Contents

[Abstract 3](#_Toc6321676)

[Chapter 1 – Introduction 4](#_Toc6321677)

[1.0 – Project Concept and Justifications 4](#_Toc6321678)

[1.1 – Aims and Objectives 4](#_Toc6321679)

[1.1.1 – Aims 4](#_Toc6321680)

[1.1.2 – Objectives 4](#_Toc6321681)

[1.1.3 – Additional Features and Objectives 5](#_Toc6321682)

[Chapter 2 – Literature Review 6](#_Toc6321683)

[2.0 – Introduction 6](#_Toc6321684)

[2.1 – Real World Weather 6](#_Toc6321685)

[2.1.1 – Snowflake Shape 6](#_Toc6321686)

[2.2 – Particle Based Systems 7](#_Toc6321687)

[2.3 – Scrolling Texture System 7](#_Toc6321688)

[2.4 – GPU Computation 7](#_Toc6321689)

[2.5 – Snow Accumulation 7](#_Toc6321690)

[2.6 – Summary 8](#_Toc6321691)

[Chapter 3 – Methodology and Process 9](#_Toc6321692)

[3.0 – Overall Design and Implementation 9](#_Toc6321693)

[3.1 – System Design 9](#_Toc6321694)

[3.2 – System Implementation 9](#_Toc6321695)

[3.2.1 – OpenGL Abstraction 10](#_Toc6321696)

[3.2.2 – Particle Systems 10](#_Toc6321697)

[3.2.3 – Collision Detection 12](#_Toc6321698)

[3.2.4 – Accumulation System 12](#_Toc6321699)

[3.2.5 – Atmospheric Effects 13](#_Toc6321700)

[3.2.6 – GUI 13](#_Toc6321701)

[3.3 – Testing and Evaluation 13](#_Toc6321702)

[Chapter 4 – Results and Reflection 14](#_Toc6321703)

[4.0 – Realism 14](#_Toc6321704)

[4.1 – Performance 14](#_Toc6321705)

[4.2 – Summary 14](#_Toc6321706)

[Chapter 5 – Conclusion and Future Work 15](#_Toc6321707)

[5.0 – Future Work 15](#_Toc6321708)

[5.1 – Conclusion 15](#_Toc6321709)

[References 16](#_Toc6321710)

[Appendices 17](#_Toc6321711)

# Abstract

When it comes to the creation of a believable 3D environment, visual effects and the physically based simulation of artefacts in that space play a very important role in immersing the viewer. The focal point of this project will be implementation of a snow simulation demo, replicating the dispersion of snow over a space under the influence of diverse environmental effects. Previous implementations and methods will be considered such as particle systems, screen-space solutions and occlusion-based snow build-up. Accumulation of snow on the surfaces of objects in the scene will be developed with a focus on physical realism while maintaining high performance in calculation and rendering through parallel processing. As maintaining real time processing speeds is essential to game programming, performance will be profiled in comparison to various scene implementations and layouts with varying levels of detail and realism.

The software is developed in C++ and OpenGL 4 with the target platform being Windows. A demonstration program will be produced to allow users to view various effects on the scene through a user interface.

# Chapter 1 – Introduction

## 1.0 – Project Concept and Justifications

In its current state, snow simulation in real time (60fps) is primarily based on occlusion implementations which leverage similar techniques as used in basic shadow mapping. Visual fidelity and realism play a huge role in in the immersion of a scene and snow can completely alter the scape of an environment. As the common hardware available to end-users improves, more performance intensive solutions can be considered viable and this project aims to demonstrate a more physics-based approach now possible due to advancements in hardware and graphics APIs.

This occlusion technique typically seen in recent and older implementations separates the two primary factors of a snow simulation: the snowfall and snow accumulation. This project intended to combine the two, linking the snow particles to the build-up of snow on a surface.

This project will be developed in C++ and supports a minimum of OpenGL 4.3. C++ was chosen due to is performance capabilities. OpenGL was selected as it is a widely known with an extensive feature set and being a higher-level graphics API, development time is reasonable, fitting with the duration of this project. Newer features in OpenGL such as Shader Storage Buffer Objects will be essential to this project.

## 1.1 – Aims and Objectives

The project follows three aims to be completed. These aims focus on the core elements of the project to ensure on the deadline that the primary features are finished.

### 1.1.1 – Aims

1. The primary aim of the project is to develop an implementation of a physics-based particle snow accumulation system.
2. The secondary aim is to develop a demonstration scene which shows the system working with visual effects to improve fidelity.
3. The tertiary aim is to allow the user to modify the settings allowing for different snow and environmental settings.

### 1.1.2 – Objectives

The focal objectives of the project follow the first aim to create the snowfall and accumulation system. These features will include:

* Snowfall and Accumulation
  + Transform feedback system for updating and rendering particles
  + Collision detection system to detect when a particle collides with an object in the scene
  + Spatial partition system for storing accumulated snow values in differing parts of the scene
* Demo Scene
  + Create several objects which can be added to the scene. This includes collision meshes and texturing.
  + Implement visual effects such as shadow mapping to improve visual fidelity.
  + Create a system for importing .obj files from the CPU side to the GPU.
  + Implement a lite abstraction of OpenGL to speed up development.
* Interaction
  + Integrate imGUI into the project to shorten development time
  + Link the GUI into the particle system to allow modification. This will allow the user to modify the scene to show how the dynamic particles behave.
  + Provide performance metrics to the user to allow for comparison of different settings. This should at a minimum provide counts for FPS, Delta Time, memory usage, GPU usage and VRAM usage.

### 1.1.3 – Additional Features and Objectives

If time permits, extra features will be considered for implementation into the project. These features are not core to the primary functionality but would improve the experience and provide further polish to the finished item.

* Real-time optimisation
  + Managing the scene and snow detail in real-time to ensure performance is not affected to a level which interrupts the user experience.
* Additional Lighting Effects
  + Implementation of OpenGL features such as Bloom, Light Scattering and HDR.
* Wind Map
  + Particles can be affected by wind and a pre-processed wind-map would allow for further realism in how the snow particles flow through the scene. Default behaviour would include a global value for wind which would only provide a uniform adjustment to all particles in the scene.

# Chapter 2 – Literature Review

## 2.0 – Introduction

Recent academic investigation into fully featured real-time snow simulations are far between but many examples of the different elements of the simulation exist such as the particle simulations presented by White (2014) and Lv & Liu (2013). Scrolling texture approaches will also be considered such as that shown by Wang & Wade (2004).

There are many different snow accumulation techniques for 3D environments, including but not limited to Occlusion, depth mapping and surface exposure calculation. These methods consist of a mixture of both online and offline solutions with one of the most well known being the implementation by Fearing (2000). This method is an offline render which produced realistic results but ran on now outdated hardware. Foldes & Benes (2007) present an occlusion based implementation which also simulates snow melt on a large scale.

## 2.1 – Real World Weather

When designing a simulation following a real-world happening, considerations must be taken to how the real phenomenon occurs both visually and what causes it to behave the way it does.

### 2.1.1 – Snowflake Shape

Koh (1989) presents a classification for types of snowflake and the temperatures at which they form. The available paper is unfortunetly of low viual quality and therefore it can be difficult to distinguish the shapes of photographed individual flakes. Table one below shows a selection of the types of snowflake seen in real life:

|  |  |  |
| --- | --- | --- |
| ID | Flake Shape | Name |
| 1 |  | Stellar crystal with sectorlike ends |
| 2 |  | Combination of needles |
| 3 |  | Plate with dendritic extensions |
| 4 |  | Hexagonal Plate |
| 5 |  | Minute Column |
| 6 |  | Plate with simple extensions |
| 7 |  | Ordinary dendritic crystal |
| 8 |  | Pyramid |
| 9 |  | Scroll |

Table 1 - Real world snowflake types (Koh, 1989)

The table presented in Koh’s paper is based on the classification system first put forward by Choji & Chung Woo (1966).

Koh states that snowflakes grow according to ambient temperature until they become large enough to fall in the form of precipitation. Assuming ambient temperature sits below 0°C, the precipitation will fall in the form of snow. Through collision and other forms of breakup, snow particles many vary further in shape.

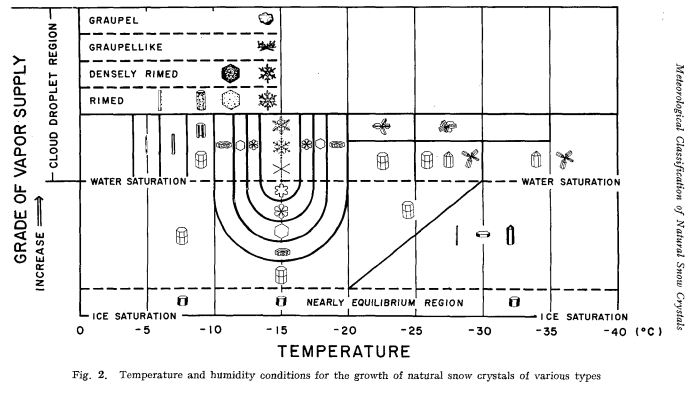


Figure 1 - A diagram showing the different snowflakes that form depending on ambient temperature and conditions (Magono & Lee, 1966, p. 327)

Figure 1 shows how as ambient temperature differs, the primary shape of falling snow particles changes. The plane shape is the most common, generally forming at temperatures of around -15°C. More obscure shapes form at lower temperatures of -30°C and below.

## 2.2 – Particle Based Systems

White (2014) presents a particle based system that covers rain and snow. The author discusses the primary differences between a CPU and GPU based approach to updating and drawing each particle with advantages and disadvantages for each. A GPU implementation is chosen for the final design for the sake of the significantly improved performance characteristics when compared to the CPU. The final results include a blended texture for each particle including transparency and while the visuals are satisfactory the simulation fails to provide complex mesh collision detection, replying on bounding boxes or bounding spheres.

Lv & Liu (2013) proposes a system of simulating particles while also covering the separate visual effect that heavy snowfall causes which is to apply a fog and blur when viewing an object at a distance through the particles. The author also discusses the implementation of a wind field to apply a gustiness to the environment, giving the particles a non-uniform movement path which is more accurate to the real life effects.

## 2.3 – Scrolling Texture System

## 2.4 – GPU Computation

## 2.5 – Snow Accumulation

Fearing (2000) describes a method of snow accumulation for offline rendering. This implementation resulted in realistic results including dusting, flutter and wind-blown snow but at the time of design, the hardware could not calculate and render these effects in real time. The primary difference with this model as opposed to others was the technique of firing particles from the surface towards the sky as opposed to the direction that would be expected.

Ohlsson & Seipel (2004) present a method of rendering accumulated snow on complex 3D models through use of a snow prediction function. This function takes into account factors such as the exposure a position has to the environment and the surface inclination. This method has the advantage of yielding realistic results for simple scenes with a singular direction of snowfall but falls short when considering the physics of how snow falls throughout a 3D space. This method also does not run at a stable 60 frames per second on the hardware used at the time of testing.

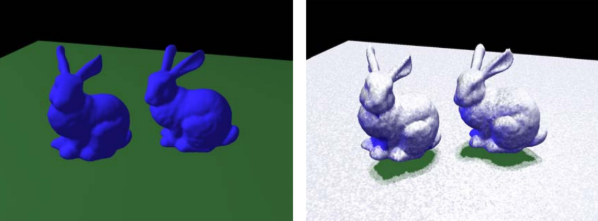


Figure 2 - The Ohlsson and Seipel (2004) method of rendering accumulated snow

Haglund, et al. (2002) explain a real-time implementation based on a 2D matrix which stores data on snow depth over areas in which snow can fall. This method has the advantage of being scalable to the hardware which is running the simulation through increasing the density and resolution but has limitations in conplex meshes. This implementation combines the simulation of particles with the accumulation though detecting collisions with the surface and applying a displacement to the mesh at the closest vertex to the point of collision. The simulation yields satisfactory results on large, flat surfaces but is not suited to more intricate scenes.

## 2.6 – Summary

The methods reviewed do not cover the full range of effects that this project wishes to encompass but provides a effective look into each element of the simulation process.

# Chapter 3 – Methodology and Process

## 3.0 – Overall Design and Implementation

The software design and implementation method that was followed throughout development was based on the Agile development cycle. This method proved successful when dealing with complicated systems, implementing and testing features iteratively to improve operation and simulation quality. Differing methods of implementation were considered with their advantages and disadvantages analysed for each primary and secondly feature to ensure the final project was created to a satisfactory level.

## 3.1 – System Design

## 3.2 – System Implementation

The simulation system can be brown down into a number of key stages. These 4 stages are shown in figure 1 below as:

* System 1 – Shadow Depth Render
  + This step is not essential to the simulation but provides further visual fidelity to the scene.
* System 2 – Collision Geometry Feedback (see 3.2.3)
  + The collision geometry feedback stage loads all the world-space geometry into a buffer from the vertex shader through transform feedback.
* System 3 – Object Rendering (see 3.2.4)
  + The object rendering state takes the data from the spatial partition which holds all the accumulation data and uses the values for the current vertex position to calculate how much to offset the vertex by.
* System 4 – Particle Simulation (see 3.2.2)
  + The particle simulation step calculates the new position for each particle and detects collisions with the world space geometry from system 2. On a collision, a value is added to the accumulation spatial partition in the bin which surrounds the collision position.

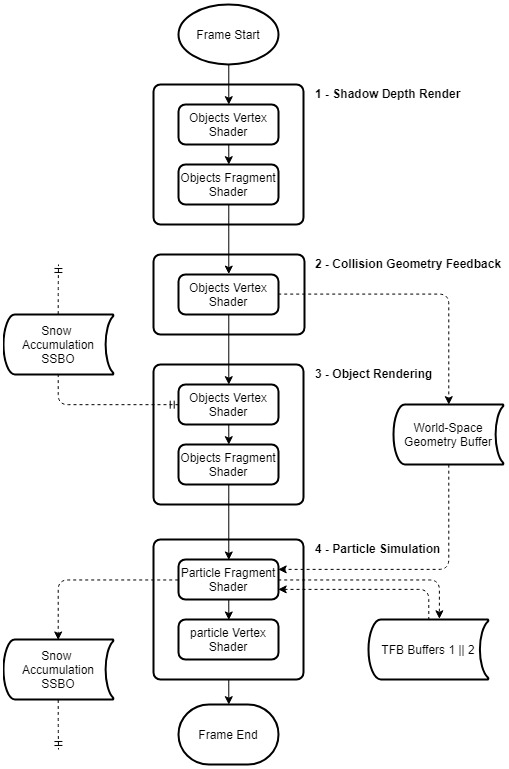


Figure 3 - Simulation Flowchart

### 3.2.1 – OpenGL Abstraction

Initial implementation of the project focused on the abstraction of the core OpenGL functionality. This covered the basics of Vertex Array/Object Buffers, Shaders and Textures. Having a solid foundation of abstraction allowed for development ahead of this stage to be streamlined and considerably quicker as opposed to using individual OpenGL calls.

### 3.2.2 – Particle Systems

From the planning stages of the project, it was evident that CPU-side particles would not give the performance required for the simulation. Two common methods of simulating particles on the GPU exist, being through Transform Feedback or via a Compute Shader. The former was chosen because of a simpler structure to fit within time constraints but a Compute Shader implementation should be considered for future work.

#### 3.2.2.1 – Transform Feedback

The transform feedback approach utilises two buffers for the simulation and rendering of particles. These buffers are interchanged with every update tick of the particle system with the primary active buffer covering the updating of particle positions and other attributes while the second handles the rendering. This is commonly known as “ping-ponging” and is a method of circumventing the issues which arise from reading and writing from a buffer in the same frame.

Due to the parallel nature of GPU computing, this provides a significant increase in performance over a CPU-side implementation.

#### 3.2.2.2 – Geometry Shader Billboarding

In order to move away from OpenGL point rendering and give the particles a defined shape, a method called billboarding was used to render a 2D textured quad in an orientation which always faces the active camera. This is a performance friendly method of drawing hundreds of thousands of snow particles in the scene.

In order to further accelerate the drawing of these particle quads, a geometry shader was created to generate the mesh from a single particle. As can be seen in Figure 2, the generated vertices are placed around the particle’s position to form a quad which is output from the geometry shader as a triangle strip.

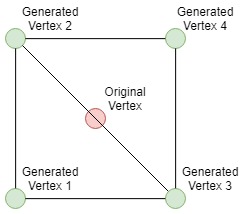


Figure 4 - Geometry Shader Output

The geometry shader also handles the orientation of the particle which as mentioned will follow the cameras rotation to always be facing it. This is achieved through expanding the point into a quad in view space, allowing for a simple translation using 2D vectors for each generated vertex’s correct position. This move to view space can be done in the vertex shader by multiplying the vertex position by the model and view matrices. Once the generated vertex is in the correct position in view space, it is then multiplied by the projection matrix to draw it correctly for the active camera. When it is then passed onto the fragment shader, a snowflake texture is applied to the quad including the transparency alpha channel.

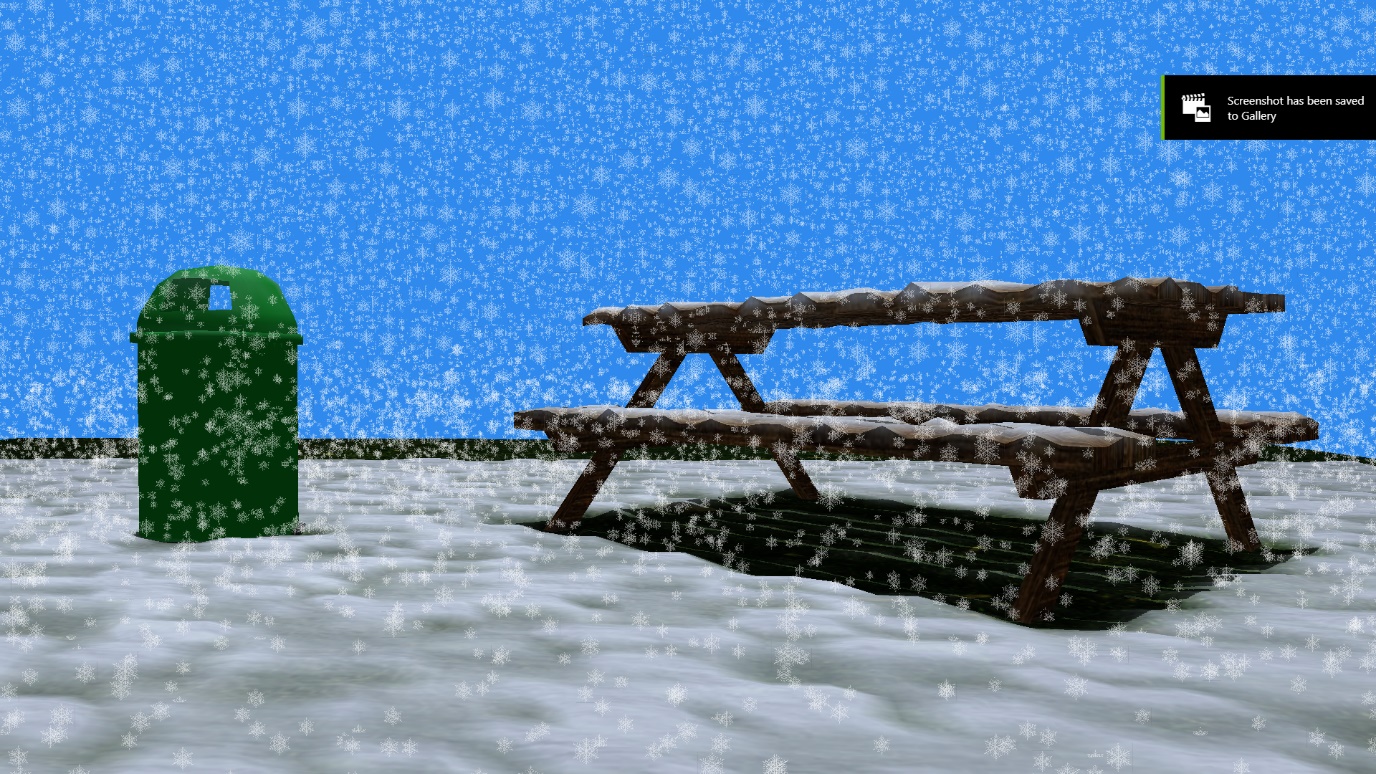


Figure 5 - Billboarded snowflake particles

### 3.2.3 – Collision Detection

Because of the physics-based nature of the simulation, a robust and accurate collision detection system was required. It became quickly apparent that having particles and mesh data on the GPU meant that the collision detection had to also stay GPU-side to avoid bottlenecks between the CPU and GPU. A similar solution to that described in The OpenGL Programming Guide (Kessenich, et al., 2017) was implemented using Transform Feedback.

This system worked through capturing the world space geometry calculated when drawing a mesh through OpenGL transform feedback. The geometry was stored in a Texture Buffer, formatted as RGBA (XYZW) with each texel representing a vertex and 3 sequential texels representing a triangle which could be tested for collision.

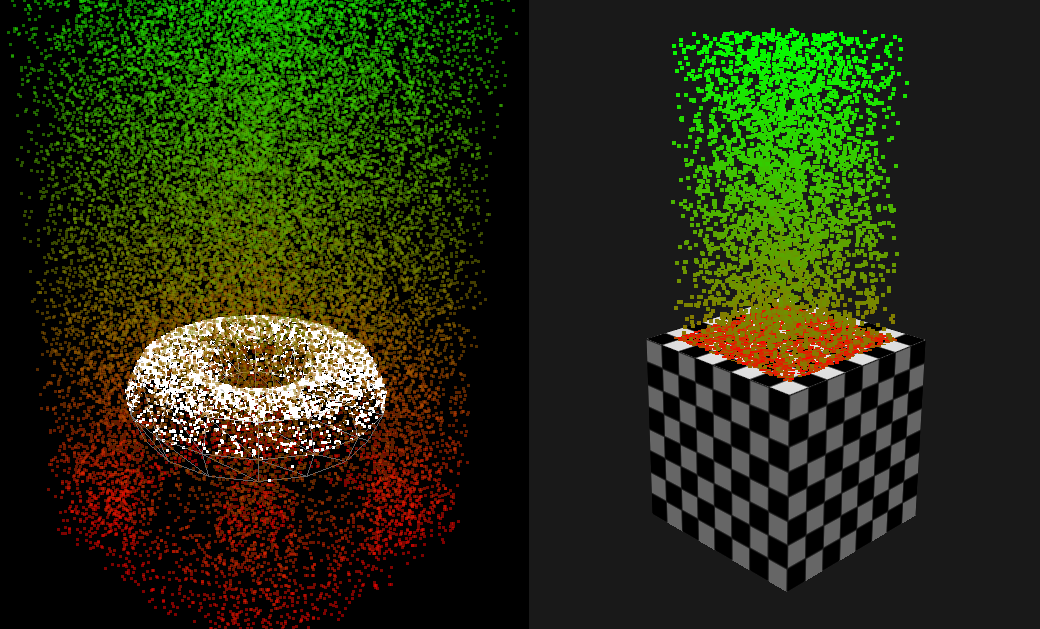


Figure 6 - Early stage collision tests

This world space geometry could then be used by the particle simulation to detect collisions by looping through the texture buffer, reading every third vertex and taking the following two vertices to form a triangle. A line segment would then be formed between the particle’s original position and its newly calculated position to test against the triangle. This method of collision detection can be seen shown in Figure 6.

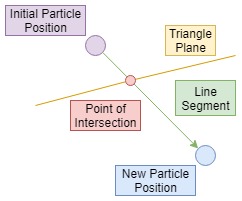


Figure 7 - Collision Detection Method

Each particle tests for collisions against the world-space geometry in parallel on the GPU, significantly improving performance over a typical single line of execution. This method also allows for all other stages of the simulation to remain on the graphics card and not involve moving any data backwards and forward from the GPU to the CPU.

### 3.2.4 – Accumulation System

The accumulation system is based around a Shader Storage Buffer Object which is accessible from both the rendering shader and the particle system shader.

### 3.2.5 – Atmospheric Effects

### 3.2.6 – GUI

## 3.3 – Testing and Evaluation

# Chapter 4 – Results and Reflection

## 4.0 – Realism

## 4.1 – Performance

## 4.2 – Summary

# Chapter 5 – Conclusion and Future Work

## 5.0 – Future Work

## 5.1 – Conclusion

# References

# Appendices