INVESTIGATION REPORT

Establishing a Framework for Interactive and Dynamic Vector Graphics Content in Real-Time Video Games

By

Cheng Yi Heng

TP058994

APU3F2408CGD

A report submitted in partial fulfilment of the requirements for the degree of

BSc (Hons) Computer Games Development

at Asia Pacific University of Technology and Innovation.

Supervised by Mr. Jacob Sow Tian You

2nd Marker: Assoc. Prof. Ts. Dr. Tan Chin Ike

27th November 2024

Acknowledgement

I would like to express my heartfelt gratitude to everyone who supported me throughout the development of **Velyst** and the **Lumina** game. First, I am deeply thankful to my teammates, whose dedication, collaboration, and creativity were essential in bringing this project to completion.

I also wish to extend my sincerest thanks to my supervisors, Mr. Jacob Sow Tian You and Assoc. Prof. Ts. Dr. Tan Chin Ike, for their invaluable guidance, insights, and encouragement throughout this journey.

Finally, I would like to thank my family and friends for their unwavering support and belief in me, which provided the motivation I needed to persevere through challenges. This project would not have been possible without each of you.

Abstract

Despite the widespread use of raster graphics in games, vector graphics remain underutilized and largely absent, with little integration and no established framework for their effective use in modern game development. This project introduces **Velyst**, a streamlined framework for integrating interactive and dynamic vector graphics into real-time video games. It leverages Typst for vector content creation and Vello for real-time rendering of dynamic vector graphics. By simplifying the process, this framework enables developers to produce high-quality and engaging content without needing to delve deeply into technical complexities. Our study employs purposive sampling to collect valuable insights from developers in the gaming and interactive application sectors. We will conduct three in-depth interviews with industry professionals to gain expert perspectives. Additionally, online survey questionnaires will be distributed to a broader developer audience to capture a wider range of opinions. This uncovers the challenges and opportunities of integrating vector graphics, with insights that contributes to the framework on reducing technical barriers, enhancing interactivity, and highlighting areas for further innovation in the field. This research aims to demonstrate the untapped potential of vector graphics in modern gaming and provide a practical solution for their seamless integration. Our approach contributes to advancing infrastructure and fostering innovation, aligning with the goals of *Sustainable Development Goal (SDG) 9*.

Keywords*: Typesetting, Markdown, Workflow, Dynamic content, Compute-centric, Typst*

SDG Goal 9: Build resilient infrastructure, promote inclusive and sustainable industrialization and foster innovation

Contents

Figures

Tables

Codes

CHAPTER 1: INTRODUCTION

1.1. Introduction

Vector graphics is a form of computer graphics where visual images are generated using mathematical formulae (Rick et al., 2024). This includes geometric shapes, such as points, lines, curves, and polygons that are defined on a Cartesian plane. The use of vector graphics in games can be tracked way back to when computer games was first developed. One of the earliest examples of video game, *Tennis for Two* as shown in Figure 1 uses vector graphics to render their game on a repurposed oscilloscope in 1958 (Filimowicz, 2023). It was not long before video games was first commercialized during the 1970s and 1980s, with the release of vector graphics rendered games like *Space Wars*, *Battlezone*, and *Tac/Scan*. These games showcases the potential of vector-based visuals to achieve fluid and interactive animations.

Figure 1: Tennis for Two (Higinbotham, 1958)

Around this time, graphical processing units (GPU) were also experiencing rapid development and growth. In 1989, Silicon Graphics Inc. (SGI) created one of the earliest graphics application programming interfaces (API) OpenGL, which forms the foundation of today's computer graphics software (Crow, 2004). As GPU advanced, support for raster graphics improved

significantly, leading to a decline in the use of vector-based rendering technology in gaming (Stanford, 2024).

Despite the rise of raster graphics, the unique benefits of vector graphics (scalability and precision) continue to offer significant potential in modern game environments. Today, vector graphics are rendered using high resolution monitors through the process of rasterization (Tian & Günther, 2022). This necessity led to the rise of algorithms specifically designed to convert mathematically defined shapes into pixels, creating a new domain of computational challenges. In addition, there is little to no tool available that effectively integrates vector graphics content into real-time, interactive game environments. The absence of such tools has hindered the widespread adoption of vector graphics in modern game development, limiting their use to methods like triangulation (DesLauriers, 2015) and sign distance field (Alvin, 2020) due to technical constraints.

1.2. Problem Statement

Scalability, precision, and animation

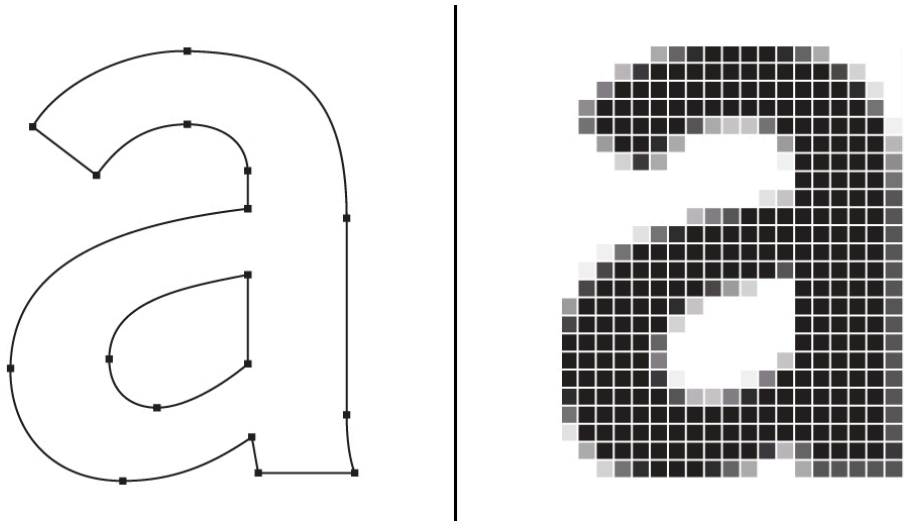

Figure 2: Vector vs Bitmap graphics (Ratermanis, 2017)

Traditional methods of rendering 2D graphics has always relied on bitmap-based texture mapping (Ray et al., 2005). While this approach is ubiquitous, it suffers a major drawback of the *pixelation* effect when being scaled beyond the original resolution (Nehab & Hoppe, 2008). Furthermore, creating animations using bitmap-based textures can be extremely limited and complex because of the rigid grid-like data structure used to store the data. Animating raster graphics are commonly done through the use of fragment shaders which directly manipulates individual pixels (Jeremias & Quilez, 2014), or relying on image sequences (a.k.a flipbooks) (Iché, 2016) which produces an illusion of movement.

Unlike raster graphics which rely on a fixed grid of pixels, vector graphics are precise and resolution independent. This means that it can scale without losing quality (shown in Figure 2). A vector illustration is composed of multiple *paths* that define *shapes* to be painted in a given order (Ganacim et al., 2014). Each of these individual paths can be traced, altered, or even morphed into a completely different shape which allows for a huge variety of animation techniques in real-time (Carlier et al., 2020; Dalstein et al., 2015; Mateja et al., 2023). Vector graphics animations can also react to dynamic runtime changes as the very definition of their shapes are extremely flexible. This property allows vector animations to be generated procedurally instead of relying on pre-recorded data (Hoyer, n.d.).

Lack of support on UI/UX creation for complex interactivity

WYSIWYG editors are visual centric tools that let users work directly within the presentation form of the content (Mädje, 2022). Most game engines in the market like Unity (Krogh-Jacobsen, 2023), Unreal Engine (Santos, 2023), and Godot (Johnson, 2024) uses a *What You See Is What You Get* (WYSIWYG) editor for creating user interfaces (UI) and user experiences (UX). In WYSIWYG editors, users normally layout their contents using a drag and drop method and then style them using a style-sheet (HeySERP-Team, 2023). To bind interactions or animations towards a content, users would need to label it with a unique tag and query them through code.

Complex content and logic would not be possible through a typical WYSIWYG editor. For instance, it is virtually impossible to author a custom polygon shape in the editor with custom math based animation which depends on a time value. This can only be achieved through code, and is often limited by the API layer provided by the WYSIWYG editor. This creates a huge distinction between the game/UI logic and the visual representation that is needed to convey the messages.

Slow iteration time and feedback loops in development

While hot-reloading is applicable for the layout and styling (and simple logic to some extend) of a content. In an Unreal Engine UI tutorial titled “UMG Best Practices” by Gagnon (2019), the author concluded that logic should be kept in the C++ language and Unreal Blueprints. It showcases that a WYSIWYG editor would not be capable of hot-reloading complex logic as these can only be achieved using code, which in most cases, requires a re-compilation. This could lead to frustration and lost of creativity due to the slow feedback loop.

1.3. Project Aim

This project aims to establish a framework that enables the integration of interactive and dynamic vector graphics into real-time video games, examining its potential to enhance visual quality, scalability, and interactivity within modern game environments.

1.4. Research Objectives

1. To assess the impact of vector graphics on visual quality, performance, and player experience in real-time games.
2. To design a framework that integrates vector graphics into real-time game environments, focusing on dynamic interactivity.
3. To identify and address the technical challenges and computational requirements needed for efficient real-time rendering of dynamic vector graphics.

1.5. Research Questions

1. What is the impact of using vector graphics on the visual quality, performance, and overall player experience in real-time games?
2. How can vector graphics be efficiently integrated into real-time game environments to enhance dynamic interactivity?
3. What are the technical challenges and computational requirements associated with rendering dynamic vector graphics in real-time?

1.6. Scope

The scope of this project invovles making a game that utilizes **Velyst**. The creation of the game will help ensure that **Velyst** is production ready by the end of this project.

Velyst	An open sourced, interactive Typst content creator using Vello and Bevy.
Lumina	A 2D top down fast paced objective based PvPvE game.

Table 1: Scope

Tasks to be executed:

1. Develop the open sourced **Velyst** crate.
    a. Develop an integrated compiler for Typst content in Bevy.
    b. Support hot-reloading of Typst content.
    c. Support interactivity between Bevy and Typst.
    d. Develop an easy-to-use framework for UI creation using Typst.

e. Support in-game textures using vector graphics content (e.g. using vector graphics texture for a crate in the game).
f. Write up a getting started documentation to make on-boarding easier for new developers.

2. Develop the **Lumina** game.
   a. Create a game design document (GDD) for the game.
   b. Integrate **Velyst** into the game.
   c. Develop all the required game mechanics for the game.
   d. Playtest the game and gather player feedbacks on the game.

Constraints

Constraint	Reason
Compatibility	Our project uses multiple cutting-edge and innovative technologies. This means that some of the technologies we depend on might be experimental or have yet to stabilize. This makes it difficult to ensure cross platform / device compatibility for the systems we built. For example, Vello requires compute shaders to render vector graphics, which means it can only runs on newer versions of browsers that supports WebGPU.
Limited Documentation	Because some of the technologies we use are experimental or less widely adopted, available documentation and community support may be limited. This slows down the development process as issues can be difficult to troubleshoot without established resources.
Limited Resource	The project is subject to limitations in terms of budget, personnel, and time. Allocating sufficient resources to develop, test, and refine the workflow for real-time vector graphics is crucial. Any constraints in these areas can impact the project's scope and delivery timeline.
Vector Graphics Constraint	It is crucial to recognize that while vector graphics offer numerous benefits, it is only suitable for representing precise shapes — such as fonts, logos, and icons. In contrast, complex images with intricate details, like a photograph of a cat are far better represented using bitmap formats.

Table 2: Constraints

What will be done in this project:

1. **Typst compiler**: A custom implementation of Typst compiler will be developed to fit the purpose of real-time Typst content rendering. This compiler should be able to re-compile Typst content on demand, allowing developers to view and reflect their saved changes immediately.

2. **Vector graphics renderer**: The **Velyst** crate will allow users to generate dynamic vector graphics content which is then rendered through the Vello renderer.

3. **Game demo prototype**: **Lumina** will be developed to showcase all of the above in a compact game format. Players will experience realistic and beautiful 2D lighting, as well as interactive vector graphics UI elements.

What will not be done in this project:

1. **Not creating an animation library**: An animation library involves preparing a huge variety of common animation effects. This takes a huge amount of time which does not fit the goal and scope of this project. To understand more about animation libraries, we strongly encourage you to look into the *Bevy MotionGfx* project.

2. **No visual editors**: It is possible to blend the Typst language with some form of visual editors, where the output would be a Typst function that takes in input parameters and produces a dynamic output content. However, our main focus for this project is to address the shortcomings of modern WYSIWYG editors. A feature as described would only be applicable after **Velyst** becomes a viable solution.

3. **Not a commercial game**: Our goal with **Lumina** is to create a game that demonstrates **Velyst**. It is not meant to be on par with a full on commercial game.

Open Source License

Velyst is dual-licensed under either:

- MIT License (http://opensource.org/licenses/MIT)
- Apache License, Version 2.0 (LICENSE-APACHE or http://www.apache.org/licenses/LICENSE-2.0)

This means you can select the license you prefer! This dual-licensing approach is the de-facto standard in the Rust ecosystem and there are very good reasons to include both. Head over to the **Velyst** GitHub repository (https://github.com/voxell-tech/velyst) to learn more!

1.7. Potential Benefit

Tangible Benefit

1. **Reduced asset size and load times**: The framework enables the use of vector graphics, which often require less storage space compared to traditional raster assets. This can result in smaller game asset sizes and improved load times, which can be quantitatively measured, providing concrete evidence of improved performance.
2. **Enhanced visual adaptability and resolution independence**: Vector graphics allow for high-quality visuals at any resolution, ensuring graphics remain crisp on various screen sizes and devices. This adaptability can be measured through resolution testing, demonstrating consistent visual quality across platforms.
3. **Improved development efficiency**: By streamlining the workflow, the framework reduces the time and resources needed for asset creation and rendering. Features like programmable content and hot-reloading can lead to faster development cycles and lower costs, measurable through development time tracking and cost analysis.

Intangible Benefit

1. **Increased creative freedom for developers**: The framework encourages developers to experiment with dynamic and interactive visuals, promoting creativity and innovation in game design. Although challenging to quantify, this freedom can result in more unique and engaging gameplay experiences, contributing to industry innovation.
2. **Enhanced player immersion and engagement**: By enabling smooth, responsive graphics, the framework enhances the overall visual appeal and interactive feel of games, potentially increasing player engagement and immersion. This contributes to player satisfaction, though it is difficult to measure directly.
3. **Long-term industry shift toward versatile graphics approaches**: The adoption of this framework may drive a gradual shift toward broader vector graphics usage in game development, promoting a more adaptable approach to graphics. Although this industry impact may not be immediately measurable, it contributes to evolving standards in visual and interactive design within games.

Target User

Velyst will particularly be targeted towards UI/UX developers, motion graphics creators, and vector graphics enthusiasts. Because **Velyst** is built on top of Rust and Bevy, the general users will come from the Rust and Bevy community.

As for **Lumina**, the target audience are gamers who loves fast paced multipalyer games like *Apex Legends* and *Astro Duel 2*. It will particularly appeal to gamers who love the mix of competitive PvP and PvE like *Destiny 2*'s Gambit game mode and *World War Z*.

1.8. Overview of the IR

This investigation report explores the establishment of a framework for integrating interactive and dynamic vector graphics in real-time video games. The project begins by providing a comprehensive introduction, outlining the context and background of the problem, where vector graphics, despite their scalability and precision, remain underutilized in modern game environments. This is followed by an articulation of the research aim, objectives, and scope, along with the potential benefits of the study. The introductory section concludes with an overview of the research design and project plan, setting the stage for a systematic investigation into the viability and advantages of vector graphics in real-time gaming contexts.

The literature review, presented in Chapter 2, delves into domain-specific research and technical explorations relevant to vector graphics and their application in game development. This section examines similar systems and related technologies, allowing for a comparison of existing approaches and identifying gaps within the current state of the art. Additionally, technical research focuses on the computational and algorithmic challenges posed by real-time vector graphics rendering, which informs the subsequent methodology and framework design. This chapter concludes with a summary that synthesizes the key insights from the literature, shaping the foundation for the methodological approach.

In Chapter 3, the methodology outlines the structured approach taken to develop and validate the proposed framework. Beginning with an introduction to the selected software development methodology (SDM) and its phases, this section describes the data gathering methods used to collect feedback from game developers and interactive application creators. This includes the design and execution of questionnaires and interviews, aimed at gathering insights on the practical demands and potential impacts of vector graphics within the industry. The methodology further explains the criteria for analysis, establishing a rigorous basis for interpreting the collected data.

Chapter 4 presents the findings from the questionnaire and interview analyses, examining trends, challenges, and developer preferences for integrating vector graphics in real-time environments. These results inform the conclusions drawn in the final section, where the potential of vector graphics in game development is assessed alongside the strengths and limitations of the proposed framework. The report closes with reflections on the study's contributions, limitations, and possible directions for future research, suggesting a pathway toward broader adoption and innovation in vector-based graphics in gaming.

1.9. Project Plan

Task Name	Duration (Day)	Start Date	End Date
Project Proposal Form	14	9th of September 2024	22nd of September 2024
Chapter 1: Introduction			
Introduction	2	23rd of September 2024	24th of September 2024
Problem Background	2	25th of September 2024	26th of September 2024
Project Aim	1	27th of September 2024	27th of September 2024
Objectives	1	28th of September 2024	28th of September 2024
Scope	1	29th of September 2024	29th of September 2024
Potential Benefits	1	30th of September 2024	30th of September 2024
Overview of IR	1	1st of October 2024	1st of October 2024
Project Plan	1	2nd of October 2024	2nd of October 2024
Chapter 2: Literature Review			
Introduction to LR	1	3rd of October 2024	3rd of October 2024
Domain Research	8	4th of October 2024	11th of October 2024
Similar System	3	12th of October 2024	14th of October 2024
Technical Research	5	15th of October 2024	19th of October 2024
Summary	1	20th of October 2024	20th of October 2024
Chapter 3: Methodology			
Introduction of SDM	1	21st of October 2024	21st of October 2024
Methodology Choice and Justification	2	22nd of October 2024	23rd of October 2024
Data Gathering Design	6	24th of October 2024	29th of October 2024
Questionnaire Design	3	30th of October 2024	1st of November 2024
Interview Design	4	2nd of November 2024	5th of November 2024
Questionnaire Analysis	3	6th of November 2024	8th of November 2024
Interview Analysis	3	9th of November 2024	11th of November 2024
Summary	1	12th of November 2024	12th of November 2024
Chapter 4: Conlusion and Reflections			
Conclusion	2	13th of November 2024	14th of November 2024

Table 3: Project Plan

CHAPTER 2: Literature Review

2.1. Domain Research

Introduction

Vector graphics are increasingly valuable in game development due to their scalability and precision, yet their integration into real-time game engines involves distinct technical challenges. Unlike raster images, which degrade when resized, vector graphics are resolution-independent, making them ideal for various screen sizes and resolutions. This advantage, however, comes at the cost of more complex rendering requirements, as vector graphics must undergo a rasterization phase to translate their shapes into pixels for display. The demands of real-time rendering make this process particularly intricate, as maintaining both high visual quality and performance can be challenging.

To address the performance demands of rasterization in game engines, various rendering techniques have been developed to streamline this process. Efficient rasterization methods help achieve the fast rendering speeds required for interactive applications, while ensuring that image quality remains high. Techniques focused on converting vector shapes into compatible formats for real-time processing have allowed developers to create more responsive and visually detailed graphics without a loss in performance. These approaches improve frame rates and visual fidelity, making vector graphics a practical choice in game settings where performance is essential.

In addition to advances in rasterization, the tools used for creating and managing graphical content have also evolved. Modern UI frameworks play a foundational role in constructing interactive experiences, providing developers with ways to structure and style visual elements effectively. The shift towards markup languages and programmable typesetting has made it easier for teams to collaborate and iterate on designs, fostering a more flexible workflow. This evolution has supported the creation of more dynamic visual components that can be reused and customized, enhancing user engagement and experience.

This review provides a foundation for analyzing the technical aspects of vector graphic rasterization and content creation tools, highlighting how these technologies can drive efficiency, scalability, and visual impact in game design and interactive media.

Rasterization of vector graphics

Vector graphics are often used in situations where scalability and precision are essential. This property comes with a cost. As mentioned in the previous chapter, rendering vector graphics in today's era requires a rasterization phase. Solving for this phase is non-trivial as it is often required to compute a partial differential equation (PDE) (Tian & Günther, 2022).

One method for rasterizing vector graphics is scaline rendering. Scanline rendering is the process of shooting rays from one side of the screen to the other while coloring pixels in between based on collision checkings with paths in between. A GPU based scanline rasterization method is proposed by parallelizing over boundary fragments while bulk processing non-boundary fragments as horizontal spans (Li et al., 2016). This method allows fully animated vector graphics to be rendered in interactive frame rates.

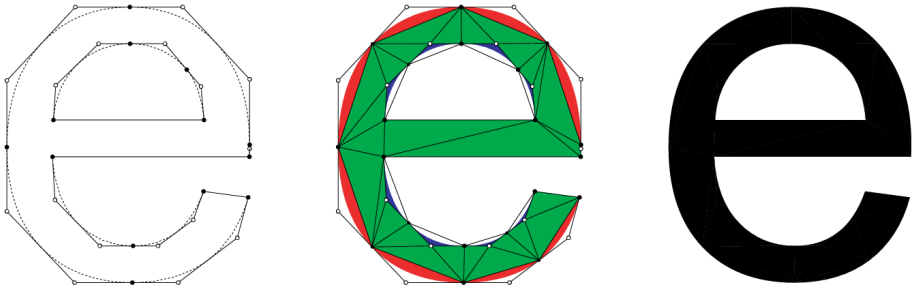

Figure 3: Tesselation with shader evaluated curves (Loop & Blinn, 2005)

Apart from scanline rasterization, tesselation method can also be used to convert vector graphics into triangles and then pushed to the GPU for hardware accelerated rasterization. Loop & Blinn (2005) further improved this method by removing the need of approximating curve segments into lines. Instead, each curve segments is evalulated in a fragment shader which can be calculated on the GPU, as shown in Figure 3. This allows for extreme zoom levels without sacrificing qualities.

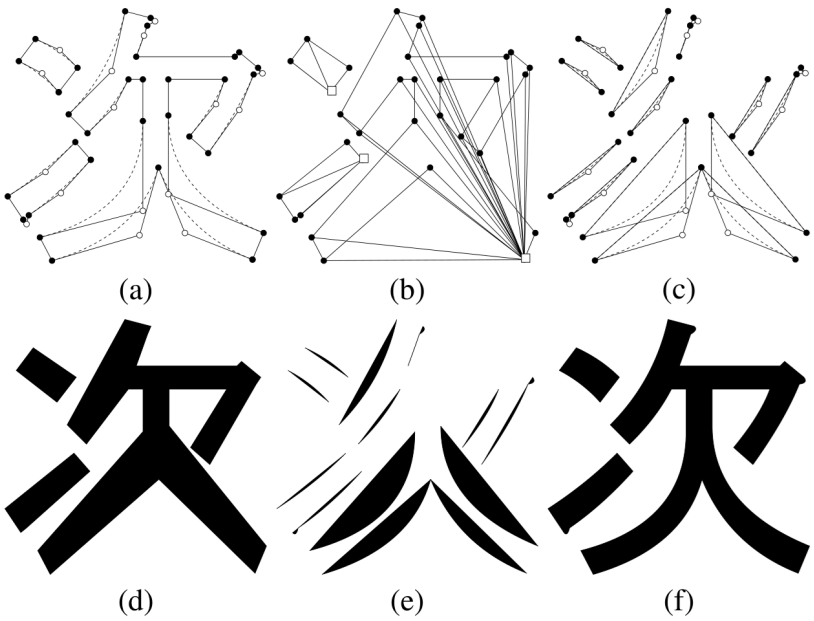

Figure 4: Triangle fans (Kokojima et al., 2006)

Re-tesselation of vector graphics can be computationally expensive, especially when it is inherently a serial algorithm that often needs to be solved on the CPU. Kokojima et al. (2006) combines the work of Loop & Blinn (2005) with the usage of GPU's stencil buffer by using triangle fans to skip the tesselation process as shown in Figure 4. This approch, however, does not extend to cubic Bézier segments as they might not be convex. Rueda et al. (2008) addressed this issue by implementing a fragment shader that evaluates the implicit equation of the Bézier curve to discard the pixels that fall outside it. The two-step "Stencil then Cover" (StC) method builds upon all of these work and unified path rendering with OpenGL's shading pipeline — `NV_path_rendering` (Kilgard & Bolz, 2012). This library was further improved upon by adding support for transparency groups, patterns, gradients, more color spaces, etc. (Batra et al., 2015). It was eventually integrated into Adobe Illustrator.

Immediate mode and retained mode

Beneath all graphical interfaces lies the underlying code that structures and renders the visual elements. The two approaches towards creating user interface frameworks are immediate-mode graphical user interface (IMGUI) and retained-mode graphical user interface (RMGUI). Some open sourced IMGUI frameworks includes Dear ImGui and Egui, while open sourced RMGUI frameworks includes Xilem and Qt. Although powerful, these UI frameworks strongly relies on hardcoded programming. In retained-mode, the application attempts to retain its previous state as much as possible and only perform changes when necessary. In contrast, the immediate-mode application reconstruct its frame in every update with no states stored in between frames (Satran & Radich, 2019). This makes retained-mode useful for applications that does not require high dynamic changes or devices that require low power consumption and vice versa (Glazkov, 2021). In the article by Satran & Radich (2019), the authors also stated that while retained-mode APIs are generally easier to use, they often come with higher memory demands and offer less flexibility than their immediate-mode counterparts.

```
ui.heading("My egui Application");
ui.horizontal(|ui| {
  ui.label("Your name: ");
  ui.text_edit_singleline(&mut name);
});
ui.add(egui::Slider::new(&mut age, 0..=120).text("age"));
if ui.button("Increment").clicked() {
  age += 1;
}
ui.label(format!("Hello '{name}', age {age}"));
ui.image(egui::include_image!("ferris.png"));
```

Code 1: Egui code example (imperative)

```
struct AppData {
  count: u32,
}

fn count_button(count: u32) -> impl View<u32, (), Element = impl Widget> {
  Button::new(format!("count: {}", count), |data| *data += 1)
}

fn app_logic(data: &mut AppData) -> impl View<AppData, (), Element = impl Widget> {
  Adapt::new(|data: &mut AppData, thunk| thunk.call(&mut data.count),
    count_button(data.count))
}
```

Code 2: Xilem code example (declarative)

In practice, IMGUI is imperative and RMGUI is declarative as shown in Code 1 and Code 2. In imperative programming, developers define each step needed for a program to reach the desired state, specifying precisely how to display UI components. Declarative programming, on the other hand, allows developers to describe what should be displayed without detailing how to achieve it. Declarative frameworks often rely on underlying imperative code to translate high-level commands into step-by-step instructions for execution (Ricardo, 2020). While imperative programming allows programmers to have complete control over system resources, it would eventually lead to higher complexity as projects scale, increasing the risk of bugs. In contrast, declarative programming minimizes state mutability by favoring more sophisticated constructs like pipelines and work graphs, which leads to better scalability to larger projects. However, it is important to understand the performance overhead and higher learning curve that comes with declarative programming (Kashivskyy, 2024).

Markup languages for graphical interface content creation

Enter the web technologies. Modern browsers typically render UI elements using markup languages like Hyper Text Markup Language (HTML) and Scalable Vector Graphics (SVG) for structuring the content and style-sheets like Cascading Style Sheets (CSS) for styling them as shown in Code 3 and Code 4. The use of markup structures allows developers to fully separate their UI layout from the codebase, simplifying the identification and management of UI components. With style sheets, developers can create, share, and reuse templates, enhancing consistency and streamlining the design process throughout the application.

```html
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>This is a heading</h1>
  <p>This is a paragraph.</p>
</body>
</html>
```

Code 3: HTML example

```css
/* styles.css */
body {
  background-color: powderblue;
}
h1 {
  color: blue;
}
p {
  color: red;
}
```

Code 4: CSS example

Markup languages also give rise to many WYSIWYG editors. These editors let users perform drag and drop actions to layout UI for quick prototyping as each components can now be represented using only markup syntax (no code required). An example of this framework's application within a game engine is the Unity UI Toolkit, which employs it to power its UI editor (*Comparison of UI Systems in Unity*, n.d.; Krogh-Jacobsen, 2023).

A major limitation of simple markup languages like HTML is that structural changes can only be achieved through code (Sigireddyviswesh, 2023). These features are often delegated to the programmer which can lead to unintended miscommunications. For example, if you want a form to disappear after button press, you would need to alter the HTML via code. Typst offers an alternative towards this problem by introducing programming capabilities into markup (Mädje, 2022).

```
\documentclass{article}
\begin{document}
First document. This is a simple example, with no
extra parameters or packages included.
\end{document}
```

Code 5: LaTeX example

Typst is a competitor of LaTeX (Code 5), designed to simplify the typesetting process with a modern and intuitive approach. Unlike its predecessors, Typst can directly embed logic (Code 6). Using the previous example, developers would only need to pass in a boolean value and Typst will automatically exclude the form from being in the layout at all. In the Typst ecosystem, developers gain enhanced flexibility by sharing their work as packages. Unlike templates, Typst packages support complex scripting, offering greater adaptability and expanded functionality.

```
#let values = (1, 2, 3, 4)
#values.pop() \
#values.len() \

#("a, b, c"
    .split(", ")
    .join[ --- ])

#"abc".len() is the same as
#str.len("abc")
```

Code 6: Typst example

Summary of key findings

The exploration of rasterization techniques for vector graphics underscores the complexity and potential of rendering mathematically defined shapes in real-time applications. Vector graphics are valued for their scalability and precision, but rendering them on modern displays requires the conversion into pixel-based formats through processes like scanline rendering and tessellation. These techniques, while effective in achieving interactive frame rates, often come with significant computational costs. Innovations such as GPU-based scanline rasterization, which parallelizes boundary fragments, and advanced methods like Stencil then Cover (StC), have enabled more efficient rendering. Additionally, the development of fragment shaders for evaluating curves at extreme zoom levels enhances the ability to handle vector graphics without sacrificing quality, illustrating the evolving landscape of vector graphics rendering in computational graphics.

In the domain of graphical interface content creation, the use of markup languages like HTML, SVG, and CSS has revolutionized the way user interfaces are structured and styled. These technologies allow for the separation of content from design, simplifying the creation, management, and customization of user interfaces. However, challenges remain in introducing dynamic interactivity and structural changes without relying on code, a limitation addressed by systems like Typst, which introduces programming capabilities directly into the markup. Typst's approach enables greater flexibility by supporting complex scripting and packages, offering a significant improvement over traditional methods. This shift towards more programmable, reusable, and adaptable UI frameworks opens new possibilities for creating responsive and user-driven interfaces in modern applications.

Altogether, advancements in both rasterization and content creation tools continue to drive the development of modern game environments. As these technologies progress, game engines can offer more sophisticated and seamless graphical experiences, enabling developers to balance aesthetic quality with technical efficiency and bringing richer visual and interactive elements into the gaming space.

2.2. Similar Systems/Works

Unity UI Toolkit

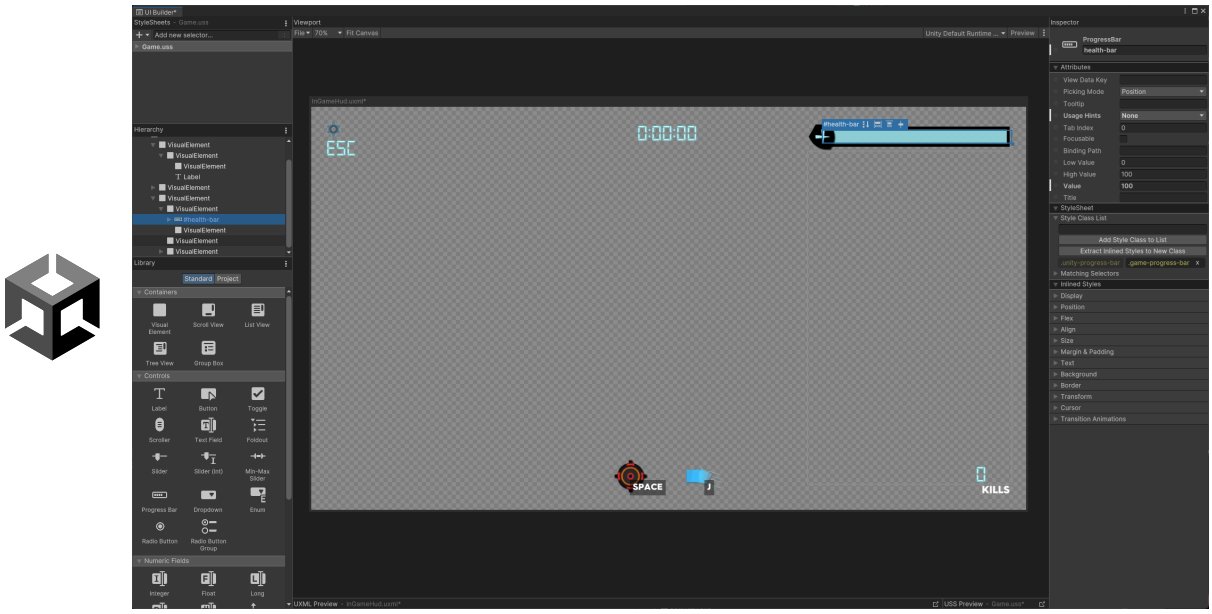

Figure 5: Unity UI Toolkit Screenshot

The Unity UI Toolkit shown in Figure 5 is a retained mode WYSIWYG editor, developed for the Unity3D Engine for editor GUI and runtime UI creation (Kok, 2021). It uses a markup language called Unity Extensible Markup Language (UXML) to define the UI structure (*Introduction to UXML*, n.d.). UXML is highly inspired by languages like HTML, Extensible Application Markup Language (XAML), and Extensible Markup Language (XML). The UXML is tailored in such a way that is efficient to work with the Unity game engine.

```xml
<?xml version="1.0" encoding="utf-8"?>
<ui:UXML ...>
  <Style src="<path-to-file>/styles.uss" />
  <ui:VisualElement name="root" />
</ui:UXML>
```

Code 7: UXML example

```css
/* styles.uss */
#root {
  width: 200px;
  height: 200px;
  background-color: red;
}
```

Code 8: USS example

The Unity3D team also developed a modified CSS language called Unity Style Sheet (USS) and Theme Style Sheet (TSS) to style UIs that are structured via UXML (*Add Styles to UXML*, n.d.), (*Theme Style Sheet (TSS)*, n.d.). TSS is just a slight variant of USS in that it is solely responsible for the overall theme that the application falls back to when there is no USS or inline styling. An example of UXML and USS is shown in Code 7 and Code 8. Users can create custom visual

elements in C# and use it in UXML. The USS can be reused across multiple UXML and can even be imported into other USS files. This prevents duplicate work and helps keep the user interface consistent across the application.

The Unity UI Toolkit also improves collaboration between artists and developers. With UI Toolkit, artists focus on UXML and USS files, handling design elements like colors and materials, while developers add behaviors and interactions exclusively through code, without modifying the design files. This separation of logic and style streamlines merging and makes style adjustments more efficient. For example, changing project-wide fonts only requires editing Panel Settings instead of individual assets. UI Builder further assists by providing a visual interface that allows artists and designers to create and edit UI without coding, enhancing teamwork and organization for larger projects.

Egui

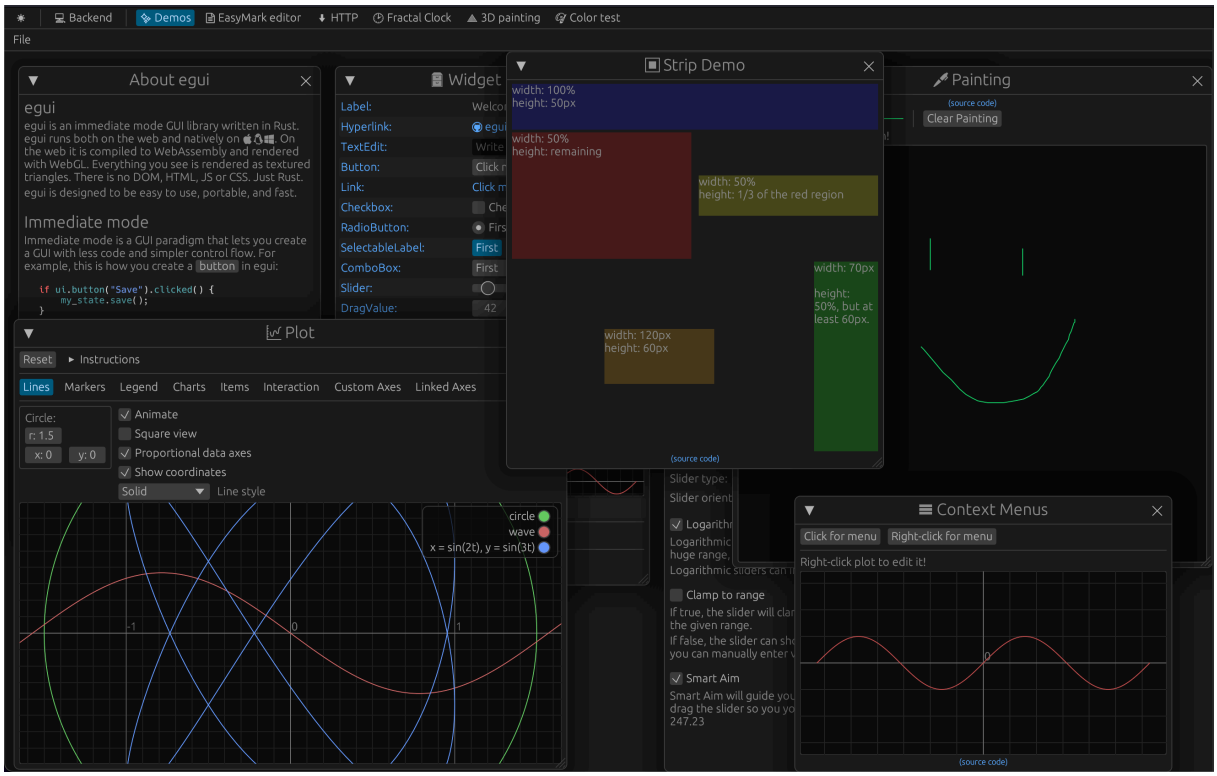

Figure 6: Egui Demo (Siwiec, 2023)

Egui is an easy-to-use immediate mode GUI in Rust that runs on both web and native. It is extremely lightweight and offers numerous graphics tools for the creation of GUI with minimal effort as shown in Figure 6. Some examples include graph plotting, text input, sliders, scroll view, painting, etc. Egui is a cross-platform UI library (Hana, 2023). Hana (2023) noted in the article that Egui is built on top of WebGPU technology. This enables it to run on both native platforms like Windows, Linux, and Mac, as well as in the browser, in the form of Web Assembly (WASM). Users can also integrate it into custom or existing game engines, custom native applications, web applications, etc. It is also highly inspired by Dear ImGui from the C++ world (Rodney, 2024).

Moreover, Egui supports extensibility by allowing users to create and integrate custom components, providing additional flexibility and customization options. When it comes to drawing capabilities, Egui offers built-in support for rendering vector graphics, including complex elements like Bézier curves, basic shapes, and geometric primitives, as noted by Hana (2023). This feature set enables developers to design visually engaging and highly interactive interfaces, suitable for both simple and complex applications, while maintaining Egui's lightweight footprint and ease of use.

Strengths and Weaknesses

The Unity UI Toolkit and Egui are two distinctive GUI frameworks with specific strengths and weaknesses, tailored to different development environments. The Unity UI Toolkit is a retained mode, WYSIWYG editor designed for Unity3D, whereas Egui is an immediate mode GUI built in Rust, emphasizing simplicity and flexibility across platforms.

Unity UI Toolkit

One of the primary strengths of the Unity UI Toolkit lies in its structured, markup-based approach, which leverages the UXML and USS languages. UXML's structure-based syntax is inspired by web languages like HTML and XAML, making it relatively accessible for developers familiar with web-based frameworks. This setup allows for clear separation between structure and styling, which is further enhanced by the USS and TSS. Such separation enables efficient style reuse and consistent design application across multiple UI components, avoiding redundant work and maintaining visual coherence. Moreover, *UI Toolkit*'s integration with Unity's *UI Builder* makes it a practical choice for artist-developer collaboration; it allows designers to construct interfaces visually, without needing to edit the underlying code. This, along with its retained mode nature, means developers can focus more on event-driven logic and less on rendering details, which can simplify UI maintenance and readability.

However, the Unity UI Toolkit has notable limitations. The framework's retained mode design, though powerful, can be more memory-intensive than immediate mode approaches, impacting performance in highly dynamic or resource-constrained applications. Additionally, since the UI Toolkit is specialized for Unity, it lacks portability outside the Unity ecosystem, restricting its utility for developers working across multiple engines or platforms. Although it offers an accessible toolset for users within Unity, it may require more complex workflows to integrate into non-Unity environments. Customization options are available through C# for creating visual elements, yet the reliance on Unity-specific markup and styles can create a learning curve for developers unfamiliar with the Unity ecosystem.

Egui

Egui, in contrast, is an immediate mode GUI that stands out for its simplicity, light memory footprint, and versatility. Written in Rust, Egui is designed to be cross-platform, operating seamlessly on both native environments (Windows, Linux, and macOS) and the web, thanks to its WebAssembly (WASM) compatibility. The immediacy of Egui's approach allows it to be incredibly lightweight; every frame is actively rendered, which can be ideal for

applications requiring high responsiveness or frequent UI updates. Its Rust-based design further enhances portability, enabling easy integration with various custom engines or standalone applications, making it suitable for projects not tied to a specific ecosystem. Additionally, Egui supports vector graphic rendering, including shapes and Bézier curves, which, combined with customizable components, provides significant flexibility for developers creating interactive and visually dynamic UI.

However, Egui's immediate mode nature can also present some challenges. Immediate mode UIs often involve redrawing the entire UI each frame, which can become computationally intensive in complex interfaces, potentially leading to higher CPU usage. While Egui is excellent for lightweight applications, it may not scale as efficiently for large-scale, interactive applications with many UI elements. Furthermore, Egui lacks a visual editor, which can be a drawback for teams with designers who prefer WYSIWYG tools. This absence means that UI creation and design adjustments must be handled directly in code, potentially complicating collaboration between designers and developers.

Conclusion

In summary, both frameworks offer valuable advantages within their respective domains. Unity UI Toolkit excels within the Unity ecosystem, providing structured, easily manageable UI elements and a design-friendly interface that streamlines collaboration. However, it is less portable and may consume more memory. Meanwhile, Egui's lightweight, immediate mode design and cross-platform flexibility make it highly suitable for Rust applications and custom engines, albeit with potential scalability and collaboration limitations for larger projects. Both frameworks cater to different development needs, and choosing between them depends on the platform, performance requirements, and team workflow preferences.

2.3. Technical Research

Bevy - Game Engine

Figure 7: Bevy Logo

According to the founder, Anderson (2020), Bevy is a refreshingly simple data-driven game engine and app framework built in Rust that is free and open-sourced. It is designed to be modular with its robust plug-in system, simple to use with its Entity Component System (ECS), and fast with its multi-core utilization and parallel by default design. In the article, the founder claim that Bevy's ECS is the most ergonomic ECS in existence (Anderson, 2020). As shown in Code 9, users can create a fully working Bevy application with just a few lines of code:

```rust
use bevy::prelude::*;

#[derive(Component)]
struct Position(f32);

fn setup(mut commands: Commands) {
  commands.spawn(Position(0.0));
}

fn movement(mut q_positions: Query<&mut Position>, time: Res<Time>) {
  for mut pos in q_positions {
    pos.0 += 1.0 * time.delta_seconds();
  }
}

fn main() {
  App::build()
    .add_plugins(DefaultPlugins)
    .add_systems(Startup, setup)
    .add_system(Update, movement)
    .run();
}
```

Code 9: Bevy example

Bevy is also cross platform. It runs on Windows, MacOS, Linux, iOS, Android, and the web. Although Bevy is relatively new compared to more established game engines like Unity or Unreal Engine — having only been around for four years as of 2024 (Anderson, 2024) — it has quickly gained attention for its modern, modular approach to game development. As a result, there has already been many volunteer plug-ins being developed for the game engine.

Some notable plug-ins that will be used of this project are:

Plug-in	**Description**
Velyst	The plug-in that we are going to develop in this project.
Bevy Vello	For renderingn vector graphics using Vello.
Lightyear	For multiplayer and networking.
Bevy Enoki	For particle effects.
Leafwing Input Manager	For managing different input devices (e.g. keyboards, mouse, controllers, etc.)
Blenvy	For level design.
Avian2d	For rigid-body physics and collision detections.

Table 4: Notable Bevy plug-ins in this project

And many more!

Vello - Compute-centric Vector Graphics Renderer

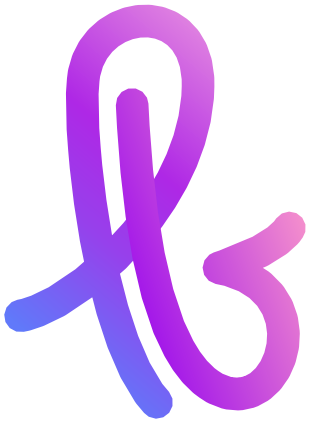

Figure 8: Linebender Logo

Vello is a compute-centric vector graphics renderer developed by the Linebender organization. The Linebender organization is a group of people who share an interest in 2D graphics and UI design (*Linebender*, 2024). Some other projects that they developed include Xilem, Masonry, Vello, Kurbo, Peniko, etc (*Linebender*, 2024).

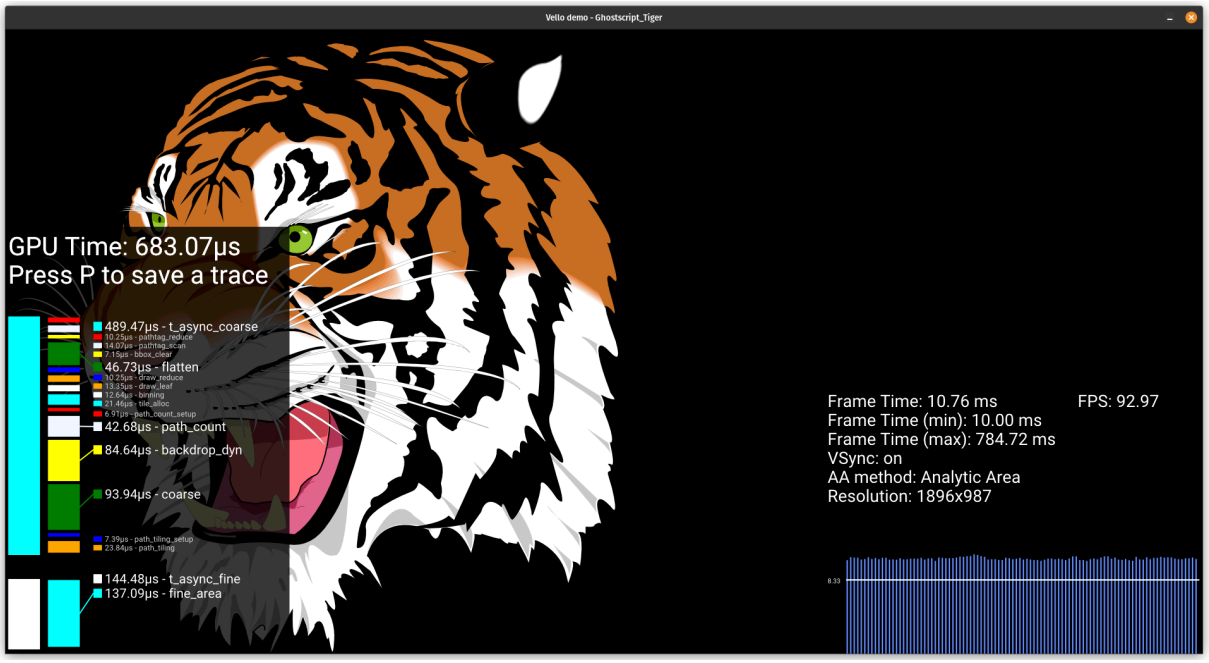

Figure 9: Vello demo

Unlike traditional vector graphics renderer, Vello strives to off-load all of the rendering steps to the GPU via compute shaders. At the moment of writing this article, Levien (2020) stated that there are plans for Vello to support retained mode in the future, but for now, its mostly an immediate mode API. This is actually only possible because of the performance gain that Vello obtain via parallel computing on the GPU.

Figure 10: Wgpu logo

Part of what makes Vello so appealing is its cross platform capability. Unlike many other research projects that uses CUDA as their GPU compute platform, Vello achieves portable GPU compute by utilizing the Wgpu library (Levien, 2020; *Roadmap for 2023*, n.d.). Wgpu runs natively on Vulkan, Metal, DirectX 12, and OpenGL ES; and browsers via WebAssembly (WASM) on WebGPU and WebGL2 (*Wgpu*, n.d.).

Typst

Figure 11: Typst logo

Typst has been mentioned multiple times in this paper. It is used as the language for creating graphical content in **Velyst**. Typst is a programmable markup language for sophisticated typesetting (Mädje, 2022). It supports programming concepts like conditionals, while and for loops, closures, functions, and many more. These concepts are directly embedded into the Typst language without any third party scripting language (Mädje, 2022).

By integrating Typst within a Rust environment through Velyst, this project combines Typst's layouting and scripting capabilities with Bevy's real-time rendering. This enables games to leverage Typst's typesetting strengths for on-the-fly generation of UI elements, text, and other graphical assets. The result is an efficient workflow where content in Typst can be manipulated and rendered dynamically within the game engine, enhancing both the visual and interactive elements of the application. This approach bridges the gap between content creation and real-time execution, offering a powerful tool for developers who seek sophisticated text and graphics rendering in games.

CHAPTER 3: Methodology

3.1. System Development Methodology

Introduction

Selecting a suitable development methodology is crucial for ensuring an efficient and adaptable workflow in any project. Existing methodologies each come with unique strengths and limitations, often tailored to specific types of projects. Among the most popular methodologies are Waterfall and Agile.

Waterfall

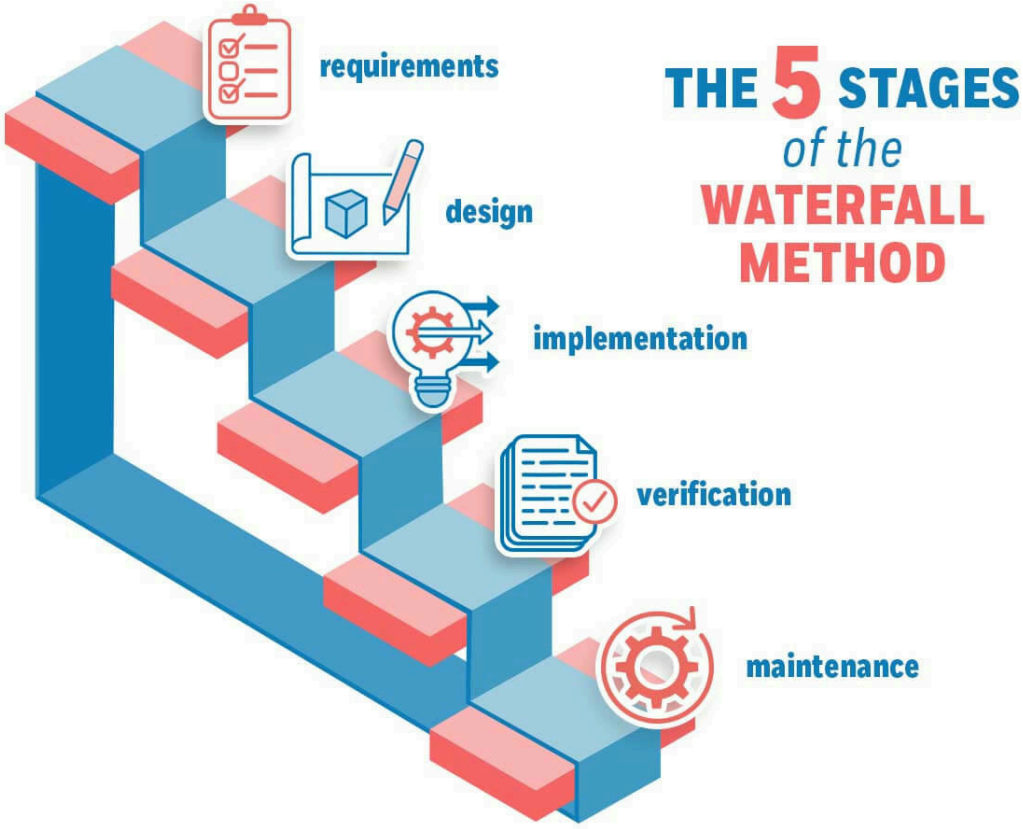

Figure 12: Waterfall methodology (Jayathilaka, 2020)

Waterfall is a linear and sequential model ideal for projects with well-defined requirements and minimal expected changes. Its strength lies in its clear structure and phase-based approach, making it predictable. However, its rigidity can be a disadvantage in projects where requirements may evolve over time, as it lacks flexibility to accommodate ongoing changes.

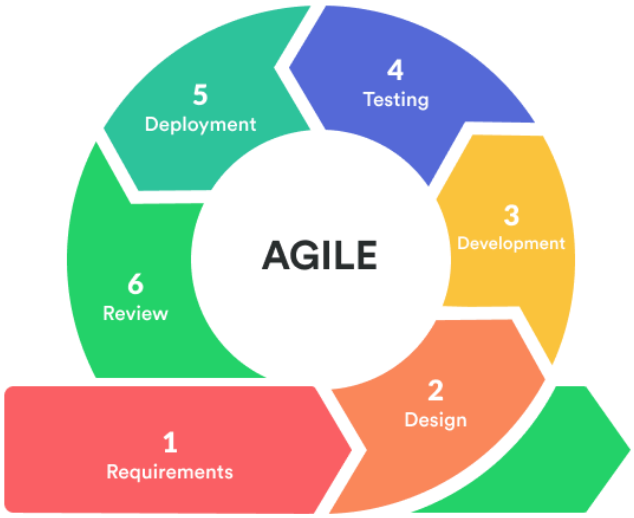

Figure 13: Agile methodology (Waseem & Hart, 2023)

Agile

Agile, on the other hand, offers a highly flexible and iterative approach that allows for continuous feedback and adjustments throughout the development process. This model excels in projects where requirements are expected to change, as it accommodates user input and evolving project goals. However, Agile can sometimes struggle in projects requiring a fixed scope and budget, as its adaptability can lead to scope creep.

Below is a simple comparison of these methodologies:

Methodology	Advantages	Disadvantages
Waterfall	• Clear structure and phase-based, making project planning predictable. • Works well for projects with well-defined requirements and goals. • Clear documentation and deliverables at each stage.	• Inflexible to changing requirements, making it hard to adapt once development starts. • Lack of early testing and feedback can lead to large-scale reworks if requirements change. • Difficult to address issues mid-project as each phase is sequential.
Agile	• Highly adaptable to change, allowing for user feedback at every stage.	• Potential for scope creep, as continuous adaptation can lead to expanding project scope.

Methodology	Advantages	Disadvantages
	• Reduces risk through continuous testing and regular check-ins. • Ideal for fast-paced environments where requirements evolve frequently. • Promotes collaboration, enhancing team responsiveness and communication.	• Requires active stakeholder engagement, which may be time-consuming. • Lacks predictability, which may make budgeting and timeline estimation challenging.

Methodology Choice and Justification

For this project, Agile methodology is the preferred choice due to its iterative and user-centered design, which allows the team to remain adaptable in response to feedback and evolving project requirements. Given the nature of the project, where user feedback and rapid iterations are essential, Agile supports a workflow that promotes ongoing testing, evaluation, and refinement at each stage.

The workflow in Agile will follow a sprint-based cycle, broken down into the following sections:

1. **Planning**
   Each sprint begins with a planning phase where tasks are identified and assigned based on the project goals for that sprint. The team outlines the deliverables, prioritizes tasks, and ensures each team member is clear on their responsibilities. Planning ensures that each sprint has a focused objective, with clear milestones to meet.

2. **Design and Development**
   During the design and development phase, the team works on implementing features and functionalities outlined in the sprint plan. Development occurs in small, manageable iterations, with regular check-ins and adjustments to ensure alignment with the sprint's objectives. This phase enables quick adaptability and minimizes the risks of large-scale reworks.

3. **Testing and Feedback**
   At the end of each sprint, the team conducts testing to ensure that newly developed features are functional and aligned with project requirements. Feedback is gathered from relevant

stakeholders, and any issues or suggested changes are documented. This feedback loop is critical in Agile, as it ensures that adjustments can be made continuously.

4. **Review and Retrospective**

   After testing, the team performs a sprint review, evaluating both the successes and challenges encountered. The retrospective phase allows the team to reflect on the process, identify areas for improvement, and incorporate lessons learned into the next sprint's planning phase. This continuous improvement approach aligns with Agile's focus on flexibility and adaptability.

By following Agile, the project benefits from a dynamic and responsive workflow that can easily adapt to any changing needs, helping to ensure timely delivery and a user-centered approach. The flexibility offered by Agile makes it the ideal choice, enabling iterative progress and a focus on evolving project goals and user expectations.

3.2. Data Gathering Design

In the research design, data gathering will be conducted using two complementary methods: a questionnaire and an interview. This dual-method approach aims to balance broad, generalized data from a larger group with in-depth insights from a select group of professionals.

The questionnaire will target a broad audience, serving as the primary method to collect quantitative data on user opinions, experiences, and preferences related to the research topic. By reaching a larger pool of respondents, the questionnaire enables the gathering of statistically relevant insights and general trends across a wide demographic. Structured in a way that includes multiple-choice questions, Likert scales, and a few open-ended items, the questionnaire will facilitate an efficient analysis while also allowing for some qualitative feedback (DEWI, 2021). This design helps to capture both objective data and individual nuances within responses, which can be further analyzed to identify general patterns and correlations. Additionally, the anonymity provided in the questionnaire encourages honest and diverse opinions from a range of backgrounds, helping to avoid biases that might be present in more direct methods of data collection.

Complementing the questionnaire, the interview component will provide a more nuanced, qualitative perspective by engaging directly with professionals who possess specialized knowledge or expertise in the area of study. The interviews are designed to gather deeper insights that would not typically be captured in a standardized questionnaire. Conducted in a semi-structured format, the interview will allow for flexibility, enabling the researcher to probe further into responses based on the interviewee's expertise. This format encourages participants to share their informed opinions, elaborate on complex ideas, and provide practical examples, thus enriching the data with professional and real-world insights (DEWI, 2021). In addition, these expert insights will offer a valuable context for interpreting the trends identified in the questionnaire data, ensuring that the analysis incorporates both user-focused feedback and professional validation.

Together, these two methods will provide a balanced and holistic dataset, with the questionnaire delivering broad quantitative data and the interviews offering rich qualitative insights. This mixed-method approach not only enhances the credibility and depth of the research findings but also supports a comprehensive understanding of the topic from both the end-user and expert perspectives. Ultimately, combining these data sources will enable the research to deliver well-rounded conclusions and recommendations that are both broadly applicable and grounded in professional expertise.

Questionnaire Design

Section 1: Introduction and Basic Information

This section gathers general demographics information from the respondents such as name, age, and gender.

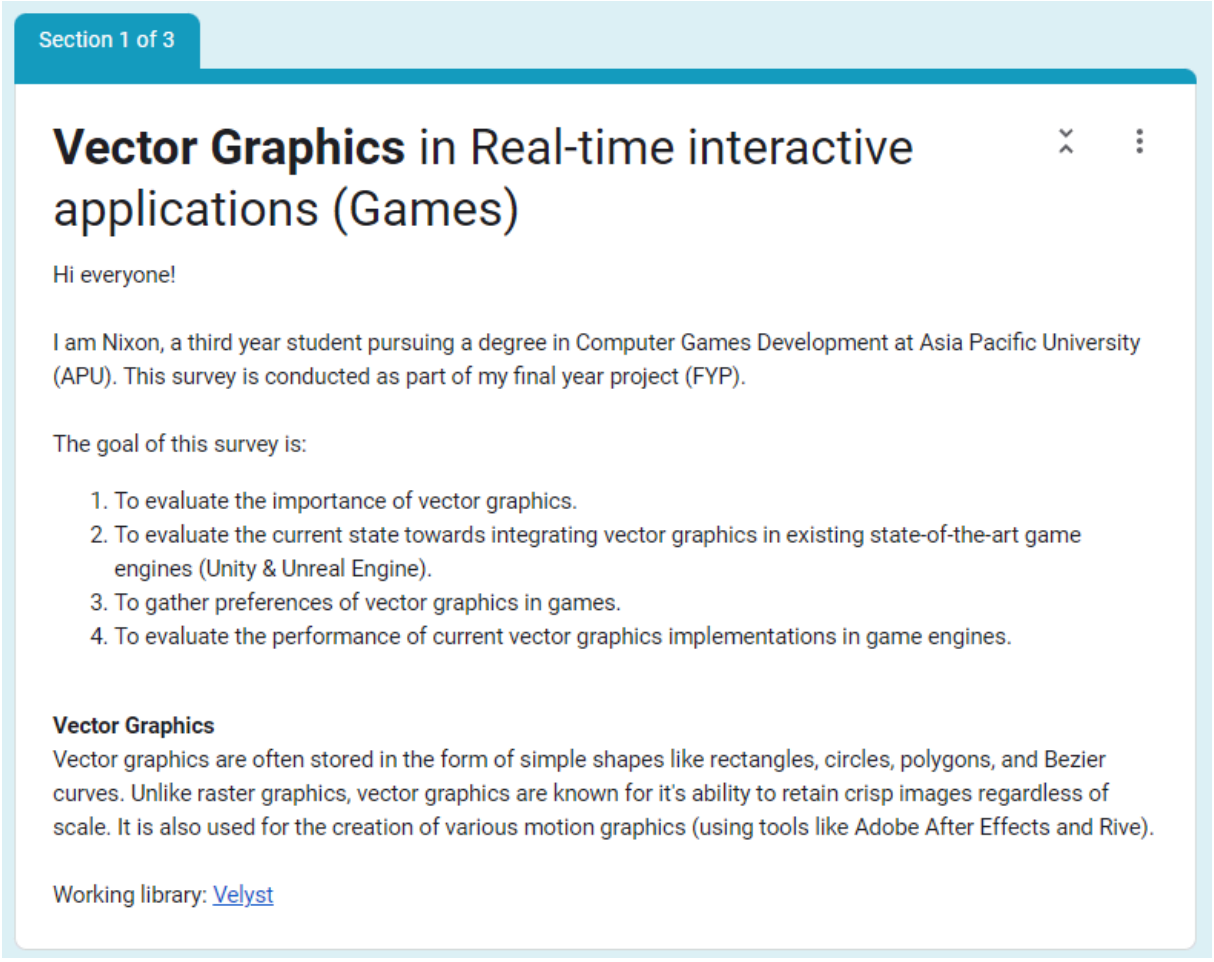

Figure 14: Screenshot of questionnaire description

No.	Question	Answer	Justification
1	Name	Short text answer	Prevent duplicate data as well contact purposes in case if any clarifications are needed.
2	Email	Short text answer	For contact purposes in case if any clarifications are needed.

No.	Question	Answer	Justification
3	Gender	Multiple Choice • Male • Female • Prefer not to say • Other:	Identify preference differences between different genders.
4	Age	Multiple Choice • < 18 • 18 - 24 • 25 - 30 • > 30	Identify preference differences across various age groups.

Table 5: Section 1 questions, answers, and justifications

Section 2: Vector graphics in games

This section provides an overview of the current state of vector graphics in game engines, focusing on feature preferences, implementations, and challenges. The questions are designed with the assumption that the respondent has a basic understanding of vector graphics. This is due to the fact that vector graphics implementations in today's game engines are still relatively limited, which means that many respondents may only be familiar with the basic concepts or have limited experience with more advanced features. Therefore, the questions aim to capture general insights rather than technical expertise, helping to gauge the broader perceptions and experiences of users with vector graphics in real-time game development.

No.	Question	Answer	Justification	Relevant Research Objective
1	How important is the use of vector graphics for creating crisp and scalable visuals in games?	Linear scale from 1-5 1 = Not important 5 = Very important	To assess the perceived significance of vector graphics for enhancing visual quality and scalability in games, which relates to their impact on player experience.	Research objective 1
2	What assets would you prefer vector graphics over raster graphics.	Checkboxes • Icons • User Interface • Particle Effects • In Game Textures • Other:	Understanding which game assets are favored for vector graphics helps identify areas where vector graphics improve visual quality and performance.	Research objective 1 and 2

No.	Question	Answer	Justification	Relevant Research Objective
3	Are you familiar with vector graphics concepts, such as scalability and the use of shapes like Bezier curves and polygons?	Multiple Choice • Yes • No	Gauges familiarity with vector graphics concepts, which informs on potential challenges in implementation.	Research Objective 3
4	How are the performance of vector graphics in said game engines?	Multiple Choice Grid Rows: • Unity • Unreal Engine Columns: • Very bad • Bad • Moderate • Well • Very well	Assess impact of vector graphics on game engine performance and identify performant implementations.	Research Objective 3
5	How challenging do you find it to implement vector graphics in said game engines?	Multiple Choice Grid Rows: • Unity • Unreal Engine Columns: • Very bad • Bad • Moderate • Well • Very well	To understand technical barriers and identifying user friendly design patterns.	Research Objective 2 and 3

Table 6: Section 2 questions, answers, and justifications

Interview Design

Interviews provide qualitative data that yields valuable insights into the research objectives of this project. The selected interviewees for this section are experienced developers who have been working professionally in the industry, bringing real-world expertise to the discussion. This targeted selection allows the interviews to be more in-depth and technical, yielding concrete evidence and examples that can enhance the understanding of practical challenges and benefits associated with vector graphics in real-time game development. These insights will help inform the design of an effective framework and guide future advancements in integrating vector graphics into game engines.

No.	Question	Justification	Relevant Research Objective
1	Could you briefly introduce yourself and your background in UI/UX or game development?	Establish context for their experience level and focus areas.	-
2	How would you describe your general workflow when designing and implementing vector graphics in applications or games?	Gather insights into workflows to identify typical processes in implementing vector graphics or similar elements.	Research Objective 2
3	What are some common challenges or frustrations you encounter with current graphics tools or technologies during development?	Understand pain points that could reveal technical limitations or unmet needs in vector and motion graphics workflows.	Research Objective 3
4	In terms of visual quality and performance, what impact do you think vector graphics have on the user experience?	Gauge perceived benefits and challenges of vector graphics on the visual and performance aspects of applications.	Research Objective 1

No.	Question	Justification	Relevant Research Objective
5	Do you currently use vector graphics for interactive or dynamic content in your projects? If so, how effective have they been, and what are the main limitations?	Learn about specific use cases and barriers in vector graphics implementation within real-time environments.	Research Objective 2 and 3
6	What tools or techniques do you typically use for integrating motion graphics, such as animations from platforms like Lottie, Rive, or After Effects?	Explore how developers are bringing in external motion graphics and the role they play within interactive applications.	Research Objective 2
7	What would you like to see improved in existing tools for integrating vector or motion graphics in real-time applications or games?	Identify specific feature or functionality gaps in current technologies.	Research Objective 3
8	In your experience, how does the integration of vector or motion graphics impact the user interface or player experience?	Understand the perceived effect of these graphics on user engagement, immersion, or overall experience.	Research Objective 1
9	Do you have any specific feedback or suggestions that could help streamline the workflow of incorporating dynamic vector or motion graphics in game engines or real-time applications?	Solicit practical suggestions to improve workflows, targeting a user-centered design approach.	Research Objective 2 and 3

Table 7: Interview questions

3.3. Analysis

Questionnaire Analysis

Interview Analysis

References

Appendices

References

Add styles to UXML. Unity. Retrieved November 13, 2024, from https://docs.unity3d.com/Manual/UIE-add-style-to-uxml.html

Alvin, O. (2020). Rendering Resolution Independent Fonts in Games and 3D-Applications. *LU-CS-EX*.

Anderson, C. (2020, August 10). *Introducing Bevy 0.1*. https://bevyengine.org/news/introducing-bevy/

Anderson, C. (2024, August 10). *Bevy's Fourth Birthday*. https://bevyengine.org/news/bevys-fourth-birthday/

Batra, V., Kilgard, M. J., Kumar, H., & Lorach, T. (2015). Accelerating vector graphics rendering using the graphics hardware pipeline. *ACM Transactions on Graphics (TOG)*, *34*(4), 1–15.

Carlier, A., Danelljan, M., Alahi, A., & Timofte, R. (2020). Deepsvg: A hierarchical generative network for vector graphics animation. *Advances in Neural Information Processing Systems*, *33*, 16351–16361.

Comparison of UI systems in Unity. Unity. Retrieved November 13, 2024, from https://docs.unity3d.com/Manual/UI-system-compare.html

Crow, T. S. (2004). Evolution of the Graphical Processing Unit. *University of Nevada*.

Dalstein, B., Ronfard, R., & Van De Panne, M. (2015). Vector graphics animation with time-varying topology. *ACM Transactions on Graphics (TOG)*, *34*(4), 1–12.

DesLauriers, M. (2015, March 9). *Drawing Lines is Hard*. https://mattdesl.svbtle.com/drawing-lines-is-hard

DEWI, I. G. A. A. O. (2021). Understanding data collection methods in qualitative research: the perspective of interpretive accounting research. *Journal of Tourism Economics and Policy*, *1*(1), 23–34.

Filimowicz, M. (2023, January 4). *History of Video Games*. https://medium.com/understanding-games/history-of-video-games-9465b2eec44c

Gagnon, C. (2019, October 1). *UMG Best Practices*. https://www.unrealengine.com/en-US/tech-blog/umg-best-practices

Ganacim, F., Lima, R. S., De Figueiredo, L. H., & Nehab, D. (2014). Massively-parallel vector graphics. *ACM Transactions on Graphics (TOG)*, *33*(6), 1–14.

Glazkov, D. (2021, November 25). *Retained and immediate mode*. https://glazkov.com/2021/11/25/retained-and-immediate-mode/

Hana, R. (2023, March 28). *Getting started with egui in Rust*. https://whoisryosuke.com/blog

HeySERP-Team. (2023, June 9). *What Are the Advantages and Disadvantages of a WYSIWYG Editors*. HeySERP. https://heyserp.com/blog/wysiwyg-editors/

Higinbotham, W. (1958). *Tennis for two*.

Hoyer, A. W. *SVG + CSS = ⭐*. Retrieved November 12, 2024, from https://andrew.wang-hoyer.com/experiments/svg-animations/

Iché, T. (2016, November 28). *Free VFX image sequences and flipbooks*. https://unity.com/blog/engine-platform/free-vfx-image-sequences-flipbooks

Introduction to UXML. Unity. Retrieved November 13, 2024, from https://docs.unity3d.com/Manual/UIE-WritingUXMLTemplate.html

Jayathilaka, C. (2020, July 30). *Agile Methodology*. https://medium.com/@chathmini96/agile-methodology-30ec4cdf3fc

Jeremias, P., & Quilez, I. (2014). Shadertoy: Learn to create everything in a fragment shader. In *SIGGRAPH Asia 2014 Courses: SIGGRAPH Asia 2014 Courses* (pp. 1–15).

Johnson, B. (2024, February 28). *Simplifying Godot UI Development for All Levels*. https://gamedevartisan.com/tutorials/understanding-godot-ui-control-nodes

Kashivskyy, A. (2024, October 7). *Imperative vs. Declarative Programming - Pros and Cons*. https://www.rootstrap.com/blog/imperative-v-declarative-ui-design-is-declarative-programming-the-future

Kilgard, M. J., & Bolz, J. (2012). GPU-accelerated path rendering. *ACM Transactions on Graphics (TOG)*, *31*(6), 1–10.

Kok, B. (2021). Custom Editor with UI Toolkit. In *Beginning Unity Editor Scripting: Create and Publish Your Game Tools* (pp. 71–110). Apress. https://doi.org/10.1007/978-1-4842-7167-4_4

Kokojima, Y., Sugita, K., Saito, T., & Takemoto, T. (2006). Resolution independent rendering of deformable vector objects using graphics hardware. In *ACM SIGGRAPH 2006 Sketches: ACM SIGGRAPH 2006 Sketches* (p. 118–es).

Krogh-Jacobsen, T. (2023, November 27). *New UI toolkit demos for programmers and artists: Unity blog*. Unity. https://unity.com/blog/engine-platform/new-ui-toolkit-demos-for-programmers-artists

Levien, R. (2020, December 10). *The piet-gpu vision*. https://github.com/linebender/vello/blob/98192612d9f4b7aed9b3223680527473ec449ee9/doc/vision.md

Li, R., Hou, Q., & Zhou, K. (2016). Efficient GPU path rendering using scanline rasterization. *ACM Transactions on Graphics (TOG)*, *35*(6), 1–12.

Linebender. (2024, July 3). Linebender. https://linebender.org/

Loop, C., & Blinn, J. (2005). Resolution independent curve rendering using programmable graphics hardware. In *ACM SIGGRAPH 2005 Papers: ACM SIGGRAPH 2005 Papers* (pp. 1000–1009).

Mateja, D., Armbruster, R., Baumert, J., Bleil, T., Langenbahn, J., Schwedhelm, J. C., Sester, S., & Heinzl, A. (2023). AnimateSVG: autonomous creation and aesthetics evaluation of scalable vector graphics animations for the case of brand logos. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(13), 15710–15716.

Mädje, L. (2022). A Programmable Markup Language for Typesetting. *Technical University of Berlin*, 1–77.

Nehab, D., & Hoppe, H. (2008). Random-access rendering of general vector graphics. *ACM Transactions on Graphics (TOG)*, *27*(5), 1–10.

Ratermanis, A. (2017, December 29). *Vector vs raster: What's best for your logo*. https://www.ratermanis.com/blog/2017/12/28/vector-vs-raster

Ray, N., Cavin, X., & Lévy, B. (2005). Vector Texture Maps on the GPU. *Inst. ALICE (Algorithms, Comput., Geometry Image Dept. INRIA Nancy Grand-Est/loria), Tech. Rep. ALICE-TR-05-003*.

Ricardo, A. (2020, September 23). *Imperative v Declarative UI Design - Is Declarative Programming the future?*. https://www.rootstrap.com/blog/imperative-v-declarative-ui-design-is-declarative-programming-the-future

Rick, D. W., McAllister, K. S., & Ruggill, J. E. (2024). Vector Graphics. In *Encyclopedia of Computer Graphics and Games: Encyclopedia of Computer Graphics and Games* (pp. 1967–1970). Springer.

Roadmap for 2023. Linebender. Retrieved November 14, 2024, from https://github.com/linebender/vello/blob/98192612d9f4b7aed9b3223680527473ec449ee9/doc/roadmap_2023.md

Rodney. (2024, July). *Trying egui: building a Cistercian Clock with Rust GUI*. https://rodneylab.com/trying-egui/

Rueda, A., De Miras, J. R., & Feito, F. R. (2008). GPU-based rendering of curved polygons using simplicial coverings. *Computers & Graphics*, *32*(5), 581–588.

Santos, R. (2023, May 25). *Unreal Engine 5 UI Tutorial*. https://www.kodeco.com/38238361-unreal-engine-5-ui-tutorial

Satran, M., & Radich, Q. (2019, August 24). *Retained Mode Versus Immediate Mode*. Microsoft Ignite. https://learn.microsoft.com/en-us/windows/win32/learnwin32/retained-mode-versus-immediate-mode

Sigireddyviswesh. (2023, August 27). *Difference between Markup and Markdown language*. https://medium.com/@sigireddyviswesh/difference-between-markup-and-markdown-language-e087bfff06d2

Siwiec, A. (2023, March 21). *Week 6: The Egui Rust Framework*. https://siwiec.us/blog/week-6-the-egui-rust-framework/

Stanford, D. (2024, January 11). *Celebrating the Vectrex and Vector Graphics in Gaming*. https://www.gloo.digital/blog/celebrating-the-vectrex-and-vector-graphics-in-gaming

Theme Style Sheet (TSS). Unity. Retrieved November 13, 2024, from https://docs.unity3d.com/Manual/UIE-tss.html

Tian, X., & Günther, T. (2022). A survey of smooth vector graphics: Recent advances in representation, creation, rasterization and image vectorization. *IEEE Transactions on Visualization and Computer Graphics*.

Waseem, A., & Hart, R. (2023, March 2). *Waterfall Methodology: History, Principles, Stages & More*. https://management.org/waterfall-methodology

wgpu. wgpu. Retrieved November 14, 2024, from https://wgpu.rs/