Real-time Global Illumination and Dynamic Compute-Centric Vector Graphics in Games

Contents

1. Introduction

Achieving visually rich and interactive content in real-time without compromising performance is a key aspect of immersive gameplay. This project addresses two major challenges in modern game development: creating dynamic, interactive user experiences and implementing accurate, real-time lighting models. Tackling these challenges requires three key innovations: a compute-centric vector graphics renderer, a programmable approach for developing interactive content, and a performant global illumination technique.

Vector graphics is a form of computer graphics where visual images are generated from geometric shapes, such as points, lines, curves, and polygons, defined on a Cartesian plane. 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.

Calculating indirect lighting is extremely computationally expensive, as it requires simulating how light bounces off surfaces and interacts with the environment. 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 too slow for real-time applications, like games. New techniques like light probes and light baking has been employed to approximate global illumination in modern game engines. However, the major issue still exists for these techniques — scalability to larger and more complex scenes.

2. Problem Statement

2.1. Vector Graphics

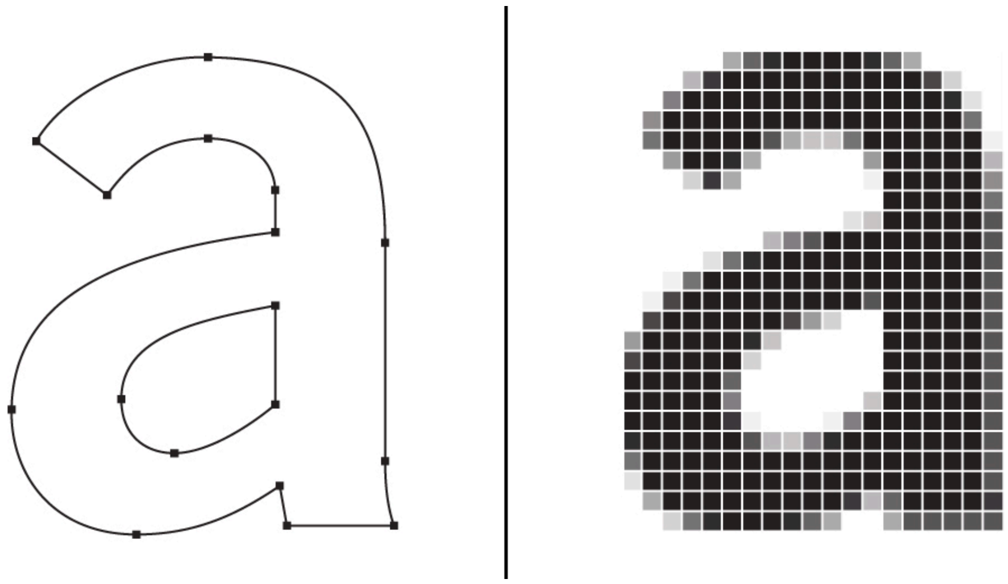


Figure 1: Vector vs Bitmap graphics (Ratermanis, 2017)

Traditional methods of rendering 2D graphics had 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.

2.2. Interactive UI/UX

Most game engines in the market like Unity, Godot, Game Maker, and Unreal Engine uses a WYSIWYG *(What You See Is What You Get)* 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 based 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 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.

In summary, WYSIWYG editors are great for prototyping but suffers from complex animations and interactions.

2.3. Global Illumination

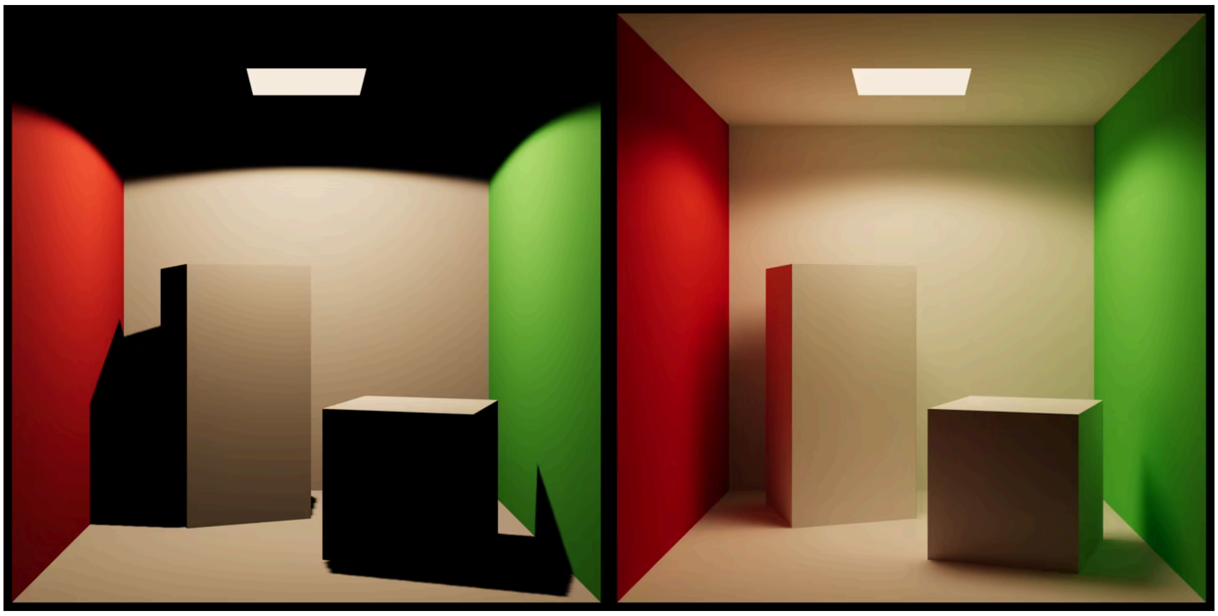


Figure 2: Local illumination vs global illumination (Jalnionis, 2022)

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. 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 into a cube map, which can be applied to nearby objects. To smoothen out the transition between probes, objects interpolate between various nearby probes weighted by distance to approximate the global radiance.

Manual placement of probes leads to questions like "how many probes should a scene have?" or "how much probes is a good approximation?". It ultimately becomes a trade-off between fidelity and 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.

3. Project Aim and Objectives

In this project, we aim to empower creators to create rich and visually appealing content in games in an efficient and streamlined workflow. This allows the creator to have the luxury of focusing most of their time on quality content rather than the technical details needed to achieve the look or feel that they envisioned.

The objectives of this project are:

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.
3. To streamline the collaboration between UI/UX developer and gameplay programmers
4. To allow creators to focus on the creative aspects of game development.
5. To implement Radiance Cascades, a technique that provides realistic lighting without sacrificing real-time performance.

4. Literature Review

5. Deliverables

This project introduces **Velyst**, an innovative approach towards generating interactive content. It utilizes **Vello**, a compute-centric vector graphics renderer (Linebender, n.d.), and **Typst**, a programmable markup language for typesetting (Mädje, 2022). Velyst provides an extremely streamlined workflow that allows both UI/UX developers and gameplay programmers to easily collaborate in a project. The following are the deliverables that will be implemented in the Velyst library:

1. Allows the user to create interactive content that responds to user inputs in real-time.
2. Allows the user to perform hot-reloading during the development phase.
3. Allows the user to synchronize components with in-app states dynamically.
4. Allows the user to embed logic directly inside Typst scripts for rendering complex scenes.

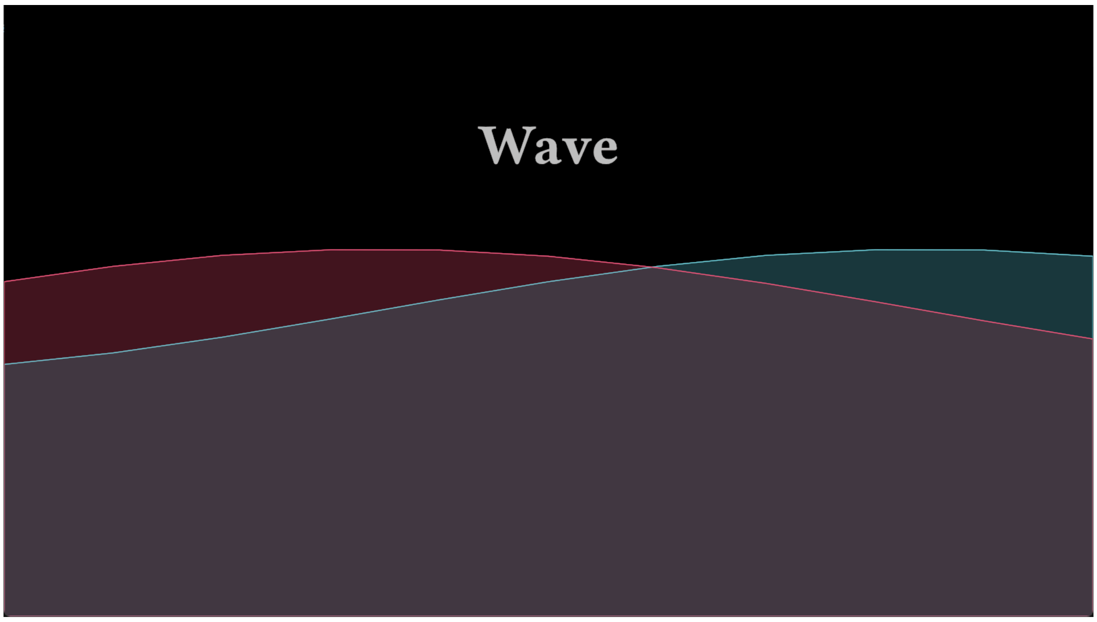
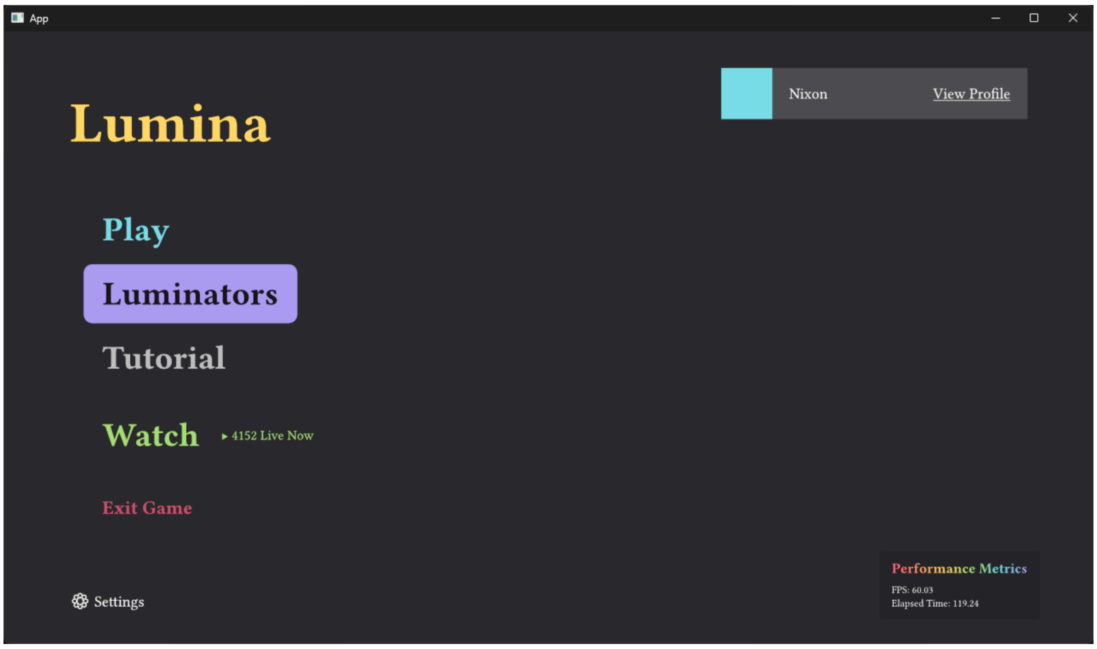


Figure 3: Velyst wave (custom shape + animation) demo



FPS: 60.03
Elapsed Time: 119.24

Figure 4: Velyst user interface demo

A library will also be created that utilizes the **Radiance Cascades** technique for rendering real-time 2D global illumination in the Bevy game engine. The following are the deliverables for the Radiance Cascades library:

1. Allows the user to implement global illumination using Radiance Cascades.
2. Allows the user to create adaptive lighting that responds to scene changes in real-time.
3. Allows the user to use arbitrary emmisive shapes to illuminate the scene in game.

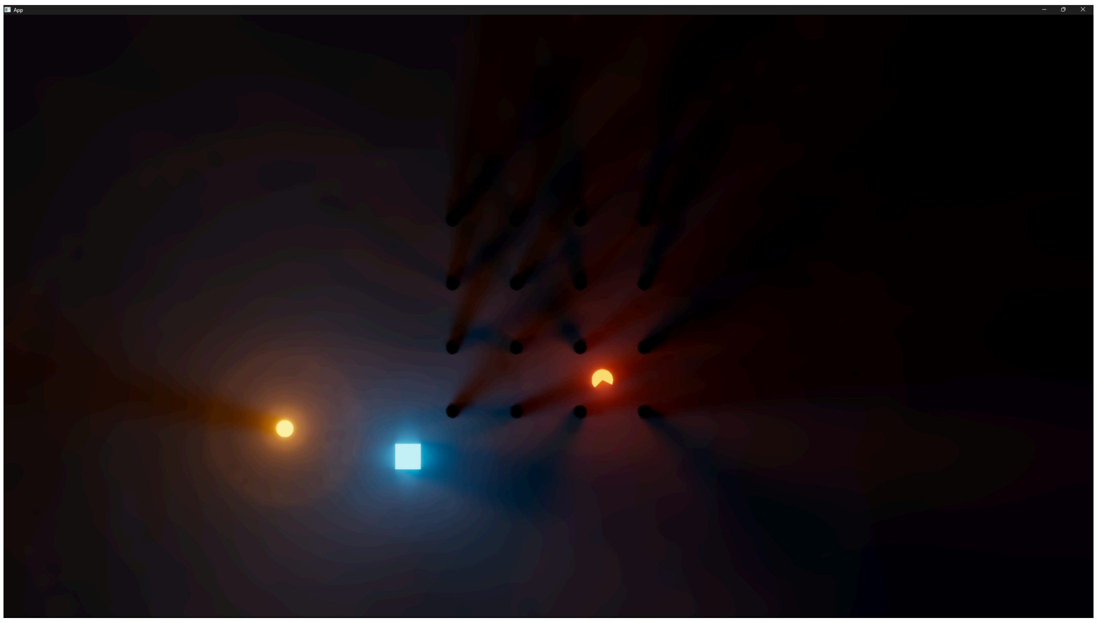


Figure 5: Radiance cascades demo

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.

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

Jalnionis, K. (2022, September). Unity. https://unity.com/blog/engine-platform/5-common-lightmapping-problems-and-tips-to-help-you-fix-them

Linebender. *Linebender/Vello: A GPU compute-centric 2D renderer.* https://github.com/linebender/vello

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). *Vector vs raster: What's best for your logo*. Arne Ratermanis. 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*.