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

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

Contents

Figures

Tables

CHAPTER 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* (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* (1977), *Battlezone* (1980), and *Tac/Scan* (1982). 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.

This project addresses the need for a simplified and efficient workflow to integrate vector graphics into video games.

Vector graphics are often used in situations where scalability and precision are essential. Common applications include: logos, typography, diagrams, charts, motion graphics, etc. Examples of softwares that generates or uses vector graphics content includes Adobe Illustrator, Adobe After Effects, Affinity Publisher, Graphite, and many more. Vector graphics is also used in a wide range of file formats including Scalable Vector Graphics (SVG), Portable Document Format (PDF), TrueType Font (TTF), OpenType Font (OTF), etc. However, these formats are rarely used in the game industry directly (they are often preprocessed into some other formats, i.e. triangulation or signed distance fields [SDF]), as game engines are often only tailored towards rendering triangles and bitmap textures instead of paths and curves that we see in the vector graphics formats.

Markup languages (i.e. Hypertext Markup Language [HTML], Extensible Markup Language [XML]) and style sheets (i.e. Cascading Style Sheets [CSS]) has dominated the way developers layout and style contents. Over the years, technologies like Unity UI Toolkit has evolved in the game industry to adopt the same pattern but with a user friendly editor, allowing users to layout content using a drag and drop manner while styling their content using sliders, color pickers, and input fields (Jacobsen, 2023). While this improves the user experience of content creation, it lacks the capability of integrating logic and custom contents right inside the user interfaces. These features are often delegated to the programmer which can lead to unintended miscommunications.

1.1. Problem Statement

Scalability, precision, and animation

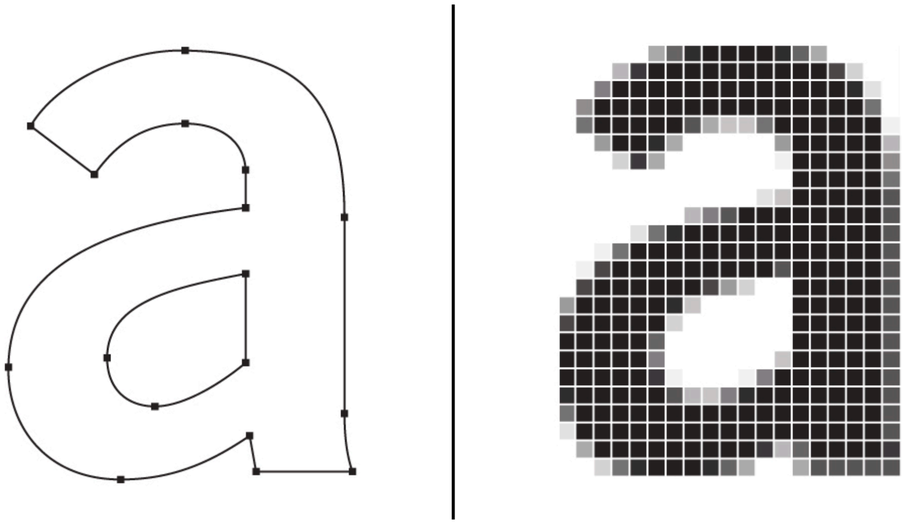

Figure 2: Vector vs Bitmap graphics (Ratermanis, 2017)

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 graphics can be extremely limited and complex because of the rigid grid-like data structure used to store the data. Animating bitmap graphics are commonly done through the use of shaders which directly manipulates the individual pixels, or relying on image sequences (flipbooks) which produces an illusion of movement.

Unlike raster graphics, which rely on a fixed grid of pixels, vector graphics are 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.

Lastly, 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 photographs of a cat are far better represented using bitmap formats.

<u>Lack of support on UI/UX creation for complex interactivity</u>

Most game engines in the market like Unity, Godot, Game Maker, and Unreal Engine uses a *What You See Is What You Get* (WYSIWYG) editor for creating user interfaces. WYSIWYG editors are visual centric tools that let users work directly within the presentation form of the content (Mädje, 2022). Users normally layout their contents using a drag and drop editor and then style them using a style-sheet. 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 application programming interface (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.

While hot-reloading is applicable for the layout and styling (and simple logic to some extend) of a content. 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.

1.2. Project Aim

This project aims to empower creators to create rich and visually appealing content in games in an efficient and streamlined workflow, by allowing them to focus most of their time on the content instead of the technical details needed to achieve the look or feel that they envisioned.

1.3. Research Objectives

The objectives of this project are as follows:

1. To utilize Vello, a compute-centric vector graphics renderer for rendering animated and dynamic vector graphics content.
2. To create an intuitive and yet powerful (programmable) workflow for generating animated and dynamic content using Typst.
3. To allow creators to focus on the creative aspects of game development.
4. To implement radiance cascades, a technique that provides realistic lighting without sacrificing real-time performance.

1.4. Scope

The scope of this project invovles making a game that utilizes 2 custom libraries (1 for global illumination and 1 for interactive vector graphics content). The creation of the game will ensure that 2 of our libraries are production ready by the end of this project.

Libraries (Crates)	
Bevy Radiance Cascades (Bevy RC)	A 2D global illumination solution for the Bevy game engine.
Velyst	An interactive Typst content creator using Vello and Bevy.
Game	
Lumina	A 2D top down fast paced objective based PvPvE game.

Tasks to be executed:

1. Develop the **Bevy RC** crate.
   a. Develop the radiance cascades algorithm.
   b. Implement radiance cascades into Bevy's 2D render graph.
   c. Support emissive and translucent materials.
   d. Support directional light beams / spot lights.
   e. Support negative lighting effects (light consumption).
   f. Support rim lighting for better visual effects.

2. Develop the **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 workflow for UI creation using Typst.
   e. Write up a getting started documentation to make on-boarding easier for new developers.

3. Develop the **Lumina** game.
   a. Create a game design document (GDD) for the game.
   b. Integrate both **Bevy RC** and **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

Constraint	Reason
	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 global illumination and vector graphics is crucial. Any constraints in these areas can impact the project's scope and delivery timeline.

<u>What will be done in this project:</u>

1. **Global illumination**: Our implementation of radiance cascades will support any kind of 2D shapes with emissive, translucent, and non-emissive color material. It is aimed to be a plug and play plugin. Users should be able to use it without the need to learn about complex configurations.

2. **Typst compiler**: A custom implementation of Typst compiler will be created 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 reflect their saved changes immediately.

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

4. **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.

<u>What will not be done in this project:</u>

1. **No reflections & refractions**: Radiance cascades by default does not support reflections and refractions. Implementing these 2 features requires a tremendous amount of additional research, experimentation, and effort. Not to mention that implementing radiance cascades itself can already be extremely challenging as an individual working in such time constrait.

2. **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.

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

1.5. Potential Benefit

Tangible Benefit

Intangible Benefit

Target User

Velyst will particularly be targeted towards UI/UX developers, motion graphics creators, and vector graphics enthusiasts. **Bevy RC** will be targetted towards visual effects (VFX) artists, game programmers, and technical artists. Because these technologies are built on top of Rust and Bevy, the general users will come from the Rust + 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*.

1.6. Overview of the IR

1.7. Project Plan

CHAPTER 2: Literature Review

2.1. Domain Research

Vector Graphics

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.

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. This allows for extreme zoom levels without sacrificing qualities.

Re-tesselation of vector graphics can be computationally expensive, especially when it's 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. 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.

Interactive UI/UX

Beneath all graphical interfaces lies the underlying code that structure and renders the visual elements. The two most notable approach towards creating user interface frameworks are immediate-mode graphical user interface (IMGUI) and retained-mode graphical user interface (RMGUI). Some popular IMGUI frameworks includes Dear IMGUI and egui (*Dear Imgui*, n.d.; *Egui*, n.d.), while some popular RMGUI frameworks includes Xilem (*Xilem*, n.d.). Although powerful, these UI frameworks strongly relies on hardcoded programming.

Enter the web technologies. Modern browsers typically render UI elements using markup languages like HTML and SVG for structuring the content and style-sheets like CSS for styling them. 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. Notable frameworks that utilizes this model includes Unity UI Toolkit, React, Vue, etc (Jacobsen, 2023; *React*, n.d.; *Vue*, n.d.).

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).

A major limitation of simple markup languages like HTML is that structure changes can only be achieved through code. 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 markdown.

Typst is a competitor of LaTeX, designed to simplify the typesetting process with a modern and intuitive approach. Unlike its predecessors, Typst can directly embed logic. 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. This currently works only in theory, as Typst is primarily a document generator without a user-friendly interface for modifying defined variables.

2.2. Similar Systems/Works

2.3. Technical Research

CHAPTER 3: Methodology

3.1. System Development Methodology

3.2. Data Gathering Design

3.3. Analysis

References

Appendices

References

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

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.

Battlezone. (1980). Atari.

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

Dear ImGui. Retrieved September 22, 2024, from https://github.com/ocornut/imgui

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

Egui. Retrieved September 22, 2024, from https://github.com/emilk/egui

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

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.

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

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

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

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).

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

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).

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*.

React. Retrieved September 22, 2024, from https://react.dev/

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.

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.

Space Wars. (1977). Cinematronics.

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

Tac/Scan. (1982). Sega.

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*.

Vue. Retrieved September 22, 2024, from https://vuejs.org/

Xilem. Retrieved September 22, 2024, from https://github.com/linebender/xilem