Degree Project in Computer Science and Engineering

Second cycle, 30 credits

Optimizing Light Samples for Real-Time Volumetric Clouds

Through Stochastic Rendering and Noise Reduction

ANDERS BLOMQVIST

Optimizing Light Samples for Real-Time Volumetric Clouds

Through Stochastic Rendering and Noise Reduction

ANDERS BLOMQVIST

Degree Programme in Computer Science and Engineering
Date: June 13, 2025

Supervisor: Christopher Peters
Examiner: Tino Weinkauf
School of Electrical Engineering and Computer Science
Swedish title: Optimering av ljusprover för realtids volumetriska moln
Swedish subtitle: Genom stokastisk rendering och brusreducering

Abstract

Volumetric clouds are popular within real-time rendering applications such as games, as they provide realism to outdoor scenes. Therefore, rendering volumetric clouds at high quality is of interest to enhance user immersion. However, rendering volumetric clouds is known to be computationally demanding due to the high number of samples required per pixel during ray marching. To address this, this thesis investigates how stochastic rendering can reduce the rendering cost while preserving visual fidelity. The thesis focuses on optimizing the positions of a few sets of light samples for large volumes, such as the Walt Disney Animation Studios' Moana cloud at half resolution, by randomly offsetting them and performing temporal and spatial filtering for noise reduction. The goal is to reduce the noise introduced by the stochastic rendering process. Various sources of noise, including White noise, Blue noise, Interleaved Gradient noise (IGN), scalar Spatiotemporal Blue noise (STBN), and scalar Filter-Adopted Spatio-Temporal noise (FAST), are evaluated along with denoising filters such as an Exponential Moving Average (EMA) temporal filter, and Gaussian sigma 1.0, Box 3x3/5x5, and Binomial 3x3/5x5 spatial filters. Performance is assessed through frames per second and image quality, measured using Root Mean Square Error (RMSE). Our results show that FAST noise, designed for specific temporal and spatial filters, consistently yields the lowest numerical error. While spatial filters offer only minor improvements post-convergence, they help reduce error in early frames. The findings highlight the importance of matching noise types to filters and suggest future user studies for perceptual validation in real-time scenarios.

Keywords

Volumetric Clouds, Stochastic Rendering, Sampling, Noise

Sammanfattning

Volymetriska moln är populära inom realtidsrendering, såsom i spel, eftersom de bidrar med realism till utomhusscener. Att rendera volymetriska moln med hög kvalitet är därför av intresse för att öka användarens upplevelse av inlevelse. Renderingen av volymetriska moln är dock känt för att vara beräkningsintensiva på grund av det stora antalet prov som krävs per pixel vid Ray Marching. För att hantera detta undersöker denna avhandling hur stokastisk rendering kan minska renderingskostnaden samtidigt som den visuella kvaliteten bibehålls. Avhandlingen fokuserar på att optimera positionerna för ett fåtal uppsättningar av ljusprover för stora volymer, såsom Walt Disney Animation Studios' Moana-moln i halv upplösning, genom att slumpmässigt förskjuta dem och tillämpa temporal och spatial filtrering för att minska brus. Målet är att minska det brus som introduceras av den stokastiska renderingsprocessen. Olika brusmetoder utvärderas, däribland vitt brus, blått brus, växelvis gradientbrus (IGN), skalärt spatiotemporalt blått brus (STBN) och skalärt filteranpassat spatiotemporalt brus (FAST), tillsammans med brusreducerande filter såsom ett temporalt filter baserat på exponentiell utjämning (EMA) samt spatiala filter som Gaussiskt filter med sigma 1.0, Box 3x3/5x5 och Binomial 3x3/5x5. Prestanda bedöms genom bilder per sekund och bildkvalitet, mätt med Root Mean Square Error (RMSE). Våra resultat visar att FAST-bruset, som är utformat för specifika temporala och spatiala filter, konsekvent ger det lägsta numeriska felet. Även om spatiala filter endast ger små förbättringar efter konvergens, hjälper de till att minska felet i de tidiga bilderna. Resultaten belyser vikten av att matcha brusmetod med filter, och föreslår framtida användarstudier för perceptuell validering i realtidsscenarier.

Nyckelord

Volumetriska moln, Stokastisk Rendering, Provtagning, Brus

Acknowledgments

There are many people who have been supportive throughout my studies and during the work of this thesis. I would specifically like to thank the following people:

- **Christopher Peters**, my supervisor at KTH, for providing me with great feedback and guidance throughout the project. He has also held some of the most interesting courses where I got introduced to the topic of volumetric clouds.
- **Tino Weinkauf**, my examiner at KTH, for good discussions and also great courses, where one laid the foundational work of this thesis.
- **KTH VIC Studio**, for creating an open and friendly environment where you as a student can try things on your own and not be afraid of failing. Learn by doing.
- **Sebastian Gaida**, for sharing his Master's Thesis on OpenVDB rendering for real-time purposes.
- **Alan Wolfe**, EA SEED, for some helpful guidance and also great resources available online at his blog: https://blog.demofox.org/.

Stockholm, June 2025
Anders Blomqvist

Contents

List of Figures

List of Tables

Listings

List of acronyms and abbreviations

EMA	Exponential Moving Average
FAST	Filter-Adopted Spatio-Temporal
FPS	Frames Per Second
GPU	Graphics Processing Unit
HDDA	Hierarchical Digital Differential Analyzer
HDRP	High Definition Render Pipeline
HLSL	High-Level Shading Language
IGN	Interleaved Gradient Noise
RMSE	Root Mean Square Error
RTE	Radiative Transfer Equation
STBN	Spatiotemporal Blue Noise
TAA	Temporal Anti-Aliasing
WDAS	Walt Disney Animation Studios

Chapter 1

Introduction

1.1 Background

Many of the latest real-time applications, such as games, are using volumetric clouds to give their outdoor scenes a believable look and feel. However, to achieve real-time frame rates at a minimum of 60 Frames Per Second (FPS), clouds are generally rendered with less detail than real-life clouds. To render clouds containing more details in real-time, such as the Disney Moana cloud* from Figure 1.1, they have to be optimized further. There are many techniques for optimizing the performance of real-time volumetric clouds. Common techniques are light approximation, lower resolution scale, stochastic rendering, and many more. This thesis will focus on the latter: stochastic rendering. Current stochastic methods for rendering clouds reduce the number of ray marching steps far below what is visually feasible. To recover image quality, the samples are randomly offset by noise and temporally filtered as the rendering cost is amortized across several frames. The visual quality is partly restored, but the performance cost is a fraction of the original. However, the introduced noise remains visible in the final image and should be minimized.

*https://disneyanimation.com/resources/clouds/

Figure 1.1: The Walt Disney Animation Studios Moana cloud rendered with their own in-house physically-based path tracer: Hyperion. Licensed under CC BY-SA 3.0 Unported License.

1.2 Problem Description

To render large and highly detailed volumes, such as the Disney Moana from Figure 1.1, one has to compute how light interacts with the volume across a large space. In previous noise-based volumetric clouds [1, 2, 3], you could get away with good-looking results with less than 10 light samples per step of the ray march. For reference, [1] uses 6, while [4] uses 4. However, these noise-based clouds are less detailed than the Disney Moana cloud, which allows for fewer light samples. As for the Disney Moana cloud, one has to use many more light steps to cover the whole volume at a similar quality.

By drastically reducing the number of light samples per ray, performance is gained at the cost of visual quality. However, a small number of samples will cause visible color banding, which is resolved by randomly offsetting each sample [1, 2, 3]. However, when randomness is added, noise is introduced in the final image. One artifact is removed by introducing another. This raises the question: How can both artifacts be minimized while keeping the performance gain from the low number of samples?

1.3 Research Question

As stated in the problem description (1.2), the objective is to minimize two artifacts: color banding and noise. This can be formulated as a general research question:

> *RQ: How can the position of light samples be optimized in a real-time volumetric cloud ray march renderer to reduce color banding and noise in the final rendered image?*

This general question investigates how the placement of light samples can minimize color banding and noise artifacts. It can be divided into two specific research questions: the first addresses the impact of different noise sources combined with temporal filtering, and the second builds on this by evaluating the effect of spatial filters. Research questions 1.3.1 and 1.3.2 must be addressed to fulfill the objective of this thesis. The noise and spatial filters evaluated are found under Sections 4.4.1 and 4.4.2.

It is not explicitly stated in any of the questions regarding performance, but no noise or filters should affect the performance in any significant way.

1.3.1 RQ1: Temporal Filtering

> *RQ1: By randomly offsetting light sample positions based on noise, what type of noise results in the lowest Root Mean Square Error (RMSE) when temporally filtered over 32 frames through an Exponential Moving Average (EMA)?*

The first question, RQ1, seeks to understand what type of noise: white noise, blue noise, Interleaved Gradient Noise (IGN) [5], scalar Spatiotemporal Blue Noise (STBN) [6], scalar Filter-Adopted Spatio-Temporal (FAST) [7] noise, together with an EMA temporal filter, has the lowest impact on the visual quality of the image, measured in RMSE. In this case, the temporal filter will have more time to converge than in RQ2. However, it is of interest to have as fast convergence as possible, as image quality increases over time due to the properties of stochastic rendering. The temporal filter combines old frames with new ones, and if all frames have unique information (although somewhat related), the combined frame will contain more information than what is possible to render in a single frame. The rendering algorithm does not have time to carefully place the light samples for optimal information, resulting in a reliance on randomness for distributing samples fast enough. As such, this

research question evaluates what type of randomness is most suitable for our rendering scenario and filter.

The evaluation of this question is found in Section 4.5.1, and the answer is given and discussed in Section 5.1.

1.3.2 RQ2: Temporal and Spatial Filtering

> *RQ2: Additionally, for each type of noise, what type of spatial filter results in the lowest RMSE when temporally filtered over 12 frames through an EMA?*

The second question is a continuation of RQ1, aiming to understand what type of spatial filter: Gaussian $\sigma = 1.0$, Box 3x3, Box 5x5, Binomial 3x3, Binomial 5x5, together with each noise type and temporal filter, has the least impact on visual quality. This question also aims to place our algorithm in a realistic use case where the temporal filter does not have time to converge, as that is typically the case in real-time applications. An example of a spatial filter is a blur on an image, causing finer details to disappear. The noise introduced is an example of finer 'detail'.

The evaluation of this question is found in Section 4.5.2, and the answer is given and discussed in Section 5.2.

1.4 Purpose

The purpose of the thesis is to reduce the computational cost of calculating the amount of light that reaches a point within the volume, while keeping the visual degradation low. There are already established techniques for this, such as amortizing the rendering over multiple frames [1, 2, 8, 3]. However, this thesis will specifically focus on reducing the visual artifacts and noise introduced when having a low amount of light samples across several frames. This is of interest for an application that renders large and highly detailed volumetric clouds, as those typically require a lot of steps to capture their details. Performance constraints do not permit a sufficient number of light steps, requiring careful placement of samples to balance performance and visual quality.

1.5 Research Methodology

To begin with, a literature study has been carried out to identify relevant state-of-the-art methods and algorithms. The identified algorithms will be used as a basis for the implementation. Once the implementation meets its requirements, an evaluation will be made based on the methods found in the literature study. The metrics considered are image quality, measured in RMSE, and performance, measured in FPS.

1.6 Scope and Delimitations

The focus of this thesis is to optimize light sample positions; therefore, a state-of-the-art volumetric cloud renderer will not be considered. Instead, a single scatter ray marching algorithm will be implemented in which lighting is approximated using *Beer-Lambert law*. A multiscatter approximation lighting algorithm is not necessary, as the goal of the thesis is to minimize color banding and noise. It would only mask the artifacts as a simpler lighting setup will cause them to stand out more.

For Temporal Anti-Aliasing (TAA), an EMA filter will suffice as an approximation for TAA. Topics such as ghosting, history rejection, and validation, which are key components of a state-of-the-art TAA algorithm [9], will not be considered but are required for a practical application.

Lastly, no formal user study will be conducted to evaluate visual results. The image quality metric RMSE will be calculated against a reference image without color banding or noise to evaluate research questions: RQ1 and RQ2, from Section 1.3.

1.7 Structure of the thesis

Chapter 2 gives an overview of the theory behind relevant topics about real-time volumetric clouds, noise for stochastic rendering, and denoising. Lastly, it outlines some of the related work within the topic of cloud and stochastic rendering. Chapter 3 explains how the methods and algorithms were implemented, while Chapter 4 shows the evaluation and its results of the proposed methods and algorithms. Finally, Chapter 5 analyzes and discusses the results, and Chapter 6 concludes the thesis by summarizing the most important takeaways and future improvements.

Chapter 2

Background

This section will provide the relevant theory behind volumetric clouds together with stochastic rendering. Related work within the area will also be covered.

2.1 Real-Time Volumetric Clouds

The rendering of volumetric clouds aimed for real-time applications uses a similar approach to the work of Kajiya and Herzen [10] introduced in 1984. The method, known as ray marching, involves traversing a ray through space while accumulating the volume's density at each step. The ray originates from the camera and follows the view direction. At each density step, a secondary ray march is also initiated toward the light source. This secondary ray gathers information about how much light reaches this particular point in space. This means that the algorithm does many more light steps than cloud steps. The procedure is illustrated in Figure 2.1 where the *cloud steps* are colored blue, and the *light steps* are colored yellow.

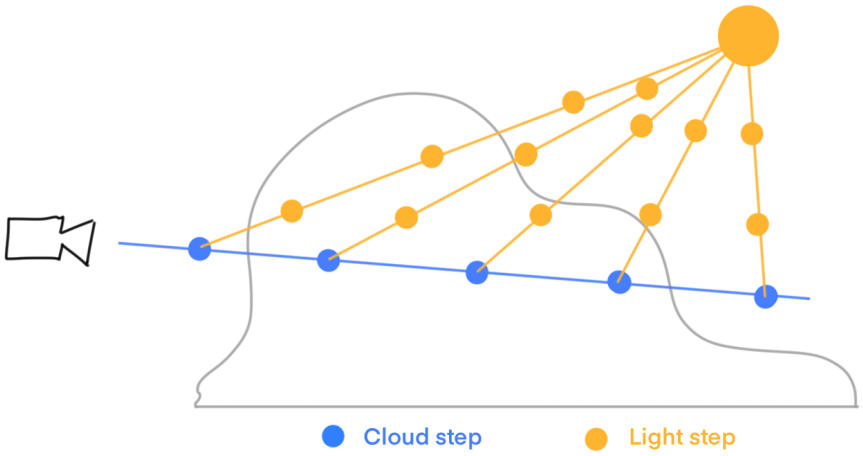

Figure 2.1: Illustration of the steps in ray marching. For each cloud step (blue), light is calculated according to Equation 2.3, which contributes to the final pixel color. However, to calculate light at a given point, a secondary ray march is started towards the light source to sample how much light reaches this point.

To render a volume, such as a cloud, the color of each pixel must be computed based on several physical properties and phenomena. Four different phenomena affect the color of a pixel, causing a positive or negative radiance [11]. These are the four terms, as they are all added together:

1. *Absorption* σ_a: describes the probability that a photon is absorbed by the medium, causing a negative radiance. Physically, the energy is often transferred into heat.

2. *Out-scattering* σ_s: describes the probability that a photon scatters out of the volume once entered, causing a negative radiance.

3. *In-scattering* σ_s: describes the probability that a photon scatters into our view, causing positive radiance.

4. *Emission*: describes the light emitted from the medium, for example, when gases become hot enough, they will start to glow. Causing positive radiance. This will not be considered, as clouds are usually not emissive.

In a single-scattering scenario, the focus is on determining how much light has been *absorbed* and *scattered* at a point in space, $\mathbf{x} \in R^3$. These two phenomena can be mathematically modeled by transmittance and extinction [3].

2.1.1 Extinction

The extinction, σ_t, is the sum of the two probabilities: *absorption* and *out-scattering*. It results in a negative radiance and is defined by Equation 2.1. These two values are configured manually for a desired look. See Table 3.1 for parameter values.

$$\sigma_t = \sigma_a + \sigma_s \tag{2.1}$$

2.1.2 Transmittance

The transmittance $T_r(\mathbf{x_0}, \mathbf{x_1})$ in Equation 2.2 is a function of extinction that models the amount of light absorbed between two points $\mathbf{x_0}$ and $\mathbf{x_1}$ in space [3]. Longer distances or higher extinction lead to less light. This equation is also known as Beer-Lambert's law.

$$T_r(\mathbf{x_0}, \mathbf{x_1}) = \exp(-\int_{x_0}^{x_1} \sigma_t(x)dx) \tag{2.2}$$

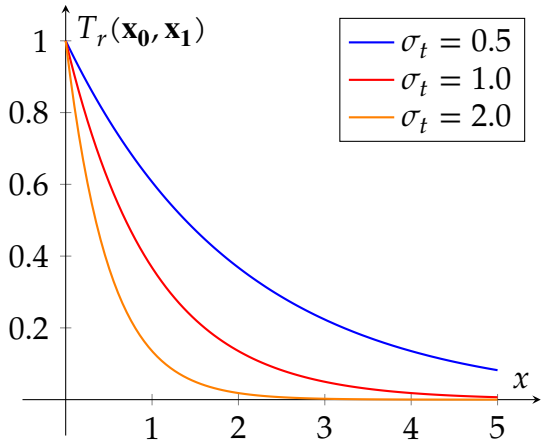

Figure 2.2: Transmittance function as a function of depth between two points in space: $\mathbf{x_0}$, $\mathbf{x_1}$, and extinction $\sigma_t = 0.5, \sigma_t = 1.0$ and $\sigma_t = 2.0$. The further the depth, the lower the transmittance, meaning more light is absorbed. Additionally, a higher extinction leads to less light, meaning the volume is more dense. Figure from [3].

2.1.3 Radiative Transfer Equation

The four phenomena are combined into one equation called the Radiative Transfer Equation (RTE) [12]. It describes the radiance, L, at some point $\mathbf{x} \in R^3$ along a vector $\vec{\omega} \in R^3$, which determines the final pixel color. Equation (2.3) shows a simplified version that does not consider emissive radiation, and since this thesis does not consider multi-scattering, L_{scat} is a constant value. The variable S is the maximum distance along the parameterized ray $\mathbf{x_t} = \mathbf{x} + t\omega$, where $t \in [0, S]$.

$$L(\mathbf{x}, \vec{\omega}) = T_r(\mathbf{x}, \mathbf{x_s})L(\mathbf{x_s}, \vec{\omega}) + \int_{t=0}^{S} T_r(\mathbf{x}, \mathbf{x_t})L_{scat}(\mathbf{x_t}, \vec{\omega})\sigma_s \, dt \tag{2.3}$$

This concludes the most relevant theory for rendering clouds and will serve as a basis for our implementation. For implementation details, see Sections 3.2 and 3.3, which outline the code implemented for Equations (2.2, 2.3).

2.2 Stochastic Rendering

Stochastic rendering is a common technique for approximating the rendering of complex natural phenomena, such as light when it travels through a cloud, among many other things [13]. Let us consider a rendering function $y = f(x)$ where $y \in R^4$ is the color of a pixel and x the input data for that pixel, such as camera position, etc. The rendering algorithm calculates the value of y. Performing an analytically accurate calculation of y can become increasingly complex and time-consuming. Instead, it can be approximated using a limited number of randomly distributed *samples* [14]. A typical rendering scenario is to limit the number of samples per frame, as they are constrained by performance, and amortize sampling over multiple frames [8].

With this approach, complex phenomena can be rendered at real-time frame rates at the cost of image quality. In our case, this technique is essential in allowing us to render large volumes in real-time. However, the cost of image quality needs to be minimized to justify the optimization.

2.3 Noise

Noise refers to a set of random functions that provide a random value for a given input. However, since a function depends on an algorithm, it is virtually

impossible to generate true random numbers on a computer. Therefore, the following functions generate *pseudorandom* numbers, meaning they only appear to be random. This is sufficient for most applications that rely on random numbers, such as stochastic rendering.

2.3.1 White Noise

White noise is a common type of noise as it provides an uncorrelated random number for each input. It follows a uniform probability density function that gives equal weight to all values. As such, it can be considered as random as it can get. An algorithm, such as Marsaglia's Xorshift [15], can be sampled at runtime without any noticeable performance cost, making it easy to use.

2.3.2 Blue Noise

Unlike white noise that has a uniform probability distribution, blue noise is generally referred to as high-frequency white noise, as it is biased towards the higher "blue" frequencies, in terms of color. Blue noise sampling has been shown to improve perceptual image quality compared to uniform sampling, such as white noise [16, 17], as it distributes the samples more "fairly" within small regions of the image space. However, unlike white noise, it has to be precomputed as it can take many minutes to generate.

The term fair in this case refers to how equally samples are distributed over space, illustrated in the bottom left and right images in Figure 2.3. For example, if samples are distributed in a perfect grid across an image, the samples are perfectly fair. However, that causes samples to follow a strict pattern, causing blocky artifacts. While white noise tends to produce clumps and empty areas for small regions, it causes us to miss important information. Blue noise uses a bit of both, where it has some structure but also randomness.

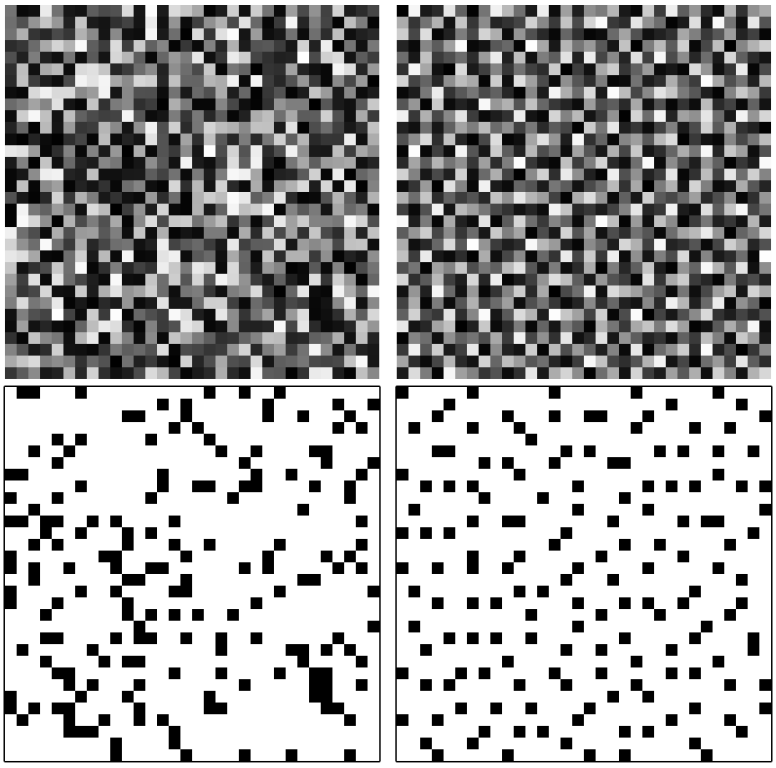

Figure 2.3: 32x32 pixel (0-255) textures of white noise (left) and blue noise (right). The bottom left and right images are same noise textures with a threshold filter of 20. The blue noise threshold image (bottom right) have its black pixels more evenly distributed than white noise (bottom left). In white noise, clumps of pixels and empty voids are more common.

2.4 Denoising

To reduce noise in an image, or *denoise*, it can be processed through different filters. In this thesis, the focus will be on *spatial filter* and *temporal filters.*

A spatial filter is a 2D filter that, for example, affects the pixels in an image. There are many different spatial filters, such as edge detection, sharpening, blur, and many more. For this thesis, blur filters will only be considered. They are low-pass filters, which means that the low frequencies pass through and the highs are removed by the filter, making the image appear *blurry*. They are performed per pixel on a 2D image, typically implemented as a local average over a small neighborhood of pixels [18].

A temporal filter is a 3D filter, compared to the 2D spatial filter, which can be define by a *sequence* represented as $S(x, y, t)$, where x, y are the spatial

coordinates (pixels) and t is time (frame number). The temporal filter modifies the sequence by performing a spatial filter in the 2D domain for every t, thus making it 3D [18].

2.4.1 Box Filter

An example of a spatial filter would be a *Box filter* of radius r at pixel $p = (x, y)$, which assigns the pixel a new value based on the average of its neighborhood $(x \pm r, y \pm r)$, including $p_0 = (x + 0, y + 0)$. It weighs all pixels equally. A radius of 1 results in taking the average over all pixels within a square of size 3x3 with the original pixel in the middle. Figure 2.4 showcases the effect of performing a box blur on a 3 by 3 pixel image. The image contains four white pixels and five black ones, resulting in $(4 * 1.0 + 5 * 0.0)/9.0 = 4/9 = 44\%$ gray.

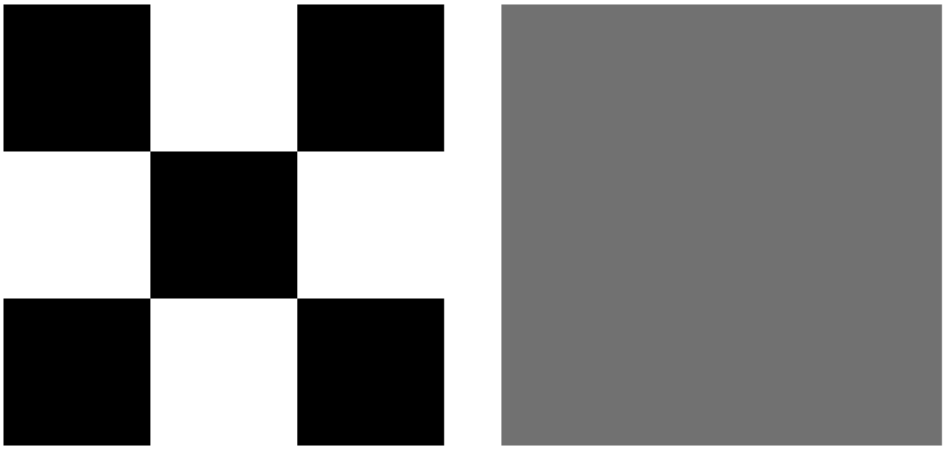

Figure 2.4: Left: 3x3 pixel image with 5 black and 4 white pixels. Right: a box blur of radius 1 performed on the left image, resulting in a single color of 44 % gray. The box filter weighs each pixels equally and can be calculated as an average over all 9 pixels: $(4 * 1.0 + 5 * 0.0)/9.0 = 4/9 = 44\%$ gray.

The blur can be implemented as a single or two-pass, meaning that it is done in a single or two-step process. The single pass is your typical nested for-loop looping over the neighborhood. While the two-step process divides the blur into a single vertical loop, and a single horizontal loop. This causes the time-complexity to go from $O(N^2)$ for a single pass, and $O(2N)$ for two passes, where N is the square side length of the neighborhood [18].

2.4.2 Gaussian Filter

A *Gaussian filter* is implemented very similarly to the box blur but weighs each pixel within the neighborhood according to the Gaussian distribution in Equation 2.4 with a given σ. In this thesis, a $\sigma = 1$ will be used.

$$g(x, y; \sigma) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \tag{2.4}$$

To use this filter, a normalized and discretized Gaussian kernel $g[n, m; \sigma]$ is used as a lookup for each pixel within the square neighborhood [18]. This can also be implemented as a single or two-pass for performance.

Furthermore, the *Binomial filter* is an approximation of the Gaussian filter using binomial coefficients [18]. The binomial coefficients are from the Pascal's triangle, meaning that a kernel of size 3x3 will have the following coefficients:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.5}$$

2.4.3 Exponential Moving Average

An example of a temporal filter is the EMA defined by Equation 2.6 from [9]. It calculates the color, f_n, of a pixel $p = (x, y)$, at frame n by blending the newest sampled color, $s_n(p)$, with its previous color, $f_{n-1}(\pi(p))$, by some factor $\alpha \in [0, 1]$. This is where stochastic rendering is used as it combines two images with similar, but slightly different, information, resulting in an image with more information.

However, as pixels move in time, the previous color is most likely *not* found at pixel p, meaning that its previous pixel position has to be read, given by the function $\pi(p)$. The function $\pi(p)$ *reprojects* a pixel to its previous position (frame $n-1$) [9]. The reprojection function is not relevant for this thesis, as all evaluation scenarios have a static camera with a static environment – meaning no pixels move.

$$f_n(p) = \alpha \cdot s_n(p) + (1 - \alpha) \cdot f_{n-1}(\pi(p)) \tag{2.6}$$

The blending factor, α, is fixed at some value, where a value of 1.0 only considers new samples – resulting in no blending, and vice versa, a value of 0.0 will only render the history. A fixed alpha value of 0.1 will result in 19 effective

samples according to Figure 4 in [9], which means that the newest frame has been blended with the last 19 frames. Thus, using at least 19 different sample positions over the last 19 frames amortizes the rendering cost across those frames, enabling higher frame rates.

2.5 Related work

This section covers some of the relevant work before this thesis.

2.5.1 Volumetric Data

The volume itself is represented as a scalar field that contains single floating-point values, typically ranging from $[0-1]$. For real-time volumetric clouds, the common way to represent the data is through 3D textures based on noise, which was popularized by Schneider for the game *Horizon: Zero Dawn* [4].

Another way of representing volumetric data is through voxels. A voxel is a 3D cube in space containing some value, such as density. However, as it requires many voxels to accurately define a 3D model, there has to be optimization data structures in place. One such data structure is OpenVDB [19]. Additionally, OpenVDB has a scaled-down version of VDB called NanoVDB [20], which is designed for Graphics Processing Units (GPUs) – allowing for even faster voxel access. Using NanoVDB enables us to render large voxel models such as the Disney Moana cloud in real-time, as shown by [21].

The work of Sebastian Gaida [21] goes into depth on rendering clouds based on NanoVDB volumes. Some key findings regarding optimization are used in this thesis as well, such as the use of the Hierarchical Digital Differential Analyzer (HDDA) algorithm by [22]. In short, the algorithm takes advantage of the sparse VDB data structure, allowing us to quickly skip empty space. Gaida [21] found that using HDDA for the first hit resulted in the lowest frame time, compared to traversing the whole volume with HDDA. The first hit acts as a search from the camera position to where the cloud begins, from there, traditional ray marching was used with a small step size of 1.0 or 2.0.

2.5.2 Rendering Clouds

Rendering clouds involves solving equations containing integrals that approximate the scattered light between two points. The longer the distance between the points, the less accurate the calculations will be. However,

increasing the number of samples is not an option due to performance. To get more accurate results across long distances, Sébastien Hillaire [3] solves this by introducing an analytical integration method:

$$\int_{x=0}^{d} e^{-\sigma_t x} \cdot S \, dx = \frac{S - S \cdot e^{-\sigma_t x}}{\sigma_t} \tag{2.7}$$

The integral calculates transmittance, as Equation (2.2), however, the transmittance depends on the scattered light, S, of the current sample. Hillaire [3] solves this by transforming the integral to a division where the scattered light can be calculated first. The implementation of this analytical integration is shown in listing (3.3) from Section 3.3.2. This is beneficial for this thesis as the goal is to reduce the number of samples, causing longer step sizes, while keeping similar image quality to a high step count.

2.5.3 Stochastic Rendering

Previous work from Toft et al. [2] and Högfeldt [1] has found that reducing the number of ray marching steps, applying a random offset to their starting positions (jitter), and finally performing TAA can greatly reduce the render time of noise-based volumetric clouds while still achieving almost the same visual result. However, the noise from the jitter is still visible in the final rendered images and should be minimized. It is important to note that both Toft et al. and Högfeldt apply their jitter to the starting position of the initial ray march and not the light steps, as this thesis will focus on.

A smarter policy on how samples are randomly positioned can minimize noise in the image. Toft et al. [2] and Högfeldt [1] randomly offset by blue noise, which is favorable for optimizing towards perceptual image quality [16]. However, Wolfe et al. [6] show that when sampling independent blue noise textures over time, it becomes white noise instead, thus losing the perceptual quality. They provide another type of noise called STBN, which stays as blue noise over time.

Furthermore, Donnelly et al. [7] go into more depth than Wolfe et al. and show that noise can also be optimized for what types of filters are used both spatially and temporally. However, it is still unclear what combination of noise/spatial filters yields the lowest amount of noise in a real-time volumetric cloud renderer, which is what this thesis aims to investigate.

2.5.4 Evaluation

Finally, the evaluation process is similar to Wolfe et al. [6] and Donnelly et al. [7] as our work share many similarities. Both measure RMSE over multiple frames: 32 in [7], and up 128 in [6]. However, [7] highlights the properties of an EMA with fixed alpha at 0.1, resulting in 19 effective samples, which can be fully reached across 32 frames. Additionally, [7] limits the temporal filter to 12 frames to account for the fact that convergence is rare within real-time applications.

Chapter 3

Implementation

This chapter describes the implemented volumetric cloud renderer and its relevant features in addition to why certain methods and algorithms were chosen and how they were implemented.

3.1 Overview

The volume renderer was implemented in Unity 6.0 High Definition Render Pipeline (HDRP)*. Unity was chosen as it was well known to the researcher, there were no technical motivations and the following implementation is capable of being engine agnostic. However, it depends on the NanoVDB C++ library. This implementation closely follows the work of [21] where the PNanoVDB.h header file is used to interact with the NanoVDB API through the DirectX 12 graphics API. The implementation, with all of the code and raw data, can be found on Github†.

The volume renderer can be divided into three main parts that contribute to the final rendered image. These parts are called *passes* and are listed below. The passes are executed in the order they are listed in.

1. **Volume Pass.** Renders the volume to a texture, S_n, with ray marching, where several input parameters are taken into account.

2. **Temporal Pass.** Receives the texture, S_n, and performs an EMA temporal filter. Outputs a new blended texture B_n.

*https://unity.com
†https://github.com/andersblomqvist/master-thesis-renderer

3. **Spatial Pass.** Performs a blur on B_n before showing the result to the camera.

The three passes are executed every frame with the exception of RQ1 (Section 1.3.1), which does not perform the spatial pass. During evaluation, the final frame is stored to disk from a render texture, along with the ground truth reference, before being processed through a separate RMSE calculation program. The RMSE calculator can process multiple frames and be easily automated with bash scripts. The final result is a text file containing the RMSE values for each frame, which are used to evaluate and answer RQ1 and RQ2 from Section 1.3.

Apart from the volume renderer, Figure 3.1 illustrates the whole implementation on a high level, including how scenes, noise, and filters are connected to the research questions. There are two scenes as the evaluation is done on two datasets, the Disney Moana cloud, and a dense cube. Details regarding datasets and scene configurations are found in Sections 4.1 and 4.2. Before the Volume Pass (Sections 3.2 and 3.3) is executed, the source of randomness (noise) is selected among five types listed in Section 4.4.1. Depending on the research question, the implementation continues in either of two paths: one for RQ1 (Section 1.3.1) and the other for RQ2 (Section 1.3.2). The Temporal Pass (Section 3.5) accumulates information using F_{max} (Equation 3.1) unique random numbers, which is 32 for RQ1 and 12 for RQ2. In both cases, 32 frames are rendered, meaning that RQ2 is reusing its 12 random numbers. The Spatial Pass (Section 3.6) is executed for RQ2 using one of the filters listed in Section 4.4.2. Finally, the two paths output an image from which the RMSE will be calculated.

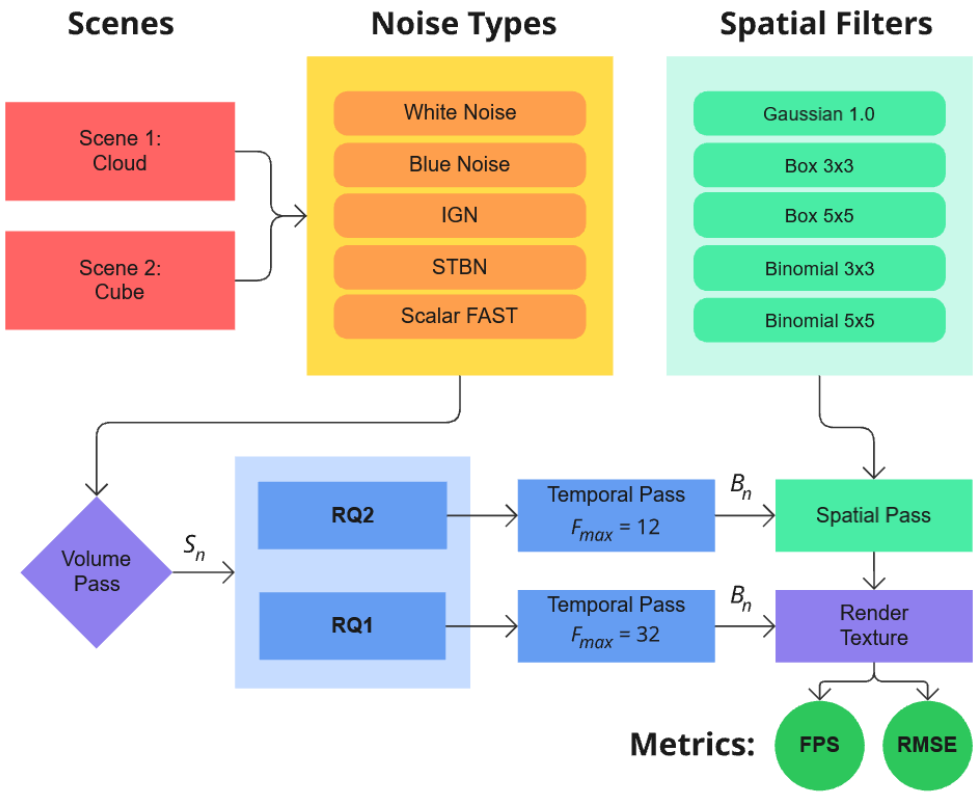

Figure 3.1: An illustration of the implementation going from scenes to metrics with the most important steps in between. A scene consists of a camera, directional light, and a volumetric dataset. It is rendered using the Volume Pass (Section 3.2 and 3.3) using one of the noise types and outputs a texture S_n. Two paths are taken depending on which research question is evaluated. For RQ1, only a Temporal pass (Section 3.5) is executed that outputs texture B_n, while RQ2 adds another Spatial pass (Section 3.6) using one of the spatial filters. The final output is an image from which the RMSE will be calculated.

3.2 Volume Pass: Ray marching

The foundation of the volume renderer is the first pass that performs the ray marching algorithm described in Section 2.1. It consists of two loops, the topmost loop for the *cloud steps* and an inner loop for the *light steps* as shown in Figure 2.1. At the end of each cloud step iteration, the light is calculated and added to the final pixel color. The general algorithm follows the example in listing 3.1. The function `HDDAFirstHit` on line 2 is an optimization described in Section 3.2.1.

Listing 3.1: General Ray Marching Algorithm

```
1 function RayMarch(Ray):
2     HDDAFirstHit()
3     for each cloud step:
4         SampleDensity(pos)
5         SampleLight(pos)    // 2nd ray march
6         Calculate light
7     return color
```

For each pixel, the ray begins at the camera world position and travels along the view direction. Note that no jitter is applied to the starting position of this ray, as is done in [1, 2]. Important parameters related to the ray marcher are listed in Table 3.1.

3.2.1 Optimizations

Two optimizations from [21] have been implemented. The first technique is to use the HDDA [22] algorithm that utilizes the OpenVDB data structure to skip empty space. In our case, it acts as a search for where the volume begins, and that is where the traditional ray marching starts.

The other technique is to skip empty space during the traditional ray march, meaning that the current loop iteration is skipped and continues to the next one. This is something every ray marcher does, as it is pointless to sample empty space, but once more, the OpenVDB data structure can help. The data structure contains metadata on the dimension of a voxel, and a dimension larger than 1 indicates empty space. Additionally, an assumption can be made that there is likely a lot of empty space in the local region, meaning that a larger step can be made without risking overstepping. In this case, the step size was set to 10 for this specific iteration. In the case of a lower density than the threshold, see *min density* in Table 3.1, the step size is not modified.

3.3 Volume Pass: Lighting

The lighting is evaluated for every pixel with a volume density greater than the *min density* threshold. Recall from Section 2.1 that a secondary ray march is started to gather information on how much light reaches the current position.

3.3.1 Sampling Light for Shadows

The secondary ray march shoots a new ray from the current position of the first ray towards the light source, in this case, the sun. This ray march uses an exponential step size of 2, resulting in more samples closer to the point and fewer samples farther away. In [21], this is called *log ray stepping*.

The algorithm is described in detail in listing 3.2, which is more or less the exact High-Level Shading Language (HLSL) code. The `sample_pos` is offset by a per-pixel random scalar (float) along the ray direction, which is also scaled by the `step_size`, shown on line 8.

Listing 3.2: Detailed Light Sample Algorithm

```
1  function SampleLight(pos):
2    float3 shadow = 1.0
3    float step_size = 1.0
4    int step = 0
5    float jitter = SampleNoise()
6    while step < LIGHT_SAMPLES
7        sample_pos = pos + step_size * LIGHT_DIR
8        sample_pos += jitter * step_size * LIGHT_DIR
9        d = SampleDensity(sample_pos)
10       d *= LIGHT_DENSITY_SCALE
11       if d < MIN_DENSITY
12           step++
13           step_size *= 2
14           continue
15       shadow *= exp(-d * step_size)
16       step++
17       step_size *= 2
18   return shadow
```

The ground truth reference image does not use the `SampleLight()` function from listing 3.2, instead, it uses a uniform step size across the whole ray with no jitter – and many more samples. Otherwise, they are alike. See Table 3.1 for details regarding the number of samples and step size.

3.3.2 Beer-Lambert Law

The transmittance Equation (2.2) from Section 2.1.2 is calculated at two occasions, the first one during the light sampling, and the other when evaluating the final color of a pixel. Listing 3.2 shows the first occasion on line 15, where the light gets exponentially reduced depending on density and step

size. It is assumed that the density is constant between two sample positions. This is a simplification as it is not true.

The second occasion is during the analytical integration method by [3], which is done at the end of each *cloud step*. A detailed example of the previous ray march listing 3.1 is shown below as listing 3.3, which highlights the light calculation. The variable `sigmaT` is from the extinction Equation (2.1) based on values in Table 3.1. This is the part where the RTE (2.3) is calculated for each step.

Listing 3.3: Detailed version of the main ray march algorithm highlighting the light calculation.

```
1  function RayMarch(Ray):
2    ...
3    float3 color = (0,0,0)
4    float transmittance = 1.0
5    for each cloud step:
6        ...
7        float d = SampleDensity(pos)
8        float3 S = SampleLight(pos)
9        float3 Sint = (S-S*exp(-d * sigmaT * step_size))
10         / sigmaT
11       color += transmittance * Sint
12       transmittance *= exp(-d * sigmaT * step_size);
13   return color
```

3.4 Noise

Random numbers are sampled from noise that is stored as textures or generated at runtime. All noise textures are 128x128x32 in size where a pixel $p = (x, y)$ at frame t reads the noise texture at

$$(x \bmod 128, y \bmod 128, t \bmod F_{max}) \tag{3.1}$$

The variable $F_{max} \in [1, 32]$ controls the total number of frames considered when temporally filtering. The random numbers are within the range of $[0, 1]$ of single float types. The textures are sampled through the red channel only. The random number is stored in the `jitter` variable on line 5 in listing 3.2. This variable randomly distributes samples along the ray depending on the type of noise used, as seen on line 8 in the same listing 3.2. Figure 3.2 showcases the effect of applying a random offset to the light

sample positions. The top right half, where the jitter is turned off (`jitter = 1`), has clear color banding artifacts, whereas the lower left half has traded those artifacts with noise instead. Remember that the purpose is to trade color banding with noise, and then choose a set of noise and denoise filters with the lowest numerical error.

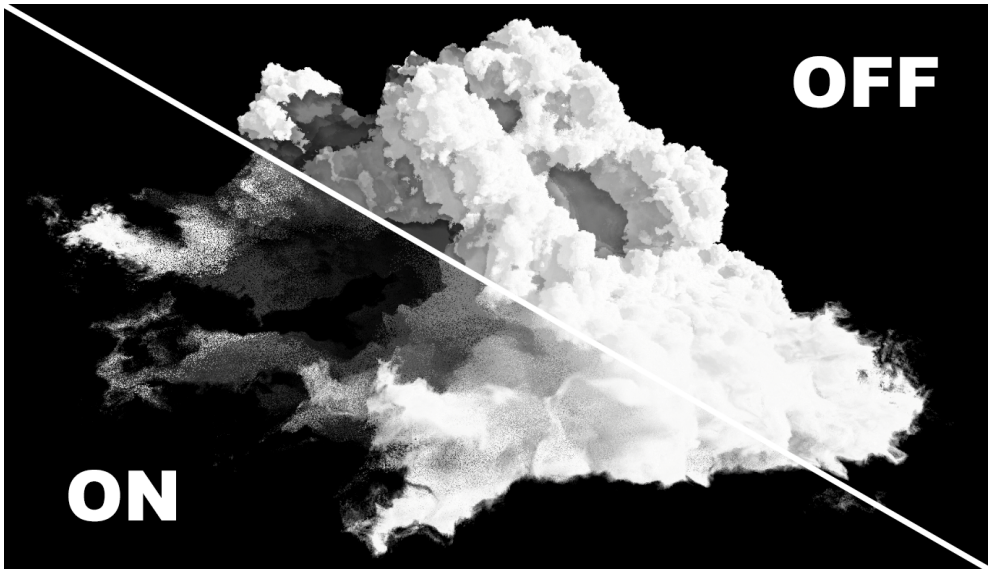

Figure 3.2: Comparison between randomly offsetting the light sample positions or not. The ”OFF” half has no random offsets, causing visible color banding, whereas the ”ON” half uses white noise to randomly offset samples, causing no color banding, but noise instead. No temporal or spatial filtering is used in this example.

3.5 Temporal Pass

The second pass in the render pipeline is the temporal pass. It blends two textures by applying the EMA Equation (2.6) for each pixel. These textures are the last rendered frame and the *frame history*. The *reprojection* function $\pi(p)$ does two transformations on the **uv** coordinates as shown by Equations (3.2, 3.3). The **VP** is the view projection matrix, and the $\mathbf{VP}_{prev}$ is the previous frame's view projection matrix. These two matrices are provided by Unity. The first Equation (3.2) transforms the **uv** coordinates to world space, hence the *wp* subscript, and the second Equation (3.3) transforms it back to screen space again with the previous view projection matrix. However, as all evaluation is done with a static camera and a static scene, the reprojection

step can be discarded. But it was still implemented as a quality of life feature making it easier to interact with the implementation.

$$\mathbf{p}_{wp} = \mathbf{VP}^{-1} \cdot \mathbf{uv}, \tag{3.2}$$

$$\mathbf{uv}_{prev} = (\mathbf{VP}_{prev} \cdot \mathbf{p}_{wp})_{xy} \tag{3.3}$$

As for the α, the filter uses a constant of 0.1, as also done by [1, 7]. The code is straightforward and outlined in listing 3.4.

Listing 3.4: Detailed version of the main ray march algorithm highlighting the light calculation.

```
function TemporalPass(
  uv, uv_prev,
  tex2D(NewSample),
  tex2D(FrameHistory)
):
  float4 history = sample_tex2d(FrameHistory, uv_prev)
  float4 new_sample = sample_tex2d(NewSample, uv)
  float alpha = 0.1
  return alpha * new_sample + (1 - alpha) * history
```

The final step is to save to the frame history. It is implemented as another pass that copies the blended result to the frame history texture.

3.6 Spatial Pass

The spatial pass receives the blended texture from the temporal pass and performs a per-pixel blur. The result is shown directly to the camera, making this the final step in the render pipeline. No other image processing is done.

Both the box and Gaussian blurs are implemented as a single pass, causing their time-complexity to reach $O(N^2)$ instead of a two-pass blur at $O(2N)$, where N is the size of the kernel. The performance cost of doing a filter with a 5x5 kernel ($N = 5$) was about 3-4 frames per second on the hardware listed in Section (4.3), compared to no spatial filter.

3.7 Parameters

Table 3.1 lists important parameters and their values related to the implementation. These are constant throughout the evaluation. Notice that

there are two density scales, one for the *cloud steps* and the other for the *light steps*, and the *light density scale* is much lower. By having a higher *cloud density scale*, the ray march algorithm accumulates faster, resulting in fewer steps taken. However, to control the appearance of the volume, such as lower or higher density, a secondary density is used for the light. Recall the Beer-Lambert law in Equation (2.2) that shows the exponential decay of transmittance over distance. The graph has a max distance of 5, which is very small in relation the volumetric data sets, meaning our σ_t has to be much lower, as distances are in the hundreds.

Parameter name	**Value**
Absorption σ_a	0
Scattering σ_s	(1,1,1)
Step size	2
Cloud density scale	1.0
Min density	0.01
Min transmittance	0.05
Light samples	10
Light step size	x2
Light density scale	0.05
Ground Truth	
Light Samples	256
Light step size	2.5

Table 3.1: Rendering parameters for the volume pass ray marching and their values. The x2 means the light step size is doubled each iteration. The *Ground Truth* uses a uniform light stepping where 256 samples are taken for every light ray.

Chapter 4

Evaluation

This chapter presents details regarding the evaluation of the implementation and the data collected. Several experiments have been conducted where image quality, measured in RMSE, and performance, measured in FPS, have been recorded. The evaluation is split into two parts, one for each research question RQ1 and RQ2 (see Sections 1.3.1, 1.3.2). For both parts, the same datasets, scene configurations, and hardware were used – details regarding those are found in Sections 4.1 through 4.3. The experiments conducted are described in Section 4.5 among their results.

4.1 Volumetric Datasets

The popular Moana cloud* dataset from Walt Disney Animation Studios (WDAS) is used to evaluate the implementation. The dataset was also used by [6, 21]. It comes in several different resolutions, the highest one is 2.8 GB, making it too large as the maximum buffer size is set to 2.0 GB by the graphics API. Therefore, the half-resolution model is used, which is scaled down by two, making it roughly 0.6 GB in size.

In addition to a real use-case dataset, such as the WDAS cloud, a simpler dataset, created by the author, was also used to evaluate the implementation. The dataset is a dense volumetric cube generated in the free 3D modeling program Blender v4.1†. It was created from a standard cube mesh of 1x1x1 scaled up 400 times and converted into a volume with the *Mesh to Volume* modifier. The parameters are shown in Figure 4.1. A scale of 400 made the cube have a similar world space size as the WDAS cloud, meaning that the

*https://disneyanimation.com/resources/clouds/
†https://blender.org

parameters from Table 3.1 can be reused. The dataset was exported as a VDB file. This dataset was created to be opposite to the WDAS cloud, as the cube is perfectly smooth while the cloud is more chaotic.

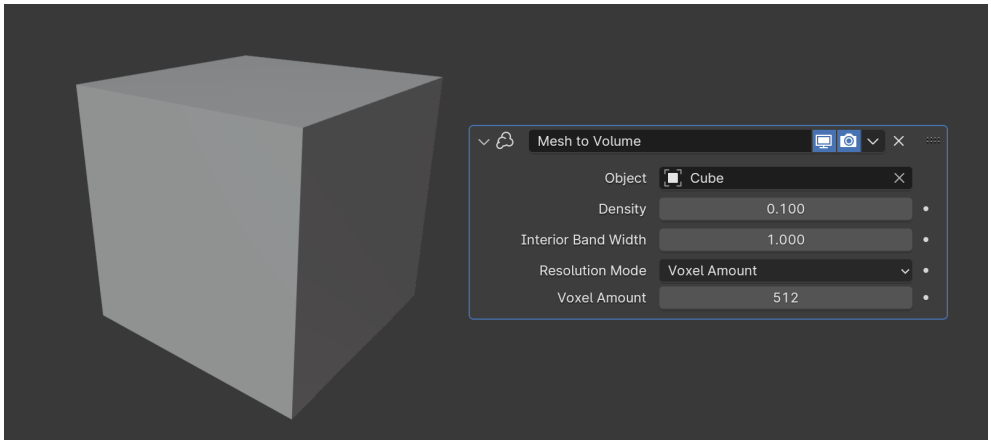

Figure 4.1: Screenshot from Blender showcasing the cube model and parameters for generating a volume with the Mesh to Volume modifier. The voxel amount is high and density low, resulting in a smooth cube with no sharp edges.

4.2 Scene Configurations

Two scenes were created to host the experiments: *Scene 1: Cloud* and *Scene 2: Cube*. As the names suggest, the first scene uses the Disney Moana cloud as its dataset, while the second has the dense cube. Each scene consists of a camera, directional light (sun), and the volumetric dataset. The camera and sun angle were placed manually by the author in a position appropriate for capturing various conditions, such as low, mid, and high amounts of light passing through the volumes. For both scenes, one or more 'regions of interest' are specified that attempt to highlight different lighting scenarios.

4.2.1 Scene 1: Cloud

In *Scene 1: Cloud*, the sun angle was set to 10° above the horizon, making it almost horizontal with long shadows. The camera angle was placed on the side to capture when the light both hits and exits the volume. The rendered result is shown in the large image in Figure 4.2. The squares 1-4 indicate the regions of interest for this scene.

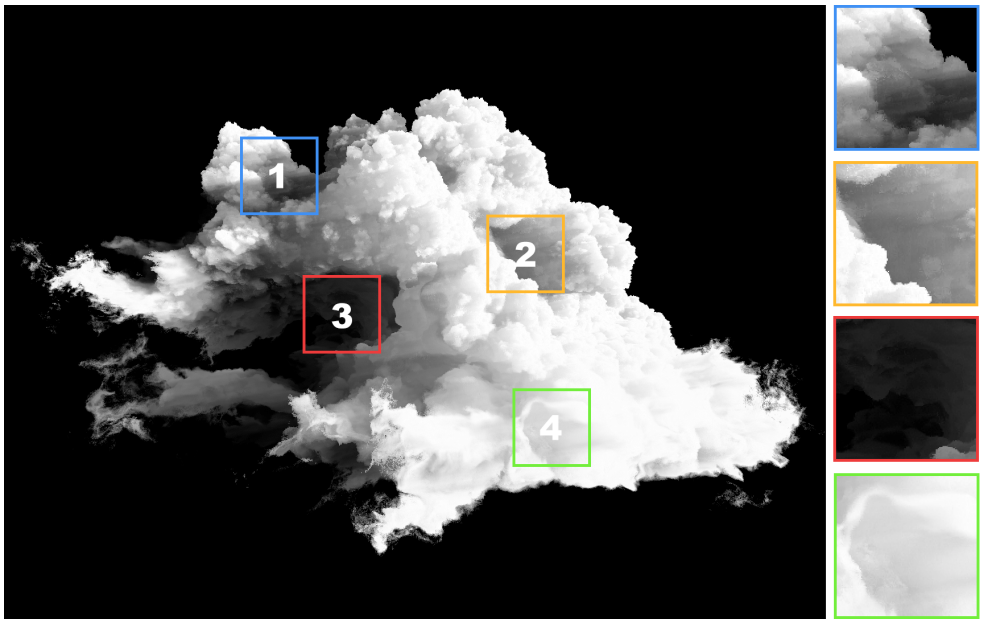

Figure 4.2: Local regions of interest 1-4 for *Scene 1: Cloud* containing various lighting conditions used for evaluation. The rendered image is the ground truth reference.

4.2.2 Scene 2: Cube

In *Scene 2: Cube*, the sun angle is the same as in *Scene 1*, except the camera is now facing towards the light source. Figure 4.3 showcases the rendered result of *Scene 2* with its single region of interest.

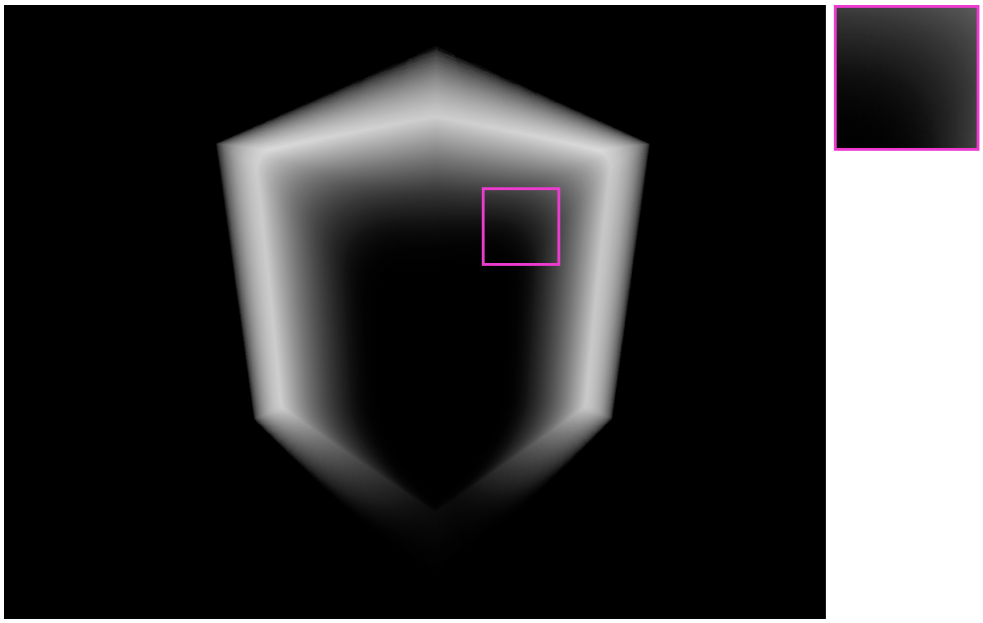

Figure 4.3: Single region of interest for *Scene 2: Cube*. The rendered image is the ground truth reference.

4.3 Hardware Specifications

All evaluation was done on a single machine with the following specifications, where clock speeds were taken from MSI Afterburner.*:

- NVIDIA GeForce RTX 3080 (10 GB GDDR6X VRAM)
  - 1950 MHz clock speed
  - 9501 MHz memory speed
- 12th Gen Intel(R) Core(TM) i5-12600K
- 32 GB of 3200 MHz RAM

4.4 Noise and Filters Evaluated

The following noise and spatial filters were considered for answering RQ1 and RQ2 (see Section 1.3.1, 1.3.2). Note that no spatial filters are used for RQ1. Blue noise, STBN, and FAST noise use precomputed textures – their footnote links to the download.

*https://www.msi.com/Landing/afterburner/graphics-cards

4.4.1 Noise Types

- White noise
- Independent blue noise*.
- Scalar STBN† [6].
- Scalar FAST‡ noise [7] optimized for EMA and each spatial filter.
- IGN [5]

The *Scalar* prefix on STBN and FAST noise indicates they are suited for random floats (scalar values). STBN and FAST have variations supporting other data types that fit the sampling space.

4.4.2 Spatial Filters

- Gaussian blur of $\sigma = 1$
- Box 3x3, and 5x5 blur
- Binomial 3x3, and 5x5 blur

4.5 Results

All images were rendered at 1920x1080 resolution and stored as grayscale R8 PNG format. The images are normalized from $[0-255]$ to $[0-1]$ before the RMSE value is calculated.

Performance was captured individually in a standalone release build, averaged over 600 frames, by reading Unity's `Time.deltaTime` variable.

Finally, the image with the lowest error is compared with NVIDIA ꟻLIP [23] to highlight the visual difference between the ground truth.

The raw data can be found inside the RQ1 and RQ2 folders at the Github repository§.

*https://momentsingraphics.de/BlueNoise.html
†https://github.com/NVIDIA-RTX/STBN/tree/main
‡https://github.com/electronicarts/fastnoise
§https://github.com/andersblomqvist/master-thesis-renderer/tree/main/RMSECalculator

4.5.1 RQ1: Temporally Filtered

To answer RQ1 (see Section 1.3.1), each scene was rendered using different sources of noise and temporally filtered over 32 frames ($F_{max} = 32$ from Equation 3.1) using an EMA with a fixed $\alpha = 0.1$. There was no temporal filtering for frame 0 as the history is undefined. In practice, the history texture was assigned to be the same as the newly rendered frame in that case.

Figure 4.4 shows the full screen RMSE on a logarithmic plot for both scenes using different noise sources. Notice that *Scene 2* has a lower range of values but a greater span. In both scenes, the scalar FAST noise results in the lowest error but performs noticeably better in *Scene 2*, where samples are less distorted by the volume. The FAST noise was optimized towards an EMA temporal filter and a Gauss 1.0 spatial filter, although no spatial filter was used here. Each type of noise shows similar convergence, but IGN is more erratic than the others. There is also no difference between white and blue noise, which is expected as they only differ perceptually, not in terms of the numerical error. Additional results comparing the Unity TAA with our EMA temporal filter can be found in the Appendix A.2. It provides a real use-case filtering scenario with the different noise sources.

The anomaly at frame 16 (2^4) in *Scene 2*, a downward spike, is reproduced each time an experiment is done. The image for frame 16 looks no different compared to frames 15 and 17. It is unclear what the cause is.

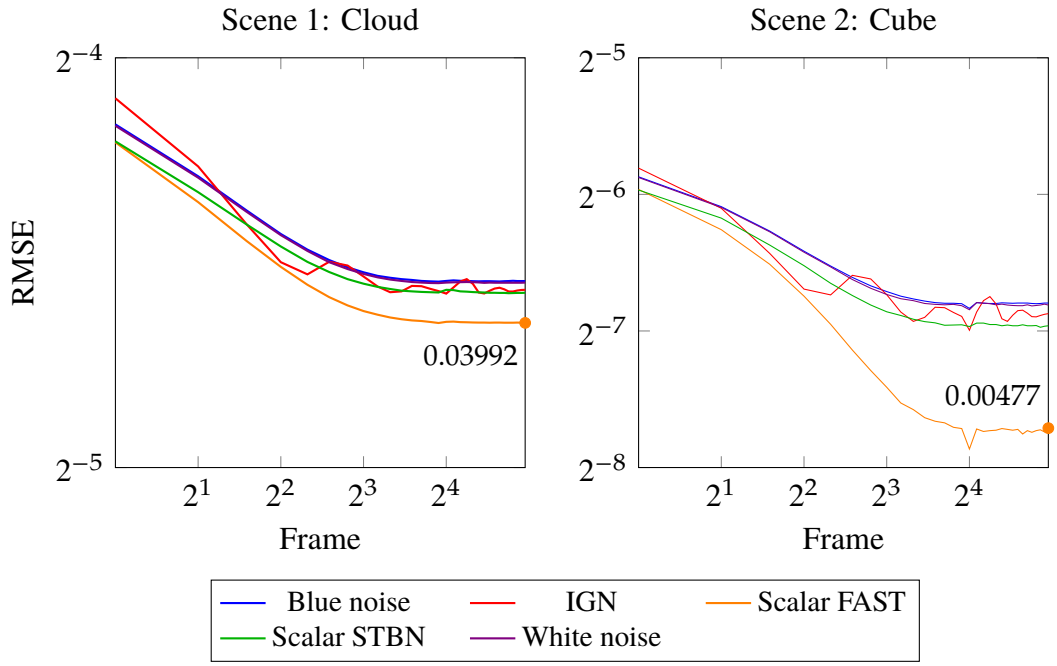

Figure 4.4: Full screen RMSE of *Scene 1* (left) and *Scene 2* (right), where FAST noise (orange) has the lowest error when temporally filtered through an EMA ($\alpha = 0.1$) across 32 frames. Notice that the left (*Scene 1*) diagram has a higher range of values than right (*Scene 2*). The FAST noise was optimized towards an EMA temporal filter and a Gauss 1.0 spatial filter, although no spatial filter was used here.

Figure 4.5 displays each image region, labeled 1-4, for *Scene 1* and their local RMSE values. Region 1 showcases the highest error where the cloud casts a shadow on itself. The ground truth reference image is darker and has more distinct shadows. And in region 3, the darkest one, all noise types appear slightly brighter than the reference image. In regions 2 and 4, where the light has traveled shorter distances, the error is smaller.

For *Scene 2*, the local region for each noise is presented in Figure 4.6. Here, the noise patterns are more visible compared to Figure 4.5. For example, in Figure 4.6, there is a visible difference between white and blue noise even though their errors are almost identical, while in *Scene 1* (Figure 4.5) white and blue noise appear visually very similar.

However, if one takes a very close look at the reference image in Figure 4.6 (recommended to view this document digitally in this case), there is noise visible. This noise is not added deliberately, as the ground truth reference uses no jitter. As such, the RMSE values are calculated against an already noisy

reference image, although this noise is very subtle.

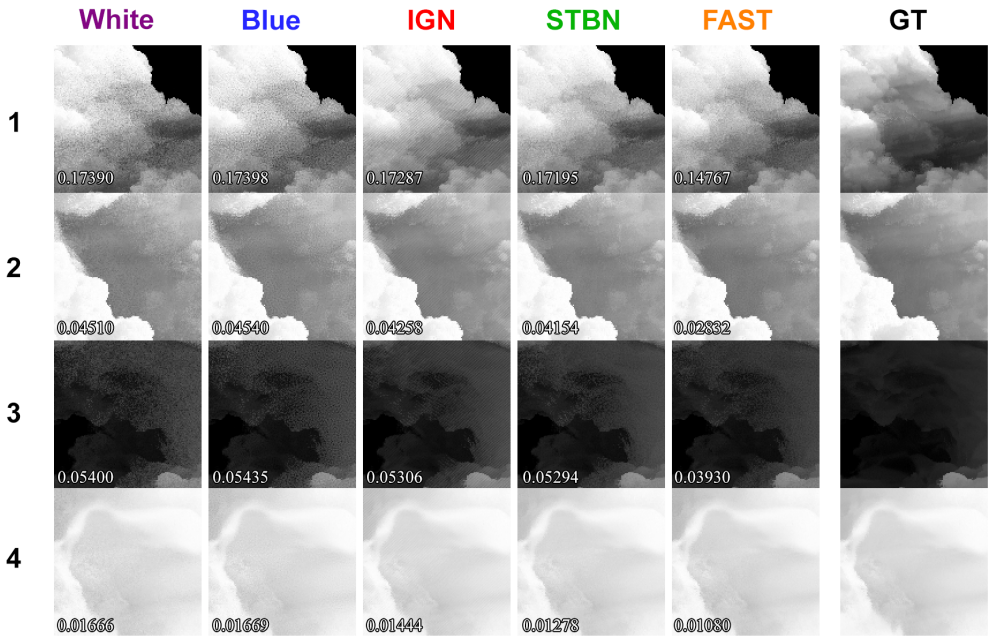

Figure 4.5: RMSE for local image regions 1-4 from Figure 4.2 for *Scene 1*. The rightmost column is ground truth. All images are from frame 31, where FAST has the lowest error for all regions.

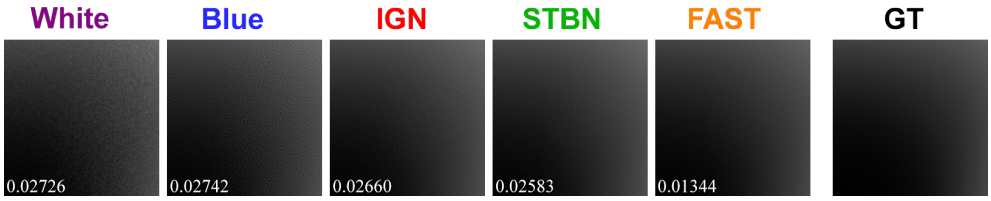

Figure 4.6: RMSE for local image region from Figure 4.3 for *Scene 2*. The rightmost column is ground truth. All images are from frame 31, where FAST has the lowest error.

And lastly, Figure 4.7 shows the performance measured in FPS for each noise – and when no jitter is applied. The performance drops by about 8 % when a random jitter is applied. There are no changes in performance otherwise.

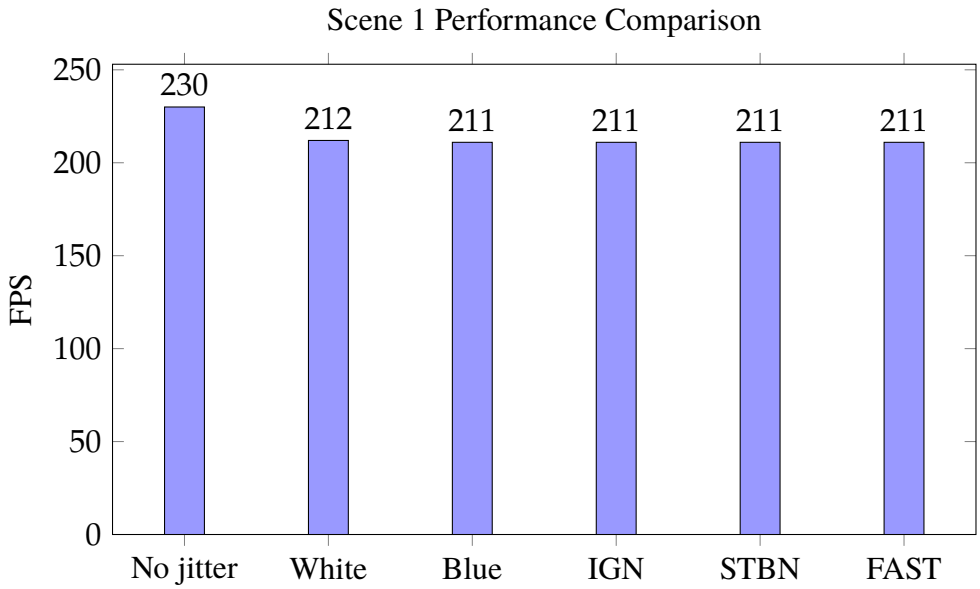

Figure 4.7: FPS for *Scene 1* using different sources of noise used for the jitter. Each noise decreases performance by 19 FPS compared to when no jitter is applied. For reference, 211 FPS is 4.7 ms.

4.5.2 RQ2: Temporally and Spatially Filtered

For RQ2 (see Section 1.3.2), the evaluation is identical to RQ1 (see Section 4.5.1), apart from the final image undergoing a spatial filter at the end, and the temporal filter is done over the last 12 frames, instead of 32. The spatial filter has equal strength across all frames.

The full screen RMSE for each spatial filter, in *Scene 1*, is shown in Figure 4.8. The scalar FAST noise is optimized for each spatial filter, meaning that there are a total of five different types of FAST noise, but they are all labeled as *Scalar FAST*. The general trend for all filters is that the error difference between the first and last frame is smaller compared to when no spatial filtering was done (Figure 4.4), as the curve is almost horizontal. The increase in error is discussed in Section 5.2.

The dataset for *Scene 2* contains less sharp edges, meaning the spatial filter does not remove as much information as in *Scene 1*. This is shown in Figure 4.9 as the final error values match the ones in Figure 4.4. The numerical difference between each spatial filter is very low, although Box 5x5 is slightly higher than the others. Once again, the spatial filter reduces the initial error values when the temporal filter has not yet converged, which can be seen by comparing the dotted gray line that represents FAST noise when no spatial filtering was done.

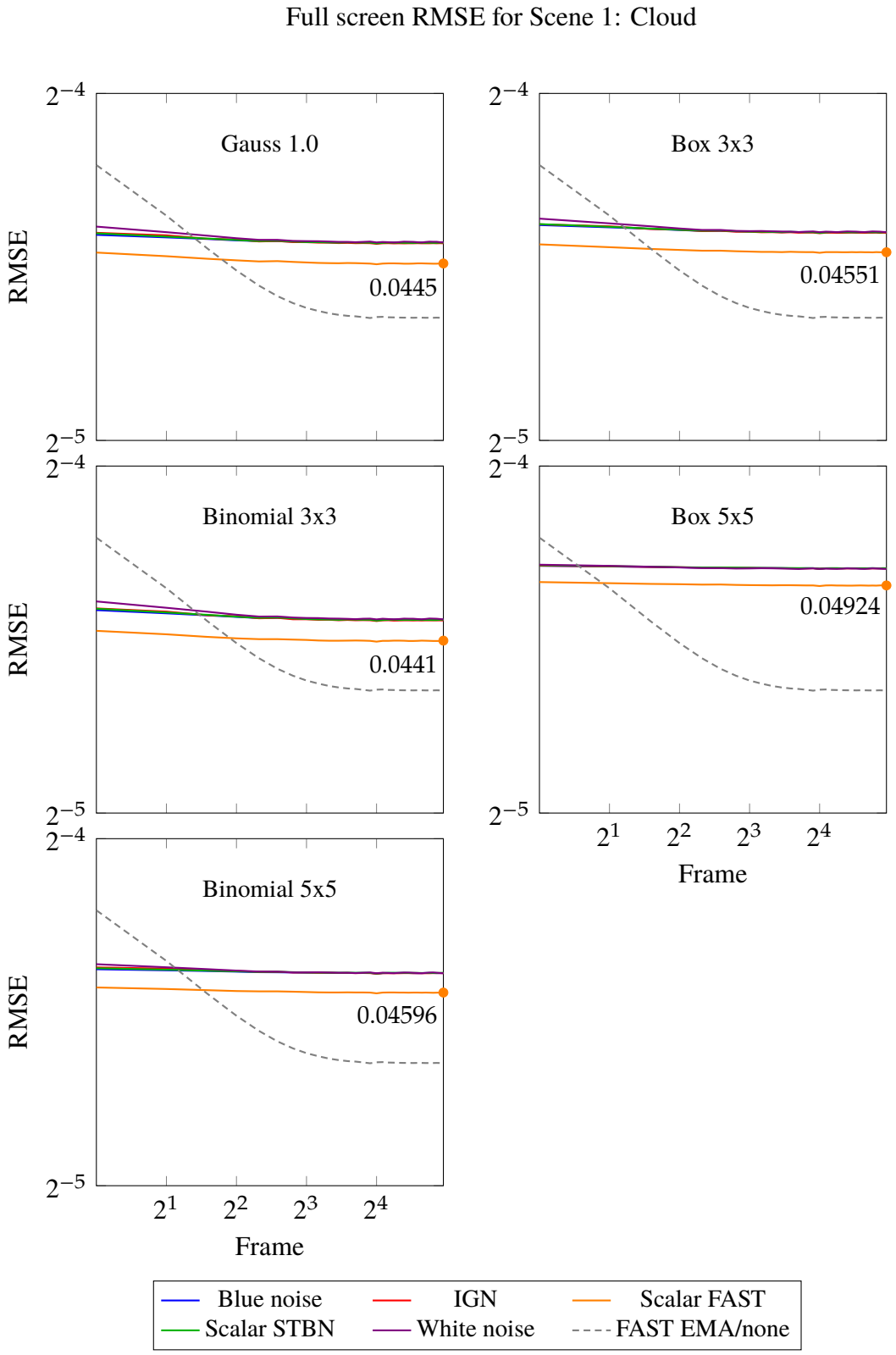

Figure 4.8: Full screen RMSE of *Scene 1* for all spatial filters applied on respective noise types. FAST noise (orange) has the lowest error for all spatial filters, where the Binomial 3x3 is lowest. All other noise types are bundled together. The dotted gray line is FAST noise using no spatial filtering (from Figure 4.4).

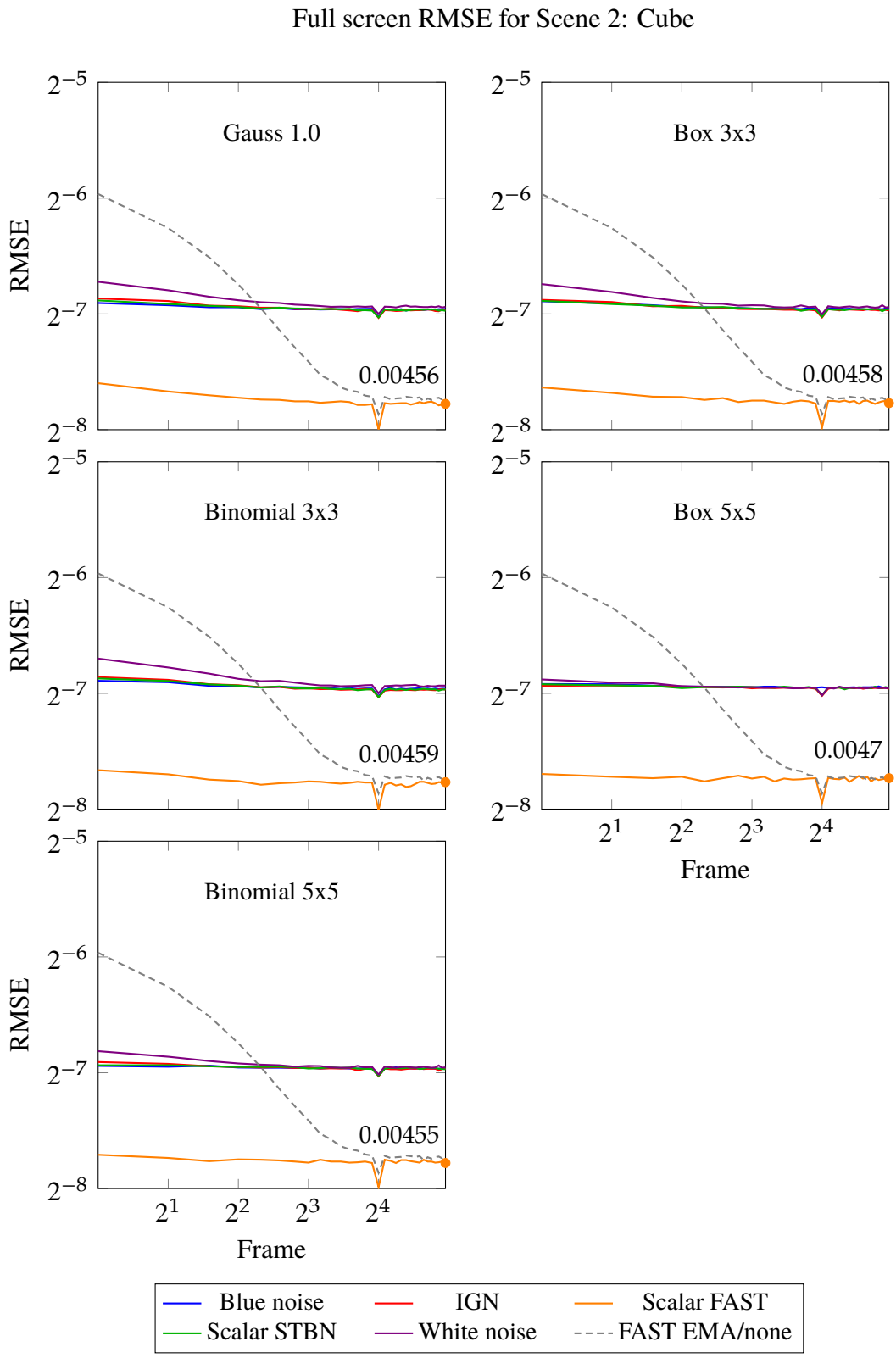

Figure 4.9: Full screen RMSE of *Scene 2* for all spatial filters applied on respective noise types. FAST noise (orange) has the lowest error for all spatial filters. The dotted gray line is FAST noise using no spatial filtering (from Figure 4.4).

Finally, Figure 4.10 shows an image diff generated using NVIDIA ꟻLIP while comparing the image rendered with FAST EMA/Binomial3x3 against the ground truth reference. It shows a high difference in the far shadowy areas and slightly lower in the medium range, which is expected from the results above. One can also see an outline of the cloud caused by the spatial filter that blurs sharp edges. This outline is not present in Figure 4.11, which has no spatial filtering. The top-right images highlight region 1 for all three images: ꟻLIP, FAST EMA/Binomial3x3, and ground truth.

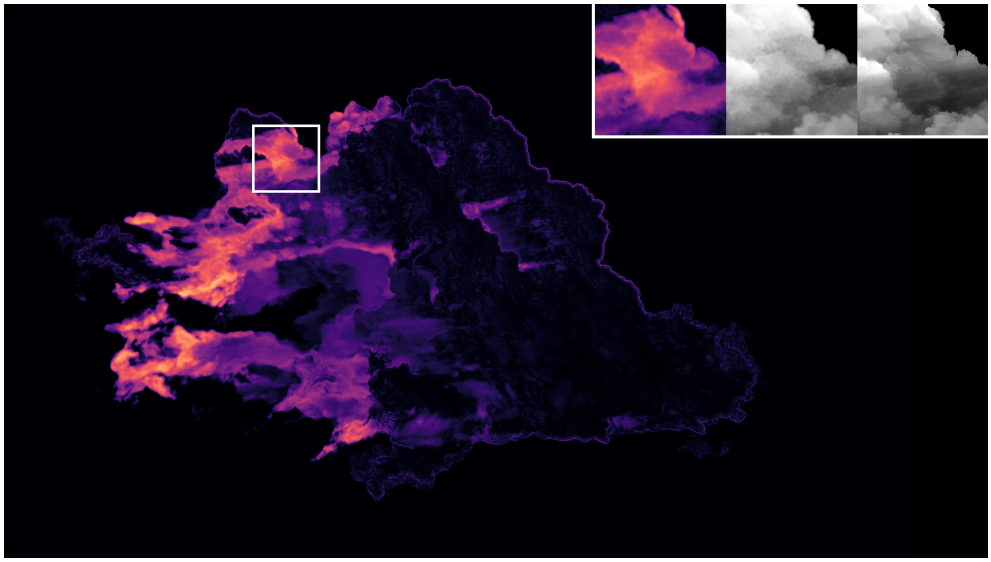

Figure 4.10: Image generated using NVIDIA ꟻLIP comparing the final frame from *Scene 1* rendered with FAST EMA/Binomial3x3 filters against ground truth. Bright areas are high differences, black is low. The top-right images highlight region 1 for all three images: ꟻLIP, FAST EMA/Binomial3x3, and ground truth. The highest difference is seen in the long-distance shadows, where samples are sparse.

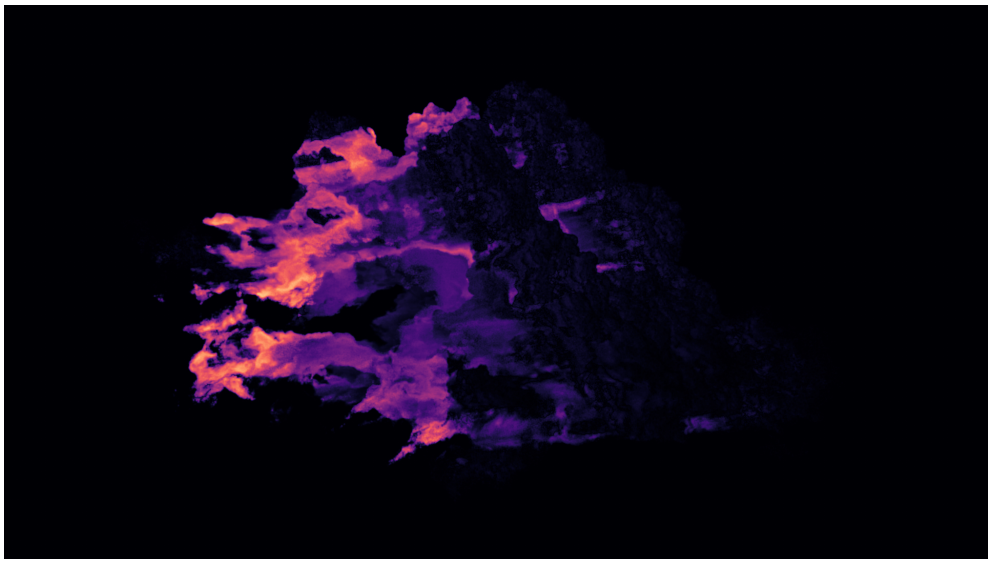

Figure 4.11: Image generated using NVIDIA ꟻLIP comparing the final frame from *Scene 1* rendered with FAST and EMA, no spatial filter. Notice there is no outline around the cloud, as shown in Figure 4.10.

Chapter 5

Discussion

This chapter will answer the research questions and discuss the results.

5.1 RQ1: Temporally Filtered

Recall from Section 1.3.1 that the first research question investigates which noise type, when combined with an EMA temporal filter, yields the lowest numerical error. As shown in Section 4.5.1 and Figures 4.4, 4.5, and 4.6, FAST consistently achieves the lowest error among the evaluated noise types. Notably, FAST is the only method specifically optimized for the EMA filter, inherently favoring its performance. In contrast, STBN is tailored for temporal Gaussian filtering, while the remaining noise lacks temporal properties. A different temporal filter may reduce STBN's error to levels comparable with FAST, as FAST extends STBN with adaptability to specific filters.

The visual comparison of image regions in Figure 4.5 revealed that the primary difference between low-sample images and the ground truth lies in the reduced shadow definition and darkness. With fewer samples, each one carries more weight, increasing the error when samples miss the volume by sampling empty space. Since the implementation uses single scattering without ambient light, some regions will appear black where light does not reach. In contrast, state-of-the-art volume renderers with ambient light and multi-scattering tend to produce brighter scenes with softer, less defined shadows, reducing the visual impact of such sampling artifacts, given that you are rendering a cloud in an Earth-like setting and not in space. For example, the path-traced image of the Disney Moana cloud shown in Figure 1.1 has soft shadows under direct lighting conditions.

Figure 4.4 showed that the error gap between the best- and worst-

performing noise types is smaller in *Scene 1* and larger in *Scene 2*. The dataset itself introduces variability, effectively distorting the intended noise patterns. This is a known limitation of low-discrepancy sampling, where sample quality decreases as scene complexity increases. Such distortion can reduce the effectiveness of filtering and obscure the advantages of specific noise types. Figure 4.6, which presented the local image regions for all noise types in *Scene 2*, showed that samples are less distorted than in *Scene 1*, resulting in a visible difference between white and blue noise despite their similar numerical errors.

5.1.1 Performance

Applying random jitter to sample positions introduces a noticeable performance cost, which was also observed by Toft et al.[2], who believe the drop in performance is due to increased cache misses. Their results (Figure 3 from [2]) showed performance went from 2.3 ms without jitter to 7.5 ms with jitter. In contrast, our data show only a minor change, from 4.3 ms (230 FPS) to 4.7 ms (211 FPS). Their larger change in performance may arise from differences in hardware, but perhaps more importantly, from the underlying memory structure: this thesis used OpenVDB, while they had dense 3D textures.

5.2 RQ2: Temporally and Spatially Filtered

The second research question from Section 1.3.2 extends RQ1 by evaluating which spatial filter, in combination with the EMA temporal filter, yields the lowest numerical error. Our results showed a clear reduction in error before temporal convergence, while after convergence, the error remains largely unchanged, except in Figure 4.8 from *Scene 1* by comparing the dotted gray line. The spatial filter acts as a temporary improvement until the temporal filter stabilizes. In real-time applications, however, convergence is rarely achieved unless the user remains completely stationary, and even then, scene elements, such as volumes, are likely to move. To minimize additional blurriness introduced by our spatial filter, a solution would be to (1) use an edge-aware filter that preserves high-frequency details, and (2) reduce filter strength when low variance is detected across multiple frames. Detecting low variance would require another algorithm that analyzes each frame with history, or another creative solution.

The numerical differences between spatial filters were minimal, making it difficult to identify an optimal filter based on our results. The increase in error observed in Figure 4.8 is likely due to an overly aggressive blur that

does not account for edges. This can be seen when comparing Figures 4.10 and 4.11, where the spatially filtered image has an outline around the cloud. Furthermore, sharp edges are common because the ground truth lacks voxel interpolation. As such, the exact error values in Figure 4.8 should be taken with a grain of salt. However, in Figure 4.9 for *Scene 2*, which contains fewer high-frequency details, the error differences between spatial filters are smaller than in *Scene 1*. This dataset is less affected by overly aggressive blurring and has smaller variations in error, suggesting that the choice of spatial filter is less important than the choice of noise type.

5.2.1 Visual Comparison

The image difference generated by ꟻLIP in Figures 4.10 and 4.11 showed a high difference in the long-distance shadows. As the light step size gets exponentially larger, it increases the likelihood of samples missing the volume, causing an inappropriate approximation for the density between points. It is believed this is the cause of the high variation between our low sample image and the ground truth. This is quite logical, as the assumption of constant density between two points is used. To get more accurate approximations, samples have to be placed more efficiently in relation to the volumetric model, but the naive solution is, of course, to increase samples. In our case, a maximum of 10 light samples were used to get a good mix between performance and visuals. The focus of this thesis was not to render a visually pleasing result, however, it is still relevant that the final render is as close to a real-use case scenario as possible. Otherwise, our findings would not apply to the general research area.

A rendered image of *Scene 1* using FAST noise optimized for EMA and binomial 3x3 filters can be found in the Appendix A.1. The ground truth is also presented there.

5.3 Overview

Thus far, our results have shown that color banding from low sample counts over large distances can be mitigated by introducing noise. This noise can be optimized in terms of numerical error by selecting noise types suited to the temporal and spatial filters. However, this thesis focuses solely on quantitative metrics and does not address qualitative aspects, which would require a user study. Numerical errors alone do not capture how users perceive the imagery presented to them, and various numerical metrics capture

different aspects of an image. For example, metrics such as ꟻLIP [23] attempt to provide a difference map (Figures 4.10 and 4.11) showcasing the difference humans perceive when flipping between a test image and ground truth. While RMSE, the metric used in this thesis, quantifies differences without accounting for human visual perception. Putting numerical metrics aside, differences between noise types, except for white noise, are difficult to identify at typical viewing distances based on our observations. This is when viewed at approximately an arm's length away on a 24" monitor at 1920x1080 resolution. Differences become noticeable when examining the image closely from about 20-30 cm away. However, images with spatial filtering are noticeably blurry at typical viewing distances, likely due to the non-edge-aware blur. As such, qualitative metrics could provide new insights into which combination of noise and filters yields the most *perceived* noise-free image.

Chapter 6

Conclusions and Future work

6.1 Conclusions

This thesis investigated how stochastic rendering with various sources of noise and filters can reduce color banding and noise artifacts in real-time volumetric cloud rendering. Two research questions were explored: identifying which noise types yield the lowest numerical error when temporally filtered (RQ1), and which spatial filters further reduce error in combination with temporal filtering (RQ2).

The results showed that FAST [7] noise, specifically optimized for the EMA temporal filter, consistently outperformed other types in both test scenes, producing the lowest RMSE. The effectiveness of noise types and filtering was scene-dependent, where most benefit was shown in scenes with low variation in the volumetric dataset (scene 2). While in a real use-case scenario, such as the Disney Moana cloud (scene 1), the effectiveness was lower as the dataset introduced variations.

For RQ2, spatial filters reduced error before temporal convergence, but had minimal impact after. The choice of spatial filter had less influence than the noise type, and over-blurring introduced new artifacts, particularly around sharp edges. Thus, spatial filters should be applied with care and preferably adapted based on scene content and motion. It is impossible to claim an optimal filter among the spatial filters evaluated, as the error difference was so small.

Overall, the thesis demonstrates that choosing the right noise type and filter design is key to minimizing visual artifacts in real-time volumetric rendering.

6.2 Limitations

Several limitations to this study affect the generalizability and interpretability of the results:

Simplified Rendering Scenario: The implementation uses a single-scattering model without ambient light or multiple scattering effects. While this simplification helps isolate and study the effects of noise and filtering on visual artifacts, it does not fully represent modern, state-of-the-art volumetric rendering pipelines. In real-world applications, more complex lighting scenarios would likely mask some of the artifacts examined here, potentially resulting in a better image. However, a more complex rendering scenario might also reduce the effect of noise and filter properties.

Ground Truth Representation: The ground truth images used for error calculation were generated with no voxel interpolation, leading to unnaturally sharp edges and subtle noise being present. This penalizes our spatial filtering unfairly, as the ground truth does not represent a realistic rendering scenario, and the filters were not edge-aware.

No Qualitative Evaluation: Only quantitative metrics (RMSE and performance) were considered. Visual perception varies between individuals, and numerical differences may not align with perceived quality. For example, differences between certain noise types were hard to distinguish at typical viewing distances. A user study would be necessary to validate the perceptual impact of different configurations.

6.3 Future Work

Improvements can be made to both the renderer and the evaluation. This section highlights suggestions to further improve the work.

Qualitative Measurement: As noted in the discussion (Section 5.3), a user study would be a logical next step for evaluating noise and filtering techniques. Such a study could involve an interactive rendering scenario, allowing users to move around while observing the volume. This setup may reveal limitations of the temporal filtering approach, which often produces noticeable blurring during motion and sharp results when stationary, an effect similar to putting your glasses on and off.

Advanced Rendering: Discussed in Section 5.1, a state-of-the-art renderer tends to produce softer, less defined shadows, thus reducing the visual impact of our sampling artifacts. This is interesting to investigate as it could

allow for fewer samples, increasing performance, although more complex lighting has to be calculated, which will hurt performance instead. Moreover, a more complex rendering scenario may degrade sampling quality, diminishing the effectiveness of noise and filtering techniques.

6.4 Sustainability

Stochastic rendering amortizes computational cost across multiple frames, enabling higher frame rates. This might suggest lower energy consumption compared to our ground truth rendering, which performs more light steps without amortization. However, uncapped rendering utilizes all available resources, leading to high energy usage. During evaluation, without a frame rate cap, the GPU consumed 300 W. Introducing a 60 FPS cap reduced consumption to 185 W. However, reducing the number of ray marching steps further lowers energy consumption when a frame rate cap is applied. Therefore, minimizing the number of samples remains beneficial, even without additional gains in frame rate.

6.5 Ethics and Societal Aspects

The ethical impact of this thesis is minimal, as it focuses on noise reduction in images. However, it contributes to the advancement of photorealistic rendering, which carries the potential for misuse in deceptive applications.

From a societal perspective, the algorithm enhances accessibility by enabling higher frame rates with comparable image quality, allowing users with less powerful hardware to experience the technology.

References

[1] R. Högfeldt, "Convincing Cloud Rendering - An Implementation of Real-Time Dynamic Volumetric Clouds in Frostbite," Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, Jun. 2016. [Pages 2, 4, 16, 22, and 26.]

[2] A. Toft, H. Bowles, and D. Zimmermann, "Optimisations for Real-Time Volumetric Cloudscapes," Sep. 2016, arXiv:1609.05344 [cs]. [Online]. Available: http://arxiv.org/abs/1609.05344 [Pages 2, 4, 16, 22, and 44.]

[3] S. Hillaire, "Physically-based & Unified Volumetric Rendering in Frostbite - Frostbite," Aug. 2015. [Online]. Available: https://www.ea.com/frostbite/news/physically-based-unified-volumetric-rendering-in-frostbite [Pages xi, 2, 4, 8, 9, 16, and 24.]

[4] A. Schneider, *GPU Pro 7: Advanced Rendering Techniques*, 1st ed. A K Peters/CRC Press, 2016. ISBN 978-0-429-16003-5 Pp. 97-127. [Pages 2 and 15.]

[5] J. Jimenez, "Next Generation Post Processing in Call of Duty: Advanced Warfare," in *ACM SIGGRAPH 2014 Courses*, ser. SIGGRAPH '14. Vancouver, Canada: Association for Computing Machinery, Aug. 2014. doi: 10.1145/2614028.2615455. ISBN 978-1-4503-2962-0 pp. 1–171. [Online]. Available: https://doi.org/10.1145/2614028.2615455 [Pages 3 and 33.]

[6] A. Wolfe, N. Morrical, T. Akenine-Möller, and R. Ramamoorthi, *Spatiotemporal Blue Noise Masks*. The Eurographics Association, 2022. ISBN 978-3-03868-187-8 ISSN: 1727-3463. [Online]. Available: https://doi.org/10.2312/sr.20221161 [Pages 3, 16, 17, 29, 33, and 56.]

[7] W. Donnelly, A. Wolfe, J. Bütepage, and J. Valdés, "FAST: Filter-Adapted Spatio-Temporal Sampling for Real-Time Rendering,"

Proceedings of the ACM on Computer Graphics and Interactive Techniques, vol. 7, no. 1, pp. 1–16, May 2024. doi: 10.1145/3651283. [Online]. Available: https://dl.acm.org/doi/10.1145/3651283 [Pages 3, 16, 17, 26, 33, 47, and 56.]

[8] L. Yang, D. Nehab, P. V. Sander, P. Sitthi-amorn, J. Lawrence, and H. Hoppe, "Amortized supersampling," *ACM Trans. Graph.*, vol. 28, no. 5, pp. 1–12, Dec. 2009. doi: 10.1145/1618452.1618481. [Online]. Available: https://doi.org/10.1145/1618452.1618481 [Pages 4 and 10.]

[9] L. Yang, S. Liu, and M. Salvi, "A Survey of Temporal Antialiasing Techniques," *Computer Graphics Forum*, vol. 39, no. 2, pp. 607–621, May 2020. doi: 10.1111/cgf.14018. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1111/cgf.14018 [Pages 5, 14, and 15.]

[10] J. T. Kajiya and B. P. Von Herzen, "Ray tracing volume densities," in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '84. New York, NY, USA: Association for Computing Machinery, Jan. 1984. doi: 10.1145/800031.808594. ISBN 978-0-89791-138-2 pp. 165–174. [Online]. Available: https://dl.acm.org/doi/10.1145/800031.808594 [Page 7.]

[11] J. Fong, M. Wrenninge, C. Kulla, and R. Habel, "Production volume rendering: SIGGRAPH 2017 course," in *ACM SIGGRAPH 2017 Courses*, ser. SIGGRAPH '17. New York, NY, USA: Association for Computing Machinery, Jul. 2017. doi: 10.1145/3084873.3084907. ISBN 978-1-4503-5014-3 pp. 1–79. [Online]. Available: https://doi.org/10.1145/3084873.3084907 [Page 8.]

[12] S. Chandrasekhar, "Radiative transfer," *Quarterly Journal of the Royal Meteorological Society*, vol. 76, no. 330, 1950. doi: 10.1002/qj.49707633016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/qj.49707633016 [Page 10.]

[13] J. Stam, "Stochastic Rendering of Density Fields," in *Graphics Interface*. CANADIAN INFORMATION PROCESSING SOCIETY, Jun. 1994, pp. 51–51. [Page 10.]

[14] R. L. Cook, "Stochastic sampling in computer graphics," *ACM Trans. Graph.*, vol. 5, no. 1, pp. 51–72, Jan. 1986. doi: 10.1145/7529.8927.

[Online]. Available: https://dl.acm.org/doi/10.1145/7529.8927 [Page 10.]

[15] G. Marsaglia, "Xorshift RNGs," *Journal of Statistical Software*, vol. 08, Jan. 2003. doi: 10.18637/jss.v008.i14 [Page 11.]

[16] D. P. Mitchell, "Generating antialiased images at low sampling densities," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, Aug. 1987. doi: 10.1145/37401.37410. ISBN 978-0-89791-227-3 pp. 65–72. [Online]. Available: https://dl.acm.org/doi/10.1145/37401.37410 [Pages 11 and 16.]

[17] I. Georgiev and M. Fajardo, "Blue-noise dithered sampling," *ACM SIGGRAPH 2016 Talks*, 2016. doi: 10.1145/2897839.2927430 [Page 11.]

[18] A. Torralba, P. Isola, and W. Freeman, *Foundations of Computer Vision*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2024. ISBN 9780262378666. [Online]. Available: https://mitpress.mit.edu/9780262048972/foundations-of-computer-vision/ [Pages 12, 13, and 14.]

[19] K. Museth, "VDB: High-resolution sparse volumes with dynamic topology," *ACM Transactions on Graphics*, vol. 32, no. 3, pp. 1–22, Jun. 2013. doi: 10.1145/2487228.2487235. [Online]. Available: https://dl.acm.org/doi/10.1145/2487228.2487235 [Page 15.]

[20] ——, "NanoVDB: A GPU-Friendly and Portable VDB Data Structure For Real-Time Rendering And Simulation," in *ACM SIGGRAPH 2021 Talks*. Virtual Event USA: ACM, Jul. 2021. doi: 10.1145/3450623.3464653. ISBN 978-1-4503-8373-8 pp. 1–2. [Online]. Available: https://dl.acm.org/doi/10.1145/3450623.3464653 [Page 15.]

[21] S. Gaida, "Real-Time Implementation of OpenVDB Rendering," Master's thesis, Universität Koblenz - Landau, Germany, Jul. 2022. [Pages 15, 19, 22, 23, and 29.]

[22] K. Museth, "Hierarchical digital differential analyzer for efficient ray-marching in OpenVDB," in *ACM SIGGRAPH 2014 Talks*. Vancouver Canada: ACM, Jul. 2014. doi: 10.1145/2614106.2614136. ISBN

978-1-4503-2960-6 pp. 1–1. [Online]. Available: https://dl.acm.org/doi/10.1145/2614106.2614136 [Pages 15 and 22.]

[23] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, K. Åström, and M. D. Fairchild, "FLIP: A Difference Evaluator for Alternating Images," *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, vol. 3, no. 2, pp. 1–23, Aug. 2020. doi: 10.1145/3406183. [Online]. Available: https://dl.acm.org/doi/10.1145/3406183 [Pages 33 and 46.]

Appendix A

Additional Results

A.1 Ground Truth and Lowest Error Image

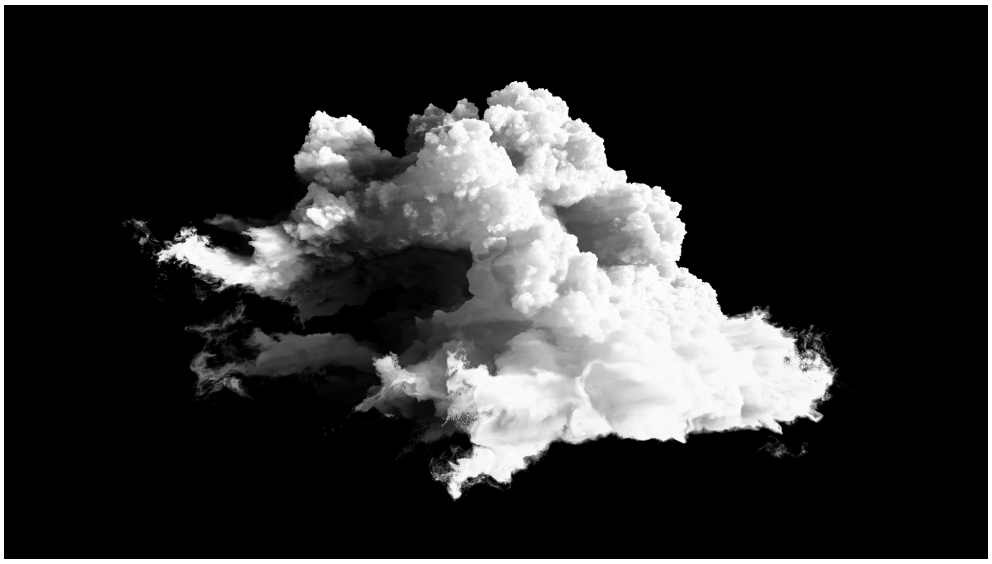

Figure A.1: Ground truth reference for *Scene 1* using uniform light stepping with 256 light samples.

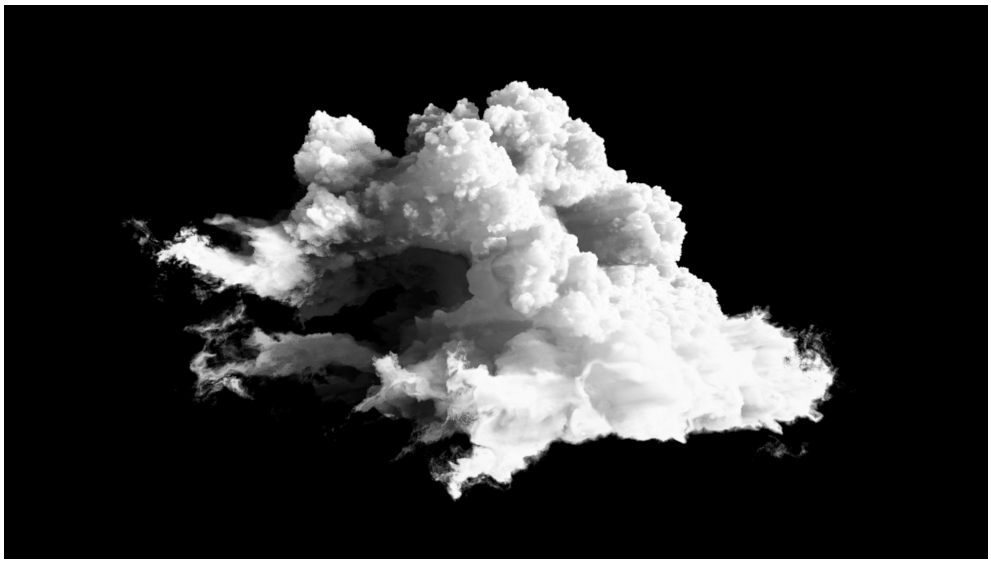

Figure A.2: Low sample render using FAST noise optimized for EMA/binomial 3x3 filters. Rendered at 209 FPS with 10 light samples per ray march step using an EMA temporal filter and a 3x3 binomial spatial filter.

A.2 Unity TAA Comparison

Recall that the implementation uses an EMA temporal filter as a simplified 'approximation' of a full TAA solution. Unity provides a complete TAA implementation typically found in industry-standard engines, enabling direct comparison of results. Figure A.3 presents the RMSE under the same conditions as the RQ1 evaluation (Section 4.5.1), where filtering was applied over 32 frames using Unity TAA instead of our EMA. The dotted lines are from Figure 4.4, using the same colors for comparison. Perhaps the most noticeable difference, apart from the lower error overall, is that IGN is less erratic and performs similarly to STBN . By examining the Unity TAA source code, it uses a 5-Tap history sample, meaning that it takes five samples from the history texture within a 4x4 pixel region. In comparison, our EMA takes one sample. The blog post *Interleaved Gradient Noise: A Different Kind of Low Discrepancy Sequence*[*] from Alan Wolfe, one of the authors of [6, 7], describes why IGN's low discrepancy properties are favorable for this kind of sampling. In essence, it has a good distribution of values within small image regions, such as 3x3 or 4x4, creating more accurate samples.

[*]Interleaved Gradient Noise: A Different Kind of Low Discrepancy Sequence

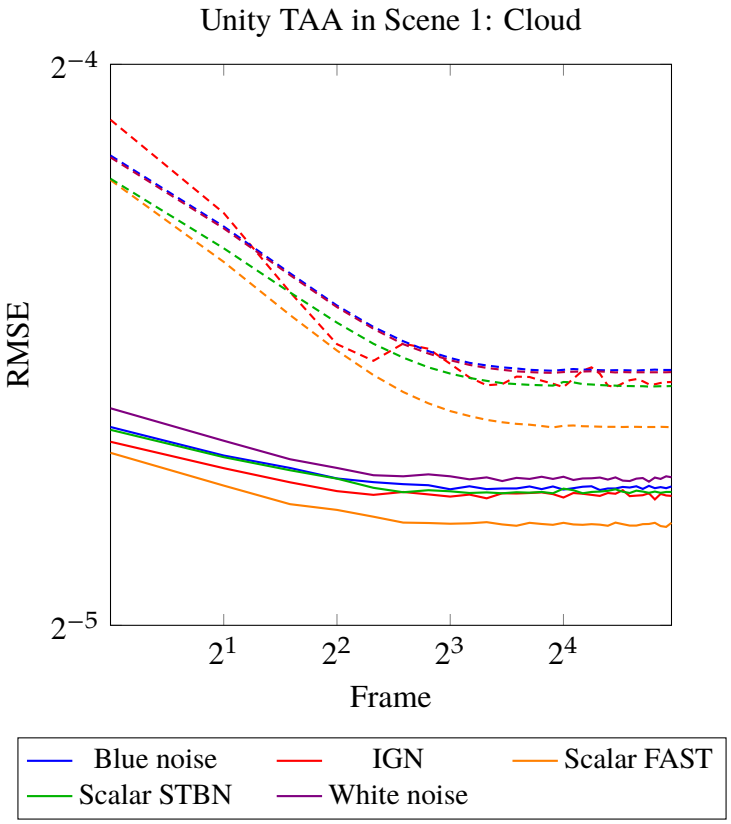

Figure A.3: Full screen RMSE of *Scene 1* using Unity TAA filtered across 32 frames. The dotted lines are from Figure 4.4 for comparison. FAST is optimized towards EMA and Gaussian 1.0 filters. Unity TAA set to the highest quality preset, resulting in five history samples.

TRITA – XXX-EX 2025:0000
Stockholm, Sweden 2025

www.kth.se

€€€€ For DIVA €€€€

{
”Author1”: { ”Last name”: ”Blomqvist”,
”First name”: ”Anders”,
”Local User Id”: ”u148wewn”,
”E-mail”: ”andblomq@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
}
},
”Cycle”: ”2”,
”Course code”: ”DA231X”,
”Credits”: ”30.0”,
”Degree1”: {”Educational program”: ”Degree Programme in Computer Science and Engineering”
,”programcode”: ”CDATE”
,”Degree”: ”Degree of Master of Science in Engineering”
,”subjectArea”: ”Computer Science and Engineering”
},
”Title”: {
”Main title”: ”Optimizing Light Samples for Real-Time Volumetric Clouds”,
”Subtitle”: ”Through Stochastic Rendering and Noise Reduction”,
”Language”: ”eng” },
”Alternative title”: {
”Main title”: ”Optimering av ljusprover för realtids volumetriska moln”,
”Subtitle”: ”Genom stokastisk rendering och brusreducering”,
”Language”: ”swe”
},
”Supervisor1”: { ”Last name”: ”Peters”,
”First name”: ”Christopher”,
”Local User Id”: ”u1qtfy8j”,
”E-mail”: ”chpeters@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
”L2”: ”Computer Science” }
},
”Examiner1”: { ”Last name”: ”Weinkauf”,
”First name”: ”Tino”,
”Local User Id”: ”u1dsgbcl”,
”E-mail”: ”weinkauf@kth.se”,
”organisation”: {”L1”: ”School of Electrical Engineering and Computer Science”,
”L2”: ”Computer Science” }
},
”National Subject Categories”: ”10201, 10206”,
”SDGs”: ”7, 12, 13”,
”Other information”: {”Year”: ”2025”, ”Number of pages”: ”xix,57”},
”Copyrightleft”: ”copyright”,
”Series”: { ”Title of series”: ”TRITA – XXX-EX” , ”No. in series”: ”2025:0000” },
”Opponents”: { ”Name”: ”A. B. Normal & A. X. E. Normalè”},
”Presentation”: { ”Date”: ”2025-06-09 13:00”
,”Language”:”eng”
,”Room”: ”via Zoom https://kth-se.zoom.us/j/ddddddddddd”
,”Address”: ”Isafjordsgatan 22 (Kistagången 16)”
,”City”: ”Stockholm” },
”Number of lang instances”: ”2”,
”Abstract[eng]”: €€€€

Volumetric clouds are popular within real-time rendering applications such as games, as they provide realism to outdoor scenes. Therefore, rendering volumetric clouds at high quality is of interest to enhance user immersion. However, rendering volumetric clouds is known to be computationally demanding due to the high number of samples required per pixel during ray marching. To address this, this thesis investigates how stochastic rendering can reduce the rendering cost while preserving visual fidelity. The thesis focuses on optimizing the positions of a few sets of light samples for large volumes, such as the Walt Disney Animation Studios' Moana cloud at half resolution, by randomly offsetting them and performing temporal and spatial filtering for noise reduction. The goal is to reduce the noise introduced by the stochastic rendering process. Various sources of noise, including White noise, Blue noise, Interleaved Gradient noise (IGN), scalar Spatiotemporal Blue noise (STBN), and scalar Filter-Adopted Spatio-Temporal noise (FAST), are evaluated along with denoising filters such as an Exponential Moving Average (EMA) temporal filter, and Gaussian sigma 1.0, Box 3x3/5x5, and Binomial 3x3/5x5 spatial filters. Performance is assessed through frames per second and image quality, measured using Root Mean Square Error (RMSE). Our results show that FAST noise, designed for specific temporal and spatial filters, consistently yields the lowest numerical error. While spatial filters offer only minor improvements post-convergence, they help reduce error in early frames. The findings highlight the importance of matching noise types to filters and suggest future user studies for perceptual validation in real-time scenarios.

€€€€,
”Keywords[eng]”: €€€€
Volumetric Clouds, Stochastic Rendering, Sampling, Noise €€€€,
”Abstract[swe]”: €€€€

Volymetriska moln är populära inom realtidsrendering, såsom i spel, eftersom de bidrar med realism till utomhusscener. Att rendera volymetriska moln med hög kvalitet är därför av intresse för att öka användarens upplevelse av inlevelse. Renderingen av volymetriska moln är dock känt för att vara beräkningsintensiva på grund av det stora antalet prov som krävs per pixel vid Ray Marching. För att hantera detta undersöker denna avhandling hur stokastisk rendering kan minska renderingskostnaden samtidigt som den visuella kvaliteten bibehålls. Avhandlingen fokuserar på att optimera positionerna för ett fåtal uppsättningar av ljusprover för stora volymer, såsom Walt Disney Animation 'Studios Moana-moln i halv upplösning, genom att slumpmässigt förskjuta dem och tillämpa temporal och spatial filtrering för att minska brus. Målet är att minska det brus som introduceras av den stokastiska renderingsprocessen. Olika brusmetoder utvärderas, däribland vitt brus, blått brus, växelvis gradientbrus (IGN), skalärt spatiotemporalt blått brus (STBN) och skalärt filteranpassat spatiotemporalt brus (FAST), tillsammans med brusreducerande filter såsom ett temporalt filter baserat på exponentiell utjämning (EMA) samt spatiala filter som Gaussiskt filter med sigma 1.0, Box 3x3/5x5 och Binomial 3x3/5x5. Prestanda bedöms genom bilder per sekund och bildkvalitet, mätt med Root Mean Square Error (RMSE). Våra resultat visar att FAST-bruset, som är utformat för specifika temporala och spatiala filter, konsekvent ger det lägsta numeriska felet. Även om spatiala filter endast ger små förbättringar efter konvergens, hjälper de till att minska felet i de tidiga bilderna. Resultaten belyser vikten av att matcha brusmetod med filter, och föreslår framtida användarstudier för perceptuell validering i realtidsscenarier.

€€€€,
"Keywords[swe]": €€€€
Volumetriska moln, Stokastisk Rendering, Provtagning, Brus €€€€,
}

acronyms.tex

```
%%% Local Variables:
%%% mode: latex
%%% TeX-master: t
%%% End:
% The following command is used with glossaries-extra
\setabbreviationstyle[acronym]{long-short}
% The form of the entries in this file is \newacronym{label}{acronym}{phrase}
%                                      or \newacronym[options]{label}{acronym}{phrase}
% see "User Manual for glossaries.sty" for the  details about the options, one example is shown below
% note the specification of the long form plural in the line below

\newacronym{EMA}{EMA}{Exponential Moving Average}
\newacronym{FAST}{FAST}{Filter-Adopted Spatio-Temporal}
\newacronym{FPS}{FPS}{Frames Per Second}
\newacronym{HDDA}{HDDA}{Hierarchical Digital Differential Analyzer}
\newacronym{HDRP}{HDRP}{High Definition Render Pipeline}
\newacronym{HLSL}{HLSL}{High-Level Shading Language}
\newacronym{GPU}{GPU}{Graphics Processing Unit}
\newacronym{IGN}{IGN}{Interleaved Gradient Noise}
\newacronym{RTE}{RTE}{Radiative Transfer Equation}
\newacronym{RMSE}{RMSE}{Root Mean Square Error}
\newacronym{STBN}{STBN}{Spatiotemporal Blue Noise}
\newacronym{TAA}{TAA}{Temporal Anti-Aliasing}
\newacronym{WDAS}{WDAS}{Walt Disney Animation Studios}
```