Using Volumetric Rendering to Generate More Photo-realistic Lighting for Video Game Environments

Chia-Yu Tung, Vinny Peng, Qingzhi You, Zhengyang Wang
Department of Computer Science, University of Southern California
ctung@usc.edu, yuzhepen@usc.edu, qingzhiy@usc.edu, zwang497@usc.edu

Abstract

Growing popularity of massive open-world games nowadays demand better and more realistic graphics to deliver immersive player experiences. Many computer graphics algorithms have been aiming to reach photo-realistic lighting effects for different terrain and environments in video games. In this paper, we present a widely used rendering algorithm along with various phase functions, emphasizing the significance of scattering. Our objective is to simulate volumentric lighting effect in diverse scenes, thereby enhancing overall lighting effects.

1 Introduction

Video games offer a captivating escape for players globally, continually pushing the boundaries of realism with each passing year. Among the myriad elements that contribute to this immersive experience, lighting stands out as a pivotal factor. In our pursuit of elevating player engagement, we hope to create photorealistic lighting effects simulating the natural lights in the real world to provide immersive digital scenes. Specifically, we aim to simulate the Tyndall effect, a phenomenon observed in the real world, within video game environments. Our objective is to craft photorealistic cave settings that authentically replicate natural lighting conditions.

To achieve this goal, our methodology revolves around utilizing the Unity3D editor as our foundational platform. We will meticulously set up scenes and implement a volumetric lighting renderer, thereby laying the groundwork for a transformative gaming experience where realism is not just a visual aesthetic but an integral part of the player's journey.

2 Related work

2.1 Raymarching

Volumetric lighting is a well know topic in the current academic world. Most of the previous work used Raymarching (Tóth and Umenhoffer, 2009) to perform illumination accumulation. Raymarching is widely used in rendering 3D volumetric effects such as light, cloud, fog and smoke. It works by casting rays from a camera through each pixel on the screen, and marching along them in small steps until they hit a surface or reach a maximum distance. The color and shading of the pixel are then computed based on the surface normal and the lighting conditions. It can produce soft and realistic shadows and lighting effects by taking into account the scattering and absorption of light by volume, creating photo-realistic lighting and scenes.

2.2 Phase function in light rendering

In the context of light rendering, many previous works have introduced a phase function (Greg Humphreys, 2023) to represent the interaction between light and materials. Essentially, a phase function describes how light scatters, similar to the way the Bidirectional Scattering Distribution Function (BSDF) models this interaction (F O Bartell, 1981).

Presently, there exist various phase functions. A fundamental example is isotropic scattering, as shown in Equation (1). For anisotropic scattering, such as Mie scattering, a widely used phase function is given by Equation (2), proposed by Henyey and Greenstein (Henyey, 1941). Furthermore, to address the limitations of back scattering inherent in the Henyey and Greenstein model, a Double Henyey–Greenstein (DHG) phase function was proposed (Feng Zhang a, 2016)."

$$P_{\text{isotropic}}(\theta) = \frac{1}{4\pi} \tag{1}$$

$$P_{\text{HG}}(\theta) = \frac{1}{4\pi}\left(\frac{1-g^2}{1+g^2+2g\cos(\theta)^{\frac{3}{2}}}\right) \tag{2}$$

3 Scene setup

To fulfill our objectives and explore the Tyndall Effect within our gaming environment, a meticulous scene setup is imperative. Our vision encompasses the creation of a diverse landscape, featuring elements like mountains and caves punctuated with openings at their summits.

To initiate this process, we have chosen to leverage the Terrain tool integrated into the Unity3D platform. This tool offers a traightforward approach to constructing terrains, utilizing Perlin Noise(Perlin, 2002) as a foundational building block. Figure 1 illustrates what we initially generated.

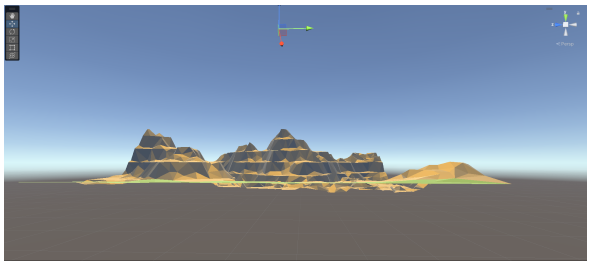


Figure 1: Mountain by Terrain Building Tool

While the initial mountain model serves as a starting point, certain issues have surfaced. Primarily, its overly rugged appearance falls short of our aspiration for a more realistic model. Additionally, the limitations of the Terrain build tool became evident as it proves incapable of generating caves with openings due to its reliance on a basic Perlin Noise(Perlin, 2002). To address these shortcomings and enhance our scene, we recognized the necessity of incorporating additional models.

Given our limited expertise in modeling, we have opted to source high-quality models from the Internet. In this endeavor, we found an impressive snow mountain model (Show in Figure 2), ideal for a comprehensive showcase, through the Unity Asset Store.

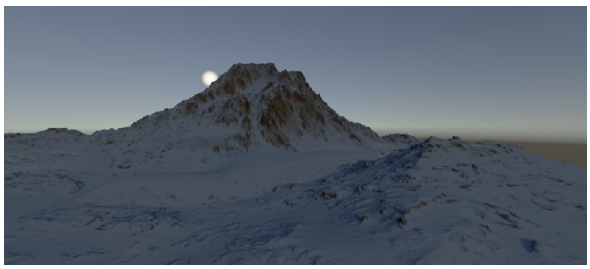
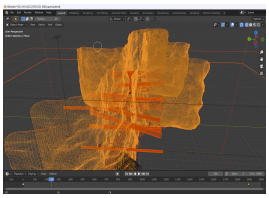


Figure 2: Snow Mountain Model

Furthermore, we obtained an artificial cave model from ArtStation. Employing Blender, we undertook some modifications to tailor it to our needs, resulting in the realization of our envisioned design (shown in Figure 3).



(a) Artificial Cave Model in Blender

(b) Modified model in Unity3D

Figure 3: Comparison between Original and Modified Models

4 Methodology

4.1 Volumetric rendering in raymarching

We chose the raymarching algorithm as our approach due to several considerable advantages. During our initial research, we found this method has been proven effective when rendering arbitrary shapes and volumes. More importantly, it can be highly customizable and expandable as it divides the light into small steps. Once we solidify the raymarching code as the skeleton, we can easily adjust the number of samples and the exact computation at each sample point to manipulate the light and optimize the outcome. It is comparatively easier to locate relative resources and information we need to implement and integrate with Unity shaderlab and HLSL, allowing us to save time as well as have a more streamlined development process.

We start the raymarching process by taking the coordinates of the starting points, and use the coordinate of the camera as ending points, to calculate ray direction and length between them. Then we divide the ray into an arbitrary number of steps and iterate through them in a loop. At each step, we checked if the ray hits the surface using Unity's shadow map and compute the light attenuation. After that, we computed the scattering of the light by applying the phase functions we implemented later. Finally, we summed up the light value and applied final fixes such as light color and opacity to get the final color value.

Additionally, we decided to add a height fog option to account for atmospheric perturbations. The general usage for a height fog is that it makes objects farther away from the ground more scattered and more blurry, assuming the viewer is on the

Earth's surface. Objects higher up are typically more affected than objects closer to the ground. To achieve this naturally occurring phenomenon, we needed to simulate the fog's density based on vertical position of each point of light in the scene. We calculated the desired fog density through adjusting the color of the volumetric medium based on the vertical position of the points being sampled along the ray. We modified the original Raymarching process by adjusting the properties of the participating media based on the height of the current position along the ray.

```
while distance(currentPosition) is less
    then maxDistance:
# Calculate the height-based fog
    density fogDensity =
    calculateDensity(currentPosition)
# Modify step size based on fog density
    or adjust media properties
    currentPosition += stepSize *
    direction * fogDensity
```

In the calculateDensity function, we modified the fog density based on the height of the current position along the ray. We added a simple exponential function, where higher altitudes result in denser fog. The fog density can then be used to modulate the step size, affecting how quickly the ray advances.

4.2 Phase function

In our pursuit of simulating realistic light interactions in virtual environments, we explored two fundamental types of light scattering: isotropic scattering and anisotropic scattering. To achieve a comprehensive comparison within the same scene, we implemented distinct phase functions for isotropic scattering by isotropic phase function (Greg Humphreys, 2023) and the Rayleigh phase function (American Meteorological Society).

Within the realm of anisotropic scattering, commonly referred to as Mie scattering in the real world, we introduced the widely used phase function: Henyey and Greenstein (Henyey, 1941) to simulate Mie scattering. Additionally, we considered the enhanced capabilities of the Double Henyey and Greenstein (DHG) phase function, as well as the computationally efficient Schlick phase function (Patapom, 2013) , serving as a more efficient alternative to the Henyey and Greenstein phase function. We also followed the concept of DHG phase function to implement a modified version (Equation (3)).

$$\frac{1}{4\pi}\left(\frac{1-g_{\text{forward}}^2}{1+g_{\text{forward}}^2-2g_{\text{forward}}\cos(\theta)} + \frac{1-g_{\text{backward}}^2}{1+g_{\text{backward}}^2-2g_{\text{backward}}\cos(\theta)}\right) \tag{3}$$

The equation is controlled by the asymmetry parameter g, which has a range of -1, 1. This parameter signifies the direction of light scattering, with positive values indicating forward scattering and negative values indicating backward scattering. Encoding g g as an adjustable parameter enables fine-tuning of the Mie scattering behavior in our simulations, allowing us to observe the shift in light scattering effects as g varies.

5 Experiment result

5.1 Cave Lighting

In order to demonstrate how Raymarching and different phase functions affect a more enclosed setting, we created a tunnels or caves and integrated the volumetric renderer. As mentioned in the scene setup section of this paper, we found an online asset of an artificial tunnel cave with a large vertical opening above and a natural cave with a small circular opening. The results of our project are shown in the pictures below.

(a) No Volumetric Lighting

(b) Directional & Spot Light

Figure 4: Difference between normal directional light & directional + spot light with volumetric rendering applied

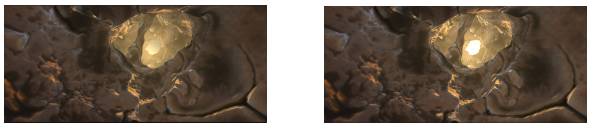
As shown above in Figure 4, the scene depicts a man-made underground tunnel with a long opening above. The image 4a to the left only shows a default Unity directional light applied to the cave, resulting in the entire interior of this artificial cave to be extremely dim and lifeless. To show the power of volumetric lighting, we then added Raymarching and Henyey-Greenstein phase function to the spotlight and direction light. With volumetric effects added, image 4b to the right demonstrates a "God Ray" lighting from above, scattering across the interior of the cave.

Similarly, the lighting effects of Henyey-Greenstein combined with Raymarching also does

(a) No Volumetric Lighting (b) Volumetric Lighting

Figure 5: Comparison of with and without volumetric lighting in a natural cave

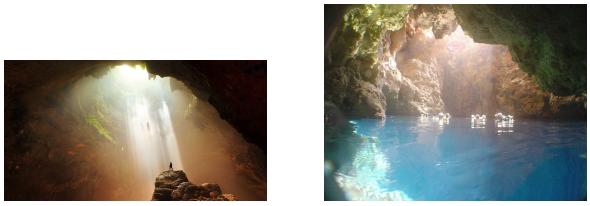
a very good job of creating realistic lighting effects in more enclosed, natural caves. As shown above in Figure 5, the image 5a depicts a cave with a very small hole at the top of the cave, allowing light to enter. The entire cave is very dim and does not resemble an actual cave with sunlight hitting directly from above. However, after applying Henyey-Greenstein volumetric rendering effects, the picture on the right illustrates a cave full of light, radiance, and energy, mimicking the sunlit effect of a very sunny Monday morning.



(a) No Height Fog (b) Height Fog Applied

Figure 6: Different between volumetric lighting with and without high fog

Additionally, we implemented height fog for Raymarching to demonstrate the atmospheric effects where objects away from the ground appear to be more blurry. Figure 6 above clearly illustrates the difference. The image 6a shows the lighting of the cave with no height fog, and the image 6b depicts a cave with height fog. The height fog contributes to the overall realism of the cave. It adds depth to the rendering, making distant objects appear more subdued and creating a realistic sense of scale and distance.



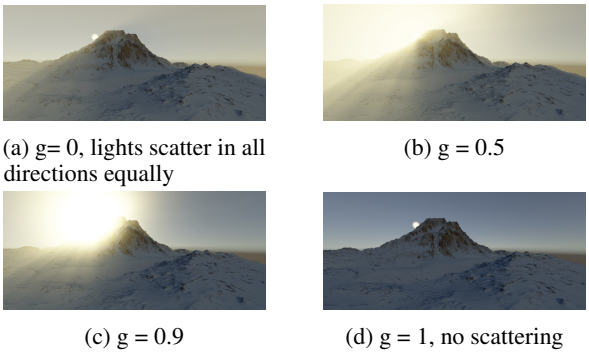
(a) Real Life Cave in Mexico (b) River Cave in Spain

Figure 7: Light Scattering Effects in Caves in Real Life

Since there is no way of quantifying the results of our project, we can only verify our work with real world pictures to see if we achieved our desire outcome. In Figure 7 above, there is a real cave in Mexico and a cave with a river in Slovenia. Both pictures have holes right above the cave, with light shinning through. Our results compared to the real life lighting seems to be quite accurate. Looking at these pictures and our rendering side by side, we can conclude that we did a fairly good job at simulating these real life lighting effects.

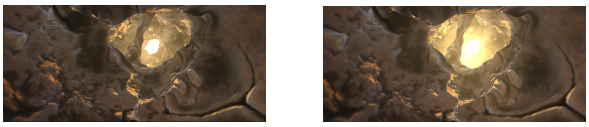
5.2 The *asymmetry parameter* g

As we mentioned above, the asymmetry parameter g is the independent variable in the phase function, and the light scattering distribution varies as g changes. As a result, we kept all other parameters the same and adjusted the g values to show its influence. The following images illustrate the different light scattering result computed with different g values using the Henyey and Greenstein phase function.



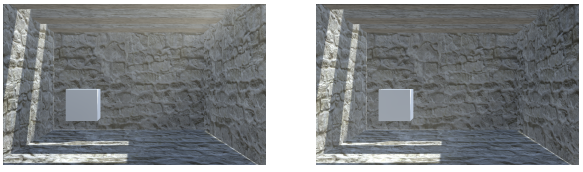
(a) g= 0, lights scatter in all directions equally (b) g = 0.5

(c) g = 0.9 (d) g = 1, no scattering



(a) Volumetric Lighting w $g = 0.5$ (b) Volumetric Lighting w $g = 0.7$

Figure 9: Different between $g_1 = 0.5, g_2 = 0.7$



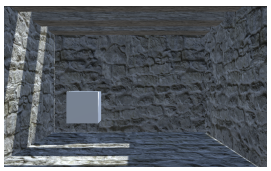
(a) Henyey and Greenstein. $g = 0.61$ (b) Henyey and Greenstein. $g = -0.33$

Figure 10: Different g of Henyey and Greenstein

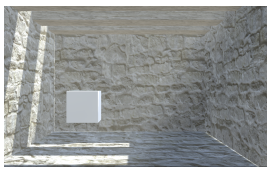
5.3 Different lighting indoors

In addition to cave environments and open snow-peaked mountains, we also want to showcase the power of volumetric lighting in different settings such as indoors. We simulated the illumination of an indoor environment with a directional light source originating from the right by applying various phase functions. In Figure 11, the isotropic scattering reveals a uniform distribution of light across the image.

In Figure 10, the inversion of light scattering direction is clearly observable due to the variation in the asymmetry parameter (g) within the Henyey and Greenstein phase function. Conversely, when examining Figure 12, which utilizes the same g factor as the Henyey and Greenstein phase function, a similar effect is observed in the scene. Figure 13 depicts Rayleigh scattering, exhibiting a similar effect to isotropic scattering. In the final comparison (Figure 14), when evaluating the Double Henyey–Greenstein (DHG) phase function and our modified version with identical parameters, it became evident that our modification produces a brighter scattering effect from the right top to the left bottom compared to DHG.



(a) Original Scene



(b) Isotropic phase function

Figure 11: Compare between Original and isotropic scattering

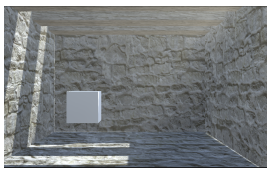


Figure 12: Schlick phase function.
$g = 0.61$

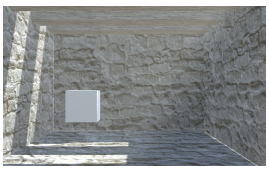
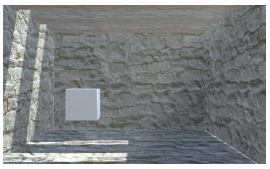


Figure 13: Rayleigh phase function

6 Conclusion

In this paper, we have explored an implementation of volumetric lighting using raymarching, which implicitly samples light scattering at each step along rays. We experimented with different phase functions for scattering light inside volumes mathematically. The project demonstrated the ability to achieve effects like aerial perspective, light shafts, and scattering through particle densities – selling the illusion of light passing through a medium in various environments. In conclusion, volumetric lighting opens up new creative possibilities for aesthetically pleasing and photorealistic scenes.



(a) DHG

(b) Modified version

Figure 14: Difference between DHG and our modified version with $g_1 = 0.49, g_2 = -0.38$

In the future, our project could continue expand to enhance the flexibility and performance of the volumetric light in several ways. For instance, instead of using a single light color, we could introduce some 3D texture to sample from and add more dynamic and versatile light colors and patterns. To accelerate rendering, an optimization could employ selective raymarching, only evaluating specific rays likely to intersect volumes based on heuristics.

References

AMS American Meteorological Society. Rayleigh phase function.

W. L Wolfe F O Bartell, E. L. Dereniak. 1981. The theory and measurement of bidirectional reflectance distribution function (brdf) and bidirectional transmittance distribution function (btdf).

Jiangnan Li Feng Zhang a, b. 2016. A note on double henyey–greenstein phase function.

Matt Pharr Greg Humphreys. 2023. *Physically Based Rendering: From Theory To Implementation*.

J. L. Henyey, L. G. Greenstein. 1941. Diffuse radiation in the galaxy. *Astrophysical Journal, vol. 93, p. 70-83 (1941).*

Patapom. 2013. Real-time volumetric rendering.

Ken Perlin. 2002. Improving noise. *ACM Trans. Graph.*, 21(3):681–682.

Balázs Tóth and Tamás Umenhoffer. 2009. Real-time volumetric lighting in participating media.