. 370–374, 1994.

[30] C. Wang and K. Sung, “Multi-Stage N-rooks Sampling Method,” *Journal of Graphics Tools*, vol. 4, no. 1, pp. 39–47, Jan. 1999, doi: 10.1080/10867651.1999.10487500.

[31] A. Webanck, Y. Cortial, E. Guérin, and E. Galin, “Procedural cloudscapes,” in *Computer Graphics Forum*, Wiley Online Library, 2018, pp. 431–442.

[32] N. Tatarchuk, S. Aaltonen, and A. Schneider, “Advances in Real-Time Rendering (Part I),” in *ACM SIGGRAPH 2023 Courses*, in SIGGRAPH ’23. New York, NY, USA: Association for Computing Machinery, Feb. 2024, p. 1. doi: 10.1145/3587423.3607877.