Volumetric Fog

Joshua Lanman

A report

submitted in partial fulfillment of the

requirements of the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2018

Project Committee:

Kelvin Sung, Chair
Jason Pace
Yusuf Pisan

Contents

Table of Figures

Abstract

Advances in computing hardware and the introduction of the Graphics Processing Unit (GPU) have empowered computer graphics in many fields, including the movie and video game industries. These advances have led to the widespread use of complex and realistic special effects such as explosions, fire, smoke, rain, fog, dust, shafts of light or shadow, and others. While many public displays of these effects in use exist, the details of their illumination modeling and implementations are often protected as trade secrets. This is especially true in the video game industry, where the difference between being a leader or just another competitor in the large sea of companies and independents can come down to the small quality achievements one company's product has over its competitors [1], [2].

The focus of this project is the understanding and implementation of one volumetric effect, fog. Our project has three goals. The first goal is to build an understanding of the illumination model of fog in general and how this effect can be approximated and created in modern imagery using a GPU. The second goal is to demonstrate our understanding by implementing an interactive system where these effects can be interacted with and examined. Our third goal is to provide a starting point for future researchers and enthusiasts by sharing both the lessons we have learned and the resources we have developed during this project.

Many challenges exist in creating realistic volumetric fog. In general, the volumetric illumination by itself is a complex process to model. The challenge becomes even greater when we consider the interaction between other objects in the scene and the fog particles. Add to this the fact that real fog varies in density as a function of distance, altitude, and time. Faced with an overwhelming number of calculations, modern techniques seek to strike a balance between hardcore mathematical modeling, advanced hardware techniques, and approximation to meet quality and performance goals.

In this project, we studied both the science behind light interactions with a participating medium and different techniques for modeling this behavior, culminating in the system we have implemented. To meet our goal of providing a starting point for future researchers and game enthusiasts, we have chosen to implement our project in the free, cross-platform game engine, Unity. Our system can generate fog either scene-wide or within finely controlled fog volumes at specified locations within the scene. Our solution can control the density of the fog as a function of both distance and altitude. The system can also introduce random, variable density into the fog along with velocity settings to simulate wind effects. This system also supports fog interaction with multiple lights within the game scene, as well as shafts of light and shadow within the fog volume. Each effect can be independently switched on and manipulated, giving the user precision control over the resulting fog. Our system, with its well-documented implementation, is an excellent learning and exploration tool for anyone interested in volumetric fog.

1 Introduction

Since the late 2000's and the rise of the Graphics Processing Unit (GPU), volumetric effects have been incorporated with increasing frequency in computer graphics for games and movies. When used effectively in a 3-dimensional scene, volumetric effects create a sense of both depth and realism [3]. They are also used to generate many complex effects that are difficult to produce using standard 3D-modeling techniques involving geometric primitives. These effects include explosions, fire, smoke, rain, fog, dust, shafts of light or shadow, and other ethereal effects [4].

1.1 Project Focus and Motivations

This project focuses on implementing volumetric fog. It attempts to simulate many of the different aspects of real-life fog, including how it interacts with different sources of light and how the density of fog varies over time and distance.

Several reasons exist for investigating and implementing volumetric fog. The first reason is that this technique has many applications, including special effects in movies and graphic artworks, as well as for implementing effects in video game development. The second reason for studying volumetric fog is that many other effects can be produced by utilizing similar techniques [4]. Effects such as sand storms, rain, murky underwater environments and many others can be approached similarly to volumetric fog, making this a gateway project.

1.2 Challenges and Solution Approach

In this section, we will discuss the challenges of understanding and implementing volumetric fog. We will discuss past approaches, more recent approaches, and finally our approach to understanding and solving these challenges.

1.2.1 Why Fog is Difficult

Realistic fog is a complex and challenging effect to model and implement. This is in large part due to the complex interactions between light and a participating volumetric medium. As a result, the illumination model can be challenging to calculate [5].

When we talk about the illumination model in a scene, we are often talking about both the local and global illumination models. Local illumination is the term used describe the modeling of light as it travels from a source and is reflected once off of a surface towards the eye of the viewer [6]. Global illumination expands on the local illumination model by attempting to measure the propagation of each light ray throughout the entire scene, adding in more layers to the model dealing with bounce lighting, refraction, and shadows [7].

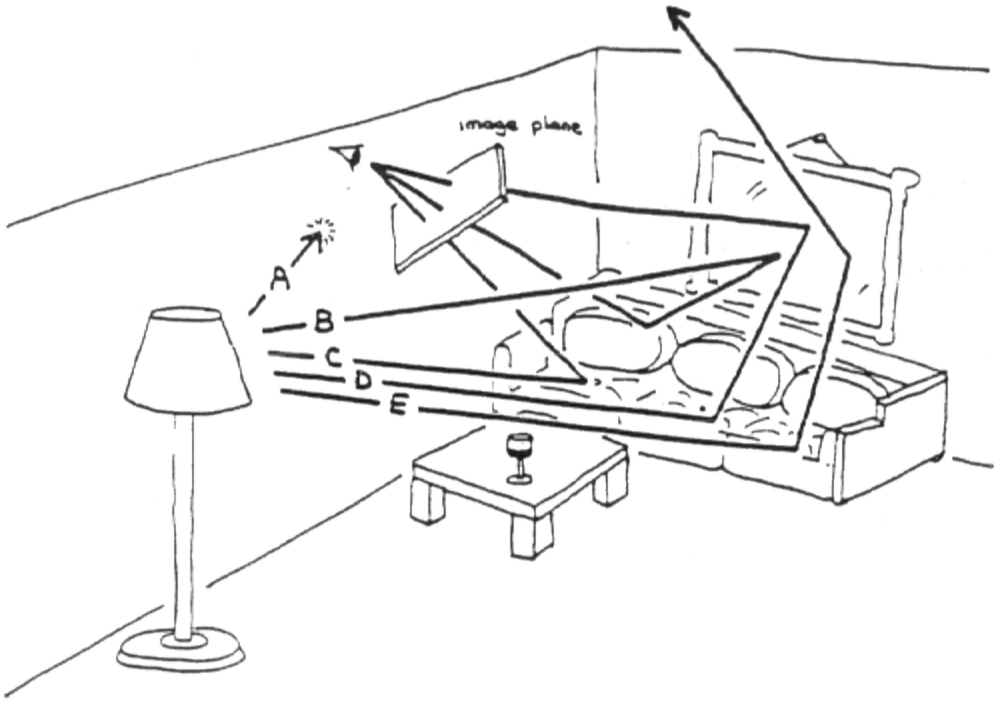


Figure 1 Illustration of Local and Global Illumination Models [7]

Figure 1 shows an example of local and global illumination. In this image, three of the five light rays depicted arrive at the image plane which defines what the viewer sees on their screen. Light ray C takes a single bounce off the couch, reflecting directly on to the image plane. This is an example of local illumination. Light rays B and D also arrive at the image plane after multiple reflections off various surfaces in the scene. Their contribution is part of the global illumination model for this scene.

In this project, we are also considering volumetric lighting effects. Volumetric lighting, which is light transported through a medium such as fog or water, complicates the model even further [4]. To illustrate how this works, imagine that the room depicted in Figure 1 is filled with smoke. All along every one of the five light rays shown in the image, some portion of the original light ray is now being lost, either absorbed by the smoke particles or scattered in a direction other than forward along the initial path. Furthermore, some of the light that has been scattered from other light rays is now being scattered into the path of the original ray.

In the scenario presented above, these complex volumetric interactions between light and smoke particles can be modeled reliably, given enough time, by a lot of complex scientific formulas [8]. When time is a factor, such as in a video game, a lot of simplifications must be made to the model in order to achieve an approximation of the real-world effect quick enough to be used in every frame [9]. One simplification that is often made when calculating volumetric effects is to only consider the last leg of each light ray's path. In Figure 1, this would be the final leg of rays B and

C as they travel from the couch to the observer as well as the last leg of ray D as it travels from the wall portrait.

1.2.2 Past Approaches to Implementing Fog

Due to the large and complex set of calculations involved in determining the illumination model, solutions are usually optimized by using approximations to accomplish the best overall effect. In years past, computer hardware was not sophisticated enough to perform these calculations within a reasonable amount of time, leading the engineer or game designer to find workarounds [10]. Modern techniques seek to strike a balance between hardcore mathematical modeling, advanced hardware techniques, and approximation. Many modern implementations perform volumetric calculations based on raymarching techniques.

1.2.3 Recent Approaches to Implementing Fog

Performed on a general-purpose GPU, raymarching is performed for each pixel, updating each pixel's final color based on light interactions with the medium that occur in the space between the fragment and the pixel that represents it [3]. This effect is often calculated as a part of the deferred rendering pipeline, which means that all the effect's calculations are performed after the vertex and fragment shaders have already computed the depth map, normal, and fragment colors current frame [11]. These techniques often incorporate a blend of real-time calculation and approximation, striving to find a balance between performance and artistry.

1.2.4 Our Approach to Understanding Fog

Given the many challenges and varied solutions discussed above, and more importantly the challenges that were not discussed, the best approach to learning about and building a system for creating fog is the hands-on approach. In the absence of a well-defined, standard approach to understanding and implementing volumetric fog, the best course of action is integrating experimental exploration with progressive refinement in both problem and solution spaces. This means evaluating where we are, researching what others have done, setting accomplishable goals and evaluation criteria based on our research, attempting to achieve the goals we set, then repeating the process.

1.3 Project Goals

This project has three main goals. The first goal is to build an understanding of volumetric fog and how this effect can be created in modern imagery using a GPU. Based on our review of literature encompassing the last ten years, the best examples of these effects often have their implementations protected as proprietary, trade secrets. What is most-often shared is high level discourse that provides an overview of the techniques used with minimal amounts of reusable code. Our second goal is to demonstrate our understanding by implementing an interactive system where these effects can be interacted with and examined. Our final goal is to provide a starting point for future researchers and enthusiasts by sharing both the lessons we have learned and the resources we have developed during this project.

1.4 Accomplishments

In the early phase of the project, we studied the physics, modeling and approximation of fog illumination. These studies gave us an understanding of both the complexities of the illumination model and a basic understanding of several approaches to approximating the different illumination qualities.

In the second phase of the project, we successfully implemented a system for creating volumetric fog using the Unity cross-platform game engine. Our system has several features that developers can enable and adjust to meet their project's needs. Our system can generate fog either scene-wide or within finely controlled fog volumes at specified locations within the scene. The density of the fog can be adjusted as a function of both distance and altitude. The system can generate random, variable density fog, and contains velocity settings to simulate wind effects. This system also supports fog interaction with multiple lights within the game scene, as well as shafts of light and shadow within the fog volume.

The system we have developed is but a first step for learning about volumetric fog. Many opportunities exist for improving both the performance and illumination model, as well as the addition of new features. Nevertheless, the existing system is already capable of generating realistic, performant fog effects for use in game and video projects.

1.5 Outline

This document is organized into chapters as follows:

Chapter 1: Introduction
- Overview of project motivations, challenges, goals, and accomplishments.

Chapter 2: Background
- External sources that have aided in the understanding and implementation of this project.

Chapter 3: Design
- Qualitative and quantitative performance goals for this project.
- Hardware and software toolset used in the project.

Chapter 4: Implementation
- Detailed discussion of how the methods were implemented in code.

Chapter 5: Results
- Comparison to the project's qualitative and performance goals.

Chapter 6: Conclusions
- Results and limitations of this project
- Future research and work

2 Background

This chapter discusses the characteristics of atmospheric fog, provides an overview of the general approaches to generating fog, and concludes with additional considerations for generating volumetric fog effects on a GPU.

2.1 Fog Characteristics

Several types of fog form in nature. The most common types of fog are advection fog, fogs of evaporation, and fogs that are a combination of both advection and evaporation [12]. Advection fog forms when a warm, moist air mass moves to a cold surface, such as air passing from a warm lake over a cold ground. As a result of the heat exchange with the cold surface, the air layers closest to the ground cool faster than the upper ones and, if the air mass is sufficiently moist, condensation starts. Fogs of evaporation appear when a large source of water vapor from a warm surface is present in a cold air mass. This type of fog formation is often encountered in sea regions at high latitudes. Examples of this type of fog are found in Norway, as well as around lakes, rivers and swamps.

Fog has several characteristics that are often identified by observers [13]. The most obvious characteristic is that fog causes reduced visibility. Another characteristic is the thickness or height of the fog, which can vary from being thin or wispy to being thick or wall-like. Variations in the density of the fog can occur due to both the conditions that the fog forms under and the distribution of the particles within the fog volume. Fog can be observed as it lifts or evaporates due to changes in the environment. In breezy conditions, fog can be seen to drift and swirl as it is blown by the wind. Lastly, variations in the occurrence, depth, and intensity of the fog can be observed at the same location over time.

2.2 Current Approaches to Implementing Fog

Video games typically use one of five different techniques for generating fog [10]. These techniques, which will be described in more detail, are:

- Billboarding
- Particle Emission
- Distance-Based Fog
- Post-Effect Image Based Solutions
- Raymarching / Volumetric Fog

2.2.1 Distance-Based Fog

An classical technique known as distance-based fogging [14], [15] can be achieved using custom surface shaders. The way this technique works is that as objects move away from the camera, their color is blended with the color of the background, causing the object to appear to fade into the distance. This technique was widely used in older games to mask the shortcomings of the graphics hardware. One benefit is that game objects no longer pop into existence as they move into the camera's viewing frustum. A second benefit of this approach is that the game could release instance resources once they moved a certain distance away from the camera.

The distance-based fogging technique has good performance, as the only variable in the calculation is the game object's depth in the scene.

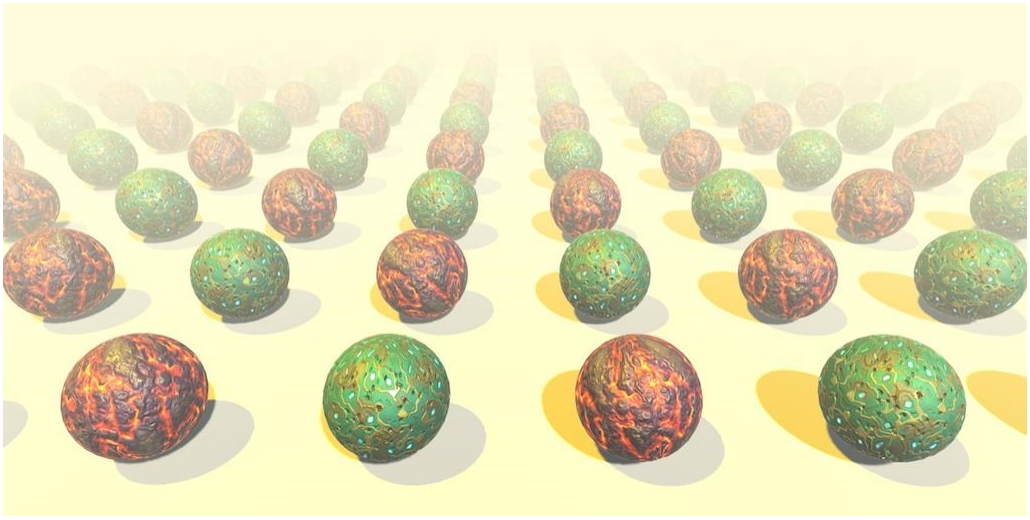


Figure 2 Distance-Based Fog [15]

Figure 2 shows an example of the distance-based fogging technique. Each of the spheres shown in the scene uses a shader that blends its default texture with the color of the background based on the sphere's distance from the camera.

The downside to this technique is that it is does not result in a very realistic, believable fog [10]. The technique relies on too many simplifications, such as having a single, totally uniform light source and constant or constantly-varying fog density [10].

2.2.2 Billboarding

Billboarding, a technique used in many games, involves drawing particle effects such as smoke or fire onto a flat surface that is always positioned facing towards the camera [16]. This technique works great when the player is always moving in the same direction or when the effects are always viewed off in the distance. This technique is not recommended for use close to the game camera. An example of this would be in a 3D game with the player walking in a circle around the game surface with the applied effect. In this instance, the 2D effect (for example an animated sprite representing a burning tree) would always be rotating to face the player. A second example where this technique is not recommended is when the 2D game surface intersects with other game objects, creating one or more hard edges in the scene that ruins the authenticity of the effect. Still, there is at least one technique called soft particles that can help to alleviate this condition [16]. Soft particles is a technique that involves fading the billboard near the intersection with other game objects, such as near buildings or the player.

Figure 3 Billboarding (Left) and Billboarding with Soft Particles (Right) [16]

Figure 3 shows an example comparing local fog in a scene. The image on the left shows the intersection of the fog billboard with the ground, which results in a hard edge that makes the fog look unrealistic. The image on the right shows the same intersection using the soft particles technique, which results in a more believable fog effect.

2.2.3 Particle Emission

Particle emitters can be used to create local, billowing fog effects [17]–[19]. The effect would be like how ground fog can be created in stage productions using a fog machine. This technique can be quick to implement with good visual quality for small, local effects.

Figure 4 Particle-Emission Fog (Captured at 3:30) [19]

Figure 4 shows an example of fog effects created using particle emitters.

While the use of particle emitters can lead to visually pleasing effects with minimal effort, the game performance can be negatively impacted when using large numbers of particles in the scene at one time.

2.2.4 Post-Effect Image Based Fog

This technique was popularized by the Unreal Engine 3.0 and the CryEngine [10]. It relied on the use of photographic post-effects, primarily bloom and radial blur. While capable of producing some excellent results under the right conditions, this technique had many weaknesses. First, the technique was purely an artistic effect and not based on physics. Second, this technique did not work well for close light sources. Also, it does not compose well with fog or transparent objects. Lastly, the effect does not work at all when the camera is not facing the direction of the light source.

Figure 5 Post-Effect Image-Based Fog [10]

Figure 5 show an example of fog based on the use of post-effect image techniques. The use of light bloom and radial blurring causes the objects off in the distance to appear faded and indistinct.

2.2.5 Raymarching / Volumetric Fog

Volumetric fog is a modern technique that is most often implemented as a post-effect, meaning that the fog density and color are calculated after the camera has calculated the depths, normals, and colors for all the scene objects that are to be rendered. The technique uses the GPU to perform per-pixel calculations of light, shadow, color, and density for the fog to be rendered. Using the raymarching technique, multiple samples of the fog are taken along a vector that runs between each pixel and the game object it represents. These results are collected, and the shader uses the results to calculate a final scene rendering that blends the fog with the initial scene.

Using this technique, it is possible to generate photo-realistic fog. A robust implementation can simulate many real-world phenomena such as light extinction and scattering. However, several limitations exist in current implementations [10]. First, most implementations are limited to generating close range “volumetric shadows” or “light shafts” and are not based on physics. This is due to the large number of samples and calculations that would be required to support fog over long distances. This high number of calculations would lead to a noticeable degradation in overall system performance. Second, most implementations are simplified by assuming that only a single, directed light source exists. Furthermore, modern implementations typically use down-sampling of depth maps to decrease the number of calculations, thereby increasing the overall performance. The result of this compromise is that it often leads to under-sampling, perspective aliasing, and edge artifacts. Lastly, this technique does not work with forward-rendered objects,

especially particles and transparent objects, since depth maps only retain depth information for the objects that are closest to the camera.

Figure 6 Volumetric Fog [20]

Figure 6 shows an example of volumetric fog created using the Unreal Engine 4. The generated image demonstrates many of the qualities of real fog: reduced visibility and detail over distance, visible shafts of light and shadow, light transmission, absorption and scattering effects.

2.2.6 Summary

While all the approaches to approximating fog have their own merits, the volumetric approach to simulating fog is both the most recently developed and the best technique to use. This is because volumetric fog follows and approximates real-life physics. This basis makes the volumetric approach the most general approach and the easiest to expand to support all the characteristics outlined in section 2.1. Based on this reasoning, the rest of this chapter will focus on volumetric fog.

2.3 Volumetric Fog Considerations

This section discusses our study of some additional considerations that go in to implementing photo-realistic fog. This expands on the introduction to raymarching and volumetric fog that were already discussed in 2.2.5 Raymarching / Volumetric Fog.

2.3.1 Graphics Rendering Pipeline

In the modern GPU, two common paths exist in the graphics rendering pipeline, the forward rendering path and the deferred rendering path [11]. The forward rendering path is the typical rendering technique that most engines use. You supply the graphics card the geometry, it projects it and breaks it down into vertices, and then those are transformed and split into fragments, or pixels, that get the final rendering treatment before they are passed onto the

screen. Similarly, the deferred rendering path also processes the scene geometry using its vertex, geometry, and fragment shaders. However, unlike the forward rendering path, in deferred rendering the fragment shader delays performing the lighting pass(es). This allows for additional calculations to be performed after the formation of the depth maps, normal maps, and colors has already been accomplished. Once these additional calculations are completed, the fragment shader then performs the lighting pass(es), culminating in the final rendering.

The deferred rendering path offers several advantages [21]. First, the scene geometry is decoupled from lighting [22]. This means that there is no limit on the number of lights that can affect a game object. Furthermore, all lights are evaluated per-pixel, which means that they all interact correctly with normal maps. Also, all lights can have cookies and cast shadows. Lastly, the processing overhead of lighting is directly proportional to the number of pixels that the light shines on, so performance can be improved by keeping lights small.

The deferred rendering path also has some disadvantages [21]. First, there is no support for anti-aliasing. In modern game engines, anti-aliasing is typically accomplished using multi-sampling techniques on the scene geometry. That information is lost to the deferred rendering path, which operates on the data contained in screen space. Also, deferred rendering doesn't work for semi-transparent game objects, which must still be generated using the forward rendering path. Additional limitations also can exist depending on the game engine or rendering software being used.

Despite the limitations, volumetric fog is usually implemented using deferred rendering [23]. This is because there is no need to generate separate depth, normal, and shadow maps with deferred rendering. They are already generated as a part of the forward rendering path.

2.3.2 Modeling of Atmospheric Absorption and Scattering of Light

In the real world, light interacts with the atmosphere in complex and fascinating ways, creating unique phenomena [8], [24].

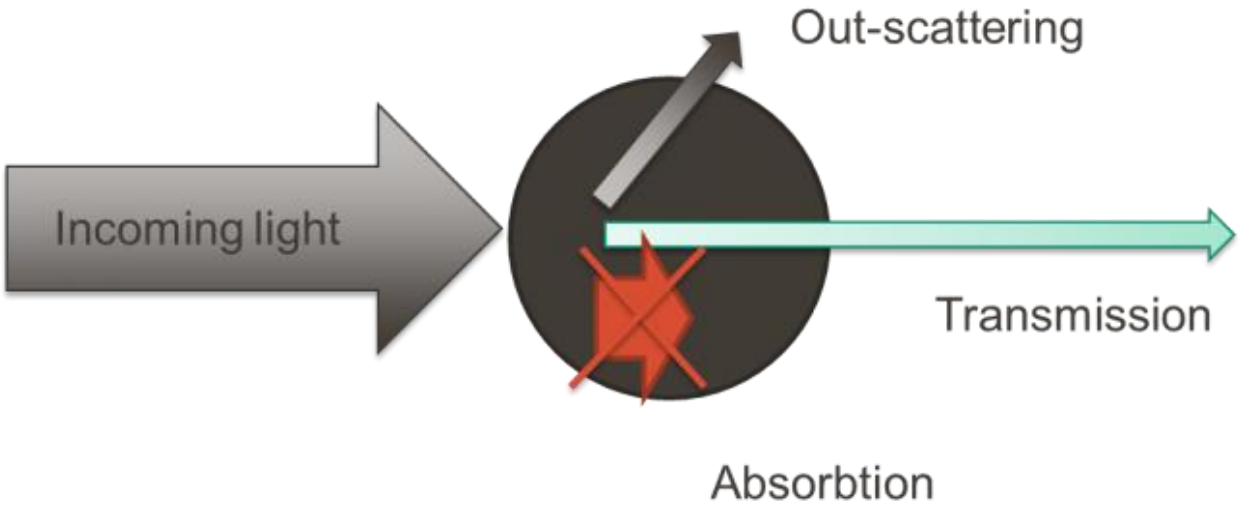


Figure 7 Light Interacting with a Participating Medium [9]

Figure 7 shows the complex interaction of a ray of light as it intercepts a molecule or particle in a participating medium, such as a water particle in a fog bank. As can be seen, some of the incoming light gets scattered away from the original ray's direction, some gets absorbed, and the remaining portion of light is transmitted in the same direction as the original ray of light.

The amount of light that reaches the observer can be modeled using the Beer-Lambert Law [25].

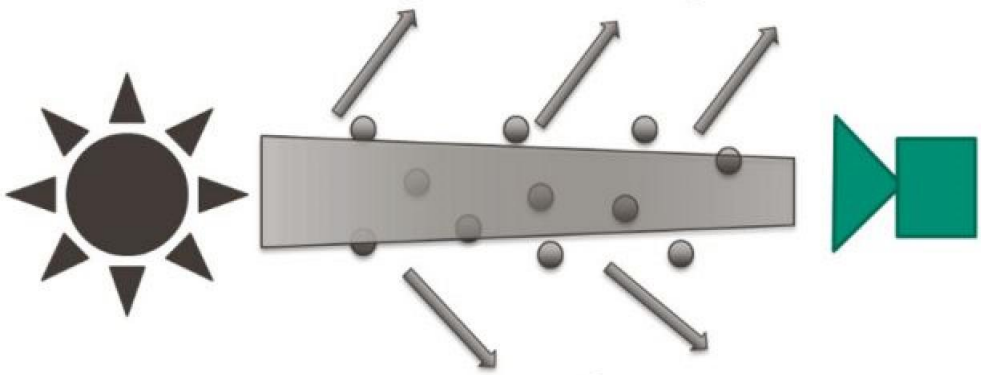


Figure 8 Transmission of Light in a Participating Medium [9]

Figure 8 shows a depiction of the Beer-Lambert Law. This law defines the proportion of light that is transported through the medium to the total incoming light from a given direction. It can be observed that light extinction is exponentially related to the distance the light travels in the medium [9], [25]. The equation modeling this behavior is:

$$I = I_0 * e^{-\beta e * x}$$

Where:

- I_0 – Initial light intensity
- x – Distance traveled through the participating medium
- βe (the extinction coefficient) – Sum of the scattering and absorption coefficients

Multiple types of scattering can occur, each dependent on the particles that form the participating medium and the wavelengths of light taking part in the interaction. The two commonly observed types of scattering in the atmosphere are Rayleigh and Mie scattering [25]. Rayleigh Scattering focuses on the electromagnetic wave interaction with the electron cloud of an atom. In these cases, the diameter of the particle being interacted with is much shorter than the wavelength of the electromagnetic wave. As a result, blue light is the most highly scattered due to its shorter wavelengths, whereas red light is mostly not scattered. Mie Scattering occurs when the particle diameter is greater than or equal to the wavelength of the electromagnetic wave. This scattering is anisotropic, which means that the light gets scattered more in some directions than others. The anisotropic effect increases with the size of the particle in the participating medium.

2.3.3 Summary

This section introduced several details that are important to the implementation of volumetric fog. Section 2.3.1 discussed the graphics rendering pipeline, the deferred rendering path, and the reasoning behind using deferred rendering to implement volumetric fog. Section 2.3.2 discussed the some of the physics behind the interactions between light and the atmosphere, which is important to understand so that we can approximate these interactions in our fog system.

2.4 Chapter Summary

This chapter summarizes the characteristics of fog which can vary as a function of both space and time. Furthermore, it discusses both the best approach to approximating fog and the fundamental model needed to create the approximation.

The next chapter discusses the goals for the project and the tools selected to achieve those goals.

3 Design

This chapter describes both the qualitative and performance goals this project aims to achieve. This chapter also describes the software tools used in the project and some of the factors that were considered in choosing them.

3.1 Project Goals

3.1.1 Qualitative / Subjective Goals

In the real world, fog has many qualities that a believable simulation should be capable of emulating:

(1) The density of fog can vary greatly as a function of position, distance, altitude, and time.
(2) The visible color of fog reflects the local and global lighting it interacts with.
(3) Local variations in density can occur, and are more perceivable when outside agitators, such as wind, cause the fog to swirl or drift.
(4) Under highly dynamic lighting conditions, shafts of light or shadow can be observed.

3.1.2 Quantitative / Performance Analysis

Our performance goal is to demonstrate a performant fog simulation for a base scene consisting of hundreds of game objects at a resolution of 1920 x 1080. Achieving a system that is capable of functioning at HD resolution is important as the visual quality of a video game can often be the deciding factor in its success or failure as a profitable product [26].

Some properties of our base scene are:

- Simple scene consisting of:
  - Ground plane
  - Several hundred geometric objects of varying size
  - Single directed light source
- Scene-wide volumetric fog:
  - Linear height-density falloff
  - Raymarching using a maximum of 128 steps per pixel
- Running in the Unity game preview window

The base scene will be analyzed with these settings, and subsequent simulations will be analyzed to determine the performance cost associated with adding in new features or qualities to the fog simulation.

See Appendix A – Development System for a complete description of the development system.

See Appendix B – Base Scene for a complete description of the base scene and script configuration.

Other features, such as shadows and local density variations, will have their performance analyzed as a direct cost comparison to the performance of the base scene.

3.2 Toolset

This project uses two tools to study and implement volumetric fog in 3D game environments:

- Unity [27] (version 2018.2.16f1), cross-platform game engine
- Microsoft Visual Studio 2017 was used for HLSL and C# code development.

This toolset offers many benefits. First, both Unity and Microsoft Visual Studio are free to use. Also, these two tools work well together, with Visual Studio now being offered as part of the Unity installation process. These tools are available for both Mac and PC, supporting the design and development process no matter what platform you work on. Being free has allowed both tools to accumulate a large worldwide user base. A large user base means good customer support from both the developers and the community, including the ability to find many free resources such as wikis, blogs, tutorials and other information sources.

One final benefit for our team was a familiarity with Unity and Visual Studio due to previous coursework using these tools. This was a major deciding factor in choosing Unity over competing products, such as the Unreal Engine 4 [28] produced by Epic Games. The benefit of this decision is that it allowed us to hit the ground running on the research and implementation of volumetric fog without the additional overhead of learning the development platform first.

3.3 Chapter Summary

This chapter summarizes the qualitative and quantitative goals for the project. This chapter also describes the toolset selected to achieve the project goals and presents the reasoning behind these choices.

The next chapter discusses the implementation of the volumetric fog system.

4 Implementation

In this project, we use the modern technique called raymarching to calculate the density and color of the fog in the scene. Raymarching is the process of taking incremental steps along the length of a vector defined between one pixel and the object it displays. At each step, environmental measurements and calculations are performed and their results are accumulated. The raymarching process is performed for every pixel that gets rendered to the screen. This makes this process computationally expensive. This process is carried out using the computer system's graphics processing unit (GPGPU), with the calculations being performed using a compute shader.

Raymarching is performed using the deferred rendering pipeline. This means that the calculations are performed after the initial frame has already been assembled. At this point, the GPU still has access to several components, such as the scene lights, the camera depth map, and the light source depth (shadow) maps. It is possible for the system to pass in other scene information, too, using scripts.

After the effects of the fog are determined for each pixel, the compute shader returns pixel information that can be blended with the initial frame rendering to create the final composite image containing the fog.

4.1 Fog Density

The density of the fog is a function of the number of steps taken within the fog volume, the size of each step, and the average density of the fog volume itself. The size of one step, calculated individually for each pixel, is:

$$stepSize = \frac{distanceToFragment}{numberOfSteps}$$

If the fog is confined to a box volume, then we also need to determine whether the step being evaluated lies within the fog or not. This check is performed using the function *NearestEdgesFromFogCG*, which compares the evaluation point in world space coordinates to the location of the fog volume. This function returns a float4 containing the three distances from the closest box edges in the x, y, and z directions and either a 1 or 0 value to designate whether the evaluation point lies within the fog volume or not.

4.1.1 Distance-Based

The distance-based fog density setting determines how much fog affects each pixel color based on how much fog is between the object the pixel represents and the scene camera. Objects that are closer to the camera will appear mostly the same color as when fog is not present, whereas objects that are far off in the distance will have their color affected more by the fog. This will cause objects to become more indistinct as they move away from the camera, appearing to fade into the fog once they reach a certain distance. The following image shows a real-world image of distance-based fog effects:

Figure 9 Distance-based fog density [29]

To determine the density of fog applied to each pixel, we first need to know how much fog the developer wants to apply per step. This is determined by a user supplied value, *distanceToFogSaturation*. With no other user settings enabled, *distanceToFogSaturation* represents the maximum distance at which objects are no longer visible when viewed through the fog. This leads to the following calculation for fog to be applied at each step within the fog volume:

$$fogDensityPerStep = \frac{stepSize}{distanceToFogSaturation}$$

The total density of fog applied to each pixel is:

$$finalFogDensity = numberOfStepsInFog * fogDensityPerStep$$

4.1.2 Height-Based

The height-based fog density setting is used to create a thinning effect at the top of the fog volume. Objects that are near the bottom of the fog will be have their color affected more by the fog than objects that are near the top of the fog volume. The following image shows a real-world image of the height-based density falloff on a fog bank:

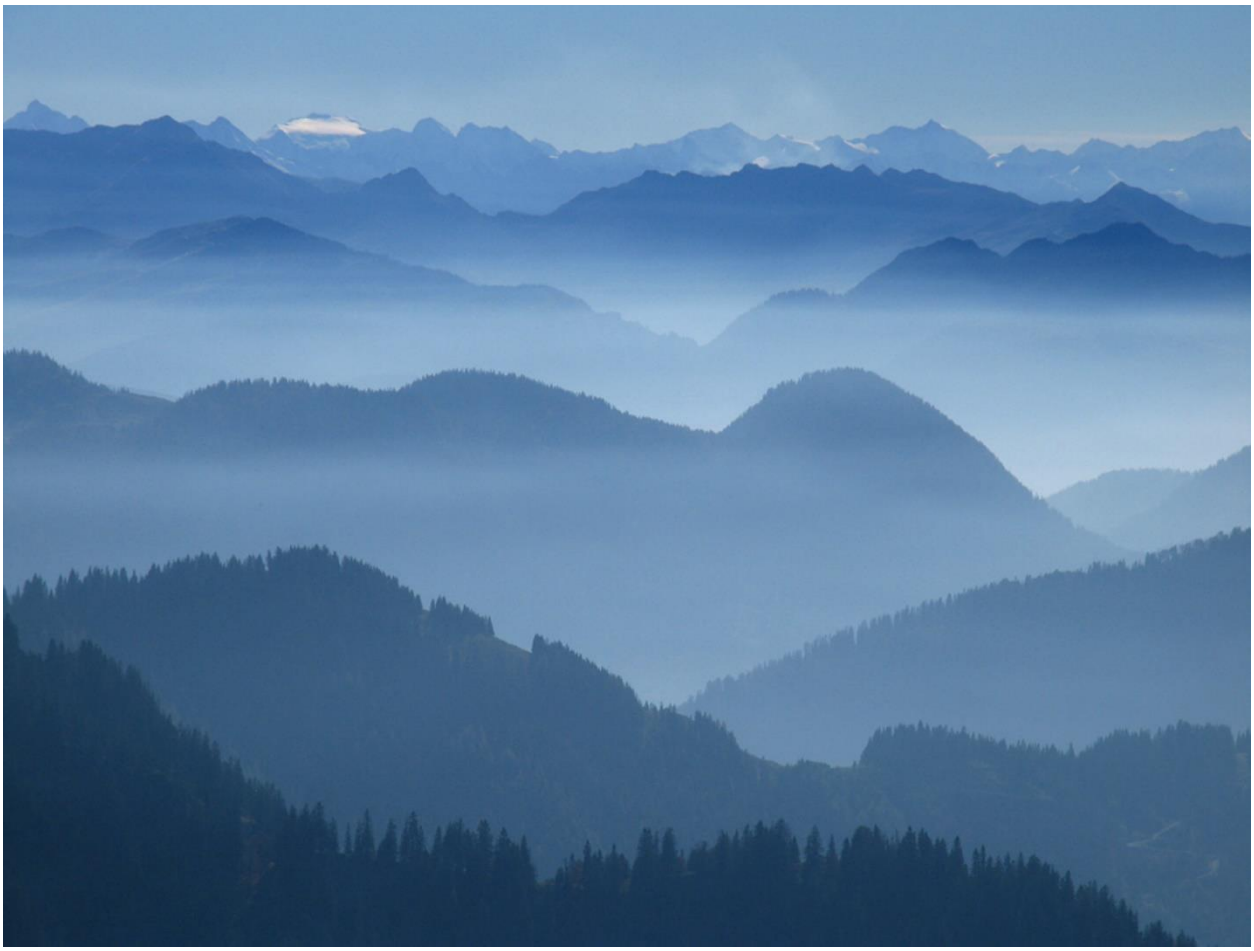


Figure 10 Height-based fog density [30]

When height-based density falloff is enabled, the density of the fog will be attenuated either linearly or exponentially above a height requested by the user. This is performed at each step using the function *FogHeightDensity*, which takes a *float3 evaluationPoint* as its input. The developer provides two inputs, *heightToStartFalloffAt* and *YFalloffDistance.*

heightToStartFalloffAt, when enabled with scene-wide fog, is the world position y value at which to start fog density adjustments. When using a boxed fog volume, this value is the y position relative to the bottom of the fog box at which to start density adjustments.

YFalloffDistance is the distance to perform the linear or exponential falloff over. If using a boxed fog volume, this distance may end up being reduced if the *heightToStartFalloffAt + YFalloffDistance* exceeds the total height of the box.

Given these two inputs, the first step is to determine if the evaluation point lies within the fog volume using the function, *NearestEdgesFromFogCG*. If the evaluation point lies outside of the fog volume, then the function returns a value of 1 to indicate that no attenuation factor is applied. Secondly, we need to determine if the evaluation point is in the attenuation area, that is, above *heightToStartFalloffAt*. If not, the function again returns a value of 1 (no attenuation

factor applied). Lastly, we need to determine the attenuation factor. If the user has selected a linear attenuation, then the equations are:

$$heightFactor = \frac{1}{YFalloffDistance}$$

$$heightDensityAttenuation = 1 - (evaluationHeight - heightToStartFalloffAt) * heightfactor$$

If the user has selected an exponential attenuation factor, then the equations are:

$$heightFactor = \frac{7.5}{YFalloffDistance}$$

Note: This value was determined experimentally to generate a falloff that approaches 0 at the requested *YFalloffDistance*.

$$heightDensityAttenuation = e^{-(evaluationHeight - heightToStartFalloffAt) * heightfactor}$$

4.1.3 Edge-Density

This factor can be used when the fog is limited to a boxed fog volume. Just as fog density varies as a function of altitude, thinning near the top of the fog bank, natural fog also appears thinner around the edges of the fog bank as well, increasing in density as one enters the fog bank.

Figure 11 Edge-based fog density [31]

This factor is calculated at each step using the function *FogEdgeDensityXZ*, which takes a *float3 evaluationPoint* as its input. The function provides either a linear or exponential attenuation factor in almost identical fashion to the height-based attenuation factor described in the previous

section. The main difference is that now the user must have provided both an x and a z position to start adjusting density at. Also, this position is not a world-based coordinate, but is instead a percentage value of the distance from the fog volume center of gravity to the edge of the fog volume in the requested direction.

Refer to the previous section (4.1.2 Height-Based) for a detailed description of the methodology.

4.2 Color

In nature, the color of fog is determined by the color of the light it scatters towards the observer. The following image shows this effect, with the city lights causing the fog to take on a white color and the bridge lighting causing the fog to take on a warmer orange color.

Figure 12 Fog color based on light interactions [32]

Our system determines the color of the fog based on the color and intensity of the light source interacting with it. With no other settings enabled, the final fog color per pixel is determined using to the following set of equations:

In the *CalculateFogDensityAndColor* compute shader:

$$litFogColor = DirectionalLightRGBColor * DirectionalLightIntensity$$

$$stepFogColor = litFogColor$$

Note: Initial stepFogColor is assumed to be the color value of the directed light source

$$finalFogColor = \sum stepFogColor * stepFogDensity$$

In the *ApplyFogToScene* compute shader, the color of the fog is blended with the original pixel color based on the final fog density for the pixel:

$$pixelColor = originalPixelColor.rgb * (1 - fogSample.a) + fogSample.rgb * fogSample.a$$

The pixel maintains the original alpha value from the forward rendering process:

$$pixelAlpha = colorSample.a$$

Note that these equations are in their final form. As other modifications are made to the fog equations for color and density, they are still applied to the final pixel value according to these two equations.

4.2.1 Shadows

In nature, crepuscular rays are rays of light and shadow that can be seen when the sun shines on objects that rest in a participating medium such as fog or dusty air. The following image shows an example of this phenomenon.

Figure 13 Crepuscular rays of light and shadow [33]

In this project, the effects of shadows within the fog volume are only determined for the main, directional light source. A script, *CopyShadows*, is attached to the main, directional light source that causes it to add its shadow map to a global command buffer. This buffer can then be accessed within the compute shader, allowing us to use the shadow map to determine if the evaluation point is within a shadowed area or not as seen from the light's viewpoint.

Four steps are used to determine the fog color when shadows are enabled:

Step 1: Gather the number of shadow cascades and their weights in the current scene.

Unity uses the term, cascade, to refer to the shadow maps for a light source. Each light source can have one, two, or four cascades. This provides a system for having highly detailed shadows close to the camera while allowing for lesser quality and higher performance shadow mapping farther away from the camera. The function, *GetUnityShadowCascadeWeights*, takes the depth in the scene based on the view space coordinate system as an input and returns a float4. A value of 1 in one of the array's values means that the evaluation point is located within the associated shadow cascade.

Step 2: Use the shadow cascade weights and the evaluation position in world space coordinates to determine the evaluation position in the shadow map coordinate system.

The function, *GetUnityShadowMapCoordinates*, uses these two inputs to determine the coordinates to use when polling the shadow map.

Step 3: Determine if the evaluation point is in the lit or shadowed region for the directed light source using the built-in Unity shader function, *UNITY_SAMPLE_SHADOW*.

The return value, shadowStrength, is 0 if the evaluation point is shadowed, otherwise it is 1.

Step 4: Determine the final fog color for each step using the following equation:

$$stepFogColor = Lerp(litFogColor * (1 - \_ShadowStrength),\ litFogColor,\ shadowStrength)$$

where *_ShadowStrength* is a value between 0 and 1, set by the developer.

The final fog color for the pixel is still accumulated according to the equations presented previously in Section 4.2.

4.2.2 Ambient Fog

This feature allows the developer to add light to the entire fog volume. One use for this would be for scenes utilizing fog at night, when the color of the directed light source is black. The developer could add a small amount of ambient light to the otherwise black fog, causing the fog to glow with the requested color and intensity. Glowing fog is not something natural, but the effect is used in many movies and games. One example of ambient fog can be seen in the following game image.

Figure 14 Ambient fog [34]

The contribution of ambient light is determined by the following equation:

$$stepFogColor = Lerp(stepFogColor, ambientFogRGBColor \;\; ambientLitFog)$$

where *ambientLitFog* is a value between 0 and 1, set by the developer, and ambientFogRGBColor is a vector with three values (0 to 1) representing the color of the ambient fog.

4.3 Noise (Variable Density)

This project uses noise to create local variations in the density of the fog. This adds to the believability of the fog by simulating the variability of fog in the real world. This feature allows the developer to add perceivable movement to the fog, simulating wind. The following image shows how this looks in nature.

Figure 15 Variable-density fog [35]

4.3.1 Simplex Noise

Perlin noise and simplex noise were developed by Ken Perlin, with simplex noise being a more performant version of Perlin noise. Simplex noise improves on the original by supporting the generation of higher dimensions of noise (3D and 4D) with almost the same performance as the original 2D Perlin noise [36].

This project uses a CgFx/HLSL implementation of simplex noise [37], ported by Eliot Lash from the GLSL version authored by Stefan Gustavson and Ian McEwan at Ashima Arts [38].

To use the library, include the file *noiseSimplex.cginc* in the compute shader. The function, *snoise()*, returns a 3D noise value based on a float3 input value representing a position within the pseudo-random noise field. It should be noted that the simplex noise function returns values between -1 and 1, so adjustments will need to be made to get the values into the range of 0 to 1.

4.3.2 Noisy Fog

Using a pseudorandom noise function, we can implement local variations in the fog density in two ways, additive and subtractive. Additive noise increases the density of the fog for each step based on the noise value, while subtractive noise decreases the fog density for each step. In this project, both ways have been implemented for comparison.

Step 1: Determine a noise value for the evaluation position.

$$simplexNoiseEvaluationPosition = evaluationPositionInWorldSpace$$

$$simplexNoiseValue0to1 = \frac{snoise(simplexNoiseEvaluationPosition)}{2} + 0.5$$

Step 2: Use the noise value to modify the fog density for the current step.

noiseStrength is a developer-assigned value that is used to control the overall contribution of the noise to the density for the current step.

Additive Noise:

$$stepFogDensity \mathrel{*}= (1 + noiseStrength * simplexNoiseValue0to1)$$

Subtractive Noise:

$$stepFogDensity \mathrel{*}= (1 - noiseStrength * simplexNoiseValue0to1)$$

Here it should be noted that the relationship of one noise value to another obtained from the simplex noise function is higher when the evaluation points are closer together. This knowledge allows us to adjust the fog appearance by adjusting the *simplexEvaluationPosition* within the noise field. Our system gives control of the *simplexEvaluationPosition* through the *noiseSize* control. This modifies the original noise value equation as follows:

$$simplexNoiseValue0to1 = \frac{1}{2} * \frac{snoise(simplexNoiseEvaluationPosition)}{noiseSize} + 0.5$$

Larger values of *noiseSize* cause the evaluation points to be closer together in the noise field, causing changes in the fog density to happen over longer distances in the scene.

4.3.3 Drifting Fog

A velocity can be given to the fog, simulating wind. Fog drift is implemented in this project by causing the *simplexEvaluationPosition* to change as a function of time. In the control panel, the developer sets the values of the 3-dimensional vector, *noiseVelocity*. Then, the equation that governs the calculation of the evaluation position in the noise filed is modified as follows:

$$simplexNoiseEvaluationPosition = \begin{bmatrix} evaluationPositionInWorldSpace.x - noiseVelocity.x * \_Time.x \\ evaluationPositionInWorldSpace.y - noiseVelocity.y * \_Time.x \\ evaluationPositionInWorldSpace.z - noiseVelocity.z * \_Time.x \end{bmatrix}$$

4.4 Additional Light Sources

The fog volume can be made to support additional light sources and types. To demonstrate this, this project supports the addition of a single point light source. It should be noted that these additional light sources do not contribute to the one shadow map used by the directed light source. The additional lights only affect the fog intensity and color for any given step.

4.4.1 Point Light

The following image shows an example of fog with local illumination. The lights in the image are having their light reflected towards the observer, causing local changes in the perceived color of

the fog. The intensity of the light falls off approximately as a function of the distance squared as it moves away from the light source.

Figure 16 Night fog with local illumination [39]

A single point light is implemented as proof of concept that our system can support multiple light sources. A point light was selected as it is the easiest to work with due to its omnidirectional shine. The compute shader needs four inputs from this light source:

- pointLightLocation
- pointLightRange
- pointLightIntensity
- pointLightColor

And three attenuation factors provided from the user:

- $A_{constant}$ – constant attenuation factor
- A_{linear} – linear attenuation factor
- A_{exp} – exponential attenuation factor

Then, at each step measured within the fog volume:

Step 1: Determine the direction to the point light.

$$pointLightDirection = pointLightLocation - evaluationPositionInWorldSpace$$

Step 2: Determine the distance to the point light source.

$$distanceToLightSource = Length(pointLightDirection)$$

If the evaluation point is within range of the point light source:

Step 3: In the real world, light attenuates inversely as a function of the distance squared [40].

$$pointLightIntensityAtCurrentLocation = \left(\frac{pointLightIntensity}{distanceToLightSource^2}\right)$$

However, this equation does not produce a good result in 3D graphics. This equation will be modified by adding in some additional ambient and linear light [40]. All three attenuation factors can be adjusted directly by the developer. The resulting equation is:

$$pointLightIntensityAtCurrentLocation = \frac{pointLightIntensity}{pointLightAttenuation}$$

Where:

$$pointLightAttenuation = A_{constant} + A_{linear} * distanceToLightSource + A_{exp} * distanceToLightSource^2$$

Step 4: Blend the point light's color with the current fog color based on the intensity of the point light at the current evaluation point.

$$stepFogColor = Lerp(stepFogColor, pointLightRGBColor, pointLightIntensityAtCurrentLocation)$$

This implementation chooses to give preference to the point light source over the global light source. This is because the directed light source would overwhelm the point light source in most cases if a ratio of the two intensities was used instead.

4.5 Performance

Large number of calculations can overwhelm the hardware, causing the framerate to drop as more features are used. One way to mitigate the performance hits is to use shader features.

4.5.1 Shader Features

Shader features in HLSL allow the GPU to compile different versions of the compute shader depending upon which keywords the user has enabled. These compilations are done at runtime, allowing the compute shader to quickly switch between different features on the fly. When a keyword is enabled or disabled, an entirely new version of the compute shader is compiled and run, adding or removing blocks of code completely from execution. This directly impacts the performance (FPS) of the compute shader and, by extension, the overall program execution.

Shader features can be toggled on or off by enabling or disabling keywords from a Unity script.

Sample Code: This code is used to enable the processing of additional light sources.

```
Bool allowExtraLights;       // This value gets set by the user
If (allowExtraLights)
{
    calculateFogDensityAndColor.EnableKeyword("MULTIPLE_LIGHTS");
} else {
    calculateFogDensityAndColor.DisableKeyword("MULTIPLE_LIGHTS");
}
```

4.6 Quality Tuning

At times, it may be possible to see visible artifacts from the raymarching process. Three examples of this are:

- Looking through a local fog volume at an angle
- Looking through shadows cast by larger scene objects
- Looking through the changing height-density towards distant objects or the skybox

At these times, what the user is seeing is the edges of the planes created by raymarching across a sparse scene where many of the steps have a similar or identical step size. One solution to this problem is to add a randomized offset to each evaluation point.

4.6.1 Randomized Evaluation Points

Randomizing the evaluation point by up to +/- half the step size will soften or even eliminate these visible artifacts. This solution is not perfect and will introduce other artifacts, as positions on the boundaries of the fog volume or the shadowed area will potentially poll in an area without fog or shadow (or vice versa). This will lead to a spotted effect along the boundaries. This spotted effect, in general, is preferable to the harsh appearance of having planes at the step (evaluation) locations.

Randomizing the evaluation points, in this project, is performed by adding an offset value of up to +/- 1/2 * step distance to every evaluation point. This random offset is determined by polling a 2D random noise texture (provided by the user) composed of spots of varying gray values. The position polled in the 2D texture is calculated randomly based on the coordinates of the current evaluation point (x, y, z) and step number, i.

4.7 Chapter Summary

This chapter summarizes the implementation the volumetric fog system.

The next chapter discusses the results of the volumetric fog system, relating the quality and performance of the system to the design goals presented in chapter 3.

5 Results

This chapter discusses the system we have implemented for generating volumetric fog. It covers the hardware and system configuration, followed by a qualitative and quantitative analysis that relates our achievements to the stated design goals.

5.1 System Setup

All results were gathered from the development machine, an ASUS VivoBook Pro 15 N580VD. A full hardware specification can be found in Appendix A – Development System.

In Unity, we constructed a basic scene consisting hundreds of Unity 3D primitives. These objects are built up to simulate a forest scene with a cottage deep in the woods near the base of four giant mountains. This base scene will be used to test different qualities of the simulated fog. The scene also consists of one directed light source and an optional (disabled) point light source. See the following image, which shows this basic scene configuration.

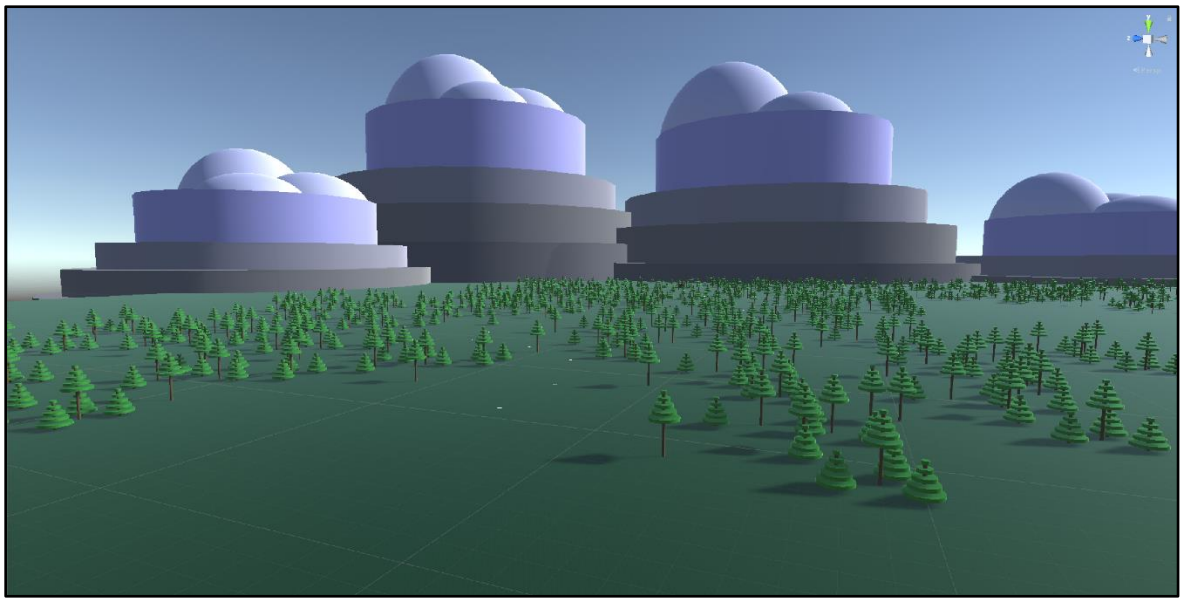


Figure 17 Basic scene configuration in Unity

The main fog script is attached to the camera, which is attached to the player object, so it can be moved around the scene. The only feature that will be enabled in the fog script, initially, is scene wide fog. There is also a script attached to the directional light source that copies the light's shadow maps into a command buffer, making the information available to the shader. For a more complete description of the system configuration, see Appendix B – Base Scene.

5.2 Qualitative / Subjective Analysis

5.2.1 Qualitative Goals (Redux)

As mentioned in Section 3.1.1., here is the set of fog qualities that our system is emulating. These goals will be mentioned again in the subsections that follow as we discuss the results of our system implementation.

(1) The density of fog can vary greatly as a function of position, distance, altitude, and time.
(2) The visible color of fog reflects the local and global lighting it interacts with.
(3) Local variations in density can occur, and are more perceivable when outside agitators, such as wind, cause the fog to swirl or drift.
(4) Under highly dynamic lighting conditions, shafts of light or shadow can be observed.

5.2.2 Distance-Based Fog

Distance-based fog is the effect that determines how much fog color to apply to each pixel based on the distance to the nearest object that pixel represents. For a constant fog density, this feature will cause objects nearby to appear with normal color and clarity while more distant objects will become indistinct and muted in color. Objects far off in the distance may no longer be visible at all to the observer if the fog is sufficiently dense.

Figure 18 Distance fog comparison between natural fog (left) and system-generated fog (right)

Figure 18 is a comparison between the natural fog shown in Figure 9 and a sample image generated within our system. This comparison highlights the distance-based fog quality that was discussed previously. The objects in our scene are more distinct the closer they get to the camera, while distant objects appear soft and muted. The most distant trees are positioned right at the edge of visibility, some of them barely discernable to the observer.

5.2.3 Height-Based Fog Density

Height-based fog density is the effect that determines the rate at which fog thins at the top of the fog volume. Real fog naturally thins at its top as a function of altitude based on environmental factors as well as the composition of the molecules it is made of.

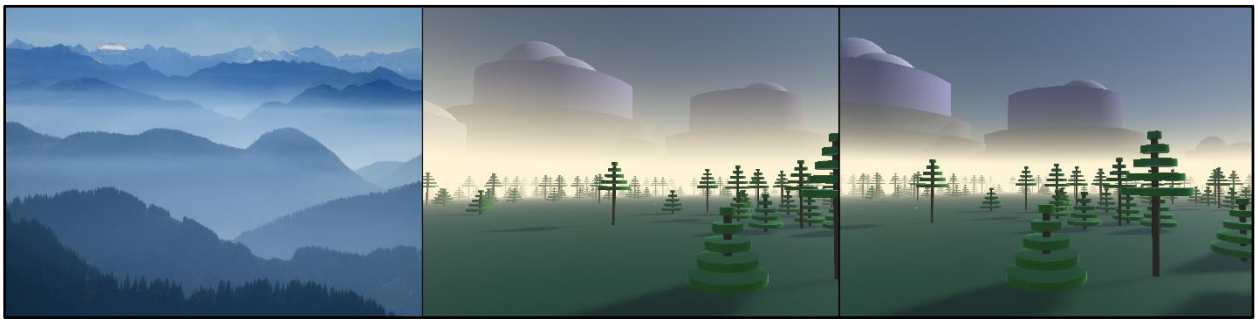


Figure 19 Height fog comparison between natural fog (left) and generated fog with linear (center) and exponential (right) falloffs

Figure 19 is a comparison between the natural fog shown in Figure 10 and two sample images generated within our system. This comparison highlights the height-based fog quality that was discussed previously. In both generated images, the fog density is constant up to the same height. At this point, the fog in both samples starts to thin out as altitude increases, with the center image thinning linearly over a requested distance and the right image thinning out exponentially. In both instances, it is observed that the bottom of all the objects are obscured normally by the fog. Furthermore, the distant mountains in both images extend above the height at which the fog starts to get thinner, becoming more visible as altitude increases. This quality mirrors that of the natural fog as seen in the image on the left.

5.2.4 Edge-Based Fog Density

Edge-based fog density is the effect that determines the rate at which fog thins at the outer boundaries of the fog volume. Real fog naturally thins at its outer bounds based on environmental factors.

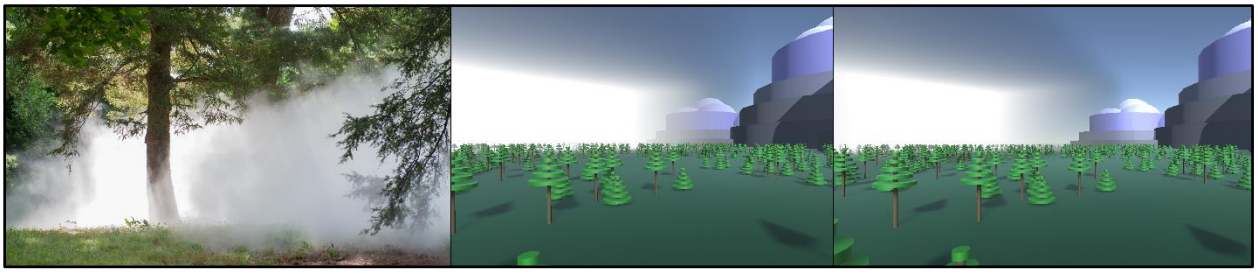


Figure 20 Edge fog comparison between natural fog (left) and generated fog with linear (center) and exponential (right) falloffs

Figure 20 is a comparison between the natural fog shown in Figure 11 and two sample images generated within our system. The comparison highlights the edge-based fog quality that was discussed above. In the natural fog image on the left, there is a perceptible thinning that occurs at the boundary of the fog. To mirror this effect, our system can generate a linear density falloff (center image) or an exponential density falloff (right image). The difference between the two methods is subtle, with the exponential falloff more closely matching the real fog quality in the sample image.

5.2.5 Color

Real-world fog absorbs, refracts, and reflects the local and global illumination it is subjected to. The final color of fog and the objects it obscures is based on this complex interaction with the lighting.

Figure 21 Colored fog comparison between natural fog (top left) and generated fog under different direct lighting conditions

Figure 21 shows a comparison between the natural fog shown in Figure 12 and a sample image generated using our system. This image on the right shows that the fog generated in our system determines its color based on the interaction with the both the local and global illumination model similar to how the color of fog is determined in the real world.

5.2.6 Shadows

In real world fog, crepuscular lighting is the result of high-intensity light such as from the sun shining on objects in a participating medium, such as fog. As a result, an observer can witness rays of light or shadow (or both).

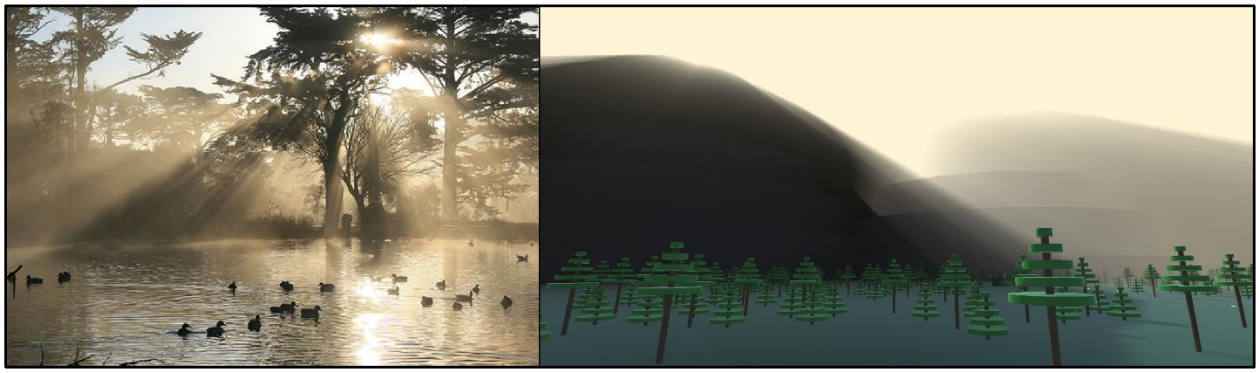


Figure 22 Shadow fog comparison between natural fog (left) and generated fog (right)

Figure 22 is a comparison between the natural fog shown in Figure 13 and a sample generated within our system. This comparison highlights the ability of our system to partially replicate crepuscular lighting like what can be observed in the real world in highly dynamic lighting conditions. This effect can be improved in our system by implementing some scattering phase functions that approximate atmospheric effects such as Raleigh or Mie scattering.

5.2.7 Ambient Fog

Ambient fog is not something that occurs naturally. This is an effect created for movies and video games to create a visible fog effect when there is not a good source of lighting.

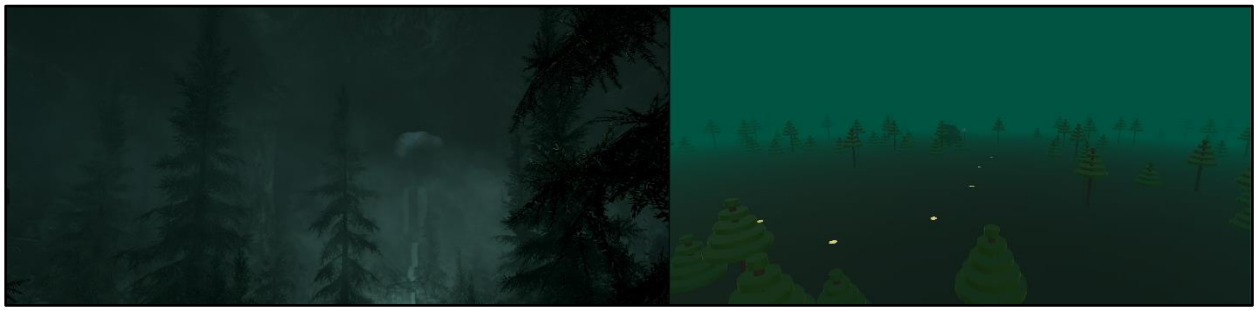


Figure 23 Ambient fog comparison between commercial product (left) and our generated fog (right)

Figure 23 provides a comparison between the ambient fog generated in a commercial game product, as shown in Figure 14, and a sample of ambient fog generated by our system. This comparison demonstrates the ability of our system to generate ambient (glowing) fog in a manner like what other products do.

5.2.8 Noisy Fog

In nature, it can often be observed that the thickness of the fog varies depending on which direction the observer is looking and the distance they are focused on. The consistency of the observed fog can vary greatly over short distances and can change over time.

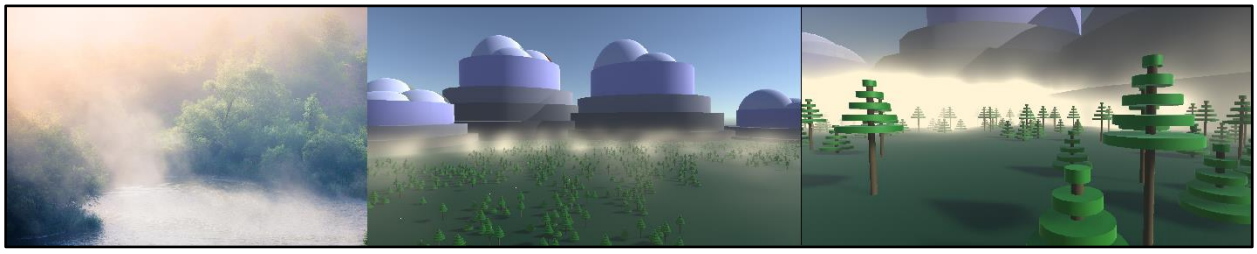


Figure 24 Noisy fog comparison between real fog (left) and generated fog (center, right)

Figure 24 shows a comparison between the natural fog shown in Figure 15 and two samples generated using our system with two different configurations. The natural fog image on the left shows an example of great variations in the fog density, with the right half of the image being slightly obscured by fog and the left half of the image being almost completely hidden from view.

The center image shows a simulated thin, patchy ground fog and the righthand image shows a thick fog bank. Both images highlight the ability of our system to produce local variations in the density of the fog. In the center image, this trait comes through in the form of a patchwork of wispy clouds drifting low over the ground. Some areas in the image are well covered by the fog, whereas other areas seem to be completely without fog. The righthand image also displays variations in the fog density. This condition is most noticeable around the base of the mountains,

as parts of the cylinders are heavily fogged and other areas are almost completely unobstructed from view.

5.2.9 Drifting Fog

Real world fog can move over time as a result of environmental factors, such as changes in temperature or wind. To simulate this effect, drifting fog has been implemented into our system. If the user has elected to have noisy fog such as that shown in Figure 24, then they can also input a velocity in world-space coordinates. This velocity will be applied to the simulated fog over time, causing the fog to drift in the requested direction.

5.2.10 Multiple Lights / Point Light

Real world fog takes its color in large part from the environmental lighting it is subjected to. Local variations in the lighting directly affect the color of the fog that is observed. To simulate this fog quality, our system can work with multiple lights to affect the color of the fog. As a proof of this concept, we have added a single point light source into our project.

Figure 25 Point light comparison between natural fog (left) and generated fog (right)

Figure 25 provides a comparison between individual point light sources in real world fog as seen in Figure 16 and the sample point light in a dark fog setting generated by our system. This comparison demonstrates that our fog system is capable of handling other types of lights beyond just the single directed light source. It also demonstrates that the visual quality of the point lights can be adjusted to provide a good approximation of its real-world equivalent.

5.2.11 Random Sampling Strategy

At times, it is possible to see artifacts from the raymarching process. One example of a visual artifact occurs when the fog volume is limited to a box area. When the fog is viewed through the box at an angle, if most of the pixels represent the skybox then it is probable that an observer will be able to see the edges of planes formed by multiple pixels with the same marching distance and step size. This effect is more pronounced when the system is set to use a smaller number of steps.

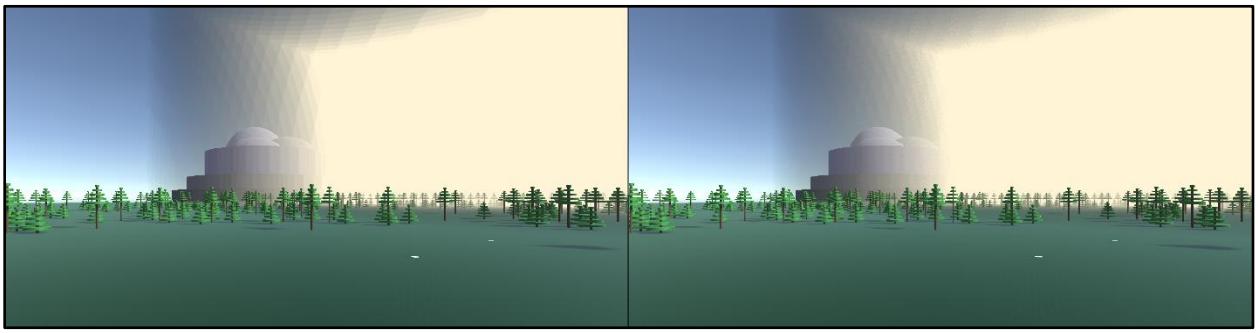


Figure 26 Edge artifacts (left) and edge artifacts with random evaluation points (right)

Figure 26 shows two images. The left image represents a boxed volume of fog with normal raymarching. In this image, it is easy to see the edges of the fog generated at each raymarching step since all the step sizes are the same for the pixels on the left side of the image. The image on the right uses a random offset for each evaluation point (+/- ½ step size offset), which softens this visual artifact.

Similarly, visual artifacts can be observed in large areas of shadow. This artifact is like the edge artifact described above, due to the simple scene we are working in.

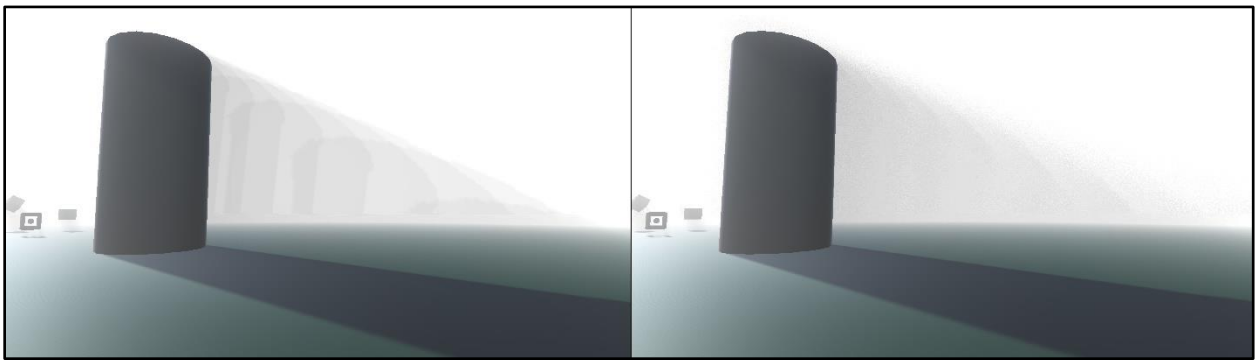


Figure 27 Shadow artifacts (left) and shadow artifacts with random evaluation points (right)

Figure 27 shows two images. The left image represents raymarching with shadows at 100% strength. This image shows planar artifacts like those shown in Figure 26. Similarly, the planar artifacts are significantly reduced or eliminated using randomized evaluation points. With random sampling points and shadows, a new visual artifact emerges due to some of the steps on both sides of the shadow / no shadow boundary now being sampled on the wrong side of the boundary line. This produces a spotted edge along the shadow boundary. Still, this artifact is preferable to the harsh planar artifacts seen in the left-hand image.

5.3 Quantitative / Performance Analysis

5.3.1 Quantitative Goals (Redux)

As mentioned in Section 3.1.2, our performance goal is to demonstrate a highly performant fog simulation at a full HD resolution of 1920 x 1080 for a game scene of moderate complexity. This minimal threshold is for the base scene running on the development system in the Unity game

preview window, with the frame count being calculated by the stats window provided in the Unity game engine. For a description of the development hardware, see Appendix A – Development System. For a description of the base scene, see Appendix B – Base Scene.

5.3.2 Trial 1: Base Scene, Minimal Feature Set

In this first trial, the goal was to make sure the system can achieve the minimal performance goal for a minimal set of features. The script configuration was as follows:

- Scene Wide Fog
- Number of Steps: 128 per pixel
- Saturation Distance: 500

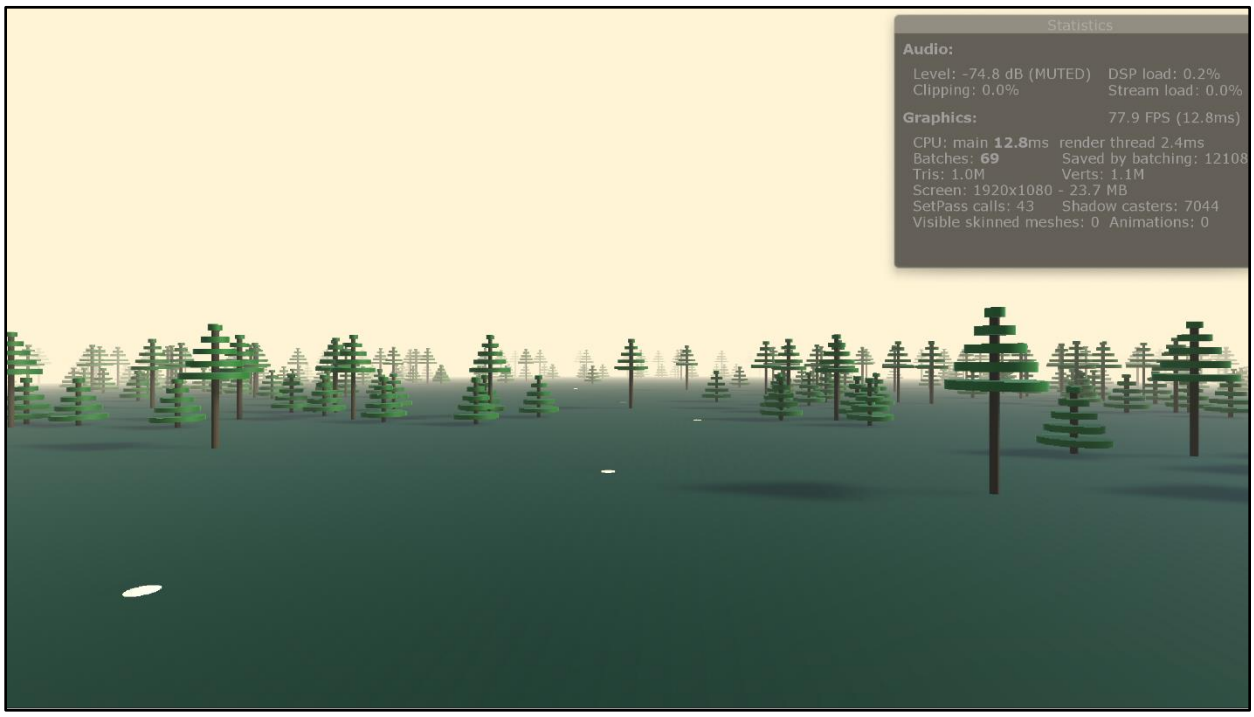


Figure 28 Trial #1: Base scene with minimal feature set enabled

With this configuration in place, the system was centered around 77 to 78 frames per second, with the minimum frame rate of 73.0 FPS and a maximum frame rate of 82.1 FPS observed. Figure 28 shows a screenshot from the trial run. The statistics window in the upper righthand corner of the image shows the screen size of 1920 by 1080 and a framerate of 77.9 FPS with 7044 of the game objects (shadow casters) that make up our base scene in the viewing frustum.

5.3.3 Trial 2: Base Scene, Medium Feature Set

In this second trial, the goal was to see how the performance of the system compares to the baseline run in Trial 1 using a typical or medium set of features. The script configuration was as follows:

- Scene Wide Fog
- Number of Steps: 128 per pixel
- Saturation Distance: 500
- Exponential Height Density: Starting height of 100, Falloff range of 200
- Shadow Strength: 0.50 (50%)

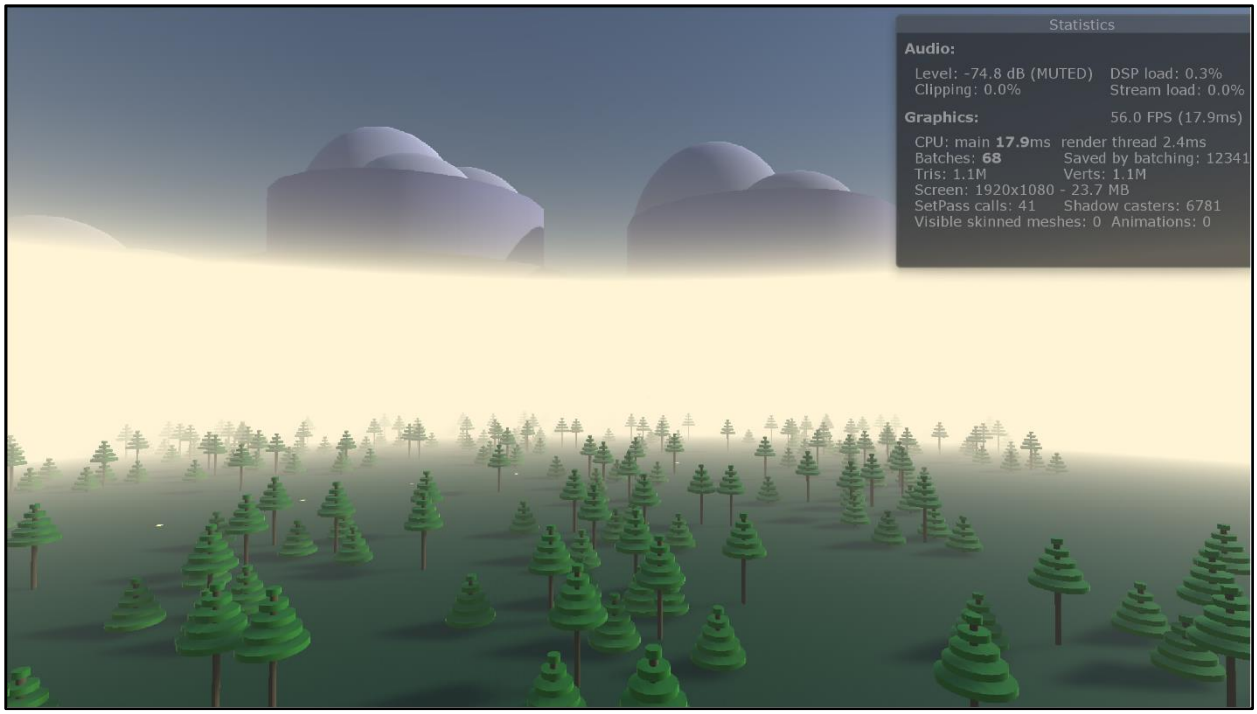


Figure 29 Trial #2: Base scene with a medium (typical) feature set enabled

Using the medium configuration, the system was centered around 56 to 58 frames per second, with the minimum frame rate of 54.0 FPS and the maximum frame rate of 63.0 FPS observed. Figure 29 shows a screenshot from the trial run. The statistics window in the upper righthand corner of the image shows the screen size of 1920 by 1080 and a framerate of 56.0 FPS with 6781 of the game objects (shadow casters) that make up our base scene in the viewing frustum.

5.3.4 Trial 3: Base Scene, Full Feature Set

In this third trial, the goal was to see how the performance of the system compares to the previous two trials using a full complement of features. The script configuration for this trial was as follows:

- Fog Box Scale: 1000, 300, 1000
- Number of Steps: 128 per pixel
- Saturation Distance: 350
- Exponential Height Density: Starting height of 100, Falloff range of 200
- Exponential Edge Density: X-direction: 35%, Z-direction: 35% (Percentage distance from center to edge of fog box).

- Shadow Strength: 0.50 (50%)
- Noise Strength: 1.0
- Noise Size: 100
- Noise Velocity: (1000, 0, 0)
- Point Light Range: 150
- Point Light Intensity: 1.0
- Point Light Color: RGB(0, 255, 0)
- Point Light Constant Attenuation: 0.1
- Point Light Linear Attenuation: 0.00005
- Point Light Exponential Attenuation: 0.0005

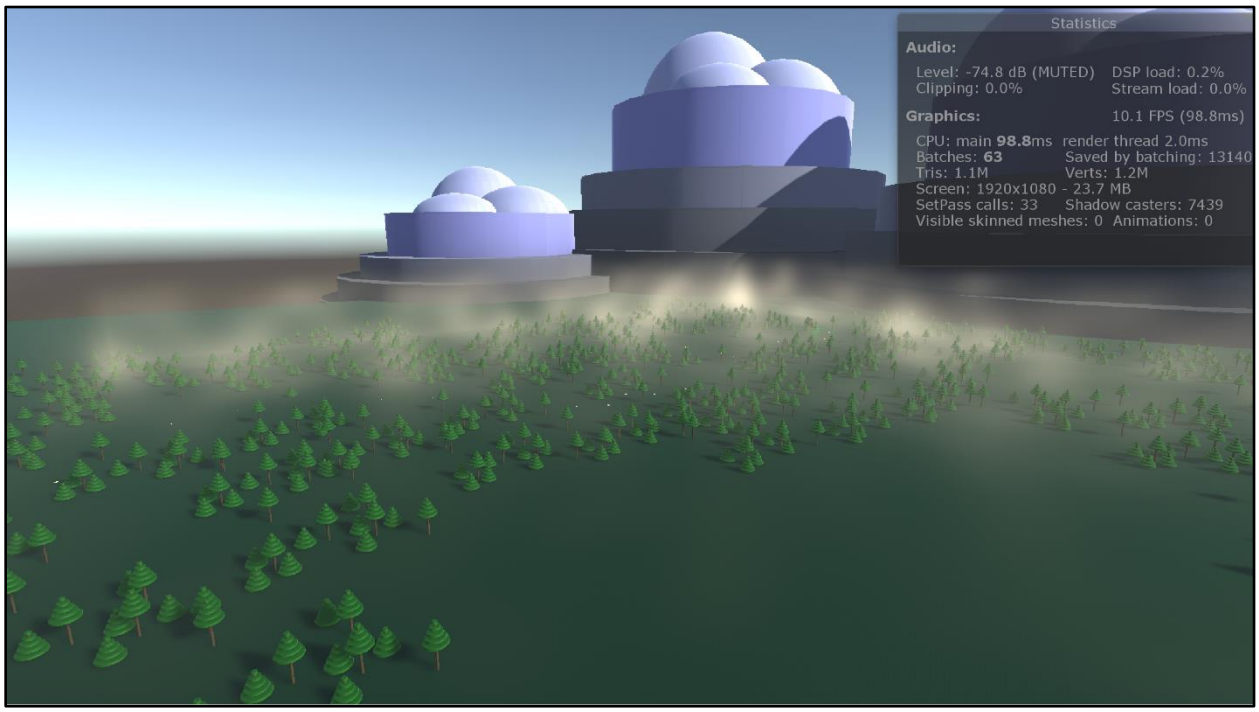


Figure 30 Trial #3: Base scene with a maximal feature set enabled

Using the maximal configuration, the system was centered around 10 frames per second, with the minimum frame rate of 9.7 FPS and the maximum frame rate of 12.2 FPS observed. Figure 30 shows a screenshot from the trial run. The statistics window in the upper righthand corner of the image shows the screen size of 1920 by 1080 and a framerate of 10.1 FPS with 7439 of the game objects (shadow casters) that make up our base scene in the viewing frustum. This is significantly below the performance of the first two trials.

5.3.5 Comparison of the Three Trials

The three trials presented in sections 5.3.2 to 5.3.4 were performed to demonstrate the capabilities of our system as it ran within the Unity game preview window on the development system. Just to recap, Table 1 gives an overall performance recap for the three runs:

Table 1 Comparison of the three trial runs, performed in the base scene with screen resolution of 1920 x 1080

Trial #	Description	Framerates Observed [FPS]			Features
		Minimum	Center Range	Maximum	
1	Minimal	73.0	77 – 78	82.1	Scene wide fog, 128 steps
2	Medium	54.0	56 – 58	63.0	Scene wide fog, 128 steps, Exponential height falloff, Shadows
3	Maximal	9.7	9.9 – 10.1	12.2	Boxed fog, 128 steps, Exponential height and edge falloffs, Shadows, Noise, Point light enabled

Table 1 shows the performance of our system for three different configurations of varying feature use. The first two trials have similar framerates, while the third trial shows significantly lower framerates. Most of this performance degradation is due to the use of the simplex noise generation function in creating the variable density fog.

We chose to use the simplex noise function over the Perlin noise function for three reasons. First, Perlin noise is a 2D noise function. Using it to create a continuous 3D noise texture would create visible repetition in the visible fog, which is undesirable. Second, Unity does not provide a Perlin noise function in its shader language. This means that we would have to generate a 3D texture from the Perlin noise function and then pass it into the shader, which consumes memory resources in the GPU. Lastly, simplex noise was created as a superior replacement to Perlin noise by Ken Perlin himself [36]. Simplex noise has better performance than simplex noise and supports the generation of 3D and 4D noise fields. And, we were able to find a library that already implements simplex noise in HLSL [37], making this function readily available for use in our compute shader.

Future work on our system should include research into optimizing the noise function used in the variable density calculation. While the use of the simplex noise function has led to a high visual quality for the variable density fog, the high cost of calling this function at every step evaluated within the fog volume makes it too expensive to use with our system in its current form in situations where framerate is a top concern. We have three suggestions for possible improvements to this system to improve performance. The first possible improvement is to look at options such as caching that allow the results of previous calls to the noise function to be stored and recalled later. A simple lookup system would be much faster than the current system of constant recalculation. A second possible improvement could come from developing a method that uses a reduced number of calls to the noise function. This could be as simple as having every other step share the noise value from the previous step or could be more involved depending on the impact to visual quality. The third possible improvement is to limit the distance of the variable density to some finite depth in the scene. Beyond this depth, the function could switch to a constant density fog, eliminating all the expensive function calls. This would have an appreciable

effect on the overall rendering performance but may have a negative impact on the visual quality of the fog depending on the desired visuals of the developer.

5.4 Chapter Summary

This chapter summarizes the results of the volumetric fog system, comparing the quality and performance of the system to the design goals presented in Chapter 3.

The next chapter concludes this report, summarizing the volumetric fog system that we implemented, discussing some of the limitations of the system and presenting some ideas about future improvements that can be made.

6 Conclusions

In this project, we implemented a system for adding volumetric fog effects to a game project. We accomplished this task using the cross-platform game engine, Unity. Our system currently provides several features for the developer, including the choice between using either a scene-wide or a boxed volume of fog, the ability to have the density fall off either linearly or exponentially as a function of height, support for using an edge-based density falloff when working with boxed fog volumes, shadowed fog, noisy fog, and others. This system is available for immediate use as a starting point for researchers and enthusiasts looking to add volumetric fog to their movies, games, or other projects [41]. For example, the Digital Future Lab (DFL), an interactive media production studio operating as a part of the University of Washington Bothell [42] is already exploring ways to use this system as a component in their upcoming game projects.

I have learned much in the two quarters spent devoted to this project. Through this project, I have gained valuable knowledge about how shaders and GPU's are used as a part of the graphics pipeline to display content to the user. Through my study of shaders, I have a greater understanding of the complex light and material interactions that get calculated, culminating in the final frame that the user sees every 60^{th} of a second. Furthermore, researching topics such as deferred rendering, compute shaders, and shadow maps has given me an understanding of how effects such as volumetric fog and other lighting techniques can be accomplished. Secondly, this project has reinforced my understanding of high-performance computing techniques, such as parallel programming using the single instruction multiple data (SIMD) paradigm. Lastly, this project helped me to put into practice many of the software engineering and management principles I studied during my time at the University of Washington

I have also gained an even greater appreciation for the teams of professionals that are able to create these amazing effects that are both visually stunning and highly performant. Effects like volumetric fog or shafts of light coming from an overhead skylight are not often important gameplay elements in and of themselves, but they are very important to the overall immersion a player experiences as they are tackling some game obstacle. The difficulty of creating these effects for today's gaming systems is often a balance between intense mathematical computation an approximation in order to achieve a compromise that gives the player the highest quality effect that can be had without degrading the performance of the gameplay.

Our system has several limitations, which means that several opportunities exist for improving the system going forward. Due to the time constraints for completing this project, here are three of the opportunities we have identified where quality and performance gains can be made. First, when shadow calculations are enabled, visible artifacts may occur at the boundaries between shadow cascades when multiple shadow maps are enabled. These artifacts appear to be due to polling for data at locations that lie outside of the shadow cascades.

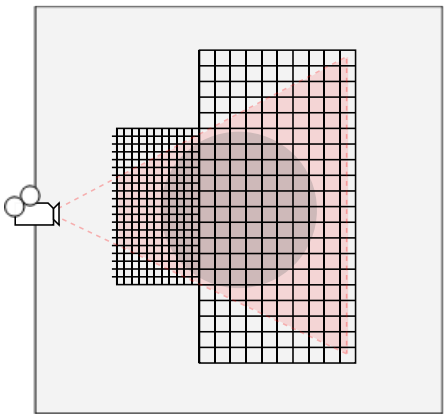


Figure 31 Unity manual: Shadow cascades in Unity [43]

Figure 31 shows how shadow cascades are configured in Unity. The cascades, up to 4 in all, all have the same resolution. That means that the cascade closest to the camera has smaller pixels than the cascades farther away, allowing it to depict shadows with greater detail than all other cascades.

Our solution to this problem is to only use a single shadow cascade to cover the entire viewing frustum. This does eliminate the artifacts at the boundaries, with the tradeoff being that this leads to noticeably lower quality of the shadows when objects are close to the camera due to perspective aliasing.

The second opportunity for improvement is to enable support for more light sources and types. Our system implements support for two light sources, the main directed light source and an optional second point light source. In the future, support could and should be extended for spot lights and for all scene light sources.

Lastly, this project supports multiple fog volumes by attaching more copies of the fog script to the camera. On the positive side, this allows each fog volume to be configured independently. The downside to this approach is that the order of the script execution is important and non-trivial. The most distant fog volumes must be executed first, with the results of each fog being added to the final frame before the next closer fog volume gets calculated. One possible fix for this is to add support within the script for an array of fog volumes instead of a single instance. This would allow for the system to perform the raymarching once for the entire scene, checking each evaluation point against all fog volumes in the array at each step. Another option would be to add a higher-level script to manage all the individual fog scripts.

Bibliography

[1] J. Sacranie, “Consumer Perceptions & Video Game Sales: A Meeting of the Minds,” *Illinois Wesleyan University Digital Commons*, 2010. [Online]. Available: http://digitalcommons.iwu.edu/econ_honproj/108.

[2] R. Verrier, “Visual-effects firms get a boost from video games - Los Angeles Times,” *Los Angeles Times*, 23-Jul-2013. [Online]. Available: http://www.latimes.com/entertainment/envelope/cotown/la-fi-ct-vfx-games-20130723-story.html. [Accessed: 10-Nov-2018].

[3] C. Wenzel, “Real-time Atmospheric Effects in Games Revisited,” *Presented at the Game Developers Conference, San Francisco (March 5-9, 2007)*. [Online]. Available: http://developer.download.nvidia.com/presentations/2007/D3DTutorial_Crytek.pdf.

[4] “GPU Gems - Chapter 39. Volume Rendering Techniques.” [Online]. Available: http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch39.html. [Accessed: 10-Nov-2018].

[5] D. Zorin, “Lighting,” *New York University*, 2005. [Online]. Available: https://mrl.nyu.edu/~dzorin/cg05/lecture08.pdf. [Accessed: 28-Oct-2018].

[6] Singh, Karan, “CSC418 Computer Graphics: Illumination,” Fall-2007. [Online]. Available: http://www.dgp.toronto.edu/~karan/courses/csc418/lectures/l15.pdf. [Accessed: 09-Nov-2018].

[7] Owen, G. Scott, “Illumination Models - Introduction,” *Illumination Models*. [Online]. Available: https://www.siggraph.org/education/materials/HyperGraph/illumin/illum1.htm. [Accessed: 10-Nov-2018].

[8] W. Jarosz, “Efficient Monte Carlo Methods for Light Transport in Scattering Media,” *Dartmouth University*, Sep-2008. [Online]. Available: https://cs.dartmouth.edu/~wjarosz/publications/dissertation/chapter4.pdf. [Accessed: 01-Jul-2018].

[9] Wronski, Bartlomiej, “Volumetric Fog: Unified Compute Shader Based Solution to Atmospheric Scattering (Presented at Siggraph 2014),” 2014. [Online]. Available: https://bartwronski.files.wordpress.com/2014/08/bwronski_volumetric_fog_siggraph2014.pdf.

[10] “Gamasutra: Bartlomiej Wronski’s Blog - Atmospheric scattering and ‘volumetric fog’ algorithm – part 1.” [Online]. Available: https://www.gamasutra.com/blogs/BartlomiejWronski/20141208/226295/Atmospheric_scattering_and_volumetric_fog_algorithm__part_1.php. [Accessed: 15-Jul-2018].

[11] Owens, Brent, “Forward Rendering vs. Deferred Rendering,” *Forward Rendering vs. Deferred Rendering*, 28-Oct-2013. [Online]. Available: https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342. [Accessed: 13-Jul-2018].

[12] Perez-Diaz, Jose L., Ivanov, Ognyan, Peshev, Zahary, and Alvarez-Valenzuela, Marco A., “Fogs: Physical Basis, Characteristic Properties, and Impacts on the Environment and Human Health,” *ResearchGate*, 20-Oct-2017. [Online]. Available: https://www.researchgate.net/publication/320531889_Fogs_Physical_Basis_Characteristic_Properties_and_Impacts_on_the_Environment_and_Human_Health. [Accessed: 11-Nov-2018].

[13] Croft, P J and University of Louisiana at Monroe, “Fog,” *Fog*, 2003. [Online]. Available: http://curry.eas.gatech.edu/Courses/6140/ency/Chapter8/Ency_Atmos/Fog.pdf. [Accessed: 10-Nov-2018].

[14] J. Mackay, “In Praise of Video Gaming’s Old Dalliance with Distance Fog,” *Waypoint*, 07-Feb-2017. [Online]. Available: https://waypoint.vice.com/en_us/article/mg9p3a/in-praise-of-video-gamings-old-dalliance-with-distance-fog. [Accessed: 29-Sep-2018].

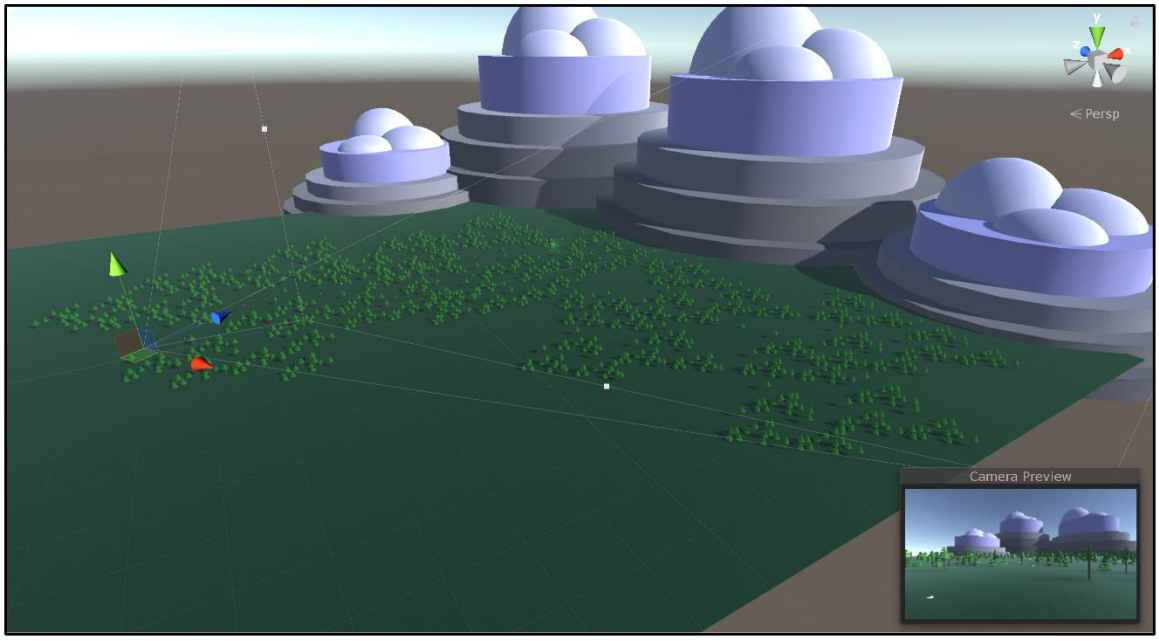
[15] Jasper Flick, “Rendering 14, Fog, a Unity Tutorial.” [Online]. Available: https://catlikecoding.com/unity/tutorials/rendering/part-14/. [Accessed: 30-Jun-2018].
[16] “Soft Particles - Wolfire Games Blog.” [Online]. Available: http://blog.wolfire.com/2010/04/Soft-Particles. [Accessed: 30-Jun-2018].
[17] “Mist System - Highly efficient flowing mist, fog or dust - Unity Forum.” [Online]. Available: https://forum.unity.com/threads/mist-system-highly-efficient-flowing-mist-fog-or-dust.269187/. [Accessed: 29-Sep-2018].
[18] A. Zanjiran, “[Tutorial] Volumetric Fog in Unity 2017 using New Particle Shader,” *YouTube*, 03-Jan-2018. [Online]. Available: https://www.youtube.com/watch?v=STugI38kD8c. [Accessed: 30-Jun-2018].
[19] SpeedTutor, “[Unity3D] Creating a mist / fog particle effect with shuriken,” *YouTube*, 26-Jun-2014. [Online]. Available: https://www.youtube.com/watch?v=lekE0Ez_go0. [Accessed: 30-Jun-2018].
[20] Gökhan Karadayı, “Unreal Engine 4 Volumetric Fog,” *Gökhan Karadayı - Environment Artist | World Creator*. [Online]. Available: https://gkan.artstation.com/projects/o1bRO. [Accessed: 12-Nov-2018].
[21] U. Technologies, “Unity - Manual: Deferred shading rendering path.” [Online]. Available: https://docs.unity3d.com/Manual/RenderTech-DeferredShading.html. [Accessed: 05-Jul-2018].
[22] “Deferred shading,” *Wikipedia*, 21-Aug-2018. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Deferred_shading&oldid=855824168. [Accessed: 30-Sep-2018].
[23] Mykhaylo Kotys, “Fast per-pixel lit volumetric fog without depth pre-pass,” 28-Apr-2015. [Online]. Available: https://www.gamasutra.com/blogs/MykhayloKotys/20150428/242150/Fast_perpixel_lit_volumetric_fog_without_depth_prepass.php. [Accessed: 11-Nov-2018].
[24] Carn, Dr. Simon A., “Fundamentals of Remote Sensing - Atmospheric Transmission,” *Michigan Technological University*, 31-Mar-2010. [Online]. Available: http://pages.mtu.edu/~scarn/teaching/GE4250/transmission_lecture.pdf. [Accessed: 11-Nov-2018].
[25] Prof. Glenn H. Chapman: SFU Eng. Science, “Lesson 7: Absorption & Scattering,” *ENSC 376: Introduction to Optical Engineering and Design*, 30-Jan-2008. [Online]. Available: http://www2.ensc.sfu.ca/~glennc/e376/e376l7.pdf. [Accessed: 01-Oct-2018].
[26] B. Kuchera, “1080p will come to Xbox One games, but at a cost - Polygon,” *Polygon*, 19-Aug-2014. [Online]. Available: https://www.polygon.com/2014/8/19/6045141/diablo-microsoft-resolution. [Accessed: 22-Sep-2018].
[27] “Unity,” *Unity*. [Online]. Available: https://unity3d.com. [Accessed: 29-Sep-2018].
[28] “What is Unreal Engine 4.” [Online]. Available: https://www.unrealengine.com/en-US/what-is-unreal-engine-4. [Accessed: 29-Sep-2018].
[29] NivCube, “Working Tractor in the Distance During the Fog,” *Storyblocks: Videoblocks*. [Online]. Available: https://www.videoblocks.com/video/working-tractor-in-the-distance-during-the-fog-rvdxkvbggivx9p4pv. [Accessed: 17-Nov-2018].
[30] G. Schollwöck, “Blaue Stunde ND (Forum für Naturfotografen),” 20-Oct-2005. [Online]. Available: https://naturfotografen-forum.de/o21898-Blaue%20Stunde%20ND. [Accessed: 17-Nov-2018].
[31] F. Nakaya, “Celebrating Our 20th Anniversary - The Emerald Necklace Conservancy,” *The Emerald Necklace Conservancy*. [Online]. Available: https://www.emeraldnecklace.org/20th/. [Accessed: 17-Nov-2018].
[32] M. Shainblum, “San Francisco Photography and Fog Timelapse,” *Symphony of Fog*, 11-May-2017. [Online]. Available: http://www.shainblumphoto.com/project/symphony-of-the-fog/. [Accessed: 17-Nov-2018].

[33] Astrobob, “Crepuscular rays – a tale of sunbeams diverging in the blue sky – Astro Bob,” *Crepuscular Rays*, 10-Jul-2011. [Online]. Available: https://astrobob.areavoices.com/2011/07/10/crepuscular-rays-a-tale-of-sunbeams-diverging-in-the-blue-sky/. [Accessed: 17-Nov-2018].
[34] Nexus Mods, “Darker Interior Ambient Fog,” *Nexus Mods :: Skyrim Special Edition*. [Online]. Available: https://www.nexusmods.com/skyrimspecialedition/mods/11197/. [Accessed: 17-Nov-2018].
[35] L. Voropai, “WLE 2015 – Cloudy and Foggy Nature – Wiki Loves Earth,” *Wiki Loves Earth*, 19-May-2015. [Online]. Available: http://wikilovesearth.org/wle-2015-cloudy-and-foggy-nature/. [Accessed: 17-Nov-2018].
[36] S. Gustavson, “Simplex noise demystified,” *Linkoping University, Sweden*, 22-Mar-2005. [Online]. Available: http://webstaff.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf. [Accessed: 27-Oct-2018].
[37] E. Lash and Gustavson, Stefan, “2D / 3D / 4D optimised Perlin Noise Cg/HLSL library (cginc) - Unity Forum,” *Unity Forum*, 18-Dec-2013. [Online]. Available: https://forum.unity.com/threads/2d-3d-4d-optimised-perlin-noise-cg-hlsl-library-cginc.218372/. [Accessed: 23-Sep-2018].
[38] S. Gustavson and Ashima, “Procedural Noise Shader Routines compatible with WebGL: ashima/webgl-noise,” *GitHub*, 16-Nov-2018. [Online]. Available: https://github.com/ashima/webgl-noise. [Accessed: 17-Nov-2018].
[39] M. Abeln, “Rome of the West,” *Dense Fog*, 04-Jan-2009. [Online]. Available: http://www.romeofthewest.com/2009/01/dense-fog.html. [Accessed: 17-Nov-2018].
[40] D. Collier, “Tutorial 20 - Point Light,” *OGLdev - Modern OpenGL Tutorials*, 2015. [Online]. Available: http://ogldev.atspace.co.uk/www/tutorial20/tutorial20.html. [Accessed: 18-Nov-2018].
[41] J. Lanman, “VolumetricFog,” *GitHub*, 17-Nov-2018. [Online]. Available: https://github.com/JoshuaLanman/VolumetricFog. [Accessed: 17-Nov-2018].
[42] “Digital Future Lab: an interactive media design studio (University of Washington Bothell),” *Digital Future Lab*. [Online]. Available: http://digitalfuturelab.com/. [Accessed: 17-Nov-2018].
[43] U. Technologies, “Unity - Manual: Directional light shadows.” [Online]. Available: https://docs.unity3d.com/Manual/DirLightShadows.html. [Accessed: 22-Jul-2018].

Appendix A – Development System

Computer	ASUS VivoBook Pro 15 N580VD
Operating System	Microsoft Windows 10 Home
Version	10.0.17763.134
CPU	7th Gen. Intel Core i7-7700HQ at 2.80GHz (6M Cache, up to 3.80 GHz)
RAM	32GB DDR4 at 2400MHz Memory
GPU	NVIDIA GeForce GTX 1050 with 4GB GDDR5
HDD	1 TB Solid State

Appendix B – Base Scene



The scene is composed of the following Unity 3D primitives (9087 total):

Ground (1 total):

Primitive: Plane

Trees (1501 total):

Primitives: Cylinders (6 per tree: 1 trunk, 5 branch layers – 9006 total)

Short Trees: 859

Tall Trees: 642

Mountains (4 total):

Primitives: Cylinders (4 per mountain – 16 total)

Spheres (3 per mountain – 12 total)

Shed:

Primitives: Cubes (12 total)

Lamp Post:

Primitives: Cylinders (2 total)

Sphere (1 total)

Point Light (1 total)

Cookie Trail:

Primitives: Spheres (1 each, 36 total)