Weather System Generation Using The Unity Game Engine

Andreea Elena Velea|170750077

BSc Computer Science with Game Engineering

Project Supervisor: Dr Graham Morgan

Abstract

This paper will describe how weather effects, such as volumetric clouds or snow, can be reproduced using computer graphics techniques and integrated into a managing weather framework. The nature of these procedures will be for the purposes of integration into video games.

The implementation process of a complete weather system will be presented along with computational and visual performance analysis highlighting the advantages and disadvantages of the final implementation.

Declaration

"I declare that this dissertation represents my own work except where otherwise stated."

Acknowledgements

I would like to thank the Game Engineering supervisors, Dr Graham Morgan, Dr Gary Ushaw and Dr Richard Davison, whose continuous guidance helped me overcome difficult obstacles and have been of great help when considering possible implementations.

I would also like to thank the entire staff at Newcastle University that I have had the pleasure of learning from during these past three years, without whom this paper would not have been possible.

I am immensely grateful for my family and friends who have provided me with continuous encouragement and useful feedback.

Lastly, I would like to show appreciation for all of the bright minds who have brought technology to where it is today.

Table of Contents

1 Introduction

1.1 Motivation

One crucial element of modern video games is the atmosphere, the feel of the world that the player is about to enter. Whether the sun is gleaming through a small forest clearing where a fabled sword is waiting to be plucked from the stone by a brave hero, or whether the rippling of the rain is mingling with the tunes of jazz in a New York detective's office, the weather can be a very powerful storytelling element. Having the right weather at the right moments can invoke high emotional intensity within the player and make for a more memorable experience. In open-world games, the weather is one of the many elements that immerse the player into an environment that feels real. Games that simulate car racing might benefit from altering physics based on weather, translating rain into more slippery roads.

In a piece of artistic production, where nothing happens without reason, there must be some symbolic value attached to the fact that "It was a dark and stormy night." This aspect naturally relates to creating the atmosphere. Floods and storms create a sombre-looking scene while sun and rainbows give way to more positive impressions.

Weather can also be used as a device to advance the story. Would Noah still build the ark if it were not for the flood? Most likely not.

In a more symbolic sense, a character that stands in the rain can be cleansed of past misdeeds or happenings, or contrarily, they can fall in the mud and get dirty.

Triple-A(AAA) games have showcased profoundly realistic and detailed weather, which is, of no small part, due to their budget and resources. A sterling example of how much weather can do to bring a game to life was displayed in Ghost of Tsushima by Sucker Punch. Aside from the stunning effects of fog, rain, and clouds, the weather element that stands out, in particular, is the wind. On the island of Tsushima, where the story is set, the wind blows in the direction of the quest. This innovative feature encourages the player to immerse themselves in the scenery, reducing the need to open a map and break the illusion of realism. Aside from particles, there are trees, grass, cloth, and ropes tuned to move appropriately under different wind conditions.

A set global wind path impacts most effects in the game; when a bomb goes off or fire is lit, the smoke drifts in the correct direction. Samples of the wind speed are also being taken to add turbulence as it increases. [1]

Adding petrichor-inducing rainfall is not an easy task for the average game developer however, as many elements go into a well-designed weather system. Anyone who dares to dream of adding this particular layer of realism into their games must first acquire learning of particle generation, shader coding, and texture rendering, among many others.

Capturing the essence of all the elements that make weather so intrinsically beautiful and translating them into code is a hurdle that stops many independent game developers from integrating weather into their games.

Learning resources specific to the problem of creating a weather system have been found to be very limited, and what material there is, was found to be outdated and difficult to put into practice.

Documenting the development of a weather manager in this thesis will hopefully shed some light on the process so that in the future, more game developers choose not to give up on weather effects.

1.2 Aim and objectives

The overall aim of this project is to study the development of an immersive weather simulation framework.

To achieve this, the following objectives must be followed:

- Identify and research existing key technologies and practices that best replicate weather and atmosphere in video games.
- Design and incrementally implement the determined individual weather elements.
- Design and implement a high-level, user-adjustable manager component.

Creating a fully-fledged weather simulator is both a technical and artistic challenge. Having poorly implemented components, such as unrealistically moving particles or badly rendered textures, would run the risk of ruining the player's immersion. Therefore, identifying and understanding the features that have successfully conveyed realism in previous weather simulators is a crucial step towards the completion of this project.

The weather phenomena will initially be implemented individually and incrementally added to the central system. Meaning, once a component works as intended by itself, it will be implemented into the weather system; when the current version of the weather system works well, the development of the following individual element begins. Tackling one feature at a time will allow for more efficient testing and bug resolution, as any issues that may arise during the initial stage will be resolved outside of the context of the collective framework.

Once the weather system's functionality has been fully completed, the issue of integrability and accessibility will be addressed. One of the main motivations of this project is to make weather integration available to developers regardless of their programming skills. Thus, the integration of the weather system into games will be done through a user interface.

1.3 Introduction summary

The introduction section provides a brief overview to the system and explains the causes that lead to the initiation of this project. The steps that must be taken in order to produce a satisfying result and the reasoning behind them are also presented.

The following chapter will provide a review of current academic work on subjects related to weather simulation in video games and will also provide insight into the technologies that were employed in the development process.

2 Background Research

2.1 Academic review

2.1.1 Player immersion in video games

Colloquially, immersion is recognised as the sense of being "in the game". In his thesis on the subject, Doctor of Psychology Aliimran Bin Nordin defines it as "a psychological absorption, where players invest their full attention, thoughts, and goals into the game as opposed to their surroundings" [2]. Indeed, video games are known to cause players to lose themselves in a foreign world, to the extent that they lose sight of what is happening in their surroundings. This feeling has importance because it plays a critical role in game enjoyment and is one of the effects of a positive gaming experience. [3]

2.1.2 Weather's role in storytelling

Weather can play an immense role in cognitively involving the player in the game, but it can also be used as a storytelling element. When weather appears in fiction, it is to accomplish certain narrative goals. Albeit being considered part of the world setting, its enigmatic and changeable nature permit it to influence characterisation, plot, and more [4].

In her article on the role weather plays in fiction, Kathryn Schulz discusses how the perception of weather shifts from "mythical to metaphorical", "with atmospheric conditions…stand[ing] in for the human condition" [5]. According to Schulz, such representations may refer to characters, interactions, or societies.

In *Long Day's Journey into Night,* a play by Eugene O'Neil, Mary Thorne, who struggles with morphine addiction, states that she adores the fog for its capacity to conceal the world. Fog is an allusion to the addicted state into which Mary takes refuge from unpleasant realities that she faces [6]. Through much of Richard Wright's novel *Native Son,* Bigger Thomas finds himself in a snow-covered Chicago, representing the isolation that he feels as a person of colour living amongst white society [7].

Symbolism aside, weather in fictional works helps authors create the right atmosphere. The way weather is characterised will have an impact on the elicited emotional response. At the start of *The Storyteller* by Saki, the narrator establishes that it was a "hot afternoon" and that "the railway carriage was correspondingly sultry" [8]. The reader already feels the irritating air even before learning that the main character will spend a train journey of an hour, trapped in the company of three rambunctious children and their aunt, a woman who is inept at making conversation.

Weather was shown to have an impact on human psychology in myriad subtle ways; the reason why this is not being immediately evident. The effects of weather on mood may be entirely physiological; excess heat causes discomfort, which causes irritability; sun exposure produces vitamin D, which promotes serotonin production improving disposition [9].

2.1.3 Weather in video games

Climate reproduction exists in various types of computer games, the degree of complexity varying with the size and budget of the company producing the game.

Weather is a crucial component within Rockstar's Red Dead Redemption game series. It adds depth and realism to the world environment while also creating atmosphere. Weather is deployed based on a time and location cycle; dust storms only happen in the arid deserts, while snow only falls in the mountainous regions. Temperature dictates the type of clothing that should be worn and can be accessed by the player via UI (User Interface). Dressing thinly in cold environments will lower the player's health, as wearing thick coats in hot climates will.

Smaller studios have also tried raising to the demand for a more realistic experience. Stardew Valley is an open-ended country-life RPG(role-playing game) developed by ConcernedApe featuring weather that changes daily and with the seasons. In Stardew Valley, weather plays a big role as a mechanic, meaning that the current weather has varying impacts upon gameplay. On a sunny day, the player must water their crops in other for them to grow, while on rainy days, there is no need to; during winter, crops do not grow at all.

2.2 Technology used

2.2.1 Particle systems

A particle system is a technique that creates and renders numerous small images or Meshes, called particles, to simulate a visual effect. Each particle in a system represents a distinct graphical element. The system mimics every particle collectively to produce the notion of the full effect. [10]

As defined by the researcher who first coined the term, William T. Reeves: "A particle system is a collection of many, many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system." [11]

Depicting dynamic objects like fire, liquids, or smoke would be tough without particle systems. These phenomena do not have well-defined surfaces, and their motions cannot be described by simple affine transformations, as they change shape fluidly and dynamically.

2.2.2 Shaders

Shaders are sets of code instructions to the Graphical Processing Unit that describe the attributes of either a vertex, pixel or primitive. [12]

A vertex shader can denote traits such as a vertex's position, colour or texture coordinates and is called for every vertex in a primitive, enabling control over the specifics of any scene involving 3D models.

Pixel shaders, also known as *fragment shaders*, work similarly, modifying attributes related to pixels like the RGB or alpha values. Fragment shaders are capable of applying shadows, specular highlighting, translucency and adjust the depth of the pixel. Since pixel shaders can be given information about surrounding fragments, they can also apply post-processing effects like edge detection, blur, or colour grading.

Geometry shaders can modify and generate entire primitives (point, lines, triangles) and are run after the vertex shaders and before the fragment shader. They allow for geometry tessellation - simple meshes, or hulls, are subdivided into finer ones which provides a better estimation of a curve. [13]

2.2.3 Volumetric rendering

Volumetric rendering emulates the propagation of light through a primitive's volume, allowing for intricate visual effects without storing the geometric surfaces of the primitive. An optical model is used to correlate the volumetric data with optical properties like colour and opacity. An image of the data is formed by sampling these properties along each viewing ray at render time. Essentially, the optical model describes how each individual volumetric element, or voxel, interacts with the light. The most simplistic model assumes that every particle in the volume simultaneously absorbs and emits light. In contrast, a more sophisticated model may incorporate volumetric shadows, ambient lighting and may also account for light scattering. [14]

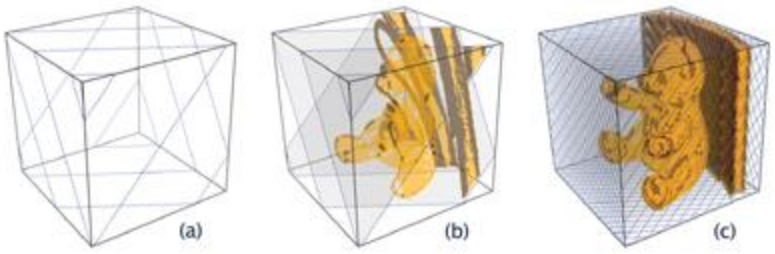


Figure 1: Volume Sampling and Composing [14]

2.2.4 Raytracing

Ray tracing is a rendering technique that simulates how light bounces off objects, producing realistic shadows and reflections. NVIDIA's Tom Peterson described the process as "a physics-based way of creating images" instead of "an artistic way of creating images" [15], referring to a previous technique of rendering lighting information called rasterisation.

The way vision works in the real-life is that light sources produce photons, or rays, that collide and bounce off surfaces until they reach our eyes. Raytracing completes this procedure in reverse, tracing individual rays from the scene's camera to the light source while sampling useful information life ambient occlusion. This results in better lighting effects while saving processing power by ignoring objects that the observer does not see. Practically, this is done by having an intersection function, that given a primitive surface and a view ray, returns the exact location where the ray hits the texture. The ray can then be tested against all relevant primitives and infer the closest intersection. [16]

2.2.5 Raymarching

Raymarching is a surface rendering technique useful for primitives that do not have a conclusive shape. Unlike Raytracing, since the target surface is not uniform, there is no simple intersection function that can be used. To get around this, the view ray marches one step at a time. The distance from the origin point (P0 below) of the ray to the nearest surface is first evaluated, which gives a safe radius where the point may be advanced without it going through anything. This process is repeated until the distance becomes small enough that collision may be deduced (P4 below). Having obtained the surface depth information, ambient occlusion samples can be taken to light and colour the pixel.

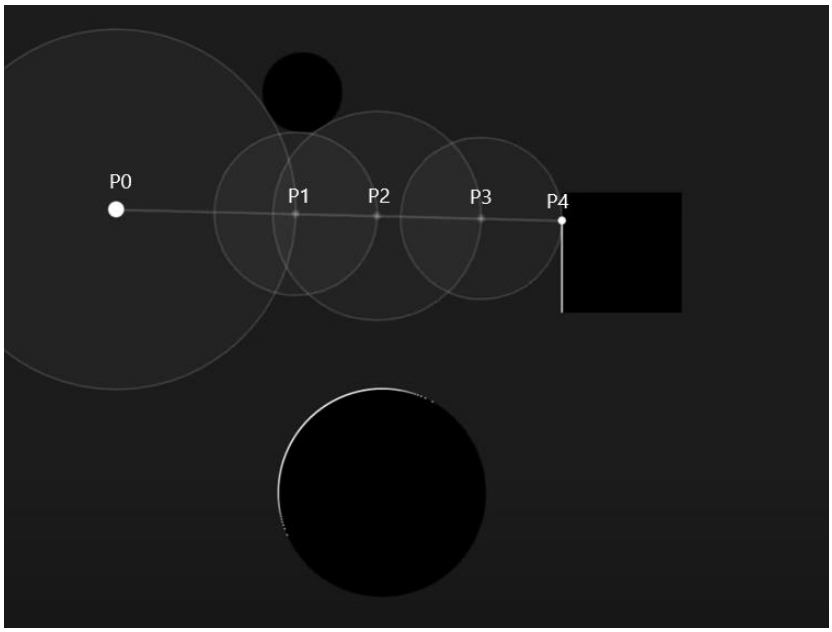


Figure 2: Raymarching trough Sphere Tracing [23]

2.2.6 The Unity engine

Unity is one of the leading game-production software in terms of versatility and usage, having been installed over 33 billion times in the past year alone [17].

The engine allows developers to deploy games to over 25 platforms, from Android and PS4 (PlayStation 4) to the lesser-known Windows Mixed Reality or Android TV. Game builds can be iterated from a lead format and ported to any of these platforms, making the creation of multiplatform games effortless.

Another advantage to using Unity is that it allows for a swift scene iteration, enabling the user to modify, preview and test their project efficiently.

Unity features a built-in particle system component allowing the user to simulate dynamic effects without the need to write any code. For convenience, the particle systems have a high range of customisability. All the properties are organised into collapsible sections called *Modules,* whose functionality can easily be enabled or disabled. For example, suppose the user wishes to emulate particles that are emitted by the movement of another object, like steps in the snow or dirt from a car's wheels. In that case, they may wish to set the particle system's emission to the *Rate over Distance* mode, and a set number of particles will be created per unit of distance travelled by the parent object [18].

2.3 Background research summary

Background research revealed some of the core functionalities that the system must have to preserve the emotional effect weather has on us. It has presented the inner workings of some recent digital graphics industry technologies such as volumetric rendering, Raytracing, and raymarching. The Unity Game Engine which was used for the development of the weather system presented in this paper was also discussed.

The ensuing chapter will follow the development process of the weather system and its components.

3 Implementation

This chapter will follow the design process and development of the weather system. The methods, technologies, and algorithms used will be discussed along with their strong and weak points. The structuring of this section will be mirroring the implementation process.

The first step in designing the weather system was to decide on an initial set of features. Several case studies were undertaken on various video games to determine the effects that would need to be depicted to create a convincing illusion of weather. Some obvious candidates were first selected: c, clouds, snow, fog, and lighting (ambient and volumetric).

Implementation began with the design and creation of a master component that would control the behaviour of the individual effects. This initial framework had no practical functionality outside of itself as this step was done previous of implementing any actual weather phenomena.

The weather system works by generating a set of boxed effects that are following the player. Therefore, the player camera must be assigned when first setting up the weather system in a scene.

3.1 The Sky

Skyboxes are a special type of six-sided material used to represent the sky. Before expecting any of the weather effects to look natural, a reliable way to freely modify the sky background had to be implemented. For this purpose, a skybox shader was created. The skybox is generated from six distinct textures, with each of them representing a view of the sky on a particular world axis. High dynamic range textures complete with a realistically placed Sun and Moon were made and used.

The shader allows for the textures, and consequentially the sky, to be tinted at will on a gradient scale. Tonal values are also customisable, enabling the sky to become as bright or as dark as needed. Lastly, the skybox can be rotated to allow for the implementation of a realistic day-night cycle.

3.2 Day-Night Cycle

Having made a fully customisable sky, the production of a day-night cycle became a straightforward task. Directional lights of appropriate intensities were added to the Sun and Moon on the skybox to add the appearance that shadows are cast from them. A script was written to control the behaviour of the skybox over time. The sky is rotated around the positive y-axis to give the illusion that the sun and moon are climbing in and out of the horizon. The exposure and tint of the sky are gradually changed in accordance with the time of day. The user may set the real-time duration of an in-game day, how dark or bright the night should be, and also modify the sky's colour.

3.3 Rain

During real rainfall, larger droplets appear as elongated cylindrical shapes, while smaller droplets are swept in spurts by the wind, creating a drizzle. This becomes more apparent the heavier the rainfall. They also tend to create ripples or splashes when they hit a surface. All these aspects were replicated by using multiple particle systems.

Firstly, a box-shaped particle system was created. The particles were made to fall downwards by adding a varying negative Y value to the *Velocity over Lifetime* module. This can also be done by enabling *Gravity to* affect the particle system.

While this approach might have been useful for physics-based games, to make the raindrops fall at the expected speed, the particle system would have had to be set up higher on the y axis, which in turn would call for a longer particle lifetime. As this would impact performance, the first method was chosen for implementation.

To replicate raindrops, a texture of a cylindrical singular shape was created and applied to the particle system. It was rendered in the stretched billboard mode, meaning that the particles will be pointing in the direction of movement. This will make the rain appear to be moving with the wind, a crucial aspect of the overall illusion.

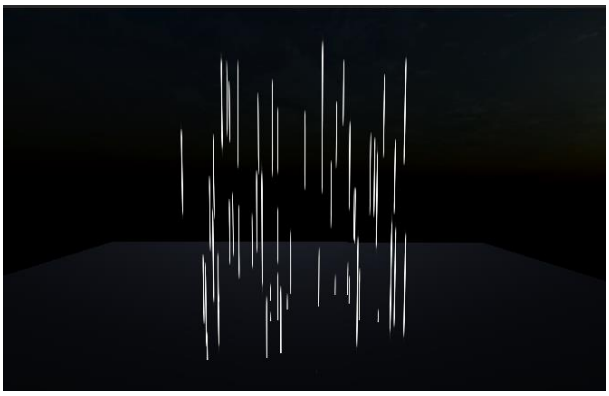


Figure 3: Rain particle system movement pattern

The 3D Start Size, Opacity, and Emission Rate modifiers were altered accordingly to further mimic the appearance of rain. The 3D Start Size variable, as the name might imply, controls the three-dimensional size of the individual particles. As real raindrops have slightly different sizes, this variable was set to output varying-sized particles. The *Opacity* modifier, which regulates the transparency of the particles, was lowered. The *Emission Rate* is the rate at which new particles are created; regarding the rain effect, it affects how heavy the rain appears to be.

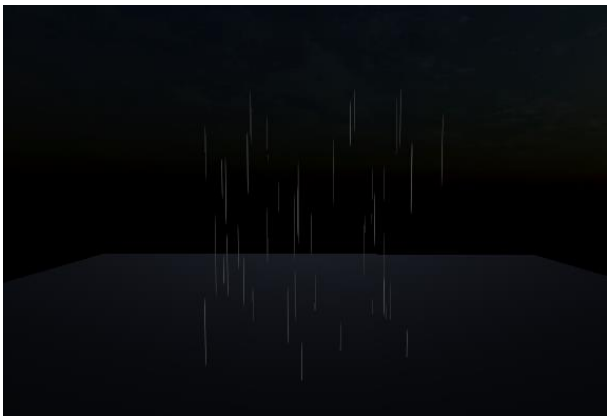


Figure 4: Rain particle system after Opacity, Size and Emission rate alterations

The steps undertaken so far had created a rudimentary rain effect. To further enhance realism, a second particle system was added to replicate the drizzle.

The central differing aspect from the first particle system lies in the texture of the particles. A texture of multiple cylindrical shapes was applied to replicate groups of smaller, lighter drops that are more easily swept by the wind.

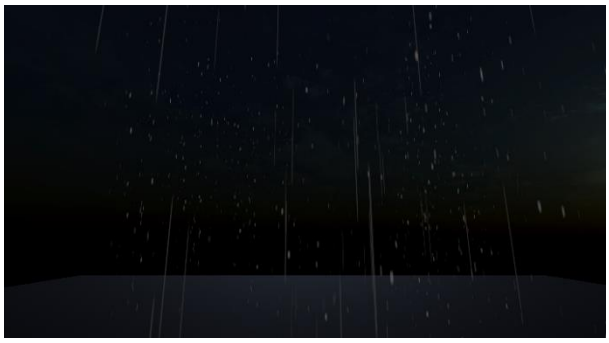


Figure 5: Rain effect with drizzle

A third particle system was added to replicate the droplets splashing when hitting a surface. For a fuller, more organic rain, an animated splash texture was created and applied. This time, the texture was rendered in the vertical billboard mode, and the *Start Speed* variable was set to zero. This means that the particles will be stretching upwards from the contact surface but will not travel.

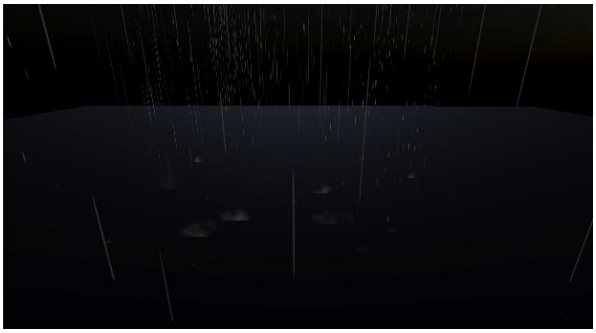


Figure 6: Rain effect with splash particle system

When it rains in warmer climates, the higher temperature causes the raindrops to begin evaporating and remain suspended in the atmosphere in the form of mist [19]. A fourth particle system was made to imitate this phenomenon.

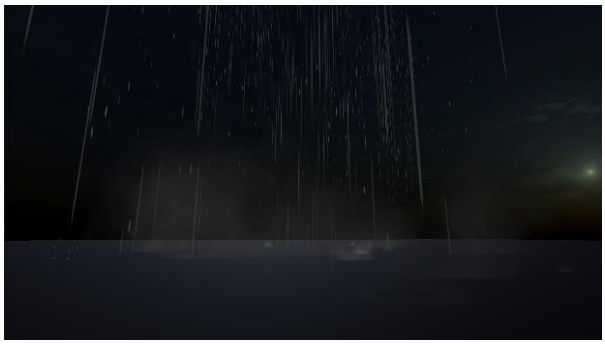


Figure 7: Rain effect finalised

3.4 Lightning

Thunderstorms are a splendid way of using nature to indicate that a point of high emotional intensity is about to be reached in the narrative course. Battling a powerful antagonist while overlooked by a dark, stormy, and electric sky is sure to instil adrenaline in most players. For that to be possible, the weather system had to be able to produce lightning effects which was done, once again, through particle systems.

A new spherical particle system of a minimum radius was created. Just as lightning would, the particles were set to last for one second and swiftly travel downwards. This movement pattern was achieved by setting the *Lifetime* modifier to one and the *Velocity over Lifetime* variable to a negative value.

For the particle system to resemble lightning, the actual particles would not be rendered but only their trails.

Figure 8: Lightning particle system movement pattern

Figure 9 : Lightning particle system after the enabling of the Trails module

Once the noise property of the trail was modified, the streak started resembling lightning. To enable the effect to move, *Scroll speed* and *Frequency* were set between two constants for a bit of randomisation. To make the particles visually behave as lightning, W*orld collision* and *Bounce* were enabled.

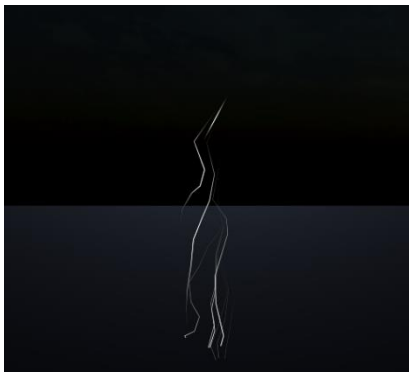


Figure 10: Lightning particle system after Noise, Scroll speed and Frequency alterations

Figure 11: Lightning particle system after enabling World collision and Bounce

A secondary emitter was added to collision points so that a smaller burst of electricity would be produced. Finally, point lights were added to make the lightning illuminate the world around them for a more dynamic look.

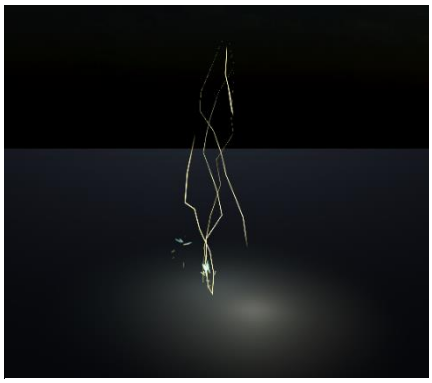


Figure 12: Lightning effect finalised

3.5 Snow

The snow effect was created in a similar manner with the rain effect; the central particle system behaved almost identically, with a few key differences that will now be discussed.

The velocity over time x and z were set to be varied to give the appearance that the particles are floating around before falling. The *World Collison* variable was enabled, *Colour Over Lifetime* was altered appropriately, and the *Bounce* module was set to zero, meaning that when the particles hit a plane, they will slowly fade away and their movement will stop.

An editor tool with intuitively named variables was created and incorporated into the master system.

3.6 Clouds

Implementation began with research on different cloud types and their corresponding atmospheric conditions to ensure a loyal image of clouds would be pursued. Clouds tend to be denser the colder the temperature, and since temperature decreases with higher altitudes, the tops of clouds tend to look rounded and full, while the lower regions look wispy and diffused. The direction of the wind also varies over altitude, which distorts the clouds. When reaching a certain level of density, clouds precipitate as snow or rain [20]. These notions were instrumental when it came to model the clouds.

Clouds have been previously depicted in video games by hand placing photographs of real clouds in a dome shape around the game scene resulting in a heavily art-directed sky [21]. Commonly known as *flat* or *billboard clouds,* these 2D solutions can be further enhanced by adding cloud animations. While being easy to implement and not demanding on the hardware, this technique presents some key disadvantages. Firstly, that the illusion of the clouds can be quite easily broken when examined closely, which may be fixed by storing several images from multiple angles and displaying them as the player moves. Doing this, however, leads to a high amount of memory being used which renders the advantage of good performance obsolete. Therefore, while this option may not be the most realistic, it is useful for games produced for low-end platforms like mobile.

The weather system described in this paper makes clouds available through two implementation methods to accommodate a higher range of hardware capabilities. The user may pick the one that best suits their hardware and visual expectations.

3.6.1 Particle Clouds

Implementing clouds through particle systems was a somewhat effortless process that yielded a convincing image of the source phenomenon with very little impact on performance.

The effect comprises two box-shaped systems with particles of differing textures, each contributing to the illusion of clouds in specific ways. One of them uses a texture imitating the thinner, lower portions, while the other uses ones that denote the denser regions. Both systems' particles were scaled up to an appropriate proportion, and their start speeds were adjusted and made dependent on each other. To emulate the feeling that the clouds are travelling over the scene, the Colour over Lifetime variable was modified to have the newly emitted particles start with a zero alpha value and slowly fade into the background, while the perishing particles vanish from full alpha to zero.

In the weather manager component, the particle system traits were included as sliders using intuitive names like *Cloud Movement Speed* or *Cloud Density* so the user may easily customise them as they see fit.

3.6.2 Volumetric Clouds

Volumetric Clouds were added to the weather framework by following the same basic techniques previously described by Andrew Schneider, the Principal FX Artist at Guerrilla Games [21].

The volumetric density of the clouds was sampled with the use of raymarching. As previously noted, ray marching involves *marching* the rays from the camera and sampling noises and sets of gradients along the way to define the density of the clouds.

3.6.2.1 *Worley noise modelling*

To model the shapes of the clouds, inverted Worley and Pelin 3D noise were layered. Worley noise replicates the rounded peaks, and Pelin noise renders the wispy portions. To create the Worley noise texture, random points were scattered inside a square where each pixel stores the distance to the nearest point. The pixel's brightness scales with the distance, black representing a distance of zero, whereas white represents the farthest distance which was normalised to be one. Colour inverting the image resulted in a texture that could imitate the billowy shapes of clouds.

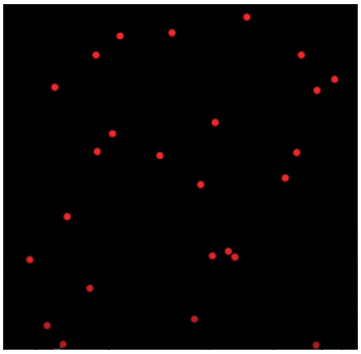


Figure 13: Worley Noise Generation: Scattered Points

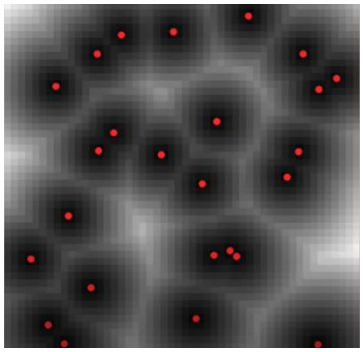


Figure 14: Worley Noise Generation: Pixel Brightness Related To Closest Point Distance

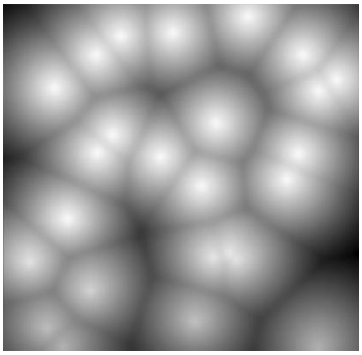


Figure 15: Inverted Worley Noise

For the texture to seamlessly repeat itself, the square was divided into smaller cells, each containing a single random point. The closest point to a particular pixel is guaranteed to be either inside its cell or in one of the adjacent ones; once the pixel's cell is found, the distance to the closest point can also be calculated.

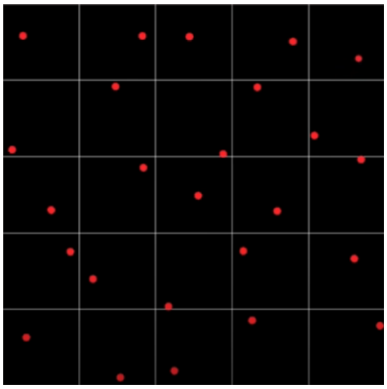


Figure 16: Worley Noise Generation: Divided Cells

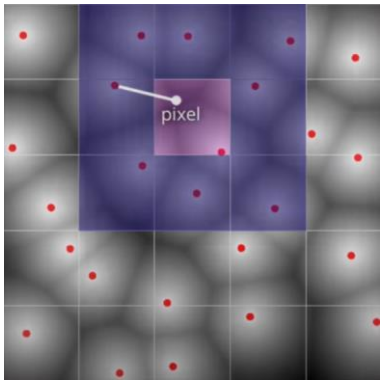


Figure 17: Worley Noise Generation: Calculating The Closest Point Distance

The random points were generated through a C# script, where a compute buffer is produced out of them and sent to a shader where the value for each pixel can be calculated in parallel on the GPU.

A noise generator component was created to enable easy editing of the textures. The tool makes it possible to specify the number of random points and overlay different frequencies of noise to add variation. Since this is a 3D texture, a slider was added to choose which slice to preview. Noise can be stored in the red, green, blue, and alpha channels of the texture, enabling complete customisation of the shape; a secondary texture of lower resolution (Pelin noise) was added to improve detailing.

3.6.2.2 *Mapping the scene through Raytracing*

Now that the clouds could be modelled, a way was needed to draw them into the scene. For this, a raytracing shader was written. First, a box object was created to act as a container for the clouds in 3D space. The distance to both sides of the box along a certain direction will be needed so that, while going through each pixel in the fragment shader, the possible location of clouds might be found. The bounds of the container were passed to the shader along with the noise textures.

Inside the shader, the current camera position and view direction were sampled and used to find out whether the view ray hits the container or not. If it does, the box can be drawn into the scene at an accurate position. However, since the shader has no information about the surface contents of the scene, the box would completely obscure everything underneath. Unity stores the distance from the geometry at each pixel to the camera in a texture from which the depth value can be inferred. Therefore, to account for the scene map, the view ray will only hit the container if the distance to it is less than the depth value.

The real goal, however, was to draw the contents of the box. To do this, a function was added to compute the density of a cloud at a given point. This point was transformed by a scale and offset parameter and used to sample from the noise texture. This value minus a threshold is returned in the red channel, making it so that anything below that threshold is empty space, while any larger value is inside the cloud.

```
float sampleDensity(float3 position) {
    float3 uvw = position * CloudScale * 0.001 + CloudOffset * 0.01;
    float4 shape = ShapeNoise.SampleLevel(samplerShapeNoise, uvw, 0);
    float density = max(0, shape.r - DensityThreshold) * DensityMultiplier;
    return density;
}
```

Figure 18: Code Snippet Responsible For Density Sampling

3.6.2.3 *Raymarching rough cloud shapes*

Next, the density of the clouds along the view ray was estimated through raymarching. Back in the fragment function, a loop was added that *marches* the view ray a small distance at a time inside the container and adds up the density values to return the estimated total density. The $e^{-total\ density}$ equation was used to get the final transmittance value.

The equation describes how the more volume the light has to travel through, the smaller the proportion of light that reaches the observer will be (because some of it will bounce off in other directions).

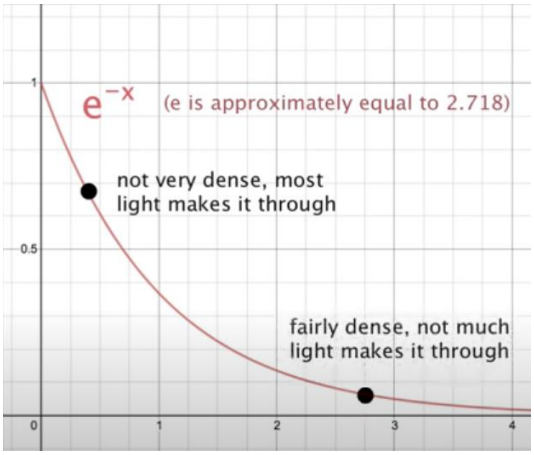


Figure 19: Transmittance Equation [23]

3.6.2.4 *Rendering Light Scattering*

The sky changes its colour at sunset (red near the sun, blue farthest away) through Rayleigh scattering by atmospheric gas particles, which are much smaller than the wavelengths of visible light. The grey/white appearance of the clouds is caused by Mie scattering by water droplets, which are of a similar size to the wavelengths of visible light, allowing them to scatter completely [22].

As you can imagine, this is a very complex process to replicate, so for the purposes of this project, a crude simplification was used.

Firstly, the amount of light passing through the cloud needs to be found out. Imagining a ray travelling from the observer through the cloud, we start by taking a point inside the cloud. To estimate how much light reaches this point, another ray is cast from this point towards the light source, taking a couple of density samples to discover how much volume the light must travel through. As some of the light will be scattered inside the cloud, the above equation will be computed both on the ray travelling towards the light and on the ray portion from the point to the observer.

This process will be repeated for multiple points along the initial ray before the lighting information of one pixel is obtained. The Graphical Processing Unit (GPU) will repeat the entire process until every pixel that the cloud covers is rendered.

3.6.2.5 *The cloud editor*

Once the implementation was completed, the resulted cloud editor component could be incorporated into the main weather framework. The noise texture editor created earlier was also included as a collapsible module for convenience; the user can preview in real-time how modifying the noise variables affects the actual shape of the cloud. Adjusting the density, brightness, and scale or tweaking speed and forward scattering has been made as accessible as possible.

Figure 20: Editing the volumetric clouds

3.7 Ambient Lighting

The scene is lighting is done through the skybox. Since directional lighting was set up to shine from the current position of the Sun or Moon, shadows are being cast in a natural manner that follows the time of day. Through the editing of the skybox component, ambient lighting can also be easily adjusted to match different weather scenarios.

3.8 Volumetric Lighting

Lighting is a critical factor for making visually appealing scenes. Colloquially known as *God rays* or *light shafts*, volumetric lights are an incredibly versatile effect that will enchant any scenery it is added to. From a technical standpoint, the effect of volumetric lighting is created by allowing light sources to affect volumetric fog.

Since Unity makes volumetric fog thicker at lower altitudes and thinner at higher ones, simply enabling the volumetric light and fog variables causes any scenes on lower planes to appear enveloped in dense fog. *Base Height* and *Mean Height* variables were tweaked to ensure the effect looked natural on all ground levels. The *Base Height* variable controls where the fog starts, while *Mean Height* modifies the dispersion of the fog.

3.9 User interaction design

One of the primary motivations for this project is to bring the weather system to any game developer that may wish to use it, regardless of their programming skills. Therefore, the system must be as effortless to apply and use as possible. This section will outline the design choices that were taken in pursuit of that.

3.9.1 Profiles

Given the complexity of this weather system, there is a very high number of modules that must be modified in order to achieve specific weather conditions. With the users' ease of access in mind, ten premade weather profiles were included. When a profile is applied, all the predetermined necessary changes are made, greatly simplifying the process. The user can further tweak these to obtain their exact desired look. The users may also create and save their own weather profiles for later use.

3.9.2 Seasons

Seasons are a series of weather profiles that change over a period of time that may be specified. By default, there are four seasons to simulate their real-life counterparts. Still, the user may add or subtract the number and fully alter the weather's behaviour during a season—a tool for swapping terrain and object textures when the season changes was integrated for maximum convenience.

To exemplify, a default Spring season lasts 30 in-game days. It will start with a high probability to display quick spurts of heavy rainfall that gradually lessens to allow for a smooth transition into Summer.

3.9.3 Weather areas

Working similarly to Unity's wind zones, weather areas allow the user to create climates or weather-specific biomes by setting up location coordinates for specific weather types. When the player nears a passing border, the phenomena particular to the current weather zone will start a slow, smooth transition into the ones of the nearing biome.

3.10 Implementation summary

The Implementation chapter has followed the development process of each component of the weather system and provided an outline of its structure. It has shown how particle systems can be used to create a plethora of effects such as rain, snow, lightning, and clouds. The chapter details how rendering techniques like raytracing and raymarching work and how they were applied in generating volumetric clouds. Several design choices were discussed, such as the inclusion of the weather profiles, seasons, and weather zones.

In the following chapter, we will look at how well the completed weather system performs both visually and in terms of framerate.

4 Results and Evaluation

This chapter will showcase the outcomes of this project from both a visual and computational performance standpoint. Although it can be fairly challenging to strike the right balance between aesthetics and performance, this section will analyse the trade-offs that can be made towards that goal.

4.1 Visual results

This section will highlight the capabilities of the weather system implementation presented in this paper by visualising the results of some settings used to create various scenarios. All the effects were applied to a scene depicting a forest landscape that was created using models made freely available by Unity. This was done to replicate how the weather system would look like in-game.

4.1.1 The Weather Profiles

As previously mentioned, several weather profiles were created for the convenience of the user. This section will showcase each of them and outline the effects that were applied to get the look.

4.1.1.1 *Cloudy*

The aforementioned cloud editor component has enabled the creation of several variations of the Cloudy profile by altering the mapping, density, and noise texture distribution of the clouds.

Figure 21: Cloudy profile first variation

Figure 22: Cloudy profile second variation

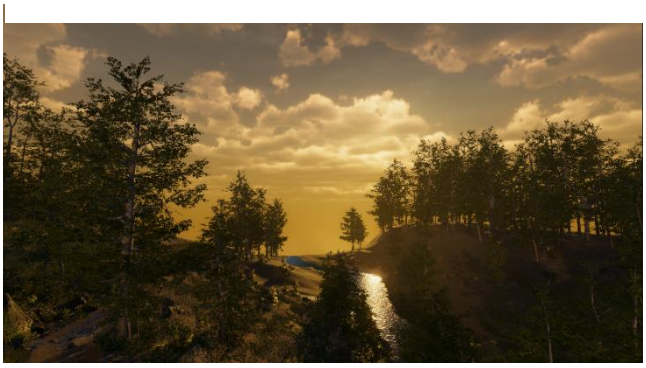


Figure 23: Cloudy profile third variation

Figure 24: Cloudy profile fourth variation

4.1.1.2 *Light rain*

The illusion of light rain is created by lowering the intensity and temperature of the ambient lighting and removing the directional lighting. The rain particle system is activated with a relatively low emission rate and no drizzle. The density of the volumetric and particle clouds is increased.

Figure 25: Light rain profile

4.1.1.3 *Heavy rain*

Adding onto the light rain profile, for a fuller rain effect, the emission rate of the rain particle system is increased, and drizzle and mist are added. The intensity and temperature of the ambient light are further decreased.

Figure 26: Heavy rain profile

4.1.1.4 *Storm*

To create a dynamic, anxiety-inducing storm, the clouds' density was increased, the rain was intensified, and lightning was set to strike periodically.

Figure 27: Storm Profile

4.1.1.5 *Snow*

The snow profile shares the same lighting settings as the light rain one; the only difference lies in the choice of particle system.

Figure 28: Snow Profile

4.1.2 Day-night cycle

Unlike some previous solutions, the weather system presented in this paper is able to produce sunsets and sunrises that are easily distinguishable from one another, mainly through the way that the skybox is rendered. As apparent from the below figures, the sun and moon cast shadows as intended. For added realism, the moon texture is swapped each consecutive night to mimic the different phases of the moon.

Figure 29: Night-time

Figure 30: Dawn

Figure 31: Sunset

Figure 32: Midday

4.2 Performance results

4.2.1 Evaluation technique and metrics

Testing was done on a Windows 10 system with an Intel Core i7 quad-core processor and an NVIDIA GTX 1050Ti graphics card.

The performance will be measured in terms of frames per second (FPS) during run time. This paper aims to presents how a weather system may be incorporated into video games. Therefore a high framerate is of utmost importance for a seamless experience. A framerate of 60 and beyond will be considered an excellent result, while 30 and below will be evaluated as unsatisfactory. We will assess how different combinations of effects of various visual quality perform during a real-time simulation. For this purpose, the weather system will be tested while applied to the same project as in the visual results section.

4.2.2 Testing

Perhaps the most memory-demanding component of the weather system is the volumetric clouds. However, their impact on framerate can be contained by lowering their coverage, density, and noise texture detail. These parameters were incrementally increased and tested while in-game. Testing was first done while no other effects were applied to gauge the impact of this specific component correctly.

Figure 33: Volumetric clouds of very low quality

Figure 34: Volumetric clouds of low quality

Figure 35: Volumetric clouds of medium quality

Figure 36: Volumetric clouds of high quality

Figure 37: Volumetric clouds of very high quality

Figure 38: Volumetric clouds of ultra quality

Clouds quality	FPS	Rendered image
Not applied	109	N/A
Very low	87	Figure 33
Low	82	Figure 34
Medium	75	Figure 35
High	73	Figure 36
Very high	67	Figure 37
Ultra	45	Figure 38

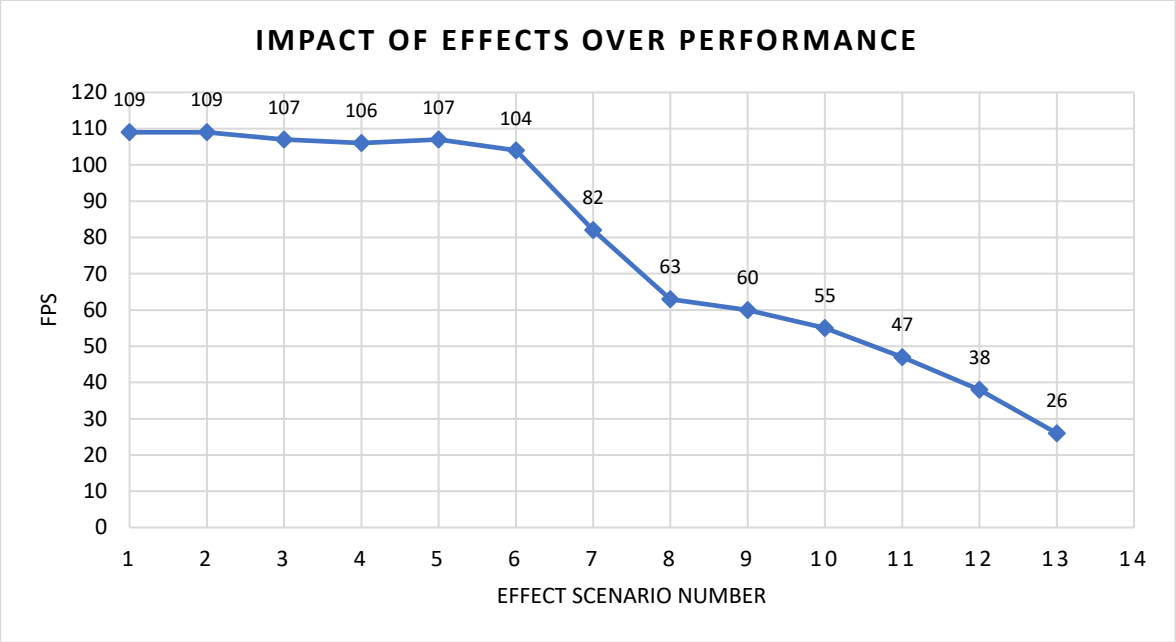
The results show a clear correlation between the clouds' level of quality and performance. As the table indicates, the higher the detail of the clouds, the harder it is to render and the lower the framerate. We can see a significant performance drop just from applying the effect on the very low quality as the framerate is lowered by 22 FPS.

There is also a substantial gap between the *Very high* and *Ultra* quality settings, the performance being 22 FPS slower for the latter. As the density and noise texture intensity parameters are increased, the shape of the cloud becomes more and more detailed, and in turn, the GPU must perform the raymarching steps more and more times which explains the slower render times. The Ultra quality clouds have more detail than a player would be able to notice while in-game and were made just for testing purposes, however, these results are important for future reference.

Now that we have a better understanding of how the most demanding component of the weather system is affecting performance on its own, testing can be done to gauge how many effects can be rendered at once before the system fails. Producing a framerate of 30 FPS and below will be deemed as a failure as the gameplay experience would become frustrating and the player's immersion into the game would be broken.

Effects will be gradually activated based on their performance demands, starting with the least demanding and ending with the most demanding.

Plot point	Particle clouds	Rain	Lightning	Snow	Volumetric light	Volumetric clouds	FPS
1	Inactive	Inactive	Inactive	Inactive	Inactive	Inactive	109
2	Active	Inactive	Inactive	Inactive	Inactive	Inactive	109
3	Active	Active [Light]	Inactive	Inactive	Inactive	Inactive	107
4	Active	Active [Heavy]	Inactive	Inactive	Inactive	Inactive	106
5	Active	Active [Heavy]	Active	Inactive	Inactive	Inactive	107
6	Active	Active [Heavy]	Active	Active	Inactive	Inactive	104
7	Active	Active [Heavy]	Active	Active	Active	Inactive	82
8	Active	Active [Heavy]	Active	Active	Active	Active [Very low quality]	63
9	Active	Active [Heavy]	Active	Active	Active	Active [Low quality]	60
10	Active	Active [Heavy]	Active	Active	Active	Active [Medium quality]	55
11	Active	Active [Heavy]	Active	Active	Active	Active [High quality]	47
12	Active	Active [Heavy]	Active	Active	Active	Active [Very high quality]	38
13	Active	Active [Heavy]	Active	Active	Active	Active [Ultra quality]	26



As apparent from the graph, the weather effects created using particle systems do not have too much of an impact on performance. Observing plot point 6, even though all the available particle systems are running, the framerate is just 5 FPS lower than when none of the effects were active.

The volumetric lights have much more of an impact, however, as activating them slows the performance by a significant 22 FPS.

The performance impact of the volumetric clouds' quality is overall consistent with the previous testing. Combining them with the other components of the weather system, ultimately causes framerates to fall below the admitted 30 FPS.

A future implementation might address these results by focusing on the optimisation of the volumetric lighting and clouds. Providing some volumetric lighting alternatives that are less taxing on the GPU may make for an adequate short-term solution.

5 Conclusions

5.1 Overall summary

The initial goal of this paper was to follow and document the development process of an integrable weather system component. The core graphical techniques that sit behind most modern video games like Volume rendering, Raytracing, or Raymarching were introduced and explained.

The findings of the research phase were put into practice through the creation of a complete realistic weather simulation framework. This paper has demonstrated how to combine recent graphics rendering technologies to achieve some visually aesthetic results of high fidelity to the real-life weather phenomena.

The trade-offs between aesthetic quality and performance were generally not substantial. The lowest recorded framerate was 26 FPS while using all the available components on their highest quality settings. This highlights the integrability of the asset on systems of a high range of computing capabilities.

5.2 Further work

There is much more that could be added to the framework to improve realism and integrability.

For instance, the system has no notion of metric parameters such as temperature or altitude. Having the seasons, time of day, and current weather mapped to a temperature scale could be instrumental for someone who may wish to integrate the weather system into their gameplay mechanics. For the same purposes, an integrated event system would also be a valuable asset to the system.

More effects could undoubtedly be added to facilitate the creation of more specific weather scenarios. A future implementation could support the production of a desert biome by the inclusion of sandstorms or of a glacial one by having aurora borealis and heavier snowstorms.

The graphical quality could be enhanced by including more detailed textures for the particle systems and the skybox. Snow and rain accumulation would hugely expand the realism of a future version, as would the inclusion of reflection probes to make the effects visible in reflective surfaces.

The weather system's functionality could be expanded to 2D games, VR, or even different 3D render pipelines.

Performance could also be further heightened. Although framerates were generally satisfactory during the testing phase, the higher quality settings would most likely perform very poorly on lower-end platforms, meaning that not all users would get to enjoy this system entirely. Optimisation can never be too good as the better this asset performs, the more freedom for users to add other features into their games.

6 References

[1] M. Vainio, "How stunning visual effects bring Ghost of Tsushima to life," 12 1 2021. [Online]. Available: https://blog.playstation.com/2021/01/12/how-stunning-visual-effects-bring-ghost-of-tsushima-to-life/. [Accessed 12 3 2021].

[2] A. B. Nordin, "Immersion And Players' Time Perception In Digital Games," 2014.

[3] C. Jennett, A. L. Cox, P. Cairns, S. Dhoparee, A. Epps, T. Tijs and A. Walton , "Measuring and defining the experience of immersion in games," vol. 66, no. 9, pp. 641-661, 2008.

[4] R. E. Gould, "Weather in Writing: a Dynamic Literary Device," 17 April 2017. [Online]. Available: https://anartfulsequenceofwords.com/2017/04/17/weather-in-writing/. [Accessed 2 April 2021].

[5] K. Schulz, "Writers in the Storm," *A Critic at Large,* 23 November 2015.

[6] E. O'Neill, Eugene O'Neill, 1956.

[7] R. Wright, Native Son, Harper, 1940.

[8] Saki, The Storyteller, 1914.

[9] M. R. Cunningham, "Weather, mood, and helping behavior: Quasi experiments with the sunshine samaritan.," *Journal of Personality and Social Psychology,* vol. 37, no. 11, p. 1947–1956, 1979.

[10] Unity Technologies, "Particle systems," 30 April 2021. [Online]. Available: https://docs.unity3d.com/Manual/ParticleSystems.html. [Accessed 4 May 2021].

[11] W. Reeves, "Particle Systems-A Technique for Modeling a Class of Fuzzy Objects," *ACM Tramsoctions on Graphics,* vol. 2, no. 2, p. 92, April 1983.

[12] M. Bailey and S. Cunningham, Graphics Shaders: Theory and Practice, 2012.

[13] J. d. Vries, Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion, 2020.

[14] NVIDIA Corporation, GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics, 2004.

[15] T. Peterson, Interviewee, *GPU Rendering & Game Graphics Pipeline Explained with nVidia.* [Interview]. 10 May 2016.

[16] K. Suffern, Ray Tracing from the Ground Up, 2016.

[17] M. Dealessandri, "What is the best game engine: is Unity right for you?," 16 January 2020. [Online]. Available: https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you. [Accessed 29 April 2021].

[18] Unity Technologies, "Emission module," Unity Technologies, 30 April 2021. [Online]. Available: https://docs.unity3d.com/Manual/PartSysEmissionModule.html. [Accessed 1 May 2021].

[19] D.L.Dunkerley, "Light and low-intensity rainfalls: A review of their classification, occurrence, and importance in landsurface, ecological and environmental processes," Melbourne, 2020.

[20] R. Clausse and L. Facy, The Clouds, 1961.

[21] A. Schneider, "The Real-Time Volumetric Cloudscapes Of Horison Zero Dawn," in *Siggrapaph*, 2017.

[22] W. Chen, Q. Yang, Y. Chen and W. Liu, "Global Mie Scattering," 2003.

[23] S. Lague, *Coding Adventure: Clouds,* 2019.