Graphical Rendering of Dynamic Weather Systems

Motivation and Rationale

The Context:

A certain degree of realism is required to fully immerse the player within the world they're playing in. Weather is a fundamental aspect in game design since it enhances immersion by a tenfold due to pathetic fallacy, ensuring that the player's emotions are directly influenced by natural phenomena. As gaming is a form of escapism, weather being an unplanned feature of our daily lives also brings in that unpredictable element and can further impose problems for the player to deal with as part of the game progression.

A great example of this is The Legend of Zelda: Breath of the Wild, where if the main character sets fire to dry grass caused by the sun, it creates an updraft that they can ride on and use as a form of transportation. This level of attention to detail shows the untapped potential development companies have when implementing gameplay mechanics. With Breath of the Wild winning Game of the Year in 2017, there can only be advantages towards adding weather effects to games. [1]

The Problem:

When your game lags, the first thing you do is reduce the graphics – namely the cloud shaders, bloom, sun rays etc. This reduces the quality of the intended program. Despite being such an integral part of immersion, sometimes weather is only an afterthought or an added embellishment to a game. [2] This is because complex weather systems require powerful machines to render, run and develop. To capture the true essence of falling snow or rain, for example, there must be a large number of particles in existence at any one time – a severe detriment to computational render times and memory usage. This is why most developers would rather trade aesthetics and animation for speed and efficiency. However, this also shows that as long as high frame rates are achieved and computational costs are low; the possibilities for rendering are endless. [3]

Consider a game development company, it's not smart to assume your target audience have the latest and greatest in hardware. To be more accessible and access a wider market in potential buyers, they must ensure each aspect of the game can be run smoothly – including the weather effects. In this project, I will focus on balancing that ratio to portray realistic reproduction of weather conditions within current consumer hardware capabilities.

My Approach:

I will be using Unity 3D, an industry-standard development platform, and Adobe Photoshop to develop custom materials. Shaders and particle systems within Unity provide me with the tools for development. All aspects of this project will be done on macOS with an Intel Iris Plus Graphics 650 graphics card, and 8GB RAM. I'm focusing on 4 distinct weather conditions: snow, rain, fog, and sun because I feel they are the most distinct states. These also have sub-conditions to further increase complexity, such as a snowstorm which will include a gust of wind and opportunities to add sound to increase immersion. Through this, I will be documenting my findings on how to reduce memory usage, and decrease computational costs enough to still reach that degree of realism which weather systems provide.

Aims and Objectives

Aim: Investigating the production of realistic weather systems efficiently – exploring the balance between performance and appearance.

Objective	Explanation
1. Research	Engage in background reading on rendering weather to form an initial understanding of how to reproduce effects on a standard setup. Investigate the current market and how atmospheric games use weather systems effectively, and the relationship between performance and appearance.
2. Identify	Research and identify key weather systems to implement, choosing carefully to produce a well-rounded project that encompasses the main weather-based immersion methods.
3. Design	Design an appropriate game world to apply the systems to. This should be a custom terrain with peaks and dips to simulate a natural environment. This

	would allow the opportunity to generate a snow cover or show the boundaries between a lower, foggy area and an elevated and clear one.
4. Implementation: Snow	Implement standard snowfall and an additional snowstorm. This includes the snow particle system and a more intense and denser counterpart. Render clouds to set a start point for the particle system. Gusts of wind to depict a snowstorm and move the particles around the game world. And add a procedural snow cover over the terrain.
5. Implementation: Rain	Implement a rain shower and an additional rainstorm over the game world. This includes the rendering rain particles for a standard shower, and then a more intense rainfall. Clouds that gradually turn darker to signify a rainstorm, occasional sounds of thunder crashes, and changes to the skybox. Finally, rain ripples and a glossy appearance over the terrain.
6. Implementation: Sun	Implement a day and night cycle, with the daylight sun clearing the weather effects prior. Used as a reset button to undo previous changes.
7. Implementation: Fog	Implement a heterogeneous fog blanket over the game world that encompasses the terrain, varying in density. This should reduce the visibility of the player-controlled camera.
8. Testing	Repeat testing of the program, ensuring each weather system works independently according to these objective aims, and also influence each other if they're to be activated in succession.
9. Evaluation of performance	In the final stage of development, I will evaluate if the project runs smoothly at 60 FPS and in full screen. A clear written assessment will be created to note down the performance, and compare the project against these objectives – calculating the success of my project.

Background

Resource	Information
Realistic Real-Time Rain Rendering [3]	Description: This paper explores how to portray rain droplets and streaks in line with real-life physics. With the researchers unsatisfied with the relation of the prevalence of rain, and how inaccurately represented it is within games. Reason: This provided me with initial insight into the depth of where my project could go, with Rousseau et al examining the physical properties of raindrops and their refraction. They also briefly investigated how snow and fog can be translated into a game setting, which helped me build an initial understanding of how my project could render these conditions within Unity's capabilities.
A Spectral-Particle Hybrid Method for Rendering Falling Snow [4]	Description: This paper describes the issues with the aim of an accurate depiction of snow, and the huge computational costs that come with it. They propose a spectral-particle method which is similar to a billboard approach, where you replace distant objects with images. In doing so they were able to show a denser snowfall with minimal cost, and in the end, the result showed a more accurate depiction than increasing particle count itself. Reason: Because I plan to also implement the storm counterpart to snow and rain, this would require me to increase the density of these conditions. Understandably, increasing the particle count would directly influence the power needed to run the program. Therefore, I needed to find ways to reproduce that effect efficiently, and this paper provided me with an alternative.
Computer Modelling of Fallen Snow [5]	Description: This paper investigates how to model a thick layer of snowfall on the ground. The accumulation pattern differs upon the surfaces it lands on, and this introduces "flake flutter," the notion that parts of an object that has no direct exposure to the sky can also be coated in snow

	Reason: Producing falling snow with particle systems makes up only half of the challenge. There is also the procedural accumulation of snow upon the game world. At first, it seemed like a straightforward task, but upon reading this paper I realise that many factors such as the directions of the object matter when rendering a blanket over their material. The introduction of flake flutter presents another consideration upon the project.
Foggy Scene Rendering Based on Transmission Map Estimation [6]	Description: This paper explores the realistic rendering of fog in game development, taking extra caution to avoid producing homogeneous fog with a uniform appearance throughout the blanket. Introduces Perlin noise and its role in generating heterogeneous fog with varying degrees of shape and density. Reason: This helps to provide an understanding of how to produce fog accurately. The paper identifies different factors to consider upon development, and I've learnt to steer away from solely using particle systems which would produce homogeneous results. It is especially helpful considering that the user will be able to move the camera around the world space, and the fog should cater to that movement by changing its appearance.
Real-Time Rendering of Volumetric Clouds [7]	Description: This dissertation paper investigates alternative methods for rendering clouds other than having a library of cloud images. This would be a sufficient solution but the cost of having a large library with high-resolution images would be excessive. Like with the previous paper, this also considers Perlin noise but with the combination of an inverted Worley noise to introduce cloud-like shapes. Reason: Within the thesis are visual goals that the author has outlined, which inspired my own implementation of clouds. This includes varying size and coverage, consideration of lighting, and the importance of soft shadows.
Rendering of 3D Dynamic Virtual Environments [8]	Description: This paper describes a custom framework developed to encompass the main features of a dynamic 3D virtual environment. This includes detailed insight into the terrain, skyboxes, and sound. Moreover, this considers past research on reproducing 3D environments. Reason: The research conducted consolidates information from previous investigations on this topic, which is why I chose this reading since it's similar to the position I'm currently at with the project. The method in which they generate the terrain is of importance, utilising heightmaps and textures. But overall, their framework contains methods on the dynamic simulation of weather, clouds, rain and lightning too, which I'll take great inspiration from in terms of the direction they took, and apply it to my understanding of Unity.

Work Plan

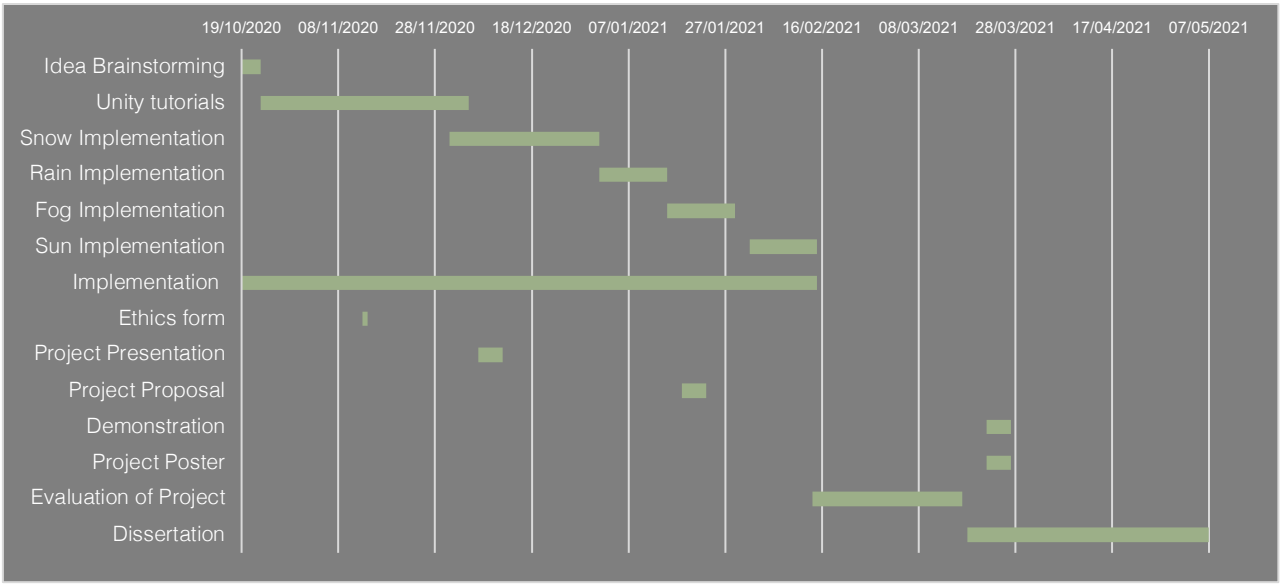

Figure 1. Gantt chart.

Start Date	End Date	Description	Days	(Weeks)
19/10/2020	23/10/2020	Idea Brainstorming	4	<1
23/10/2020	01/12/2020	Unity tutorials	43	6
01/12/2020	31/12/2020	Snow Implementation	31	4
01/01/2021	14/01/2021	Rain Implementation	14	2
15/01/2021	31/01/2021	Fog Implementation	14	2
01/02/2021	14/02/2021	Sun Implementation	14	2
19/10/2020	14/02/2021	Implementation	119	17
13/11/2020	13/11/2020	Ethics form	1	<1
07/12/2020	11/12/2020	Project Presentation	5	<1
18/01/2021	22/01/2021	Project Proposal	5	<1
22/03/2021	26/03/2021	Demonstration	5	<1
22/03/2021	26/03/2021	Project Poster	5	<1
14/02/2021	17/03/2021	Evaluation of Project	31	4
18/03/2021	07/05/2021	Dissertation	50	7

Figure 2. Supplementary information from the Gantt chart.

Overview:

Figure 1 visualises the project timeline, with Figure 2 expanding on the time frames of each activity. I attend fortnightly meetings with my supervisors to discuss objectives for the next two weeks. An agile approach to the project – implementation, documentation, and then presenting for feedback. Throughout all of these, I will also be focusing on the other dissertation assignments on the side.

To begin with, I brainstormed ideas of projects to undertake at the start of the year, then I started to learn how to use Unity to generate different weather systems. As I'm also taking a Graphics for Games module that focuses on teaching Unity, I've learnt the inner workings of the platform, and the time frame in which I complete this section of the project will be quicker. By allocating 4 months towards development, my plan will take me to mid-February at the latest.

However, because of the COVID-19 pandemic, I have limited access to equipment on campus, so development will be slightly hindered due to technology restrictions. But this also further supports my project since I'll be developing on a standard machine. On top of this, the new examination plans because of the pandemic have caused all of my modules to be coursework based. Because of this, I have had to adjust the time I put aside towards the dissertation in Semester 1 to Semester 2.

I will dedicate most of my time towards development during term time. However, if I fall behind on schedule I have the periods between semesters to catch up. Most of the development will have elapsed during Christmas break, and further into the start of 2021.

I begin the project straight into implementation to form the basis of the project. The Gantt chart shows the various systems I plan to include, and as of now, I used the Christmas break to make considerable progress with the implementation on all systems, nearly finishing rain, snow and their respective sub-conditions. Sun and fog are at an early development stage but since I've become more accustomed to Unity, the time to create subsequent systems decrease exponentially. This leaves me at the 5/6th objective.

By Easter break, I will move onto evaluating the project. In that time, I will be debugging and refining the program, and liaising with my supervisors to see if there are any improvements to be made. I will also be comparing the program with my objectives, identifying which aspects of my goals were outside the scope of this dissertation and forwarding it to the final document for evaluation.

From March I aim to focus on the documentation and writing the dissertation itself, consolidating the feedback given by my supervisors, and finalising the system and its performance requirements for the dissertation deadline.

References

[1] 2017 | History | The Game Awards. 2017. 2017 | History | The Game Awards. [online] Available at: <https://thegameawards.com/history/year-2017> [Accessed 22 January 2021]. (Rousseau, et al., 2006)

[2] Barton, M., 2008. How's the Weather: Simulating Weather in Virtual Environments. Game Studies, [online] (1). Available at: <http://gamestudies.org/0801/articles/barton> [Accessed 22 January 2021].

[3] Rousseau, P., Jolivet, V. and Ghazanfarpour, D., 2006. Realistic real-time rain rendering. Computers & Graphics, 30(4), pp.507-518.

[4] Langer, M., Zhang, L., Klein, A., Bhatia, A., Pereira, J. and Rekhi, D., 2004. A spectral-particle hybrid method for rendering falling snow. Eurographics Symposium on Rendering.

[5] Fearing, P., 2000. Computer modelling of fallen snow. Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00.

[6] Guo, F., Tang, J. and Xiao, X., 2014. Foggy Scene Rendering Based on Transmission Map Estimation. International Journal of Computer Games Technology, 2014, pp.1-13.

[7] Häggström, F., 2018. Real-Time Rendering of Volumetric Clouds. Master of Science in Engineering. Umeå University.

[8] Ferrara, E., Catanese, S., Fiumara, G. and Pagano, F., 2011. Rendering of 3D Dynamic Virtual Environments. Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques.