Real-Time Atmospheric Cloud Rendering System

Parker Ford
Computing and Software Systems
University of Washington Bothell
Bothell, United States
parker.g.ford@gmail.com

Kelvin Sung
Computing and Software Systems
University of Washington Bothell
Bothell, United States
ksung@uw.edu

***Abstract*— High-quality cloud rendering is important when creating believable virtual outdoor environments. However, because of the complex physical processes involved in cloud formation and illumination, rendering realistic clouds in real-time is challenging. Research in the area has targeted limited-scope issues in relative isolation and presented impressive but focused results. For example, solutions for cloud modeling without details on rendering, or approximations of specific inter-scattering illumination without details on the rendering framework. These isolated solutions, combined with the required advanced system knowledge for efficient implementation, present a high barrier for holistic exploration and experimentation. This paper presents a real-time cloud rendering system that integrates an effective cloud modeling solution [1] with an efficient implementation [2] of a volumetric rendering framework [3] and incorporates recent findings in inter-droplet scattering [4] [5]. Additionally, a novel Temporal Anti-Aliasing solution is introduced to unify and support independent control over the sampling of both pixel areas and volumetric intervals. The resulting real-time cloud rendering system exemplifies recent advancements in the field and can serve as a convenient starting point for further investigations into cloud formation and illumination.**

Keywords—Atmospheric Clouds, Real-Time, Modeling, Rendering, Multiple Scattering, Phase Function

I. Introduction

There are high demands for realistic clouds in virtual outdoor environments across applications ranging from video games and films [1] [6], to environmental simulations [7]. These demands have prompted significant research, yielding impressive results.

Rendering clouds in real-time is challenging due to the complex physical processes involved in their formation [8]. Recent approaches address this challenge by focusing on specific issues with limited scope. For example, Schneider's efficient noise-based procedural method focuses on the details of modeling the shape of clouds, but does not discuss the specific implementation details of the rendering process [1]. Similarly, sophisticated cloud illumination effects such as multiple scattering [4] or directional distribution of droplet interreflection [5] are typically presented in isolation from the formation or general rendering framework of clouds.

While these results are exciting, it is also true that integrating them into a coherent system supporting further exploration and experimentation is non-trivial. This is especially challenging when real-time interactions are desired, as developing these applications require elaborate knowledge of GPU architecture and shading language.

This paper presents a novel integration of recent advances in generating realistic clouds and a novel Temporal Anti-Aliasing (TAA) strategy that unifies the sampling of pixel areas and volumetric intervals. The system pairs Schneider's computationally efficient and visually appealing cloud modeling [1] with an extension of Fong et. al.'s simple and flexible volumetric rendering framework [3]. Additionally, the rendering framework is generalized to support multi-bounce diffusion of light within a volume [4], and the directional distribution of scattering between light and microscopic water droplets [5]. For increased efficiency and accuracy, the integration of radiance along volumetric intervals is based on the energy-conserving scattering approximation from Hillaire [2]. The novel TAA strategy is based on pre-computed blue-noise [9] and n-rooks sample position offsets [10], where the choice of pixel and volumetric sample positions are unified by associating corresponding stratified regions in the pixel area and volumetric interval with the frame being rendered.

The resulting system,[1] in addition to showcasing the novel TAA, serves as an example that integrates cloud modeling [1], rendering [3], and generalizes the involved formulations required to support advanced inter-droplet scattering [4] [5] in an efficient implementation [2]. The system is capable of rendering expansive cloudscapes at real-time rates. For modeling, users can control and experiment with cloud coverage, type, and distribution over altitude by modifying or replacing the corresponding texture maps. Additionally, the different illumination effects can be toggled, and, for the more advanced user, modified or replaced for further investigations.

II. Background

Building a coherent system to support the investigation and experimentation of cloud formation and rendering in real-time requires an understanding of the general approaches to existing solutions. Although it is possible to create illusions of clouds via imposter-based rendering [11], or through artistic drafting with commercial systems [12], these approaches lack proper underlying formulations and do not offer systematic opportunities for investigation into clouds and their rendering.

Clouds form when water evaporates, rises, reaches cool pockets of air, condenses onto dust particles, and forms into clusters of water droplets. Depending on the atmospheric conditions, clouds with varying appearances can form at different altitude ranges [13]. Many cloud generation solutions follow a two-step method to follow this cloud formation process: modeling and rendering. Modeling approximates the shapes of clouds, while rendering produces images of these shapes by applying an appropriate illumination model [14].

A. Cloud Modeling

The modeling of clouds can be based on simulations that replicate the natural processes involved in cloud formation. For example, groups of water molecules can be simulated moving through the atmosphere, following the laws of thermodynamics [15]. While capable of producing realistic

[1] Accessible off public repository: https://github.com/parker-ford/Real-Time-Atmospheric-Cloud-Rendering-System

979-8-3503-7603-6/24/$31.00 ©2024 IEEE

looking and physically accurate clouds, these types of approaches require significant computational resources and do not lend themselves well to real-time applications.

An alternative approach to cloud modeling trades physical accuracy for computational efficiency while attempting to maintain a realistic appearance. For example, Schneider points out that water droplet density distribution in the atmosphere can be modeled as a series of functions that depend on the likelihood of a cloud being present, the type of cloud, and the variation of cloud presence over altitude [1]. He then demonstrates that these functions can be approximated using lookup tables in the form of textures, based on appropriate pre-computations or artistic creations. With suitable textures, this approach can generate visually appealing clouds. Due to the efficiency of texture lookup, this approach is appropriate for real-time applications.

B. Cloud Rendering

The results of the cloud modeling step are water droplet density distributions and may include details such as droplet size and impurity content. Cloud rendering is the process of visualizing this information and can be formulated as the scattering of light as it propagates through the droplets [3].

$$L(c,-v) = T(c,w)L_w(w,v) + \int_{t=0}^{||w-c||} T(c,c-vt)L_s(c-vt,v)\sigma_s(c-vt)dt \quad (1)$$

Assuming a camera located at c with a visible position w behind a volumetric medium, and v being the direction from w to c through the volumetric medium, Equation 1 describes the radiance, or energy associated with a light ray arriving at c along v [3]. In this formulation, $T(a,b)$ is the transmittance, or, the percentage of light capable of passing through the interval from position a to b, L_w and L_s are the radiance at position w and from all light sources that scattered towards direction v, and σ_s is the scattering coefficient, or, the likelihood that scattering will occur at a given position. Note that in the absence of volumetric density, or when light travels through a vacuum, T would be one and σ_s would be zero.

While simple, Equation 1 is also elegant, with the abstract functions, T, L_s, and σ_s offering flexibility for generalization to model advanced effects such as droplet interreflection and directional dependencies of scattered light. Additionally, the formulation aligns well with ray-cast based implementations that can be readily parallelized on modern GPUs.

III. Solution Design

A coherent system for the investigation of real-time cloud formation and rendering should present and interface two modules: modeling and rendering.

A. The Cloud Modeling Module

Schneider's approach is chosen for the potential realism, real-time performance, and, very importantly, the parameters for adjustments. Our solution generally follows the original design, where the likelihood of cloud presence (cloud coverage), the type of cloud (cloud type), and the variation of cloud presence based on altitude (vertical profile) are represented by three separate lookup table texture maps. The coverage and type maps are accessed based on the longitude and latitude (x and z coordinates) of an atmospheric position. The vertical profile is accessed using the cloud type value and the altitude (y coordinate) of the position. In this way, given an atmospheric position, the three texture lookups result in two values: cloud coverage and vertical profile. The product of these two values forms the dimensional profile, which determines the probability of cloud density being present at a given position.

Based on the dimensional profile, Schneider proposes to capture the details of cloud surface variation over altitude by blending two types of noise functions: Perlin for wispy details at lower altitudes and Inverted Worley Noise for billowy details at higher altitudes. This blended noise is then used to erode the dimensional profile by subtracting the inverse of the dimensional profile to produce the actual cloud density that defines the shape of clouds.

B. The Cloud Rendering Module

Based on Equation 1, the cloud rendering solution design involves defining the abstract functions: transmittance (T), scattered radiance at the visible position (L_w) and from the light sources (L_s), and the scattering coefficient (σ_s). The following discussion is organized to first describe solutions that follow Fong et. at. [3], and then details the novel integration of other volumetric solutions.

1) Following Existing Design

The transmittance, T, in Equation 1 is defined according to the Beer-Lambert Law, with σ_t being the extinction coefficient, or, the density of the medium. The transmittance from position a to b is defined as follows:

$$T(a,b) = e^{-\int_a^b \sigma_t dx} \quad (2)$$

The radiance at the visible position scattered towards the camera direction, L_w, can be approximated based on any illumination model, e.g., Phong or a skybox texture. The scattering coefficient, σ_s, or the likelihood of scattering occurring at some position, is a user defined scaling of the cloud density at the same location computed in the cloud modeling module.

2) Integration of Recent Results

Fong et. al. models the scattered radiance of light sources from position x towards the camera direction, v, as

$$L_s(x,v) = \sum_i^{all\ lights} T(x,y_i)L_i(y_i,x-y_i)P(v,x-y_i) \quad (3)$$

Where $T(x,y_i)$ is the transmittance from the position x to the i^{th} light source position y_i; L_i is the radiance from the light position that is emitted towards x, and P is the Phase function, which models the probability that the radiance from the light source will scatter towards the camera direction, v. Details of the Phase function will be discussed in a later section. For cloud rendering, it is reasonable to assume a single light source, the Sun, which emits energy uniformly in all directions. With a single light source, and letting $l = x - y$ be the light direction from the Sun, Equation 3 becomes,

$$L_s(x,v) = T(x,\mathrm{y})\, L_{sun}(y,l)\, P(v,l) \quad (4)$$

a) Multiple Scattering

The L_{sun} term in Equation 4 is the radiance from the Sun that is emitted towards the position x, and $L_s$is the percentage that scattered towards the camera, or, the radiance towards the camera after a single scattering. In general, additional radiance will arrive at position x as a result of Sun light scattering from neighboring water droplets. A portion of this radiance will also scatter towards the camera, or, the radiance towards the

camera after multiple scattering. Equation 4 does not model multiple scattering.

Accurate simulation of multiple scattering requires accounting for all scattered radiance from all water droplets, which is algorithmically complicated and computationally expensive. Wrenninge et al. proposed to approximate the radiance from secondary and subsequent generations of light scattering by strategically diminishing the original single scattering radiance as light continues to scatter in between water droplets [4]. With the assumption that the density of water droplets (σ_t) is a constant, the transmittance from position x to y in Equation 2 evaluates to $e^{-\sigma_t\|y-x\|}$, or, $e^{-\sigma_t\|l\|}$ (since $l = x - y$), Wrenninge et. al. modelled multiple scattering as,

$$L_{mult}(x,v) = \sum_{i=0}^{N-1} e^{-\sigma_t a^i\|l\|}\, b^i L_{sun}(y,l)\; P(v,l) \qquad (5)$$

Equation 5 models the total radiance arriving at x and scattering towards v by summing N generations of scatterings. The constants a and b are user controlled with values between 0 and 1. Note that $i = 0$ corresponds to the first-generation single scattering of Equation 4. The terms associated with $i > 0$ correspond to subsequent generations that arrive at x and scatter towards v. The a^i term modulates the transmittance distance with the assumption that neighboring water droplet scatterings have shorter distances to travel. The b^i term reduces the associated radiance modeling the exponential decrease in subsequent scattering generations.

b) The Phase Function

As mentioned, a Phase function describes the directional distribution of scattering when a light ray collides with a particle. This distribution depends on the angle of the incoming light ray and the properties of the particles, e.g., size, shape, reflective index.

The Phase function of clouds is determined by the size of the microscopic water droplets, which, in this case, are comparable in size to the wavelength of light. For this reason, Mie scattering can serve as an appropriate cloud Phase function. However, Mie scattering is complex and computationally expensive, often modelled as extensive lookup tables. Our implementation approximates Mie scattering by adopting the model proposed by Jendersie & d'Eon [5] where the diameter of water particles is assumed to be $20\mu m$.

IV. Implementation

Implementation of the presented design must include a user interface for parameter manipulation, efficient texture access for computing cloud density based on Schneider's formulation, and per-pixel computation for approximating Equation 1. Additionally, the real-time requirement suggests the need for parallelization of the per-pixel operations. Based on these requirements, we have chosen Unity3D [16] as the implementation base. As a game engine, Unity3D offers extensive support for user interface customization, an advanced graphics API for GPU access, including the underlying SIMD architecture, and most importantly, Unity3D is free for educators and students.

A. System Structure and User Interface

Based on custom scripts and shaders, the system receives per-frame parameters from the user to configure and perform the per-pixel computations in parallel with the shaders. For parallel pixel computation, the Unity3D Compute Shader's process is invoked twice. First, to approximate Equation 1 and save results as separate per-frame buffers. Second, to combine relevant buffers in a sliding window that implements the novel TAA strategy (to be discussed). Lastly, the Unity3D post-process is triggered to transfer the TAA results as the per-frame rendered image.

The UI supports per-frame parameter manipulation. For cloud modeling users can adjust the altitudes of the lower and upper atmospheres where clouds can form, choose textures for cloud density computations (σ_s), and set the distance that a texture spans across the atmosphere. For cloud rendering, users have control over the ray march step size, the intensity and color of the Sun (L_{sun}), the number of multiple scattering generations (N in Equation 5), the constants of the Phase function, and the number and resolution of frames in the TAA sliding window.

B. Equation 1 Approximation

Since Equation 1 models the transport of radiance by following a single ray, the traditional ray marching algorithm [17] is ideal for implementation. A custom shader is developed to approximate Equation 1 for individual pixels where each Unity3D Compute Shader's frame cycle corresponds to computing one color, or taking one sample per-pixel for the entire frame.

With the assumption that the Earth is spherical, the lower and upper atmospheric altitudes define two corresponding concentric spheres centered around the Earth and the camera is defined to be located at the surface of the Earth. Rays, or camera rays, are cast from the camera through each pixel area to intersect with these spheres, resulting in visible intervals between the lower and upper shells. This corresponds to the $\|w - c\|$ integral range of Equation 1. A typical ray march of n-steps involves dividing this interval into $\Delta = \frac{\|w-c\|}{n}$ segments and approximating the integral based on the mid-point rule.

Hillaire [2] considered the mid-point rule by first dividing the integral according to the n segments and reformulating the integral in Equation 1. For readability, let $d_t = c - vt$, and p_i be a position in each interval,

$$\int_{t=0}^{\|w-c\|} T(c,d_t)L_s(d_t,v)\sigma_s(d_t)dt = \sum_{i=0}^{n-1}\int_{t=i*\Delta}^{(i+1)\Delta} T(c,p_i)L_s(p_i,v)\sigma_s(p_i)dt \qquad (6)$$

They went on to point out that L_s, the radiance from the light source scattered in the v direction, and σ_s, the scattering coefficient based on the cloud density, are computationally costly. For this reason, in typical implementations, these two terms are approximated once at a single selected position, p_i, and assumed to be constant across the entire segment. As such, these two terms can be taken out and the approximation of the integral can be expressed as,

$$\int_{t=0}^{\|p-c\|} T(c,d_t)L_s(d_t,v)\sigma_s(d_t)dt \approx \sum_{i=0}^{n-1} L_s(p_i,v)\sigma_s(p_i)\int_{t=i*\Delta}^{(i+1)\Delta} T(c,p_i)dt \qquad (7)$$

With transmittance, T, defined from Equation 2 and σ_t (density of the water droplets) being a constant, recall that $\Delta =$

$\frac{\|w-c\|}{n}$ is the length of segments of the volumetric interval, Hillaire showed that the righthand side of the equation evaluates to,

$$\sum_{i=0}^{n-1} L_s(p_i, v)\sigma_s(p_i)\frac{(1-e^{-(i+1)\Delta})}{\sigma_t} \quad (8)$$

Our compute shader implementation approximates the integral in Equation 1 with Equation 8 where the L_s term is approximated using Equation 5, with x being p_i and the user having control over N. Note that the transmittance in Equation 5 is from the sample position p_i towards the light source while in Equation 8 it is from p_i towards the camera. These two are distinct. To properly approximate the transmittance in Equation 5, at every p_i a separate light ray is cast to march towards the Sun. Lastly, the Phase function in Equation 5 is based the model proposed by Jendersie & d'Eon [5] with a particle diameter of $20\mu m$.

The scattering coefficient, σ_s, is computed by invoking Schneider's density derivation function (ρ, to be discussed) and scaling the results with a user-controlled parameter. In this way, the two conceptual steps of cloud modeling and rendering are implemented as an integrated computation approximating Equation 1. In other words, there is no separate process to derive the cloud density before rendering.

C. Cloud Density Determination

While intuitive and elegant, one of the major challenges in generating clouds based on Schneider's approach is determining the contents of the texture maps: the coverage of clouds, the types of clouds, and how the density of cloud types change over altitude. We generated these texture maps algorithmically based on observations of clouds in the real world [18].

As depicted in Figure 1, the left image is our generated coverage map where accumulated frequencies of inverted Worley noise are remapped by Perlin noise to model varied, natural looking cloud shapes with dense centers that gradually fade away near their edges. The center image is our cloud type texture generated based on multiple frequencies of inverted Worley noise to capture the variations of cloud types across large regions as well as the small-scale height variations that are present in clouds.

The image on the right of Figure 1 shows the cloud vertical profile. The horizontal axis represents cloud type where values are obtained from cloud type texture lookup. The vertical axis represents altitude with values defined by the y-coordinate of a sample position. In this way, from left to right, this texture describes the chances of a particular type of cloud being present at specific altitude. For example, cloud types with smaller x-values can only exist at lower altitudes (smaller y values). In this way, our implementation interprets cloud type values from 0 to 1 as cloud formations from low to high altitudes.

Recall that the dimensional profile is a product of the coverage and vertical profile texture lookup results and that it is eroded by a noise function to generate the actual cloud density distribution. Following Schneider's approach, our eroding noise function is a piecewise linear interpolation between wispy Perlin noise at low altitudes and billowy inverted Worley noise at high altitudes.

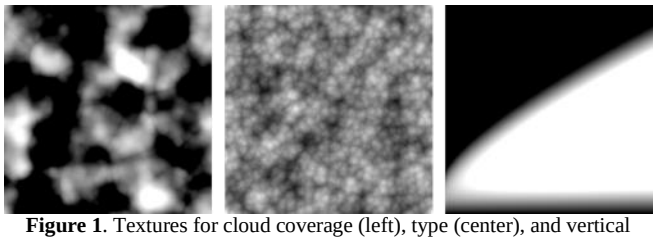

Figure 1. Textures for cloud coverage (left), type (center), and vertical profile (right).

At runtime, when approximating Equation 1, the coordinate values of sample position p_i are normalized for texture lookup of cloud coverage and vertical profile. The dimensional profile, d, is then computed, eroded by the noise function, N, and modulated by the user controlled ρ_c to compute the actual cloud density, ρ,

$$\rho(p_i) = (N(p_i) - (1-d))\rho_c \quad (10)$$

D. Unified Temporal Anti-Aliasing

Volumetric rendering involves sampling in two separate dimensions: the areas within pixels and intervals within the volumetric medium. Traditional anti-aliasing strategies focus on pixel-area and typically do not consider other dimensions [19]. The general approach is to independently sample additional dimensions based on each pixel sample, e.g., taking v number of volumetric samples per pixel sample. This approach can lead to over-sampling. For example, m samples per pixel would mean evaluating the volumetric interval with mv volumetric samples, an inefficient oversampling.

A unified anti-aliasing approach should consolidate the considerations of all dimensions and support increasing sampling rate in each dimension independently. For example, it should allow doubling the sampling rate of pixel area without affecting that of the volumetric interval, i.e., maintaining a total of v-number volumetric samples when increasing the pixel area sampling from one to two.

In our approach, the considerations for the two separate dimensions are unified by averaging single-sample frames generated with strategically distributed sample positions in both dimensions. The first frame uses sampling positions that are offsets from the center based on pre-computed blue noise values [9]. Three sets of blue noise values, two for pixel and the third for volumetric position offsets, are pre-computed and stored as the three components in a color texture.

Super sampling of the pixel area is implemented by filtering subsequent single-sample frames with sample positions that are offsets from the first frame by a corresponding pre-computed N-Rooks pattern [10]. The super sampling of the volumetric interval along each camera ray is unified with the current pixel sampling rate. When v-volumetric samples are desired, each of the v-volumetric intervals along a camera ray would be divided into n segments, where n is the current pixel area super sampling rate. In each single-sample frame, only the corresponding $(v \bmod n)^{th}$ segments are sampled.

The unified TAA maintains a window of n single-sample frames and averages them into the final rendered image. The window slides to contain newly rendered frames by removing the oldest with appropriate scaling. By unifying the considerations for the dimensions, this approach supports the independent refinement of each.

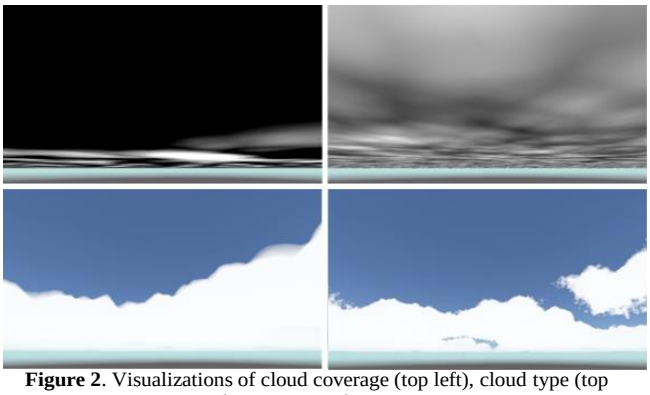

Figure 2. Visualizations of cloud coverage (top left), cloud type (top right), dimensional profile (bottom left), and actual cloud density distribution (bottom right).

E. Other Details

Straightforward optimizations are also implemented including early exit when transmittance, radiance, or cloud density values are small. Additionally, the approach of varying ray march step size according to current cloud density values [1] are adopted. Our implementation also supports rendering at half or quarter resolutions. Due to the fluffy, blurry, and semi-transparent low-frequency content, upscaling cloud images can be acceptable, especially during early phases of investigations.

V. RESULTS

The results presented in this section are based on textures from Figure 1. It is important to note that these textures can be replaced in real-time. The following images are rendered on a modest 82YA Lenovo laptop with an 13th generation Intel-i7 processor and an NVIDIA RTX 4060 GPU. The low- and high-altitude spheres are 200 and 1200 meters with 100-meter volumetric sample interval length. On this machine with these settings, the average rendering time for the presented images are approximately between 7 to 8 msec.

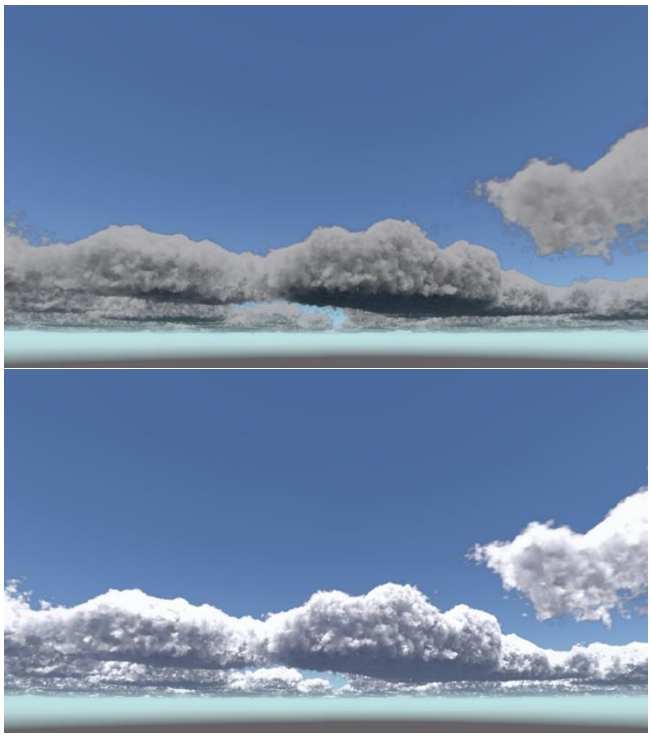

Figure 3. Results from single (top) and multiple scattering (bottom).

The top images of Figure 2 are the cloud coverage and type textures mapped onto the sphere representing the lower atmosphere. The bottom left image shows the computed dimensional profile. Notice where regions without coverage, the darker regions in the top left image, are absent of cloud density. The bottom right image is the result of our cloud modeling steps after the noise erosion is applied to the dimensional profile. Without illumination, the interior region of the cloud remains uniformly white.

The top image of Figure 3 is the result of approximating Equation 1 based on single scattering, or, approximating the L_s term using Equation 5 with $N = 0$. This revealed some details of the cloud structure, such as shadows and highlighted interior areas. The dark-grey, dull, and smoke-like appearance is the result of not accounting for scatterings from multiple inter-reflections. The bottom image is the result of approximating the same L_s term with $N = 8$, which resulted in significant brightening and increased realism of the clouds.

Figure 4 illustrates the subtle yet important Phase function. The top two images are rendered with $P(v, l) = \frac{1}{4\pi}$ in Equation 5, or the isometric Phase function in which light scatters equally in all directions, while the bottom images are rendered with the Phase function proposed by Jendersie & d'Eon [5] with the diameter of water particles defined to be $20\mu m$. In the bottom row, notice that the edges of clouds appear brighter as a result of increased forward scattering. These bright edges, or "silver lining", are more prominent in areas that are close to the direction from the Sun towards the camera. The right images show that this effect can be dramatized with appropriately adjusted color for the Sun.

While the results presented thus far all include our Unified TAA, Figure 5 illustrate a selected region of the image focusing specifically on the improvements. The top left image in Figure 5 is rendered with one uniform pixel and volumetric sample. The top-right image is the result of introducing our blue noise randomization into the pixel area sampling and volumetric interval sampling. The bottom left image replaced the single volumetric sample with interval integration approximated by Equation 8 resulting in a much more improved transmittance estimation. Lastly, the bottom right image introduces TAA of 8 frames. In this case, each volumetric interval is subdivided into fixed-length segments

Figure 4. Top row without and bottom row with Phase function.

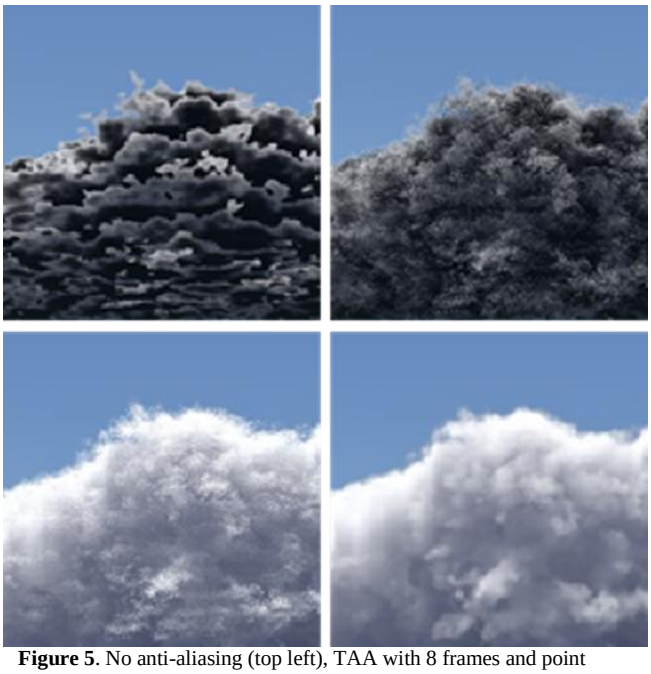

Figure 5. No anti-aliasing (top left), TAA with 8 frames and point sampled transmittance (top right), TAA and improved volumetric integration (bottom left), and Unified TAA (bottom right),

in coordination with pixel sampling rate and each TAA-frame samples an appropriate subset of volumetric segments.

VI. Conclusion

This paper presents a system for real-time atmospheric cloud generation and rendering based on the modeling of cloud density as proposed by Schneider [1] and the volumetric rendering framework as described Fong [3]. The rendering system is generalized to integrate recent advancements including the approximation of multiple scattering [4], a Phase function that models Mie scattering [5], and a volumetric integration which is based on an improved energy-conserving formulation [2]. Additionally, a novel TAA strategy is detailed that unifies the control of sampling over pixel areas and volumetric intervals. As a simple scene of the Unity3D game engine, this system serves to exemplify an integration of the important but isolated results, and the publicly accessible source code is a demonstration of straightforward GPU based parallelization.

The resulting system is capable of rendering cloudscapes in real-time, achieving frame rates of between 6 and 7 milliseconds on a typical laptop with reasonable quality settings. The shapes of clouds can be customized by user-defined texture maps and the rendering can be tailored through a readily accessible user interface, allowing for adjustments of effects and quality.

Currently, the system does not support cloud interaction with other objects in the scene. Additionally, the solution does not account for scenarios where the camera is positioned within the cloud and cannot represent altitude gaps between cloud layers. Lastly, serving as a system for investigating cloud modeling and rendering techniques, a clear interface should be defined to decouple the two modules. Based on this system, future work could derive texture generation approaches to support the generation of cloud maps that are based on the physics of cloud formation rather than observation. For example, approaches that integrate the excellent improvements from Schneider, [2] or, take into considerations the time of day, temperature, humidity, wind conditions, and terrain.

References

[1] 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.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[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] 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.

[10] 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.

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

[12] “A Guide to Creating Realistic Clouds in Houdini.” Accessed: Jun. 14, 2024. [Online]. Available: https://80.lv/articles/a-guide-in-creating-realistic-clouds-on-houdini/

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

[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] 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.

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

[17] 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.

[18] P. Ford, “Real-Time Rendering of Atmospheric Clouds,” M.S., University of Washington, United States -- Washington, 2024. [Online]. Available: https://www.proquest.com/dissertations-theses/real-time-rendering-atmospheric-clouds/docview/3081565707/se-2?accountid=14784

[19] 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.

[2] https://www.schneidervfx.com