Graphical rendering of dynamic weather systems

Connor Lee

May 2018

BSc Computer Science with Game Engineering

Dr Gary Ushaw

Word count: 9967 words

Abstract

Weather systems are one component of making a believable, immersive game world. More and more games are making weather systems which affect a game world in various ways. The significance of having a weather system which affects the world helps to immerse the player more into the game.

I find the idea of weather such as rain making hair look wet in games interesting. My project aims to investigate and build a weather system which affects an interactive world in this kind of way.

Declaration

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

Acknowledgements

I would like to thank Dr William Blewitt, Dr Gary Ushaw and Dr Richard Davison, for their help and support throughout the course of this project.

Table of Contents

Table of Figures

Table of Tables

Chapter 1 – Introduction

1.1 Motivation and rationale

This project will be about the graphical rendering of dynamic weather systems. It involves using graphical programming techniques to render weather conditions in a virtual world. In this proposal, I will be talking about why I chose this project and how I plan on achieving my aim.

Often, weather systems are used to enhance the immersion in a video game and thus they are typically seen in many games. A weather system is the state of the atmosphere in an environment. Examples of this includes weather such as rain or snow. Weather systems help player's get immersed because rain could be used to invoke sombre mood or make the game realistic through careful imitation of our own world's weather systems - day and night, sunny spells to rainy showers.

In modern game development, the goal is to develop a world which feels believable (Barton, 2008). This led me to consider how weather simulation can be used to help represent a virtual environment and how such a feature may affect the immersion of the player. Imagine a city building simulation game, if the weather was always sunny and never changed, the city itself would feel fake and unnatural.

Weather systems in games are often made using particle systems. A particle system is a collection of many minute particles that together represent a fuzzy object (Reeves, 1983). We can use this technique in graphics to simulate visual effects such as clouds, smoke and rain.

The aim of the project is to develop a weather system and then make the weather system affect an interactive world. I will examine how I can use a weather system to affect the environment that it is used in. For example, in the real-world, rain would make a road look glossier or snow would settle on the ground. This is desirable as weather systems are implemented into many games but they only consist of visual effects rather than playing a part of the physics of the world and affecting it. This can make the world feel less lifelike to a player. An example of a game which does this is Final Fantasy XV where they use rain to make things look wet (Elcott et al., 2016).

Figure 1: Screenshot of Final Fantasy XV's rain effect by Square Enix

This project is appealing to me because I found that I enjoyed seeing weather visually in games and it inspired me to produce my own version of a weather system.

1.2 Aim and objectives

Aim: To implement a dynamic weather system which affects an interactive world in a realistic manner and to determine its performance in real time.

Objectives:

1. To research into weather conditions in the real world, existing systems and the various technologies behind graphical rendering of weather systems.

This objective is what I will use to test if I have achieved my aim to build a dynamic weather system which affects an interactive world. I will achieve this objective by considering rain, snow, games such as Final Fantasy XV and technologies such as OpenGL. I will define weather types that I can consider implementing in my weather system.

2. To design the graphical environment and weather conditions based off the research I have done.

Using the research I have done, I will make a design that allows me to showcase all the main attributes of the weather conditions. Furthermore, I will outline the basic features that my weather conditions must show from the research as well. The attributes defined earlier will be used as part of the design of my weather conditions for how weather affects the world.

3. To develop graphical environment and weather system using the designs I have created.

The initial implementation of my environment and weather system will be done using particle systems, OpenGL and my previous designs. For this to be achieved the environment should be as close to the design as possible and the basic features of the weather conditions that were outlined earlier must also be fulfilled as close as it can be.

4. To implement how my weather system will affect the interactive world.

Once the initial implementation has been created, I will then focus on making my weather system interact with the environment. This will be based off the design defined earlier and how well this is achieved will be dependent on how close the implementation of the attributes is to the design.

5. To evaluate the performance of my weather system and whether the requirements have been satisfied.

I will evaluate my project in two ways. Firstly, I will use frame time and memory usage to measure the performance of my weather system in real time. Secondly, I will check

if my weather system fulfilled the requirements I laid out before and subjectively evaluating how close it is to achieving my original aim.

1.3 Dissertation outline

Chapter 1 - Introduction

This chapter describes the project which has been undertaken, the reasoning behind it and the aims and objectives of the project. The contents of this chapter:

- Motivation and rationale,
- Aim and objectives,
- Dissertation outline.

Chapter 2 - Background research

This chapter explores the technologies used in this area of work, existing systems and the weather. The contents of this chapter:

- Literature reviews,
- Technologies,
- Particle systems,
- Heightmap,
- Skybox,
- Existing systems,
- Weather system.

Chapter 3 - Design

This chapter illustrates the methodology that will be used, the architecture of the system to be developed and the design of the particle systems and environment. The contents of this chapter:

- Development methodology,
- Particle system design,
- Graphical environment design,
- User interaction,
- Skybox,
- Requirements,
- System architecture.

Chapter 4 - Implementation

This chapter shows the implementation of the various parts of the system such as the environment, weather system and night/day system. The contents of this chapter:

- Heightmap,
- Particle system,

    - Snow weather,
    - Rain weather,
- Wetness/snow dusting,
- Snow accumulation,
- Night/day system.

Chapter 5 - Testing

This chapter compares the performance of the system through frame time and memory imprint. The contents of this chapter:

- Methodology,
- Results.

Chapter 6 - Evaluation

This chapter reviews the test results from the previous chapter and evaluates the project based on the requirements laid out earlier in chapter three. The contents of this chapter:

- Development methodology evaluation,
- Results evaluation,
- Requirements evaluation,
- Subjective evaluation.

Chapter 7 - Conclusion

This is the final chapter of the dissertation which summarises the project, reviewing the original aims and objectives, discussing my own thoughts and looking at future work. The contents of this chapter:

- Review aims and objectives,
- Personal development,
- Future work,
- Final thoughts?

Chapter 2 – Background Research

2.1 Literature reviews

Paper: (Barton, 2008) "How's the Weather: Simulating Weather in Virtual Environments"
Summary: This article talks about how weather is used in video games and its role in a virtual world.
Relevance: This source is relevant to my project as it talks about why it is useful to use weather systems in video games and how it can be used to help make the world believable to the player.

Paper: (Fearing, 2000) "Computer Modelling of Fallen Snow"
Summary: This paper talks about an algorithm to simulate snow accumulation in an environment.
Relevance: This is relevant to my project as one of my aims is specifically to make my weather affect the environment and this is one way of showing snow impacting a virtual world.

Paper: (Latta, 2004) "Building a Million Particle System"
Summary: This paper is about utilising the GPU for particle systems to have a larger number of particles rendered at once and in real time.
Relevance: This is relevant to my project because I will be rendering particle systems on the GPU via OpenGL.

Paper: (Reeves, 1983) "Particle Systems – A Technique for Modelling a Class of Fuzzy Objects"
Summary: This paper is about particle systems. It gives an overview about how particle systems work and their applications.
Relevance: This source is relevant to my project as I will be using particle systems to create my weather system. Furthermore, it helps explains how it works to me so I understand the underlying concepts of a particle system.

Paper: (Wang et al., 2006) "Real-time Rendering of Realistic Rain"
Summary: This paper showcases a method to generate realistic rain in real time over a video.
Relevance: This source is relevant because my project will be running in real time and it also suggests something that could be done to make their own work more realistic. In this case, they suggest adding "rain splatters to the ground" which could be something I could add as part of my aim to "affect an interactive world".

2.2 Technologies

C++ is a general purpose object-oriented programming language. It is an extension of the C language and therefore can be coded in "C style" or "object-oriented style". C++ is used for game development because as it is deterministic, it offers high performance and it offers enough abstraction to be higher level than other fast languages (Stackoverflow.com, 2010) Furthermore, it provides low-level memory manipulation which allows programmers to control memory with more precision. This is important in games as games can often use a large amount of memory and thus memory management is crucial.

DirectX is a set of APIs which allows software to write instructions to audio and video hardware. One of the benefits to using DirectX is that Windows comes with DirectX built-in and is supported by Microsoft. This is good because 95.96% (Hockenhull, 2017) of the Steam community runs windows.

OpenGL is a graphics API which allows the development of interactive 2D and 3D graphics applications. One of the main benefits to using OpenGL for game development is due to the fact it is "…open, vendor-neutral, multiplatform graphics standard." (Group Inc, n.d.). This means that we can develop on one platform using OpenGL and it can work across different platforms and graphic cards.

In my project, I have chosen to use OpenGL due to its nature of being open platform over DirectX. I will also be coding in C++ due to its high performance and ability to manage memory.

2.3 Particle systems

As described earlier in chapter one, a particle system is a collection of many minute particles that together represent a fuzzy object (Reeves, 1983). They are often used in many different graphical applications such as fire, smoke and other visual effects.

In a particle system, there can be thousands of particles. Each particle has their own unique attributes such as the following:

- Position,
- Size,
- Velocity,
- Colour,
- Lifetime.

Particle systems control the attributes of each particle. A particle system typically follows these sequences of steps:

1. Generating new particles,
2. Assigning individual attributes to each particle,
3. Any particles which past their lifetime are destroyed/reset,
4. Moving any living particles according to their attributes,

5. Rendering the particle.

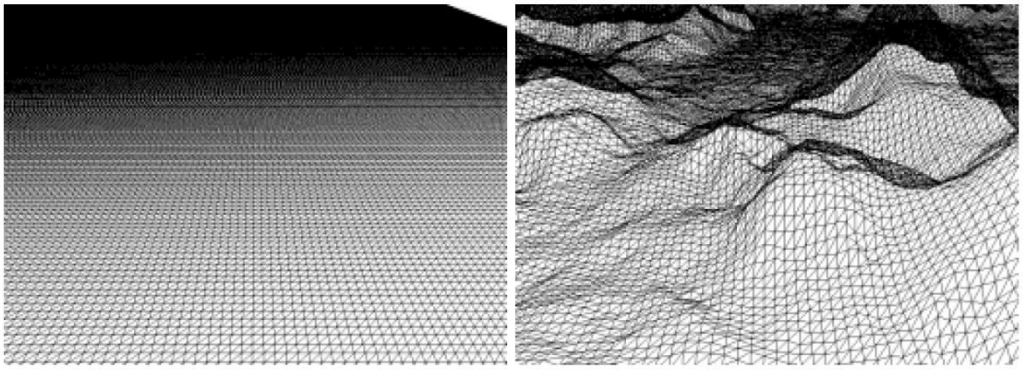
All particles are continuously updated during the running of the particle system and after reaching the end of their lifetime, they will be reset so they can be produce the same effect from the beginning.

Weather in the real world often have thousands upon thousands of raindrops or snowflakes. Due to the properties of a particle system, this makes a particle system very suitable for the generation of weather effects. Therefore, I will be using a particle system to simulate the effects of weather in my project.

2.4 Heightmap

A heightmap is an image used to store values such as surface elevation data. Heightmaps are typically used for bump mapping or for terrain. This works by having the grey values in the image correspond to different heights of a piece of terrain. This is a useful technique to use as it uses less memory for level of detail you get and it is an easy way to generate complex terrain (see Figure 2).

In my project, this is good to show the interactive environment I desire as it has multiple different heights and flat surfaces - a variation of terrain to work with. This will be useful to showcase snow accumulation for example.



Left: A flat grid of triangles. Right: The same grid, with vertex heights adjusted to create a landscape

Figure 2: Heightmap example (Davison, 2012)

2.5 Skybox

A skybox is a six-sided box which is made up of six different images. The six different images are placed on the different faces of the cube to create a seamless image. Skyboxes are a solution used to display skies, along with other distant, unreachable areas such as distant mountains (see Figure 3).

This is useful to my project as a sky is one important part in creating a believable weather system.

Figure 3: Skybox example (Meiri, n.d.)

2.6 Existing systems

Final Fantasy XV

Final Fantasy XV has a widely complex weather system for the world they have created. For their sky, they generate the whole sky procedurally - sun, stars, moon, clouds and atmospheric scattering. The weather system they use links the "weather" (set of parameters) with game entities. The kind of parameters they have are things like: sky, rain or wind. This would affect the shading, visual effects and animation of game entities.
The rain weather they have in the game is produced via a GPU particle system. The falling particles are centred on the camera and splash particles are emitted from the surface. Then they have a shader for making things look wet. This consists of increasing specular, decreasing roughness, darkening diffuse and distortion of normal depending on wetness and flood levels of a material. An example of this is shown earlier in Figure 1. (Elcott et al., 2016)

This work is relevant to my project objectives as one of the objectives is to create weather which affects an interactive world. The weather system in Final Fantasy XV: has rain making things look wet, splash particles when rain hits the surface and weather system "parameters" for game entities. These are the kind of things I could aim to achieve in my project.

Horizon Zero Dawn

Horizon Zero Dawn creates real time volumetric clouds and uses the technique they have developed to model a cloud system. The cloud system contains various controls such as "Cloud density", "Precipitation" and "Cloud type". In the weather system, there is a function for "Cloud coverage", "Cloud type" and "Precipitation". The "Precipitation" control is used to draw rain clouds. When there is a "precipitation signal", the cloud coverage is changed to 70% (thereby changing the look of the clouds

in the sky). The reason why they modelled it like this is because when the density of clouds is high, precipitation occurs such as rain or snow. As well as changing the cloud coverage, the precipitation control also creates the rain effects (see Figure 4).

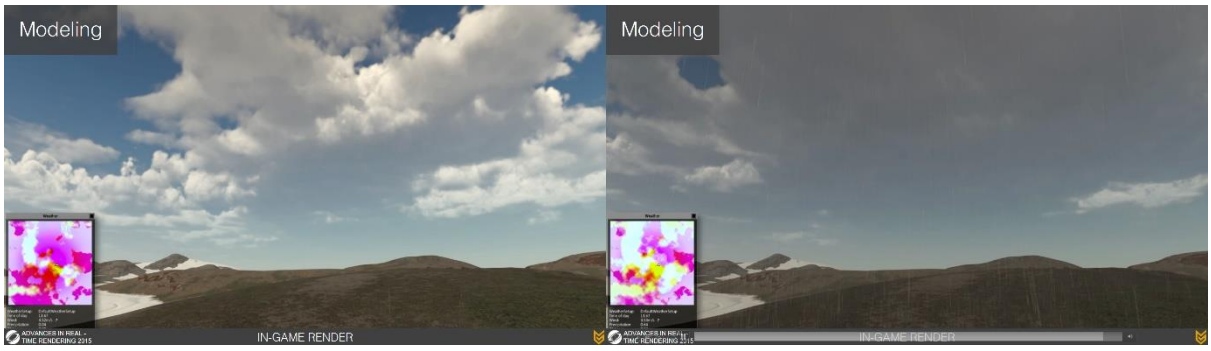


Figure 4: Horizon Zero Dawn, different precipitation levels (Schneider, 2015)

This work is relevant to my project objectives as one of the objectives is to create weather which affects an interactive world. The weather system in this game, modifies the density of the clouds to match what would occur in the real world to a degree when precipitation occurs.

2.6 Weather system

Rain

Rain is typically "longer" looking than snow is and moves fast.

The kind of attributes that rain has on the environment are:

- Rain puddles,
- Wetness,
- Splash effects upon hitting the ground.

Snow

Snow is typically more "rounded" and moves slowly typically (unless it's a very heavy snow storm)

The kind of attributes that snow has on the environment are:

- Snow accumulation,
- Snow covering/dusting (thin layer of snow),
- Deformable snow.

Chapter 3 - Design

3.1 Development methodology

The development methodology that was undertaken was an agile methodology. Agile methodology is an approach to software development emphasising incremental development and continuous planning. Agile development focuses on individuals and interactions, working software, responding to changes in development and customer collaboration.

Due to the nature of regular meetings with my supervisors, this was a suitable development methodology to use. It allowed feedback from my supervisors early as development was done continuously and it made it easy to decide set weekly goals to complete. Furthermore, it meant that if I ever ran out of time, I had a piece of software that was partially functional.

3.2 Particle system design

In my project, I will be creating two types of weather: snow and rain.

Both weather types will consist of particles which have the following attributes:

- Position - determines its location in the world,
- Colour - the colour of the particle,
- Velocity - how fast it goes and in what direction,
- Life - how long the particle lives for.

These attributes were based off the basic particle system model described in chapter 2.3. The size of all particles will be set to pre-set size.

The particle system will contain:

- Number of particles - how many particles are outputted at once,
- Wind speed - change the direction of particles based on a wind factor,
- Blend factor - to determine how “wet” or “snowy” something looks.

The wind speed factor allows us to simulate wind in our weather system and the blend factor enables levels of wetness or snow covering/dusting in our environment.

Snow weather particles will look like circles and will be coloured white (see Figure 5, this is coloured black so you can see it). The circle shape was chosen as it is the simplest way to show a snowflake. They will affect the environment by making the textures on the heightmap look snowier (this is the snow covering/dusting attribute talked about in chapter 2.6) and snow will be able to accumulate on the surfaces of the environment (this is the snow accumulation attribute).

Figure 5: Snow particle image

Rain weather particles will look like a long, thin oval shape and will be coloured light blue (see Figure 6). This shape was chosen for rain as it often looks like thin lines when they fall in real life. They will affect the environment by making the surfaces of the environment look more wet.

Figure 6: Rain particle image

3.3 Graphical environment design

The graphical environment will consist of a heightmap. This was chosen due to the fact a heightmap allows complex terrain to be generated. The complex nature of the terrain allows us to showcase snow accumulation because we will be able to see snow levels increase on more flat terrain and gradually rise against the mountains/steep terrain generated. Furthermore, it can show how snow/rain does not settle on high slopes.

3.4 User interaction

The user will be able to press different buttons on the keyboard to enable different weather effects to occur. The user interaction that will be done is:

- "1" to render snow,
- "2" to render rain,
- "3" to reset the simulation,
- "6" to increase wind speed in the x direction,
- "7" to decrease wind speed in the x direction,
- "8" to increase wind speed in the z direction,
- "9" to decrease wind speed in the z direction,
- "0" to reset the wind speed to 0.

3.5 Skybox

This will consist of two cube-maps, one for day time (see Figure 7) and one for night time (see Figure 8). I decided to use two cube-map textures as it made it easy to allow the program the option to swap between textures rather than swapping between two separate skyboxes. Whilst the program is running, it will automatically change between night and day. It will blend into the other based on a blend factor until it is fully covers the other and thus is that time of day. The blending works by taking the

time of day factor, using that to get a certain percentage of one texture and the other and then merging the textures together. Once it becomes “night-time” or “day-time”, the process is then reversed to get the full cycle.

Figure 7: Day cube map

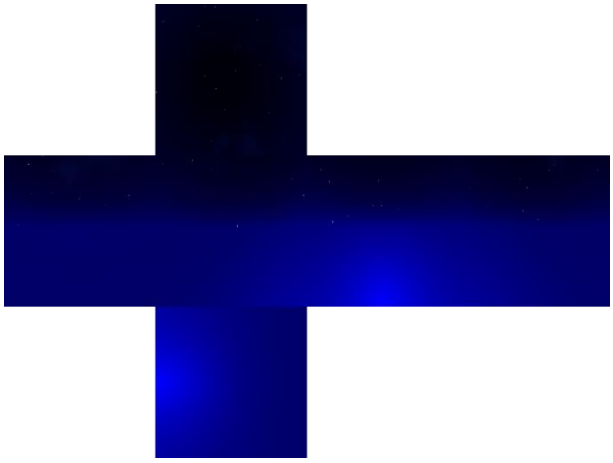


Figure 8: Night cube map

3.6 Requirements

The functional requirements that are required in this project will be listed below.

No.	*Functional Requirements*
1	The system should display a heightmap.
2	The system should display a particle system which looks like rain.
3	The system should display a particle system which looks like snow.
4	The rain particle system should make the environment texture look wet.
5	The snow particle system should make the environment texture look snowier.
6	The snow particle system should accumulate snow on the environment.
7	There should be a skybox which looks like day time.
8	There should be a skybox which looks like night time.
9	The user should be able to choose to show snow weather.

10	The user should be able to choose to show rain weather.
11	The user should be able to reset the simulation to the default state.
12	The user should be able to increase the wind speed in the x direction.
13	The user should be able to decrease the wind speed in the x direction.
14	The user should be able to increase the wind speed in the z direction.
15	The user should be able to decrease the wind speed in the z direction.

Table 1: Functional requirements

The non-functional requirements are listed below.

No.	*Non-functional requirements*
1	The application should run smoothly.
2	The application should run efficiently.
3	The application shouldn't crash.
4	The application should run on different machines.

Table 2: Non-functional requirements

I will measure the success of the non-functional requirements by:

1. Looking at the frame time to see how long it takes to render a single frame.
2. Check the memory usage of the application (is it low?).
3. Make sure the application doesn't crash (the mean time between failures is low).
4. Run the application on at least three different machines and see if it runs as expected.

3.7 System architecture

Figure 9 depicts the architecture of the system, showing the core parts of the system.

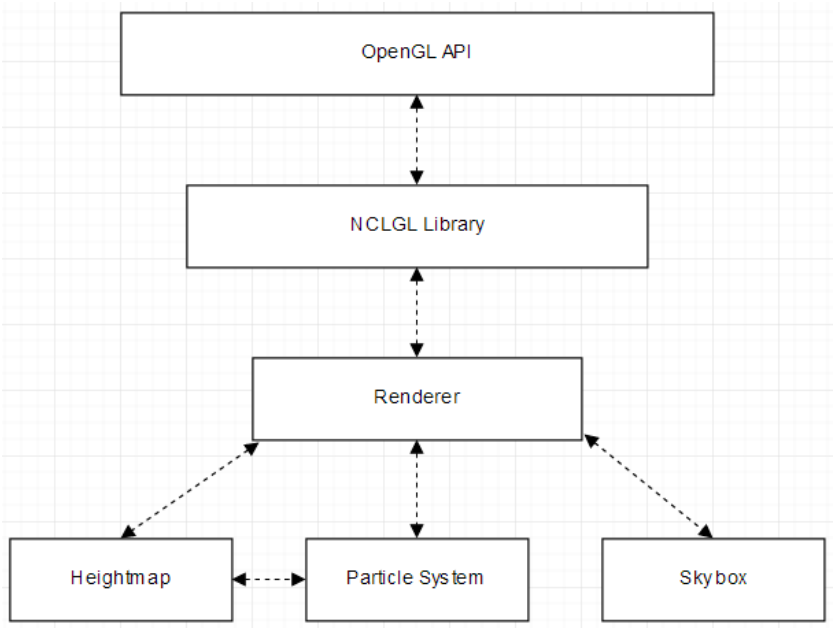


Figure 9: System architecture model

There are six main parts to the system:

- OpenGL API - allows graphics data to be sent to graphics hardware for processing.
- NCLGL Library - an OpenGL framework which provides an OpenGL context and other useful functionality.
- Renderer - the OpenGL context inherited from the OGLRenderer in the NCLGL library which allows you to render objects.
- Heightmap - the graphical environment which can communicate with the particle system so it can be modified depending on the weather.
- Particle System - the weather system for displaying weather.
- Skybox - the day/night cycle which automatically changes between day and night.

The heightmap, particle system and skybox are passed into the renderer to be rendered which then passes it back to OpenGL context created so it can be rendered on the graphics hardware.

This is a class diagram which depicts the Heightmap and Particle System classes. They both inherit from the Mesh class which is a class from the NCLGL library. The particle system is used to create the different weather types in my weather system. The heightmap is used to create the environment.

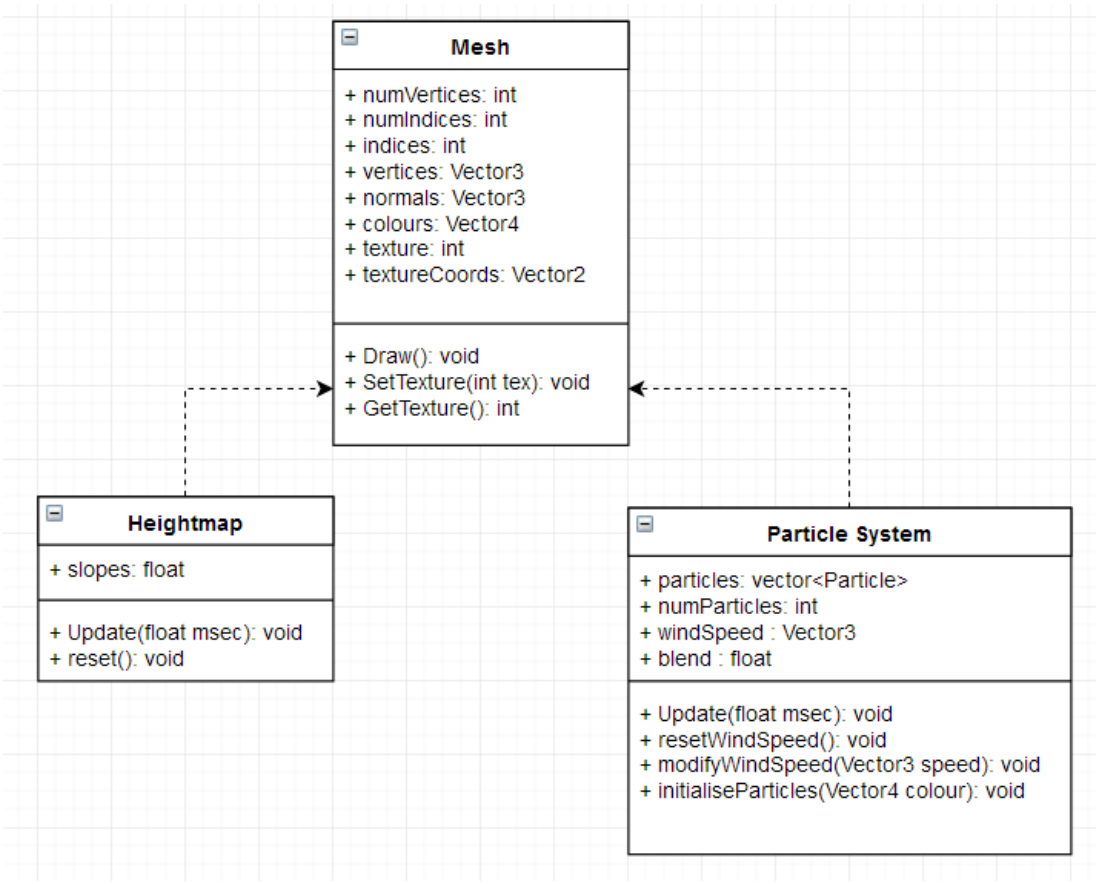


Figure 10: Class diagram of the Heightmap and Particle System classes

This is the use case diagram, showing the user interaction that the user can have in the system. The user must first display a weather to be able to do any other command. The first two options are required to do any other option the system and they also fulfil requirements nine and ten. After that, the wind speed modifications correspond to twelve, thirteen, fourteen and fifteen respectively. The reset simulation and reset wind speed use case corresponds to requirement eleven.

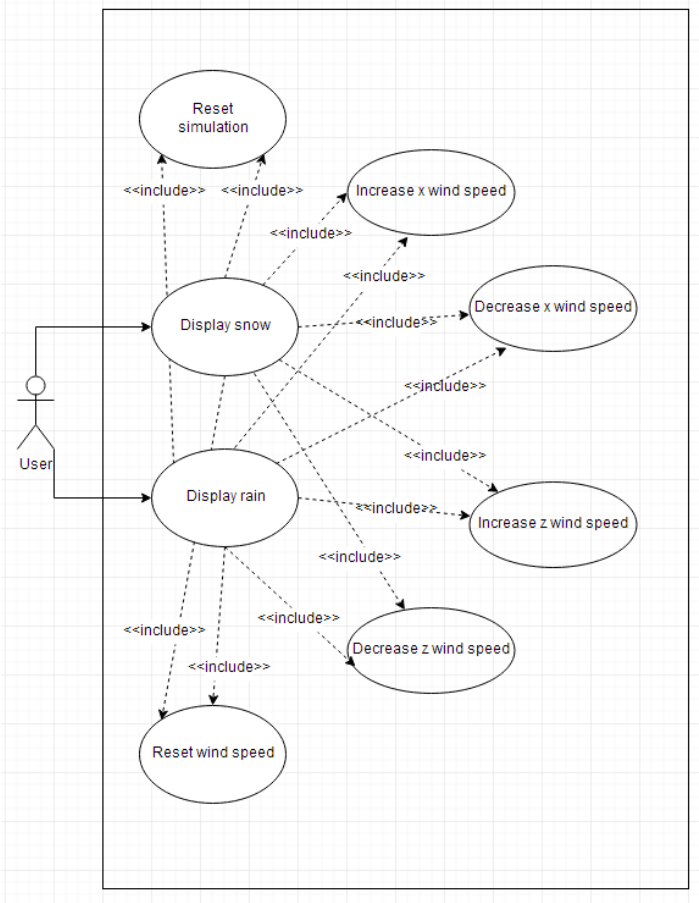


Figure 11: Use case diagram

Chapter 4 - Implementation

4.1 Heightmap

The Heightmap class inherits from the Mesh class in the NCLGL library and it would read in a heightmap file into an array variable we call data. These height values, from the heightmap file, are used as the "y" values in the vertices generated by the Heightmap. The "x" and "z" values are generated based on the grid size (which are defined in the class) and the position in the data array. Once we have the "x", "y" and "z" values, we have all the vertices required to generate indices. This data can then be buffered into the graphics card, ready to be used.

After generating our vertex data, we use two shaders, the vertex and fragment shader, to turn vertex data into an image on the screen.

Our heightmap vertex shader transforms incoming vertices from their local space into clip space. To do this we pass in three matrices: model, view and projection matrices. These three matrices form what is called a MVP matrix. The model matrix transforms vertices from their local space into world space. The view matrix transforms all objects from world space into view space based off the camera in the scene. Lastly, the projection matrix transforms world space into clip space. Following the completion of these transformations, fragments are generated which are then used in the fragment shader.

Our heightmap fragment shader simply takes in the results of the vertex shader and then proceeds to sample textures onto the fragments created. Once texturing has been completed, the final image is then rendered on the screen.

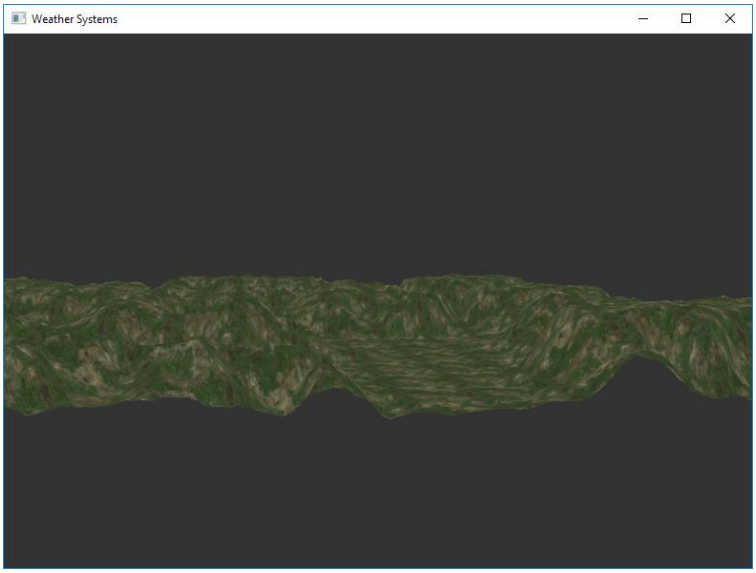


Figure 12: Initial heightmap

4.2 Particle system

To begin creating the weather our Particle System class inherits from the Mesh class and a constructor with several parameters is used to help initialise the weather. The core functionality of the particle system consists of three things: Initialisation, Draw and Update. The main technique we use to also help create the particle system is billboarding. Billboarding allows us to display an object that always face a given camera.

Within the Initialisation stage, we create a vector of particles with the four attributes described in chapter 3.2 (position, colour, velocity and life). The number of particles in the vector is determined by a parameter from the constructor and for every particle we do the following:

- Set its colour attribute based off a parameter from the constructor.
- Randomise the position, setting the position to be above the heightmap and contained within the heightmap.
- Set the lifetime to a slightly randomised value.
- Set the velocity to a slightly randomised value.

The lifetime is set to a slightly randomised value to allow the weather system to look like particles are coming out at different rates. This is because later in the Update function we "reset" each particle once its lifetime is over. Once the particles have been created, it can then be used in the Draw call.

The Draw stage consisted of setting up the vertex buffers to allow the data to be used in the shaders. I had encountered two issues at this stage. The first issue was where the shaders wouldn't correctly texture the particles correctly. This was fixed by fixing the shader programs I had written, which allowed the particles to be then textured correctly again. The second issue was a memory issue. When the Particle System ran, the memory would consistently increase due to each frame updating all the vertices that it had contained. To fix this, a new method called "*RebufferData()*" was added which used "*glBufferSubData*" instead of "*glBufferData*". This works because "*glBufferData*" reallocates memory each time it is called whilst "*glBufferSubData*" avoids allocating new memory.

We use three shaders in our particle system - vertex, fragment and geometry shader. The vertex and fragment shader perform the same tasks as the shaders in Heightmap except before sending the fragments to the fragment shader, we send them to the geometry shader first then send it to fragments shader last. The geometry shader allowed us to turn points into quads. This allows us to provide four points the fragment shader could use to texture with.

The Update stage consisted of updating the position and lifetime of every particle in the particle system. Once the lifetime had reached zero, that particle would be "reset" by having its position randomised near the top again and given a new lifetime.

Snow weather

To initialise snow weather, I set the texture of the particle system to be the same as the one shown in chapter 3.2 for snow. I also set the speed of the fall to be a low value to try to simulate a slow-moving snow weather effect and set the colour of the particles to white.

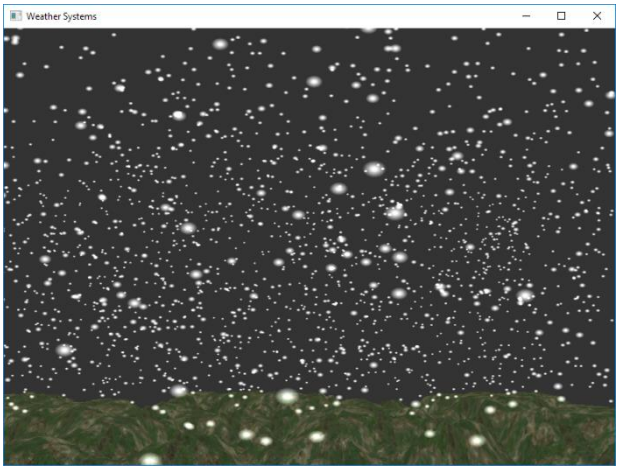


Figure 13: Snow weather

Rain weather

To initialise the rain weather, I set the texture of the particle system to be the one shown in chapter 3.2 for rain. I set the speed to be a somewhat high value as rain typically moves down fast and set the colour to a light blue colour. I would have preferred to do a more transparent look but it was easiest to showcase the difference between my rain and snow weather condition through the two colours.

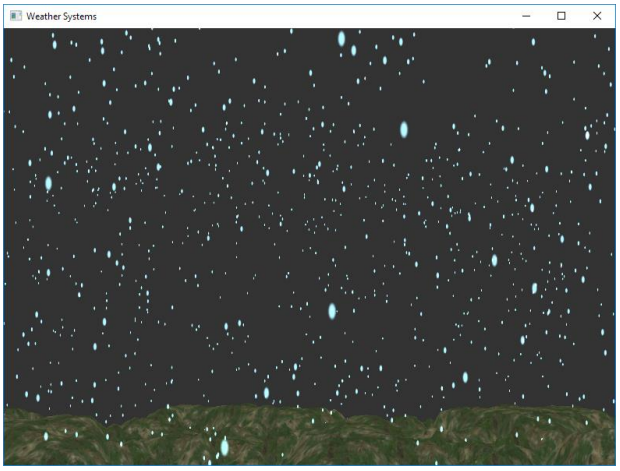


Figure 14: Rain weather

Wind

Wind was initially set to zero in the particle system. Wind was implemented by adding a velocity to the particles so that they would move in a certain direction. I added the

controls mentioned in chapter 3.4 so the user can modify the wind speed themselves. This control simply modified the velocity of the particles to make them move in a certain direction. An example of it being moved in the x direction is shown below in figure 15.

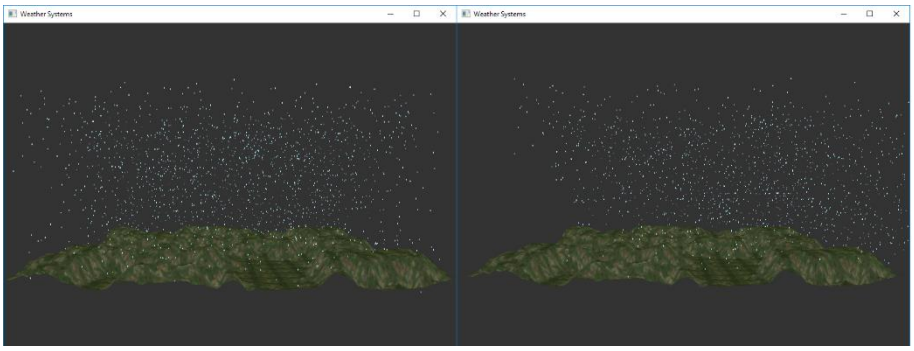


Figure 15: Wind increased in the x direction

4.3 Wetness/snow dusting

The method I used to create the "wetness" and "snow dusting" were texture changes. The idea behind the texture changes was merging two textures (the original heightmap texture and the wetness/snow dusting texture) to create this look. The first initial version of this method simply changed the texture of the heightmap as shown below in Figure 16.

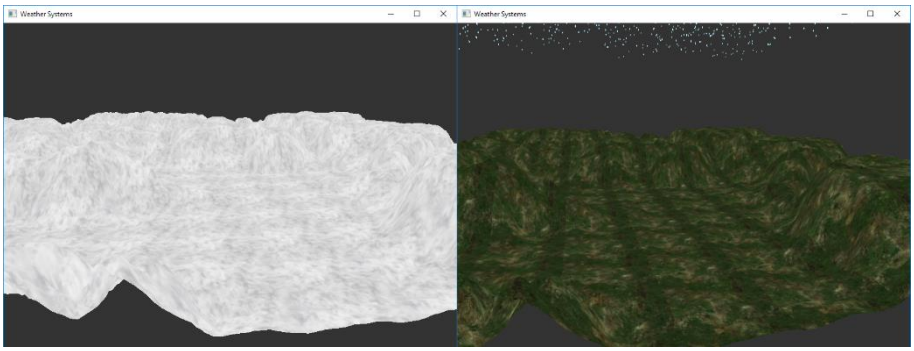


Figure 16: Initial snow dusting/wetness

This didn't look great nor did it feel as accurate as it could be to how it works in real life. Thus, in the second version, I made the textures slowly blend into the new texture and used slopes to create more accurate patches of wetness/snow dusting.

To do the blend factor, I updated the heightmap fragment shader to take in the new blend value. This would then be used to determine how much the heightmap texture and the wetness/snow dusting texture should be blended. The blend value would be then increased over time and the texture would update every frame. This looked better as it was a more gradual change over time rather than it simply changing to the snow/wetness texture as soon as snow/rain occurred.

To create the slopes, I would calculate the normal of the heightmap and then use that value to calculate the slope values for all the vertices. I can then use this slope value in the heightmap fragment shader to determine whether a vertex should be textured with only the heightmap texture or with both the heightmap texture and the wetness/snow dusting texture. Any slopes which were low slopes would use the heightmap texture blended with the wetness/snow dusting texture based on the blend factor and any other slopes higher than this value, would use the heightmap texture. This would simulate how snow/rain typically doesn't stay on the sides of buildings - they would slide down to the bottom. You can see the blend factor and slope-based texturing in figure 17.

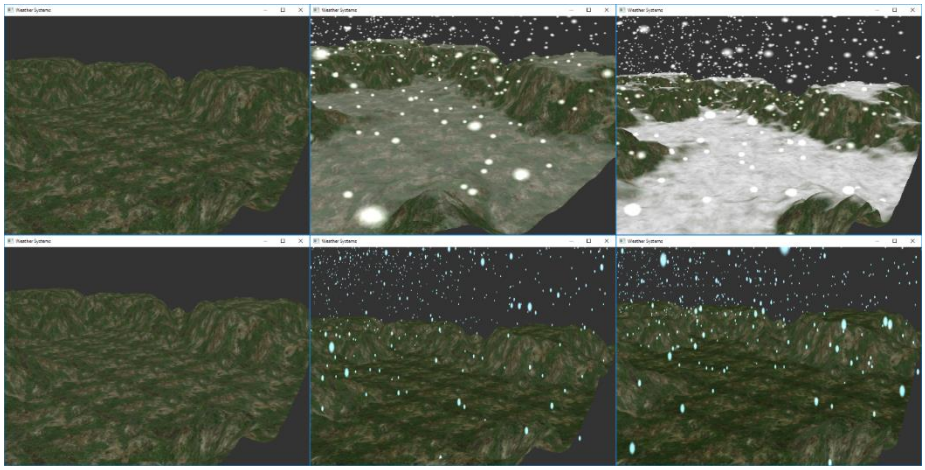


Figure 17: Blend factor and Slope-based texturing

The first picture shows the environment with no texture changes. The second picture shows the environment with a partial texture change. The last picture shows the environment with the full texture change. Top row of pictures is the snow dusting effect and the bottom row of pictures is the wetness effect.

4.4 Snow accumulation

Snow accumulation required two things to happen: the detection of snow landing on the heightmap (collision detection) and the ability to draw snow on the heightmap.

To detect the collisions with the heightmap, I added in a new method in the heightmap to calculate the particle's position on the heightmap. Any time a particle passed a y position of 0, we calculate its position on the heightmap by taking in the x and z coordinates of a particle and converting this into grid position on the heightmap. Using this grid position, we calculate which triangle the particle collided with and from this we know the y value of where the particle collided. This y value can then be returned to the particle so we can place the particle at that y value in the world.

My initial idea for snow accumulation (shown in figure 18) was to check for collision detection when a particle hits the heightmap and where the particle hits the

heightmap, change the values of the particle so that it no longer moves and its lifetime would be changed so it doesn't disappear instantly. This achieved two things, allowed snow to accumulate on the heightmap surfaces and allowed the idea of snow particles eventually decaying (for example if the sun had come out). One thing that you would then have to constantly store is where all particles which aren't moving are (meaning that if you want the snow to grow, it needs to know if snow has landed here before so you can place a new one on top of it). This could lead to an issue where a high amount of memory would be used as time passes by (due to how many particles would be stored).

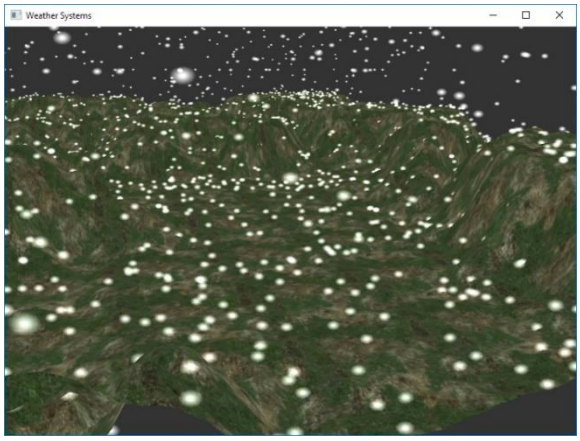


Figure 18: Initial snow accumulation idea

Another idea I had for snow accumulation was to create a thin rectangle underneath the heightmap and have that grow over time. This would have a snow texture and as it grew up over time, the heightmap would start to show this thin rectangle, simulating the snow accumulation effect. This was no good as it lacked a lot of detail and precision in controlling the snow accumulation. However, through the weekly meetings with my supervisors, we had decided on using the vertices in the heightmap and "pulling" them upwards to create the snow accumulation effect. We chose this as this both saved memory (we are using vertices which are already there) and the sheer number of vertices in the heightmap allowed a good level of detail to be completed. With this change, I updated the collision detection method which instead pull the vertices a small amount if the particle had collided instead of returning the y value. Due to the slope based texturing I had implemented earlier, I decided to also add in a check to see if the vertices of that triangle had a low slope and if they did, they would be allowed to be pulled. I decided to add this as, in real life snow doesn't typically stick to walls of a building (high slopes) and would fall to the ground (low slopes).

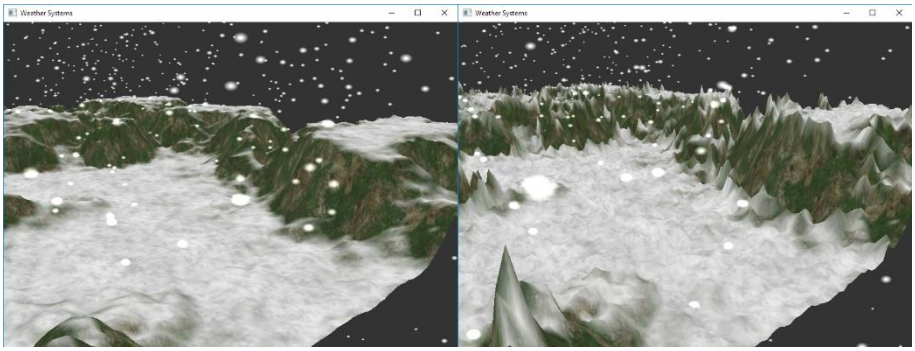


Figure 19: Finished snow accumulation

The finished snow accumulation shown in Figure 19, shows two pictures of snow accumulation: one at the start of snow accumulation and the second after five minutes has passed. As you can tell the snow is slowly increasing but has a very spiky nature. I attempted to reduce this spikiness by instead of pulling one single vertex during a collision, I would instead pull the triangle the particle had collided with. This looked slightly better than previously but still had some spikiness to it.

4.5 Night/day system

The night/day system was created using the skybox technique and changing the texture based on a blend factor. This system used two different cube maps: one for day time and one for night time. Over time the system would update a time variable and we would use this time variable to determine the blend factor for the textures. Below we see the night/day system in three different variations: one at day time, one during the transition to night time from day time and then the last is night time.

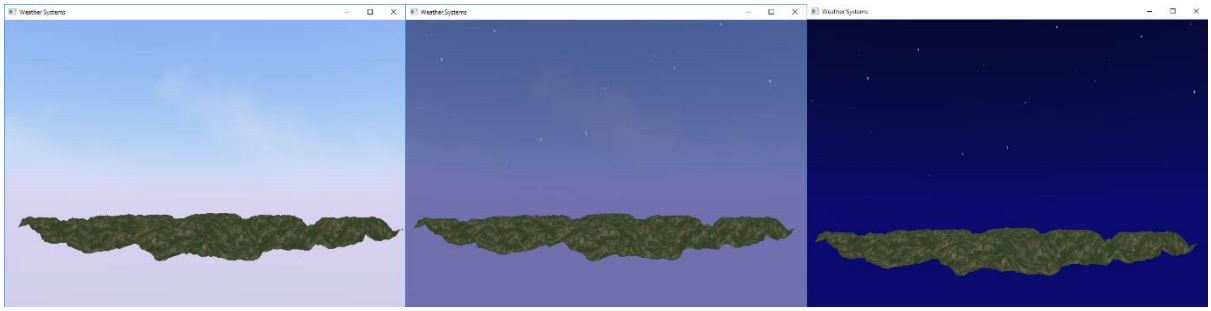


Figure 20: Day/night cycle

Chapter 5 - Testing

5.1 Methodology

The system environment that I use to do my testing is as follows:

- OS: Windows 10,
- CPU: Intel Core i7-5500U,
- GPU: Nvidia GeForce 920M,
- RAM: 8GB.
- SSD: 128GB,
- Visual Studio 2017.

There are two main methods of testing I will be doing: frame time tests and memory usage tests.

Frame time is the time, in milliseconds, needed to draw a frame (averaged over one second). To calculate frame time, I take the number of frames rendered in one second and divide 1000 (this being the number of milliseconds in one second) by the number of frames. I will be testing for frame time of rain and snow averaged over a time period of 5 minutes vs different number of particles.

Memory usage tests will be based on how much memory the program is using. This is done by using Visual Studio's diagnostic tools to check memory usage. I will be testing memory usage for both rain and snow averaged over 5 minutes vs different number of particles.

5.2 Results

Frame time

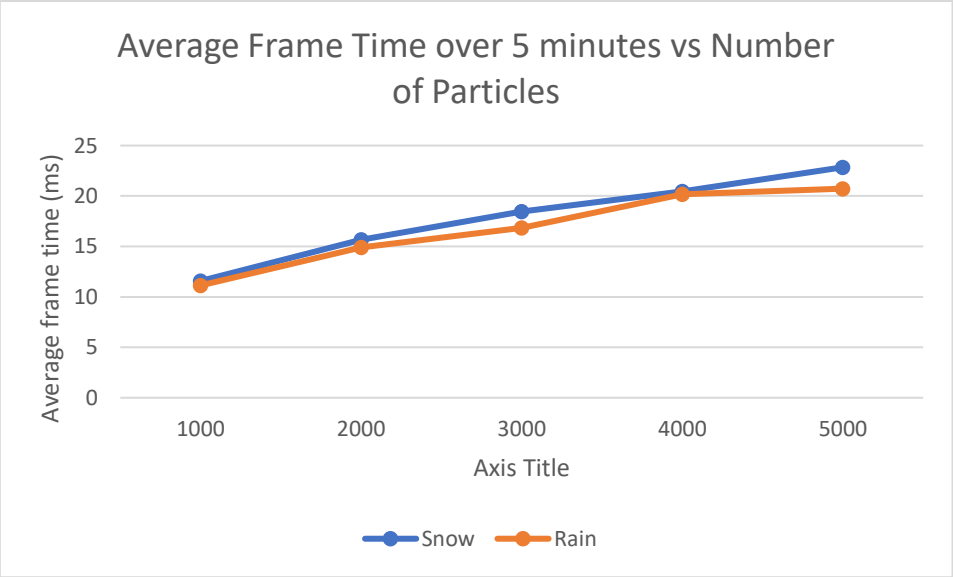


Figure 21: Average Frame Time vs Number of Particles

The average frame time over 5 minutes was calculated by taking the values of the frame time every second until 5 minutes had passed, adding them all up and then dividing that value by the number of pieces of data obtained (in this case 300 pieces of data). A lower frame time is a better value. The relationship it shows is that as the number of particles increases, the average frame time increases.

Memory usage

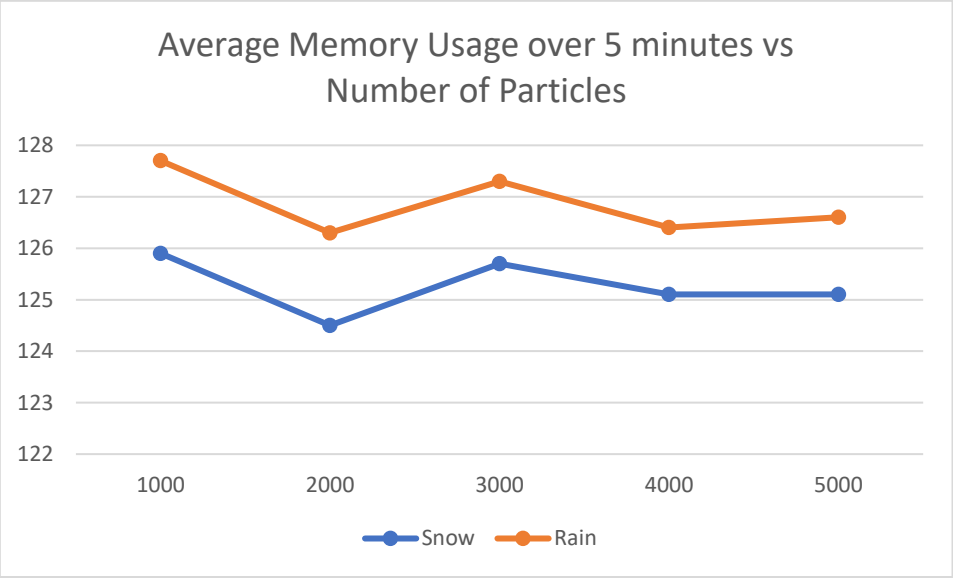


Figure 22: Average Memory Usage vs Number of Particles

The average memory usage was calculated by taking the memory usage at intervals of 1 minute until 5 minutes had passed, adding them up and then dividing that value by the number of pieces of data obtained (in this case 5 pieces of data). A lower memory usage is a better value. This doesn't show any real relationship between the number of particles and memory usage.

Chapter 6 – Evaluation

6.1 Development methodology evaluation

During the course of this project, I had followed the agile development methodology. I believe that following this development methodology was good as it enabled me to try ideas out and see if they worked or not. If they didn't work, I could simply re-iterate on them. Furthermore, it fit well with the weekly meetings I had with my supervisors, enabling them to give advice to me based on the most current work I had done.

Due to the nature of agile development, it meant that I was constantly working on new tasks and I felt at times I was lacking in knowledge in some areas as my plans were not as detailed as it could have been if I were to follow a different development methodology. However, this negative was mostly overcome due to the experience of my supervisors, offering useful guidance to me such as referring to me the idea of doing snow accumulation by pulling vertices up.

Overall, I believe this methodology helped improve the speed of development and keep me on track to steady completion of the project. To improve upon this, I would try to make sure my plans are a bit more detailed to ensure that my development would have more ideas to go on. For example, techniques such as using the compute shader for rendering of particle systems.

6.2 Results evaluation

"The problem with using FPS to measure performance, from a mathematical perspective, is that it is non-linear" (Dunlop, 2003). Dunlop further describes the issue of using FPS to measure performance and recommends using frame time to measure performance due to its linearity. Frame time is more useful as it is linear and tells us how long it takes to render each frame.

Analysing the results from section 5.2, we see that the average frame time increases as the number of particles increases. This shows that it requires more processing power, the more we increase the number of particles. The typical aim for average frame time of a game running at 60 FPS is 16.6 ms. We can see that anything past approximately 2500 particles for both rain and snow, starts to make the game's framerate drop below that value. Both rain and snow have similar average frame times, with snow being slightly higher. This makes sense due to the fact the snow weather also has snow accumulation to do.

Analysing the results from section 5.2, we see that memory usage does not fluctuate very much between data points for both rain and snow. Rain on average has a very minor increase in memory usage over snow. I feel that this increase could be to do with the textures used in the wetness/snow dusting method - the wetness texture is slightly larger than the snow dusting texture. Another reason as to the low/minor changes in memory, for both rain and snow, could be to do with the fact each particle

is simply a point in memory but expanded into a quad (four-point shape) during processing. As we only store one vertex rather than four, this saves a lot of memory required when increasing the number of particles but increases the processing power in return.

Overall, these graphs show me that my program is requires more processing power than it does memory. It also shows me that my limit for smooth, real time gameplay would be about 2500 particles. If I wanted more particles, I would have to find out how to reduce the number of computations occurring in my program. An idea to reduce the number of computations is to look at my collision detection code and see if I could save any execution time there.

6.3 Requirements evaluation

Functional requirements

Here I will talk about the functional requirements which were displayed in table one on page 20. I feel most of the requirements were met but could use improvements.

Requirement one was to display a heightmap. My program shows the user a heightmap which is drawn based off a heightmap file. I feel this was suitably completed as the heightmap is environment I expected to use.

Requirement two was to display a particle system which looks like rain and requirement ten was to be able to choose rain weather. By pressing "2" on the keyboard, a rain particle system would appear on the screen. I believe this requirement has been satisfied. Requirement two I rendered each particle blue with a long and thin oblong shape. I believe this requirement could be fulfilled better as rain is typically more transparent and a better shape could be used.

Requirement three was to display a particle system which looks like snow and requirement nine was to be able to choose snow weather. By pressing "1" on the keyboard, a snow particle system would appear on the screen. I believe this requirement has been satisfied. Requirement three I rendered each particle white with a circle shape. This requirement could be fulfilled better by using a more "snow-like" shape such as a snowflake. The colour seems suitable for this type of weather.

Requirement four was to make the environment texture look wet and requirement five was to make the environment texture look snowier. Both these requirements were satisfied well overall as we blend the textures with the heightmap. However, I feel this could be improved by doing something like Perlin noise based texturing so the blending of the two textures is more than simply what percentage of one texture should we use to colour a pixel. Another idea we could have done is to add another factor to control different levels of wetness/snow dusting depending on how much precipitation there is (like how Final Fantasy XV has differing wetness levels).

Requirement six was to have snow accumulate on the environment. This requirement was fulfilled (snow did indeed grow on the environment) but I feel it could have been satisfied better. The reason why I feel it does not satisfy the requirement well enough is due to the fact that the snow accumulation is very spiky and not smooth. So, the requirement is satisfied but does not look great.

Requirement seven was to show a skybox which looks like day time and requirement eight was to show a skybox which looks like night time. This was suitably completed as I had two skyboxes which cycled between each other depending on the time. This could be further improved by implementing several other states in the time of day such as sunrise or sunset.

Requirement nine was to let the user choose snow weather and requirement ten was to let the user choose rain weather. This requirement was fulfilled as they could press "0" for snow weather and "1" for rain weather.

Requirement eleven was to be able to reset the simulation to the default state. By pressing "3", the system would turn off the particle system but it would not reset the skybox to its initial state. You could also press "0" to reset the wind speed but this would not reset the positions of the particles so some would now be falling outside of the heightmap. Overall, I feel it was satisfied but could use some adjustments such as resetting the skybox as well.

Requirements twelve up to fifteen was to be able to manipulate the wind speeds in x and z directions. By pressing "6" and "7", the user could change the wind speed in the x direction and by pressing "8" and "9", the user could change the wind speed in the z direction. I feel this was satisfied suitably.

Non-functional requirements

Here I will talk about the non-functional requirements which were displayed in table two on page 21.

Requirement one was for the program to run smoothly. Up to approximately 2500 particles of rain or snow, the program runs at the frame rate of 60 or above. Anything past 2500, then the program will start to lag. I believe this could use some work as the number of particles that my program can render without starting to lag is low. This was satisfied somewhat well.

Requirement two was for the program to run efficiently. The memory usage of the program is efficient and I can't say for certain whether the processing is as efficient as it could be. I feel that the collision detections could possibly be improved for example. This was satisfied somewhat well.

Requirement three was for the program not to crash. The program only crashes on exit which leads me to believe that something is being deleted incorrectly. This wasn't satisfied if we look at the program crashing on exit but other than that, the program does not crash during run time.

Requirement four was for the application to run on different machines. This was not tested so I can't say how much it has been satisfied.

6.4 Subjective evaluation

Overall, I feel the weather system is suitable and achieves my aim of producing one which affects a world. However, as mentioned previously, the rain and snow weather both particles could look better to help aid the visual looks of my weather system. The snow accumulation is unrealistic and definitely could use more work. I would have liked the snow accumulation to not look so spiky and grow more naturally. I am happy with how the wetness looks but I feel it could be further improved. I implemented it by creating a wetness texture and blending it with the normal texture. An interesting idea might be to create "wet" textures in real time programmatically. Snow dusting could look better too with a better blend function (like Perlin noise texturing). Day/night cycle looked good overall but perhaps could use more states.

Chapter 7 – Conclusion

7.1 Review aims and objectives

Objectives:

1. To research into weather conditions in the real world, existing systems and the various technologies behind graphical rendering of weather systems.

I believe I completed this well but perhaps more research could have been done. For example, I only researched snow and rain weather effects and I could have considered other weather effects (like fog or dust storms). For existing systems, I had considered two games and perhaps could have looked at more. Then described the various technologies behind graphical rendering of weather systems but I had focussed more on the methods than the technology used. I could have perhaps talked about how shaders worked and the other methods other than geometry shader for particle systems rendering.

2. To design the graphical environment and weather conditions based off the research I have done.

The design of the graphical environment was minimal as it was created using a heightmap file which I did not design myself (taken from NCLGL tutorials). However, the graphical environment created by the heightmap was more than suitable for showcasing the attributes I wanted to show for my weather system. The weather designs were also fulfilled well, in part due to the research I had done earlier. Overall, I am satisfied with the fulfilment of this objective.

3. To develop graphical environment and weather system using the designs I have created.

I feel this was suitably fulfilled. The initial implementation of the environment and weather system went well. I followed the designs of both as accurately as I could. Furthermore, I used the techniques I had described in my research step and used OpenGL. This was in part thanks to the NCLGL library, making it easier to use OpenGL.

4. To implement how my weather system will affect the interactive world.

After creating the environment and weather system, I then could add in interactions with the environment such as snow accumulation and wetness/snow dusting. This was achieved well as I followed the attributes I had set out earlier in the design stage. I feel that this could have been achieved better however. For example, Final Fantasy XV uses a weather system which links “weather” with game entities by a set of parameters (Elcott et al., 2016). This could be a better idea than simply programming it to affect just the environment. Despite this being a better method to take, I did fulfil the objective requirements.

5. To evaluate the performance of my weather system and whether the requirements have been satisfied.

I fulfilled this objective as I looked at the performance of my weather system by looking at frame time and memory usage. Then, I checked if the weather system fulfilled the requirements set out in design stage and subjectively looked at the weather system overall to see if I felt like I had set out what I intended to do.

Aim: To implement a dynamic weather system which affects an interactive world in a realistic manner and to determine its performance in real time.

Due to completing all the objectives, I feel that I have indeed fulfilled the aim of my project. However, I feel there could be many improvements as talked about previously.

7.2 Personal development

During the project, I have learned a great deal about various techniques surrounding the graphical rendering of weather systems. I also have learned how to use the agile methodology. Furthermore, I have learned various skills through the completion of my dissertation such as background research and technical writing. By doing this project, it helped further my understanding of OpenGL and 3D graphics development.

In the project, I feel one of the things that went well was the development of my program. Meeting my supervisors weekly and building upon my program through several discussions made sure that my development went smoothly. Initially, my understanding of OpenGL and 3D graphics technology was weak but through the course of my project, I feel my understanding improved a lot more.

7.3 Future work

If the project could be redone, I feel I would like to have performed the research better, consider other areas of weather systems and improve upon the features I currently have. To improve the research, I would have liked to find more academic sources regarding weather systems and researched other graphic techniques that could be used in weather systems such as Perlin Noise texturing. A feature I'd like to improve upon would be snow accumulation. As stated previously, the snow accumulation looks very spiky. I would like to make it look closer to the real world and less spiky.

Future work of my environment could include other entities such as buildings or trees. This could further showcase interaction with the world as I have other game objects that could get wet or have snow accumulate on it.

The weather system could include other weather types such as hail, fog or dust storms. This would allow simulation of a larger variety of weather types and the opportunity to add different interaction with the environment based on these new weather types. For example, you could make it harder to see things for the player if fog occurs.

Other interactions I could have added for rain and snow could be: deformable snow, splash effect and puddles. Deformable snow would show player footprints when a player walks onto snow, splash effect would occur when rain particles collide with a surface and rain puddles would generate if the terrain surfaces are suitable (e.g. small holes in the ground).

The day/night system could include other times of day such as sunrise or sunset. This too could interact with the environment if I include a lighting system - changing the lighting based on the time of day. Furthermore, I could add in an interaction such as snow melting when it is day time and sunny.

Following on what has previously said, I could add in a lighting system for an object such as the sun/moon which would change based on time of day as previously said. This could also interact with other parts of the weather system such as specular reflection in puddles.

Another area of weather systems that I could have considered was clouds. Clouds are useful towards making a weather system interact with the world (they up a large amount of the sky). For example, how Horizon Zero Dawn makes clouds look denser when precipitation occurs.

Another method that could be done in future versions is using the compute shader to help accelerate game rendering. Using the compute shader would help free my CPU to do game code (Thomas, 2014).

7.4 Final thoughts

Looking back on the project, I have learned a lot of different things - background research, graphical techniques and technical writing. From a personal perspective, it has enabled me to understand just how impressive work in current games truly is. I am pleased overall with the result of my work. There are many things I would have liked to add and I can see that there are many improvements that could be made. Despite the flaws, it was a thoroughly enjoyable project to partake in.

References

Barton, M. (2008). *How's the Weather: Simulating Weather in Virtual Environments*. International Journal of Computer Game Research, 8(1).

Davison, R. (2012). *Tutorial 8: Index Buffers & Face Culling*. [online] Available at: https://research.ncl.ac.uk/game/mastersdegree/graphicsforgames/indexbuffers/Tutorial%208%20-%20Index%20Buffers.pdf [Accessed 1 May 2018].

Developer.valvesoftware.com. (n.d.). *Skybox Basics - Valve Developer Community*. [online] Available at: https://developer.valvesoftware.com/wiki/Skybox_Basics [Accessed 2 May 2018].

Dunlop, R. (2003). *FPS Versus Frame Time*. [online] Mvps.org. Available at: https://www.mvps.org/directx/articles/fps_versus_frame_time.htm [Accessed 4 May 2018].

Elcott, S., Chang, K., Miyamoto, M. and Metaaphanon, N. (2016). *Rendering techniques of Final Fantasy XV*. ACM SIGGRAPH 2016 Talks on - SIGGRAPH '16, [online] pp.71-74. Available at: http://www.jp.square-enix.com/tech/library/pdf/s16_final.pdf [Accessed 30 Nov. 2017].

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

Group Inc, T. (n.d.). *OpenGL Overview.* [online] Opengl.org. Available at: https://www.opengl.org/about/ [Accessed 1 May 2018].

Hockenhull, S. (2017). *Why would a game developer choose Direct3D instead of something with more cross-platform support?*. [online] Gamedev.stackexchange.com. Available at: https://gamedev.stackexchange.com/questions/138200/why-would-a-game-developer-choose-direct3d-instead-of-something-with-more-cross [Accessed 1 May 2018].

Hope, C. (2017). *What is DirectX?*. [online] Computerhope.com. Available at: https://www.computerhope.com/jargon/d/directx.htm [Accessed 1 May 2018].

Lagarde, S. (2012). *Observe rainy world*. [Blog] Sébastien Lagarde | Random thoughts about graphics in games. Available at: https://seblagarde.wordpress.com/2012/12/10/observe-rainy-world/ [Accessed 30 Nov. 2017].

Latta, L. (2004). *Building a Million Particle System*. [ebook] Available at: https://www.gdcvault.com/play/1022856/Building-a-Million-Particle [Accessed 28 Nov. 2017].

Meiri, E. (n.d.). *Skybox.* [image] Available at: http://ogldev.atspace.co.uk/www/tutorial25/skybox.jpg [Accessed 2 May 2018].

Reeves, W. (1983). *Particle Systems---a Technique for Modeling a Class of Fuzzy Objects. ACM Transactions on Graphics*, 2(2), pp.91-108.

Stackoverflow.com. (2010). *Why is c++ that powerful concerning game development?*. [online] Available at: https://stackoverflow.com/questions/3318898/why-is-c-that-powerful-concerning-game-development [Accessed 1 May 2018].

Schneider, A. (2015). *The Real-time Volumetric Cloudscapes of Horizon: Zero Dawn*. [online] Available at: http://killzone.dl.playstation.net/killzone/horizonzerodawn/presentations/Siggraph15_Schneider_Real-Time_Volumetric_Cloudscapes_of_Horizon_Zero_Dawn.pdf [Accessed 2 May 2018].

Techopedia.com. (n.d.). *What is the C++ Programming Language? - Definition from Techopedia*. [online] Available at: https://www.techopedia.com/definition/26184/c-programming-language [Accessed 1 May 2018].

Thomas, G. (2014). *Compute-based GPU Particle Systems*. [ebook] p.3. Available at: http://twvideo01.ubm-us.net/o1/vault/GDC2014/Presentations/Gareth_Thomas_Compute-based_GPU_Particle.pdf [Accessed 4 May 2018].

Wang, L., Lin, Z., Fang, T., Yang, X., Yu, X. and Kang, S. (2006*). Real-time rendering of realistic rain.* ACM SIGGRAPH 2006 Sketches on - SIGGRAPH '06.