Nicolette Orispää

Optimization in Mobile Game Development:

Leveraging 2.5D Visuals for Performance and Aesthetic Appeal

Metropolia University of Applied Sciences

Bachelor of Culture and Arts

Design

Thesis

14.4.2025

Tiivistelmä

Tekijä(t):	Nicolette Orispää
Otsikko:	Optimointi mobiilipeli kehityksessä: 2.5D-visualisoinnin hyödyntäminen suorituskyvyn ja esteettisen vetovoiman parantamiseksi
Sivumäärä:	51 sivua + 1 liite
Aika:	14.4.2025
Tutkinto:	Muotoilija (AMK)
Tutkinto-ohjelma:	Muotoilun tutkinto-ohjelma
Pääaine:	Visuaalisen viestinnän muotoilu
Ohjaaja(t):	Lehtori Tania Chumaira

2.5D-visualisointi, joka on alunperin kehitetty PC- ja konsolipelaamisen varhaisten aikojen laitteistorajoitusten kiertämiseksi, on jälleen noussut merkityksellisiksi mobiilipelaamisen kasvun myötä. Sen soveltamisesta mobiilipelikehitykseen on kuitenkin vain vähän aiempaa tutkimusta. Tässä opinnäytetyössä tutkitaan mobiilipelien kehittämisen haasteita ja selvitetään, kuinka 2.5D-visuaalinen tyyli voi auttaa voittamaan nämä rajoitteet säilyttäen samalla ainutlaatuisen ulkoasun.

Opinnäytetyön yhteydessä kehitettiin kaksi versiota prototyyppipelistä: toinen hyödynsi tutkimukseen perustuvia optimointitekniikoita ja toinen rakennettiin ilman optimointia. Molempien versioiden suorituskykyä analysoitiin optimointitekniikoiden vaikutuksen arvioimiseksi. Vaikka tutkimuksen laajuus ei ollut riittävä lopullisten johtopäätösten tekemiseen, ja tulokset osoittivat vain vähäisiä suorituskykyeroja, tutkimuksessa selvisi, että optimoitu versio kulutti vähemmän videomuistia kuin optimoimaton versio.

Tämä opinnäytetyö luo pohjan 2.5D:n tarkemmalle tutkimukselle mobiilipelien optimointikeinona. Vaikka lisätutkimuksia tarvitaan, tulokset viittaavat siihen, että 2.5D voi parantaa suorituskykyä pelikehityksessä, samalla säilyttäen visuaalisen omaleimaisuuden.

Asiasanat: mobiilipelit, 2.5D, optimointi

Abstract

Author(s):	Nicolette Orispää
Title:	Optimization in Mobile Game Development: Leveraging 2.5D Visuals for Performance and Aesthetic Appeal
Number of Pages:	51 pages + 1 appendix
Date:	14 April 2025
Degree:	Bachelor of Culture and Arts
Degree Programme:	Design
Major:	Visual Communication Design
Instructor(s):	Tania Chumaira, Senior Lecturer

Originally used to overcome hardware limitations in PC and console gaming back in the early days of game development, 2.5D has risen in relevance again with the rise of mobile gaming. However, there is limited research on its application in mobile game development. This thesis examines the challenges surrounding developing games for mobile devices and explores how a 2.5D visual style can help overcome these constraints while maintaining unique visuals.

To investigate this, two versions of a prototype game were developed alongside this thesis: one using researched optimization techniques and another without optimization. A performance analysis compared both versions to assess the impact of these techniques. While the study's scope was not large enough for definitive conclusions, results showed minor performance differences, with the optimized version reducing video memory usage compared to its unoptimized counterpart.

Although further research is required, this thesis provides a foundation for studying 2.5D as a tool for mobile game optimization, suggesting it can enhance performance while preserving visual uniqueness.

Keywords: mobile games, 2.5D, optimization

Table of Contents

1 Introduction

Not quite two-dimensional (2D) but not fully three-dimensional (3D) either, two-and-a-half dimensional (2.5D) is a visual style that blends elements of both 2D and 3D. In the early days of game development, it was used to overcome hardware limitations on PCs and consoles. Early implementations of 2.5D included parallax scrolling, where 2D planes moved at different speeds to create the illusion of depth. As 3D environments and camera movement became more common, 2.5D evolved to incorporate 2D sprites within 3D spaces. While modern PC and console games now use 2.5D mainly as an aesthetic choice rather than an optimization technique, its practical benefits may still be valuable in mobile game development, where hardware constraints remain a challenge.

In recent years, mobile gaming has become the most widely played form of gaming, making the need for visually engaging yet performance-friendly games more relevant than ever. Despite advances in mobile device hardware, developers still face limitations due to power capacity, rendering performance, physical size constraints, and file size restrictions. While there exists research on optimizing 2D and 3D games for mobile devices, there is a lack of research focusing on 2.5D within this context. This highlights an underexplored area in mobile game optimization: while 2.5D is commonly used for stylistic purposes, its potential to balance visual quality with performance optimization on mobile platforms has not been studied enough.

In this thesis I will explore whether 2.5D is a viable visual style for overcoming the constraints of mobile game development while maintaining visual appeal. By investigating how 2D and 3D elements can be effectively blended and optimized, this research aims to provide practical insights for mobile game developers. I will conduct this through a combination of literature review and hands-on testing.

To investigate this, I will conduct a literature review that explores the fundamentals of mobile game development and the principles of 2.5D. This will include an analysis of mobile hardware limitations, optimization techniques, and best practices. I will also examine the definition of 2.5D, along with its advantages and challenges in the context of game development. Additionally, I will discuss my choice of Godot as the primary game engine for this thesis, outlining its strengths and weaknesses concerning mobile game development.

The second part of this thesis follows a research-through-design approach, where I will document the process of creating optimized 2D and 3D game assets and integrating them into a 2.5D mobile game using Godot. This case study will compare different asset workflows, analyze the results, and evaluate the viability of these methods. To implement the project, I have used open-source tools such as Krita for 2D assets and textures, Blender for 3D modeling, and Godot for development. The final evaluation will assess performance, visual quality, and overall usefulness.

Throughout this thesis, I have utilized the ChatGPT (GPT-4o) for assistance with text structure, grammar, and idea generation. However, as the author of this thesis, I take full responsibility for all content in this thesis.

2 Literature Review

2.1 Fundamentals of Mobile Game Development

The gaming market in 2024 indicated that the mobile game industry is continuously growing, accounting for approximately 49% of global gaming revenue, according to an analysis by Isa Muhammad on Pocket Gamer (Muhammad 2024). As mobile gaming rises in popularity, game developers need to make games accessible and playable on a wide range of mobile hardware (Sergeev 2021).

However, despite the increasing power of modern smartphones, they still face hardware limitations, as well as challenges such as small screens and controls limited to touchscreen inputs, which must be considered in mobile game development (Budiu 2015; Sergeev 2021). Another key constraint is device fragmentation, meaning not all mobile devices have the same processing capabilities. While high-end mobile devices can rival low-end PCs in terms of hardware capabilities, lower-end mobile devices must also be accounted for. To ensure accessibility across a wide range of devices, games need to be developed in a way that enables adaptive graphics settings, allowing players to adjust visual quality based on their device's performance. (Sergeev 2021.)

Despite the limitations of mobile devices, game developers have found ways to overcome these constraints, enabling some of the most successful and well-known games to run smoothly on mobile platforms. For instance, Epic Games' *Fortnite* and MiHoYo's *Genshin Impact* are both large-scale, graphically intensive games that have been successfully adapted for mobile devices. Modern game engines include built-in optimization techniques originally developed for PC games, many of which also apply to mobile development (Sergeev 2021).

While mobile game optimization is a broad topic on its own, in this thesis I will primarily focus on the constraints of mobile devices and how to overcome these limitations through key asset optimization practices. Additionally, I will briefly discuss techniques such as level of detail (LOD) adjustments in Chapter 2.1.2, which, while not utilized in my case study, are still significant for optimizing 3D mobile game performance.

2.1.1 Constraints of Developing Games for Mobile

Unlike PCs and game consoles which use separate components for processing tasks, mobile devices integrate all computing functions into a single microchip known as a system on a chip (SoC). This means that the central processing unit (CPU), which handles general computations and game logic, and the graphics

processing unit (GPU), which is responsible for rendering graphics, are combined within the same microchip. (Verdict n.d.) Additionally, mobile devices have significantly lower power capacity compared to PCs, which affects their ability to run games efficiently. Intensive, graphically demanding games can lead to increased power consumption, negatively impacting both performance and battery life of a mobile device. (Sergeev 2021.)

While mobile devices are designed to be small and portable, their compact screens limit the amount of content that can be displayed compared to PC monitors and TVs (Budiu 2015). On smaller screens, visual elements need to prioritize clarity at a larger scale as smaller details may go unnoticed and may create unnecessarily larger file sizes (Sergeev 2021). This impacts both game environments, which must display key information clearly, and user interface (UI) elements, which need to be scalable, easily readable, and easy to interact with on a small screen (Budiu 2015). This leads to another constraint of mobile devices: touch screens.

Most mobile games rely on touchscreen controls for user input. Unlike using a mouse or controller, mobile games are controlled through multiple gestures like tapping, swiping, and dragging, depending on the operating system. Touch controls are less precise and can feel clumsy compared to using a mouse. Additionally, since interactions occur directly on the screen, a portion of the display may be obstructed by the player’s fingers, which must be considered in both gameplay and UI design. (Budiu 2015.)

Besides hardware limitations, releasing a game on mobile app stores comes with its own constraints. Large file sizes not only require sufficient storage space on the user’s device but also result in longer download times (Google n.d.). While well-known companies can often get away with large game files and long download times, smaller development teams may face more pushback if their games take up excessive storage. Both Google and Apple have file size limits on applications, although their documentation and approaches differ. Google Play recommends a compressed file size of 150MB for download size,

while Apple allows applications to be up to 4GB uncompressed. (Apple Developer n.d.; Google n.d.) Additionally, Google strongly recommends that developers keep file sizes well below the limit to improve installation rates and user retention (Google n.d.).

2.1.2 Best Practices for Mobile Game Development

Since mobile devices rely on SoCs for all processing tasks and operate with limited power supply, optimizing GPU and CPU workloads is essential for maintaining performance and energy efficiency (Sergeev 2021; Verdict n.d.). One key strategy is asset management, which not only enhances performance but also conserves system resources. This can be achieved by minimizing asset file sizes, such as compressing textures and optimizing mesh data, and reducing the number of draw calls. (Linietsky & Manzur & the Godot Engine Community 2024b.) Techniques like LOD management and texture atlases allow multiple assets to be processed more efficiently, improving rendering performance and ensuring smoother gameplay (Android Developers 2023b; Android Developers 2024a).

Textures are one of the more processing heavy aspects of a mobile game. While PCs and consoles can brute force the rendering of high-quality graphics, mobile devices have a limited amount of power they can draw on, so therefore by reducing texture sizes as much as possible, game performance can be optimized. (Sergeev 2021.) Reducing the size of a texture can be achieved through image compression and minimizing data within the texture, like color information. Ideally, texture files should feature a limited color palette and rely on solid colors whenever possible. (Android Developers 2024a.) When using mobile rendering methods in Godot, the displayed color range is constrained by low dynamic range rendering. Meaning the renderer automatically reduces the number of colors being displayed on screen. While workarounds exist, simpler graphics with solid colors are better suited for mobile devices. (Linietsky & Manzur & the Godot Engine Community 2024a.) Additionally, textures with

excessive detail may be impractical, as finer details may become indistinguishable on small screens (Android Developers 2024a).

Textures can also be used to create atlases that are then used for 3D meshes or 2D sprites. I will go more in depth on texture atlases and provide an example of what they are and how they are used in Chapter 4.1.1, but merging images and textures into one image is a keyway of reducing draw calls and lifting the workload of a mobile device's CPU. (Android Developers 2024a.) Individual images and sprites can get expensive performance-wise if there are multiple of them in the game environment. By combining all the elements on one sheet, the CPU only has to load up one image and draw the information from it rather than load multiple images at once. (Avisekhar 2016, 120; Android Developers 2024a.)

3D asset optimization follows the same fundamental principles as 2D optimization: less is better. 3D meshes are created with vertices and triangles that contain information that mobile devices use to render out the final shape (Avisekhar 2016, 135). To ensure efficient rendering, meshes should have the lowest possible polygon count while maintaining clean topology (Android Developers 2023b). While I will go into the basics of 3D mesh creation and optimization in Chapter 4.2.1, it is important to note that, unlike PCs and consoles, which can handle high polygon (high poly) meshes, mobile devices have limited processing power. (Linietsky et al. 2024b.) For smooth performance, meshes must be simplified and retopologized to maintain a clean, evenly distributed structure (Android Developers 2023b).

To save on power usage, mobile devices use a technique called tile-based rendering, where the screen gets divided into a grid, and information in each cell gets rendered out one at a time. If there is too much information in one of these cells, for example a high poly mesh, this causes the mobile devices GPU to have to work harder than necessary and worsens performance. Situations where high amounts of vertices are being rendered on screen include high counts of meshes being rendered from far away. A way to avoid this kind of

problem would be to create a few different meshes that each have a different level of detail, or in other words LODs. This is useful when an object is far from the camera and does not need to be highly detailed, so a completely new mesh is created with lower detail to be loaded in at different viewing distances, as illustrated in Figure 1. (Linietsky et al. 2024b.)

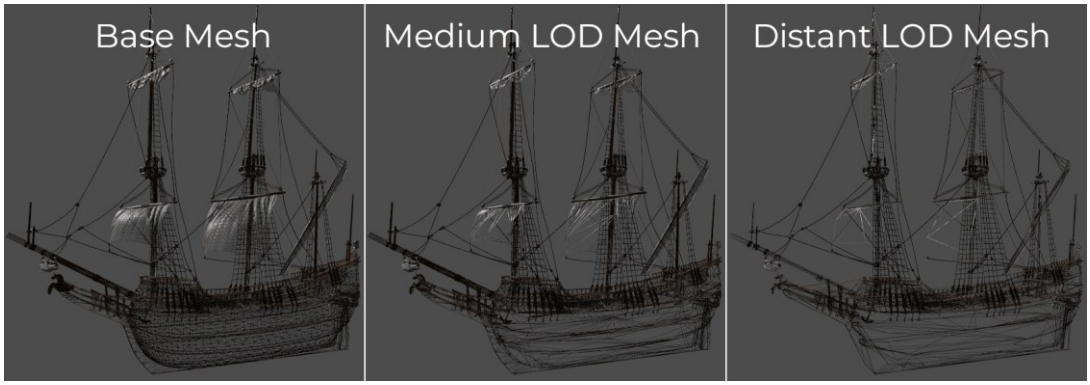

Figure 1. Mesh comparison across different LOD levels (Linietsky & Manzur & the Godot Engine community 2024).

Another optimization technique similar to LODs is creating billboards. As the name suggests, a quad mesh is created and a 2D sprite or image is rendered onto the mesh creating a billboard effect. This method is useful when rendering objects that only appear far in the background, like buildings or trees. Too many quads can still cause performance issues with tile-based rendering if there are multiple of them in one spot, but the point of billboarding is to reduce the number of detailed 3D meshes being rendered. (Linietsky & Manzur & the Godot Engine Community 2025b.)

Lighting is another performance heavy feature in mobile game development (Linietsky et al. 2025b). Android Developers documentation suggests removing the use of lighting entirely. This means either designing a mobile game to look visually appealing without the use of shading, or in Godot documentation, it is recommended to at least bake in the lighting. This means the shadows and lights in a mobile game will not be dynamic, but it will give some depth to 3D objects compared to having no lighting at all. (Android Developers 2024b; Linietsky et al. 2025b.)

Finally, another practice to keep in mind in mobile game development is to minimize or avoid the use of transparency. Opaque objects are easy to render, as mobile devices can handle rendering them in any order, but if there is a transparent texture or material it causes the mobile renderer to have to focus on drawing everything in a particular order. This becomes especially performance-heavy when multiple transparent objects overlap, as the mobile renderer must determine the correct layering for each object. (Android Developers 2024b; Linietsky et al. 2024b.)

2.2 Overview of Godot Engine for Mobile Game Development

I chose the Godot Engine for this thesis and my case study primarily due to its lightweight nature. Based on my personal experience with Unity and Unreal Engine, both tend to have significantly longer startup times in comparison. Unreal Engine is notably resource-intensive, as even a new, unmodified project takes a considerable amount of time to load on mobile devices. Given these factors, Godot provides a more efficient and accessible option for mobile development (L'Italien 2024). It is designed to minimize the GPU workload as much as possible, making it an ideal choice for using in my case study (Linietsky et al. 2024b).

Godot Engine is a free and open-source game engine first released in 2014 (L'Italien 2024). It provides a unified interface for developing both 2D and 3D games. Notably, the Godot editor itself operates as a game running on its own engine. Because it is open source and licensed under the MIT license, developers do not have to pay royalties for games created with Godot. Also, the engine supports easy export to multiple platforms, including Windows, macOS, Linux, and mobile platforms such as Android and iOS. (Linietsky & Manzur & the Godot Engine Community 2025a.) This thesis will focus on the development of mobile games specifically for Android devices.

Godot has traditionally been stronger in 2D game development, particularly in versions prior to 4.0. However, with the release of Godot 4.0 in 2023, significant

improvements were introduced for 3D game development. One of the most notable additions was Vulkan, a high-performance renderer designed for higher-end devices. Developers, however, can still opt for OpenGL-based rendering, which is compatible with mid to lower-end hardware. (Godot Contributors 2023.) Another key improvement is occlusion culling, which optimizes rendering by preventing hidden objects from being rendered, reducing unnecessary processing, and improving performance on mobile devices (Godot Contributors 2023; Linietsky et al. 2025b).

Godot supports game development through both visual node-based editing and traditional coding. Its 2D and 3D editors include built-in tools for animation, tilemaps, and shader creation, alongside a debugger and profiler for performance tracking. (L'Italien 2024.)

A major advantage of Godot is its open-source nature, which holds an active development community (Linietsky et al. 2025a). With over six hundred contributors, users can inspect and modify the source code of the engine, address issues, and create custom plug-ins to enhance development. However, as a newer engine compared to Unity and Unreal, Godot’s community remains smaller, and external documentation, tutorials, and asset libraries are not as extensive. (L'Italien 2024.)

2.3 Understanding 2.5D in Video Games

The term 2.5D is used to describe games that blend 2D and 3D elements together in a video game (VFX Apprentice 2023). While 2.5D used to be commonly associated with 2D games that mimicked the visuals of 3D, nowadays 3D games are also being classified as 2.5D through techniques like trying to imitate the appearance of 2D (VFX Apprentice 2023; Wikipedia 202). 2.5D used to be used to overcome the limitations of hardware back in the old days, as PCs and consoles were not powerful enough to render fully 3D environments (Wikipedia 2025a). Nowadays 2.5D is mostly used for aesthetic purposes rather than as a workaround for hardware constraints when it comes

to PC and consoles (VFX Apprentice 2023). Mobile devices on the other hand still have numerous constraints when it comes to rendering games (Wikipedia 2025a).

Because hardware and memory space on PCs and consoles were limited back in the day, game developers had to find ways to create visually engaging games while keeping file sizes minimal. One of the oldest examples of 2.5D is *Interceptor* (1975), a combat flight simulator developed by Taito. The game utilized joystick controls to aim with a crosshair on the screen, with the objective to shoot down enemy aircraft that moved on screen. The enemy aircraft were scaled according to how close they were to the player, thus creating the illusion of depth. (Wikipedia 2025a.)

Following this, still within the realm of 2D games, game developers began to utilize methods from animation, such as layering multiple image planes to produce a parallax effect. *Moon Patrol* (1982) was among the first games to implement this method, influencing later titles such as Sega's *Sonic the Hedgehog* and Nintendo's *Mario* franchises. (VFX Apprentice 2023.)

As hardware capabilities improved, so did the complexity of game development, though limitations still remained. *Doom* (1993) and *Wolfenstein* (1994) were some of the first 3D games that featured a 3D environment but utilized 2D sprites for all visual elements to save on performance. *Doom* also utilized early methods of pre-rendered sprites, where 3D models were created and then rendered out as 2D sprites to optimize performance. (Wikipedia 2025b.)

Today, 2.5D is used primarily as a visual style rather than a method for optimizing performance on PCs and consoles (VFX Apprentice 2023). A modern example of how 2.5D is being utilized is the game *New Super Mario Bros. U Deluxe* (2019), created by Nintendo. Although the game's assets appear fully 3D, as illustrated in Figure 2, the camera is fixed in a side-scrolling perspective. The player cannot manipulate the in-game camera, which limits the viewpoint to a 2D perspective. (Game Domain 2018.)

Figure 2. Characters and objects appear as 3D elements here, but the environment is locked to a 2D perspective. Screenshot from *New Super Mario Bros. U Deluxe* (2019).

Alongside 2.5D, there has been a rise in the term HD-2D, a visual style that blends the nostalgic aesthetic of classic games with the capabilities of modern graphics and game engines. Square Enix's *Octopath Traveler* game series is one of the most well-known examples of HD-2D games, with a mixture of 2D pixel art and 3D elements utilizing pixel textures, as illustrated in Figure 3. *Octopath Traveler* and other HD-2D titles have been praised for their visual appeal, particularly on handheld consoles and mobile devices. (Duraj 2023.)

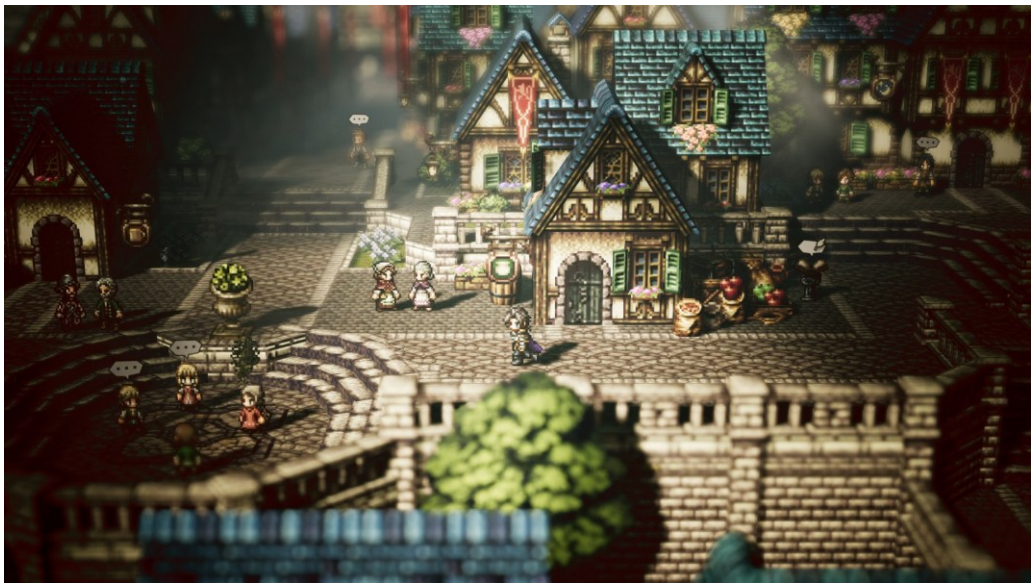

Figure 3. An example of an HD-2D game that blends 2D pixel sprites into a stylized 3D environment. Screenshot from *Octopath Traveler* (2018).

Initially, before conducting research into this thesis, I considered *Octopath Traveler* as an example of 2.5D game design. However, the further I investigated this topic, the more frequently I came across the term HD-2D in relation to these styles of games. This raises the question of whether HD-2D represents a specific subcategory or a change in the definition of 2.5D.

2.3.1 Ways of Implementing 2.5D in Video Games

The implementation of 2.5D can vary depending on whether the base game is primarily 2D or 3D. In 2D-based games, elements are often layered to create the illusion of depth, commonly referred to as parallax scrolling (Root 2019). Graphical projections can be used to create the illusion of depth by drawing 2D assets in a way that projects them visually on three axes (Wikipedia 2025a). In a 3D-based game, 2.5D is typically achieved through fixed camera perspectives that mimic the constraints of 2D gameplay (Game Domain 2018). Visual style can also play a role in 3D-based games; 3D elements may be styled to resemble flat 2D artwork. This effect is usually achieved using shader and rendering techniques within game engines. (VFX Apprentice 2023.)

In both 2D and 3D based games camera perspective is a key factor in defining 2.5D (Game Domain 2018; Wikipedia 2025a). By locking the viewpoint of the camera, 2D assets can be designed to appear in 3D without allowing movement that would otherwise break the illusion (Wikipedia 2025a). The same principle applies when 3D games are referred to as 2.5D, the camera is fixed at a specific angle or perspective (Game Domain 2018). To achieve 2.5D, several techniques can be used, including parallax scrolling, axonometric and oblique projection, and billboarding (Wikipedia 2025a).

Parallax scrolling is one of the most traditional ways of implementing 2.5D. It is not exclusive to video games; it is a method used also in animation (VFX Apprentice 2023; Wikipedia 2025a). Parallax scrolling is achieved by layering multiple images on top of each other to create the illusion of depth, as illustrated in Figure 4. The farther away an element is supposed to be from the camera, the slower the elements move, and for the opposite, the closer something appears to the camera the faster it moves. (Root 2019.) Classic examples of parallax scrolling can be seen in games like the *Super Mario* and *Sonic the Hedgehog* franchise.

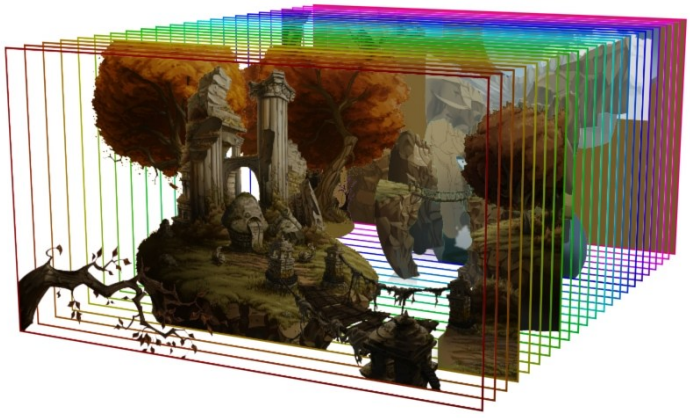

Figure 4. Images split into layers to be used in parallax scrolling (Paletta n.d.).

Parallax scrolling is not limited to horizontal movement along the X-axis; it can also be applied to vertical movement. In such cases, the scrolling speed is reduced on the Y-axis, with sprites farther from the camera moving more slowly than those closer to it, which scroll at faster speeds. Some games have also

attempted to mimic the effect of rotation, where distant objects move in the opposite direction of the character's movement, often at an exaggerated speed, but this can be hard to implement and may cause visual disorientation if done badly. (Root 2019.)

Axonometric and oblique projections are types of parallel projection, where objects are drawn at an angle to create the illusion of depth in otherwise flat visuals. These projections use a fixed perspective scale, known as orthographic projection, meaning that all objects are rendered at the same scale regardless of their position, resulting in a viewpoint with no depth perception. Figure 5 illustrates several projection types commonly used in 2D games to simulate depth and perspective. While perspective camera viewpoints are less common in 2D games, some titles have successfully implemented perspective projection despite being entirely 2D. (Wikipedia 2025a.)

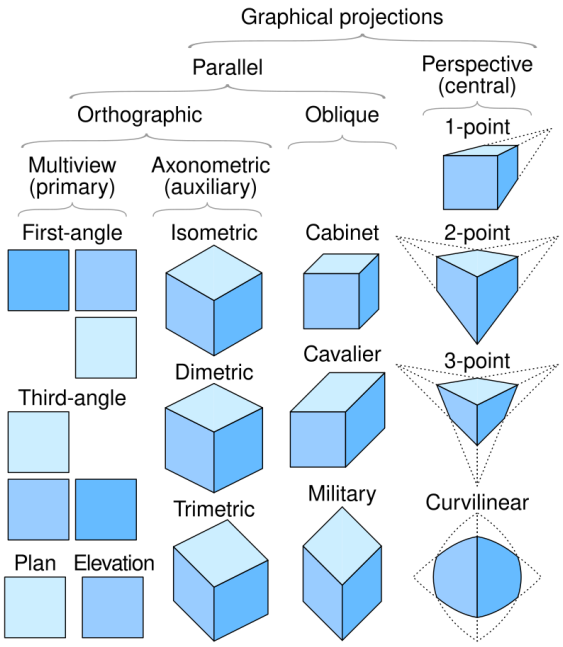

Figure 5. An overview of various graphical projections, showing the differences in perspective and styles (CMG Lee n.d.).

Isometric perspective, a type of axonometric projection, is one of the most used forms of parallel projection (VFX Apprentice 2023; Wikipedia 2025a). In isometric projection, objects are drawn along three axes using an orthographic

view, meaning that all the elements share the same scale regardless of their position (Wikipedia 2025a). The *Hades* series by Supergiant Games are an interesting example of isometric projection, as illustrated in Figure 6. The first game, *Hades* (2017), is a fully 2D game created with a 2D engine, whereas its sequel, *Hades* II (2024), transitioned to a 3D engine and implemented more 3D methods and rendering techniques into the game design. (Game Informer 2024.)

Figure 6. Comparison of Hades (left) a fully 2D game and Hades II (right) a 3D-based game. Screenshot from *Hades* (2020); Screenshot from *Hades II* (2024).

On a slightly different note, billboarding is a technique used in 3D game development where 2D sprites are placed into a 3D environment through the method of placing an image on a flat quad, or “billboards” (Wikipedia 2025a). Billboarding was first utilized in early first-person games like *Doom*, where the game environment is rendered in 3D, but otherwise all the visuals were 2D sprites (Wikipedia 2025b). *Donkey Kong Country* (1994) is another example of how billboarding can also be utilized in 2D game development. 3D models were created and then rendered into flat images, which were placed in a 2D game environment. Parallax scrolling was then used to create the illusion of depth. (Taylor 2024.) These techniques were workarounds to overcome the hardware limitations of the hardware they were developed for (Taylor 2024; Wikipedia 2025a).

This leads to another closely related technique: using 3D models as a base and pre-rendering them into 2D sprites. *Dead Cells* (2018), developed by Motion Twins, is a documented example of this approach. The playable character was

first created as a 3D model and animated, which was then rendered out as 2D sprites. This significantly sped up the animation process for the 2D sprites, as noted by Thomas Vasseur (2018) in an article published on Game Developer. Using these 3D bases, the studio also created normal maps, a technique that utilizes special textures to create the illusion of depth and detail on a 2D sprite or 3D meshes when light interacts with it. (Vasseur 2018.)

2.3.2 Advantages and Challenges of 2.5D Implementation

Blending 2D and 3D elements together in game development can provide both creative and technical solutions (VFX Apprentice 2023; Wikipedia 2025a). In terms of 2D game development, it allows game developers to create environments with depth and the illusion of 3D utilizing 2D elements. This can be done through methods like parallax scrolling or isometric projections. In 3D-based games, the use of 2D sprites in place of complex 3D models can serve as a form of optimization and reduce performance costs. (Wikipedia 2025a.)

However, 2.5D can also be difficult to implement. Seamlessly blending 2D and 3D elements can be difficult to achieve, and I encountered several issues with this in my own case study, which I will discuss in more detail in Chapter 5.3. *Donkey Kong Country* serves as an example of inconsistent lighting in pre-rendered assets. Upon closer examination, many elements within the game do not have uniform lighting across different objects in the scenes (Taylor 2024). Additionally, if not implemented carefully, 2.5D can lead to player confusion. Root (2019) points out that in the game *Jim Power: The Lost Dimension* (1993), the game tries to simulate a rotating movement in a 2D environment but falls short of this with confusing movement speeds between the foreground and background elements. (Root 2019.)

3 Research Methodology

To deepen my understanding of mobile game development, I took a hands-on approach by creating a couple of 2.5D mobile game prototypes as case studies

to support the findings presented throughout this thesis. As part of my research process, I kept a research diary throughout the development of these mobile game prototypes. This diary, included as an appendix to this thesis, served as a tool for tracking progress, documenting design decisions, and noting down technical findings. I used it to log each stage of the project, from initial concepting and asset creation to the performance evaluations conducted using Godot's profiler and debugger tools. Key observations and challenges were noted throughout the prototype's development, along with the strategies and solutions I used to address them. The development of the prototypes began in January 2025 and continued through April of the same year, so four months of development time as of writing this thesis. However, development remained ongoing beyond the thesis submission, as the final version of the prototypes was due at a later date. By documenting each step of the development process, I aimed to gain a clearer understanding of asset creation and optimization techniques while providing insights for analysis in this thesis. Since 2.5D combines both 2D and 3D elements, ensuring visual cohesion between these assets was essential. Additionally, having a clear reason for choosing which elements should be created in 2D versus 3D was a key consideration, as this decision forms one of the core research questions of this thesis: Why opt for 2.5D instead of exclusively using 2D or 3D?

For this research study, I developed a small-scale 2.5D game prototype, designed primarily as a visual demo rather than a fully functional game. The game utilizes an orthographic camera, which removes depth and perspective, eliminating the need for LODs and mipmaps, which saves on memory usage (Android Developers 2024a). The game environment itself is minimal, featuring a small explorable area with a few scattered assets. To analyze the impact of asset optimization on performance, I created two main versions of the same environment:

- Environment A built using optimized assets that include compressed textures, sprite sheets, and low-poly models.

- Environment B deliberately unoptimized, featuring uncompressed textures, individual sprite files, and a high poly version of a selected model to highlight performance differences.

Animations within the project were kept minimal, with the only animated element being the playable 2D character.

The asset creation workflow involved using Krita for 2D textures and sprites, Blender for 3D modeling and UV mapping, and Godot as the game engine. Performance testing was conducted using Godot's built-in debugging tools alongside playtesting on a range of Android devices of varying ages. The main objective was to analyze how different optimization techniques impacted frame rates, GPU load, and overall performance across devices with different hardware capabilities. Key factors such as draw calls, texture compression, and file size limitations were examined, with a focus on how these optimizations helped the game remain small in file size.

4 Asset Creation and Optimization Comparison

4.1 2D Assets

Whether a mobile game is fully 2D or 3D, it will always contain 2D elements, which means it is essential to understand how 2D optimization works (Avisekhar 2016, 119). UI, splash screens, logos, and materials are just some of the things that make up the 2D portion of a game. For the scope of this thesis, I will focus on the creation and optimization of 2D sprite objects within the game environment, the playable character's texture atlas, and the textures used for the 3D models.

When it comes to optimizing 2D assets for mobile games, there are three essential steps to follow: size, data, and process optimization (Avisekhar 2016, 147–148). While these techniques are categorized under 2D optimization, some of them carry over into the 3D optimization as textures are part of the 3D modeling process. This chapter will expand on the topics covered in Chapter

2.1.2, where I discussed the best practices when it comes to asset optimization. First, I will discuss the use of texture and sprite sheets to batch similar 2D elements together, reducing the number of draw calls. After which, I will examine color-related considerations such as color banding, and filtering modes. Finally, I will examine general performance considerations that are beneficial to understand, such as texture sizes, compression, and how Godot handles color rendering.

4.1.1 Texture Atlases and Sprite Sheets

Texture atlases as well as sprite sheets are images that contain multiple graphical elements within one image (Avisekhar 2016, 120; Android Developers 2024a). While the two terms are sometimes used interchangeably, they serve distinct purposes in game development (Udacity 2015).

A texture atlas is a multipurpose image that consists of various 2D assets, including sprites, UI elements, and textures applied to 3D meshes, into a single file, as illustrated in Figure 7 (Udacity 2015). This technique enhances performance by reducing the number of textures being called during rendering. Additionally, texture atlases can incorporate mipmaps, which are varying sizes and resolutions of the same texture that are used depending on an object's distance from the camera. Although I did not utilize mipmaps in my own case study, they are an important optimization technique similar to LODs, which I briefly discussed in Chapter 2.1.2. (Android Developers 2024a.)

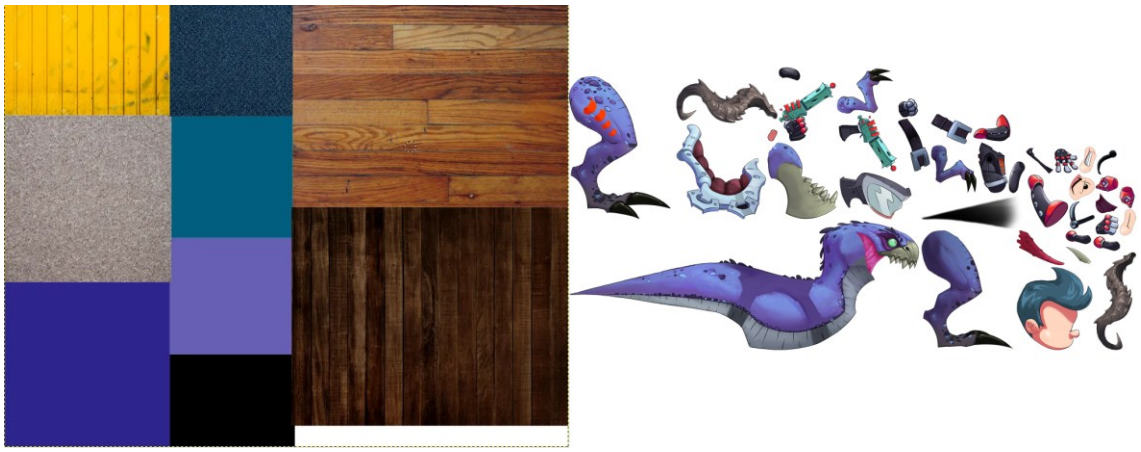

Figure 7. Two examples of texture atlases: the left atlas is designed for use with 3D meshes, while the right atlas is used for a 2D character rig (Bernardi 2018; Esoteric Software 2020).

Sprite sheets, on the other hand, are specifically used in the context of 2D animations. They contain all the animation frames of an object, such as a character sprite's walk cycle or a particle effect, arranged within a single image file, as illustrated in Figure 8. (Udacity 2015.) While sprite sheets are primarily used in 2D games, they can also be applied in 3D environments using the billboarding technique mentioned in Chapter 2.3.1 (Linietsky et al. 2025b).

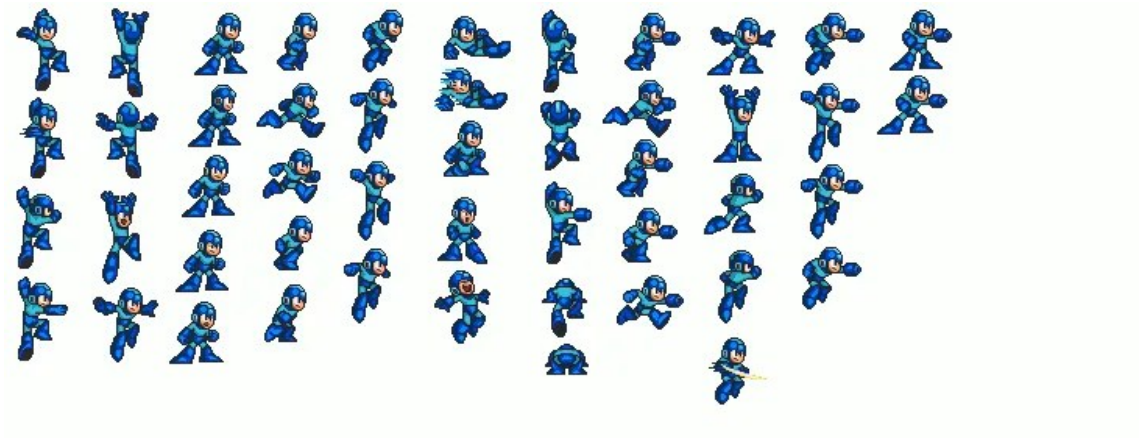

Figure 8. Sprite sheet containing the animation frames for the character Mega Man (Chen 2020).

As textures are one of the demanding aspects of game performance, reducing the number of sprites and textures in a game, especially on mobile, is crucial

(Android Developers 2024a; Sergeev 2021). Not only is it important to group smaller image assets together, but it is also recommended to limit the number of total texture sets being rendered to a maximum of five. Using too many textures requires more texture sampling, which leads to increased power usage on mobile devices. (Android Developers 2024b.) One advantage of Godot in this regard is that the engine reuses textures whenever possible. If multiple objects use the same texture, Godot will only render the texture once rather than rendering it separately for each object. (Linietsky et al. 2024b.)

4.1.2 Color and Texture Filtering Modes

Handling colors and texture filtering effectively is essential in mobile game development, particularly due to hardware limitations and the small size of mobile screens (Android Developers 2024a; Sergeev 2021). Due to these factors, visual clarity becomes a top priority. High levels of detail may go unnoticed and be more of an unnecessary strain on a mobile device's performance. (Sergeev 2021) For this reason, simplified textures or even flat colors are often the better choice for mobile game development (Moscardo Ribes 2024; Sergeev 2021).

Texture filtering refers to rendering techniques that determine the appearance of rendered triangles on an in-game object. The choice of filtering mode can significantly influence the overall visuals of a mobile game as well as the performance. There are three main types of texture filtering: nearest, bilinear, and trilinear. (Android Developers 2024a.)

Nearest filtering is the cheapest in terms of performance, but it comes at the cost of producing sharp, pixelated textures. As the name suggests, nearest filtering renders textures by selecting the color of the texel closest to each pixel. Bilinear filtering, the next level up from nearest, attempts to smooth the transition between texels by averaging the colors of surrounding ones, creating a gradient-like and slightly blurred look, as illustrated in Figure 9. (Android Developers 2024a.)

Figure 9. Nearest filtering, on the left, creates more pixelated textures compared to linear filtering, on the right, which blurs the pixels (Orispää 2025).

Trilinear filtering is useful when textured objects like a ground plane, have varying distances from the camera. Mipmapping is then used to render the textures at these varying distances, and trilinear filtering then blends the mipmaps to create seamless visuals. However, trilinear filtering is the most demanding of the three filtering modes and can significantly impact performance. (Android Developers 2024a.)

According to Android Developers documentation, texture filtering can take up half of the total GPU energy consumption on mobile devices. Therefore, when deciding on a texture filter mode, it is incredibly important to decide whether using heavier filtering is necessary. In most cases, bilinear filtering is considered the optimal choice in mobile development. (Android Developers 2024a.)

4.1.3 Texture Performance Considerations for Mobile

While the use of texture atlases is highly recommended, it is also important to keep in mind that there is a limit to texture sizes when it comes to mobile games. According to Godot Engine's documentation, a single texture file for mobile should be limited to a maximum resolution of 4096×4096 (4k). High-

resolution textures are one of the main factors that can slow down a mobile device's performance, and while 4k is the maximum recommended size, smaller sizes are safer options and easier for low-end devices to handle (Avisekhar 2016, 146; Linietsky et al. 2024a). It is also highly recommended to keep texture dimensions in powers of two, as lower-end device's GPUs may not support tiling textures that do not follow this standard. (Linietsky et al. 2024a.)

In addition to size limits, it is also crucial to compress textures to reduce their file sizes. Godot's documentation states that the game engine compresses textures by default, but textures can be compressed further in whatever image handling program they are made with. (Linietsky et al. 2024b.) For example, as illustrated in Figure 10, there is a significant difference in file sizes between a texture exported out of Krita with no compression and full compression.

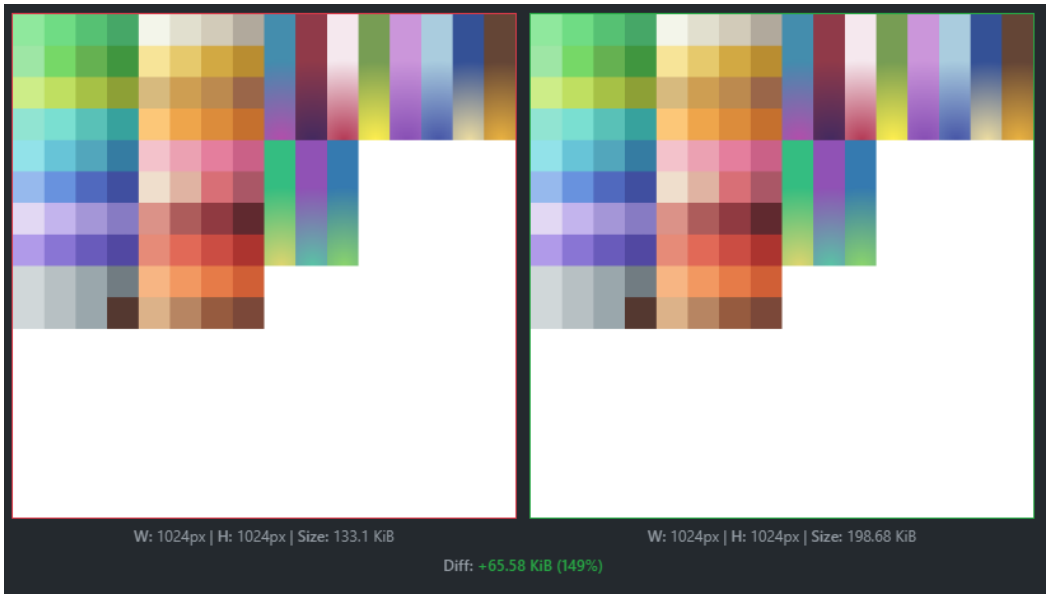

Figure 10. A comparison of the same image where the texture on left is compressed, and the texture on right is uncompressed (Orispää 2025).

There is a fine balance when it comes to compressing textures. Excessive compression can lead to blurry pixels between elements, which lowers the visual quality of the texture. It may also cause color bleeding, resulting in unwanted artifacts. (Linietsky et al. 2024b; Moscardo Ribes 2024.) In 3D objects, compression is generally less noticeable, as geometry plays a more

vital role visually. However, in 2D games, texture compression can result in visible artifacts depending on the complexity and details of the texture. (Linietsky et al. 2024b)

One important consideration in color handling within Godot is that the engine automatically attempts to optimize textures, including reducing the color range rendered in a texture. This can lead to banding, especially in gradients, which may not be the desired look, as illustrated in Figure 11. (Linietsky et al. 2024a.)

Figure 11. An example of how Godot reduces colors causing banding, compared to the same texture with the banding issue resolved (Linietsky & Manzur & the Godot Engine Community 2024).

Godot documentation provides a few solutions for reducing banding. The first is enabling a built-in project setting that applies a debanding shader which is lightweight in performance. The second method involves adding noise directly into the texture before exporting it from an image editing software. Alternatively, noise can be applied though shaders within Godot, though this option is only available when working with 3D. (Linietsky et al. 2024a.)

4.2 3D Assets

While mobile devices have gotten better at running 3D games, processing 3D graphics remains more computationally demanding compared to processing 2D

graphics (Android Developers 2023b; Avisekhar 2016, 151). While textures impact performance in terms of large file size and CPU usage, 3D processing involves processing geometry which is GPU resource-intensive (Android Developers 2024a; Linietsky et al. 2024b).

Unlike 2D optimization, which primarily focuses on texture optimization, 3D optimization requires balancing both textures and geometry. While some optimization techniques from 2D workflow, such as texture sizes and detail, apply to 3D asset creation, the main challenge in 3D optimization is managing the vertices and triangle count of meshes (Android Developers 2023b). There is a fine balance however between detail created through texture versus geometry when it comes to 3D modeling. Sometimes using a few extra vertices to create more detail on a mesh is more efficient than using image texturing to achieve more detail. (Moscardó Ribes 2024.)

There are many ways of optimizing 3D models, ranging from reducing the number of triangles and using LODs to using normal maps and creating custom vertex normals (Android Developers 2023b). 3D optimization is a broad topic and could be an entire thesis on its own. In this chapter, I will focus mainly on optimization through mesh topology, examining how vertex count, triangle density, and polygon distribution affect rendering performance. Additionally, I will explore methods of enhancing visual appeal through shader techniques, as these were methods utilized in my case study.

4.2.1 Mesh Geometry

3D meshes are geometry that are made up of vertices, which form edges and faces that define the shape of the mesh (Android Developers 2023b). The GPU uses the vertex information to render out the shape of a 3D mesh. Generally, the more vertices a mesh contains, the more detailed it appears. However, this increase in information can lead to performance issues, particularly when a high number of vertices are concentrated in a small area on the screen. In these

situations, the GPU requires more time to process and render the data. (Android Developers 2023b; Linietsky et al. 2024b.)

According to Android's documentation, a single mesh can have a maximum of 65 535 vertices, but staying below this count is recommended for compatibility with low-end devices (Android Developers 2023b). In 3D games, it is almost always expected that rendering will impact performance more heavily compared to fully 2D games made with 2D elements (Avisekhar 2016, 151).

One thing to note about 3D mesh rendering is that, even if a model was built using quads, shapes with four vertices, these will be converted into triangles during the rendering process on mobile devices. The most fundamental optimization technique in 3D modeling is to reduce the triangle count of a mesh as much as possible. Reducing the triangle count of individual meshes, as illustrated in Figure 12, can impact the total triangle count in the final build of a game. While high-end devices are typically capable of rendering high poly meshes without issues, low-end devices may struggle to maintain the same performance levels. (Android Developers 2023b.)

Figure 12. Comparison of two meshes, with the left having a larger triangle count than the mesh on the right, yet visually they look the same. Screenshot from Maxon (2025).

Additionally, the acceptable complexity of a mesh depends on the number of other objects surrounding it. In minimal scenes with only a few assets, meshes can be more detailed. However, in scenes containing hundreds of objects, for example, each mesh must be kept as simple as possible to maintain performance. (Android Developers 2023b.)

Beyond vertex and triangle count, the distribution of vertices is also crucial. High concentrations of vertices in a small area can create equally small but high amounts of triangles, which puts a heavy strain on a mobile device's GPU. (Android Developers 2023b; Linietsky et al. 2024b.) Due to the fact that mobile devices utilize tiled rendering, if there are too many vertices and triangles being rendered in one spot it can cause performance issues. (Linietsky et al. 2024b.)

4.2.2 Texturing and UV Mapping

As discussed in Chapter 4.1.1, texture atlases are images composed of multiple graphical grouped into a single file. While texture atlases are important in both 2D and 3D workflows, UV mapping is specific to 3D workflow and defines how textures are projected onto the surface area of 3D meshes, as illustrated in Figure 13 (Android Developers 2024a; Bernardi 2018).

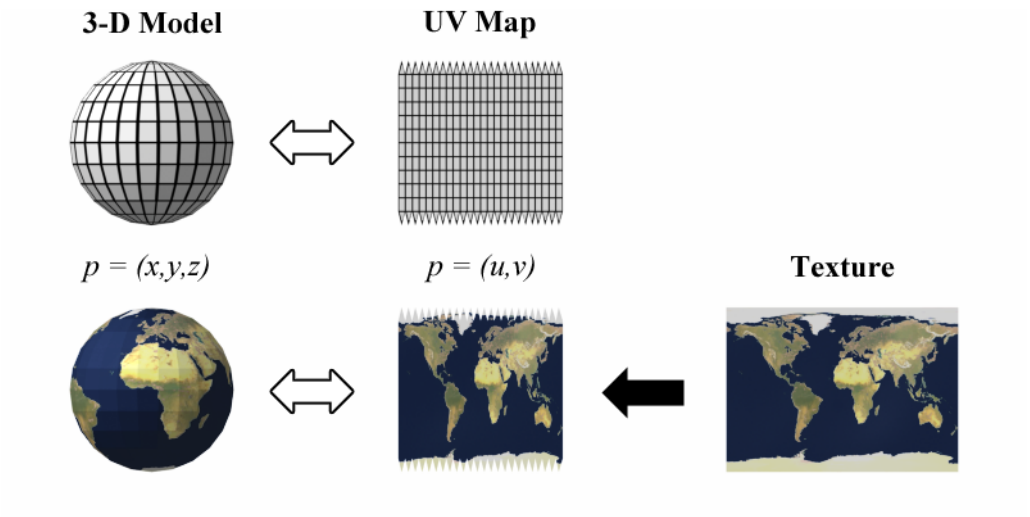

Figure 13. Visualization of how textures are mapped on to 3D models (Tschmits 2008).

UVs are the two-dimensional coordinates used to map faces of a 3D mesh onto a 2D texture (Bernardi 2018). The way UVs are unwrapped and positioned on a texture significantly impacts the visual quality of the mesh. If UVs are poorly aligned, the texture can appear stretched or distorted (Android Developers 2024a).

However, not all meshes require detailed UV unwrapping. While detailed texture atlases benefit from clean UV layouts, stylized methods, such as flat color or gradient textures, can be more forgiving (Android Developers 2024a; Moscardó Ribes 2024). These techniques simulate lighting and shading using color gradients instead of actual lighting, making them ideal for mobile games (Moscardó Ribes 2024). Figure 14 illustrates a mesh which has UVs projected onto vertical gradient textures, where the darker tones represent shadowed areas, and lighter tones highlight the top of the mesh or areas where light would normally hit.

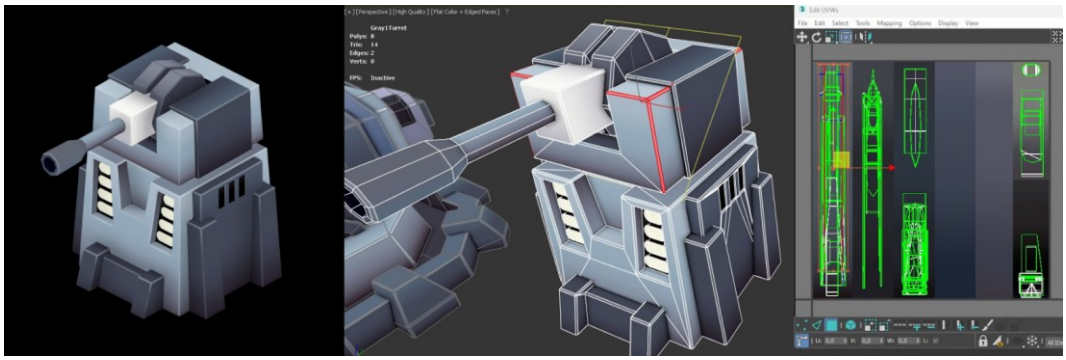

Figure 14. A meshes UVs are projected onto a gradient map where darker colors are at the bottom and gradually lighten to the top (Moscardó Ribes 2024).

This method is beneficial in mobile game development, where avoiding real-time lighting can reduce processing demands. While it may require more detailed meshes to create detail that would otherwise be baked into textures, this trade-off is often acceptable as long as vertex density remains balanced. (Linietsky et al. 2024b; Moscardó Ribes 2024.)

4.2.3 Materials and Shaders

Building on the topic of textures, materials and shaders are another core aspect of game development. Materials are used to apply textures to 3D meshes within a game engine, while shaders are responsible for rendering all visuals onto the screen. Like textures, the general rule for materials is the fewer, the better. Android Developers documentation for mobile development recommends using a maximum of five materials at a time. Using any more than this can start to negatively impact mobile device performance. (Android Developers 2024b.)

Shaders are in away like materials and textures in that they determine the final visual appearance of objects in-game. However, shaders differ in that they primarily function through code run on the GPU rather than image-based textures. Shaders define how light and shadow interact with objects, relying on GPU rendering to produce visual effects. (Amir 2023.)

One important consideration when using shaders is that they utilize mathematical calculations. While simple operations like addition, subtraction, and multiplication are low-cost, more complex operations, like division or transcendental functions, can lead to performance issues, especially on mobile hardware. (Amir 2023; Android Developers 2024b.)

5 Design and Development of a 2.5D Mobile Game

5.1 Concept and Design Choices of the 2.5D Prototypes

During the early stages of researching mobile game optimization, the scope of my case stufy that I was aiming for was much larger, as reflected in the early entries of my research diary. However, as my focus gradually shifted towards writing, I had to make the decision to scale back the game’s scale. Despite these changes, the core concept remained the same: to develop a mobile game prototype that showcases 2.5D visuals, using optimized assets while maintaining visual appeal.

Rather than aiming to create a complete, fully playable game, the main purpose of these prototypes was to serve as an opportunity to experiment with 2.5D development techniques in mobile game development. While the optimized and unoptimized versions of the build were slightly different in terms of performance-related techniques, the design and implementation of the assets remained mostly identical, with no major visual differences between the two.

From the start of this research, my goal was to use simple visuals, following the optimization logic found in the literature review. This meant avoiding high detailed textures and complex mesh geometry due to their impact on performance and instead using simple graphics and shapes. (Android Developers 2023b; Android Developers 2024a.) To help guide myself during this process, I created a moodboard consisting of references from games and animations that had similar visuals to what I was envisioning.

Due to time constraints, I chose not to draw full concept art for each part needed for the game. Instead, the moodboard served as the main reference for visual design, including objects such as furniture, buildings, plants, and decorative items. The theme I decided to go with was a fantasy cartoon aesthetic, characterized by flat colors, rounded shapes, and a nature-inspired color palette. This decision was also supported by the findings in the literature review, where I examined how flat colors are the easiest on mobile rendering performance-wise (Moscardó Ribes 2024). Although I did not have the time to create any more concept art, I did manage to draw at least one image in Krita that reflected the intended color scheme and shape language for the mobile game’s assets, as illustrated in Figure 15.

Figure 15. The only concept art I created for the prototype's visual direction (Orispää 2025).

My approach to creating this kind of simplified, rounded aesthetic was to use low polygon (low poly) modeling and create solid color blocks for any details. This approach also made it easier to match the appearance of both 2D and 3D assets using flat colors and cel shading. One of the motivations behind using cel shading was to avoid relying on real-time lighting to create depth. This was in line with Google's recommendation to avoid lighting where possible in mobile development (Android Developers 2024b). To create shadows on the 3D models, I chose to implement a toon shader, and for the 2D elements I had to draw the shading by hand.

Once I had gathered enough reference material, I began planning which elements would be implemented in either 2D or 3D. I had already planned from the beginning of this project that I would be integrating 2D and 3D in a 3D environment. Objects that were simple in shape would be made with 3D, while more detailed or intricate designs were drawn in 2D. This decision was influenced by the fact that I am more efficient when it comes to 2D asset creation than I am at 3D modeling. For example, as illustrated in Figure 16, the

large strawberry structure was created as a 3D model, while the strawberry planter boxes next to it are 2D sprites.

Figure 16. Example of how I blended 2D and 3D together in the prototype (Orispää 2025).

Another design choice I made early in development was to create a 2D playable character utilizing pseudo-3D techniques. The character was animated in eight directions, which when moving around creates the illusion of volume and 3D movement, despite it being a fully 2D rig. This is a technique often referred to as pseudo-3D. (Wikipedia 2024.)

From the beginning of this research, I had a clear visual goal for how I wanted to approach 2.5D implementation within mobile game development. During the process of this project, I deepened my understanding of asset creation and mobile optimization techniques.

5.2 Implementation of 2.5D Techniques

The 2.5D techniques I utilized in my prototypes included billboarding, pre-rendering 3D models as reference bases for 2D elements, and applying a toon shader to give the 3D objects a stylized appearance. To make sure that the 2D and 3D assets worked together visually, I used an isometric camera view to lock

the perspective. This way when the camera moves around in the game, the 2D elements did not visually break.

The first step in developing my prototypes was creating a playable character. I followed a workflow inspired by the methods used by Twin Motion to make *Dead Cells*, where a 3D model is used as a reference base for the corresponding 2D element (Vasseur 2018). Although it took me an entire day to make the 3D model, it significantly sped up the drawing and animating process. Using the model as a base, I was able to assemble and almost fully animate the 2D character the following day. This 3D base was particularly useful for animating the character turnaround, as illustrated in Figure 17.

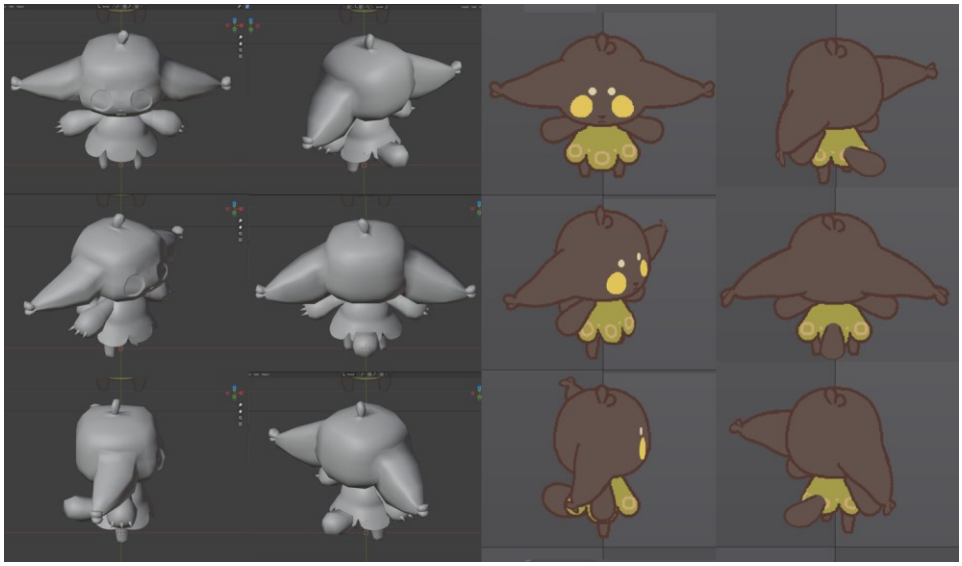

Figure 17. Comparison of the 3D models (left) used as a base for the 2D character art (right) (Orispää 2025a; Orispää 2025b).

With the character base ready, I moved on to setting up the game environment in Godot. At this point I had implemented one of the most important parts of 2.5D in my game: an isometric camera view. This is what influenced the overall visual appearance of my game by creating a distinct angle from which all the objects in the game are viewed. While I was set on the graphical projection, I did have some back-and-forth on how steep of an angle I wanted the camera to be at (see entries on February 18 and March 8).

Initially, I angled the camera to 45-degree on the X-axis and set it to follow the player character. However, I later adjusted it to 35-degree, as I did not like the way the 3D models were being distorted at 45-degree. I also experimented with rotating the entire environment container node in Godot to a 45-degree angle on the Y-axis to achieve a more isometric look. This created problems with the orientation of the 2D assets, which I had to then counter-rotate manually when placing them in the environment.

Before moving on to developing the 3D assets, I created a texture atlas containing flat colors and gradients. This served as the texture base for the UVs of the 3D models, and as the color palette for drawing the 2D sprites. Unlike the methods described by Moscardó Ribes (2024) in his article about gradient textures, where gradients are used as lightweight shading methods, I used the gradients in my color palette mostly for visual experimenting. I only created a few color patches at first for test purposes, and then on March 8th I added more colors and the gradients to the texture atlas. Even though I made quite a few gradient texture patches, in the end I only used it on one model, as illustrated in Figure 18.

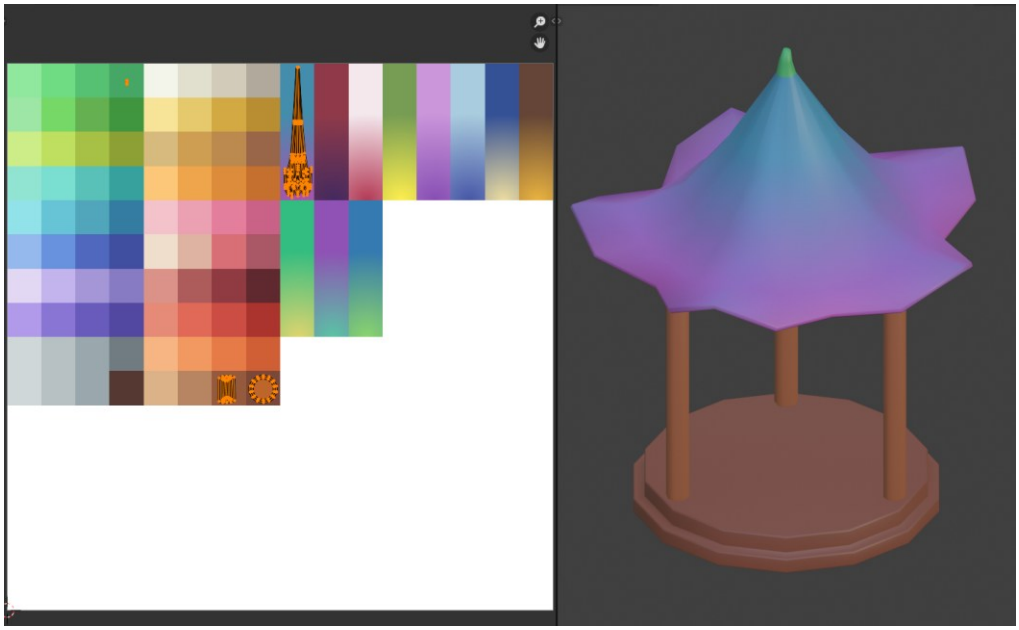

Figure 18. Example of how I mapped the gazebo 3D model's UVs to the color texture atlas (Orispää 2025).

When it came to asset creation and implementation, I first started with modeling the 3D assets. Similar to the idea of pre-rendering, it was easier to have at least one or two 3D meshes to use as a visual reference for when I started designing the 2D sprites. The first 3D model I started creating was the strawberry shaped hut, as I had previously made concept art for it. In a way I did the reverse of the pre-rendering method, where I used the 2D art as a reference base and matched the 3D to the general shape.

With at least one 3D asset for reference, I then moved on to making the 2D sprites. The sprites were drawn directly onto a texture atlas in Krita, as illustrated in Figure 19. For the unoptimized version of the prototype, I exported each individual sprite separately from this texture atlas.

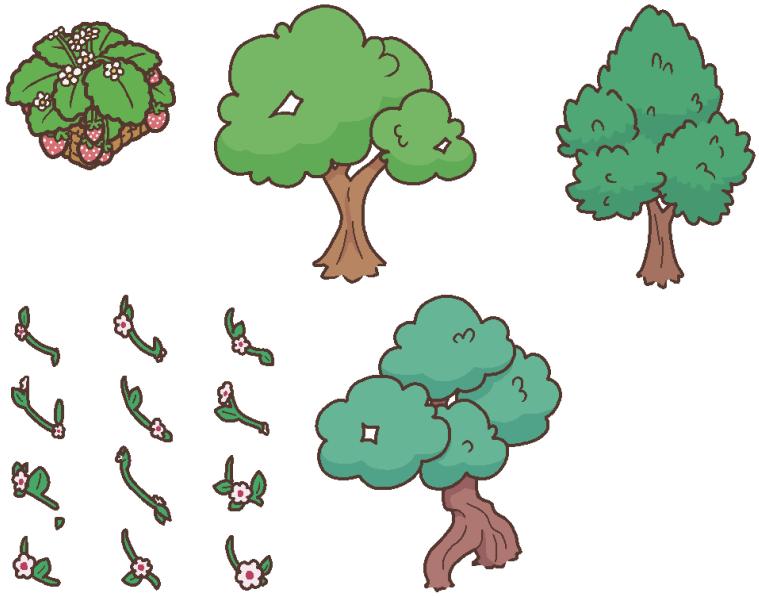

Figure 19. Texture atlas containing all the sprites for the prototypes (Orispää 2025).

To integrate the sprites into Godot, I used 3D sprite nodes to billboard the sprites into the game environment (Linietsky et al. 2025b). While no further steps were required when working with the individual sprites, integrating the full texture atlas for the optimized prototype required a workaround.

Using Blender, I UV mapped quads to specific portions of the texture atlas for each sprite, as illustrated in Figure 20. I then exported these quads into Godot, and using empty 3D nodes in Godot I billboarded the meshes into the game environment. This allowed me to use the texture atlas, optimizing the draw calls in this version of the game (Linietsky et al. 2024b).

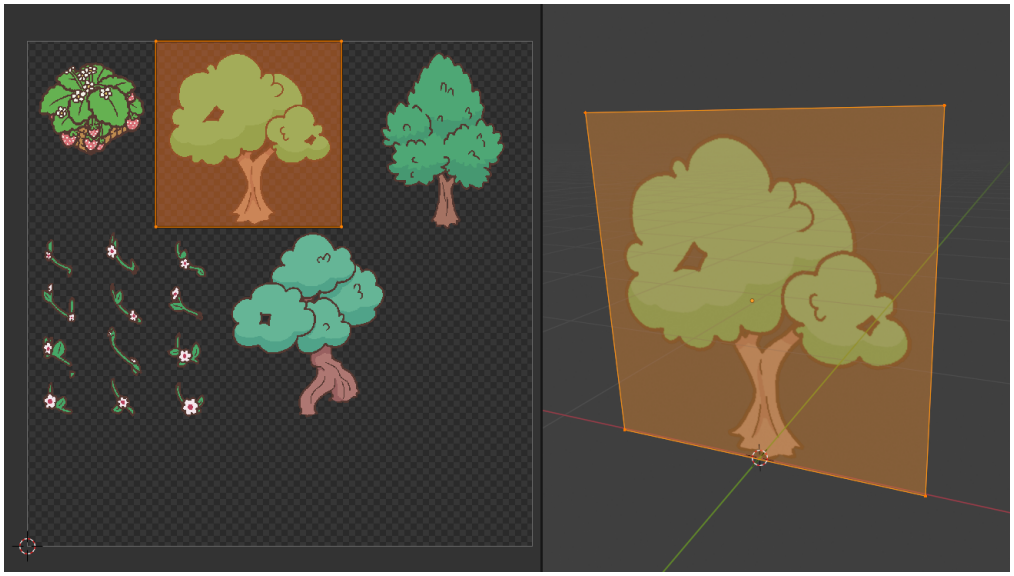

Figure 20. UV mapping the quad (right) to the sprite texture atlas (left) (Orispää 2025).

The final 2.5D technique I utilized was toon and cel shading. For the 2D elements, shading only required simple coloring directly on the texture sheet. However, to achieve the same cartoony effect on the 3D models, I needed to use a custom shader. Since I am not a programmer, and coding was not the focus of this project, I chose to use a shader from Godot's asset library. Fortunately, I found a toon shader shared by a Godot user under a CC0 license. With assistance from both an acquaintance and AI tools, I was able to implement the shader into my prototype. A link to this shader is documented in my March 14 research diary entry.

5.3 Challenges and Solutions Faced During Development

Throughout the development of the prototypes, I encountered a number of challenges with implementing 2.5D and game development in general. Most of these issues involved the integration of 2D and 3D elements, especially when trying to achieve a cohesive style. Other challenges were more technical, like drawing 2D elements with the correct perspective, and scaling them properly in the environment.

The greatest challenge I faced was integrating the 2D sprites into the 3D environment. Although I used 3D bases as perspective references when drawing the 2D sprites, it was still difficult to make the sprites match the 3D meshes cohesively. The outlines of the 2D sprites were often thinner visually compared to the outlines of the 3D meshes. Additionally, because the 2D sprites could not be rotated it was tedious to have to redraw a sprite again if I was not satisfied with the appearance of a sprite.

One way I tackled this problem, especially with the 2D sprites that were to be placed alongside a 3D mesh, was to first create the mesh and decide on the final positioning and orientation. Once I was content with the positioning of the 3D mesh, I could then create the sprite and place it where it needed to be. The gazebo mesh was the only 3D model where I truly attempted to blend 2D and 3D together. I added 2D vine meshes along the columns, in an optimized version of this mesh, as illustrated in Figure 21.

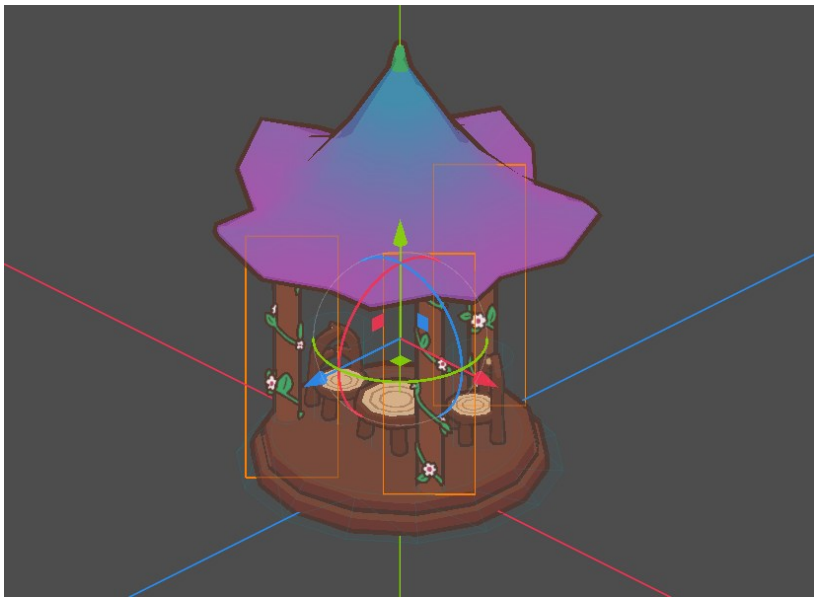

Figure 21. The 2D sprites were placed after finalizing the 3D meshes orientation (Orispää 2025).

Another issue I had while developing the prototype was getting the right camera angle in isometric view. The positioning of the camera distorted the sprites in the environment by essentially squashing them when rendered on screen. This is a problem I encountered in a previous game project I worked on. One solution that was considered for countering this problem was to rotate the sprites to match the camera angle, but this can lead to complications such as clipping and broken collision boxes.

Instead, I opted to stretch the sprites vertically to compensate for the camera distortion, the solution I also used in that previous project. Using AI, I calculated the needed adjustments for a 35-degree downward angle, which came out to stretching the sprites by 1.155 on the Z-axis. Figure 22 illustrates a comparison between a non-stretched and stretched sprite.

Figure 22. Comparison of the same sprite without stretching (left) and with the calculated stretch (right) (Orispää 2025).

Matching the visual appearance of shadows between the 2D and 3D elements was also challenging. The 3D meshes used the toon shader I had previously mentioned in Chapter 5.2, which was simple to adjust. The 2D sprites required me to hand draw the shading. Using the visual style produced by the shader on the 3D meshes, I essentially just tried to draw and match the 2D shading as cohesively as I could, as illustrated in Figure 23. This manual approach worked for this prototype, since the game does not use any real-time lighting.

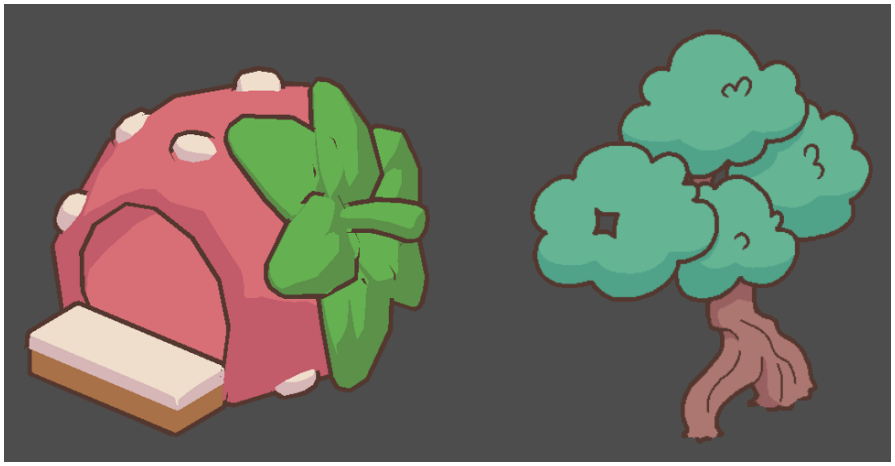

Figure 23. 3D model with toon shading (left) and hand-drawn 2D sprite shadow (right) (Orispää 2025a; Orispää 2025b).

If I had considered using dynamic lighting in these case study, I might have used normal maps for the 2D sprites, the same method used in *Dead Cells*

(Vasseur 2018). However, because I am more efficient with 2D art than 3D modeling, drawing the shading manually was faster for the scale of this prototype.

Other than issues with blending 2D and 3D, I encountered several smaller issues due to my lack of experience with game development and the Godot engine. One frequent problem was handling sprite flipping and collision boxes. When sprites were mirrored in the scene, their collision box would cease to work. I noted this issue on March 12 in my research diary but also wrote that it may be a bug specific to Godot rather than an error in my implementation.

6 Results and Performance of the Game Prototype

6.1 Benchmarking Performance of the Prototypes

To evaluate the performance of the mobile game prototypes, I used Godot's built-in profiler to observe the performance metrics across two Android devices and a PC. The mobile devices used for testing were a Samsung Galaxy A54 (2023) and a Xiaomi 11 Lite (2021). Despite the differences in release years and slight differences in hardware capabilities, both devices produce nearly identical results when running the prototypes, likely due to the small project scope and limited computational load.

Godot provides a wide range of performance metrics to track, but due to time constraints and my limited understanding of engine rendering and optimization, I focused on a select few. These included frames per second (FPS), video memory usage (VRAM), overall game file size, GPU usage, and the number of draw calls and rendered primitives.

While analyzing these metrics, I noticed minimal differences in performance between devices, highlighting the relatively low demands of the prototype builds. However, when comparing optimized versus unoptimized versions of the games, more worthwhile performance differences were noticed.

Table 1 presents the general differences between the two prototypes. Both versions shared the same overall setup, with several identical 3D models and the 2D sprites placed in the same spots in both scenes. The main difference was that the optimized prototype used a texture atlas for the 2D sprites, while the unoptimized version used individual sprite files. Additionally, one 3D mesh was specifically used to test mesh optimization, which affected the primitive count in the unoptimized prototype.

Table 1. Comparison of the optimized and unoptimized prototypes.

Properties	**Optimized**	**Unoptimized**
Objects	67	64
Primitives	30 733	57 783
Draw Calls	67	64
VRAM	4.44 MiB	8.13 MiB

The most notable difference between the two versions was in VRAM usage, with the unoptimized prototype consuming nearly 49% more memory than the optimized version. The difference in primitive count was primarily due to one 3D model in the unoptimized version, which had a significantly higher vertex count compared to the simplified mesh used in the optimized prototype, as illustrated in Figure 24.

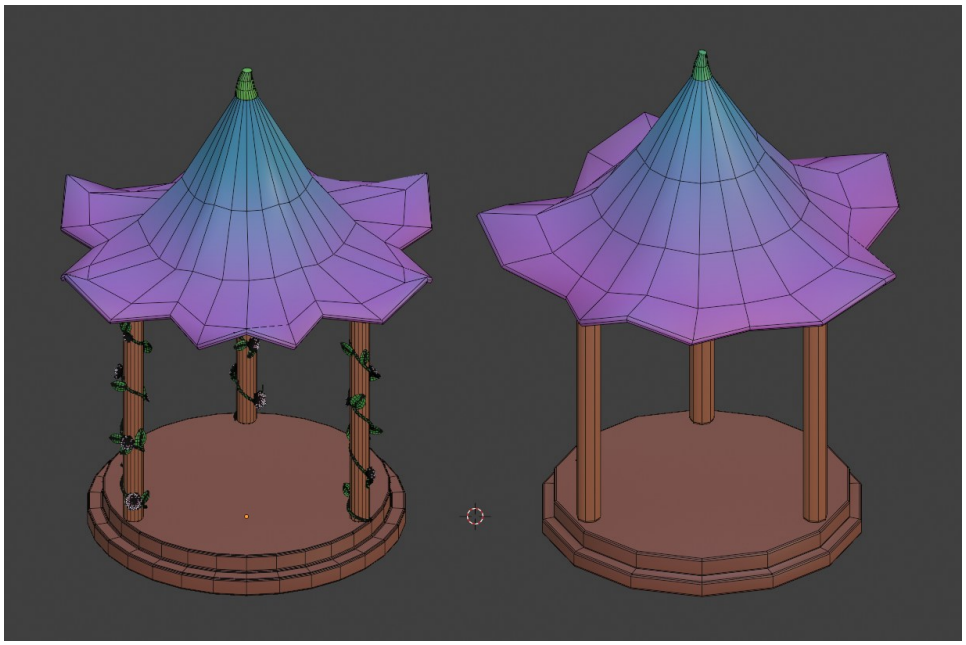

Figure 24. Comparison of high-poly (left) (12,336 vertices) and low-poly (right) (978 vertices) gazebo models (Orispää 2025).

To further test performance differences, I created two new prototype scenes specifically for testing the performance differences between the low poly gazebo model and the high poly version. Both of these scenes contained a total of 150 instances of the gazebo models, maybe somewhat excessive, but the purpose was to do an extreme stress test. However, this test had a major flaw, as I have noted in the final entry of my research diary on April 6. The main issue being the 2D vines used with the low poly 3D model. In my testing I did not think of creating separate sprites just for the vines, so when rendering, the whole sprite texture atlas was being drawn. Still this test illustrated the risks of unoptimized asset usage, as shown in Table 2.

Table 2. Performance properties of the two heavily unoptimized scenes.

Properties	**Low Poly Model**	**High Poly Model**
FPS	36	50
Processing	28.65 ms	26.69 ms
Memory (Static)	177.9 MiB	155.1 MiB

Properties	Low Poly Model	High Poly Model
VRAM	1.66 GiB	4.44 MiB

Despite having fewer vertices, the low poly scene with the sprite-based vines used significantly more VRAM due to inefficiency texture handling. It also significantly impacted the performance of the Godot editor on my PC, as when I tried to open and edit the scene it caused a significant lag to occur. This demonstrates the importance of texture optimization and how, if poorly done, can lead to severe performance issues, even on PC let alone on mobile. I chose not to test these scenes on mobile, given the already substantial performance drop on desktop.

6.2 Limitations of the Approach

Due to the small scale of the mobile game prototypes at the time of writing this thesis, it was difficult to draw any conclusive results in terms of performance. For more reliable results, larger scenes with more game objects and rendering demands would have been necessary. Additionally, my approach to performance testing lacked consistency and control, making the comparisons between the different builds limited in validity.

Another significant limitation was the small range of mobile devices available for testing. I was only able to test on two different Android devices, but in a testing environment with more valid testing methods, I would have needed a wider range of testing hardware, ranging from low-end to high-end devices. Furthermore, iOS devices were not even considered for testing during my research, despite Godot having full iOS build and renderer support.

Time constraints also affected the depth of my testing. I was only able to create one 3D mesh with two different polygon counts, but did not test how that affects performance through Godot’s tile rendering on mobile. Furthermore, since my game used an orthographic camera, I could not evaluate how the high poly model might have impacted performance when viewed from different distances.

Finally, it would have been beneficial if I had spent more time reading documentation on Godot's internal systems. Although I had a basic understanding of concepts like draw calls, I did not fully learn how Godot manages and optimizes materials and shaders behind the scenes. I mentioned in my research diary on the entry for March 25 that when trying to see the file size of the actual game application on mobile devices, it remained the same no matter if it was the optimized or unoptimized version.

6.3 Summary

Although the prototypes were small in scale and it was difficult to draw definitive conclusions, there were still noticeable differences in performance between the optimized and unoptimized versions of the mobile game. Using texture atlases showed how they improve VRAM performance, and even a scene built with low poly meshes can still cause significant performance issues if the textures attached to them are not optimized properly.

An interesting part that I noticed when using Godot was the minimal amount of optimization settings I needed to adjust. While a larger project would require a more extensive understanding of optimization settings available in the engine, when making the prototypes I only used a few. These included enabling VRAM compression for imported textures, setting the transparency for the 2D sprites to alpha scissoring, and adjusting their transparency threshold. For the 3D meshes, there were no optimization methods used in Godot, they fully relied on optimization through topology when being made in Blender.

It is important to note that not all the testing between the different versions was entirely fair, and there is room for improvement in terms of experimental design. However, the aim of this project was to explore different 2.5D techniques and observe basic optimization performance, and this objective was successfully achieved despite the limitations in testing.

Overall, the mobile game prototypes I developed and the research around it serve as a starting point for further research. This project provides valuable

insight into the considerations involved in creating assets for mobile games and gave me an opportunity to deepen my understanding of Godot Engine and game development. While I had some prior experience with the engine, it was not extensive, and this project allowed me to explore it more thoroughly.

7 Conclusion

In this thesis, I set out to investigate the constraints surrounding mobile game development and how 2.5D methods can be leveraged to improve performance while maintaining visual appeal. This research began with a literature review that outlined the limitations of mobile hardware, particularly in processing power and energy efficiency, and examined optimization strategies to address these challenges. A key finding was the heavy resource cost of textures in mobile rendering, emphasizing the importance of effective texture management. Additionally, while 3D processing is demanding on mobile devices, assets can be optimized through a solid understanding of mesh topology and efficient geometry management.

The Godot Engine was explored as a tool for mobile game development, demonstrating a strong emphasis on GPU-side optimization, such as default texture compression and automatic material and shader reuse. I then examined the concept of 2.5D, originally used as a workaround for limited hardware capabilities in early PC and console game development. While today it is mostly used as a stylistic choice, 2.5D techniques can still provide performance benefits when it comes to mobile game development.

Building on the information from the literature review, I compared 2D and 3D asset creation and highlighted the importance of optimization in both areas. Whether a game is fully 2D or 3D, understanding how to manage and optimize textures is essential for performance. In terms of 3D asset creation, reducing triangle count, maintaining clean mesh structure, and opting for shading methods that minimize the reliance on real-time lighting is the starting point for performance optimization.

The concepts studied were then put into practice in a case study involving two prototypes of the same game developed using different asset creation methods. By blending 2D and 3D elements in a 3D environment using 2.5D techniques, such as an isometric camera and billboarding 2D sprites, I explored the visual and technical impact of these methods. Although the prototypes were small in scope, the results offered insight into performance, particularly in terms of video memory usage. The use of texture atlases for 2D assets proved to be more efficient than individual sprite rendering. While the testing was limited to a couple of mid-range devices, the findings still highlight practical strategies for optimizing mobile games in Godot.

Through researching this topic, I gained valuable insight and deepened my understanding of asset creation and optimization in mobile games. Although the scope of this thesis covered only the fundamental elements, it serves as a starting point for further exploration. 2.5D methods have potential for mobile games, and there is much more to be studied in terms of implementation and comparative analysis of different techniques. As Godot continues to evolve, its focus on GPU optimization makes it a promising tool for future game development. There is still a great deal of information in this field to explore and understand, and I intend to pursue further knowledge and experience in mobile game development and optimization.

References

Amir 2023. Art and Science of Shaders in Video Games. Website. Medium. https://medium.com/@7019727855a/art-and-science-of-shaders-in-video-games-1953103cdefa (accessed 9.4.2025).

Android Developers 2023a. Reduce game size. Website. Android Developer. https://developer.android.com/games/optimize/game-size (accessed 13.4.2025).

Android Developers 2023b. Geometry. Website. Android Developer. https://developer.android.com/games/optimize/geometry (accessed 13.4.2025).

Android Developers 2024a. Textures. Website. Android Developer. https://developer.android.com/games/optimize/textures (accessed 13.4.2025).

Android Developers 2024b. Materials and Shaders. Website. Android Developer. https://developer.android.com/games/optimize/materials (accessed 13.4.2025).

Apple Developer n.d. Maximum build file sizes. Website. Apple Developer. https://developer.apple.com/help/app-store-connect/reference/maximum-build-file-sizes/ (accessed 3.3.2025).

Avisekhar, Roy 2016. The Android Game Developer's Handbook. E-book. Birmingham: Packt Publishing Ltd. https://metropolia.finna.fi/Record/nelli15.3710000000842341?sid=4965802996 (accessed 24.3.2025). Limited access.

Bernardi, Joe 2018. Texture Atlasing: An Inside Look At Optimizing 3D Worlds!. Website. Medium. https://medium.com/immersedteam/texture-atlasing-an-inside-look-at-optimizing-3d-worlds-8a07145856d7 (accessed 24.3.2025).

Budiu, Raluca 2015. Mobile User Experience: Limitations and Strengths. Website. NN/Group. https://www.nngroup.com/articles/mobile-ux/(accessed 26.2.2025).

Duraj, Maciej 2023. How Square Enix's HD-2D Gaming Aesthetics Are Bringing New Life to Gaming. Website. Medium. https://maciejduraj-75226.medium.com/how-square-enixs-hd-2d-gaming-aesthetics-are-bringing-new-life-to-gaming-2fb6128e7a0d (accessed 3.4.2025).

Game Domain 2018. What is a 2.5D Game? Online video 11.6.2028. YouTube. 3:15. https://www.youtube.com/watch?v=3y87Hk48BVI (accessed 20.3.2025).

Game Informer 2024. The Art of Hades II with Supergiant Games' Jen Zee | Interview. Online video 6.6.2024. YouTube 16:25. https://www.youtube.com/watch?v=4r9vkQ7i3Fg (accessed 3.4.2025).

Godot Contributors 2023. Godot 4.0 sets sail: All aboard for new horizons. Website. Godot Engine. https://godotengine.org/article/godot-4-0-sets-sail/ (accessed 3.4.2025).

Google n.d. Optimize your app's size and stay within Google Play app size limits. Website. Google Play Console Help. https://support.google.com/google-play/android-developer/answer/9859372?hl=en (accessed 6.3.2025).

Linietsky, Juan & Manzur, Ariel & the Godot Engine community 2024a. 3D rendering limitations. Website. Godot Engine. https://docs.godotengine.org/en/stable/tutorials/3d/3d_rendering_limitations.html (accessed 13.4.2025).

Linietsky, Juan & Manzur, Ariel & the Godot Engine community 2024b. GPU optimization. Website. Godot Engine. https://docs.godotengine.org/en/stable/tutorials/performance/gpu_optimization.html (accessed 13.4.2025).

Linietsky, Juan & Manzur, Ariel & the Godot Engine community 2025a. Introduction. Website. Godot Engine. https://docs.godotengine.org/en/stable/about/introduction.html#about-godot-engine (accessed 13.4.2025).

Linietsky, Juan & Manzur, Ariel & the Godot Engine community 2025b. Optimizing 3D performance. Website. Godot Engine. https://docs.godotengine.org/en/stable/tutorials/performance/optimizing_3d_performance.html (accessed 13.4.2025).

L'Italien, Ryan 2024. Godot vs. the Giants: What is Godot and How It Competes with Unity and Unreal. Website. Perforce. https://www.perforce.com/blog/vcs/what-is-godot (accessed 3.4.2025).

Moscardó Ribes, Vicente 2024. Optimization of 3D Texturing for Mobile Games: Definition of the 'Gradient Texturing' Process and Analysis of its Efficiency and Performance in Engine. Website. itch.io. https://itch.io/blog/797457/optimization-of-3d-texturing-for-mobile-games-definition-of-the-gradient-texturing-process-and-analysis-of-its-efficiency-and-performance-in-engine (accessed 31.3.2025).

Muhammad, Isa 2024. Mobile accounts for 49% of global games market share, despite 2% revenue dip. Website. Pocket Gamer.biz. https://www.pocketgamer.biz/mobile-accounts-for-49-of-global-games-market-share-despite-2-revenue-dip/ (accessed 20.3.2025).

Root, Dan 2019. The Beauty of Parallax. Online video 19.4.2019. YouTube 10:22. https://www.youtube.com/watch?v=z9tBce8eFqE (accessed 1.4.2025).

Sergeev, Arti 2021. Mobile Game Development: Differences, Challenges, New Solutions. Website. 80 Level. https://80.lv/articles/mobile-game-development-differences-challenges-new-solutions/ (accessed 24.3.2025).

Taylor, Derrick 2024. Smoothing the 3D 2D Pipeline for Art Assets in Games. Online video 26.9.2024. YouTube 1:25:04. https://www.youtube.com/watch?v=Wp5QRppLBQE (accessed 16.3.2025).

Udacity 2015. Atlases vs Spritesheets - HTML5 Game Development. Online video 23.2.2025. YouTube. 0:51. https://www.youtube.com/watch?v=tDGshZfBx5c (accessed 29.3.2025).

Vasseur, Thomas 2018. Art Design Deep Dive: Using a 3D pipeline for 2D animation in Dead Cells. Website. Game Developer. https://www.gamedeveloper.com/production/art-design-deep-dive-using-a-3d-pipeline-for-2d-animation-in-i-dead-cells-i- (accessed 3.4.2025).

Verdict n.d. What is mobile gaming and why does it matter?. Website. Verdict. https://www.verdict.co.uk/what-is-mobile-gaming-and-why-does-it-matter/ (accessed 13.4.2025).

VFX Apprentice 2023. What is 2.5D? The Look Dominating Animation and Video Games. Website. VFX Apprentice. https://www.vfxapprentice.com/blog/what-is-2-5d-animation-games (accessed 20.3.2025).

Wikipedia 2025a. 2.5D. Website 28.3.2025. https://en.wikipedia.org/w/index.php?title=2.5D&oldid=1282781889 (accessed 13.4.2025).

Wikipedia 2025b. Doom (1993 video game). Website 5.4.2025. https://en.wikipedia.org/w/index.php?title=Doom_(1993_video_game)&oldid=1284109645 (accessed 13.4.2025).

Image References

Figure 1. Linietsky, Juan & Manzur, Ariel & the Godot Engine community 2024. Mesh level of detail (LOD). Godot Engine. Website. https://docs.godotengine.org/en/stable/tutorials/3d/mesh_lod.html (accessed 13.4.2025). CC BY 3.0.

Figure 2. New Super Mario Bros. U 2012. Nintendo. Screenshot from a game.

Figure 3. Octopath Traveler 2018. Square Enix. Screenshot from a game.

Figure 4. Paletta, Claas n.d. Parallax scrolling layers in the computer game „The Whispered World“ – side view. Wikimedia Commons. Website. https://commons.wikimedia.org/wiki/File:The_Whispered_World_parallax_scrolling_sample_1.jpg (accessed 6.4.2025). CC BY-SA 3.0.

Figure 5. CMG Lee n.d. Comparison of graphical projections. Wikimedia Commons. Website. https://commons.wikimedia.org/wiki/File:Comparison_of_graphical_projections.svg#/media/File:Comparison_of_graphical_projections.svg (accessed 6.4.2025). CC BY-SA 4.0.

Figure 6. Image Collection. Images from left to right.

- Hades 2020. Supergiant Games. Screenshot from a game.
- Hades II 2024. Supergiant Games. Screenshot from a game.

Figure 7. Image Collection. Images from left to right.

- Bernardi, Joe 2018. Texture Atlasing: An Inside Look At Optimizing 3D Worlds!. Medium. Website. https://medium.com/immersedteam/texture-atlasing-an-inside-look-at-optimizing-3d-worlds-8a07145856d7 (accessed 29.3.2025).
- Esoteric Software 2020. Embedding assets with Spine Web Player. Esoteric Software. Website. https://esotericsoftware.com/blog/Embedding-assets-with-Spine-Web-Player (accessed 29.3.2025).

Figure 8. Chen, Michael 2020. Working With Sprite Sheets. Medium. Website. https://michaelychen.medium.com/working-with-sprite-sheets-2cbca2d1938f (accessed 29.3.2025).

Figure 9. Orispää, Nicolette 2025. Screenshot comparison from mobile game prototype.

Figure 10. Orispää, Nicolette 2025. Screenshot of a texture file size comparison.

Figure 11. Linietsky, Juan & Manzur, Ariel & the Godot Engine community 2024. 3D rendering limitations. Godot Engine. Website. https://docs.godotengine.org/en/stable/tutorials/3d/3d_rendering_limitations.html (accessed 8.4.2025). CC BY 3.0.

Figure 12. Maxon 2025. Polygon Reduction. Screenshot from website https://www.maxon.net/en/cinema-4d/features/polygon-reduction?srsltid=AfmBOoraVKl8CkL9YZVV1Dk_s4mmYc4D5Hm7GCUhxrhS4EYiPJ80k5ku (accessed 13.4.2025).

Figure 13. Tschmits 2008. UV Mapping. Wikimedia Commons. Website. https://commons.wikimedia.org/wiki/File:UVMapping.png (accessed 13.4.2025). CC BY-SA 3.0.

Figure 14. Moscardó Ribes, Vicente 2024. Optimization of 3D Texturing for Mobile Games: Definition of the 'Gradient Texturing' Process and Analysis of its Efficiency and Performance in Engine. Itch.io. Website https://itch.io/blog/797457/optimization-of-3d-texturing-for-mobile-games-definition-of-the-gradient-texturing-process-and-analysis-of-its-efficiency-and-performance-in-engine (accessed 9.4.2025).

Figures 15–24. Orispää, Nicolette 2025. Screenshots from the mobile game prototypes.

Appendices

Appendix 1. Research Diary

Date	Activity	Hours	Key Insights / Reflections
11.1.2025	Started planning my project as well as making a schedule for both this project and my thesis. Kind of 50/50 on whether to ask my previous internship if I can use my work that I did for them as part of my thesis. Also brainstormed a project idea for if I decide not to do the project with my previous employer. If I do the project from scratch, I'll take inspiration from my internship and make a mobile game that integrates both 2D and 3D. The challenge at this point is to come up with a game idea and art style that's not a straight up copy of what I did last fall.	3	Planning own project that's not a copy of previous work. Considering the definition of 2.5D and why would I use it.
15.1.2025	Trying to narrow down my project and thesis topic today. So far, my topic is leaning towards creating a visually cohesive mobile game by blending 2D and 3D together. My plan so far with this is to understand how to blend 2D and 3D together, for example through perspective, color, and lighting. By understanding these fundamentals, I will then implement this knowledge into my project so that I can use it as a case study.	5	**Possible research question:** How to blend 2D and 3D together cohesively in a mobile game.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
17.1.2025	Planning thesis and project that will be done together. I had the idea of making custom texture overlays for my game's visuals, so I tested out making my own in Krita and brought them into Godot to test how I could apply it onto basic shapes. I may need to edit my brush preset that I'm using because the pattern is awfully repetitive and looks a little rough.	3	Early testing of visual design and direction.
19.1.2025	Gathering information for my thesis and project topic. Finding sources for game development is kinda difficult. There's plenty of game development documentation, but actual academic research is fairly limited. Same with mobile app development. There's a limited amount of research, and a lot of the sources I've found so far are a few years old, so I'm wondering about the validity.	2	Academic research on topic is limited. Finding reliable sources is a challenge.
20.1.2025	Prepared a presentation for my topic suggestion and made a rough project proposal to go with my thesis. My topic hasn't changed too much in the past couple of days, but I found some thesis that I can hopefully read through and use their sources to help with my topic.	5	Found other academic research to build off of.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
21.1.2025	Doodled some concept art for my project while sitting in class today. So far, my vision for my game is a cute cozy forest vibe. There's a slight problem with my game ideas visuals, as they are kind of similar to my internship's work, but I'll try my best to come up with different visuals. I also plan on making completely different looking characters for my game, so hopefully that helps separate my project from my internship work.	2	Deciding on project's visual direction. Potential overlap with past work might be a problem.
25.1.2025	Brainstorming project ideas. Idea so far: 2D/3D card game Had a brainstorming session with my partner just to get the ball rolling with my project. Came up with an idea that the game could be an isometric environment, where you can control the playable character and challenge NPCs to a simple card game. The NPCs could be Spine 2D rigs, which is the 2D aspect of the game, and then the game environment assets could be 3D. Since the camera view will be isometric, I have to take that into account when making the sprites so that they look good at a top-down view.	2.5	**Key visual direction:** Isometric camera. Design choice of 2D characters in a 3D environment.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
28.1.2025	Created a moodboard and started to take notes from similar games that have inspired my project. Most of the games I have on my list are not on mobile, except for *Octopath Traveler*. I've put some image references from *Hades* just because I love the art style. Thinking if I could implement a simple character control system so that the playable character can move in eight directions, but that also means that I would need to design the Spine rig to move in those directions as well. *Cult of the Lamb* is another game that I'm taking visual inspiration from. It might be one of the biggest inspirations besides my internship work.	3	Noted there's a lack of 2.5D games for mobile that can be used as an example. Considered an 8-direction movement 2D character. Inspiration from a variety of sources.
30.1.2025	Wrote a quick idea document to present tomorrow. Mostly wrote down everything I've thought of so far, and what programs I plan on using to make my project. I plan on making the 2D sprites using Krita and Spine and then 3D assets using Blender. I'll use the Godot game engine since it's open source and lightweight.	2.5	Decided on tools for creating the game: Krita Spine 2D Blender Godot
10.2.2025	Wrote down some text in the concept chapter of my thesis. Outlined the idea of my project and how I plan on incorporating it into my thesis. The overall idea hasn't changed too much, but I think schedule wise, I'll really only have time to create a game that is visually appealing and cohesive, and there won't be a whole lot of action or plot.	1.5	Decided to downscale my project outline. Concentrate on visuals and implementation rather than utility.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
11.2.2025	Research topic: 2.5D and reading through both Google and Godot documentation on optimizing mobile game assets.	3	Decided on 2.5D as research topic.
13.2.2025	Sketched out some concept art for the game and looked for Godot tutorials on whether it's possible to do an isometric styled game, but with a 3D environment. I may be hoping for the impossible, but I managed to find one video from GDC where someone showcased a playable 3D area that looked isometric. Need to give it a try. Afterwards I started sketching out how I want the overall map to look like. Right now, my idea is to make 5 zones, all with their own theme, but everything is connected together on one map. I also drew some concept art for items that could be placed in those zones. So far, the zone themes are: Forest, Berry Garden, Mushroom Grove, Fairy Grove, and Quilt Land. I also did some character designing. They need to be made with simple shapes if I want to make the characters moveable in 8 different directions. I drew a few different character designs, before deciding to recycle an old character I made a few years back for another project. With that decided, I made a few variants of the character to fit the themes, and then I started to make a character turnaround for reference.	8	Researched whether implementing an isometric camera view in a 3D environment using Godot was possible and how. Designed a simple character that is easy to implement in a 2.5D environment.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
14.2.2025	Worked on designing the 2D characters today. At first, I sketched out the art facing straight towards the camera, but I realized that if my game is in an isometric view or top-down then the character art needs to be drawn from that angle. I was having difficulties getting the perspective right on the art, so I opted to make a simple 3D model that I'll use to help with perspective. I got the model to the point where I can use it for all the characters in the game and use it as a reference when drawing the art.	5	Designed a character that needs to work visually with a top-down camera angle. **Key 2.5D implementation:** Using pre-rendered 3D model as reference base for 2D.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
17.2.2025	Do I slightly regret the choice of making a whole stinking character turn around in Spine in top-down view? Maybe. Was it worth it? Absolutely. Learned a lot today which was fun. Focused on getting the base character rig done today. The character needs to be as simple as possible, because I want to animate it in eight different directions and have a limited amount of time. In order to avoid having to draw an astronomical amount of art, I made the character as round and simple as possible so I can create the illusion of 3D by rotating the 2D pieces around. I used the 3D model I made as reference and drew all the pieces that I'll need for Spine. Once the art was done, I exported the files into Spine and started building the rig. It took a few attempts to get everything aligned and looking correct. I had to go back and simplify some of the base art, since I'm trying to avoid making an overly complicated rig. After trial and error, I managed to create an animation with five directions, and now I just need to figure out how to mirror the duplicate directions and the turnaround is complete. I wasn't intending to make the character appear as if it were a 3D model, I just wanted to make the direction blend nicely, but it somehow worked out, so I'm hoping that it'll fit in with the game once I have the 3D models made.	9.5	3D model base significantly sped up the entire process of the 2D character art and animation. Successfully implemented pseudo-3D using the 2D character rig.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
18.2.2025	Finished the character turnaround today. You'd think it'd be as simple as mirroring keyframes for the duplicate directions, but it wasn't. I wrestled with the animation before finally getting it to do a full 360 or at least give the illusion of one. It's looking pretty pseudo-3D, if I do say so myself. Once that was done, I duplicated and split each direction, north-facing, south, northeast, etc. into its own file. My final goal was setting up the Godot project and getting the Spine rig moving on a plane. I'm not a coder and don't plan to focus on gameplay, so I used the help of ChatGPT, YouTube tutorials, and a friend to get the skeleton moving. Godot's workflow with Spine 2D is a bit messy, and I doubt I could've done it alone. The way Spine rigs run in a 3D Godot scene also feels awkward. The rig is attached to a subviewport, which I then billboarded into the environment. I also did some calculations for the sprite. Since it was designed for a straight-on camera, but the game uses a 3D camera angled down at 45-degrees, the sprites looked flattened. To fix this, I stretched the sprite by 1.155 on the Y-axis to match the angle and preserve its intended look. I got that tip from ChatGPT, but I still need to find proper documentation on the theory.	5.5	**Key 2.5D implementation:** Billboarded the character into the 3D game environment. 2D sprites get distorted when viewed top-down in a 3D environment. Fix for this was to stretch the sprite to negate the distortion.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
19.2.2025	Made some changes to the Spine rig. I added a tail to my character, which meant I had to fix the animation to include this new piece. Once the character was updated, I started working on the player movement. I had to use a lot of outside help, like AI, YouTube and my partner to make it work. I managed to get a character movement script that plays the animations I've made so far in the correct directions and flip the Spine sprite for the directions that are duplicates. It's pretty satisfying to see the Spine rig physics working in Godot. It adds to the 2.5D effect. I also managed to make the controls work in touch screen mode, so hopefully I can port the game to my phone at some point and test that out. Finally, I made some adjustments to the game environment and camera. I attached the camera to the character, so it follows the player around, rather than stay static. I've decided at this point that I want my game to be in an isometric view, so I adjusted the world Node in Godot to turn everything in it to a 45-degree angle. Easier to do this than make the camera 45-degrees because the character movement gets messed up. Taking visual inspiration from *Hades* at the moment. I really like the visual style.	4.5	Enhanced the workflow for the isometric camera. Rotated the world environment rather than the camera to create a isometric view, because everything placed here will automatically be rotated.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
21.2.2025	Wrote about my research method plan today. My research question is “How can 2D and 3D assets be integrated to create a 2.5D mobile game using the Godot Engine, while keeping optimization and visual cohesion in mind?” and my plan is to use my specialization project to test out different ways of creating assets, whether it’s comparing workflows, testing optimization techniques such as transparent image file sizes, vs non transparent images. Basically, just wrote about how I plan on connecting my thesis to my project.	3	Solidified my approach to 2.5D implementation and how it provides optimization
24.2.2025	Worked on prototyping the game environment as well as doing some character animations. Roughly blocked out the game area and where I wanted to place all the objects. I also spent a little time looking at references and gathering them to a moodboard for the game, as environment creation is not my strongest suit. Started working on character run animation. Not sure how many animations I’ll make, since it’s not the main focus.	3	Using references to make-up for shortcomings and to speed up the design process.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
25.2.2025	Worked on a few things today. Created a test ground texture for the full play area. Since the game is isometric but the plane is flat and rotated 45-degrees, I treated the texture as a flat image, no depth, given the 3D environment and orthographic camera. The first 512x512 version looked too rough, so I bumped it to 1054x1054, which feels like the upper limit. After importing into Godot, I saw the character camera was too zoomed out, and the texture looked stretched at 1x1x1 even with the higher res. I halved the prototype area and zoomed the camera in, which helped. The texture still looks a bit rough, so I might need to increase the resolution again. Can't scale the world down more or the character rig becomes oversized. Also worked on concept art for the strawberry zone. Came up with a solid idea: an NPC inside a hollowed-out strawberry with a table and card game. Made a base color palette texture for both 3D models and 2D art. Decorations will be 2D to handle their visual complexity.	4	Balancing visual quality with performance. Experimenting with ground texture resolution. Concept designing of blending 2D and 3D assets.

Date	Activity	Hours	Key Insights / Reflections
26.2.2025	Started working on making the 3D model based on yesterday's concept art. It’s been a hot minute since I did any kind of 3D modeling so I ended up spending more time on this than I probably should have, and the topology on my mesh isn’t the best. At one point the strawberry building part by itself had almost 2,000 tris, but I managed to fix it and now the total count for everything in the model is roughly 3,000. Still kind of hefty, but nothing crazy. Depending on how much time I have, I may redo the model just to see if I can improve it. UVs are a bit of a mess, but I need to research how much that affects run time. I’m using an image texture with a grid layout filled with different colors for the material. Learned during my internship that this is extremely lightweight and render friendly for mobile games. I also started working on the item sprite sheet. I made a strawberry planter as my first item. Right now, I have no idea what scale everything should be. The main sprite sheet is about 2048 x 2048, and the plant takes up about 256 x 256, but I’ll have to place everything in Godot and see how it looks. I made the art with a solid pen that has no transparency in it, but I want to compare the current version of this drawing with a version of the drawn using a brush with varying opacity pixels to see how big of a difference it makes in file size.	5.3	Optimization in 3D topology requires more practice and iteration. Visual scaling and asset testing is important. Especially when trying to blend 2D sprites in 3D environment.

27.2.2025	Research Alpha scissors, compression modes: Currently using VRAM Compressed as instructed by previous coworker, but I don't fully understand the reason for this. Spent most of my day learning how to code using GDScript. Not the greatest coder, nor have I done much coding in general, so this is probably the biggest challenge of making a game for me. Copy and pasted an outline shader from an old project I did a while ago to this current project. Adjusted the thickness and color of the line and it was good to go. At the same time, I made a material object for both 3D and 2D objects that I can just drag on to the assets once I've imported them into Godot. Made a base item scene that I can then use for all the assets, that's preloaded with colliders, NPC scene slots and all the base mesh transformations that I need to copy to all items. Battled with setting up an NPC Spine character for the game. Prepared a short presentation for tomorrow. Gathered material from everything I've done so far and put it into a PowerPoint. Realized that I had made the wrong calculations for stretching the 2D sprites!! Originally, I had calculated the stretch to match a camera view from 35 degrees, but since my camera is at a 45-degree angle, I need to stretch the 2D sprites by 1.414, not 1.155. I thought things looked a little off, and now I know it wasn't my imagination.	8	Coding is the biggest hurdle in all of this. Workflow optimization is important to speed up the process. Technical problem-solving is an ongoing problem. Correcting sprite stretch and distortion.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
5.3.2025	Information gathering on optimization techniques for mobile game development.	4	
7.3.2025	Started making more 3D assets for the game. Took inspiration from a cute flowery/fairy gazebo I saw on Pinterest, so I'm trying to recreate it in Blender. I tried to make the most of the structure as simple as possible, because I know the flower is going to be rather hefty in polygons. I made 3 prototypes for the roof, but I still feel like I could reduce the polygon count and simplify it more.	2.5	Balancing visual style with performance. **Iterative design process:** Created multiple prototypes of the same mesh to try and refine the design.

8.3.2025	Continued working on yesterday's model. I redid the roof of the gazebo and managed to reduce the polygon count. The old roof model alone had over 700 polygons, and the new one has about 450, so a pretty good reduction. I also lowered the polygons of the supporting pillars of the gazebo as they were a little unnecessarily detailed. I've been using a png image with a few solid colors in it as the color palette/textures for the two assets I've made so far, but it was lacking a lot of colors. I decided to spend a couple of hours making a bigger selection of colors as well as trying to create some gradient options. According to Godot's documentation, gradients are a little pricey and require a few extra steps to work in the engine, but I still want to try. I've made the color blocks a little bigger than they probably should be, but I can shrink them later if needed. Once I got the color palette done, I attached the gazebo UVs to it by simply projecting from view. I wanted the roof to at least use one of the gradient colors, and the rest of the structure was placed on solid colors. I brought the model into Godot, struggled with creating a ramp that the character can go up, made some collisions for the gazebo, and placed it in the game environment. I'm not sure if I like the 45-degree angle, so I turned the camera to 35 degrees, but that meant I had to rescale all the 2D elements to be 1.155 instead of 1.414 on the Z-axis.	5	More optimization through iteration by redesigning the gazebo mesh. Explored custom gradient despite the extra complexity in Godot. More camera angle adjustments to improve visual preference. Because of this had to rescale the sprites again.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
9.3.2025	Made more 3D models today. I feel like the gazebo asset looks a little empty/boring, so I tried making a garden table set for the inside. I also thought about making some vines/leaves that would entwine the gazebo, but doing that with 3D seems like it would increase the polygon count way too much. The other option would be to make a 2D sprite and create an illusion of vines covering the 3D asset. The garden table set was easy to do, and I'm pretty happy with the final results. The table has roughly 550 polygons and the chair is 1000. I could have reduced the count even further, but because I'm not using custom textures for my 3D models, in order to create the rings on a tree stump like in real life, I had to create some extra topology that I can place on the texture grid. Exported the assets into Godot and placed everything inside the gazebo. Looking pretty good but still lacking the plants. I'll need to work on that next.	2.5	Implemented 2D optimization by choosing to use 2D sprites over modeling 3D vines. Design trade-offs between detail and efficiency with the topology in the chair and table meshes. Iterative asset development.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
10.3.2025	Reading through more asset optimization documentation for mobile game development. Found some information on the maximum vertex count for a single mesh, which is about 65 535. That sounds like a crazy amount, and something only high-end devices are maybe able to run. Made me feel not so bad about the models I've made so far, as even the gazebo is only like 12 000 vertices. Also read through texture filtering modes and how they affect performance. I'll probably just stick to the nearest filtering on the 2D textures due to the shader outline on the 3D meshes, but I could give bilinear a try at some point.	4.5	Gained perspective on performance limits. Balancing fidelity and efficiency through using nearest filtering on the textures.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
12.3.2025	My laptop really struggles with 4K files, definitely a reminder to keep file sizes small. Next issue: GitHub won't accept the 4K texture since it's over 100MB, so I'll need another way to get it into the game. **Update:** I can't import any 4K textures for now, which isn't a huge deal, just highlights how hefty they are. I had to rethink the project scope. The original environment was too large for the time limit I have, so I'm scaling back to one small area with a few objects and some foliage. Current plan: two versions of the gazebo, one fully 3D high poly, and one low poly with 2D elements. Also planning two versions of the playable area: one optimized and one full quality, with uncompressed textures and un-atlased sprites. Made some 2D tree sprites and brought them into Godot using quads. The biggest challenge with this 2.5D style is making 2D sprites work in a 3D space. Everything's fine unless I try to mirror them, then collisions break, and the debugger throws rotation errors. I've tried various fixes, but so far nothing's worked. Might be a Godot-specific issue, but it's slowed me down today.	5	Scaled the project plan down even further due to time constraints. **Core optimization point:** 4k textures are incredibly expensive performance-wise. **Key challenge:** Visually blending 2D into a 3D environment is challenging.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
13.3.2025	Made the high poly gazebo model, and oh boy, it sure does have a lot of tris. I 3D modeled vines going up each pillar along with some leaves and flowers. The total count is 21 000 tris and 12 336 verts. My plan is to now use what I've done with this model as a base reference and make 2D sprites of the vines for the more optimized model.	2.5	Leveraged 3D models as base for 2D but also as a comparison for performance.
14.3.2025	Had to make the decision to scale down the prototype. I redrew the concept to be smaller and more focused. I planned a few quick to make items to drop into Godot soon for performance testing. Blending the 2D vine sprites with the 3D gazebo was tricky, it took a few tries to get right. My first attempt clipped outside the roof, so I ended up halving the drawing. The outlines looked too thin at first, but scaling the sprite up to fit the pillars fixed that. Once I had one version I liked, I made two more for variety. If I rotate the gazebo later, I'll need to tweak the sprites, but it's manageable. After the vines, I experimented with a toon shader for the 3D assets. I found a CC0 shader on Godot shaders and used ChatGPT and a friend to get it working. It now allows me to adjust light direction via XYZ scaling and tweak the shadow and light colors as needed. Finally, I tested a new idea: a large tree with a 3D trunk and 2D leaves. I modeled a round trunk and sketched the tree fully in 3D, but I'll have to test the 2D/3D combo later since I ran out of time today.	8	**Toon shader:** https://godotshaders.com/shader/update-botw-toon-shader/ Design iteration through trial and error with 2D sprites. Implemented 2.5D in 3D models using a 2D shader.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
24.3.2025	Set up the unoptimized prototype version. I exported uncompressed textures for the 3D models and set Godot's import to lossless. I also split the sprite texture atlas, so each sprite has its own uncompressed texture. Duplicated all 3D models and assigned them new materials using the uncompressed textures. Then I recreated the optimized setup using these unoptimized assets and tested it on my phone. **Some issues:** I forgot to apply the uncompressed material to the garden set in the gazebo, and I used a different name for the unoptimized texture, so I'll need to fix that. Testing wasn't consistent, the sprites weren't all visible at once, so draw call comparisons between versions aren't accurate. Also, the sprite atlas is too large with a lot of empty space. Despite using uncompressed individual sprites, their total file size is still smaller than the sheet. I need to reduce the atlas size.	3.5	Inconsistent asset usage and environment setups affected performance comparisons. Workflow iteration and error tracking. Optimization awareness where texture atlas needs fixing.

Date	Activity	Hours	Key Insights / Reflections
25.3.2025	More performance testing. Tried figuring out if there's a way to export only the scene of choice, so either the optimized or unoptimized Godot scene. So far there's almost no file size difference between the two, and in fact the unoptimized version seems to be smaller in size? Not really understanding how exporting works in Godot at the moment. Also need to remember to test the games with the FPS uncapped, to see if that changes anything. Hard to measure the FPS since both versions run perfectly fine at the current cap of 90 FPS.	1.5	Unclear understanding of Godot's export system is making things difficult. Optimization isn't always reflected in file size.

Appendix 1

Date	Activity	Hours	Key Insights / Reflections
6.4.2025	Did another performance test but I changed things up a little. Edited both versions of the gazebo so that each item is its own draw call or counts as its own separate item. So, in the case of the gazebo with the sprites, I made the sprites unique. Built two new scenes both with like 150 of only the gazebos, so one scene has the sprite + gazebo, and the other scene has the fully 3D gazebo. The scene with the sprite gazebo combo was definitely way bigger in video memory usage size clocking in at 1.656 GiB, and it took Godot a while to even open the scene. Meanwhile the fully 3D gazebo scene was 4.44 MiB, and not so heavy to open. The problem with my testing and this comparison is that the 2D sprite was drawing the whole sprite sheet, not just the vine sprite, so I don't think this was a fair comparison. If I wanted to make things fair, I should have made the 2D vine sprites their own sheets and done the test that way. But I guess this just shows how important understanding how textures work.	1.5	2D assets aren't always lighter if poorly set up. Testing must be consistent and equal. Texture management is absolutely essential and important to fully understand.