Real-time volumetric cloud rendering for games and simulations

Mark Loboda, Ciril Bohak, Matija Marolt, Žiga Lesar

University of Ljubljana, Faculty of Computer and Information Science
E-pošta: ml7363@student.uni-lj.si, {ciril.bohak, matija.marolt, ziga.lesar}@fri.uni-lj.si

Abstract

We present a real-time volumetric cloud rendering system combining procedural 3D noise, a weather map for large-scale control, and a physically-based lighting model. Our system supports dynamic parameter tuning, simulates wind through texture scrolling, and maintains high visual fidelity at interactive frame rates. It is implemented using WebGPU API, it is scalable, configurable, and capable of rendering world-scale cloudscapes efficiently.

1 Introduction

Rendering visually pleasing clouds in video games is a significant challenge in computer graphics. The challenge is typically tackled with semi-transparent billboards [4], resulting in a flat appearance, especially when moving the camera through the scene. Recently, GPUs have become powerful enough to simulate basic volumetric light interactions in real time [1], unlocking the potential for rendering much more realistic and immersive cloudscapes. Although general volume rendering is not yet feasible in real time, it is possible to take advantage of the characteristics specific to clouds – a wide and shallow volume with highly random shapes and a high albedo.

In this paper, we present a real-time volumetric cloud renderer based on ray marching and procedural noise techniques. It generates dynamic, three-dimensional cloudscapes by evaluating cloud density and lighting at various points in space. The system is highly configurable, enabling users to create custom cloud formations, simulate realistic atmospheric lighting effects, and interactively tune rendering parameters in real time. The novelty of our approach lies in enabling the low-cost rendering of large, dynamic cloudscapes with configurable types and densities, from rain clouds to barely visible formations, controlled through input parameters. While not yet capable of rendering cloudscapes at a global scale, the system was developed with scalability in mind to support such use cases in the future. The implementation has been made publicly available.[1]

2 Related work

One of the earliest examples of volumetric cloud rendering was introduced by Kajiya and Von Herzen [5], who represented clouds with densities within a volume grid. They developed light scattering equations and introduced a diffusion approximation to the radiative transfer problem suitable for computer graphics applications. However, real-time rendering of volumetric clouds remained computationally intractable until the advent of GPU acceleration.

Harris and Lastra [4] developed an approximation of multiple forward scattering and first-order anisotropic scattering, precomputing these effects for a set of impostors that were then rendered in real time using alpha compositing. Bouthors et al. [2] presented a radiosity-based approach and modeled the clouds with procedural noise. Schpok et al. [10] use procedural noise with user-tunable parameters and render the clouds with a slice-based volume renderer. The mentioned techniques were more recently combined by Guerilla Games in their game Horizon: Zero Dawn [1]. Their work heavily inspired our implementation.

A more modern approach by Webanck et al. [12] involves procedural generation of different cloud types, such as stratus and cumulus, and their animation based on key frames and the characteristics of the terrain. More recently, Kallweit et al. [6] explored the use of deep neural networks to approximate higher-order scattering terms in the radiative transfer equation. Their approach produces the most realistic and accurate results, but is designed for offline rendering to accelerate cloud preview generation in film production workflows.

For a more comprehensive review of the field of cloud modeling, rendering, and simulation, we refer the reader to the survey by Goswami [3].

3 Implementation

The volumetric cloud renderer is implemented in C++ using the WebGPU API with the wgpu-native backend. The user interface is implemented using ImGui[2], enabling real-time parameter tuning and the window for rendering settings and performance statistics. GLFW[3] handles window creation and user inputs, while GLM[4] is used for vector and matrix mathematics throughout the rendering pipeline.

[1]`https://github.com/markloboda/SandboxEngine`

[2]`https://github.com/ocornut/imgui`
[3]`https://www.glfw.org`
[4]`https://github.com/g-truc/glm`

3.1 Ray marching and light scattering

The core technique used for volumetric cloud rendering in our method is ray marching [8], which simulates the interaction of light with clouds by sampling density and accumulating radiance along a light ray. For each pixel, a ray is cast from the camera into the scene. The ray is marched through the volume, accumulating radiance while reducing transmittance. Radiance is split into in-scattered radiance L_s and ambient radiance L_a:

$$L_s = L_{\text{sun}} T f_p, \tag{1}$$

$$L_a = L_{\text{ambient}}(1 - e^{-\rho}). \tag{2}$$

L_s is evaluated by ray marching secondary light rays toward an external light source (the sun) with user-defined radiance L_{sun} and taking into account the phase function f_p. We use a single-scattering model with a Henyey-Greenstein phase function [11] with positive anisotropy, which is typical of atmospheric particles like water droplets present in clouds. L_a is calculated as a user-defined ambient radiance L_{ambient} scaled by the local scattering probability $1 - e^{-\rho}$, which depends on the cloud density ρ. This is a simplified, non-physical model that offers visually convincing results suitable for real-time applications.

At each step k of length Δs along a light ray, we sample the cloud density ρ, and update radiance L and transmittance T:

$$L_k = L_{k-1} + (L_a + L_s) T_k \Delta s, \tag{3}$$

$$T_k = T_{k-1} e^{-\alpha \rho \Delta s}, \tag{4}$$

starting with $L_0 = 0$ and $T_0 = 1$. Here, α is a user-defined light absorption coefficient that controls the attenuation of light as it travels through the cloud medium. The process stops when the light ray exits the bounding box of the cloud or reaches a transmittance value of 0.

3.2 Ray marching optimizations

The application supports dynamic step size during ray marching to optimize performance. Ray marching starts with long steps to quickly traverse empty space. Once a non-zero density is detected, the algorithm takes a step back and switches to a smaller step size to accurately capture cloud details. If several consecutive small-step samples return zero density, the renderer switches back to long steps to continue traversal efficiently. Since clouds are typically wide and shallow, the step size can be adjusted further based on the angle of the view ray, as shorter steps are required horizontally than vertically to maintain visual detail. Furthermore, the renderer defines a cloud layer with user-configurable minimum and maximum heights. Ray marching is limited to the segment of the view ray that intersects this cloud layer, effectively skipping large empty areas above and below the clouds.

Another optimization involves using two types of density samples: a computationally cheap sample and a more expensive, detailed sample. The renderer first evaluates the cheap sample, and only if it returns a non-zero value does it compute and add the expensive sample contribution. A preliminary low-cost test avoids expensive sampling in empty or sparse regions, improving overall rendering efficiency.

3.3 Modeling clouds with procedural noise

To represent the complex structure of volumetric clouds, the system combines several types of procedural noise through a set of texture inputs. The approach uses one 2D texture and two 3D textures to describe large-scale weather patterns and small-scale cloud details efficiently.

The 2D weather map shown in Figure 1 defines the overall cloud coverage over a given area, enables efficient representation of large-scale atmospheric patterns, and can be modified at runtime to simulate dynamic weather when driven by an external system. The red channel stores the base cloud coverage between values of 0 (no cloud coverage) and 1 (full cloud coverage), and the blue channel encodes the cloud type. The cloud type is a normalized value between 0 and 1, corresponding to different cloud types: 0 for stratus, 0.5 for stratocumulus, and 1 for cumulus. This type of value influences the vertical shape of the cloud by selecting a height gradient used in density evaluation.

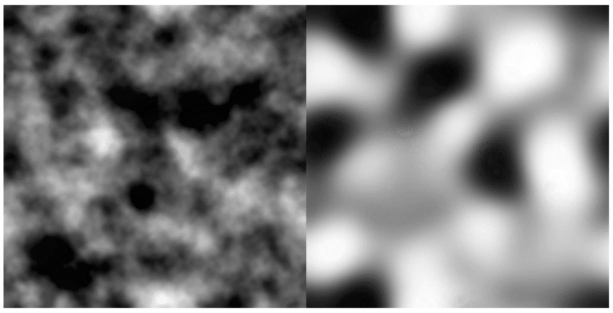


Figure 1: Noise texture used as a weather map. The red channel (left) stores the base cloud coverage, and the blue channel (right) encodes the cloud type.

A large 3D texture (128×128×128) shown in Figure 2 provides the base noise data. The red channel contains a combined Perlin-Worley noise (Perlin [9] multiplied by inverse Worley [13]), which produces large, organic structures. The remaining three channels store low-frequency inverse Worley noises at increasing frequencies. The low-frequency inverse Worley noises are combined using a weighted dot product to form a fractional Brownian motion (fBm) [7]. This value is then used to dynamically remap the Perlin-Worley noise, effectively raising the density threshold based on the fBm value and allowing the added cloud detail only where large-scale cloud structure supports it.

Edge detail and erosion are added using a second, lower resolution 3D texture (32×32×32) shown in Figure 3, which contains high-frequency inverse Worley noise at increasing frequencies. This texture is used to erode the edges of the clouds, giving them a soft and natural appearance without requiring high-resolution volume textures.

The noise patterns are further modulated by a linear height-based gradient that increases from the bottom to the top of the cloud layer. This layered approach enables the

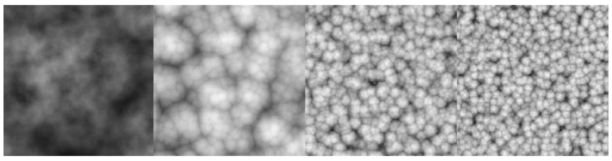


Figure 2: Large 3D texture for base cloud noise. The red channel (left) contains a Perlin-Worley noise, and the remaining three channels (right) contain an inverse Worley noise at increasing frequencies.

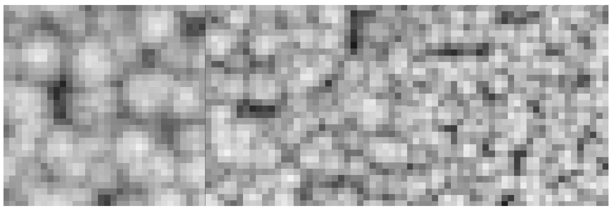


Figure 3: Small 3D texture for edge detail and erosion. Three channels (from left to right) contain inverse Worley noise at increasing frequencies.

rendering of vast, visually rich and realistic cloud shapes while minimizing memory usage and computational cost.

To simulate wind, we scroll both 3D noise textures over time in the horizontal plane. This creates the illusion of dynamic moving cloud formations without altering their positions as defined by the weather map. Since the weather map determines static cloud coverage and type, only the internal structure and shading of the clouds evolve with the wind. This technique allows the clouds to exhibit dynamic motion and evolving appearance while maintaining consistent large-scale patterns and positions, ensuring coherence with the broader weather simulation.

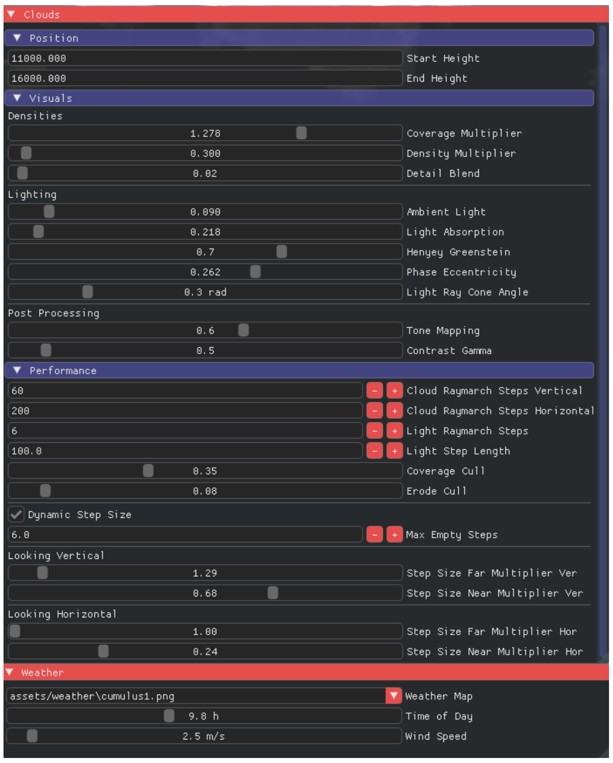
3.4 Configuration

The application provides a wide range of configurable parameters through a graphical user interface, shown in Figure 4. These settings allow users to explore different cloud formations, lighting conditions, and performance characteristics interactively. In a given application, several configuration presets might be created to allow balancing between visual quality and rendering performance.

3.5 Scalability considerations

The design of the system is centered around scalability, both in terms of data input and rendering flexibility. Cloud coverage and cloud type are defined in a tileable 2D texture called the cloud map, while the actual cloud shapes are procedurally generated using 3D noise textures. This separation allows the system to remain lightweight and flexible, while still supporting real-time parameter adjustments. The long-term vision is to implement a tiling mechanism that can dynamically stream weather data such as cloud coverage and type—from third-party providers. Combined with localized parameter tuning, this would enable simulation and visualization of atmospheric phenomena at a global scale.

Another key scalability feature is the system's high degree of tunability. Users can adjust a wide range of parameters affecting visuals, lighting, and performance, allowing the system to adapt to different hardware constraints and visual fidelity requirements. This makes it suitable for everything from small-scale to large-scale real-time applications and simulations.



```
▼ Clouds
▼ Position
11000.000                         Start Height
16000.000                         End Height
▼ Visuals
Densities
1.278                             Coverage Multiplier
0.300                             Density Multiplier
0.02                              Detail Blend
Lighting
0.090                             Ambient Light
0.218                             Light Absorption
0.7                               Henyey Greenstein
0.262                             Phase Eccentricity
0.3 rad                           Light Ray Cone Angle
Post Processing
0.6                               Tone Mapping
0.5                               Contrast Gamma
▼ Performance
60                          - +   Cloud Raymarch Steps Vertical
200                         - +   Cloud Raymarch Steps Horizontal
6                           - +   Light Raymarch Steps
100.0                       - +   Light Step Length
0.35                              Coverage Cull
0.08                              Erode Cull
[✓] Dynamic Step Size
6.0                         - +   Max Empty Steps
Looking Vertical
1.29                              Step Size Far Multiplier Ver
0.68                              Step Size Near Multiplier Ver
Looking Horizontal
1.00                              Step Size Far Multiplier Hor
0.24                              Step Size Near Multiplier Hor
▼ Weather
assets/weather\cumulus1.png   ▼   Weather Map
9.8 h                             Time of Day
2.5 m/s                           Wind Speed
```

Figure 4: The editor and the parameters for configuring the renderer.

4 Results

Example images shown in Figure 5 demonstrate the system's ability to produce visually rich and varied volumetric cloudscapes. The examples highlight different cloud types, lighting conditions, and configurations, showcasing the flexibility of the system.

We tested the cloud rendering system on a commodity desktop computer with an AMD Ryzen 5 7600X CPU and AMD Radeon RX 7900 XT GPU with 20 GB VRAM, running Windows 11. The implementation runs in real time, though closely tied to the required visual fidelity. The image in Figure 6 demonstrates a highly optimized render achieving a frame time of 2.6 ms at 1440p resolution. However, the same settings may produce noticeable aliasing when the camera is inside the cloud volume. While some of these issues may be addressed with parameter tuning, there remains room for improvement in close-up scenarios. Note that all cloud examples were rendered using a cloud layer thickness of 15 km. Reducing this height could yield major performance gains by enabling smaller and fewer ray marching steps. Combining this with a static skybox layer could allow high visual quality at an even lower computational cost.

Our approach cannot render two clouds vertically layered at the same position, and ray marching, though real-

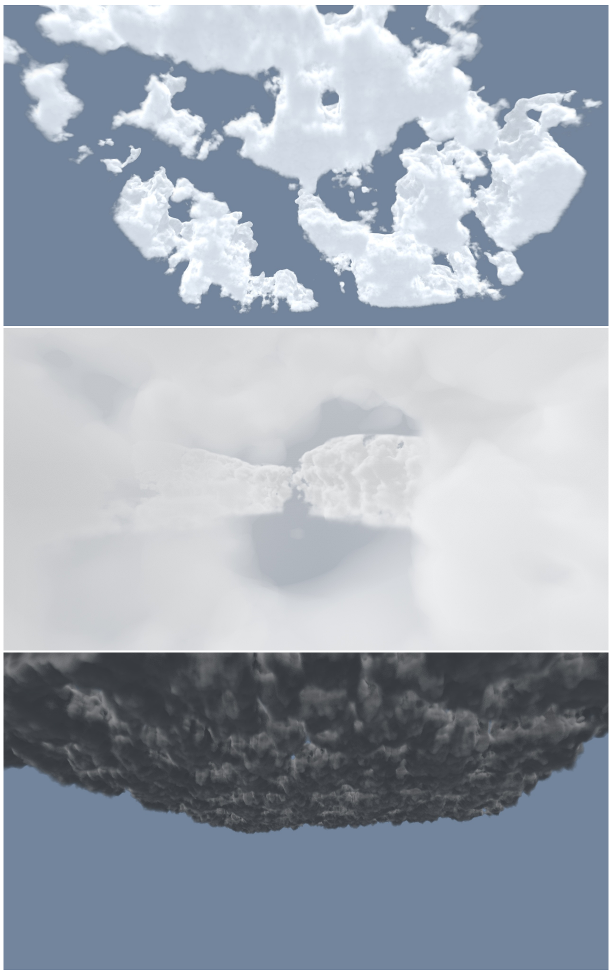


Figure 5: Three example renderings: clouds in a sunny setting with low absorption and high ambient light (top), rendering from inside the clouds (middle), and rain clouds with high absorption and low ambient light (bottom). The other parameters are similar to those in Figure 4.

time, is more costly than other approaches and varies with cloud conditions.

5 Conclusion

The paper presents the design, implementation, and results of a physically-based volumetric cloud rendering system. By leveraging a combination of procedural 3D noise, weather maps, and dynamic rendering parameters, the system is capable of producing convincing and diverse cloudscapes in real time. The separation between global weather patterns and local detail offers scalability and flexibility, with the potential to integrate real-world meteorological data. The renderer can be tuned for various use cases, from cinematic visuals to lightweight interactive scenes.

The resulting cloud visuals are rich and responsive, and the performance is competitive even at high resolutions. Nevertheless, several areas for future improvement have been identified, including enhancing visual quality when flying through clouds, better aliasing control, further optimization of ray marching, and deeper integration of dynamic weather behavior beyond static maps and wind

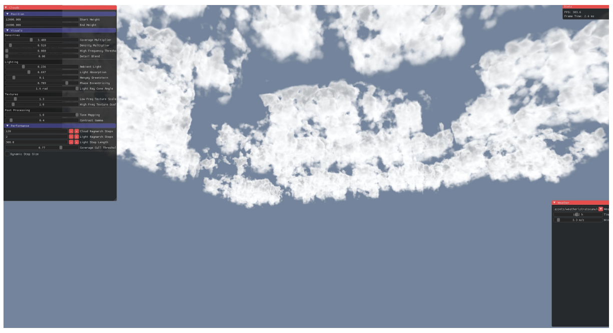


Figure 6: Cloud rendering optimized for performance with frame time of 2.6 ms.

scrolling. These directions offer promising opportunities for extending the system into a more complete and physically faithful atmospheric simulation.

References

[1] N. Vos A. Schneider. The real-time volumetric cloudscapes of horizon: Zero dawn. In *ACM SIGGRAPH 2015: Advances in Real-Time Rendering Course*, 2015. Course presentation.

[2] A. Bouthors, F. Neyret, and S. Lefebvre. Real-time realistic illumination and shading of stratiform clouds. In *Eurographics Workshop on Natural Phenomena*, 2006.

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

[4] M. J. Harris and A. Lastra. Real-time cloud rendering. *Computer Graphics Forum*, 20:76–85, 9 2001.

[5] J. T. Kajiya and B. P. Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 18(3):165–174, 1984.

[6] S. Kallweit, T. Müller, B. Mcwilliams, M. Gross, and J. Novák. Deep scattering: Rendering atmospheric clouds with radiance-predicting neural networks. *ACM Transactions on Graphics*, 36:1–11, 12 2017.

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

[8] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1:99–108, 6 1995.

[9] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

[10] J. Schpok, J. Simons, D. S. Ebert, and C. Hansen. A real-time cloud modeling, rendering, and animation system. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA ’03, page 160–166, Goslar, DEU, 2003. Eurographics Association.

[11] D. Toublanc. Henyey–greenstein and mie phase functions in monte carlo radiative transfer computations. *Applied Optics*, 35:3270, 6 1996.

[12] A. Webanck, Y. Cortial, E. Guérin, and E. Galin. Procedural cloudscapes. *Computer Graphics Forum*, 37:431–442, 5 2018.

[13] S. Worley. A cellular texture basis function. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 291–294, 1996.