VOLUMETRIC RENDERING OF PARTICLE SYSTEMS USING INTERVAL SHADING

A dissertation submitted for the degree of MSc. Computer Games Technology

by

Sandilya Yasodhar Jandhyala

School of Design, Informatics and Business,
Abertay University.

August, 2025

Acknowledgements

Thanks to Erin Hughes and Ruth Falconer for their review of this document and their guidance throughout the production of this artefact.

Thanks to Jamie Buttenshaw for his guidance in the area of volumetric rendering methods and particularly the mathematical modelling involved.

Thanks to Thomas Howson for his review of the mathematical arguments presented.

Finally, we would like to thank Lucas Chatwin, Morgan Skillicorn, Maja Oszczypała, James Marsh, Timon Holzer and Aria Dzhumaliyski for their unwavering support.

“The lyf so short, the craft so long to lerne.”
Geoffrey Chaucer, The Parliament of Birds

Abstract

Participating media such as smoke, clouds and fog are common elements of modern real-time graphics applications such as games. Traditional methods of participating media rendering include ray-marching density fields or various forms of voxel-based integration. While these methods produce visually plausible results in many cases, they may suffer from aliasing artefacts or temporal ghosting when the media is animated or high-frequency lights or shadows are introduced.

This project introduces a novel volumetric particle rendering method based on a recently published technique that uses tetrahedrons as primitives for real-time volume rendering. This method was evaluated using a qualitative assessment of the visuals as well as a quantitative analysis of its performance against various parameters. The method was found to support most characteristics required for plausible rendering of participating media, and is capable of running comfortably at real-time frame rates. Furthermore, the media can be animated and supports high-frequency light animation and shadows without any temporal ghosting or other objectionable artefacts.

There are opportunities for further work to improve performance of this method, improve visual fidelity, incorporate into a unified volumetric rendering system and to extend this method beyond particles to render tetrahedral meshes.

Table of Contents

List of Figures

List of Equations

1 Introduction

1.1 Context

Participating media refers to volumes of particles that participate in light transport by absorbing or scattering light that passes through them (Pharr, Jakob and Humphreys, 2023). Common examples of participating media in graphics applications include steam, smoke, fog, clouds and even air and water. Since such examples of participating media are so commonly found in the real world, they form a crucial component of modern games and graphics applications.

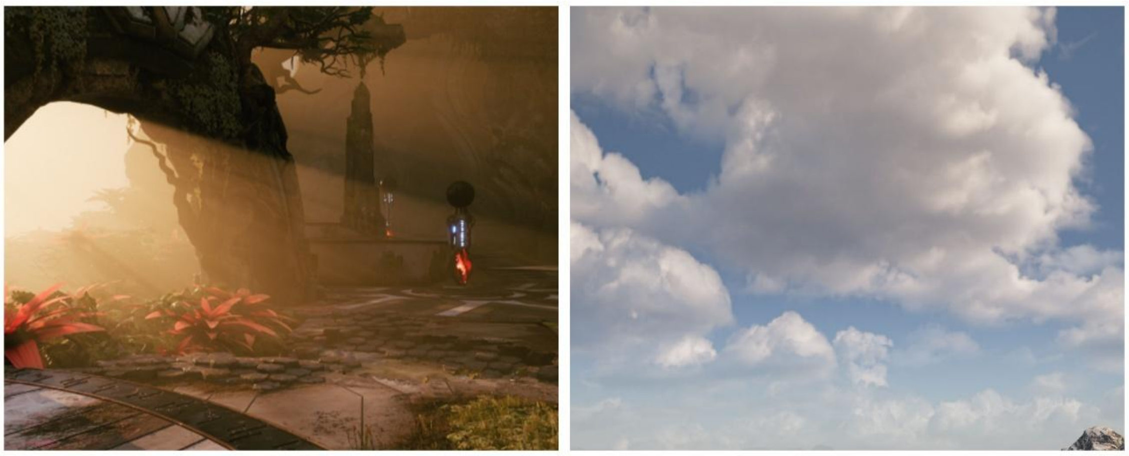


Figure 1: Examples of participating media in games. Left: Volumetric fog in the Unreal Engine (Epic Games, no date). Right: Clouds in the Decima Engine (Schneider, 2023)

A common technique used to render smoke and clouds is to use billboards, which are textures that are always rotated to face the camera. The billboards themselves are typically emitted as part of a particle system. This system is relatively cheap to render and easy to animate, but is difficult to render in a physically based manner.

To render participating media in a more physically based manner, they are often represented using voxels or density textures. The light contribution for a given pixel is integrated through ray-marching, which involves casting a ray from the camera through the surface, taking samples of the lighting contribution and summing them up. For accurate self-shadowing and in-scattering, a second ray-marching step towards the light is needed at each sampling point to compute the light arriving at the sample point from the light source. The medium itself is typically a density field, either modelled analytically or sampled from a 3D texture. These methods often result in visually complex scattering and shadowing phenomena but suffer from the drawbacks of being expensive to compute and inflexible to animate. This process can be computationally accelerated using froxels (Hillaire, 2015), but such methods may suffer from temporal ghosting issues under motion. Interval shading (Tricard, 2024) is a technique that takes advantage of the recently introduced mesh shading pipeline to represent volume data using tetrahedrons, which can be rendered through rasterization. These tetrahedrons could theoretically be used to model a system of volumetric particles. Using interval shading, the optical depth can be computed for a tetrahedron without the need to maintain a voxel grid or resort to expensive ray-marching techniques. Furthermore, since the participating medium is represented using particles rather than voxels, it would be easy to animate. However, Tricard only demonstrates order-independent volume rendering and a solution for physically based lighting of tetrahedron primitives is yet to be determined.

This project proposes and evaluates a method of lighting a particle system composed of these tetrahedron primitives. Each tetrahedron is composed of a participating medium. Since interval shading can only be applied to order-independent rendering, this project extends the method for order-dependent volume rendering using a fast GPU-accelerated sorting algorithm.

1.2 Research question

Can the interval shading technique be used for real-time physically based rendering of participating media represented as a particle system, while maintaining visual fidelity under motion?

1.3 Aim

To develop and evaluate a rasterization-based method for physically based rendering of participating media represented as a particle system, using the interval shading technique. The method should be performant enough to be suitable for real-time applications, and should maintain visual fidelity under animation of the medium, light or camera without objectionable artefacts. This method will be referred to as the Interval Shaded Volumetrics method, or ISV method.

2 Background

2.1 Physically based light scattering theory

This section introduces the elements of modern physically based light scattering theory, as this is required to understand the state-of-the-art literature.

Chandrasekhar's *Radiative Transfer* (Chandrasekhar, 1960) is the seminal book on volume light transport. It introduces the radiative transfer equation, which describes the distribution of radiance in volumes. This equation is described by many authors (Hillaire, 2015; Pharr, Jakob and Humphreys, 2023; Fong *et al.,* 2017) and has been used to derive the volume rendering equation, used to compute the incident light along a view ray passing through participating media. This project focuses solely on single scattering, which considers only one bounce of light on the particles constituting the participating media. Multiple scattering tracks more than one bounce per light path and is out of scope of this project beyond a simple approximation.

Participating media materials are defined by the following parameters:

1. The absorption σ_a. This defines the light absorbed by the medium per unit distance.
2. The scattering σ_s. This defines the light scattered per unit distance.
3. The extinction coefficient $\sigma_t = \sigma_a + \sigma_s$. This describes the attenuation of the light in the medium.
4. The albedo $\rho = \frac{\sigma_s}{\sigma_s + \sigma_a}$. This effectively describes the colour of the medium.
5. The phase function p. This describes the distribution of light bounce directions in the medium.

Note that participating media materials can be described using a combination either of absorption and scattering, or extinction and albedo. The latter combination is often more intuitive for artists to understand (Hillaire, 2015) and is the convention used in this project.

This project uses Hillaire's volume scattering equation (Hillaire, 2015) to compute lighting. Let there be a single light with direction L whose radiance is given by R. For convenience of notation, any point along the view ray V is represented by its distance to the camera z, and z_{min} and z_{max} are the near and far bounds of the participating medium respectively. Let C denote the incoming radiance along the view ray from the other side of the participating medium. The lighting model only accounts for single scattering. The total radiance C_o along a given view ray is modelled as:

$$C_o = CT_V(z_{min}, z_{max}) + \int_{z_{min}}^{z_{max}} R\,\rho T_L(z) T_V(z_{min}, z) \mathrm{Vis}(z) p(L, V) \sigma_t(z)\, dz \qquad (1)$$

Equation 1

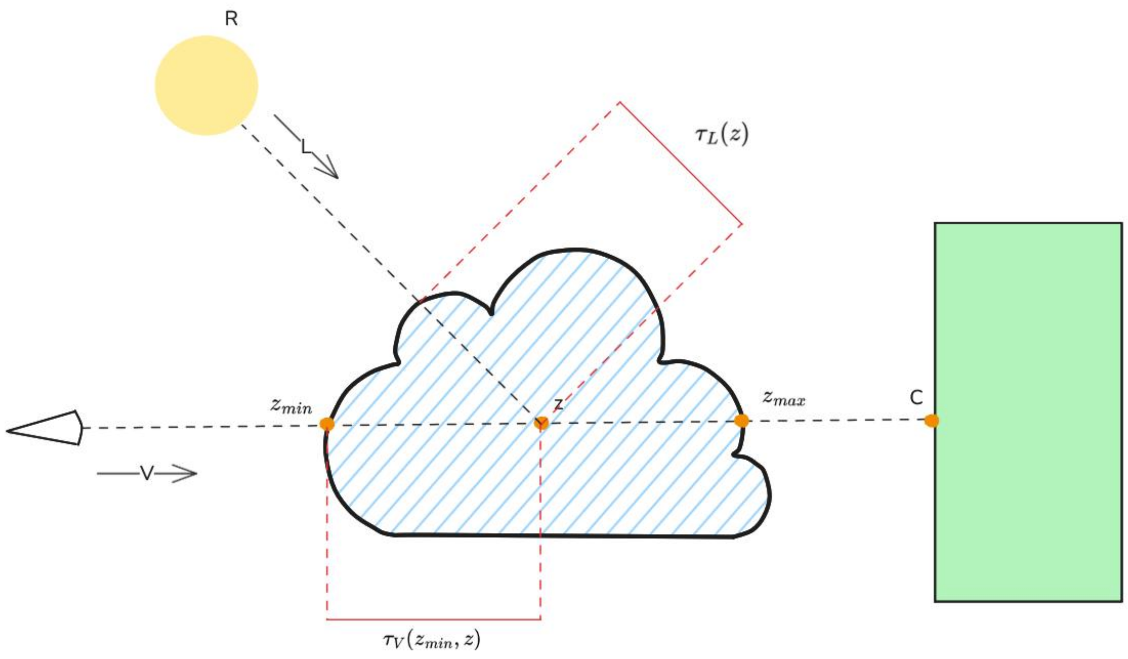


Figure 2: The components of the volumetric rendering equation

Here, T_V and T_L denote the transmittance along the view ray and the light ray respectively.

$$T_V(z_1, z_2) = e^{(-\tau_V(z_1, z_2))} \qquad (2)$$

Equation 2

where

$$\tau_V(z_1, z_2) = \int_{z_1}^{z_2} \sigma_t(x)\, dx$$

$\tau_V(z_1, z_2)$ is the optical thickness between z_1 and z_2 along the view ray V. T_L and τ_L are similar, except that the light ray is used instead of the view ray.

$$T_L(z) = e^{-\tau_L(z)} \tag{3}$$

Equation 3

The rendering equation can also be rewritten as follows:

$$C_o = A \times C + C_{scat} \tag{4}$$

Equation 4

Here, $A = T_V(z_{min}, z_{max})$ and C_{scat} is the first term containing the integral from Equation 1. Intuitively, C_{scat} represents the light scattered through the volume, C is the light on the other side of the volume and A is the attenuation applied to C through the volume.

2.2 Prior work and motivation

One of the first and most influential techniques for physically based volumetric rendering is Ubisoft's Volumetric Fog (Wronski, 2014). As pointed out by Wronski himself in his presentation, prior approaches that involved billboards and particles were artist-dependent, tedious to create and did not respond correctly to different lighting conditions. Volumetric Fog accumulates scattering and density information in a view frustum aligned 3D texture (called *froxels*), then the scattering equation is solved by marching through this texture along the view ray. This method provides convincing results for phenomena such as crepuscular rays, but is only suitable for large-scale media such as fog and does not account for self-shadowing along the light ray.

A more unified volumetric rendering system used by the Frostbite engine was presented by (Hillaire, 2015). Hillaire's presentation forms the basis of volumetric rendering systems used in modern titles such as The Last of Us Part II (Kovalovs, Aug 17, 2020) and Red Dead Redemption II (Bauer, 2019). This system is also voxel-based and supports self-shadowing. It makes use of multiple frustum-aligned

3D textures to accumulate material properties, light scattering and extinction and finally to integrate volumetric lighting.

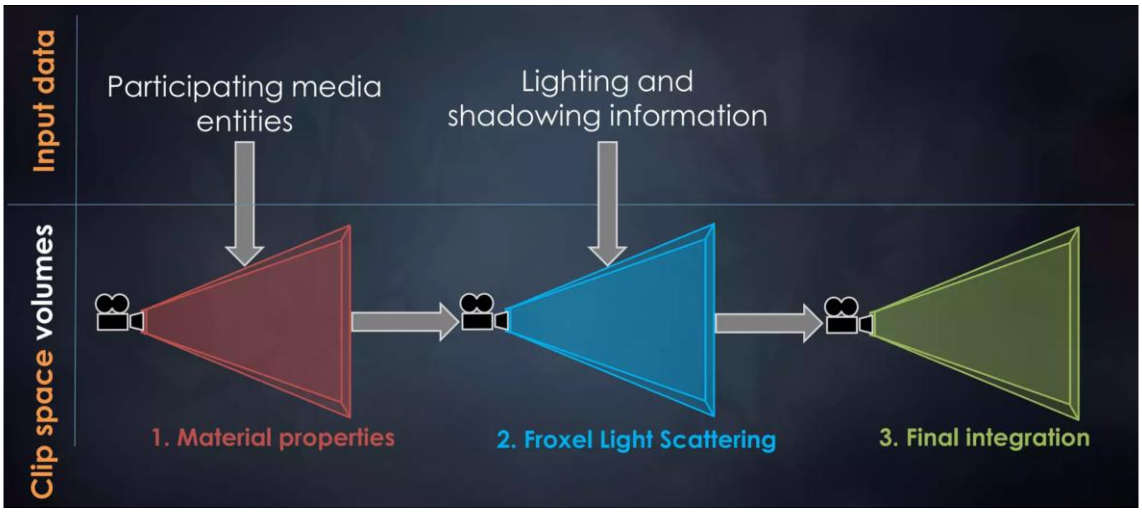

Input data
Clip space volumes
Participating media entities
Lighting and shadowing information
1. Material properties
2. Froxel Light Scattering
3. Final integration

Figure 3: The frustum-aligned volumes used by the Frostbite engine (Hillaire, 2015)

The method begins by first sampling sources of participating media to populate the material properties froxels (the so-called “V-buffer”, shown in red in Figure 1). Participating media data can come from different sources, such as depth fog, height fog or local fog volumes. Variation may be added by sampling density from an artist-authored 3D input texture. The scattering, extinction, emissive and average phase parameter g are sampled once per froxel and stored.

Next, the incoming scattered light and extinction are computed and stored in the second volume (blue in Figure 3). The incoming scattered light is computed as $R\ \rho\ Vis(z)\ \sigma_t(z)\ p(L,V)$ for each contributing light and summed.

Finally, this volume is used to integrate scattering and extinction along the view ray. This computes C_{scat} and $T_V(z_{min}, z_{max})$ for each froxel, where z_{max} is taken as the position of each froxel. Then when rendering surfaces, this volume can be sampled and C_{scat} and $T_V(z_{min}, z_{max})$ can be trivially applied similar to pre-multiplied colour and alpha.

Naively applying this method results in aliasing under camera motion or from high-frequency lights or shadows due to under-sampling of the media, since only one sample per voxel is taken per frame. To address this, (Hillaire, 2015) proposes to jitter sampling points within each froxel. The same jitter offset is used for all samples along the view ray. The scattering and material samples must be jittered in sync with one another. The previous frame's scattering and extinction volume is reprojected and blended with the current volume. This is effective in removing most aliasing artefacts, particularly the aliasing under camera motion and flickering of shadows. However, this temporal reprojection still causes ghosting when light sources are animated and further workarounds are needed for such cases (Epic Games). Thus, there is a need for a high-performance physically based volumetric rendering method that supports animation without causing temporal artefacts.

(Tricard, 2024) proposes a novel volume rendering technique called interval shading. He leverages the newly introduced mesh shading pipeline (Kubisch, 2018) to generate and emit triangle proxies that can be rasterized for each tetrahedron. The vertex attributes of these triangle proxies are set in such a way that after interpolation, the fragment shader receives two depth values z_{min} and z_{max} as input: z_{min} is the depth of the front-facing side of the tetrahedron and z_{max} is the depth of the rear-facing side. This depth interval can then be
used for shading. Since these tetrahedrons are simply rasterized without the need for any intermediate structure, they are easy to animate. If each tetrahedron is composed of participating media, a particle system of this
tetrahedrons could be used to model spatially varying participating media. Interval shading shows great promise for volume rendering as it is a method of cheaply computing the distance travelled through participating media, regardless of the distribution of tetrahedrons in space. However, the method must be adapted for physically based and order-dependent rendering.

2.3 The Interval Shading method

2.3.1 Overview

This section details the interval shading technique since it forms the foundation of the proposed method.

(Tricard, 2024) proposes to use tetrahedrons as primitives for volume rendering. He introduces an *interval shader* as a fragment shader that receives a depth interval for a single fragment. One value corresponds to the depth of the front faces of the tetrahedron, and the other to the back faces. The interval shader can then use this depth interval to compute the current fragment’s colour for volume rendering. Tricard proposes a method using mesh shaders to coerce the rasterizer to compute two depths per fragment. He first shows how this can be done for a triangular prismoid shape and later shows how a tetrahedron can be decomposed into multiple such prismoids.

2.3.2 Generating depth intervals for a prismoid

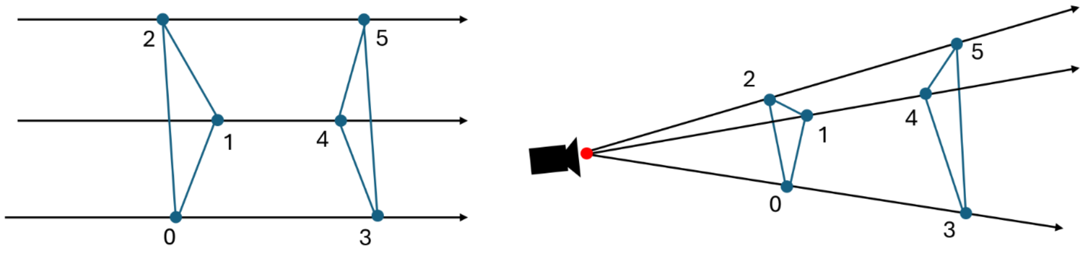


Figure 4: A triangular prismoid. Left: in projected space. Right: in world space (Tricard, 2024)

Consider the scenario in Figure 4. The triangular prismoid on the left is in projected space and is composed of six vertices with v_0, v_1, v_2 forming the first base and v_3, v_4, v_5 forming the second. We consider the case where both bases of the prismoid project to a single triangle in screen space, such that:

$$\frac{v_0.xy}{v_0.w} = \frac{v_3.xy}{v_3.w} = \frac{v_1.xy}{v_1.w} = \frac{v_4.xy}{v_4.w} = \frac{v_2.xy}{v_2.w} = \frac{v_5.xy}{v_5.w}$$

Any point p on the screen space triangle is defined as follows:

$$\begin{aligned} p.xy &= \frac{v_0.xy}{v_0.w}.\lambda_0 + \frac{v_1.xy}{v_1.w}.\lambda_1 + \frac{v_2.xy}{v_2.w}.\lambda_2 \\ &= \frac{v_3.xy}{v_3.w}.\lambda_0 + \frac{v_4.xy}{v_4.w}.\lambda_1 + \frac{v_5.xy}{v_5.w}.\lambda_2 \end{aligned}$$

Here, λ_0, λ_1 and λ_2 are the screen space barycentric coordinates of the point p. The z coordinates of the two bases of the prism can be found using:

$$\frac{1}{\mathrm{p.z_0}} = \frac{\mathrm{v_0.w}}{\mathrm{v_0.z}} \times \lambda_0 + \frac{\mathrm{v_1.w}}{\mathrm{v_1.z}} \times \lambda_1 + \frac{\mathrm{v_2.w}}{\mathrm{v_2.z}} \times \lambda_2 \tag{5}$$

Equation 5

$$\frac{1}{p.z_1} = \frac{v_3.w}{v_3.z} \times \lambda_0 + \frac{v_4.w}{v_4.z} \times \lambda_1 + \frac{v_5.w}{v_5.z} \times \lambda_2$$

where $p.z_0$ is the z coordinate of the point p on the first base of the prism and $p.z_1$ the z coordinate of the point p on the second. We then have $z_{min} = min(p.z_0, p.z_1)$ and $z_{max} = max(p.z_0, p_{z_1})$. (Tricard, 2024) proposes to coerce the rasterizer to compute $p.z_0$ and $p.z_1$, so as to have them as input variables in the fragment shader. This is done by emitting a triangle proxy for the prismoid with the following vertices:

$$\{\frac{v_0.x}{v_0.w}, \frac{v_0.y}{v_0.w}, 0,1\}, \{\frac{v_1.x}{v_1.w}, \frac{v_1.y}{v_1.w}, 0,1\}, \{\frac{v_2.x}{v_2.w}, \frac{v_2.y}{v_2.w}, 0,1\}$$

and with the following vertex attributes Z:

$$\{\frac{v_0.w}{v_0.z}, \frac{v_3.w}{v_3.z}\}, \{\frac{v_1.w}{v_1.z}, \frac{v_4.w}{v_4.z}\}, \{\frac{v_2.w}{v_2.z}, \frac{v_5.w}{v_5.z}\}$$

Setting the z coordinates to 0 forces the rasterizer to use screen space barycentric coordinates to interpolate the vertex attributes. Interpolation of Z by the screen space barycentric coordinates gives us $\frac{1}{p.z_0}$ and $\frac{1}{p.z_1}$ as defined above. Thus, we have two depths as input in the fragment shader.

2.3.3 Extension to tetrahedrons

To use the approach above for any general shape, that shape must first be decomposed into prismoids. This decomposition can be done for a tetrahedron analytically in real time. There are two cases to consider:

1. Case 1: When the tetrahedron projects onto a single triangle in screen space, the decomposition creates three prismoids.
2. Case 2: When the tetrahedron projects onto a quad in screen space, the decomposition creates four prismoids.

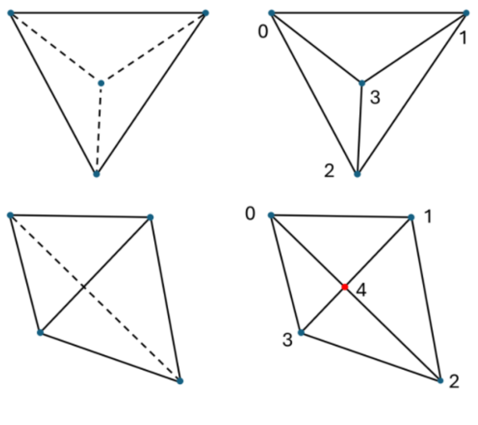


Figure 5: Two possible prismoid decomposition cases for a tetrahedron. Left: Wireframe of the tetrahedron. Right: Proxy generated by the mesh shader with vertex indices labelled. Top: Case 1, three triangles are created. Bottom: Case 2, four triangles are created.

To create and emit the triangle proxies, we start by defining t_i as the i^{th} vertex of the tetrahedron where $i \in [0,3]$ and p_j as the j^{th} vertex of the proxy.

2.3.3.1 Case 1

We define $p_{0:2}$ as the points on the silhouette of the proxy and p_3 as the point not on the silhouette.

$$p_{0:3} = \frac{t_{a:d}.xy}{t_{a:d}.w}$$

Where a, b, c and d are indices of the tetrahedron vertices, chosen so that $p_{0:2}$ are ordered in a clockwise direction as shown in Figure 5. For all vertices $p_{0:2}$ on the silhouette, the depths of the front and back faces of the prism are the same. Therefore, we define:

$$z_{min0:2} = z_{max0:2} = \frac{t_{a:c}.z}{t_{a:c}.w}$$

Our task is to find both depths corresponding to p_3, so that we can emit the triangle proxies. As p_3 is itself the projection of t_d we find:

$$z_0 = \frac{t_d.z}{t_d.w}$$

Next p_3 must be projected onto the face opposite. We follow Equation 5:

$$\frac{1}{z_1} = \frac{t_a.w}{t_a.z} \times \lambda_0 + \frac{t_b.w}{t_b.z} \times \lambda_1 + \frac{t_c.w}{t_c.z} \times \lambda_2$$

Then we have $z_{min} = min(z_0, z_1)$ and $z_{max} = max(z_0, z_1)$. Finally, we can emit three triangle proxies as described in 2.3.2: $\{p_0, p_1, p_3\}$, $\{p_1, p_2, p_3\}$ and $\{p_2, p_0, p_3\}$ (see Figure 5).

2.3.3.2 Case 2

Similar to the first case, we define $p_{0:3}$ as the points on the silhouette of the proxy and p_4 as the point not on the silhouette. Once again, the depths of $p_{0:3}$ are the same for both the front and back faces.

$$p_{0:3} = \frac{t_{a:d}.xy}{t_{a:d}.w}$$

$$z_{min0:3} = z_{max0:3} = \frac{t_{a:d}.z}{t_{a:d}.w}$$

Where a, b, c and d are indices of the tetrahedron vertices, chosen so that $p_{0:3}$ are ordered in a clockwise direction as shown in Figure 2. p_4 can be found by solving the intersection of the segments p_0p_2 and p_1p_3 such that:

$$p_4 = p_0 + (p_2 - p_0) \cdot t = p_1 + (p_3 - p_1) \cdot s$$

We adapt Equation (5) to find z_0 and z_1 for p_4, giving us:

$$\frac{1}{z_1} = \frac{t_a.w}{t_a.z} \cdot (1 - t) + \frac{t_c.w}{t_c.z} \cdot t$$

$$\frac{1}{z_0} = \frac{t_b.w}{t_b.z} \cdot (1 - s) + \frac{t_d.w}{t_d.z} \cdot s$$

Then we have $z_{min4} = min(z_0, z_1)$ and $z_{max4} = max(z_0, z_1)$. Finally, we emit four proxies as described in 2.3.2: $\{p_0, p_1, p_4\}$, $\{p_1, p_2, p_4\}$, $\{p_2, p_3, p_4\}$ and $\{p_3, p_0, p_4\}$ (see Figure 5).

2.3.4 Clipping of tetrahedrons

As explained in 2.3.2, when creating the proxy, we store the inverse of the depths as vertex attributes. If the tetrahedron is too close to the near plane, z_{min} approaches zero and the interpolation of $\frac{1}{z_{min}}$ will create numerical instability. (Tricard, 2024) proposes to solve this by clipping the tetrahedrons at some small distance ϵ before the near plane. However, this was omitted from the ISV method for the sake of simplicity. Tetrahedrons that get too close to the near plane are simply culled instead.

3 Methodology

3.1 The physically based rendering equation

Recall the volume scattering Equation 1. This section adapts the equation to the particulars of the artefact's test scenario. The scene will involve a single directional light with direction L whose radiance is given by R. The tetrahedrons are not emissive and are assumed to each have a uniform extinction coefficient σ and albedo ρ. $p(L, V)$ is the phase function. As we are considering only a single directional light, both p and R are independent of z and can be taken out of the integral. The rendering equation then becomes:

$$C_o = R\rho \times p(L,V) \int_{z_{min}}^{z_{max}} T_L(z) T_V(z_{min}, z) \mathrm{Vis}(z) \sigma_t(z)\ dz + C \times T_V(z_{min}, z_{max}) \qquad (6)$$

Equation 6

3.2 Using blending to solve the rendering equation

DirectX's blending functionality can be used to compute the physically based rendering equation, using a process similar to pre-multiplied alpha blending.

To solve the rendering equation using blending, the tetrahedrons must be rendered back-to-front. The GPU radix sort algorithm (Harada and Howes, 2011) was used to sort tetrahedrons by their depth. After sorting, the tetrahedrons are drawn to the render target using interval shading. In the pixel shader, C_{scat} and A are computed for a single tetrahedron and set as the RGB and alpha values respectively. Since C is already present in the render target, equation (5) can be computed using DirectX 12's blend stage by setting the blend operation to addition, the source blend factor to 1 and the destination blend factor to use source alpha. The resulting colour C_o will then be used as C for the next tetrahedron and hence the rendering equation is solved for the entire volume.

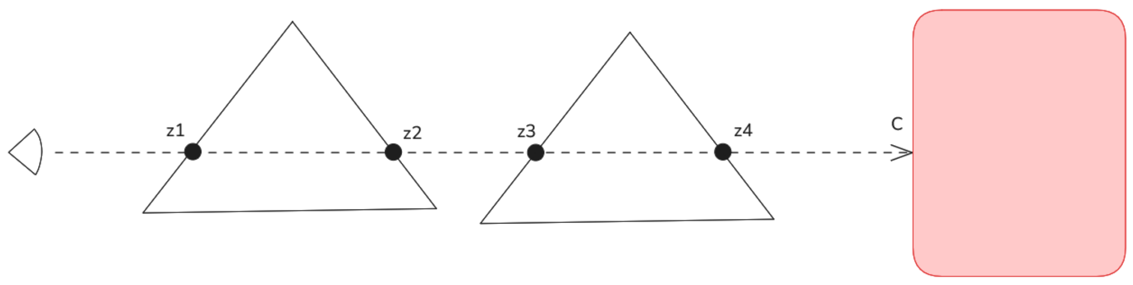


Figure 6: Integrating the rendering equation using blending

Consider the scenario in Figure 6. The view ray passes through two tetrahedrons in succession and encounters a colour C on the other side. The view ray intersects the tetrahedrons at z_1, z_2, z_3 and z_4. We assume $z_1 < z_2 < z_3 < z_4$. For brevity, let $K = R\rho \times p(L,V)$ and $f(z) = T_L(z)Vis(z)\sigma_t(z)$.

Applying the method above, the colour C_o seen by the camera is given by:

$$C_o = \left(CT_V(z_3,z_4) + K\int_{z_3}^{z_4} T_V(z_3,z)f(z)\ dz\right)T_V(z_1,z_2) + K\int_{z_1}^{z_2} T_V(z_1,z)f(z)\ dz$$

Since $T_V(z_1,z_2)T_V(z_3,z_4) = T_V(z_1,z_4)$ and $T_V(z_1,z_2)T_V(z_3,z) = T_V(z_1,z)$, we have:

$$C_o = CT_V(z_1,z_4) + K\int_{z_1}^{z_2} T_V(z_1,z)f(z)\ dz + K\int_{z_3}^{z_4} T_V(z_1,z)f(z)\ dz$$

The integral $\int_{z_2}^{z_3} T_V(z_1,z)f(z)\ dz$ is zero since there is no medium present. Therefore, we arrive at:

$$C_o = CT_V(z_1,z_4) + K\int_{z_1}^{z_4} T_V(z_1,z)f(z)\ dz$$

This is equivalent to applying Equation 6 to the interval $[z_1, z_4]$, hence proving that the blending method solves the rendering equation correctly for this idealised case. This can be trivially extended to multiple tetrahedrons.

In practice, this equation is not fully accurate since $\sigma_t(z)$ may vary between tetrahedrons and the tetrahedrons may overlap. However, this method still produces plausible results as shown in 4.1.

3.3 Choice of phase function

The phase function $p(L,V)$ describes the angular distribution of scattered light in a participating medium. The simplest phase function is the isotropic phase function (Pharr, Jakob and Humphreys, 2023):

$$p_{iso}(L,V) = \frac{1}{4\pi}$$

This function describes equal scattering in all directions and is hence independent of L and V. However, many naturally occurring media scatter light anisotropically, either towards or away from the incident direction. This method makes use of the Henyey-Greenstein phase function (Henyey and Greenstein, 1941) to model anisotropic as it is easy to implement and is used in a wide range of modern game engines such as the Decima engine (Guerrilla Games, 2017). It is as follows:

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

Where $cos\theta = dot(L,V)$ and $g \in (-1,1)$ controls the distribution of scattered light. Although the Henyey-Greenstein phase function is more interesting than the isotropic phase function, it is often not sufficient to describe complex scattering distributions found in nature. More complex phase functions can be modelled by a weighted sum of n simpler phase functions (Pharr, Jakob and Humphreys, 2023) as follows:

$$p(L,V) = \sum_{i=1}^{n} w_i\, p_i(L,V)$$

Where the weights w_i all sum to 1 to maintain normalisation. In this project, the phase function is computed as a weighted sum of the isotropic and Henyey-Greenstein phase functions as follows:

$$p(L,V) = (1-a)p_{iso}(L,V) + a \times p_{HG}(L,V)$$

Here, $a \in [0,1]$ is the *anisotropy* parameter, which describes the contribution of the Henyey-Greenstein phase function. This is distinct from the *asymmetry* parameter g, which controls the degree to which light is scattered towards or away from the direction of incident light.

It should be noted that this phase function was chosen for its simplicity, but any other phase function could easily be used with this method.

3.4 Computing optical thickness and transmittance along the light ray using a volumetric shadow map

To solve the physically based rendering equation, the transmittance along the light ray T_L at any point in the volume is computed by rendering the optical thickness along the light ray into a 3D texture, known as a volumetric shadow map. Since the extinction coefficient may vary between particles, it is insufficient to merely store the optical depth in the texture. The optical thickness from the light is accumulated at multiple slices placed perpendicular to the light direction, and these slices are in turn copied into the 3D texture for ease of sampling. This volumetric shadow map can be sampled in a later pass to compute $\tau_L(z)$ and hence compute $T_L(z)$.

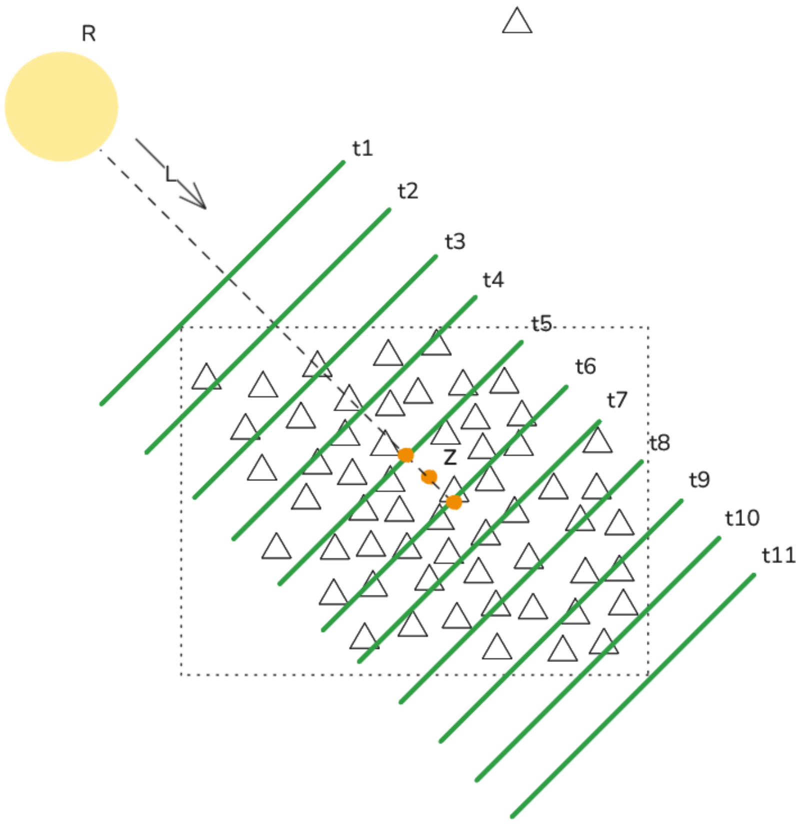


Figure 7: Volumetric shadow map for a cloud of tetrahedrons

In Figure 7, optical depths are interpolated between slices t_5 and t_6 to compute $\tau_L(z)$.

The algorithm to produce the volumetric shadow map makes use of an intermediate texture t and populates a 3D texture v (both storing 32-bit floats) with the contents of the volumetric shadow map:

1. Clear the intermediate texture t to be zero everywhere.
2. Set the first depth slice of v to be zero everywhere.
3. Set the blend state to add the incoming pixel to the existing render target.
4. For each depth slice with index i ranging from 1 to $n - 1$, where n is the number of depth slices in v, do the following:
    a. Set the far plane distance to $i \times \frac{f}{n-1}$, where f is the maximum depth of the shadow map, in world space units.
    b. Set the near plane distance to $(i - 1) \times \frac{f}{n-1}$.
    c. Set t as the render target and render the particle system from the point of view of the light, as one would with a conventional shadow map. Cull

any particles that intersect with the near plane. This prevents particles from being drawn more than once.

d. In the pixel shader, compute and output $\tau_V(z_{min}, z_{max})$, using Equation 2. Different solutions to this equation can be found in 3.7 and 3.9.

3.5 Implementation of interval shading

Tricard's original interval shading mesh shader (Tricard, 2024) was adapted for use in this project. Tricard proposes to clip tetrahedrons that intersect with the near plane, but this feature was omitted from this project for simplicity. Tetrahedrons that intersect with the near plane are simply culled instead. Since each tetrahedron is typically small in this case, this does not result in objectionable artefacts.

3.6 Algorithm to solve the physically based rendering equation

The algorithm to solve the physically based rendering Equation 6 is therefore as follows:

1. Render the volumetric shadow map.
2. Render a conventional shadow map.
3. Render all opaque geometry.
4. Sort the tetrahedrons by depth.
5. Draw the tetrahedrons back-to-front with the blend state configured as described in 3.2.
6. In the pixel shader, sample τ_{min} and τ_{max} from the volumetric shadow map.
7. Sample $Vis(z)$ from the conventional shadow map.
8. In the pixel shader, compute and output C_{scat} as the RGB colour.
9. In the pixel shader, compute and output $A = T_V(z_{min}, z_{max})$ as the alpha.

Filtering techniques such as large kernel PCF (Reeves, Salesin and Cook, 1987) can also be applied when sampling $Vis(z)$. This makes the shadows in the medium appear smoother, but at a higher computational cost.

Different methods of computing C_{scat} and $T_V(z_{min}, z)$ were attempted: an analytical solution assuming constant extinction, a Taylor series approximation with variable extinction and a numerical integration with variable extinction using Simpson's rule. These methods are detailed below.

3.7 Analytical solution assuming constant extinction

C_{scat} is given by:

$$C_{scat} = R\rho \times p(L,V) \int_{z_{min}}^{z_{max}} T_L(z) T_V(z_{min}, z) \mathrm{Vis}(z) \sigma_t(z)\ dz \tag{7}$$

Equation 7

To approximate $T_L(z)$, we assume that $\tau_L(z)$ varies linearly between z_{min} and z_{max}. Let $\tau_{min} = \tau_L(z_{min})$ and $\tau_{max} = \tau_L(z_{max})$.

$$T_L(z) = e^{\left(-\tau_{min} - \frac{(\tau_{max}-\tau_{min})(z-z_{min})}{z_{max}-z_{min}}\right)} \tag{8}$$

Equation 8

As a first step, the extinction $\sigma_t(z)$ was assumed to be constant throughout a tetrahedron, i.e. $\sigma_t(z) = \sigma, \forall\ z$. $\tau(z_1, z_2)$ then becomes $\sigma(z_2 - z_1)$. For simplicity, $Vis(z)$ was assumed to be 1 everywhere. Equation 7 can then be integrated analytically:

$$C_{scat} = R\ \rho \times p(L,V)$$
$$\times \frac{\sigma(z_{max} - z_{min})\left(e^{(\sigma z_{max} + \tau_{max})} - e^{(\sigma z_{min} + \tau_{min})}\right) e^{(-\sigma z_{max} - \tau_{max} - \tau_{min})}}{\sigma z_{max} - \sigma z_{min} + \tau_{max} - \tau_{min}}$$

Also:

$$A = T_V(z_{min}, z_{max}) = e^{-\sigma(z_{max} - z_{min})}$$

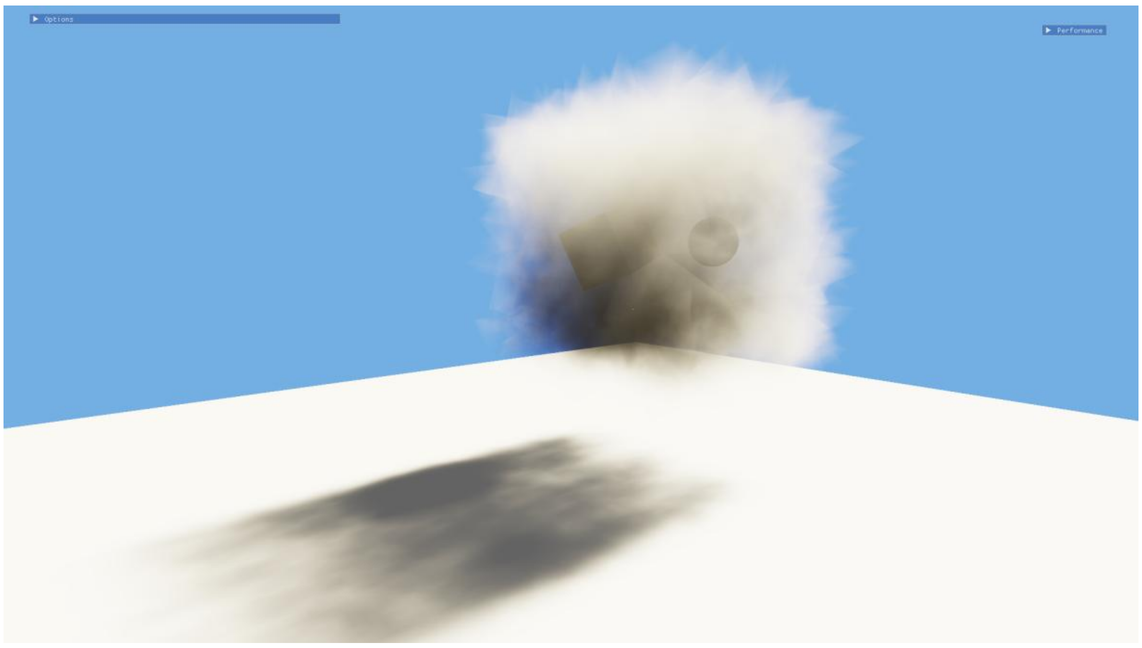


Figure 8: The results of the analytical shading method with constant extinction

Since Equation 7 and $T_V(z_{min}, z_{max})$ can be integrated analytically, they were trivial to compute and output in the interval shader. The resulting visuals resembles a cloud of smoke, especially with the volumetric shadows. However, the constant extinction value causes the entire tetrahedron to be filled with participating medium, resulting in the tetrahedron edges being visible. These sharp geometric edges are not visible in real smoke or clouds.

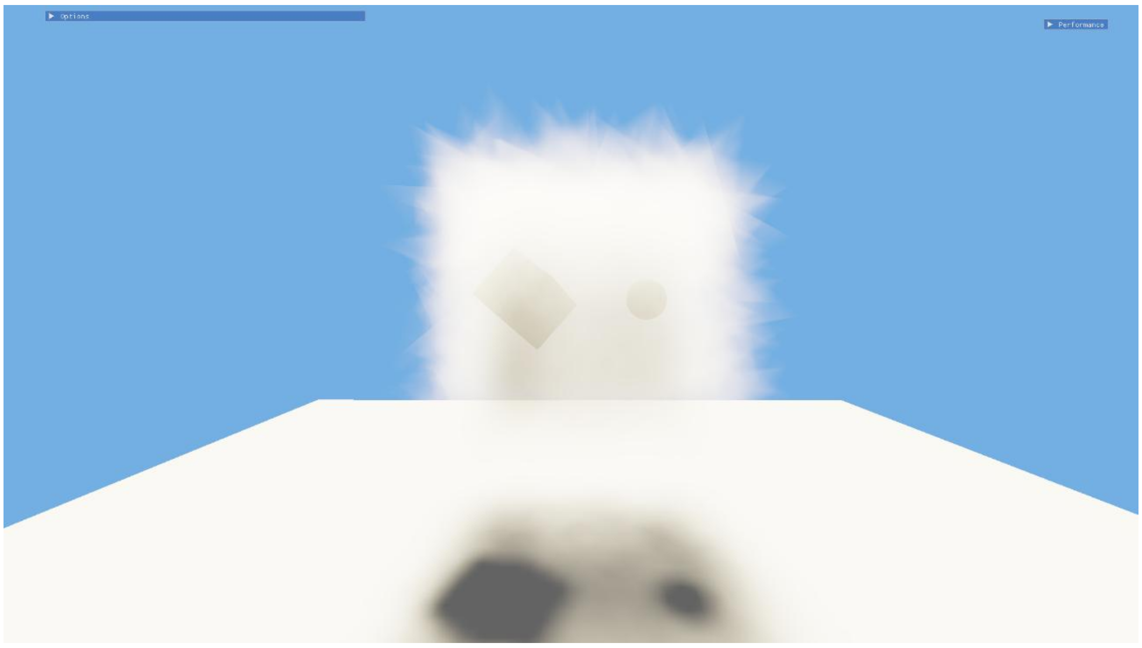


Figure 9: Constant extinction within a tetrahedron causes edges to be visible

3.8 Hiding of tetrahedron edges

To hide the tetrahedron edges, the extinction $\sigma_t(z)$ is faded exponentially with distance from the tetrahedron centre.

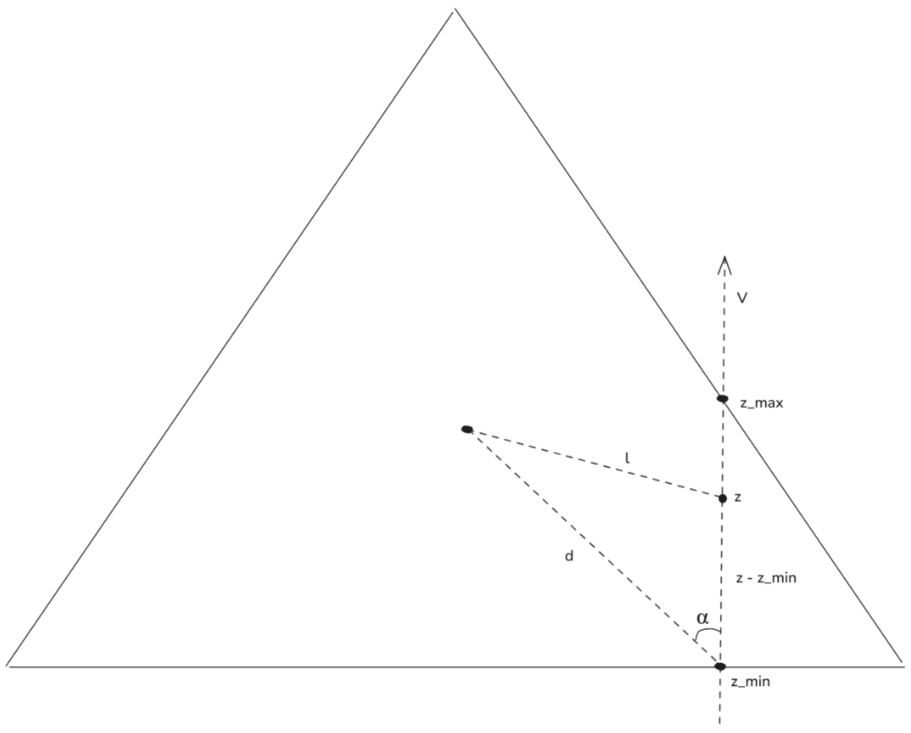


Figure 10: Extinction fading

Consider the tetrahedron shown in Figure 10. The pixel along the view ray V is currently being shaded. The points of entry and exit are labelled z_{min} and z_{max} respectively. The vector from z_{min} to the centre has a length d and angle α with V. Using the law of cosines, the distance $l(z)$ from any point z along V to the centre is given by:

$$l(z) = \sqrt{-2\,d(z - z_{min})\cos(\alpha) + d^2 + (z - z_{min})^2}$$

Using this, we can fade the extinction exponentially:

$$\sigma_t\big(l(z)\big) = e^{\frac{-50l(z)^2}{u^2}}$$

u is a parameter that can be used to control the extinction falloff distance. Finally, $\sigma_t(z)$ is therefore:

$$\sigma_t(z) = \sigma e^{\left(\frac{50\left(2\,d(z-z_{min})\cos(\alpha)-d^2-(z-z_{min})^2\right)}{u^2}\right)} \qquad (9)$$

Equation 9

3.9 Integrating C_{scat} using a Taylor series expansion

C_{scat} and A must be computed again using the faded extinction function. From Equation 9 and Equation 2, we have:

$$T_V(z_{min}, z) = e^{\left(-\frac{1}{20}\sqrt{2}\sqrt{\pi}\sigma u\left(erf\left(\frac{5\sqrt{2}d\cos(\alpha)}{u}\right)-erf\left(\frac{5\sqrt{2}(d\cos(\alpha)-z+z_{min})}{u}\right)\right)e^{\left(\frac{50\,d^2\cos^2(\alpha)}{u^2}-\frac{50\,d^2}{u^2}\right)}\right)} \quad (10)$$

Equation 10

HLSL does not provide an intrinsic function to compute erf. Tokuyoshi's erf code (Tokuyoshi, 2022) was used instead.

It is impractical to analytically integrate Equation 7 when using the faded extinction function. An approximation using the Taylor series expansion was attempted instead. The Taylor series expansion of a function $f(x)$ at a real or complex number a is given by:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots = \sum_{n=0}^{\infty}\frac{f^{(n)}(a)}{n!}(x-a)^n$$

Integrating this, we have:

$$\int f(x)dx = f(a)(x-a) + \frac{f'(a)}{2!}(x-a)^2 + \frac{f''(a)}{3!}(x-a)^3 + \cdots$$
$$= \sum_{\mathrm{n}=0}^{\infty}\frac{f^{(n)}(a)}{(n+1)!}(x-a)^{n+1}$$

To maximise accuracy, the expansion point a was chosen to be $\frac{z_{min}+z_{max}}{2}$. The n-th order derivative $f^{(n)}(a)$ was computed numerically using the finite forward difference. The first four terms of the series above were computed and used to approximate the integral in (7). However, this approach did not result in a visually accurate appearance. Many geometric edges are still visible, and the image contains objectionable firefly artefacts. Owing to these shortcomings, this approach was not explored further.

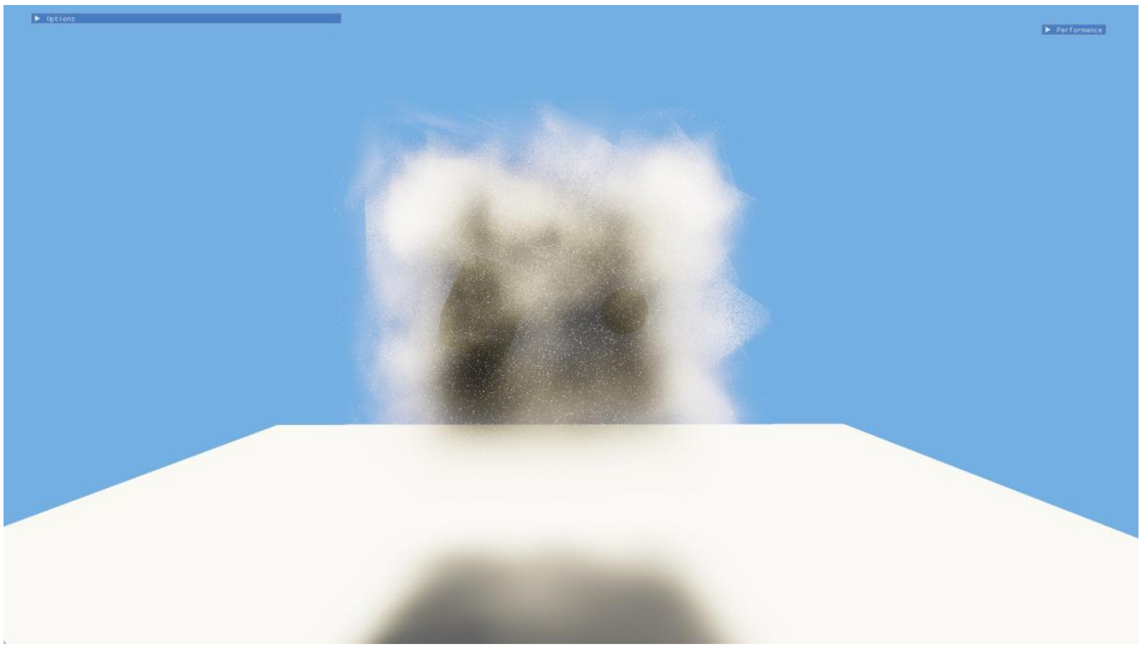


Figure 11: Taylor series integration results in firefly artefacts

3.10 Integrating C_{scat} using Simpson's rule

Since the Taylor series integration method proved to be inaccurate, an alternative numerical integration approach using Simpson's rule was explored. Simpson's rule is as follows:

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right)$$

The accuracy of this method can be improved by dividing the interval $[a, b]$ into n steps or sub-intervals, applying Simpson's rule to each interval in turn and then summing the results. As n increases, the error diminishes. Since a high step count incurs a performance cost, the step count n was made configurable in the artefact. By default, only a single step spanning the entire interval $[z_{min}, z_{max}]$ is used.

To apply Simpson's rule to Equation (7), $T_L(z)$ and $T_V(z_{min}, z)$ are computed using Equation 8 and Equation 10 respectively. $\mathrm{Vis}(z)$ must be sampled from a shadow map, and $\sigma_t(z)$ is computed using Equation 9.

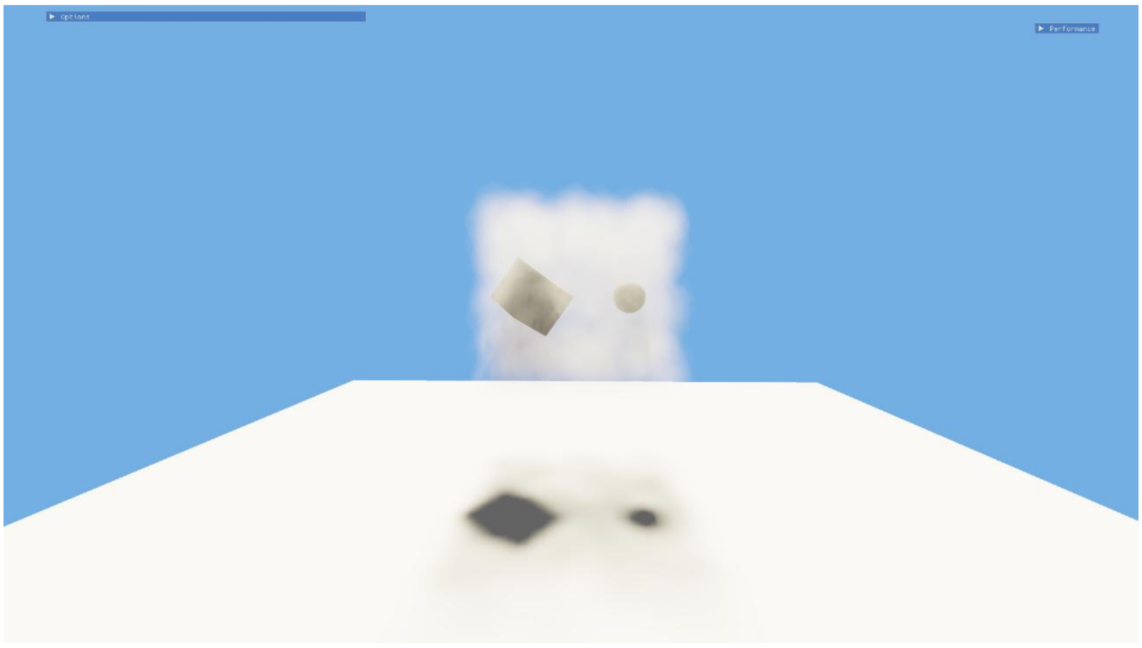


Figure 12: The Simpson’s rule method avoids objectionable artefacts

As shown above, the Simpson’s rule method produces plausible results. There are subtle artefacts detailed in 4.1, but these do not detract from the overall appearance of the medium.

3.11 Early return for invisible pixels

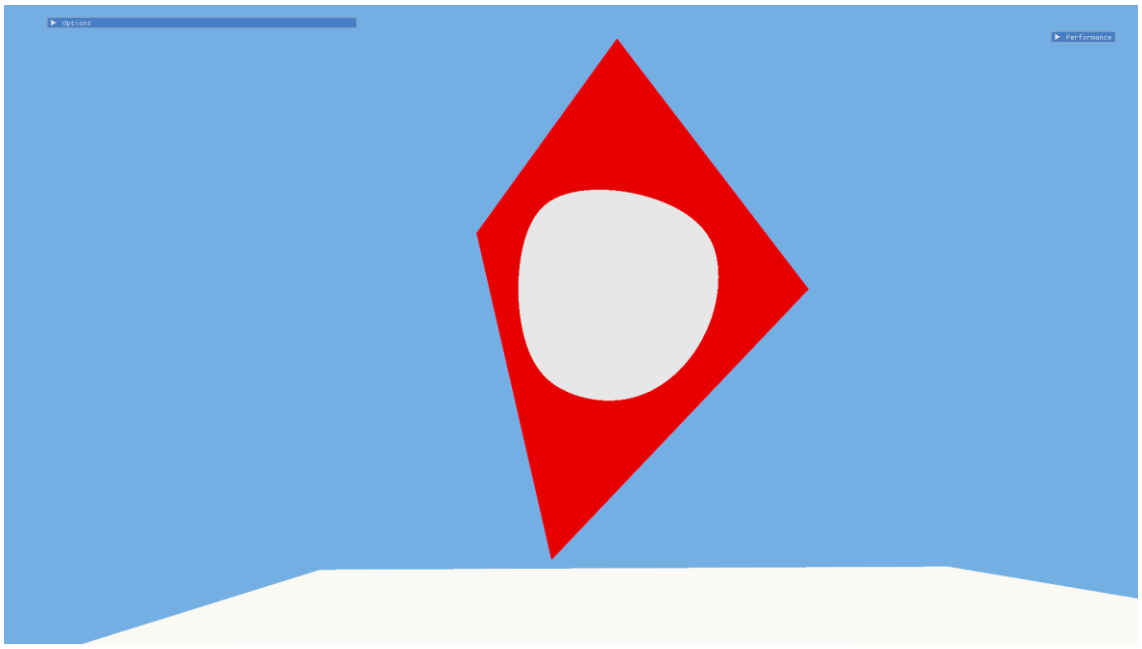


Figure 13: Wasted pixels visualised for a single particle with u=1.6. Visible pixels are shaded in green, while pixels with extinction below the cutoff are shaded red

When the faded extinction from Equation 9 is applied, the optical thickness at many pixels covered by a tetrahedron becomes so low as to be invisible. However, the pixel shader is still invoked for these pixels, resulting in unnecessary work done.

These “wasted pixels” are visualised above in red. To mitigate this performance loss, the optical thickness is set to 0 if $\tau_V(z_{min}, z_{max})$ is found to be less than some small threshold ϵ. This results in significant performance gains and is documented in 4.2.6.

3.12 Multiple scattering approximation

The model presented thus far only accounts for single scattering. To approximate multiple scattering, a term C_{mul} is computed as follows:

$$C_{mul} = \frac{\sigma_t\left(\frac{z_{min} + z_{max}}{2}\right)}{\sigma} \times \rho\, R\, p(L, V) \times 0.001 \times m \tag{11}$$

Equation 11

Here, $m \in [0,1]$ is a parameter to control the intensity of the multiple scattering term. Though this term is crude and not physically based, it improves the visual fidelity as explained in 4.1.3.

3.13 Volumetric shadows on opaque objects

The volumetric shadow map can also be sampled to cast shadows onto opaque objects. This process is similar to sampling a conventional shadow map. The algorithm for volumetric shadowing is as follows:

1. In the pixel shader, convert the world space coordinates of the pixel being shaded into shadow map space.
2. Sample the optical thickness τ from the volumetric shadow map.
3. Compute the transmittance $T = e^{-\tau}$.
4. Use the transmittance T as the shadow factor when computing lighting.

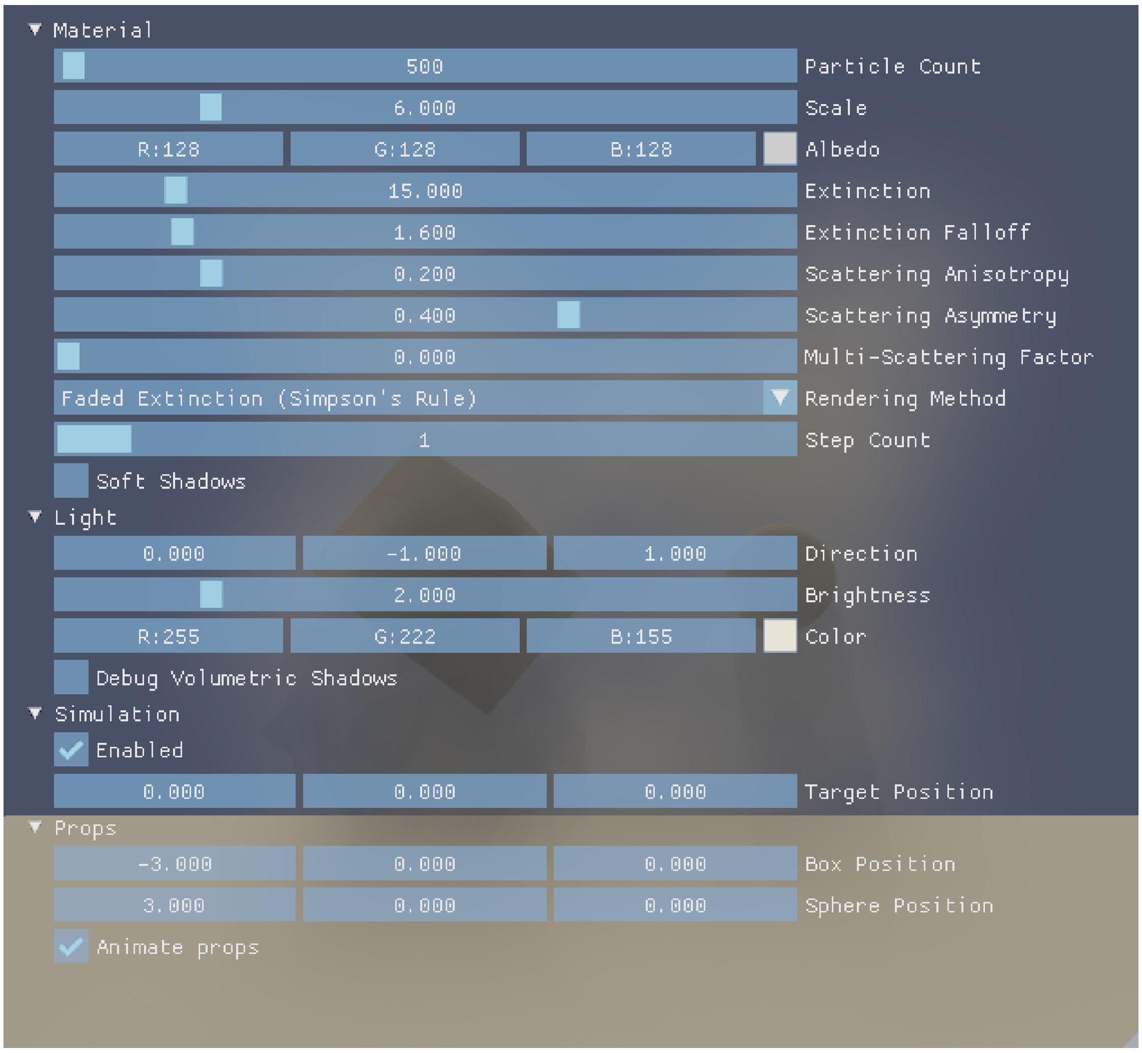
3.14 Animation of the medium

To demonstrate that the rendering method is without artefacts even in motion, a simple particle simulation was implemented. Each particle is assigned a target position within a cube. The entire cube can be moved, and particles will accelerate towards their target positions over time with their velocities damped by viscosity. Additionally, the user can press the left mouse button to apply an impulse to any particles that are close to the mouse cursor. The simulation logic was kept deliberately simple to allow for focus on the rendering method.

3.15 Occlusion of the medium

Since the interval shading technique works by splatting triangle proxies onto the near plane, the tetrahedron particles are not occluded by the preceding contents of the depth buffer. To overcome this limitation, the pixel shader computes and writes a depth to the depth buffer using DirectX's `SV_Depth` semantic. The pixel depth is computed as the depth of the near point of the interval.

3.16 Parameterisation



```
▼ Material
  500                                     Particle Count
  6.000                                   Scale
  R:128       G:128       B:128           Albedo
  15.000                                  Extinction
  1.600                                   Extinction Falloff
  0.200                                   Scattering Anisotropy
  0.400                                   Scattering Asymmetry
  0.000                                   Multi-Scattering Factor
  Faded Extinction (Simpson's Rule)   ▼   Rendering Method
  1                                       Step Count
  ☐ Soft Shadows
▼ Light
  0.000       -1.000      1.000           Direction
  2.000                                   Brightness
  R:255       G:222       B:155           Color
  ☐ Debug Volumetric Shadows
▼ Simulation
  ✓ Enabled
  0.000       0.000       0.000           Target Position
▼ Props
  -3.000      0.000       0.000           Box Position
  3.000       0.000       0.000           Sphere Position
  ✓ Animate props
```

Figure 14: Available parameters

The project has many parameters, as shown above in Figure 14. The main parameters concerning the rendering method can be found under the Material section. Some of these parameters are adjusted slightly for user-friendliness, but they all map to variables used by this method as explained below:

1. The Particle Count slider controls the number of particles. It ranges from 1 to 65,535.
2. The Scale slider adjusts the size of each particle. A scale of 1 means that the distance from any vertex to the centre of a tetrahedron is 1m.
3. The Albedo is the colour of the medium, mapping directly to ρ in the rendering equation.
4. The Extinction parameter varies from 0 to 100 and maps to σ, adjusted slightly for user-friendliness. In addition, each particle has a randomized extinction multiplier in the range $[5,10]$. σ is calculated as: $extinction\ \times extinction\ multiplier\ \times 0.0001$.
5. The Extinction Falloff ranges from 0 to 10 and controls the falloff parameter u in Equation (8), which is computed using $u = particle\ scale\ \times falloff$.
6. The Scattering Anisotropy controls the parameter a from 3.3.
7. The Scattering Asymmetry controls the parameter g from 3.3.
8. The Multi-Scattering Factor is the multiple scattering parameter m in Equation 11.
9. The Rendering Method dropdown can be used to toggle the rendering method between Simpson's rule (3.10), the Taylor series integration (3.9) or the analytical solution without faded extinction (3.7). There is also an option to visualise wasted pixels.
10. The Step Count slider ranges from 1 to 10 and controls the number of Simpson's rule steps taken, if the Simpson's rule rendering method is selected.
11. The Soft Shadows checkbox can be used to toggle the use of large kernel PCF when sampling $Vis(z)$.

In addition to these, controls to modify the light (whose irradiance is mapped to R), move the medium, pause the particle simulation or move the props can be found in the other UI sections.

3.17 Testing and validation

To assess whether the method can effectively render participating media, a qualitative analysis of the following desired characteristics was conducted:

1. Scattering of light towards the camera
2. Self-shadowing of the medium

3. Volumetric shadows cast by opaque objects into the medium
4. Volumetric shadows cast by the medium onto opaque objects
5. Anisotropic scattering of light
6. Visual fidelity under motion

To assess the performance characteristics of the method, the frame time in milliseconds was measured and plotted as the following parameters were varied:

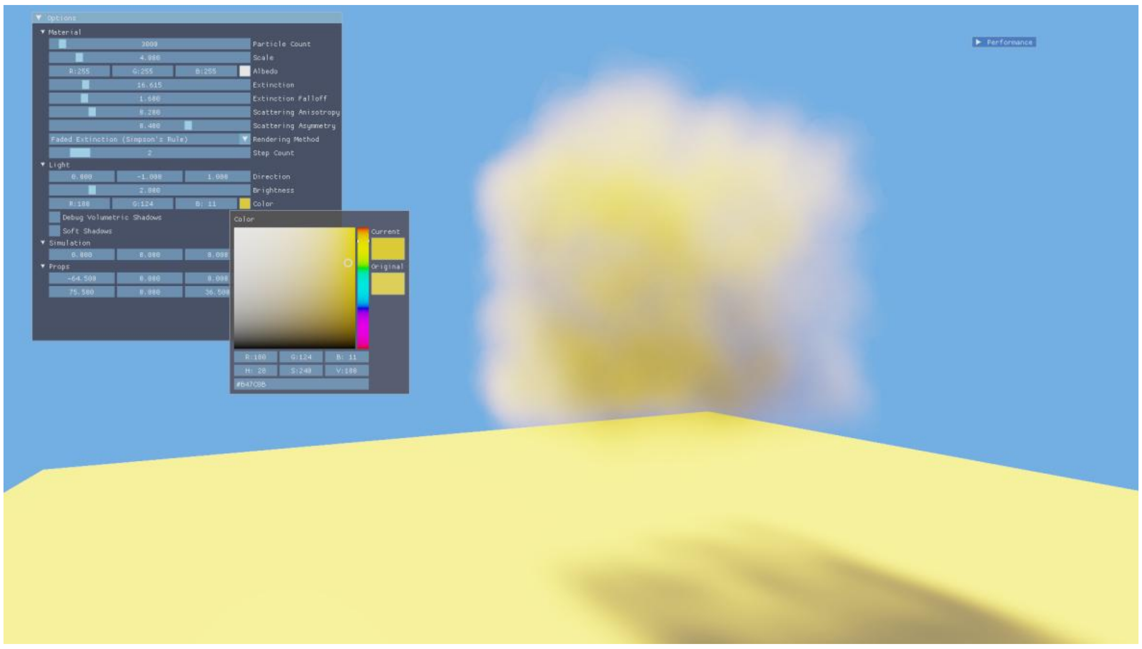
1. Simpson's rule steps
2. Particle count
3. Camera distance
4. Particle size
5. Large kernel PCF (on vs off)
6. Extinction fading
7. Volumetric shadow map depth slices
8. Volumetric shadow map resolution

4 Results

4.1 Qualitative analysis of the visuals

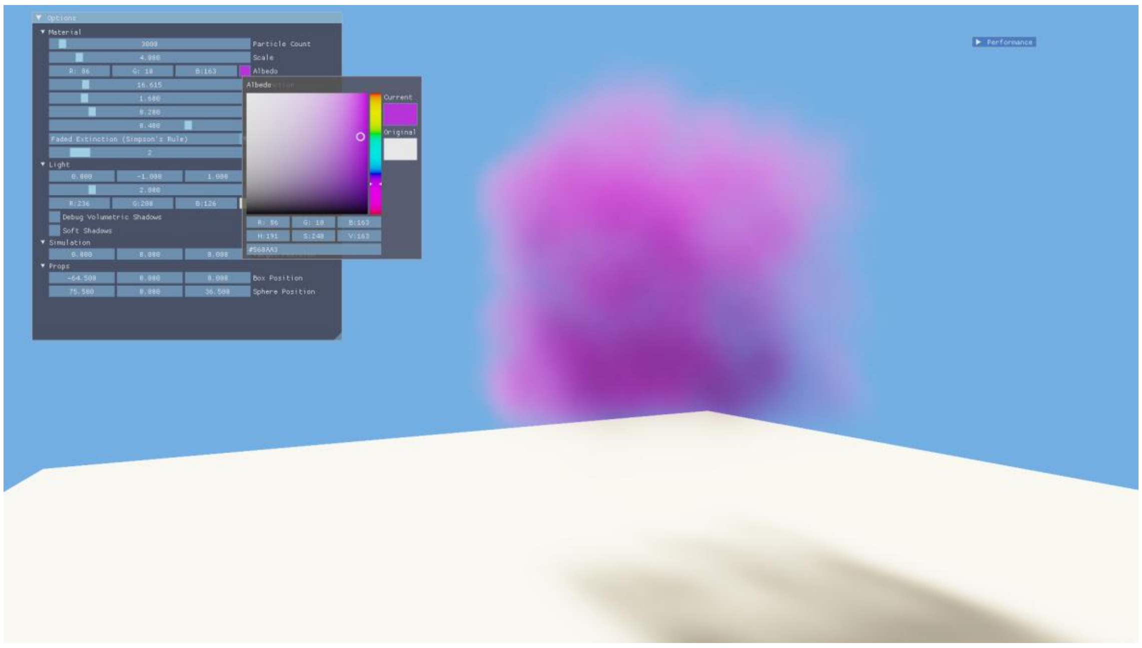
4.1.1 Scattering of light towards the camera

Light scattering in participating media make illumination visible even in pixels where there are no visible surfaces to reflect it (Pharr, Jakob and Humphreys, 2023). This scattering phenomenon is observable in the artefact, and the scattered light colour is affected both by the colour of the light and the albedo of the medium.


```
▼ Options
▼ Material
  3000                                      Particle Count
  4.000                                     Scale
  R:255        G:255        B:255           Albedo
  16.615                                    Extinction
  1.600                                     Extinction Falloff
  0.200                                     Scattering Anisotropy
  0.400                                     Scattering Asymmetry
  Faded Extinction (Simpson's Rule)         Rendering Method
  2                                         Step Count
▼ Light
  0.000        -1.000       1.000           Direction
  2.000                                     Brightness
  R:180        G:124        B: 11           Color
  Debug Volumetric Shadows
  Soft Shadows
▼ Simulation
  0.000        0.000        0.000
▼ Props
  -64.500      0.000        0.000
  75.500       0.000        36.500

Color
  Current
  Original
  R:180   G:124   B: 11
  H: 28   S:240   V:180
  #B47C0B
```

Figure 15: The colour of the participating medium is affected by the colour of incident light



3000 Particle Count
4.000 Scale
R: 86 G: 10 B:163 Albedo
16.615
1.600
0.200
0.400
Faded Extinction (Simpson's Rule)
2
0.000 -1.000 1.000
2.000
R:236 G:200 B:126
Debug Volumetric Shadows
Soft Shadows
0.000 0.000 0.000
-64.500 0.000 0.000 Box Position
75.500 0.000 36.500 Sphere Position
Albedo
R: 86 G: 10 B:163
H:191 S:240 V:163
#560AA3

Figure 16: The colour of the participating medium is affected by its own albedo

4.1.2 Self-shadowing of the medium

Light is scattered or absorbed as it passes through the medium, resulting in some parts of the medium appearing darker than others.

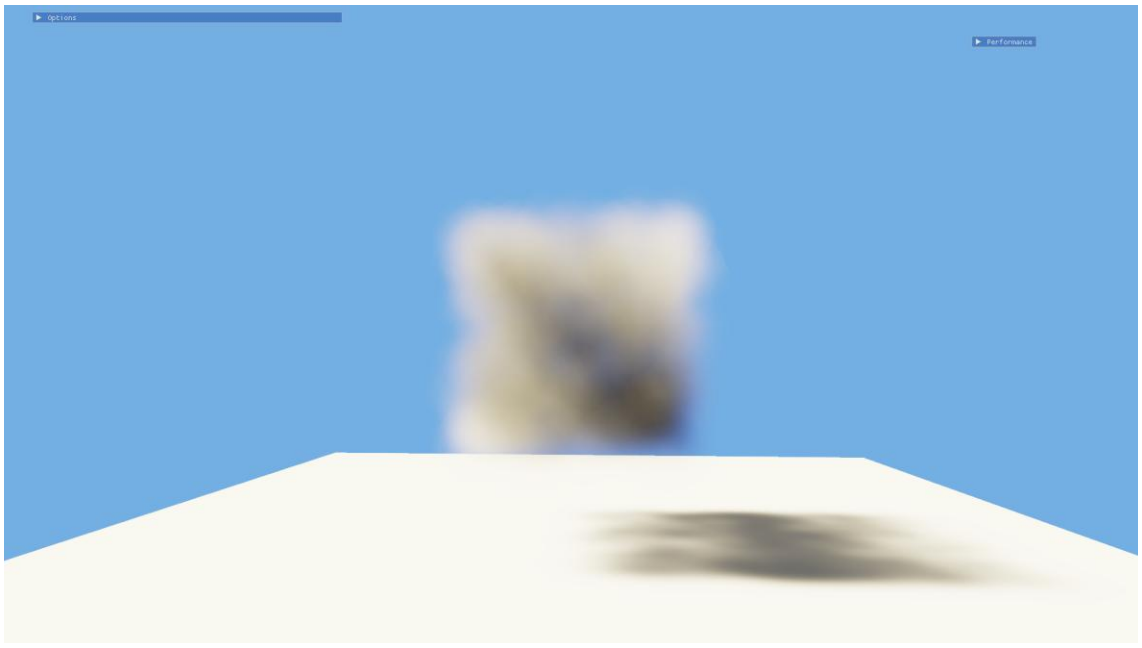


Figure 17: The medium shadows itself, resulting in the bottom right portion appearing dark

4.1.3 Volumetric shadows cast by opaque objects

When light passing through a participating medium is occluded, the occluded parts of the medium do not scatter light towards the camera, resulting in a volumetric shadow in the medium. These volumetric shadows are observable in the artefact as well.

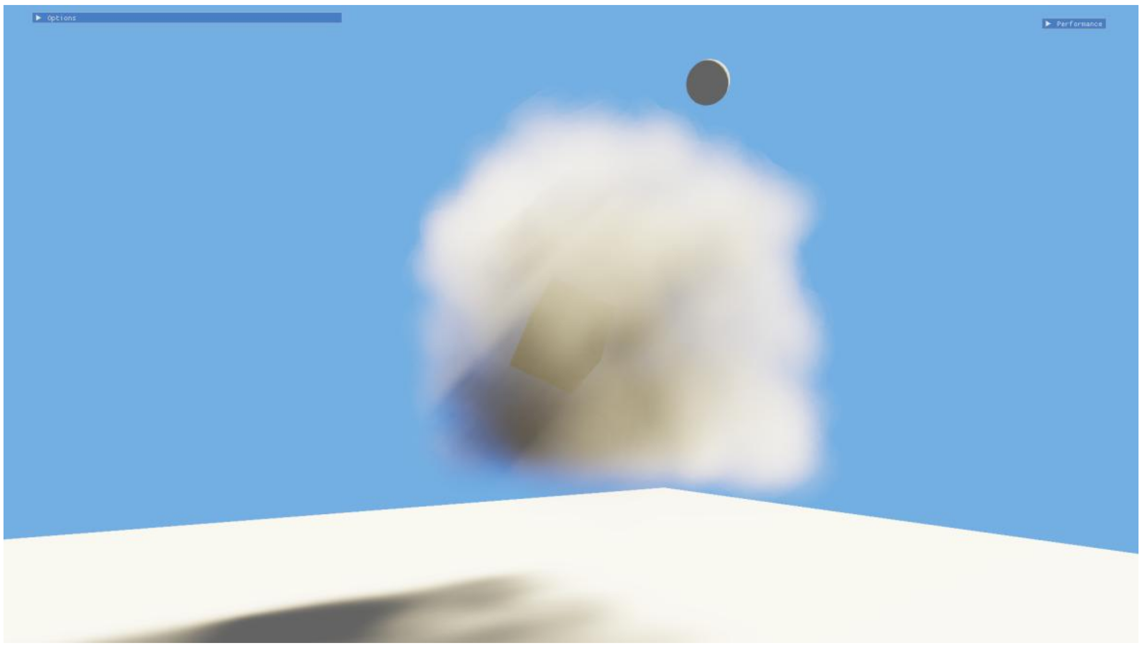


Figure 18: The cube and the sphere both occlude light, casting volumetric shadows into the participating medium

If the particles are large and few in number, aliasing artefacts can appear due to under-sampling of $Vis(z)$. These artefacts can be removed either by increasing the Simpson's rule step count or by increasing the number of particles, as shown in Figure 19.

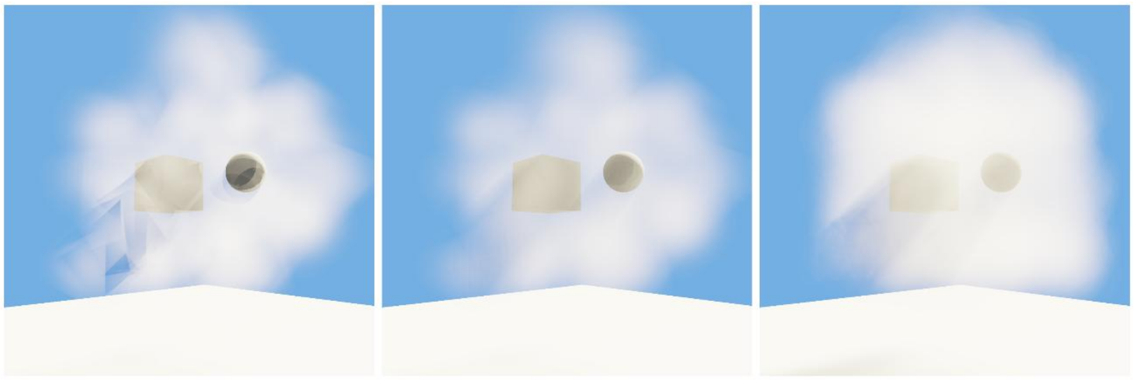


Figure 19: Left: 100 particles, size 6, 1 step. Centre: 100 particles, size 6, 10 steps. Right: 5000 particles, size 6, 1 step

When viewing a volumetric shadow head-on, it can sometimes look like a hole through the medium, as shown in Figure 20.

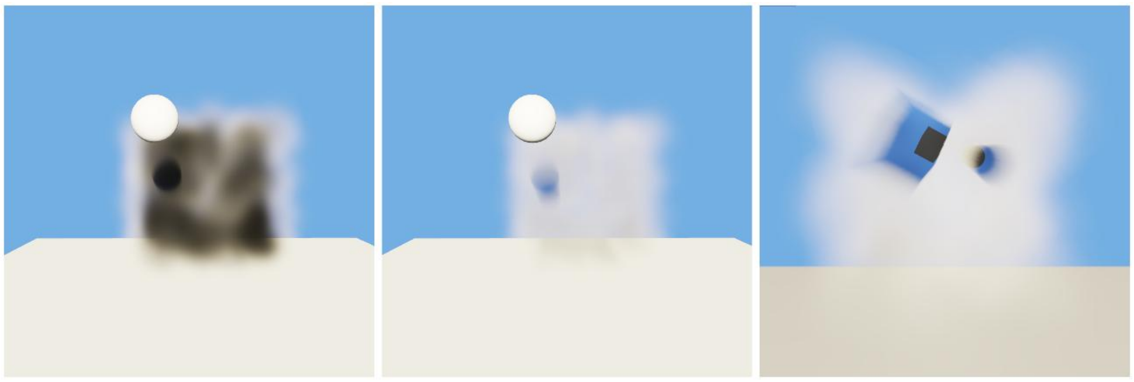


Figure 20: Left: A shadow cast onto a high extinction medium. Centre: When a shadow is cast onto a low extinction medium, it looks like a hole. Right: Hole-like shadows viewed from the other side of the medium

This phenomenon occurs because no light is scattered towards the eye from within the shadow, making the space appear empty. Multiple scattering is required to alleviate this effect, as shown in Figure 21.

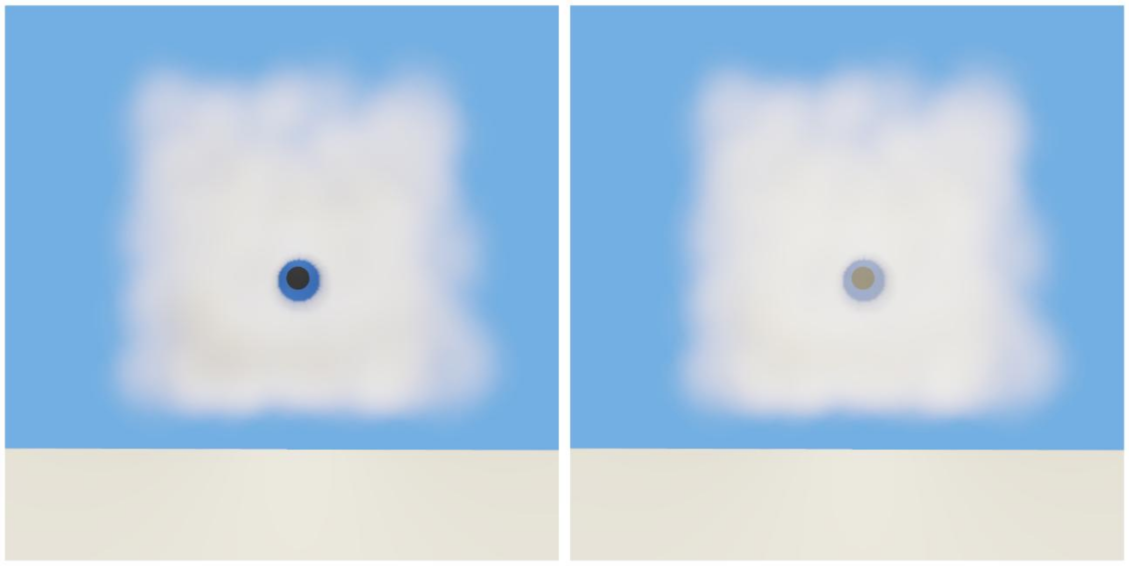


Figure 21: Multiple scattering (right) can improve volumetric shadows

4.1.4 Volumetric shadows cast onto opaque objects

Light is attenuated as it passes through the participating medium, resulting in shadows cast onto other objects (Pharr, Jakob and Humphreys, 2023).

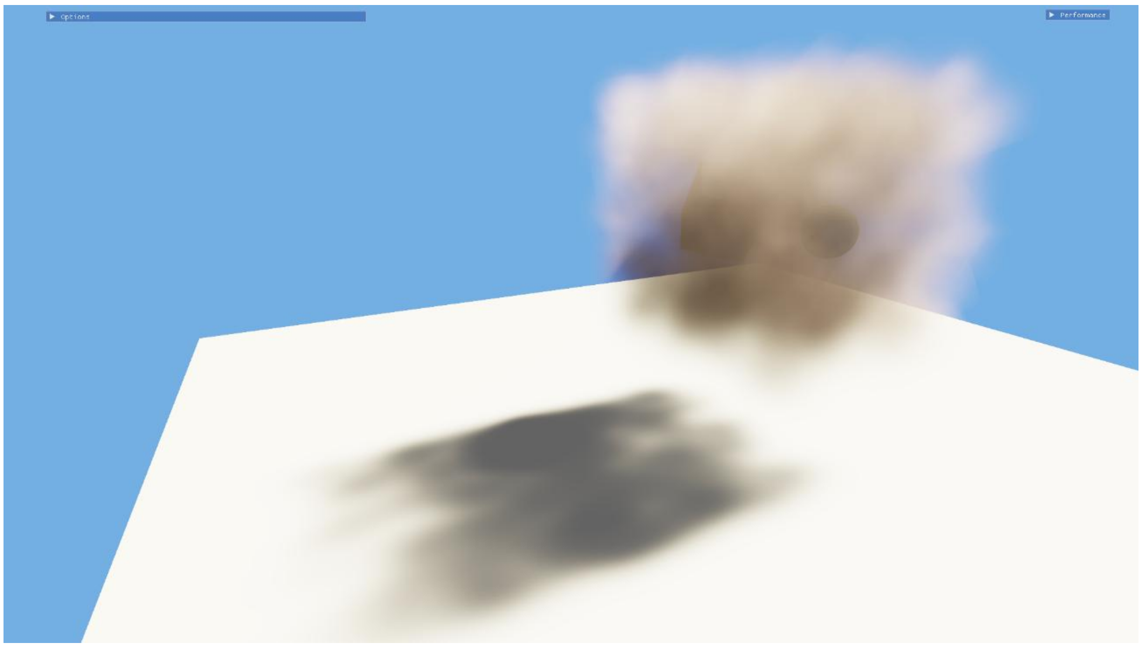


Figure 22: The participating medium casts a shadow onto the ground

Participating media in the artifact do indeed cast shadows onto the ground. However, these shadows do not take on the colour of the medium and are greyscale instead.

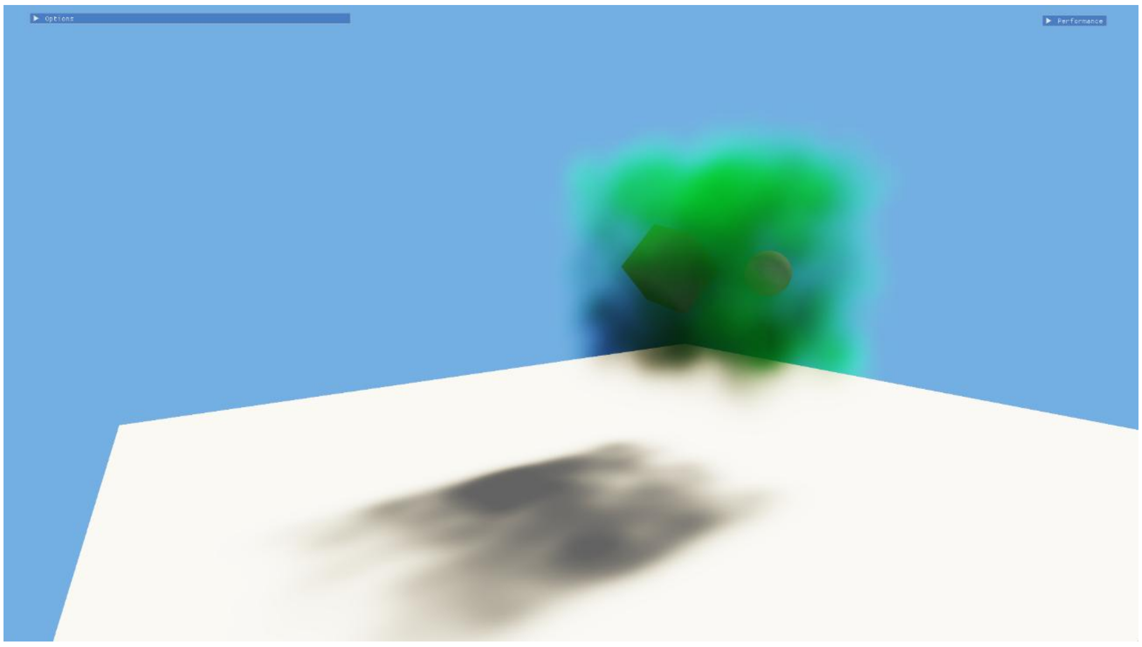


Figure 23: The shadow does not take on the colour of the medium

4.1.5 Anisotropic scattering of light

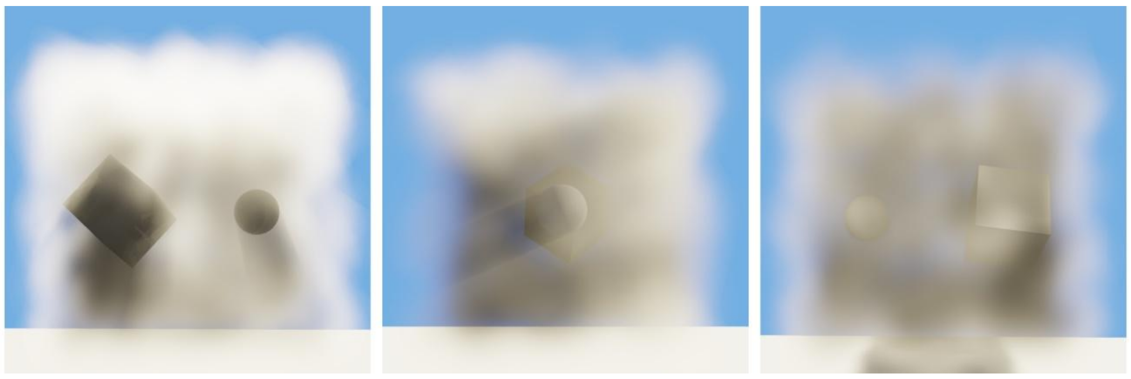


Figure 24: The rendered medium viewed while looking towards the light, orthogonal to the light and away from the light

In many naturally occurring media, the scattering of light is dependent on the angle between the incident and outgoing direction. In the artefact, the rendered medium looks different when viewed from different angles, and this anisotropy is configurable.

4.1.6 Maintenance of visual fidelity under motion

All of the above properties are observable in this artefact even when the particles, light, camera or all three are in motion, appearing correctly without the presence of any visual glitches or artefacts.

4.2 Quality settings and performance

These performance metrics were captured on a NVIDIA GeForce RTX 4070 Mobile graphics card.

4.2.1 Simpson's rule steps

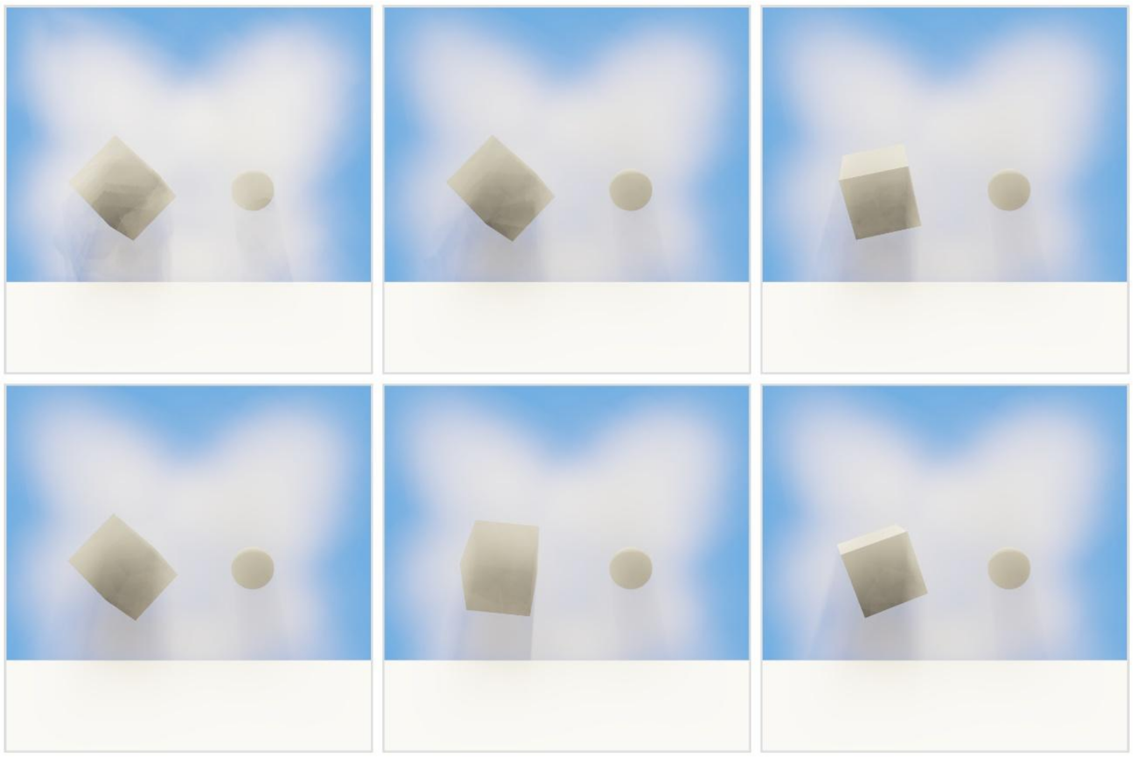


Figure 25: 500 particles rendered with 1, 2, 3, 8, 9, and 10 Simpson's rule steps

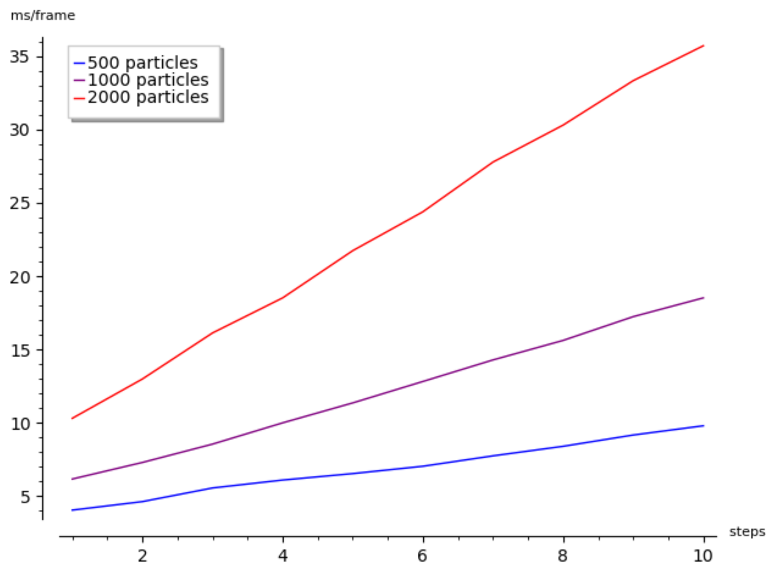


Figure 26: Frame time vs step count for different particle counts

The frame time increases linearly with the Simpson's rule step count, with a higher slope for a larger number of particles. As shown in Figure 25, there is a dramatic visual difference between one step and two steps. Further increases to the step count only offer marginal improvements in visual quality, with the changes being most visible in the volumetric shadows.

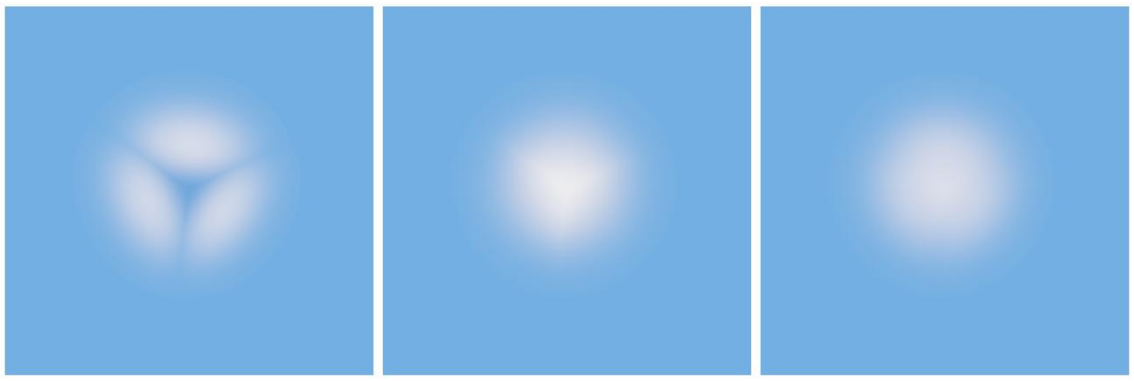


Figure 27: A single particle with u=1 with 1, 2 and 3 Simpson's rule steps. A lower step count results in artefacts due to under-sampling of the extinction

Under-sampling of the extinction function $\sigma_t(z)$ can result in the tetrahedron edges being visible. Figure 27 shows that increasing the Simpson's rule steps causes the rendered image to converge towards the expected result.

4.2.2 Screen resolution

Most of the computation involved in rendering the medium is done in the pixel shader. Therefore, the frame time increases as the screen resolution increases, since there are more pixels to be shaded.

Resolution	**Frame Time**
800 x 600	1.60 ms
1920 x 1080	3.80 ms
2560 x 1600	7.87 ms

Table 1: Frame time at different resolutions

4.2.3 Particle count and camera distance

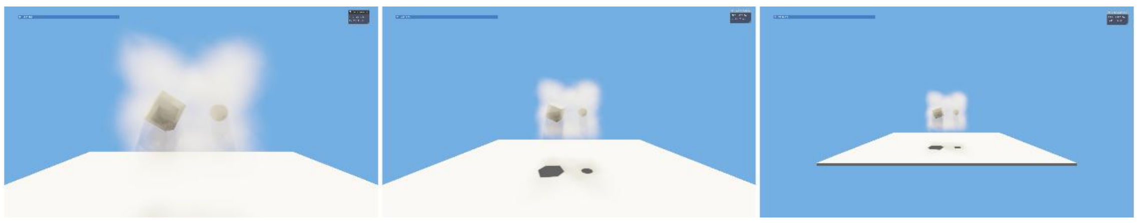


Figure 28: Particles rendered at 20m, 40m and 60m

Figure 29 shows that the frame time increases linearly with the number of particles rendered, with a greater slope depending on the camera distance.

Since the method is pixel shader bound, the frame time also increases as the particles take up more space on the screen. This can be simulated by moving the camera closer to the medium.

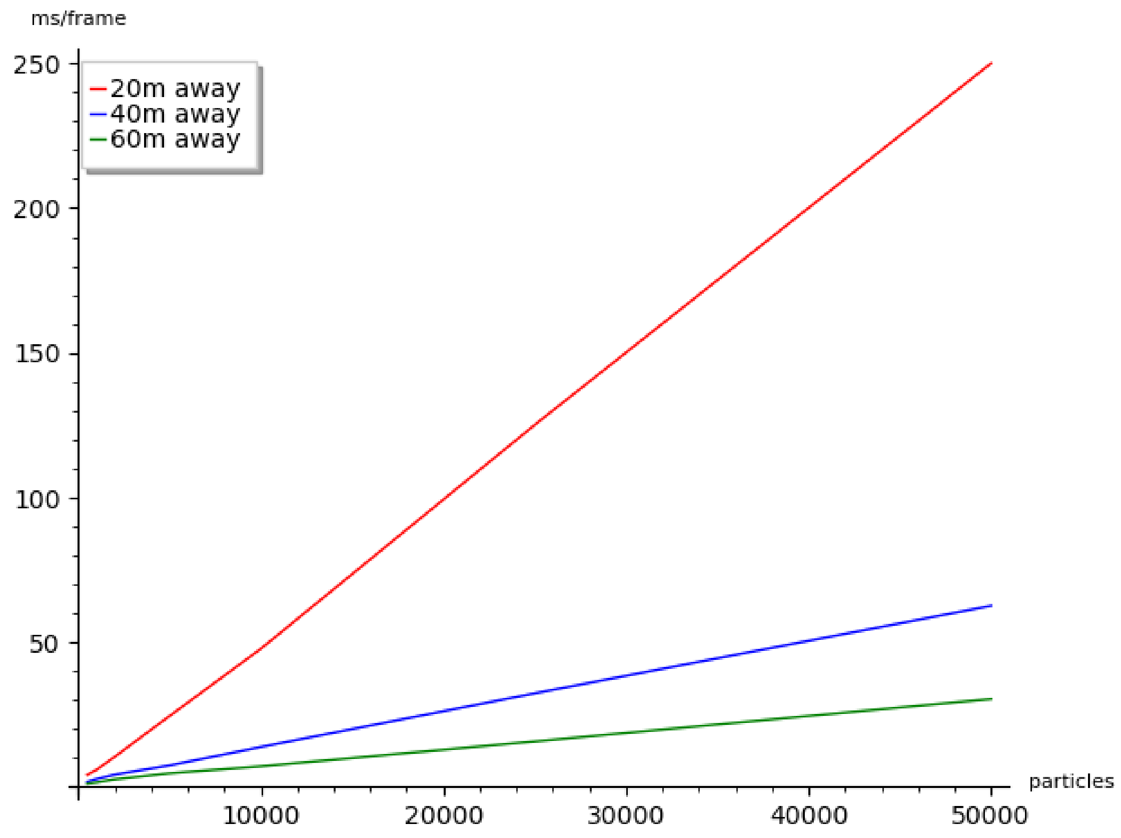


Figure 29: Frame time vs particle count at different distances

4.2.4 Particle size

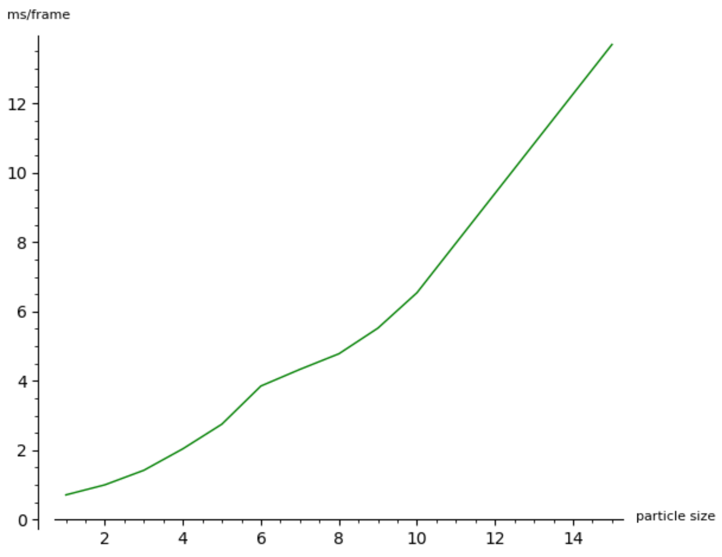


Figure 30: Frame time vs particle size

Increasing particle size also leads to an increase in frame time, since larger particles result in more pixels covered on the screen. From Figure 30, we observe that the frame time increases roughly exponentially with particle size. Note that sizes over 15 did not result in a drop in performance, since at this point the particles intersected with the near plane and were culled as a result.

4.2.5 Large kernel PCF

When sampling the opaque shadow map, the artefact provides the option to use large kernel PCF (Reeves, Salesin and Cook, 1987). HLSL's `SampleCmp` function was used to implement this with a 5x5 grid pattern.
The performance without large kernel PCF in the default configuration is 3.24 ms/frame and increases to 7.63 ms/frame with large kernel PCF. Furthermore, this scales with both particle count and step count.

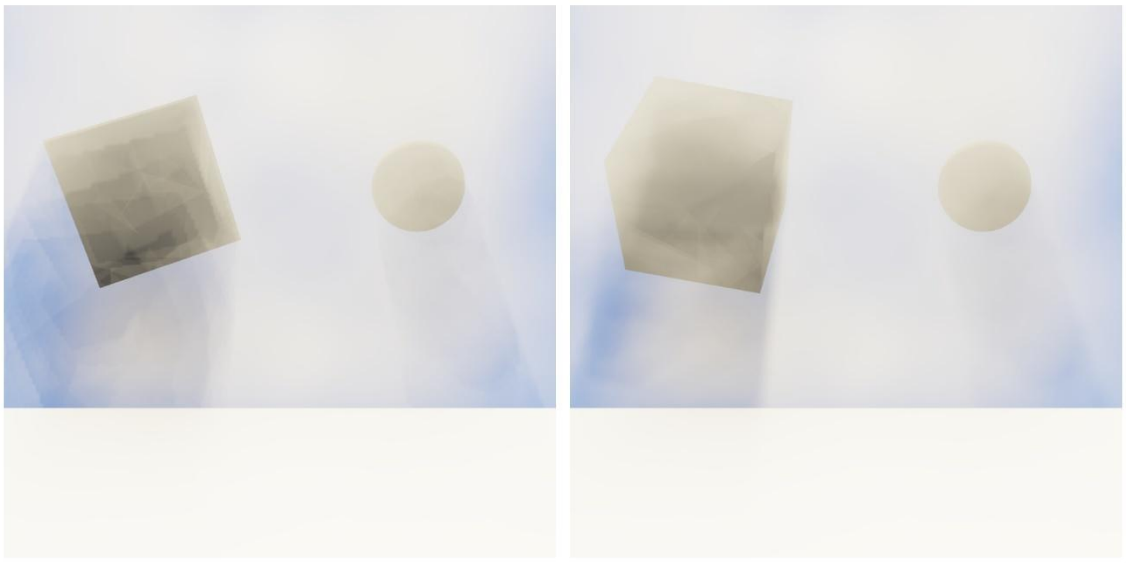


Figure 31: With and without large kernel PCF

Figure 31 shows the medium rendered with and without large kernel PCF. We observe some softening of the volumetric shadows. However, better softening can be achieved at a lower cost simply by increasing the step count. Therefore, we do not recommend the use of large kernel PCF with this method.

4.2.6 Extinction fading

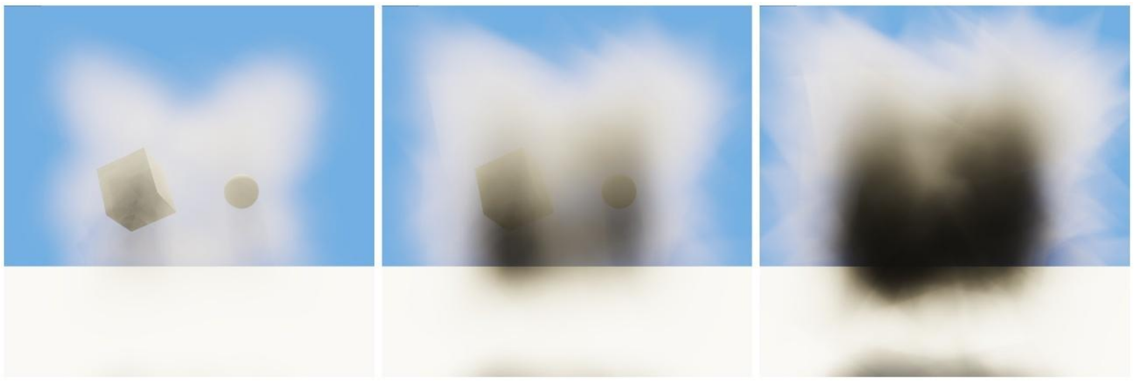


Figure 32: The medium with two Simpson's rule steps and fading factor 1.6, 3 and 10

The extinction falloff is exposed in the UI as an artist-friendly parameter f that ranges from 0 to 10. The actual extinction falloff is computed using $u = particle\ scale\ \times f$.

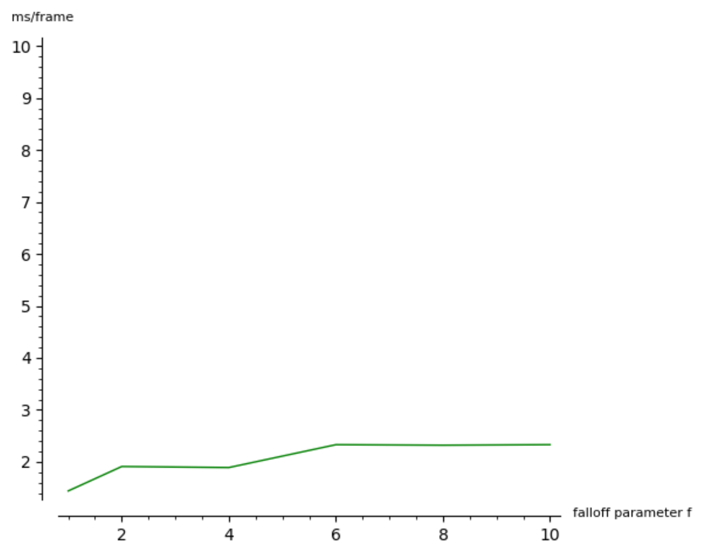


Figure 33: Frame time vs falloff parameter f

From Figure 33, we observe that the falloff parameter does indeed improve performance as it decreases and begins to take effect around the value 6. Through empirical testing, the value $f = 1.6$ was found to eliminate the hard edges for lower extinction values while still making use of as many of the tetrahedron pixels as possible (see figure 32). As the extinction increases, lower values such as $f = 1.2$ are needed in conjunction with a larger step count to eliminate edge artefacts.

4.2.7 Shadow map depth slices

To better illustrate minute details in the shadows, these tests were performed with 5000 particles of size 4 with an extinction of 50 and a step count of 3.

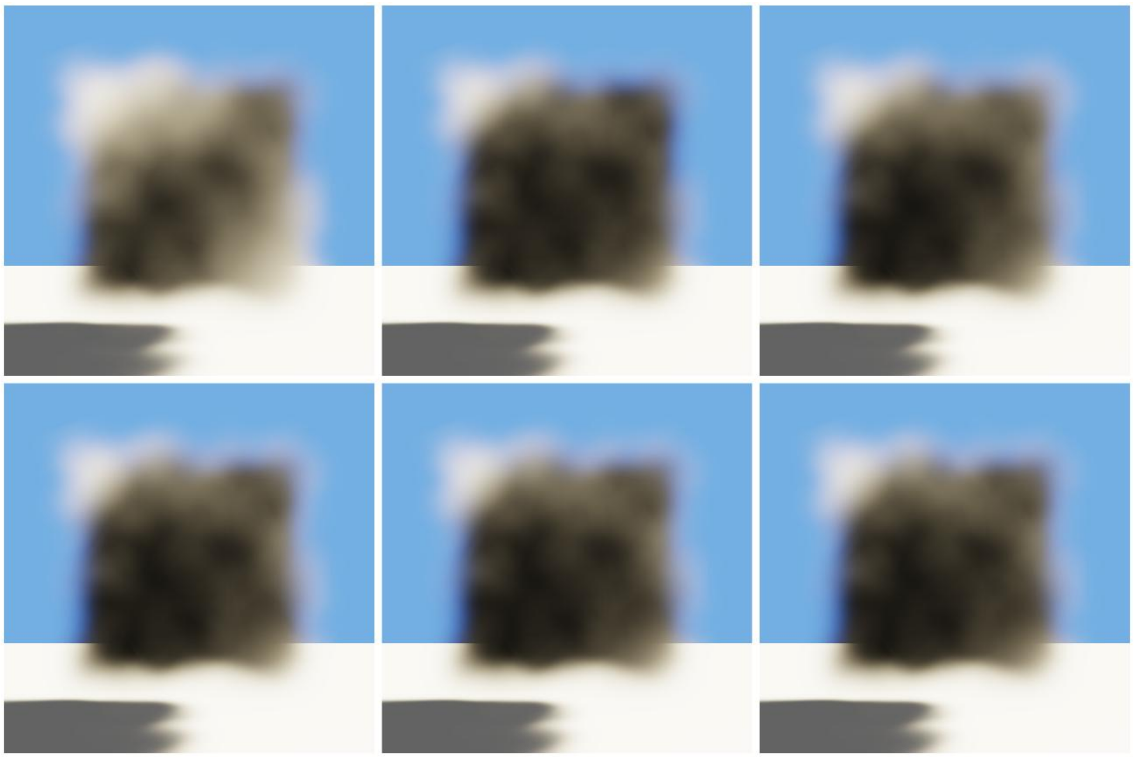


Figure 34: From top, left to right: The medium with 2, 5, 10, 20, 50 and 100 shadow slices

We observe that the appearance and visualisation change visibly as the slices are increased to 20, but further slices do not produce an appreciable difference. Furthermore, the appearances of the medium with 10 and 20 slices are almost identical, although the visualisation has some differences.

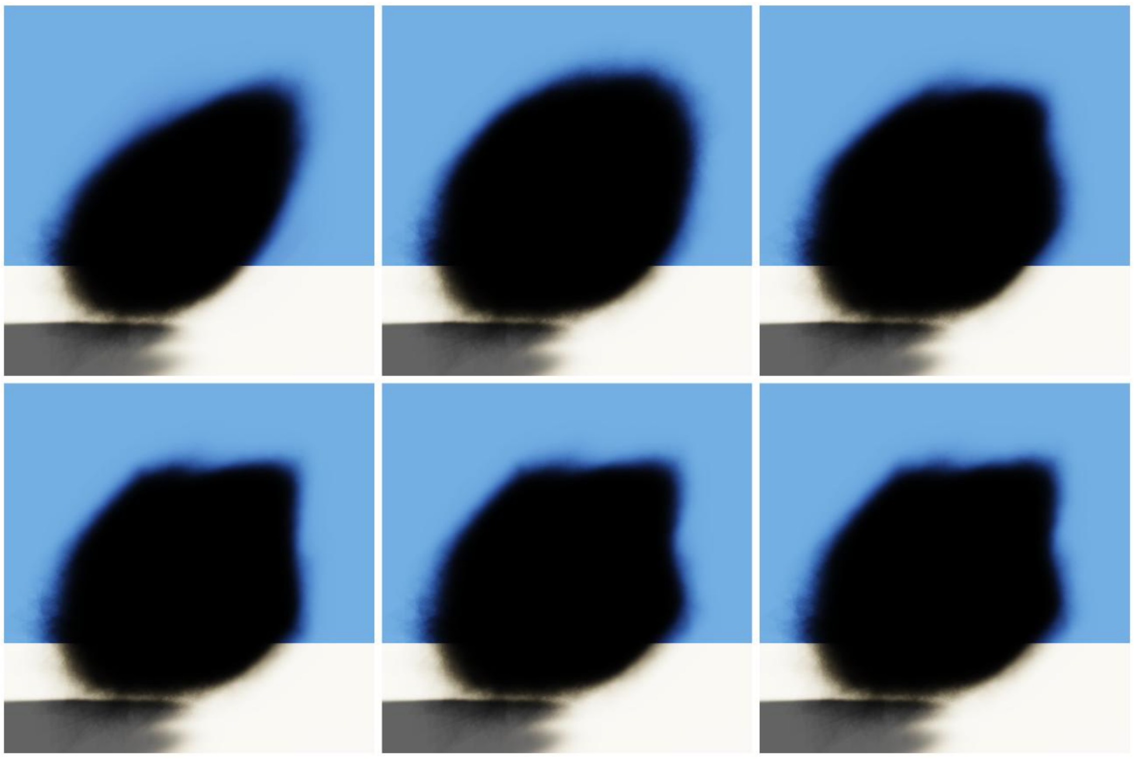


Figure 35: From top, left to right: The volumetric shadows visualised with 2, 5, 10, 20, 50 and 100 shadow slices

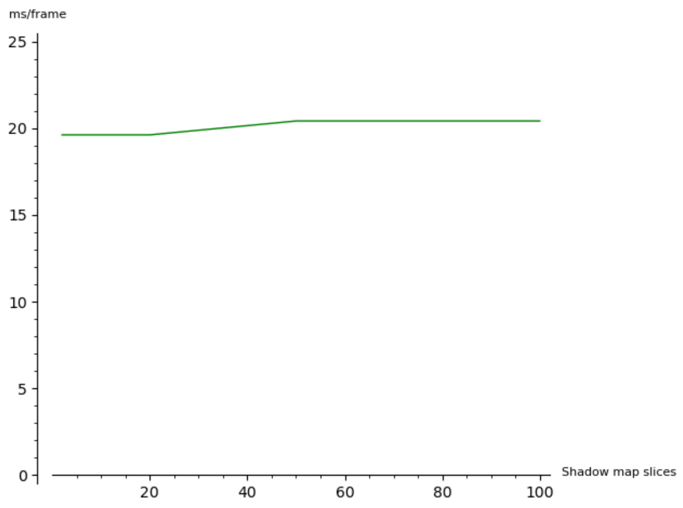


Figure 36: Frame time vs shadow map depth slices

We observe in Figure 36 that the frame time barely increases with the depth slice count. We conclude that a slice count between 10-20 is likely to be optimal for most scenarios.

4.2.8 Shadow map resolution

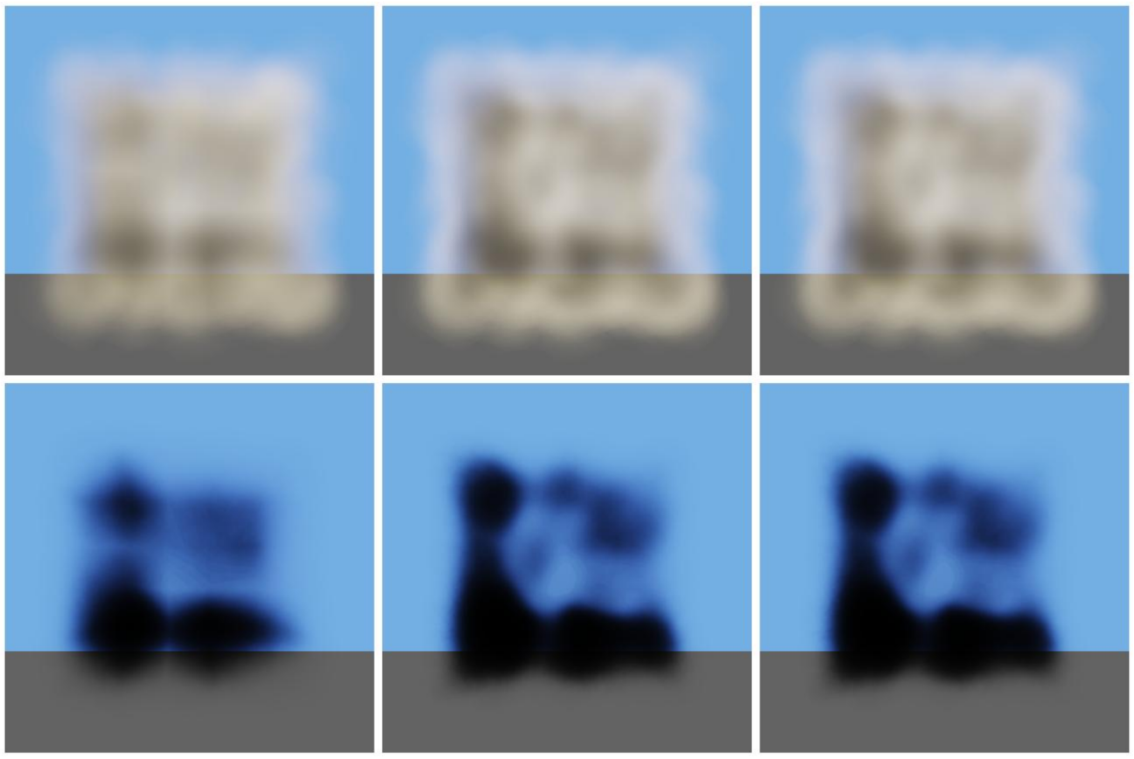


Figure 37: The medium and the volumetric shadow map visualised at various resolutions. From left to right: 32x32, 256x256, 1024x1024

These metrics were captured with 5000 particles of size 4 with an extinction of 15 and a step count of 3. In Figure 37, we observe that there is a significant visual difference between a volumetric shadow map with width 32 vs width 256, but increasing the resolution further does not result in any appreciable changes. Furthermore, volumetric shadow maps with widths lower than 256 result in banding artifacts when the particles are in motion. From Figure 38, we observe that the frame time grows very gradually as the resolution is increased. We conclude that a volumetric shadow map resolution of 256x256 achieves good visual fidelity while minimising frame time and avoiding any aliasing artifacts in motion.

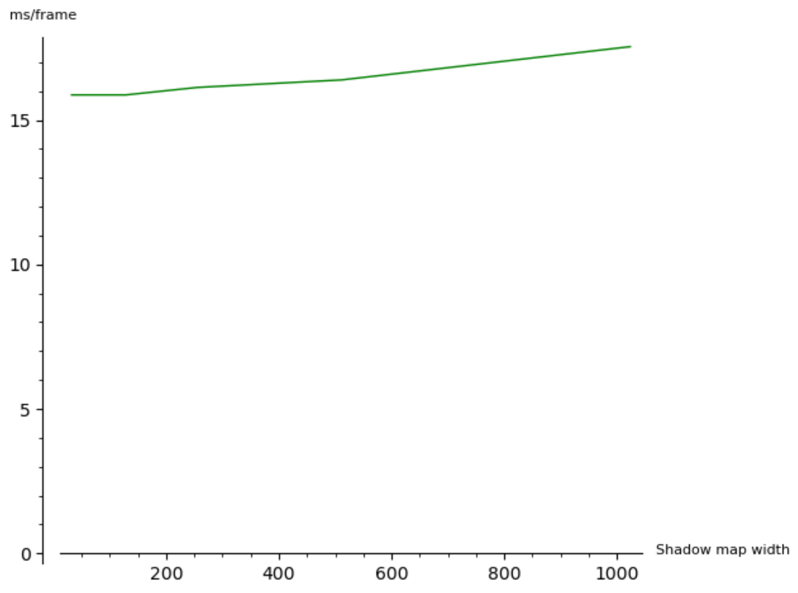


Figure 38: Frame time vs volumetric shadow map width

5 Discussion

5.1 Visual Fidelity

As determined by the qualitative analysis, the method produces plausible visuals, particularly with higher quality settings. Notably, it maintains this fidelity even when the particles, lights or camera are in motion. The lighting model is lacking an emissive component, though this would be trivial to add. Nevertheless, the method is already able to model a variety of participating media such as fog, clouds and smoke.

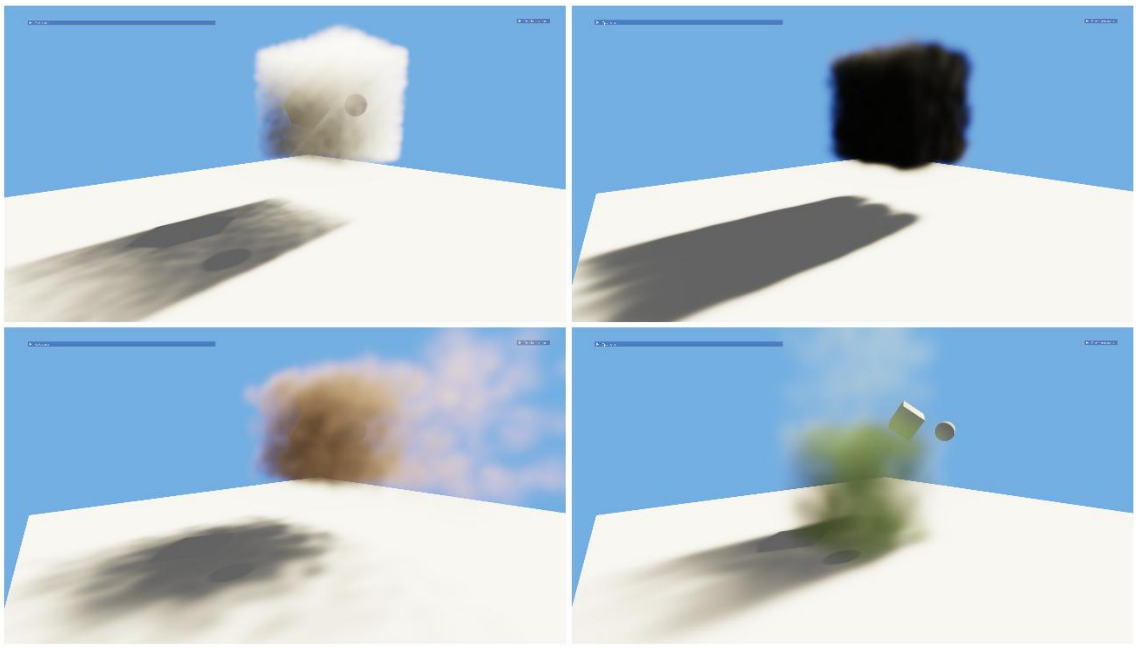


Figure 39: A variety of different media

The visuals are currently somewhat limited by the simplicity of the phase function used. A relatively straightforward enhancement would be to support more complex functions, perhaps implemented as weighted combinations of multiple Henyey-Greenstein lobes. (Gkioulekas *et al.,* 2013) showed that sums of Henyey-Greenstein and von Mises-Fisher lobes can accurately represent scattering in many materials.

Though this was not tested, one theoretical limitation of this method is its inability to interact correctly with other translucent objects. Translucent objects must be rendered in the correct order, but since the particles themselves are translucent, the objects themselves must be sorted together with the particles. For applications where it might be rare for translucent objects to be rendered within the particle system, the bounding box of the particle system could perhaps be used for coarse-grained sorting with the other translucent objects instead.

Another potential limitation is the lack of coloured shadows. However, the greyscale shadow still contributes a lot to the plausibility of the visuals, and if the particles themselves are not coloured, then this method will of course be sufficient.

5.2 Performance

The performance of this method is bottlenecked by the pixel shader, which is invoked once per pixel per particle. We observe that the frame time increases linearly with particle count. One might therefore propose to render a small number of particles to improve performance. However, the particle size may have to be increased as a result to prevent the medium from looking excessively thin. This of course has a negative effect on performance since more pixels are being shaded. Tuning the performance of this method therefore involves carefully increasing the particle size while decreasing the count, or vice versa.

Providing a single representative performance figure is difficult since the frame time depends on the view, the resolution and many configuration settings. The configuration in Figure 40 may provide some guidance. Here, there are 500 particles of size 6 each with a falloff factor $f = 1.6$, rendered with 2 Simpson's rule steps. In the view below, the frame time hovers between 3.6 – 4.0 milliseconds per frame at a resolution of 1920x1080. When the camera was positioned to fill the entire frame with the medium, the frame time was found to not exceed 12.20 milliseconds per frame.

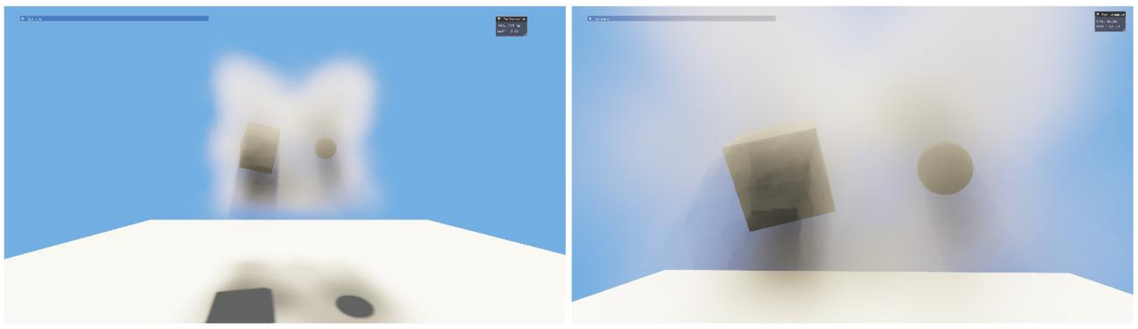


Figure 40: A "typical" configuration and its worst case performance scenario

6 Conclusion and Future Work

6.1 Conclusion

The aim of this project was to assess whether the interval shading technique could be used for volumetric rendering of animated participating media in real-time applications. As demonstrated from the results, it can be argued that the method presented in this dissertation was successful, especially in terms of visual fidelity. The rendered medium satisfied the qualitative characteristics outlined in 3.17 with few exceptions. The maintenance of fidelity under motion of the medium or light source is a significant strength of this method and addresses the limitations of state-of-the-art froxel-based methods. The method is already quite performant, and there is scope for further improvements to be made. The most prominent limitation of this method is its incompatibility with other translucent objects. However, this can be worked around to some extent using coarse-grained sorting with bounding boxes.

6.2 Future Work

Given the performance characteristics presented in 5.2, this method can be used to render participating media in some real-time applications using commodity hardware. However, improved performance would broaden the feasibility of this method, particularly for performance-sensitive applications.

One major inefficiency is the wastage of pixels, as described in 3.11. This method mitigates this through an early return, but further performance gains could be found through making more efficient use of the pixels to begin with. To estimate the potential performance gain, consider the scenario in Figure 41.

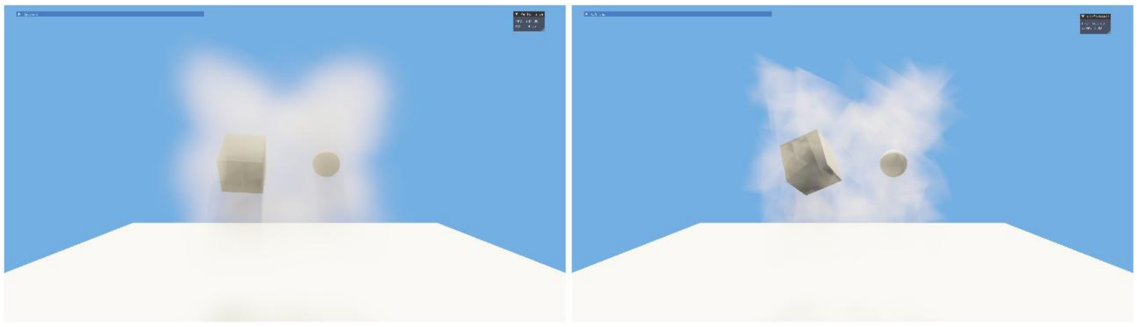


Figure 41: Left: The default configuration vs Right: No extinction fading and size reduction to take up approximately the same volume

On the left, the medium is configured typically as described in 5.2. On the right, the falloff factor f was set to a very high value of 10, effectively disabling the extinction falloff. This resulted in the medium taking up a much larger volume than the original configuration, since the empty space in the tetrahedrons had now been filled. The tetrahedrons were then reduced in size until they appeared to take up approximately the same space as before, resulting in the image on the right. The image on the right therefore approximates the performance characteristics in the scenario where few or no pixels are wasted. The frame time was found to be 1.50ms, which is considerably lower than the 3.6ms frame time for the typical scenario. This suggests that there are large performance gains to be found by reducing pixel wastage.

One way to reduce pixel wastage would be to replace the tetrahedron shape with some proxy geometry that better approximates a spherical shape, such as a square or hexagonal billboard or even a spherical mesh. This would preclude the use of the interval shading technique. However, a shading interval could still be computed in the pixel shader by performing a ray-sphere intersection with a sphere centred at the particle centre. The intersection test may incur some cost, but this would likely be offset by the performance gain from more efficient pixel usage. This method would also allow for early-Z culling to work. Since it doesn't require the use of mesh shaders, this method is compatible with older hardware.

However, the interval shading technique has applicability beyond particle systems. As demonstrated by Tricard himself (Tricard, 2024), tetrahedrons can be arranged to form meshes, which can then be rendered using interval shading. The lighting method presented in this project could be used to perform physically based participating media rendering of such tetrahedral meshes. Care must be taken to ensure that the tetrahedrons are of similar sizes and shapes to avoid sorting issues.

If the spherical proxy geometry approach is not used, occlusion culling using a hierarchical Z-buffer (Greene, Kass and Miller, 1993) can be performed in the mesh shader to avoid drawing particles that are completely occluded.

Another way to reduce the heavy pixel shader cost would be to render the particle system to a lower resolution render target, then composite this onto the final render target. This also offers the opportunity to blur the render target to hide any lingering tetrahedron edges and remove high-frequency details.

There is scope for future work to find a solution to rendering coloured volumetric shadows. One possible solution could be to accumulate the output radiance C_o along the light direction into another 3D texture. This colour could be sampled and used as irradiance when rendering opaque objects.

The current method only supports directional lights and would benefit from extension to support point lights and spotlights. In these cases, both the radiance of the light R and the light vector L are spatially varying rather than constant. Thus, R becomes $R(z)$, L becomes $L(z)$ and the phase function becomes $p(L(z), V)$. The terms $R(z)$ and $p(L(z), V)$ would have to remain inside the integral in Equation 6. These functions would then need to be sampled when solving the integral using Simpson's rule.

The current multiple scattering approximation is not physically based. A more physically based multiple scattering approach could enhance the visual fidelity of this method. Real-time multiple scattering remains a difficult problem in computer graphics, with the most recent work being (Billeter, Sintorn and Assarsson, 2012).

Mixing these particles with other translucent objects also presents a challenging topic for future research. Indeed, a notable advantage of (Hillaire, 2015) are its unified capabilities and its ability to render participating media from various sources together with translucent objects in a scene. Incorporating the ISV method into a unified solution would greatly increase its applicability.

7 List of References

Bauer, F. (2019) 'Creating the atmospheric world of *Red Dead Redemption 2*: a complete and integrated solution', *Advances in Real-Time Rendering in Games, SIGGRAPH 2019 Courses*. ACM, Los Angeles, CA, 28 July – 1 August.

Billeter, M., Sintorn, E. and Assarsson, U. (2012) 'Real-time multiple scattering using light propagation volumes.', *I3D,* 12, pp. 119–126.

Chandrasekhar, S. (1960) *Radiative transfer*. New York: Dover Publications.

Epic Games (no date) *Volumetric Fog in Unreal Engine.* Available at: https://dev.epicgames.com/documentation/en-us/unreal-engine/volumetric-fog-in-unreal-engine#temporal-reprojection (Accessed: 21 Aug, 2025).

Fong, J., Wrenninge, M., Kulla, C. and Habel, R. (2017) 'Production volume rendering', *ACM SIGGRAPH 2017 Courses*, pp. 1–79.

Gkioulekas, I., Xiao, B., Zhao, S., Adelson, E.H., Zickler, T. and Bala, K. (2013) 'Understanding the role of phase function in translucent appearance', *ACM Transactions on graphics (TOG),* 32(5), pp. 1–19.

Greene, N., Kass, M. and Miller, G. (1993) 'Hierarchical Z-buffer visibility', *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '93)*, pp. 231–238.

Guerrilla Games (2017) *Nubis: Authoring real-time volumetric cloudscapes with the Decima engine*. Available at: https://www.guerrilla-games.com/read/nubis-authoring-real-time-volumetric-cloudscapes-with-the-decima-engine (Accessed 21 August 2025)

Harada, T. and Howes, L. (2011) 'Introduction to GPU radix sort', *Heterogeneous Computing with OpenCL.* Morgan Kaufman.

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

Hillaire, S. (2015) *Physically Based and Unified Volumetric Rendering in Frostbite.* Available at: https://www.ea.com/frostbite/news/physically-based-unified-volumetric-rendering-in-frostbite (Accessed: Aug 21, 2025).

Kovalovs, A. (Aug 17, 2020) *Volumetric Effects of The Last of Us: Part Two.* New York, NY, USA: ACM, pp. 1.

Kubisch, C. (2018) *Introduction to Turing Mesh Shaders.* Available at: https://developer.nvidia.com/blog/introduction-turing-mesh-shaders/ (Accessed: 21 August 2025).

Pharr, M., Jakob, W. and Humphreys, G. (2023) *Physically based rendering: From theory to implementation.* 4th ed edn. Cambridge: The MIT Press.

Reeves, W.T., Salesin, D.H. and Cook, R.L. (1987) 'Rendering antialiased shadows with depth maps', *Computer graphics (New York, N.Y.),* 21(4), pp. 283–291 Available at: https://doi.org/10.1145/37402.37435.

Schneider, A. (2023) *Nubis Cubed: Methods (and madness) to model and render immersive real-time voxel-based clouds.* Available at: https://advances.realtimerendering.com/s2023/index.html#Nubis3 (Accessed: 21 Aug 2025).

Tokuyoshi, Y. (2022) *Accurate Diffuse Lighting from Spherical Gaussian Lights (Supplementary Document).* Available at: https://yusuketokuyoshi.com/papers/2022/Accurate_Diffuse_Lighting_from_Spherical_Gaussian_Lights_(supplemental).pdf (Accessed: Aug 21 2025).

Tricard, T. (2024) 'Interval Shading: using Mesh Shaders to generate shading intervals for volume rendering', *Proceedings of the ACM on Computer Graphics and Interactive Techniques,* 7(3), pp. 1–11.

Wronski, B. (2014) *Volumetric Fog: Unified compute shader based solution to atmospheric scattering*. Available at: https://bartwronski.com/wp-content/uploads/2014/08/bwronski_volumetric_fog_siggraph2014.pdf (Accessed: 21 August 2025).