INVESTIGATION REPORT

Establishing a Workflow for Real-time Global Illumination and Vector Graphics in Enhancing Dynamic Game Environments

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: Dr. Tan Chin Ike

27th November 2024

Acknowledgement

Abstract

Achieving visually rich and interactive content in real-time without compromising performance is a key aspect of immersive gameplay. This project seeks to establish a streamlined workflow that incorporates real-time global illumination (GI) and compute-centric vector graphics (VG) into dynamic game environments in a way that is accessible and adaptable for game developers. These integrations will help improve visual appeal and interactive feedback in gameplay. Our method utilizes *Radiance Cascades* for enabling real-time GI, *Vello* for rendering dynamic VG in real-time, and *Typst* for VG content creation. Using purposive sampling, this study targets developers in the game development industry through the use of questionnaires. It is aimed to gather insights on the importance of GI and VG as well as the level of integrations in state-of-the-art game engines. This paper provides an in-depth look at the challenges and potential methods for integrating these technologies into game development, with an emphasis on their impact on interactive and adaptable content creation. Our approach contributes to advancing infrastructure and fostering innovation, aligning with the goals of Sustainable Development Goal (SDG) 9.

Keywords — radiance cascades, indirect lighting, typesetting, markdown, interactivity, dynamic content

Contents

Figures

Tables

CHAPTER 1: INTRODUCTION

1.1. Problem Background

Limitations of scalability in current real-time global illumination techniques

Global illumination has been a notoriously hard problem to solve in computer graphics. To put things into perspective, global illumination intends to solve the *many to many* interactions between light, obstacles, and volumes. Ray tracing is an algorithm that calculates these light interactions by tracing lights from the camera into the scene, following their paths as they bounce off surfaces and interact with materials. Each bounce contributes to the final color and lighting of the scene, accounting for reflections, refractions, and scattering.

Unfortunately, ray tracing is just too slow for real-time applications. In real-time game engines like Unity and Unreal Engine, light probes (a.k.a radiance probes) are placed around the scene to capture lighting information, which can then be applied to nearby objects. To smoothen out the transition between probes, objects interpolate between nearest surrounding probes, weighted by distance to approximate the global radiance. This technique leads to questions like “how many probes should a scene have?” or “how much probes is a good approximation?”. Ultimately, it becomes a trade-off between fidelity versus performance, with more probes resulting in better approximation, while fewer probes improve performance. This paradoxical issue raises the challenge of finding the optimal balance. This dilemma underscores the need for smarter, adaptive techniques, ensuring both visual fidelity and efficiency.

Advantages of vector graphics over bitmap graphics in terms of animation

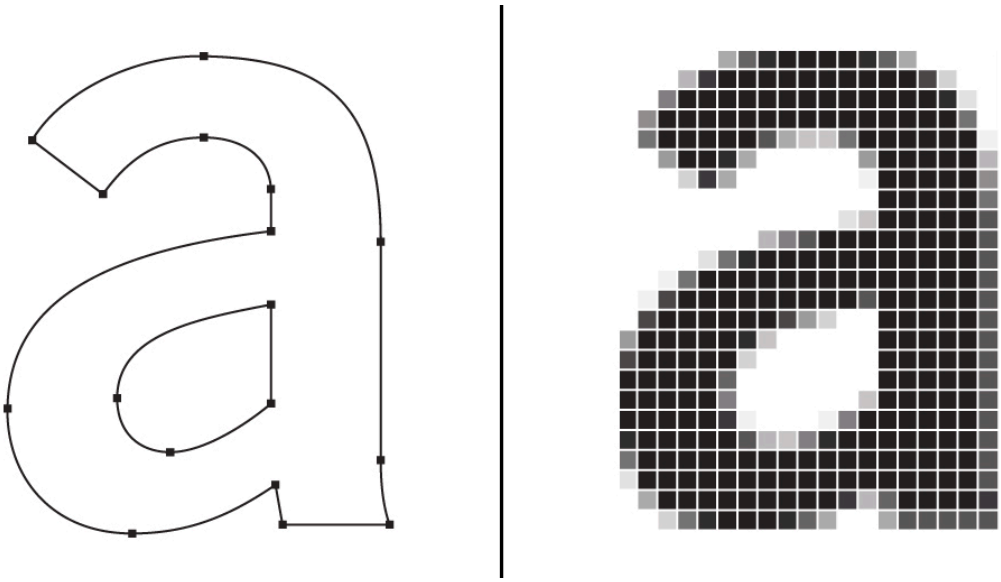

Figure 1: Vector vs Bitmap graphics (Ratermanis, 2017)

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 1). 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.

Lack of support on UI/UX creation for complex interactivity

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 wouldn't 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. 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	

What will do in this project:

What will not be done in this project:

1.5. Potential Benefit

Tangible Benefit

Intangible Benefit

Target User

References

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.

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*.