Jenna Toivonen

Creating a dynamic snow shader asset with shader-player interactions

Industry pipelines in shader production

Bachelor's thesis

Bachelor of Culture and Arts

Degree programme in Game Design

2024

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Culture and Arts
Author	Jenna Toivonen
Thesis title	Creating a dynamic snow shader asset with shader-player interactions: Industry pipelines in shader production
Commissioned by	-
Year	2024
Pages	69 pages, 15 pages of appendices
Supervisor	Panu Vuoristo

ABSTRACT

The objective of the thesis was to construct an interactive snow shader that demonstrates shader-player relationships in a video game environment. Relevant literature and research were used to gain a theoretical and practical understanding of shader production and to ensure the successful execution of the practical project. The information acquired was applied to the production of the snow shader asset.

Two qualitative methods were utilised to ensure the shader asset's quality compared to industry standards as well as existing snow systems in games. The author interviewed individuals who have experience with shaders to gain insights into shader production. The interviewees were asked about their opinions and personal experiences regarding various shader-related topics. Additionally, a comparative analysis was conducted to study the differences in visual aspects and gameplay impacts of snow in various published games. The author's snow shader was part of the analysis to determine whether the shader system is sufficient when compared to other snow systems in games.

The study showed that shader production revolves around balancing limitations, requirements and priorities. Therefore, no standardised industry pipelines regarding shaders exist at this moment. However, there are common practices to ensure a successful creation of a shader. It turned out that the author's snow shader compared well to other snow systems seen in published titles. As shader production requires well-defined priorities it was safe to assume that in each project the wanted features are chosen based on the desired gameplay feel, priorities and platform or project-related limitations.

Keywords: shaders, shader interactivity, game development, video game

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

APPENDICES

1 INTRODUCTION

In the field of computer graphics, shaders are responsible for how objects are rendered on the screen. In video game development, shaders are one of the essential tools a developer can use to create immersive visuals. Shaders are not only powerful pieces of code that control how the lighting interacts with surfaces but also a way to create interesting effects. They can direct an art style with precise control over how the game is rendered.

Where shaders most commonly describe surfaces, interactive shaders can create dynamic visuals that react to other game components. A few examples are grass, sand, dirt, snow or wind moving the leaves of a tree. Interactive shaders play a key part in achieving interesting and impactful visuals which are in direct contact with the player while providing a grounded and authentic experience in the game world. However, these shaders are not relevant for all games due to either design decisions, or stylistic and platform related requirements.

This thesis aims to study the interactions between shaders and players in the context of a video game world and define how these relationships are produced utilising an industry-relevant development pipeline. The thesis features background research on the theoretical side of shader production as well as learning the practicalities of developing shaders in a game engine. The study features qualitative research conducted in parallel with producing a shader asset which in the end demonstrates the applied research outcomes.

The topic of the study was chosen from the author’s interest in shader interactivity and improving the authenticity of game worlds and gameplay feel. Pre-existing research on shaders is widely available but academic literature regarding shader-player interactions was not found while writing this paper. Additionally, practical information is often difficult to find in a consumable form as shaders can quickly become very complex. Therefore, it became relevant to study how shader-player interactions are constructed and define the standard processes of shader production in a more practical matter. The paper aims to

provide the reader with a comprehensive overview of the basics of shader creation. This study can work as a valuable tool in understanding the basic principles of shaders as well as common practices in shader production.

The thesis does not have a commissioner but rather works as a valuable personal learning experience for the author, getting them closer to working with shaders and visual effects. The project is planned and produced by the author who makes all the creative decisions for the shader system. The responsibilities of the author include careful planning and scheduling of both the practical project and conducting the research for the thesis as well as showcasing the outcome.

2 RESEARCH DESIGN

The thesis process aside from the paper includes producing a practical project in the form of a snow shader which demonstrates an established shader-player relationship. To study the topic of shader interactivity, qualitative research was conducted. It was utilised to better understand practical applications and processes. The results of the conducted research are applied to the practical project, therefore, ensuring the final shader asset matches the qualitative requirements of the game industry.

2.1 Objective

The outcome of completing this thesis is a functional and visually appealing dynamic snow shader made inside the Godot game engine. The shader should demonstrate an understanding of the interactive shader workflows and the usage of appropriate techniques relevant to the current industry pipelines. The final shader asset displaces the surface of the snow according to the player's movement therefore establishing an interaction. Aside from the player leaving tracks in the snow, the snow also sticks to the player's body and slows their movement while establishing a mutual relationship that affects both components simultaneously. Therefore, the practical project supports the study of defining how a shader and a player can affect one another in a game world and how it relates to immersion and the player's impact on the world.

The main objective of the thesis is to produce a viable snow shader asset that demonstrates shader-player interactions. While working on the practical implementation of this thesis, the author aims to acquire information on production methods and shader theory. This aids in successfully constructing a functional and visually appealing shader. Through research and working on the shader asset the author aims to answer the question: “How are dynamic shader-player interactions created in a game production environment?”. Additionally, the author studies industry-relevant pipelines and practices in making game-ready shaders. Therefore, some sub-questions were formed, which the author also aims to address. The said questions are: “What are the industry standard practices for the production of shaders?” and “What actions must the developer take to achieve professional and visually appealing results?”.

To form a foundation for the study, a considerable amount of background research is conducted. The information acquisition is divided into two sections: seeking theoretical knowledge and learning practical applications. The acquisition of theoretical information consists of reading relevant literature and studying the technical aspects of shaders such as shader rendering pipelines and different shader types. The acquisition of practical knowledge consists of learning from online courses and YouTube tutorial series. This helps to build the required skill set for the successful execution of the practical project. Additionally, smaller, individual shader practices were included to support the learning process as the Godot game engine and constructing shaders with shader languages are not previously familiar to the author.

The thesis aims to provide the reader with a comprehensive understanding of the basics of shaders and their development pipelines while applying research outcomes to the practical project. The theoretical background presented in the study is universal for shader creation regardless of the game engine. The thesis does not function as a tutorial but rather as qualitative research on how to achieve appealing and viable results with relevant methods. A breakdown of the outcome is provided at the end of the paper showcasing the results of the study.

2.2 Research Methods

As for the research section of this thesis, two research methods were chosen to conduct the planned research. These methods were interviews (Hirsjärvi et al. 2009, 205) and comparative analysis (Mello et al. 2021, 2). The study heavily focuses on producing a shader with good quality, examining the visual appeal and fluency in production methods as well as optimisation. Therefore, both research methods chosen for the thesis are qualitative. The interviews were conducted as a part of the information acquisition to receive insights from individuals with experience in the field of visual effects (VFX) and technical art. The interview data was analysed by utilising the thematic analysis method (Caulfield 2019; Crosley 2021).

As the information seeking is completed the author begins to develop the snow shader asset where the insights from the previously gathered knowledge are applied. The finalised prototype of the shader asset is then compared to other snow systems present in commercial games on the market. The criteria on which the author compares their snow shader asset to other snow systems is formed based on the interview data and collected information. By conducting the comparative analysis, the author can evaluate how the quality and functionalities of their shader asset differ from shader systems seen in other video games.

3 BACKGROUND OF THE STUDY

The theoretical background presented in the study covers relevant academic literature as well as various sources acquired from different origins. Some of these are not peer-reviewed as specific academic information on the topic was extremely difficult to find or did not exist while writing the paper. The resources used in this thesis were found while seeking information by utilising XAMK Kaakkuri, Google Scholar, YouTube, and general Google searches for game industry articles. The main keywords used for the search were: “shader”, “shader player interactivity”, “interactive shader”, “video game interactive snow”, “role of VFX in games” and “shaders in video games”.

As shaders in computer graphics date back to the 1990s, there is a large variety of research and literature regarding shader production and basic shader theory. However, most material available is highly technical, covering low-level concepts of shaders, and therefore, is outside of the scope of the study. Literature such as the books by Bailey & Cunningham (2012) and Vivo & Lowe (2015) offered a thorough overview of shader theory and described basic practices while also delving into more in-depth topics. The Unity shader bible by Espindola (2021) provides information with practical examples but is fully Unity game engine specific. However, the shader bible and other Unity books are beneficial as most information is translatable to other engines and development platforms.

A large amount of specific information was found from a variety of sources including web pages, web articles, blogs, YouTube videos and wiki pages. As the backgrounds of the people behind these sources are partially unknown, the full reliability of the sources cannot be assumed. Due to the lack of relevant literature, some of these sources were utilised in the conducted study.

No academic sources or literature seemed to exist addressing shader interactivity or player-shader relationships. However, the role of VFX in games and as a gameplay mechanic was discussed in articles by Magic Media (2023) and Veselinovikj (2024). Similar topics were also briefly mentioned in talks at the Game Developers Conference (GDC) which were published on YouTube. Especially Barre-Brisebois' (2017) talk at GDC 2014 features constructing an interactive snow system for the game Batman: Arkham Origins (2013).

There appeared to be no previously conducted studies when searching for pre-existing research on shader-player interactivity or the player's influence on the shaders. The article by Barton (2008) explores the ideas of weather simulation and weather affecting the player in video games but does not feature any research on the matter. Various theses regarding shaders have been published in the thesis publishing platform Theseus, none of which explore the concepts of shader interactivity or player-shader relationships.

4 SHADERS

Shaders are programs containing a set of instructions for the GPU which define how the graphics should be rendered on the screen. The instructions are executed simultaneously and for every single pixel on the screen (Vivo & Lowe 2015). Generally, shaders take in inputs like vertices, textures, and view angles. They make changes to the input data according to the given instructions and render the results on the monitor. In video game production, shaders are used for real-time rendering where the view is optimally rendered at 60 frames per second. (Polycount 2020.)

Video games require large amounts of processing power compared to other programs as every pixel on the screen needs to be carefully computed. Additionally, in 3D video games, geometries and perspectives must be calculated as well (Vivo & Lowe 2015). A standard 1920x1080 pixel monitor has over two million pixels and each of these pixels has to pass through the shader at least 60 times per second. For this to be possible the shader code must run extremely fast. The computer's graphics processing unit (GPU) is designed to be a parallel processor (Figure 1) which has a massive number of tiny microprocessors that are all run simultaneously (Vivo & Lowe 2015). The GPU receives information and operates on it, but as all processes are computed at the same time it cannot reuse information. Therefore, each pass must be calculated separately. (Polycount 2014). Due to this limitation, information cannot be fetched from other pixels as the individual microprocessors are only capable of processing the information that they are receiving.

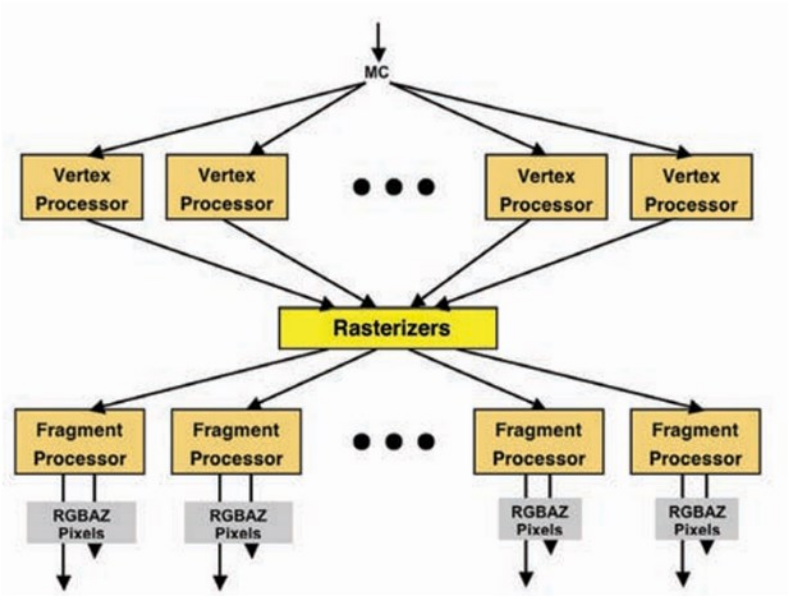

Figure 1. Abstracted parallelism in computer GPUs (Bailey & Cunningham 2012, 91)

GPUs support various application programming interfaces (API) which perform graphics rendering and allow developers to target different GPU hardware. They also “act as a bridge between the software and graphics hardware used to play games” (Garbett 2022). Shaders are closely tied to the graphics API as different APIs use their preferred shading languages and therefore affect the rendering pipeline. At their core, various graphics APIs share the same pipeline but act differently when it comes to the way each library utilises vertices (Garbett 2022). The most common graphics APIs are DirectX and OpenGL. DirectX by Microsoft is the most frequently used API nowadays, but it only supports development for the Windows and Xbox platforms. On the contrary, OpenGL by Khronos Group is more popular in developing for platforms other than Windows due to its versatile and open-source nature. (G2A 2024.)

4.1 Different Shader Types

Originally pixel shaders were the primary type of shader to perform operations in computer graphics (Hergaarden 2011). As shaders are now treated as a more general term, they can be categorised into different types based on how they alter the input before the final render. The common user-defined shaders in the

graphics pipeline are vertex shaders, tessellation shaders, geometry shaders, and fragment shaders (Bailey & Cunningham 2012, 39). Outside of the graphics pipeline is the compute shader which is structurally very different compared to the other shader types as it does not have a visual output (Espindola 2021, 303).

The vertex shader is a mandatory part of the rendering pipeline (OpenGL 2022). Its key function is to take all attribute values of the vertices and either use them or copy them into variables for later shaders to use (Bailey & Cunningham 2012, 43.) The vertex shader can be thought of as a loop where all the operations defined in the shader are applied to every vertex in the mesh. In other words, the vertex shader receives each one of the vertices and outputs a modified one (Borromeo 2021, 129). While vertex shaders can transform and modify geometry (Figure 2), they cannot add new vertices to the mesh (Hergaarden 2011). They are often used for animating water or leaves swaying in the wind. In case a custom vertex shader is not required, the shader can be left untouched, so the rendering pipeline uses the standard vertex shaders to render. (Holmér 2021.)

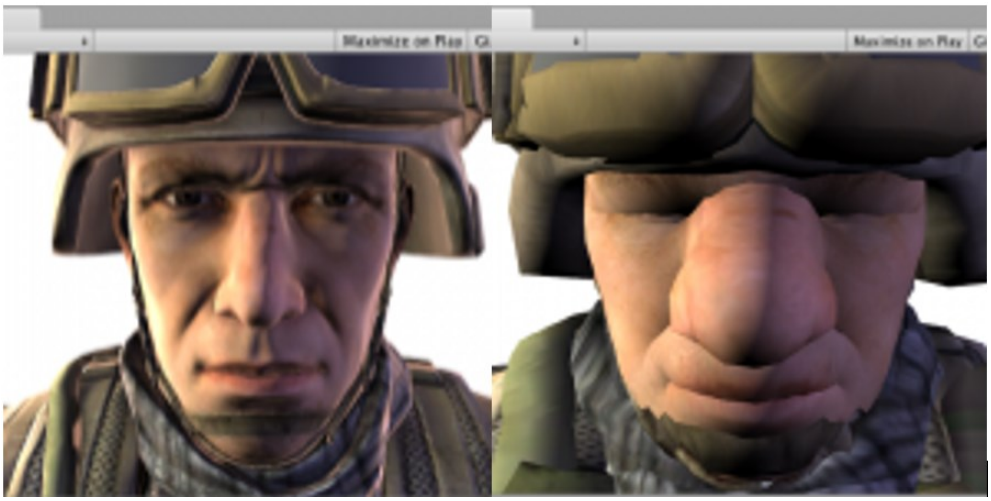

Figure 2. An example of a vertex shader moving the vertices along the normal of the 3D mesh (Hergaarden 2011)

Tessellation shaders are responsible for the amount of geometry in the scene (Figure 3). They follow the vertex shader in the rendering pipeline as they take in vertex data and interpolate the vertices in the mesh to create new ones and add geometry. Additionally, tessellation shaders allow the user to perform adaptive

subdivision of the geometry to increase the quality of the render, manage the level of detail (LOD) or apply displacement maps without defining detailed geometry (Bailey & Cunningham 2012, 50). Overall, tessellation shaders allow the user to improve the quality of the final image. The tessellation consists of two separate shaders: the tessellation control shader and the tessellation evaluation shader. As a process, it is optional in the rendering pipeline. (OpenGL 2022.)

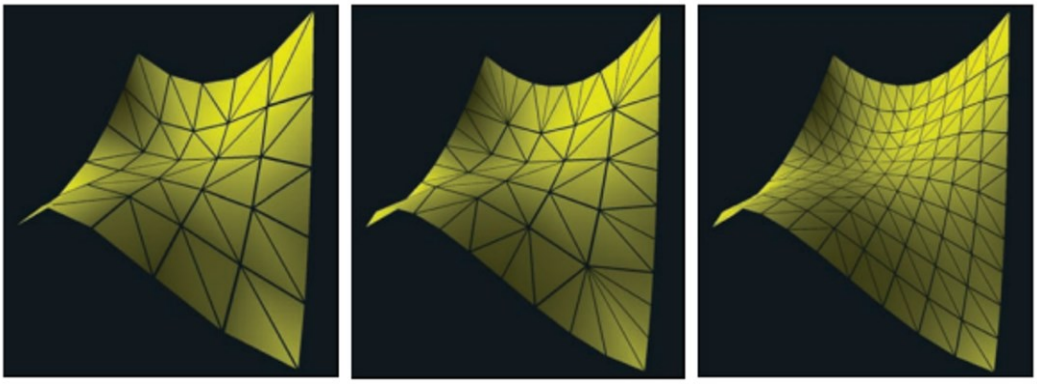

Figure 3. An example of a tessellation shader adding geometry with different tessellation levels (Bailey & Cunningham 2012, 232)

The geometry shader can effectively modify the geometry of a mesh as its operations can change or expand the original geometry by forming new vertices and vertex groups as shown in Figure 4 (Bailey & Cunningham 2012, 53). This feature makes it the only other shader type aside from the tessellation shader which can add new vertices to the original polygon mesh. Unlike the vertex shader which takes in a single vertex, the geometry shader takes in the vertices of a full primitive (Microsoft Learn 2022). The geometry shader has multiple uses due to its geometry expansion features and it can effectively be used in processes such as procedural content creation (Hergaarden 2011). Geometry operations however are not mandatory for the rendering pipeline, so the usage of the geometry shader remains optional for the user (OpenGL 2022).

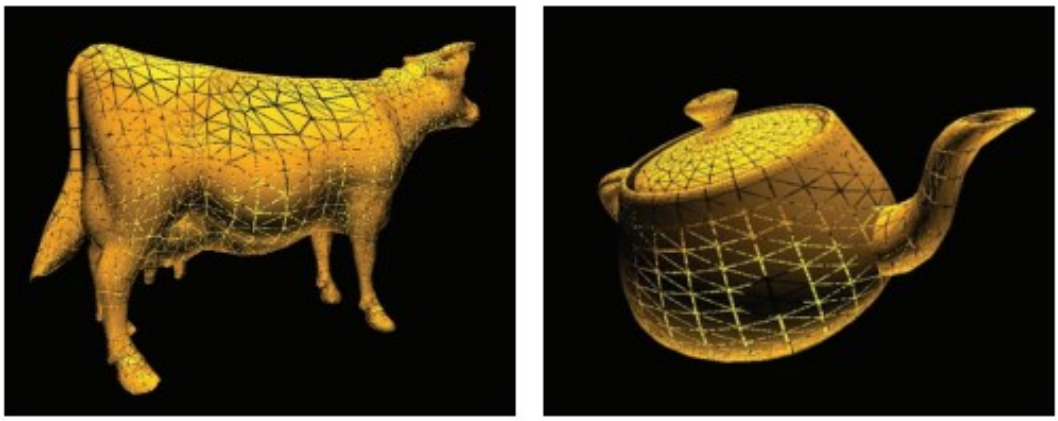

Figure 4. An example of two geometric figures with their component triangles shrunken (Bailey & Cunningham 2012, 53)

Fragment- and pixel shaders define the final colour or the RGBA (red, green, blue, alpha) value of each pixel that is rendered on the screen. Similarly to the vertex shader, the fragment shader can also be thought of as a loop except for every fragment in the primitive. All the operations defined inside the fragment shader are applied to every fragment composed in the rasterization stage (Holmér 2021). Fragment shaders are very versatile as aside from outputting a colour value they also sample textures, make lighting calculations, add normal details according to normal maps and do physically based rendering (PBR). This makes the fragment shader one of the most customisable shader types. (Borromeo 2021, 130.)

As fragment shaders are user-defined and fully optional to the pipeline they can also be left unused. In this scenario, they output an undefined colour for the rendered mesh (OpenGL 2022). Fragment shaders are the core of how surfaces are rendered and are most often used in defining surface details such as albedo, normal, roughness, metallic or opacity values. They can also be used for creating visual effects such as distortions, disintegrations or holograms. Fragment shaders allow the user extensive control over how each of the pixels is rendered as in the example in Figure 5. (Borromeo 2021, 130.)

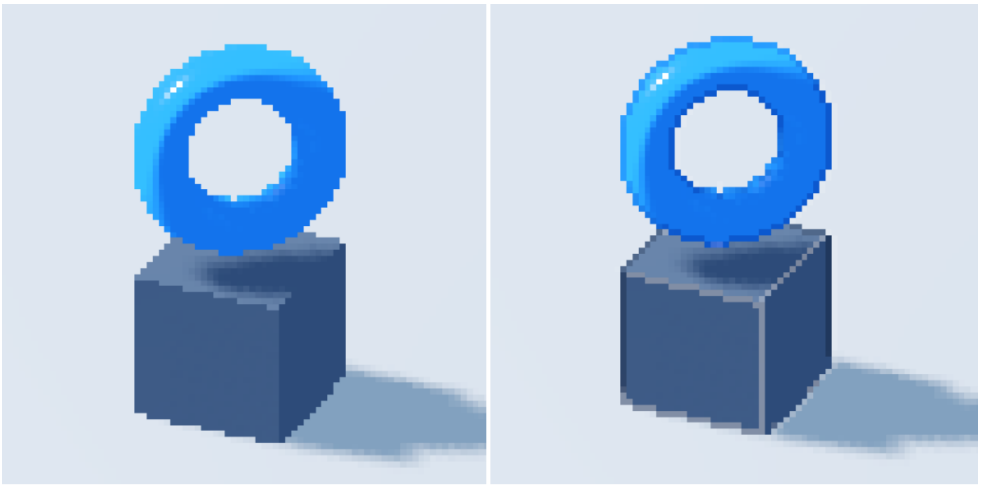

Figure 5. An example of how the fragment shader can be used to create pixelation to the viewport (Godot Shaders 2023)

Compute shaders are shader programs that run on the GPU outside of the regular graphics rendering pipeline. While they can also do rendering that is not directly connected to drawing primitives, the compute shaders are mostly used for computing arbitrary information (OpenGL 2019). They can be utilised to accelerate parts of the game rendering by running simulations. To be able to efficiently use compute shaders, in-depth knowledge of GPU architectures and parallel algorithms is often required. (Unity 2024a.)

4.2 Rendering Pipeline

A rendering pipeline is a sequence of steps the graphics APIs take when rendering objects on screen (OpenGL 2022). This pipeline varies slightly depending on which graphics API the hardware is currently using but the base order of execution is generally the same. The pipeline gathers vertex data and transforms that into a rendered image by moving through the steps presented in Figure 6. The rendering pipeline presented here is a simplification as they are often very complex and thereby not in the scope of this thesis. Additionally, the rendering pipeline here is presented according to the OpenGL API pipeline.

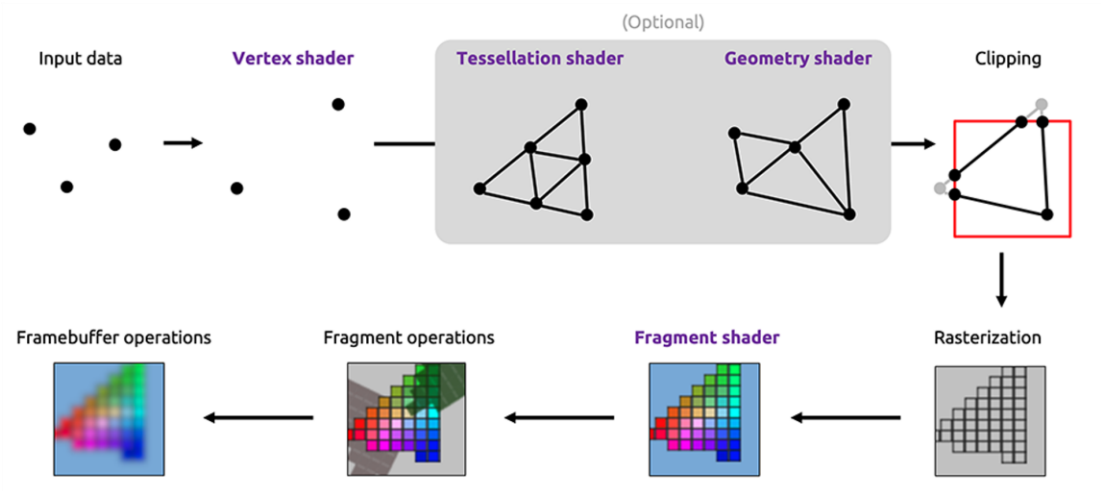

Figure 6. Real-time rendering pipeline simplified (CS1230)

The rendering pipeline begins with the vertex processing stage as the application receives a list of vertices that define the boundaries of the primitive to be rendered. Primitives are basic shapes like triangles, lines or points. The list of vertices called input data is fed to the vertex shader that outputs the positions of each vertex and transforms those positions to a different space for projection (Espindola 2021, 23). As the vertex shader can contain user-defined rules the vertex shader outputs the modified vertex positions instead of outputting the original ones in case modifications were defined. If the user wishes to utilise tessellation or geometry shaders those take place after the vertex shader. However, these shaders are optional to the rendering pipeline. (OpenGL 2022.)

As the vertex operations are completed the application moves to vertex post-processing that contains the fixed-function primitive assembly and clipping. The primitive assembly takes the output data from the prior stages of the pipeline and composes it into a sequence of primitives. As the primitives have been created, they are clipped according to what is being rendered on the screen. If the whole primitive does not fit inside the viewing volume, the parts that remain outside the bounds are clipped and left unrendered. At this stage, the vertex positions are transformed from clip space to view space to translate the coordinates to the 2D screen where the image is rendered. (OpenGL 2022.)

Now the application has received the vertex positions of the primitives to be rendered, and the parts left outside of the frame have been clipped away. However, at this point, the data is purely numerical and does not result in a rendered image. To transform the primitive into a rendered image the fixed-function rasterization stage takes the primitive with its attributes and converts it into a sequence of fragments as shown in Figure 7 (Borromeo 2021, 130). A fragment is a set of states that includes relevant data that is later used to compute the final data for a pixel. A variety of operations happen at the rasterization stage, but the user has close to no control over the process. (OpenGL 2022.)

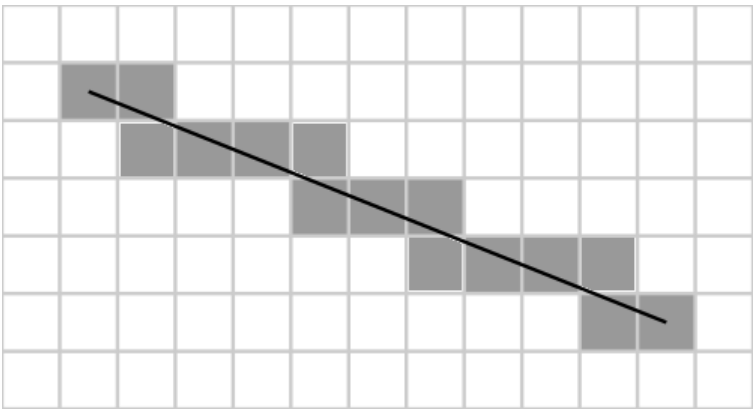

Figure 7. Visualisation of the rasterizing stage rasterizing a line primitive to a sequence of pixels (Sloka-Frey 2013)

Once the application has the fragment data, the fragments are processed, and the colour of the image can be set in the fragment shader (Figure 8). The fragment shader outputs a list of colours for each fragment while including the depth values that are used for depth calculations in the scene. As fragment shaders are optional for the pipeline, the image can be rendered without a fragment shader resulting in the colours left undefined. (OpenGL 2022.)

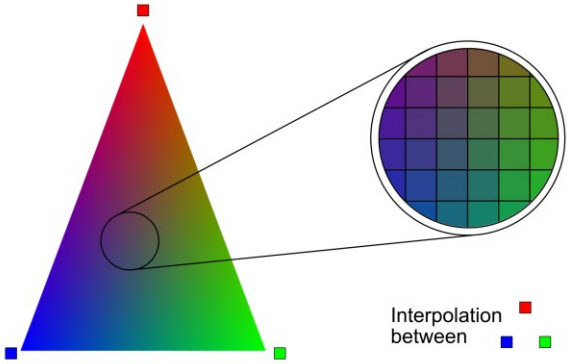

Figure 8. Visualisation of the fragment shader (Kodeco 2024)

The final stage of the rendering pipeline is sample operations which consists of multiple tests and fixed-function operations. Depth testing is one of the multiple tests that happen during this stage. Depth testing checks if the pixel to be rendered is in front or behind other pixels in the same position, guaranteeing that the closest pixel to the position of the camera is always drawn on top (Borromeo 2021, 131). After completing all the necessary tests, the pixels rendered are blended with the already existing pixels in that location in the colour blending operation. The colour blending has the most noticeable effect with transparency as the objects rendered further away from the camera position can be seen through the translucent object. Finally, the fragment data can be written to the framebuffer and is ready to be rendered on the screen. Any user-defined masking operations will take place just before writing to the framebuffer preventing certain values from being written. (OpenGL 2022.)

4.3 Shader Code and Visual Tools

Developing computer graphics in its early days was very tedious as every hardware needed its own software. OpenGL and DirectX were later introduced to provide more accessible APIs (Hergaarden 2011). Throughout the past few decades, shader production has been developing rapidly, streamlining the production of computer graphics. Computers are now capable of producing more realistic results faster as the demand constantly is growing. Shaders have become more accessible for users by becoming high-level in their programming patterns. Visual node-based tools have been developed and are already seen in multiple game engines making shaders more accessible for everyone.

According to OpenGL “shader languages are the interface used to program key parts of the modern graphics pipeline which have previously been fixed function state machines without programmability” (2006). Shader code works as the set of instructions for the shader to execute. Nearly all shading languages are similar to C languages often used in general programming and have the general features associated with them. Modern game engines can utilise different shader languages depending on the development software and requirements of the target platform. The most common shader languages are high-level shading language (HLSL) and OpenGL shading language (GLSL). HLSL is primarily used with the DirectX API on Windows platforms whereas GLSL is more flexible with platforms and is used with OpenGL and Vulkan APIs. (G2A 2024; Galvan 2022.)

To make shader creation more approachable, multiple game engines and development software have established their own visual node-based shader editors to streamline the creation of shader assets for video games. Game engines like Unity, Unreal Engine and Godot feature visual shader editors that are native to the engine. Unity has shader graphs (Unity 2024b), Unreal Engine has the material editor (Epic Games 2024), and Godot has VisualShader (Godot Docs 2024c). Node-based shader tools work very similarly to shader code, but instead of writing code, the shaders are constructed with a fully visual node-based interface (Figure 9).

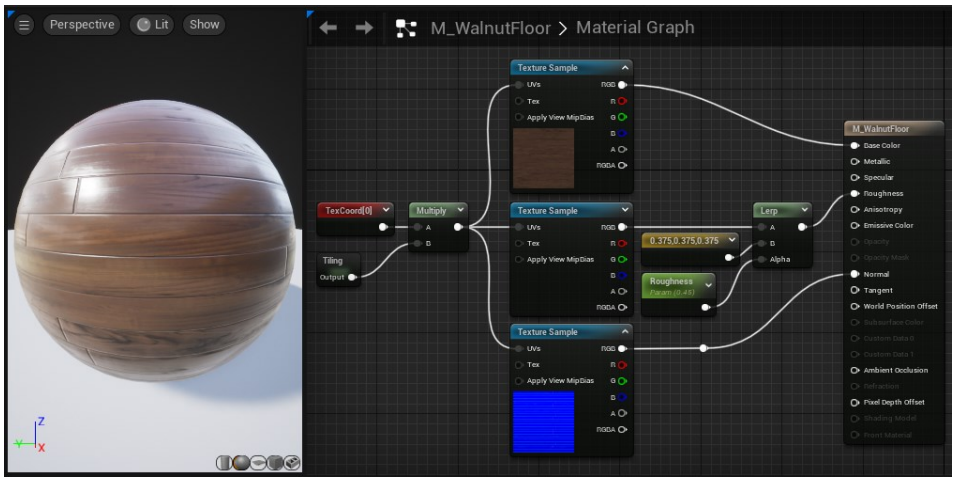

Figure 9. Unreal Engine’s material editor (Epic Games 2024)

These node-based graphs feature numerous different nodes with integrated calculations and parameters that can be connected to create various effects. Most node-based systems also provide a simple system to expose parameters to the editor view, therefore, avoiding the need to return to the node graph when the values need to be adjusted. The visual shader tools simplify shader creation and give instantaneous node-specific visual feedback.

5 SHADER-PLAYER INTERACTIONS

Visual effects in video game development can be overlooked as they are often depicted as merely adding polish to a fully functioning game. While visual effects play a significant part in polishing, they are crucial in making the game world feel immersive and engaging. They can set the tone of the world, create a sense of realism, and communicate information in a visual form (Magic Media 2023). However, visual effects can also influence the gameplay. Visual effects such as fog can work as a gameplay mechanic as they can restrict the player's view, and therefore, drive the player's exploration (Veselinovijk 2024). Especially shaders are often used to describe surfaces, and commonly, they feature no interactivity. In some cases, interesting effects can be created by adding responsiveness to ground the player in the game world but also affect the gameplay itself.

Responsiveness of visual effects can play a bigger role in video games than what is commonly perceived. Visual effects as a communication device do not appear overly noticeable but the absence of interactions has a profound effect on one's ability to understand the results of actions in a virtual environment (Magic Media 2023). Since the beginning of computer graphics and the development of video games, the goal of rendering has been to achieve results that appear so realistic that the player cannot tell the virtual world apart from the real world. According to developers and game critics, immersion is achieved when the player forgets that they are playing a game. Realistic graphics are easier to process than the more minimalist and abstract graphics often used in early video games. (Barton 2008.)

Weather systems are an instance of more intricate VFX systems which can include shader-player interactivity. They are difficult to execute as numerous factors affect their believability. Many games try to portray weather but fall short as it would be extremely time-consuming to make the character respond and behave differently according to various weather conditions. Weather is a profoundly complex phenomenon and, to replicate it accurately, each raindrop and snowflake would have to be rendered separately. (Barton 2008.) Modern rendering techniques are aiming to achieve this, and shaders often play a big part in portraying weather as they can be used to describe surfaces like mud, puddles, or snow. In games like Batman: Arkham Origins (2013) the developers had a clear motivation to enhance the game world with deformable snow (Figure 10). According to Barre-Brisebois (2017), more simple solutions such as decals were not immersive enough. The developers aimed to have an organic approach where the snow can react to player movement even when sliding or rolling on it to achieve a more iconic and organic gameplay feel (Barre-Brisebois 2017).

Figure 10. Deformable snow in Batman: Arkham Origins (Barre-Brisebois 2017)

While studying shader-player interactivity, it can be noticed that every game has its own requirements for realism and immersion it aims to achieve. Especially in older games the technical limitations at the time set many of the limitations for the game's rendering. Common ways to demonstrate shader-player interactions are the surfaces or visual effects that respond to the player and their location in the

world. Pro Pilkki 2 (2003) and Pokémon Brilliant Diamond (2021) have the player interacting with snow by leaving footprints. In Pro Pilkki 2 the player only leaves behind footprints on the faint layer of snow (Figure 11) whereas in Pokémon Brilliant Diamond the player can also walk in deeper snow leaving behind a trail deforming the snow (Figure 12). In both games, the player influences the environment, but the environment does not influence the player's movement or behaviour. As portraying weather in video games aids in producing the illusion of a dynamic and lived environment both games set the tone of the area and communicate the impact of the player's actions in the environment (Barton 2008).

Figure 11. Screenshot of Pro Pilkki 2 (Procyon Products 2003)

Figure 12. Screenshot of Pokémon Brillant Diamond (Nintendo 2021)

Shader-player interactivity can also be created by establishing more complex relationships between the two components of the game. Instead of having only the player affect the environment, additionally, the environment can also affect the player. It is important to note that in this interaction, it is not only the shader that creates the combination of effects and behaviours that creates the illusion of interactivity. Effects like changes in player behaviour are done as a part of the game programming. Super Mario Odyssey (2017) is an example of this where the player character starts to shiver if the player spends too much time standing still (Figure 13). This mechanic does not have a gameplay purpose and will not result in the character freezing to death. Instead, it most likely functions as a visual touch to make the character feel present in the game world.

Figure 13. Screenshot of Super Mario Odyssey (Nintendo 2017)

Responsiveness can also be achieved with a combination of multiple shaders like in the game Journey (2012) where the characters' clothes get snowy after traversing the snow-covered terrain (Figure 14). The snow sticking to the player's clothes makes the player feel present in the game world as they are physically affected by their surroundings. Journey's developers spent a large amount of time ensuring that walking in the world feels engaging and fun rather than focusing on making the game look extremely real (Edwards 2018).

Figure 14. Screenshot of Journey (Sony Computer Entertainment 2012)

Due to the high demand for larger games and better graphics, modern video games are getting larger and more challenging to develop with a high focus on the game looking and feeling as close as possible to real life (Veselinovikj 2024). Bigger titles released by massive AAA companies tend to focus more on achieving immersion through realism on a level like no other. Red Dead Redemption 2 (2018) and God of War Ragnarök (2022) are known for their incredibly realistic snow effects. Both games have succeeded in making a snow system that looks and acts like snow in the real world (Figure 15; Figure 16). Red Dead Redemption 2 (2018) also allowed the snow to affect the player's movement where it is possible to tell how the freezing temperatures slow down the playable character. Both the player and environment feel like living, dynamic beings in the game world where they interact with each other in different ways just like they would in the real world.

Figure 15. Screenshot of Red Dead Redemption 2 (Rockstar Games 2018)

Figure 16. Screenshot of God of War Ragnarök (Sony Interactive Entertainment 2022)

Weather can play a larger role than only existing to be coherent with the real world. It can also set the tone and play a critical dramatic role in storytelling. It has been used as a part of the narrative by writers and filmmakers due to its conscious and subconscious significance (Barton 2008). The responsiveness in video games is unique as similar immersion cannot be achieved with movies or books. The weather conditions and snow in both Red Dead Redemption 2 (2018) and God of War Ragnarök (2022) are examples of how these effects can play a massive part in environmental storytelling and gameplay. This shows that visual effects in video games can do more than just add visual polish.

6 PRACTICAL PROJECT

The practical project done as a part of the thesis process is an interactive snow shader that showcases an established player-shader relationship. It serves as a fundamental part of the thesis as it demonstrates how applying the acquired knowledge and research outcomes to production can improve the quality of the outcome. If the qualitative research is successful at the end of the thesis process, the author has constructed a snow system that is on a similar level to other snow systems in published games and as well as is usable as a game-ready asset.

To start the production process the author defined the initial idea and scope of the snow shader asset. As the author's schedule was limited by the deadlines set by the university, careful planning and scoping had to take place to ensure that the project was completed on time. The author also determined the game engine of choice. Finally, as the author had limited knowledge of shader creation some self-study and research on the practicalities of making shaders was done before starting the practical project. The aspect of shader-player interactivity also played a large part in the production. As there are limited resources on the topic, the author occasionally had to rely on limited knowledge to navigate their limitations. Throughout the production of the practical project, the author continued to study and learn about shader production to support the process.

6.1 Scope – Snow Shader Asset

The functionality design of the snow shader revolves around a simple interaction cycle (Figure 17) that establishes how the different components in the demo scene influence each other. The snow shader itself is only a smaller portion of the whole interaction system as it is in charge of rendering the snow on the ground and displacing it according to the player's path. The snow shader determines the look and feel of the snow as well as how much the player's movement affects it. At the same time, the displacement data is read by the player controller making the player move slower while traversing in deeper snow.

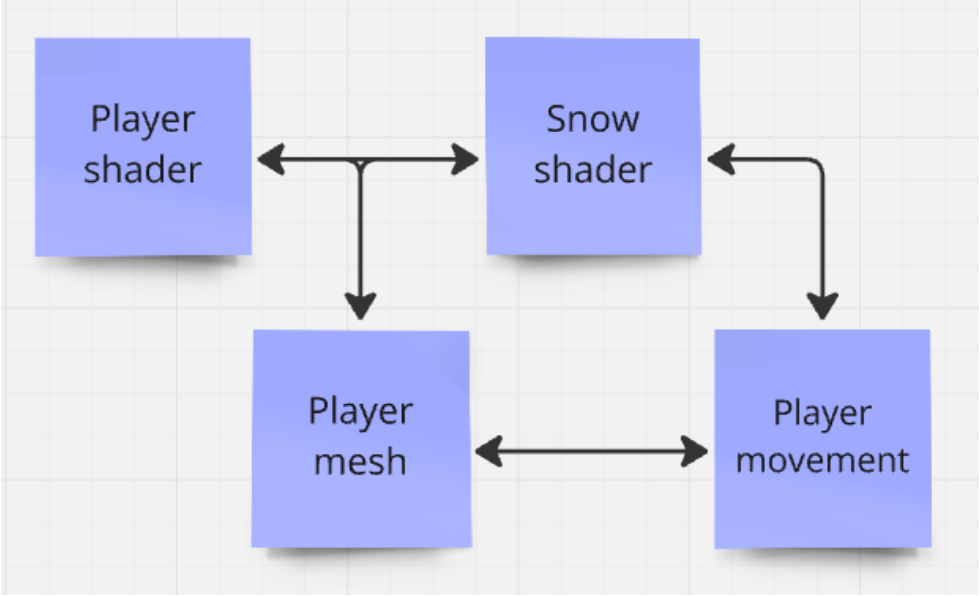

Figure 17. A graph displaying the shader-player interaction functionalities in the demo scene

The player mesh refers to the 3D model of the character. This mesh is connected to the player controller and works as the visualisation of the player character. The player controller influences the character's movement both in terms of locomotion and controlling the animation tree. The player's movement loops back to the snow shader as the visualised player character’s movement affects the surface displacement of the snow. Finally, the snow shader together with the player mesh affects the player shader which makes the effect of snow sticking to the player's body. The player shader takes influence from the snow shader’s surface displacement information; therefore, the snow only gets caught in the player when the character is walking in deep snow. The player character’s mesh affects the player shader as that shader defines how the mesh is rendered on screen.

The art style is heavily influenced by the schedule and scope of the project. The art style (Figure 18) mainly draws inspiration from the Journey’s (2012) snowy areas towards the end of the game, but references and inspirations are also drawn from games like Super Mario Galaxy (2007) and God of War Ragnarök (2022). The approach to the snow is stylised rather than fully realistic as realistic snow is difficult and time-consuming to produce, and it would not be feasible to

execute sufficiently within the timeframe and skill set of the author. Regardless, some real-life examples and locations are used as a reference for the shader to make it feel more familiar to the player.

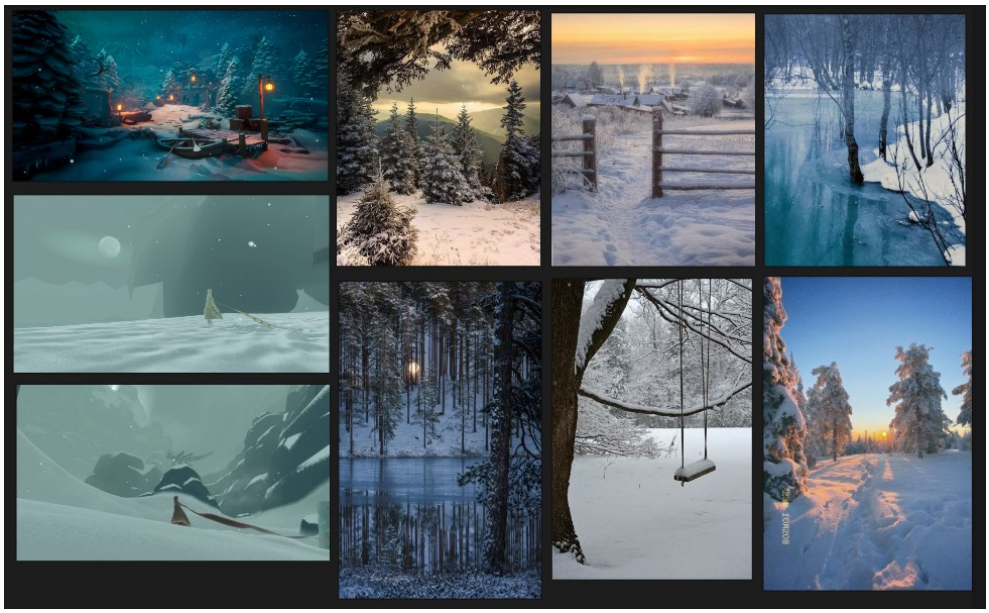

Figure 18. Reference board for the snow shader

The snow system is demonstrated in a small game scene with a simple 3D environment. The scene mostly consists of plain terrain, but some trees, rocks, and fence posts are added to improve the presentation of the shader system. The demo scene aims to showcase the final shader asset and other constructed functionalities in a game-like, visual environment. The final demo scene was meant to feature an icy lake, providing the player with a clear area to feel the difference between walking in snow and on a clear surface. Due to time constraints, this feature was cut. Some snowfall is added as an additional touch.

6.2 Tool – Godot Game Engine

The game engine of choice for the practical project was the fully open-source Godot game engine. At the moment of writing the paper, Godot is not considered an industry-standard engine. However, its popularity has recently increased (Tkachenko 2024). The author originally chose to utilise Godot due to a personal interest in it. However, as the author was studying the engine, it was found that Godot has more limitations in terms of shaders than the industry-standard

engines such as Unity or Unreal Engine. This allowed for a unique learning opportunity to navigate a larger number of limitations and manually write functions that would normally be provided. This further solidified the choice of the engine. The plain approach allowed the author to understand concepts behind shaders that would normally be mostly automated. Additionally, the author wanted to compare the development experience with Godot to other engines, but this was outside the scope of the thesis.

Godot game engine was originally founded by Juan Linietsky and Ariel Manzur in 2014 and is now being developed by a team of hundreds of regular and occasional contributors all around the globe. The version 1.0 was released back in 2014, and the engine is frequently being updated with the latest version 4.3 released on the 15th of August 2024 (Endoflife.date 2024). Godot is a real-time 2D and 3D engine used for game development featuring innovative design and numerous plugins from the active community allowing flexibility and customisability. Godot uses its own scripting language called GD script. Godot also offers support for C#, C++, and others through the GDExtention plugin. Development is possible on the standard PC environments and games can easily be exported to multiple platforms including PC, mobile, web, consoles, and mixed reality devices. (Godot 2024.) Godot is entirely funded by sponsorships and donations from individuals, game companies, and organisations like Khronos Group, OSSC, Google Play, Re-Logic, Pirate Software and many others (Godot 2024; Stitching Godot 2024).

Godot, like many other engines, draws everything with shaders (Godot Docs 2024a). The shaders are written in a shading language called Godot Shading Language which is based on the popular GLSL language. However, Godot's language features additional quality-of-life updates and simplifications making it more accessible for the user (Godot Docs 2024b). Godot also features a node-based visual scripting alternative called Godot VisualShaders that provides the user with additional visual convenience (Godot Docs 2024c). Outside of the visual rendering pipeline, the user can utilise compute shaders that work as a useful tool to offload calculations to the GPU (Godot Docs 2024d).

As an addition to the standard material renderers for 2D and 3D, Godot has multiple shader types for user-defined shaders. The five different shader types are spatial, canvas item, particles, sky, and fog. These are specific to Godot and therefore are separate from the standard shader types discussed in Chapter 4.1. Each shader type has render modes that alter the way Godot applies the shader. Godot has spatial shaders for 3D rendering, and canvas item shaders for 2D rendering. Both shaders have vertex and fragment functions for the corresponding shaders as well as feature an additional light function that runs for every pixel and every light. The particle shader type is used with particle systems and simulations, while the sky shader modifies sky boxes in the game world. Lastly, the fog shaders are used to alter the volumetric fog in the game scene. All the shader types are user-defined and do not have to be modified as Godot has standard materials and shaders for all of them. (Godot Docs 2024a.)

6.3 Skill – Learning to make shaders

At the beginning of the thesis process, the author had limited knowledge of shader creation and no prior experience in using the Godot game engine. The author had some experience in utilising visual node-based tools. Before starting to work on the snow shader asset, the author spent around a month studying shaders and making smaller personal experiments in Godot.

To start with a solid foundation, the author went through an online course on the Udemy platform by Gamedev.tv (2024). The course taught the basics of constructing shaders in Godot. The course featured three sections, first learning about the basic shader concepts and the syntax of the Godot shading language. The second part featured 2D shaders and finally, the third part 3D shaders. The Godot shading language was utilised throughout the course which gave the author a foundation for understanding the text-based shader workflow and logic.

YouTube has a wide range of online shader tutorials for multiple game engines and especially due to Godot’s open-source nature, many developers are openly creating tutorials and sharing source files. The author aimed to follow a variety

of different tutorials for a couple of weeks while trying to understand shader workflows and gain practical experience working with shaders. When finally feeling confident enough to start the production of the snow shader, the author still utilised the help of tutorials and videos to get started. The most helpful ones regarding the snow shader were the videos by Donald (2021) and Albukhari (2020). Both videos used a similar method of creating snow which was remotely easy to implement and iterate upon. Aside from these videos, various other video tutorials were used to construct the different systems or features that are present in the final demo scene. Additionally, references and help were acquired from the Godot Shaders (2024) website which functions as an open publishing platform for shaders made for Godot. The code regarding the shaders is published on the website along with additional guides and instructions.

To acquire an understanding of the engine specifics Godot Docs (2024) was used to acquire precise knowledge of the syntaxes and features. The documentation is kept fully up to date and maintained by the active Godot community, so information is easily accessible and reliable. Information in Godot Docs (2024) is well organised and helped the author debug syntax errors and learn more about the capabilities of the shader editor inside the Godot engine.

7 INTERVIEWS

To scope relevant pipelines and standards regarding shader creation, the author arranged interviews with people who have experience in shader production. Interviews are great for qualitative research as they allow the author flexibility regarding the acquisition of the material depending on the situation and the interviewee (Hirsjärvi et al. 2009, 205). The interview method has allowed the author to talk with game industry professionals and a student from different backgrounds regarding shader production. This allowed the author to acquire valuable insights and hear experiences that proved to be greatly useful in the thesis process. The interviews were kept casual for the participants, allowing the author to learn about performing interviews as part of the research.

The interviews were performed in virtual calls by utilising different communications software such as Discord or Google Meet that are available to use for free on the internet. The interview method chosen for these interviews was the semi-structured interview in which the themes of the questions were predefined (Hirsjärvi et al. 2009, 208). The defined interview themes were *knowledge and experience, production methods, optimisation and scalability*. Despite having a list of prepared questions (Appendix 1), the author did not want to limit the conversation, therefore, follow-up questions were allowed. This interview method was the most fitting for the requirements of the research as the author wanted to avoid the conversation branching outside of the study's topic. At the beginning of the virtual call, each interviewee had the opportunity to decide whether they wished for the interview to be recorded and if the author is allowed to mention them by name in the thesis paper. Each of the interviews lasted from 45 minutes to an hour to validate the interview as a method over a questionnaire.

7.1 Interviewees

Four people were interviewed, three of whom were from different backgrounds and currently working in various roles in the game industry while one of the interviewees was a final year student. All the interviewees shared the same passion and interest for shaders and VFX or technical art. The interviewees have all agreed to be mentioned by name in this paper, but all personal and work-related details have been disclosed. The author was fortunate to acquire people from different backgrounds and roles, allowing for more potential viewpoints and insights to surface during the interviews. The summaries of each interview are included in the appendixes.

Joni Helén (Appendix 2) is a last-year student studying game programming. He does general-purpose programming but has picked up an interest in shaders and has been developing them for more than two years. Most of his experience in shaders comes from experimentation and personal projects and he aims to keep developing shaders in the future as well.

Freya Holmér (Appendix 3) is a technical artist and online educator with a background in 3D modelling and level art. She has worked in the game industry creating various shader effects for games but now focuses on tool development and is currently working on a 3D modelling tool built inside the Unity game engine. She is also running her own YouTube channel where she shares educational videos and lectures about technical art and game development math.

Teemu Väisänen (Appendix 4) is a game development technical director at a mobile game company and has a background in 3D modelling. He has worked for various companies in different roles and throughout his career he has had a growing interest in building a bridge between programmers and artists. His goal has been to understand the needs of the team and to ensure that the artist's vision is carried through in the production. In his current job title, he does not craft individual shaders but rather focuses more on pipelines and production.

Ilaria Cislaghi (Appendix 5) is a freelance technical artist focusing on visual effects and a contributor to the Godot game engine. She has a background in computer engineering but has also been personally involved in art. During her career, she has worked in various roles in game development before fully moving to the field of VFX. She currently maintains the particle system in Godot in her free time.

7.2 Thematic Analysis

The interviews provided rich insights regarding shaders, shader production, and their common limitations and challenges. To analyse the data acquired from the interview transcriptions a thematic analysis was used to identify the themes and categorise the results according to them (Caulfield 2019; Crosley 2021). Due to the nature of the semi-structured interview method, the author had pre-emptively set some themes to be discussed in the interviews. However, the concluding themes were redefined to better suit the analysis of the interview data. The following main themes were recognised from the interviews: *limitations, production, and knowledge*.

As the focus of the study is shader-player interactivity, all the interviewees were asked about their experiences with shader-player interactions and whether they had prior experience in creating such effects. All the interviewees have experimented with shader interactivity while some have had more experience, having made more complex effects on commercial projects. Examples of the projects mentioned where shader-player interactivity was utilised were snow, mud, and grass shaders.

The first theme recognised from the interviews was *limitations*. It also included topics like approaching limitations, compromises, and priorities as the results were all closely related to each other. When discussing general concepts of shader-player interactivity all the interviewees mentioned the common limitations which can surface when working with shader-player interactions. It was acknowledged that interactive effects often require further planning than the more common shader-based visual effects. The developer must know the details of how the interactions are going to happen, how many are going to play out simultaneously, what needs to be stored in the memory, and how that memory is going to be utilised. The interviewees brought up a few features like a dynamic number of interactive components as well as scaling of the interactive area which can create complexities with shader interactivity.

> *Let's say you want to have an interactive grass, and you want some objects to like deform the grass. Then you have to have like an array of things you want the grass to affect, or like things you want the grass to know about. But, but the array cannot be dynamically sized because you don't have dynamically sized buffers. So, you have to know how many things there are beforehand. And that creates design complexities you have to take into consideration when doing your effects.* (Helén, 17 September 2024.)

> *Well, I think there, there are a lot of like problems you kind of have to solve that are a little bit outside of what you usually do when working with visual shaders when you do this kind of thing. So, you need to be like very, very careful about how you store the data of the state that you're changing based on player actions. […] And, and some of those things are relatively complicated as soon as you want to scale them.* (Holmér, 19 September 2024.)

All the interviewees described some general workflows when working with shader-player interactions. They shared their approaches from previous projects or described how they would approach the problem if they did not have extensive experience with these types of shader effects. The solutions varied massively based on the requirements and designed features of the project but also based on the person's subjective preferences. Therefore, it was difficult to identify a singular correct approach to a certain problem. Usually, there were several possible solutions that the interviewees considered, and the challenge was finding the one that best fits the requirements of the game and target platform in that specific scenario. It was mentioned that generally, the limitations are set by shader rendering or current technology, generating some challenges that lack a defined, correct solution.

> *I don't think there is a good, good kind of solution. You have to have some kind of upper limit that you that you're fine with and you use that, or the other approach is to have some kind of dynamic build system…* (Väisänen, 24 September 2024.)

> *And the snow is using a completely different approach. It has different limitation. The snow supports X amount of, of players. There's really no limit on how many how many players you can support from the snow. There is a limit, however, on the area where you can show those footprints because the way that it works is that each entity that can interact with the snow draws on a 2D canvas that follows the main camera, and it draws from above. So, you can have as many entities that have those little brushes that draw into this canvas and then this canvas is passed into the shader. However, this canvas has like limited resolution and limited distance size. So, which means that you can't have your trail in the snow render forever.* (Cislaghi, 26 September 2024.)

Having worked closely with shader-player interactivity in different shader effects both through personal experimentation as well as projects related to prior work-related projects they have worked on, Väisänen and Cislaghi shared their insights and key takeaways on approaching interactivity. Both highlighted the importance of planning and knowing one's limitations before starting to work on the interactive shader asset. Creating complex effects with shader-player interactions is about setting the priorities and requirements for the outcome as it is often impossible to achieve a perfect result.

> *Well, I would say you have to just kind of like think about it beforehand and you know, know the kind of the downside and the upsides of each, each kind of approach and kind of plan accordingly. […] So, it's usually quite easy to kind of think about the limitations of your game and fit your solution around. So, I think it's mostly a planning issue. But of course, you have to have some experience on working with this, all these approaches. So, you know that they exist.* (Väisänen, 24 September 2024.)

> *You will never make a perfect shader. You will just have to know what are the limitations that matter for your shaders and which limitations don't matter for you and just kind of have to adapt. We're cheaters. Technical art in VFX. It's about cheating.* (Cislaghi, 26 September 2024.)

Priorities and compromises are closely related to the theme of limitations as the interview data revealed that at times it can be impossible to achieve a perfect result. According to the interview results, shaders appeared to not have an industry-standard production pipeline like many other disciplines of game art have. Their limitations and requirements define the production pipeline. All the interviewees described their priorities in shader production. Väisänen and Cislaghi also elaborated on how they form priorities in team settings. Väisänen mentioned the viewpoint of working within a team where the artists carry the vision, but he is the person to set the priorities and limitations of the shader while trying to follow the original vision. Cislaghi also mentioned how the limitations and requirements for the shader asset can often come from the development team or stakeholders rather than oneself. It was discussed with the interviewees how one’s background in game development can affect the priorities in shader creation as it is more natural to focus on the aspects of shaders that are closer to pre-existing experience.

> *Yeah, I think, I think like prioritising visuals first I think is the best because what, what tends to happen in games is that you quite often get new ideas as you work on them. Like you don't always know ahead of time exactly what you're going to end up with.* (Holmér, 19 September 2024.)

> *Depends a little bit, like if it's something that, for example, an artist is asking for, like they have some kind of like effect that they're looking for, they want to have in the game. I think then you have to think about like end result first. Of course, it doesn't make sense to do something else if they ask something. […] So, you have to kind of like limit them a little bit and try to figure out the way to do what's behind the thing that they want to do. Like similar enough that they're happy. Sometimes you have to do compromises.* (Väisänen, 24 September 2024.)

> *I think it's always about priorities. It's always about choosing to prioritise one thing over the other. Do you want a better visual fidelity or do you want better performance? They do not go together. You have to choose one […] But in general, you have to make choices. It's always about communicating with your stakeholders, be it the art director, the engineering team, etcetera. You have to talk with people and say, hey, what are our priorities?* (Cislaghi, 26 September 2024.)

Holmér also mentioned the intended target platform setting the priorities of the shader as some platforms require a larger focus on optimisation than others do. According to her, developing shaders for platforms like virtual reality (VR) devices required a much larger focus on optimisation than for PC or consoles as they do not require as much optimisation which allows for more freedom in terms of the visuals. Regardless of the platform, Holmér emphasised the importance of being mindful of the performance to avoid issues later in the production.

> *I think it depends on the shader and it depends on your target platform. If you're developing for like VR, especially like portable VR, like the the Quest, you have to be super mindful of performance. And, and so I think on platforms like that, you kind of have to think about that ahead of time. You can't just like barrel forward and like do the coolest shader ever and then you're like, you're just going to set yourself up for disappointment. I think. So at the very least, like getting, getting to know the limitations of the platform beforehand is really important.* (Holmér, 19 September 2024.)

The second theme that emerged from the interview data was *the production* of shaders. Due to the varying limitations and requirements that set the priorities of the shader asset processes such as optimisation turned out to not have an industry-standard approach. Instead, the interviewees mentioned some common practices that should be avoided as they can cause performance issues. The practices mentioned were dealing with depth, transparency, and certain shader types like geometry shaders. As no common requirements were mentioned it can be assumed that framerate and overall performance are the main optimisation factors to consider. Väisänen mentioned that in case of a performance issue, optimisation should not happen blindly but rather he emphasised the importance of first trying to find the cause as it may not always be straightforward. Allocating resources was also mentioned as in some cases more expensive solutions can

be acceptable if the shader plays a massive role in the game. This is tied to setting the priorities of the shader which can determine how expensive the shaders are allowed to be.

> *Well, there really aren't any standardised things you can do. Like standardised, I feel standardised means that it's well defined somewhere, in like a Bible, that states that this is the way to do this. But I don't think there exists such a thing as a shading standard, so to say. So, there are just things that people do, and they work usually…* (Helén, 17 September 2024.)

> *I don't think that there's like a requirement, especially not standardised requirements. But I would say that the most important thing is that you're like, if you think about the optimisation and you're like looking at the FPS counter, it says 30 FPS and your target is 60 FPS. The first thing shouldn't be that, OK, let's start to optimise something. It should be that you go to the profiler and you actually figure out what's the bottleneck because it, it doesn't make any sense to, for example, optimise vertex counts or put all your artists in the work and they reduce the where this is when you actually a bottleneck is let's say fragment shader or fill rate on the screen.* (Väisänen, 24 September 2024.)

> *So, it's complicated to its, its like we said before, its trade-offs, right? […] But if you're making Sea of Thieves and you're making the water shader, you don't care about making the most optimised shader possible because water is so important in your game that you want to allocate resources like graphic resources to that. So probably they do have loops in their shader in the, in the water shader of Sea of Thieves, and that's ok. It really depends.* (Cislaghi, 26 September 2024.)

Scalability is related to optimisation and therefore the theme of production. Such choices regarding scalability can determine many of the priorities set for shaders. The interviewees were asked whether shaders should be scalable or reusable within a game project or between multiple projects. All interviewees shared similar opinions on the matter, declaring that shaders are mostly developed for specific purposes rather than making them massive systems that can work in a variety of scenarios. Holmér described how games can have a hybrid approach where a master shader is used to define the overall look of the game, but some specific shaders are used to determine the look for aspects of the game requiring higher customisability. Helén also mentioned how shader libraries can aid when it is required for the shader to have multiple functionalities which can be easily disabled if they are not needed with that specific instance.

> *Depends on the project, yeah. Depends on the target platform. Depends on your performance, like limitations. Depends on the kind of workflow you want. Yeah, depends on a lot of different factors. I'm not sure if there's like one good answer for this. Some people really like having custom shaders for everything. […] But in quite a lot of projects you tend to have kind of like a master shader that a lot of different things use. […] But, but that obviously puts some restrictions on your like what's possible to do. And that's kind of annoying sometimes. But, but yeah, so, so usually there's, from what I can tell in most projects, they kind of have a hybrid approach.* (Holmér, 19 September 2024.)

> *Well, like usually with the (shader) library there comes the scalability, because you can define the library such that with the combination of the variants you can create a struct, or create the inputs and outputs of the shader, such that you can have a shader variant that only has the things you want in the shader. And then you can define the shader as you want. […] So you can create this input and output struct that has everything you could ever want, and then just disable and enable some of those features depending on what you want to use.* (Helén, 17 September 2024.)

Cislaghi also introduced the important viewpoint of the developers themselves remaining flexible and setting up the workflow in a way that scaling or editing the shader is easy. Some of the interviewees also brought up shader readability when discussing flexibility and editing the shader later in the production. They emphasised the importance of making everything in the node graph or code file readable to ensure efficiency when editing or adding new features.

> *It’s not about the shader itself being, being flexible. It's about the structure and the way that you set up your game, your workflow. If your workflow is not flexible, you're just going to go and harm the project just because you can't move and put yourself in a situation where you can't move.* (Cislaghi, 24 September 2024.)

Tools are a vital part of the theme of production as they play an important part in defining the pipeline in shader production. Visual node-based tools have gained a lot of popularity in recent years and are now often compared to the more traditional shader programming. All the interviewees were asked about their opinions and experiences regarding these two different tools in shader creation. All of them had experience with both tools but also had a personal preference for a tool that better suited their needs.

The topic of node-based tools compared to shader programming raised the most similar answers out of the interview questions as all the interviewees said that node-based tools are generally easier to learn and great for fast-paced, iterative, or experimentative processes. Sometimes through easy experimentation, it is even possible to accidentally create solutions or effects that can be used later. However, it was mentioned that the node-based tools cannot reach low-level architectures as they tend to oversimplify some of the features that would be useful for a developer to have. The clear strength of the node-based tool brought up by the interviewees was the visual node-specific feedback which can be a massive help when trying to reach a certain visual outcome or when debugging.

> *Maybe if you're trying to create something quickly and efficiently, you usually use the node tools because it's so simple. And like if you, for example, if you're creating a shader in, let's say, Unity's high-definition pipeline, you don't, I don't even think you can write those by hand because they are so complicated to write by hand. You don't even want to try. So it's much simpler to use those tools and they work relatively well.* (Helén, 17 September 2024.)

> *They have different tradeoffs, I would say. I think so. So like I mentioned, I think node-based shader editors are really good because they're visual. So if you're starting out and learning shaders, they are super valuable just for that aspect alone. […] And another really big advantage with node-based editors is that once you're working with way more complex render pipelines, node-based editors have a tendency to abstract away a lot of a lot of stuff that's really tedious to do by hand.* (Holmér, 19 September 2024.)

On the other hand, the text-based shader programming workflows were described as fast, powerful, and more performant ways to develop shaders. According to the interviewees the programming-based workflow, however, requires a larger understanding of the technicalities of shaders. Node-based tools take care of this part of the process automatically. The interviewees mentioned that only people who know how to write text-based shaders can maintain text-based shaders which might restrict some of the artists from being able to make small adjustments to the shader. At the end of the discussion about this topic, all the interviewees claimed that one tool is not superior to the other but rather the choice of the tool comes down to the use case, requirements, personal experience, and preferences.

> *Text shaders are more performant in general, they have less limitations, but they require intention and only people that know how to write text shader can maintain a text shader.* (Cislaghi, 26 September 2024.)

> *It's different use case. So, you can't really say that OK, it's better. Node graph is better for certain use case and text is better for certain use case.* (Väisänen, 24 September 2024.)

The final theme identified from the interviews was *knowledge* about shaders and how knowledge can help with problem-solving. Shaders are known for their challenging nature due to their many complexities and limitations regarding development and sometimes the problems can be difficult to recognise. The interviewees were asked about their experiences with shaders that turned out to be challenging to produce and they shared their experiences and described some of their prior shader projects. It was mentioned that persistence and patience are the keys to shader development as sometimes solutions might be quite abstract and take a lot of time to find. The interviewees highlighted the importance of curiosity and experimentation to keep exploring new shaders and solutions.

> *Yeah. I mean, or you just discover it at some point, you just notice that it doesn't work and then you try to figure out why. Right. Like that's, that's how a lot of people start out. (Holmér, 19 September 2024.)*

> *I think I did like six different approaches for that thing. And all of them had their downsides and all of them had some kind of like upsides. Some of them didn't work at all. So it's lots of just trying out things and then you finally figure out something that is good enough to pull on with that.* (Väisänen, 24 September 2024.)

> *Yes if, if there's one thing that I can tell you about making shaders, making the effects, it's like caramel believe in it. It will turn out good. It goes into all the possible stages of horrible trash. […] And then at the end, if you stay, if you keep at it, it will come out ok.* (Cislaghi, 26 September 2024.)

Knowing mathematics is also a vital part of shader development as everything in shaders is generally done with mathematics. Helén emphasised the importance of mathematics. It was mentioned that regardless of the level of the effect, mathematical calculations or functions are always required to create a shader. When talking about how knowledgeable one should be in mathematics should one know when starting with shaders Helén mentioned that previous experience

with complex calculations is not necessarily required if one is passionate as the shaders will teach the calculations that are required. However, reaching more complex effects and solutions is faster with prior experience with mathematics.

> *Then like, and with the deeper level things as well, is like, the most important thing is to know the math. Because basically everything you do with shaders, like every effect, everything you ever do, is. It's all math. […] I think if you're passionate enough, you can learn the math because, for example, if you want to make some effect and you're really passionate about the effect, then creating the effect will teach you the math required if you're just persistent enough.* (Helén, 17 September 2024.)

7.3 Key Takeaways

The interviews brought up various important topics within the shader production pipelines and general knowledge that is often required to create shaders. Based on the results of the thematic analysis, the key takeaways were sorted under the same three themes identified from the interview data: limitations, production, and knowledge. The themes and topics were partially predefined in the interview questions (Appendix 1), but the final themes were reformed in the thematic analysis. The topics discussed in the key takeaways were always mentioned by multiple interviewees during the interview process and were especially highlighted by some of them.

Shader production is always going to be about setting priorities as it is often not possible to make a perfect shader. There are going to be trade-offs as the developer needs to carefully pick where to allocate the available resources. The limitations shape the shaders and shader pipelines, and the priorities often are set by either the stakeholders or employees in lead positions in a company setting. Everything in shader development depends on how vital the shader is for the game to function and how the shader can work around the limitations set by the requirements or priorities. One’s background in game development can also affect setting personal priorities for a shader asset.

Generally, iterative workflows are the key to shader development as there is no straightforward path from a concept to a finished shader asset. The amount of planning invested in shader development is quite personal as some might find it

easiest to start experimenting immediately whereas some might find comfort in doing some research and forming a solid concept first before starting the production process. There are no generalised pipelines or requirements for a shader asset as the process is heavily dependent on the limitations and priorities that shape the properties of the output. Usually, shaders are built for a specific purpose rather than as a complex shader system, but these solutions are also heavily dependent on the project and use case.

The tools and workflows used in shader production depend on the use case and personal preferences. Visual node-based shader tools are great for quick iterations due to their node-specific visual outputs and allow the user to do most of what can be done with shader code, but they tend to abstract and simplify away information that can be useful for the developer. Shader code on the other hand is fast to work with as the logic can often be more straightforward to read but writing code requires a lot more knowledge to understand. Neither tool is superior to the other. To work efficiently with different shader tools, it is important to ensure that the shaders are readable to make editing and adding features easier later in the production.

Knowledge plays a massive role in shader production as a developer needs to know various aspects to be able to create shaders in a video game environment. Knowing the fundamental limitations of shaders is important but through experience and experimentation, one can learn more about specific situations that can cause burdens on the development. Knowing shader quirks can help the developer avoid sub-optimal solutions early on to navigate around the shader limitations more efficiently. Limitations can sometimes inspire complex solutions, so it is important to know what to optimise if the set game performance is not achieved. Shaders can be optimised in multiple ways but there are no standardised optimisation techniques that apply to every shader. Individual shaders are hard to debug and detecting bottlenecks can at times be difficult. Knowing the mathematics behind shaders is vital as everything inside shaders

programs happens with a collection of mathematical functions. Mathematics is not a necessity to get started with shaders as the required functions are learned through experience, but it will help one achieve more complex results faster.

8 COMPARATIVE ANALYSIS

To ensure that the author's snow shader was of the required quality, a comparative analysis (Mello et al. 2021, 2) was performed. The data samples consisted of snow systems in chosen published games and the author's snow shader asset. The criteria for comparison were divided into two categories. The author evaluated the games by personally playing them and experimenting in the snowy areas.

When conducting data sampling, the author utilised the purposeful selection method when choosing which games' snow systems to compare (Mello et al. 2021, 25). This method allowed the author to pick the most relevant samples for the study. However, utilising purposeful selection was not an entirely reliable sampling method as it may have introduced bias. The author's subjective choices regarding the sampled data could have influenced the comparative analysis and its results.

Three different games were picked based on their overall style and scope, their year of publication, and the author's access to the game. The games chosen were Journey (2012), Super Mario Odyssey (2017), and God of War Ragnarök (2022). The availability of the games was a crucial factor as it was important for the author to play the selected games themselves. A video recording of the gameplay does not communicate the gameplay feel as clearly compared to experiencing it oneself. The games had different qualities regarding style and overall gameplay, but they all featured snow in their game worlds. They were also published in various years by different companies utilising various tools which ensured variety in the comparative analysis. The sampled games were analysed comparatively to the author's design choices regarding the snow shader asset produced during the thesis process.

The criteria for comparative analysis were initially to be set by the major topics or themes of the interview questions: *knowledge and experience, production methods, optimisation, and scalability* previously mentioned in Chapter 7. Based on the results of the interviews, shaders do not have standardised optimisation or scalability requirements. Therefore, those criteria were not considered in the comparative analysis. Moreover, due to the limited literature and publications of production details regarding the picked games, the criteria had to be limited to the features which the author could reliably evaluate. The picked criteria were visual aspects and gameplay impact. Because of the limitations, the author was not able to evaluate, for instance, the optimisation or production methods in the chosen games. However, the author could provide an overview of the gameplay implications and visual attributes in the snow systems of the compared data.

The comparison criteria were divided into two categories: visual aspects and gameplay impact. Each category features a set of qualities that were chosen based on the original design decisions of the author and some of the restrictions of the author's snow shader. In each category, the qualities were compared between the sampled games as well as the author's snow shader as shown in Table 1.

Table 1. Table of qualities observed during the comparative analysis

Project Information	Project / Game Title	**Journey**	**Super Mario Odyssey**	**God of War Ragnarök**	**Author's Snow Shader**
	Publishing Year	2012	2017	2022	2024
	Game Level Progression	Linear	Linear	Open world	Demo scene
Visual Aspects	Art Direction	Stylised	Stylised	Realistic	Stylised
	Saving	Distance-based	Distance-based	Area-based	Area-based
	Fidelity	Low poly	Low poly	High poly (tessellation)	Mid poly
	Snow type	Deformable	Decals & deformable	Deformable	Deformable
	Track forming based on	Trail (feet)	Footprints & trail (body)	Trail (body)	Trail (drawing)
Gameplay Impact	Affects the player's movement	x	-	-	x
	Affects the player's clothing	x	x	x	x
	Supports multiple characters	x	-	x	-

The goal of the comparative analysis was to determine how the author's snow shader compared to existing snow shaders which are featured in video games on the market. As the features included in the comparative analysis were set by the author's design decisions, it was evaluated if these features also appeared in commercial games and how they differed from one another. As the comparison was finalised, the author had an idea of the feasibility of the produced snow shader and how similar the approaches were to the compared games.

8.1 Visual Aspects

With snow, reoccurring visual aspects can frequently be observed in various snow systems. The snow is primarily defined by the game's art style as well as how the player interacts with it. Achieving great visuals can lead to sacrifices in the game performance but there are some common approaches to navigate limitations and achieve appealing results.

The visual style between the compared samples varied quite a lot. The author's snow shader (Figure 19d) was designed to take a stylised approach as it gave the author more room for experimentation rather than focusing on achieving perfect realism. Games like God of War Ragnarök (Figure 19c) achieved the goal of realism with their highly regarded art direction. The snow felt convincingly realistic to walk on as the tracks left behind produced a very convincing effect. However, a stylised approach of Journey (Figure 19a) yielded different results but did not take away from the gameplay feel of the snow. Journey achieved snow with a more simplistic approach, but it did not lessen the feeling one gets when first seeing the snow. Super Mario Odyssey (Figure 19b) much like Journey also had a very stylised approach to its graphics.

Figure 19. Comparison of the visual style in the compared samples

Weather in games can be meaningful aside from only providing the player familiarity and consistency to the real world. Weather conditions can be used as a method of visual storytelling as weather can depict tone and mood as well as imply previously occurred events (Barton 2008). In God of War Ragnarök (2022), some areas had bloody tracks in the snow to imply that an injured animal had been in the area (Figure 20). Such visual storytelling gave the player a physical trail to follow and locate the animal by utilising the tracks in the snow. This made the tracking feel more meaningful compared to solely relying on the dialogue.

Figure 20. Screenshot showcasing the visual storytelling in God of War Ragnarök (2022)

When inspecting the technicalities of constructing a snow shader, there are numerous methods available to produce tracks in the snow. Two methods were identified from the compared data based on the visual qualities of the snow: footprints made with decals and deformable snow with displacement maps. Decals utilise the technique of stamping footprint textures on top of the snow material that does not in most cases displace the surface as seen in some of the snow areas in Super Mario Odyssey (Figure 21b1). However, Super Mario Odyssey also featured deformable snow (Figure 21a1) in specific locations of the game world. Journey (Figure 21a) featured deformable snow on a larger area and God of War Ragnarök (Figure 21c) had a portion of the map covered in snow. The author's snow shader (Figure 21d) also featured deformable snow much like the one seen in Journey. However, due to time limitations, the snow shader only functions in a smaller rectangular demo scene rather than in a larger terrain-based area.

Figure 21. Comparison of the trails left in snow in the comparison samples

One of the bigger challenges of deformable snow is in saving the tracks. It can become expensive to save deformation information and therefore the snow seen in games often returns to its original state after a while. Journey (2012) and Super Mario Odyssey (2017) both had the tracks fade away some distance behind the player. In Journey, the fading of the tracks was less noticeable as the distance was greater than in Super Mario Odyssey where the decal-based tracks faded close behind the player. The deformable snow in Super Mario Odyssey,

however, was area-based and the tracks did not disappear until after leaving the area. This was most likely done as the deformable snow only functioned in a confined space.

In God of War Ragnarök (2022), the tracks were also based on the area the player was in. As the game is captured in a single take without cuts or loading screens the area separation is cleverly hidden. When moving through tighter spaces like caves the game loads the next area and most likely in this process, the previous area's snow trails disappear. The author's snow shader (2024) also utilised an area-based saving for the snow trails. As the shader was only functioning in a singular demo scene the area-based disappearing cannot be properly tested or showcased. However, it could be implemented when new areas are added.

One of the bigger hurdles when deforming a surface with a shader is the number of polygons in the terrain as it affects the quality of the displacement. By only utilising a vertex shader, great-looking results are hard to achieve as the polygon count of the terrain would have to be very high to create convincing effects. High polygon counts however are not great in terms of game performance. The effect of using a lower polycount in the mesh can be seen in Super Mario Odyssey (Figure 22b) as the snow appeared very sharp and unnatural. As there were no resources on how the snow in Journey (Figure 22a) was done the author could not define the exact production method or fidelity by observing the snow. It was noted that the method might not be tessellation as when the character is thrown to the ground by an enemy, the intense collision can cause sharp artefacts in the snow shader as seen in Figure 23. However, the possibility of such sharp edges appearing was low.

Figure 22. Comparison of the fidelity of snow in the compared samples

Figure 23. Screenshot of Journey (2012) showcasing snow fidelity

God of War Ragnarök (2022) utilised tessellation shaders in its snow system to ensure better game performance while also achieving convincing results (Sergeev 2023). Through tessellation, it was possible to add more geometry only to the areas where the snow would be displaced therefore ensuring a better game performance. On top of this, the over-the-shoulder camera was positioned in a way where the player does not often see the character's feet. It was possible to see the deformation of the snow while walking backwards and turning the camera to be top-down. Therefore, it can be assumed that these were most likely creative decisions to ensure possible mishaps would not happen in plain sight.

Due to Godot not supporting tessellation shaders the author's snow shader (2024) relied on a higher polycount of the ground mesh while using a vertex shader to handle the deformation. In the context of the demo scene, it was acceptable to utilise this but in a bigger game world, the method would prove to be quite unoptimised. Through the higher visual fidelity, the shader produced visually appealing results as seen in Figure 24. As a trade-off, the shader relies on a lesser priority on performance which could become an issue later in the production if the shader was utilised in a full game.

Figure 24. A screenshot showing the snow fidelity in the author's snow shader

To summarise the observations on the visual aspects of snow in the compared samples it was noted that there were multiple different methods of achieving snow. In the compared samples, snow was depicted through different art styles but features such as the player leaving tracks on the snow often appeared to be a standard feature. It can be assumed that this is often done due to better grounding the player into the game world and improving immersion. It turned out that snow and weather elements can also be used as a method of visual storytelling as they can define previous events that occurred in an area.

Leaving tracks in the snow in the compared samples was achieved most likely through two different methods identified based on visual qualities: decals and displacement. Both methods have their trade-offs, and the choice of method can vary depending on the style and platform limitations. Deformable snow requires a larger focus on optimisation as displacing terrain with a lower polycount on the 3D mesh can easily produce very sharp and unappealing results. Defining the method and requirements of the snow shader closely ties back to navigating limitations and setting the priorities and requirements for the outcome of the project. The author’s snow shader seemed similar to a few of the sampled games’ approaches and showcased a personalised visual style. The author’s snow shader similarly to the other games also deforms the snow leaving behind a trail into the mid-poly terrain. However, due to the limitations and time constraints, the author's snow shader did not reflect functionality in the context of a whole game due to it only being showcased in a demo scene.

8.2 Gameplay Impact

Snow in games can sometimes play a bigger role than being only a visual aspect. In some use cases, visual effects can be utilised in level design and even as a core gameplay mechanic (Veselinovijk 2024). The environment affecting the gameplay is not unusual but there are some common features often seen in games which feature snow. However, all the features are not standardised essentials but rather a part of a series of design decisions which help determine the gameplay feel.

When playing and comparing the sampled games, only a few of them had the snow affect the player character's movement. In the real world walking in deep snow is quite taxing but in the virtual worlds, the characters seemed to walk through the snow with ease. In the author’s snow shader, the player’s movement is heavily influenced by the depth of the snow as the character can only walk in deeper snow but easily run at flattened areas. This design decision was made early to experiment how this feature affects interactivity as well as to compare if it appeared in published games.

In God of War Ragnarök (2022) and Super Mario Odyssey (2017), the snow did not affect the player's movement. Especially in God of War Ragnarök, the layer of snow was not perceivably deep, so it seemed natural that the characters were able to walk in it quite effortlessly. In Super Mario Odyssey, however, the player was able to travel through piles of snow four times the character's height with ease. Although in Journey (2012) the layer of snow was not too deep, while in snow, the main character was not able to perform sliding or jumping actions that were previously possible. When spending time traversing through the snowy area in Journey, the character got visibly slower and more tired when progressing through the area. Such a feature differed from the rest of the games compared.

In the real world, falling snow tends to stick to clothing, which happened to all the characters in the compared sampled, although there were stylistic differences. In the more stylistic games such as Journey (Figure 25a), Super Mario Odyssey (Figure 25b) and the author's snow shader (Figure 25d) the snow stuck more as an overlay rather than following the actual collisions with the snow. Whereas in God of War Ragnarök (Figure 25c) possibly due to a more realistic approach snow only stuck to the clothing that touched it as seen in the boots or main characters' cape when rolling to dodge attacks.

Figure 25. Comparison of snow sticking to the character's clothing in the compared samples

Due to shader limitations, supporting multiple characters or interactive objects within one shader system can at times be challenging. The support for multiple interactions exists depending on whether the game is meant to be played alone

or with other people. However, the interactions are not only limited to players as single-player games can have numerous non-playable characters (NPC) that also interact with the environment. In Super Mario Odyssey (Figure 26b) no other interactions aside from playable character were identified. Most of the other characters in the snow region were either stationary or did not leave footprints in the snow. However, it is important to mention that only a portion of the snow level was played so this might not be true for the entire snow-covered area.

Journey (Figure 26a) and God of War Ragnarök (Figure 26c) supported multiple characters. Journey is technically a multiplayer game as one can share their journey with another player whom they cannot communicate with. God of War Ragnarök on the other hand, is a fully single-player experience but has numerous NPCs and companion characters that also interact with the snow similarly to the player. The author's snow shader (Figure 26d) currently only supports one character but due to the chosen production method and its support of dynamic interaction counts, it is easy to add multiple character interactivity.

Figure 26. Comparison of the snow shaders supporting multiple characters

To summarise the finds from the gameplay impact category, it was noted that all the sampled games supported the compared features differently. The stylisation of the game seems to have a considerable impact on the snow's effect on the gameplay as the target is not realism but rather a more tailored experience. According to Barton (2008), many developers believe that immersion

is best achieved with realism as the games can feel more familiar compared to abstract graphics. Whether snow restricts the player's movement seemed to be a conscious choice made to fit the mechanics for the desired player experience.

Some aspects such as snow sticking to characters or clothing seemed to be a feature that is so closely associated with snow in the real world, that the experience would feel odd without it. The support of multiple characters simultaneously seemed to depend on the game as well as the desired style. Characters leaving tracks in the snow was a feature that closely grounds them to the game world and therefore influences gameplay feel and immersion. The author’s snow shader seemed to compare rather well to the sampled games as it supported most of the features often seen in snow systems (Table 1). Overall, in games, the choice of how the snow affects player movement is tailored according to the playable world and allows for a unique gameplay feel. The author’s snow shader featured a system for limiting the player's movement to showcase the interactivity as well as create a unique gameplay feel. The shader system also impacts the player by having the snow stick to the character when in deeper snow. Although the shader does not currently feature the support for multiple characters it can be easily added due to the flexibility of the production method.

9 FINAL SHADER ASSET

As the practical implementation of the thesis, the author aimed to produce an interactive snow shader made in the Godot game engine. The shader asset functioned as a base for applying the learning outcomes as well as showcasing the results of the qualitative research. The author succeeded in constructing a viable snow shader asset which includes the designed features previously stated in Chapter 6.1. However, as the shader is functional, it is not yet complete as the author aims to improve the result in the future.

The main interactions were achieved with a viewport texture rendered through a sub-viewport while the game is running (Figure 27). The viewport texture was used as the displacement mask for the tracks in the snow material as well as the indicator of whether to switch player movement to a different state. The viewport

reads the player's position and draws on the texture by marking the player’s location utilising a custom sprite. Without clearing the viewport texture on the next frame, the previous marks remain, drawing a pattern on the texture. The viewport texture is not visible to the main camera and cannot be seen in the game world. The snow material then displaces and blends the bottom layer (Figure 28) based on masking by the alpha value in the viewport texture.

Simultaneously the player controller receives the alpha value of the pixel in the detection position and determines whether the player is walking in deep snow (white area) or in a flat area (black area). The movement state in the player controller is adjusted if a certain opacity threshold is crossed. The states then control the player’s locomotion and animation tree. To view the changes in the movement and animation, a showcase and breakdown of the snow system are presented in the form of a video (Kataeart 2024) that demonstrates the functionality and features of the snow system. The video provides a better look at the player's movement system and the deformation of the snow.

Figure 27. The viewport texture drawing in the snow shader demo

The opacity of the drawing sprite is adjusted to allow the player to walk through the snow a few times before fully pressing it down as shown in Figure 28. This allows for a more realistic approach as the snow is pushed down gradually rather than instantly. The randomness of the trail is achieved through UV distortion on

the viewport texture. It takes the UV map of the terrain and distorts it based on a noise map and customisable parameters. A slight colour tint (Figure 28a) was added and mixed with a dirt blend (Figure 28b) allowing parts of the ground to be visible beneath the snow. The colour tints also make the trail more distinct from the untouched snow. The snow material features multiple adjustable parameters which control the visual features as well as the height and displacement values of the snow allowing for high customisability and accessibility through the inspector.

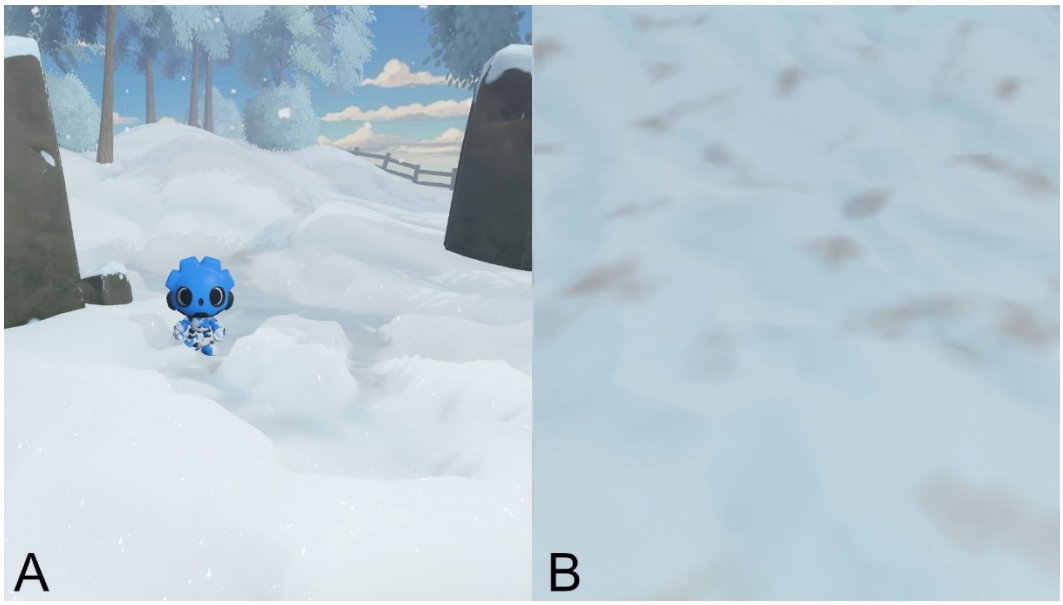

Figure 28. A, The colour tint of the bottom layer. B, The dirt blend of the bottom layer

The snow shader is showcased in a small rectangular demo area (Figure 29) which has a set dressing with simple props to form a little environment to explore. Some materials, the player character, audio, and a portion of the props like the stones and fences are not made by the author. Instead, they are downloaded from the internet under the Creative Commons 0 licence meaning that they are free to utilise and modify both in commercial and non-commercial projects. The full list of credits and contributions to the project can be found in Appendix 6.

Figure 29. Snow shader demo scene from above.

Finally, if the player proceeds to walk in deep snow, some snow sticks to the character as seen in Figure 30a. As the player walks into an area with less snow, the snow on the character's body eventually fades away revealing the default texture of the character as shown in Figure 30b. The change in the character's body is done with a simple customisable shader which adds the snow material on top of the player's texture masked with a noise map. The player shader receives information from the movement state, adjusting the snow's opacity on the character with fade-in and fade-out effects based on a timer and current state.

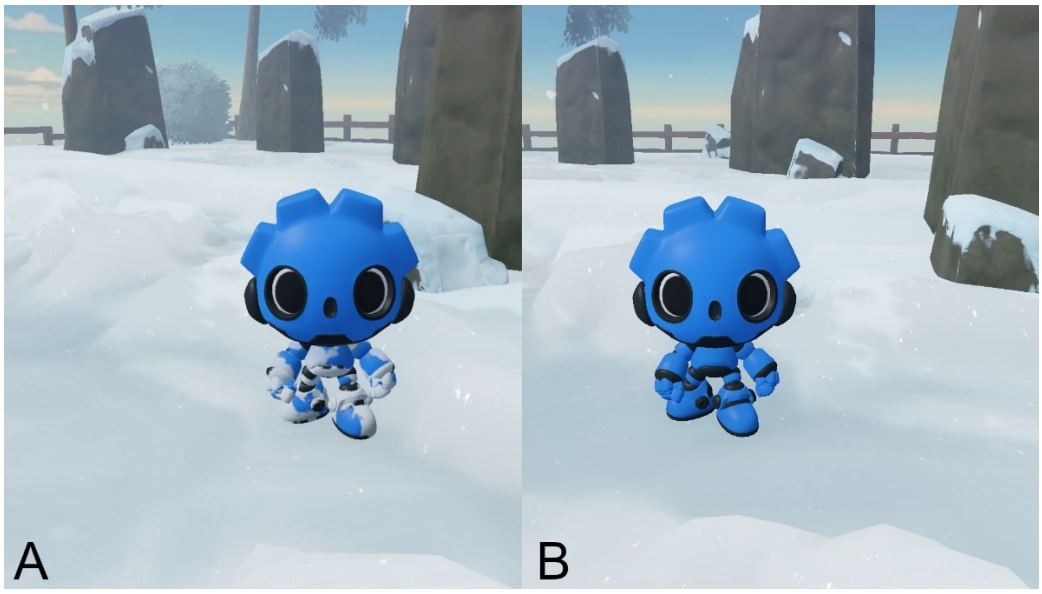

Figure 30. A, Snow sticking to the player. B, the player with default textures

The snow shader is ready for initial showcasing but is still far from complete. The author wants to add support for multiple characters which due to the viewport texture drawing method is simple to implement. Other additions the author is thinking of adding are better scalability and recognising alternative surface materials. As the demo scene is currently a small rectangular area, the shader does not demonstrate the aspect of scalability. In the future, the author aims to construct a system for larger terrain, but it would feature more design complexities and limitations to be solved. As of now, the whole terrain is read as snow excluding the path the player has already walked. The addition of material detection would allow lakes and snow-free areas to be included in the environment. For easier implementation of different material areas, the author brainstormed the idea of building an in-engine vertex paint tool. It is safe to say that the final version of the snow shader for the thesis has produced great results, but future additions are going to be made once the thesis process has concluded.

10 CONCLUSION

This study was conducted to understand the fundamentals of shader production and acquire practical skills to create dynamic shaders with shader-player interactions for virtual game worlds. The study aimed to answer the question of “How are dynamic shader-player interactions created in a game production environment?” as well as define whether there is an industry standard for shader production. The process featured the production of a dynamic snow shader asset that showcases shaders’ interactive qualities discussed in the paper. To ensure the quality of the produced asset, methods of qualitative research were utilised. The research featured interviews with professionals and a student from the game industry as well as comparative research of functionalities and aesthetics of snow from a selection of video games and the author’s snow shader.

By conducting wide-scale information acquisition, the author formed a thorough understanding of the basics of shader theory and the practicalities of shader production. The author was able to distinguish differences between the common

shader types and define what role each type plays in the shader rendering pipeline. Different shader workflows were also explored such as visual node-based tools now provided by various game engines as well as traditional shading languages.

The topic of shader-player interactivity was explored in the bounds of the available sources. Shader-player interactivity lacks a defined concept and therefore no academic sources were found. However, the author was able to acquire some sources to define the relationships between the player and the environment. Different interactions and roles regarding VFX were also explored.

As a part of the qualitative research, the author organised semi-structured interviews with a student and professionals from the game industry. The author interviewed four individuals from different backgrounds, acquiring valuable knowledge and insights into shader workflows. Based on the interview data the author found that shader interactivity includes limitations which can cause design complexities. It turned out that shader production heavily relies on priorities and trade-offs are common as a higher focus on visual fidelity can result in a lower game performance. Shaders do not have standardised optimisation or scalability requirements as shader creation depends on the priorities of the project and demands set by the target platform. The author also found that visual shader tools and traditional shader programming are not superior to one another. Instead, they are suitable for certain use cases and the choice can depend on the user’s personal preferences. Knowing shader limitations and shader production takes time but through curiosity, experimentation, persistence and fun one can learn the intricacies and mathematics behind shaders.

The author also conducted a practical comparative analysis to study the differences between the produced snow shader asset and other snow systems in commercial games. A purposeful selection method was utilised when choosing the comparison samples with the best relevance. However, due to the sampling method as well as the limited source material on the production of the sampled games, the comparative analysis was limited to comparing visual aspects and

gameplay impacts of snow systems. When comparing visual aspects, the author recognised differences between the sampled games' styles and interactions with the player. Aspects like trail type, trail saving and snow fidelity were compared among the sampled games and the author's snow shader. Regarding the gameplay impact, the author compared the snow's effect on player movement, player's clothing and explored whether the snow supported multiple characters. The author found that their snow shader asset compared to the commercial games on a similar level. Most of the choices regarding these features likely surface from the limitations and priorities set during production.

Throughout the thesis process, the author studied the practicalities of constructing shaders in the Godot game engine. The author completed smaller experiments before starting the practical project of the thesis. The author aimed to demonstrate a mutual interaction between the player and the environment. Therefore, the snow shader slows the player's movement but also leaves a trail of displaced snow. Aside from this, the snow also sticks to the player on interaction. The author successfully created a small demo environment where the snow shader functions as designed. The shader at its current state does not demonstrate scalability or optimisation but is a viable shader asset which with further additions could be used in a video game.

The study was limited by some factors during the thesis process. The lack of sources on player-shader interactions resulted in the author defining some concepts with an absence of academic sources. Additionally, the scarcity of available sources regarding snow production of the compared games in the comparative analysis narrowed the criteria. Due to said limitations, the author could not fully define the quality of the produced snow shader asset. Additionally, the author's sampled data in the comparative analysis and the number of interviewees were limited. The author was still satisfied with the produced results. Nevertheless, the thesis serves as a relevant study on shader interactivity as prior research was not found. It functions as a valuable starting point for approaching shader interactivity and shader workflows in the game industry.

The thesis successfully addresses the main research question through the practical project largely supported by research. Through theoretical and practical information acquisition as well as insights from the interview data, the author succeeded in creating a practical project that demonstrates how shader interactions are created. The secondary questions regarding the industry standards and achieving visually appealing yet professional results were answered largely in the interviews. It was found that there are no defined industry standards for shader production as everything is dependent on the use case and project priorities. Professional and aesthetic results can be achieved through experience which is gained through experimentation, curiosity and fun. Great results require recognition and knowledge of shader concepts but might occasionally demand sacrifices in the game performance.

For future research, many aspects of the study can be improved. The shader-player interactions' impact on immersion could be studied more by conducting A/B testing with variations of interaction systems. This could provide more data on how players can feel more grounded in the game world and how much their actions can affect their surroundings. With a bigger scope, the study could be expanded to other interactive elements seen in video games instead of only focusing on snow. Aside from the research methods, more information acquisition could be done or in the case of lacking information more viable solutions could be formed over a longer timeframe.

LIST OF REFERENCES

Albukhari, A. 2020. Godot Tutorial – Dynamic Snow. YouTube. Video clip. 18 April 2020. Available at: https://www.youtube.com/watch?v=F_KZyc_kLYs&list=PLK8uhHZ8rvj-4IH0UTXyRkGhw5sD-X4oi&index=28 [Accessed 26 October 2024].

Bailey, M & Cunningham, S. 2012. Graphics Shaders: Theory and Practice. 2nd ed. Boca Raton: CRC Press.

Barre-Brisebois, C. 2017. Deformable Snow Rendering in Batman: Arkham Origins. YouTube. Video clip. 6 May 2017. Available at: https://www.youtube.com/watch?v=87rg95XBaIE&list=WL&index=8&t=6s [Accessed 21 October 2024].

Barton, M. 2008. How's the Weather: Simulating Weather in Virtual Environments. *Game Studies*, 8 (1), E-journal. Available at: https://gamestudies.org/0801/articles/barton [Accessed 7 August 2024].

Borromeo, N. A. 2021. Hands-on Unity 2021 game development: create, customize, and optimize your own professional games from scratch with Unity 2021. 2nd ed. Birmingham: Packt Publishing.

Claufield, J. 2019. How to Do Thematic Analysis, Step-by-Step Guide & Examples. Web page. Available at: https://www.scribbr.com/methodology/thematic-analysis/ [Accessed 2 December 2024].

Crosley, J. 2021. What is Thematic Analysis?. Web page. Available at: https://gradcoach.com/what-is-thematic-analysis/ [Accessed 2 December 2024].

Donald, M. 2021. Christmas special, making it snow. YouTube. Video clip. 8 January 2020. Available at: https://www.youtube.com/watch?v=F_KZyc_kLYs&list=PLK8uhHZ8rvj-4IH0UTXyRkGhw5sD-X4oi&index=28 [Accessed 26 October 2024].

Edwards, J. 2018. Sand Rendering in Journey. YouTube. Video clip. 25 February 2018. Available at: https://www.youtube.com/watch?v=wt2yYnBRD3U&list=WL&index=7 [Accessed 21 October 2024].

Endoflife.date. 2024. Godot. Web page. Available at: https://endoflife.date/godot [Accessed 3 September 2024].

Espindola, F. 2021. The Unity Shaders Bible. Chile: Jetterly. E-book. Available at: https://www.jettelly.com/store/books/the-unity-shaders-bible/ [Accessed 9 September 2024].

G2A. 2024. OpenGL vs DirectX: A Guide for Gamers. Web page. Available at: https://www.g2a.com/news/features/opengl-vs-directx-guide/ [Accessed 11 September 2024].

Galvan, A. 2022. A Review of Shader Languages. Blog. 13 February 2024. Available at: https://alain.xyz/blog/a-review-of-shader-languages [Accessed 8 August 2024].

Gamedev.tv. 2024. Godot 4 Shaders: Craft Stunning Visuals. Udemy. Online course. Available at: https://www.udemy.com/course/godot-4-shaders/?couponCode=JUST4U02223 [Accessed 26 October 2024].

Garbett, S. L. 2022. OpenGL vs. DirectX: Which Should You Use for Game Development?. Web page. Available at: https://www.makeuseof.com/opengl-vs-directx-game-development-best/ [Accessed 11 September 2024].

Godot. 2024. Godot Engine. Web page. Available at: https://godotengine.org/ [Accessed 3 September 2024].

Godot Docs. 2024a. Introduction to shaders. Web page. Available at: https://docs.godotengine.org/en/stable/tutorials/shaders/introduction_to_shaders.html [Accessed 3 September 2024].

Godot Docs. 2024b. Shading language. Web page. Available at: https://docs.godotengine.org/en/stable/tutorials/shaders/shader_reference/shading_language.html#doc-shading-language [Accessed 3 September 2024].

Godot Docs. 2024c. Using VisualShaders. Web page. Available at: https://docs.godotengine.org/en/stable/tutorials/shaders/visual_shaders.html [Accessed 12 August 2024].

Godot Docs. 2024d. Using Compute Shaders. Web page. Available at: https://docs.godotengine.org/en/stable/tutorials/shaders/compute_shaders.html [Accessed 2024].

Godot Shaders. 2024. Godot Shaders. Web page. Available at: https://godotshaders.com/ [Accessed 12 November 2024].

Hergaarden, M. 2011. Graphic Shaders. Web page. Available at: https://www.cs.vu.nl/~eliens/download/literatuur-shaders.pdf [Accessed 8 August 2024].

Hirsjärvi, S., Remes, P. & Sajavaara, P. 2009. Tutki ja kirjoita. Helsinki: Tammi.

Holmér, F. 2021. Shader Basics, Blending & Textures • Shaders for Game Devs [Part 1]. YouTube. Video clip. 26 February 2021. Available at: https://www.youtube.com/watch?v=kfM-yu0iQBk&t=2896s [Accessed 11 September 2024].

Kataeart. 2024. Bachelor's thesis 2024 // Interactive Snow Shader Showcase. YouTube. Video clip. 14 November 2024. Available at: https://youtu.be/QlodncUcfAA [Accessed 14 November 2024].

Magic Media. 2023. The Ultimate Guide to VFX for Gaming: From Explosions to Environments. Web page. Available at: https://magicmedia.studio/news-insights/guide-to-vfx-for-gaming/ [Accessed 3 September 2024].

Mello, P. A., Brockhaus, M., Ide, T., Anderson, M. A., Harkness, S. K., Swinkels, M., Giordono, L., Boudet, H., Gard-Murray, A. & Castillo-Ortiz, P. 2021. Qualitative Comparative Analysis: An Introduction to Research Design and Application. Washington, D.C.: Georgetown University Press. E-book. Available at: https://ebookcentral.proquest.com/lib/xamk-ebooks/reader.action?docID=31308801&ppg=35 [Accessed 7 November 2024].

Microsoft Learn. 2022. Graphics pipeline. Web page. Available at: https://learn.microsoft.com/en-us/windows/uwp/graphics-concepts/graphics-pipeline [Accessed 18 September 2024].

Nintendo. 2007. Super Mario Galaxy. Video game. Kyoto: Nintendo.

Nintendo. 2017. Super Mario Odyssey. Video game. Kyoto: Nintendo.

OpenGL. 2022. Rendering Pipeline Overview. Web page. Available at: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview [Accessed 9 September 2024].

OpenGL. 2019. Compute shader. Web page. Available at: https://www.khronos.org/opengl/wiki/Compute_Shader [Accessed 11 September 2024].

OpenGL. 2006. Shading Languages: General. Web page. Available at: https://www.khronos.org/opengl/wiki/Shading_languages:_General [Accessed 2 October 2024].

Polycount. 2020. Shaders. Web page. Available at: http://wiki.polycount.com/wiki/Shaders [Accessed 9 September 2024].

Polycount. 2014. Shaders for artists. Web page. Available at: http://wiki.polycount.com/wiki/Shaders_for_Artists [Accessed 9 September 2024].

Procyon Products. 2003. Pro Pilkki 2. Video game. Helsinki: Procyon Products.

Rockstar Games. 2018. Red Dead Redemption 2. Video game. New York City: Rockstar Games.

Sergeev, A. 2023. Santa Monica's Senior Programmer on How God of War Ragnarök's Snow System Was Made. *80.lv.* 16 October 2023. Digital Content Publishing Network. Available at: https://80.lv/articles/santa-monica-s-senior-

programmer-on-how-god-of-war-ragnar-k-s-snow-system-was-made/ [Accessed 9 November 2024].

Stitching Godot. 2024. Godot Foundation. Web page. Available at: https://godot.foundation/ [Accessed 3 September 2024].

Sony Computer Entertainment. 2012. Journey. Video game. San Mateo: Sony Computer Entertainment.

Sony Interactive Entertainment. 2022. God of War Ragnarök. Video game. San Mateo: Sony Interactive Entertainment.

The Pokémon Company, Nintendo. 2021. Pokémon Brilliant Diamond. Video game. Tokyo: The Pokémon Company, Kyoto: Nintendo.

Tkachenko, K. 2024. Godot Engine's Rise to Prominence in 2024: Revolutionizing Mobile Game Development. Web page. Available at: https://www.gamelight.io/post/godot-engine-s-rise-to-prominence-in-2024-revolutionizing-mobile-game-development [Accessed 14 November 2024].

Unity. 2024a. Compute shaders. Web page. Available at: https://docs-alpha.unity3d.com/Manual/class-ComputeShader.html [Accessed 11 September 2024].

Unity. 2024b. Using Shader Graph. Web page. Available at: https://docs.unity3d.com/Manual/shader-graph.html [Accessed 2 October 2024].

Veselinovikj, B. 2024. VFX in Gaming: The Ultimate Guide to Visual Effects in Video Games. Web page. Available at: https://borisfx.com/blog/vfx-in-gaming-ultimate-guide-visual-effects-games/#role-of-vfx-in-gaming [Accessed 3 September 2024].

Vivo, P.G. & Lowe, J. 2015. Book of Shaders. Web page. Available at: https://thebookofshaders.com/01/ [Accessed 9 September 2024].

Warner Bros. Interactive Entertainment. 2013. Batman: Arkham Origins. Video game. Burbank: Warner Bros. Interactive Entertainment.

LIST OF FIGURES

LIST OF TABLES

Appendix 1/1

INTERVIEW QUESTIONS

1. Can you tell me a little about your background in the game development field and in shaders?
    - What's your current role in your current project?
2. Have you worked with / developed shaders that have shader-player interactions where both the shader and player affect each other in some way - for example, snow, dirt, etc.?
    - How did you approach such shaders?
    - What were the key takeaways?
3. What do you think is considered the highest priority when developing a shader effect for a game - is it the visual output, the technicalities such as production method and optimisation or the combination of both?
    - Can you very briefly describe your pipeline from planning to production?
4. Are there any standardised or general requirements for an optimised shader asset aside from the game running at 60 fps?
5. Should shaders be made into being scalable and reusable?
    - If yes what would be a solid foundation for a shader asset that supports scalability and iterations?
    - Towards what platform or platforms should they be developed?
6. Is there a significant difference in using shader code or visual shader graphs when it comes to industry standards?
    - Do you think one is superior to the other at the moment?
7. What has been the most challenging shader you have worked on?
    - What made it so challenging?
    - How did you overcome the challenge?
8. What's your favourite shader you have worked on?
9. Do you have any tips or recommendations on what to focus on for an individual who wants to get to work shaders and VFX in the game industry?
10. Is there anything else you would like to add to the topic that could be useful to mention?

Appendix 2/1

SUMMARY OF THE INTERVIEW WITH JONI HELÉN

Joni Helén is a final-year student studying game programming. He does general-purpose programming but has picked up an interest in shaders and has been developing them for over two years. Most of his experience in shaders comes from experimentation and personal projects and he aims to keep developing shaders in the future.

Helén has some experience with shader-player interactions but has mainly been doing them as an experiment. He describes some of the experiments which he has done featuring shader interactivity. He mentions that they are difficult to execute well as they require the developer to think about the interactions and be knowledgeable of multiple factors before the number of interactions happens. Dynamic changes in the count of interactions can create design complexities that one has to consider when creating such effects. Helén mentions different takeaways he has acquired from his projects and mentions some methods such as having specified buffers or structs to approach limitations present in interactive shaders.

Helén comes from a programming background and states that when working on a new shader his first priority is to define what is going to happen and how it is going to be done. He mentions that limitations are always going to be present, so he tries to solve many of them early. However, most of the time compromises must be made later in the production process. He also adds that after coming up with the first possible solution, he thinks whether there is a more efficient way to achieve the same result. According to him, if an approach seems unoptimised from the start, it is not worth considering and is discarded and another method is tried. In other terms, his workflow is very iterative. He also mentions that if at the end of the process, the game performance is not great, he can still proceed to strip features or compress textures to make the shader more performant.

When discussing shader optimisation methods Helén states that in his opinion there are no standardised optimisation techniques but rather some processes to avoid when working with the limitations of shaders. Helén mentions minimising the use of transparency as one can easily cause overdraw that negatively affects the performance. He also mentions the practice of caching as one can avoid repeating the same calculations on each frame. However, he also states that caching usually adds complexity to the shader programs. Helén says that due to the shaders' abstract nature, there is no standardised list of optimisation techniques. One must try to find the best approaches that function for specific use cases and contexts.

As shader scalability and reusability are closely tied to optimisation, Helén mentions the use of shader libraries and shader variants. With these, one can create the inputs and outputs of a shader so that the shader variant can be customised to the specific requirements. This allows the user to define the properties of the shader. Helén adds that shader libraries can be an efficient way to add reusability as they can be translated to multiple shading languages as well as made to function on different platforms.

Talking about the differences between text-based shaders and visual node graph tools, Helén mentions that node-based tools are a good choice when one needs to create something quickly and efficiently. According to him, the graph tools in Unity do not need extra setup with the rendering pipeline like text-based shaders often do. Helén adds that sometimes text-based shaders are so complicated to write manually due to them requiring support for all the features of the rendering pipeline that it is not worth trying. He thinks that node-based tools can act as an easy way to approach shaders, but they do not provide the user with all the features that shaders can offer. Additionally, the compilers for the shader programs can make poor compilations as they need to be extremely fast. In some cases, it can be useful to write shader code by hand to be more concise. Helén does not think one workflow is superior to the other but rather the choice of the tool depends on different factors and requirements of the project.

Coming from a programming-oriented background, Helén experiments with more complex and technical shader concepts like ray marching. He mentions a couple of challenging projects he has worked on over the years, and how he has managed to solve some of the challenges and limitations. According to him, most of the problems he faces relate to the limitations of shaders. Helén aims to find efficient solutions to these problems. He finds these technical challenges interesting and describes a project that he has enjoyed the most in which he got to experiment with ray marching as he made a lava lamp effect inside another object in the game world.

Helén states that there is no shader one can write without using mathematics. He emphasises the importance of knowing the mathematics behind shaders but mentions that one does not need to the intricacies of graphics hardware. This is only needed while trying to create something that works on a low level, close to the graphics hardware. In his opinion, if one is passionate about learning shaders and persistently experimenting with various effects, the effects will teach the required skills in mathematics. However, knowing mathematical concepts beforehand will help reach more complex solutions quicker.

Appendix 3/1

SUMMARY OF THE INTERVIEW WITH FREYA HOLMÉR

Freya Holmér is a technical artist and online educator with a background in 3D modelling and level art. She has worked in the game industry creating various shader effects for games but now focuses on tool development and is currently working on a 3D modelling tool built inside the Unity game engine. She is also running her own YouTube channel where she shares educational videos and lectures about technical art topics and game development math.

Holmér mentions that she does not have extensive experience with spatial shaders which have shader-player interactivity. However, she has been experimenting with shaders that react to player input. According to her some of the problems regarding shader-player interactions in shaders can be outside of the practices one usually does when working with shaders for visual effects. She mentions that limitations often emerge from how the data is stored and emphasises that shaders become gradually more complex as they are scaled outside of a ten-by-ten rectangle. She adds that various approaches to shader limitations feature tedious work when it comes to what a technical artist does.

Holmér mentions that she personally tends to prioritise the visuals when working on a new shader asset. She likes to keep her workflows very iterative and flexible as the desired outcome is not always clear. Therefore, she has the chance to try different approaches and optimise later in production. She does not prefer planning largely ahead of time as shaders are generally self-contained. If working on something interactive, she likes to keep the time and distance between editing and testing as short as possible. Holmér also highlights the importance of considering priorities based on the target platform as platforms like virtual reality devices or mobile require heavier optimisation than PC and consoles. She emphasises that although she prefers to work with a priority on visual results, it is always very important to be mindful of the performance. In addition, it was discussed that the background of the developer coming from an artistic or technical background can also affect the priorities when making shaders.

When talking about shader optimisation, Holmér mentions that there are some common practices to avoid when thinking about approaches. She mentions that features like geometry shaders can allow a great amount of flexibility but are generally very expensive to use performance-wise. In addition, she mentions the depth buffer and how it can affect transparent objects in a game scene. She highlights how knowing limitations and learning which practices are sufficient comes with practice and discovery but can at times feel frustrating.

Going from optimising individual shaders to shader scalability and reusability, Holmér does not think there is a single good answer to it. She describes how in some cases the developers would want a custom shader for everything but in most projects, the developers tend to have a master shader that maintains the main shading of the game. She says that this approach is often more performant as more objects can be batched together in a single draw call. Holmér mentions that a shader can be very specialised in its purpose, but it is important to ensure readability in case some modifications or additions need to be made later in the production. She also mentions that it is common to take aspects of a shader and transfer them to another shader, but this modularisation does not often need to be thought ahead of time.

Regarding differences in text-based shaders and visual tools, Holmér does not think that there are many differences between the workflows on the artistic side. When it comes to technicalities, the node-based shaders tend to abstract away some information that in Holmér's opinion can lead to not recognising the separation between vertex and fragment shaders. She likes how the node-based tools are visual which allows users to see the output even inside specific nodes in the node network. She highlights that visual feedback is very useful for beginners who are still learning shaders. However, node tools do not give access to all the information that the text-based shaders do such as compute shaders in Unity. She mentions that as one acquires a better understanding of the technicalities of shaders, text-based shaders open more features and possibilities than node-based tools. She emphasises that in Unity text-based shaders include tedious

manual tasks that have to be written by hand to support all features of the rendering pipelines. Holmér does not think one workflow is superior to one another but rather it is about trade-offs. Choosing the right workflow depends on the priorities, goals, experience and complexity of the rendering pipeline.

Holmér describes herself as a person who loves challenges, especially if they involve mathematics. She describes how for her the most tedious tasks are the boring and repetitive tasks that do not allow for artistic freedom. She also mentions how shaders, due to their limitations, are difficult to troubleshoot if problems arise. According to her, it can be very challenging to locate the issue that causes problems and finding the solution might sometimes take days. She mentions that these problems are often frustrating as the solution is usually quite simple. She describes the kind of projects she enjoys and expresses her interest in procedural workflows as well.

Holmér emphasises the importance of focusing on learning shaders in the game development context as shaders can also be produced for uses outside of video games. She mentions that if one wants to work in the game industry it is important to focus on learning and developing game-ready shader assets. They should be developed with the intent of being a part of a game to acquire an understanding of the practicalities of shader development in the game development context. She also highlights optimisation. According to her valuable knowledge can be acquired by thinking about how to make the shaders viable and performant on different platforms.

Appendix 4/1

SUMMARY OF THE INTERVIEW WITH TEEMU VÄISÄNEN

Teemu Väisänen is a game development technical director at a mobile game company and has a background in 3D modelling. He has worked for various companies in different roles and throughout his career he has had a growing interest in building a bridge between programmers and artists. His goal has been to understand the needs of the team and to ensure that the artist's vision is carried through in the production. In his current job title, he does not craft individual shaders but rather focuses more on pipelines and production.

Väisänen has experience working with shader-player interactivity and he describes a couple of his previous projects and how the dynamic shader effects for those projects were created. He says that interactive shaders are quite difficult to execute well due to limitations. For instance, the developer must know details like the number of interactions beforehand and sometimes that is not possible like in the cases of dynamic player counts. He adds that he does not think there is a good solution to these limitations but rather emphasises the importance of planning. In addition, he mentions learning to know the common limitations and possible approaches to navigate around them.

Regarding priorities when developing shaders Väisänen talks about the importance of listening to one's team and catering to the needs of the artists and their vision. He adds that sometimes he has to make compromises and try to find an outcome that pleases everyone as some effects requested by the team would not be possible to create within the boundaries of the limitations or performance requirements. Väisänen describes his own shader production pipeline to be very iterative where each step hopefully takes him closer to the end goal. He tends to focus on trying to create a result that is as close as possible to the artists' original vision of the effect. He also mentions that it does not make sense to recreate something that has already been done by someone else, so he also frequently visits asset stores to look for bases or foundations for his effects.

Due to his art-driven background, Väisänen claims to be less concerned about the optimisation of shaders during the early stages of the project. He tends to think about optimising when he has an approach that is closer to being production ready. He mentions that he does not think that there are standardised methods for shader optimisation, but he emphasises the importance of knowing where to optimise. He highlights the importance of finding the reason for the bottleneck rather than blindly trying to optimise as it is not efficient nor produces great results. He also adds that individual shaders tend to be more difficult to optimise as it is difficult to locate the cause of the performance drop within them.

When discussing shader scalability and reusability Väisänen mentions how usually the focus is to build something that works for a specific purpose rather than building complex and reusable shader systems. He thinks that sometimes it might be useful to build reusable systems for instance for multiple games with the same style, but that largely depends on the situation. He also adds that some general-purpose shaders from the asset stores often struggle to be performant as they have been developed to be very versatile and flexible. Väisänen mentions that it is generally easier to start from the specific purpose that is required and add additional features rather than trying to build a massive shader system.

Väisänen states that he has more experience with the traditional text-based shaders but has also used visual node tools to create various shader effects. When talking about the differences in the workflows, he mentions the clear benefits of node-based tools. He describes them as an easy way for artists to be involved with shaders and add additional adjustments to existing shaders. He also highlights how the developers can always be sure that the shader works with the graphic settings as the node-based tools abstract away the technical details. Väisänen also describes the workflow when working with text-based shaders. He says that when making shaders by code working is faster and it is easier to share shaders between graphics programs. However, text-based shaders are less

approachable as they require some base-level knowledge to understand. He does not think one workflow is superior to the other but rather emphasises how it depends on the use case.

When talking about challenges in shader creation Väisänen claims that it is often not the shader itself that is challenging but rather the research behind it when trying to create something new. He describes a project he did some time ago that took him approximately half a year of research and multiple different approaches to get to a viable result. He defines how usually there is no one correct solution, but rather great amounts of existing research. According to him the best solution comes with experimentation and learning what the pros and cons of each approach are. Väisänen also mentions that he finds this project to be one of his favourites aside from one other project he mentioned.

Väisänen emphasises that creating shaders requires a level of base knowledge about limitations and approaches. He describes that this base knowledge grows over time as one keeps practising and experimenting with shaders and shader workflows. He repeats the importance of building the groundwork by doing multiple smaller practices so that when starting a project, one already has an idea of the possible problems and approaches to solve them. He also mentions how in the beginning it is also useful to focus on a specific area of effects and expand to a larger variety as more knowledge has been acquired.

Appendix 5/1

SUMMARY OF THE INTERVIEW WITH ILARIA CISLAGHI

Ilaria Cislaghi is a freelance technical artist focusing on visual effects and a contributor to the Godot game engine. She has a background in computer engineering but has also been personally involved in art. During her career, she has worked in various roles in game development before fully moving to the field of VFX. She currently maintains the particle system in Godot in her free time.

Cislaghi has experience with shader-player interactivity and she describes prior projects where she has utilised said interactions. She also talks about the production methods and limitations she has encountered in those projects. All the limitations in the various assets are quite different and Cislaghi emphasises that one can never make the perfect shader. She mentions that rather it is important to know the limitations that affect the shader and know how to adapt to them. In addition, she mentions that shaders heavily rely on priorities: working on shaders is often about trade-offs, as some techniques will have higher visual fidelity, but lower performance. She emphasises that the priorities are usually set by one's team, stakeholders and the context of the game. Cislaghi describes her pipeline with shaders as being very iterative, but she tends to rely on references and concept art. She prefers to have a clear plan from the beginning and does initial research on pre-existing similar shaders before starting to work on her own.

When talking about shader optimisation Cislaghi mentions that trade-offs are a massive factor when it comes to shaders. According to her, everything depends on the context of the game and its requirements, but in addition, she mentions general practices that are not great with shader workflows. These practices are for instance having loops or if statements inside shaders. Cislaghi describes that these practices tend to be unpredictable and cause issues later in the production process and therefore should be avoided. However, as everything depends on the context, she also mentions that in some cases it is also acceptable to make unoptimised choices if one has the allocated resources to do so.

Regarding scalability and reusability, Cislaghi describes herself as a big fan of iterative work. She mentions flexibility as the core of a VFX or shader artist's workflow as video games while in development are always prone to changes and discarding work is common. She emphasises that changes come quickly therefore it is not often worth spending a large amount of time building a perfect shader. Instead, one should iterate as much as possible as parts from previous iterations might be usable as parts of other shaders in the project. It was also mentioned that generally, it is better to construct smaller shaders that maintain their specific tasks. If it appears that shaders could be either combined or broken into even smaller shaders then that operation should be done when needed.

Cislaghi has experience both with text-based shaders and node-based tools and she mentions that she can interact with both. She describes visual node-based tools to be great for experimentation and prototyping as they are quick, and results can be easily achieved. Additionally, she mentions that some results might also come accidentally. Text-based shaders, however, require a lot more intention. She mentions that text-based shaders are more performant in general and have fewer limitations although they require more understanding of shaders and shading languages to be utilised efficiently. Cislaghi claims that she does not think one workflow is superior to the other but rather the choice of tool depends on the priorities and the utilised system. She mentions that shaders require both artistic knowledge and technical knowledge. Therefore, it is quite difficult to find people with both these qualities as the two worlds are generally quite separate. She adds that people tend to work with the tools they are given and ensuring a functional pipeline is essential.

When discussing challenges in shader production Cislaghi remembers a project that proved to be challenging as small mistakes in text-based shaders happen very often and are difficult to locate. She mentions the difficulty of debugging shaders as shader languages cannot print values like general-purpose

programming. Instead, shaders output colours and patterns, and they can be challenging to interpret. She also adds that sometimes issues are not related to the shader or the workflow but rather to the game engine itself. According to her, these issues are even more difficult to locate. Cislaghi emphasises that persistence and patience are the keys to shader creation as sometimes shaders take a large amount of time to be amazing but one should always trust the process. She mentions that making fun effects and keeping one's internal motivation high are important when learning and working with shaders.

Cislaghi also describes the state of the game industry and highlights the importance of talking and interacting with people in the industry. She emphasises that even short encounters can lead to unexpected results. As numerous people are hired based on their portfolios, Cislaghi points out that in her case, networking was crucial. Connections and networks are very important. Cislaghi also suggests not focusing on a singular game engine but rather trying to work with different engines. Having a wide range of knowledge can open more opportunities and most of the systems and practicalities are translatable to other common game engines. She adds that knowing multiple game engines can help but people adapt quickly. According to her learning a new game engine is going to take approximately the same time as onboarding for the project.

Appendix 6/1

CREDITS AND CONTRIBUTIONS TO THE PRACTICAL PROJECT

210101 Winter quiet uneventful outside amb by TRP
Available at: https://freesound.org/people/TRP/sounds/616882/
[Accessed 12 November 2024].

3D Character “Gobot” by GDQuest.
Available at: https://github.com/gdquest-demos/godot-4-3D-Characters
[Accessed 12 November 2024].

Appalachian Snowy Afternoon 1 amb by LakeWoodSound
Available at: https://freesound.org/people/lakewoodsound/sounds/719851/
[Accessed 12 November 2024].

Character Animation “Walking in Snow” by Christopher Solis.
On request for the project.

Ground 02 Material by Free Stylized.
Available at: https://freestylized.com/material/ground_02/
[Accessed 12 November 2024].

Rocks – Asset Pack by Free Stylized.
Available at: https://freestylized.com/asset_pack/rocks-01/
[Accessed 12 November 2024].

SFX Snow Footstep 01 by Bajko
Available at: https://freesound.org/people/bajko/sounds/378056/
[Accessed 13 November 2024].

Skybox 05 Material by Free Stylized.
Available at: https://freestylized.com/skybox/sky_05/
[Accessed 13 November 2024].

Appendix 6/2

Snow 01 Material by Free Stylized.
Available at: https://freestylized.com/material/snow_01/
[Accessed 12 November 2024].

Song “Lumimusa” by Kuutti Paartola.
On request for the project.

Stylized Fence by Konow3D.
Available at: https://sketchfab.com/3d-models/stylized-fence-8d0b667004764c1da10645831bc87811
[Accessed 12 November 2024].

Super Mario 64 Penguin by Alec Pike
Available at: https://www.models-resource.com/nintendo_64/supermario64/model/929/
[Accessed 13 November 2024].