3D Game Engine Development with C++ and OpenGL

Book · February 2024

CITATIONS
0

READS
3,210

1 author:

Franc Pouhela
Deutsches Forschungszentrum für Künstliche Intelligenz
9 PUBLICATIONS **4** CITATIONS

SEE PROFILE

A MASTER PIECE BY

FRANC POUHELA

3D GAME ENGINE DEVELOPMENT

Learn how to Build a Cross-Platform 3D Game Engine with C++ and OpenGL

3D GAME ENGINE DEVELOPMENT

Learn how to Build a Cross-Platform 3D Game Engine with C++ and OpenGL.

First Edition, March 30, 2024

ISBN: 9798878081344

Acknowledgement

With heartfelt humility and profound gratitude, I wish to express my profound appreciation to my family, who wholeheartedly supported me in bringing this remarkable book to fruition. Additionally, I extend my sincere thanks to you, as your desire to learn has played an integral role in bringing this project to fruition. Finally, I extend my gratitude to God for bestowing upon me life, inspiration, and unwavering strength.

Foreword

To maximize your learning experience with this book, we have prepared a dedicated GitHub repository for you. This repository houses all the code and additional materials featured within the book. If you encounter any challenges while progressing through the various parts of this book, you can easily clone specific versions of the source code from the repository's distinct branches.

1. Piracy

Copyright piracy is an ongoing problem. We take intellectual property protection and licensing very seriously. If you come across any illegal copies of this book, in any form, on the Internet, please contact us as soon as possible using the email address below. We appreciate your support in safeguarding our writers' safety and our ability to supply you with crucial information.

2. Errata

Despite our best efforts to ensure the accuracy of our material, errors occasionally occur. If you notice a mistake in this book, whether it is in the text or in the code, we would appreciate it if you could let us know. By doing so, you save other readers time and effort while also helping us improve future editions of this work. Send us an email at *madsycode@gmail.com* if you find any errors.

Preface

This book is not meant for individuals who are completely novices in C++ programming or any other programming language in general. To fully benefit from the material provided in this book, you should possess a grasp of fundamental coding concepts, including object-oriented programming, inheritance, template functions, pointers, and related topics. Rest assured, no prior experience with OpenGL is required, but it is highly recommended.

1. Expectation

You are embarking on a journey to build a cross-platform 3D game engine from the ground up using C++ and OpenGL. The adventure begins with setting up a versatile development environment and a robust build system, laying the foundation for the challenges that lie ahead. We dive into core functionalities such as message logging, window events input handling, etc. As the journey progresses, we venture into more advanced terrain, tackling the implementation of critical features such as graphics rendering, physics, scripting, serialization, etc. Finally, it culminates with the implementation of a graphical user interface to improve interaction with the engine's features and game creation.

2. Limitation

It is essential to emphasize that this book is not asserting that the game engine built within its pages is production-ready. Instead, our aim is to lay down solid foundations for you to shape according to your aspirations. Approach the knowledge presented here with a discerning eye, recognizing it as a guiding light rather than absolute truth. This book stands as a testament to the extent of my own odyssey in learning computer graphics and programming, a testament to the ever-evolving quest for knowledge in this dynamic field.

M.Eng. Franc Pouhela, Germany, March 30, 2024

Contents

Part I.

Getting Started

1. Introduction

> *All hard work brings a profit, but mere talk leads only to poverty.*
> (King Solomon)

I have always been fascinated by how games come together. The sheer magic of objects moving, animations unfolding, and effects blending seamlessly always amazed me. I wanted to dive into that world, understand it, and eventually create it myself. So, I began by learning the basics of programming. Then, I took the plunge into creating simple games, one step at a time. After many years of learning and experimenting, it finally clicked for me. I realized that I had to share this knowledge with others who might be going through the same experience.

Developing a custom game engine in today's landscape might raise eyebrows among many, as there's an abundance of really good free and open-source alternatives readily accessible. To challenge this prevailing wisdom might appear unconventional at best. However, I firmly believe that there are compelling reasons for you to explore the intricate world of game engine development. This pursuit not only fuels the industry's motion but also provides a remarkable avenue for expanding your technological proficiency. Building a custom game engine allows you, as a developer, to tailor the engine to your specific needs and preferences. This gives you greater control over the game development process and allows you to create games or applications that are unique and differentiated from those made with commonly used engines.

Now, let me be clear from the outset: Creating a game engine from scratch is not a task for the fainthearted. It's a pursuit that demands a solid grasp of computers, programming, mathematics, and physics. It beckons you to embark on a journey where you will unravel intricate concepts and then transmute them into lines of code capable of breathing life into a vast array of virtual experience ideas. Once you emerge from the tunnel's far end, you will possess a solid foundation for designing and developing high-performance game engines. With that being said, I would like to encourage you not to give up.

1.1. What is a Game Engine?

A game engine is a piece of software or a platform that provides the core functionalities and tools required to develop, run, and manage video games. It serves as the technological backbone for game development, offering a set of prebuilt modules, libraries, and systems that streamline the creation of interactive digital experiences. Game engines are also versatile tools used not only for video game development but also for simulations, architectural visualization, training, and interactive media across various industries. Their continuous evolution and improvement empower game developers to focus on creativity and gameplay while leveraging the engine's technological capabilities.

Over the past two decades, the development of game engines has undergone a remarkable transformation. Advancements in hardware and software have led to the creation of highly sophisticated engines, enabling breathtaking graphics, physics, and interactivity. Open-source engines such as Godot and Unreal have democratized game development, allowing a wider audience to create immersive experiences. Additionally, mobile games have exploded, demanding lightweight engines for portable devices. The industry has also seen a shift toward real-time ray tracing and Virtual Reality (VR) technology. In summary, the last 20 years have witnessed a rapid evolution in game engine technology, making game development more accessible and pushing the boundaries of what's possible in gaming.

1.1.1. Popular Game Engines

Numerous acclaimed game engines are being embraced worldwide. Here is a list of some of the prominent ones.

❑ **Unity**: A cross-platform game engine popular among independent developers and small studios, valued for its user-friendly interface and broad platform support, including PC, console, mobile, and web.

❏ **Unreal**: A powerful and widely used game engine developed by Epic Games, known for creating various game genres, including first-person shooters, action games, and Role-Playing Games (RPGs).

❏ **Godot**: Developed as an open-source game engine, Godot stands out for its user-friendly interface, flexibility, and community-driven development. It offers a node-based scene system that simplifies game design and enables both 2D and 3D game development

❏ **GameMaker**: A favorite among hobbyists and beginner game developers, GameMaker's simplicity and drag-and-drop interface facilitate 2D game development without extensive programming.

❏ **CryEngine**: Developed by Crytek, CryEngine stands out for its high-quality graphics and extensive platform compatibility, making it suitable for first-person shooters and action games.

❏ **Lumberyard**: Developed by Amazon and based on CryEngine, Amazon Lumberyard is a free cross-platform game engine used for various genres of games, including Massively Multiplayer Onlines (MMOs), RPGs and open-world games.

Figure 1.1.: Popular Engines [1]

These are merely a handful of notable game engines. Numerous

others exist, each possessing its own distinctive set of features and capabilities. In addition, there are independent game engines being developed by content creators on platforms such as YouTube, such as the Hazel Engine [34] and the Cave Engine [27].

1.2. Game Engine Design

There are many different approaches to the design of game engines, and the specific design of a game engine will depend on the needs and goals of the game to which it is used. Quite often, we have the tendency to associate the term ”game engine” with the idea of a user interface for creating games, but this perception is somewhat misconceived. A game engine does not necessarily need to encompass all the tools commonly found in existing popular ones. However, some common features of game engines include:

❏ **Rendering Engine**: Responsible for generating the visual output of the game, including 3D models, textures, and special effects. It utilizes graphics Application Programming Interface (API) such as OpenGL, Vulkan, or Direct3D to draw the game world on the screen.

❏ **Physics Engine**: Simulates the physical properties of objects in the game world, such as gravity, mass, and friction. It enables realistic interactions between objects and the environment. Software Development Kits (SDKs) such as NVIDIA PhysX [16], Bullet Physics [25] are commonly used for this purpose.

❏ **Sound Engine**: manages the audio aspects of the game, including sound effects, music, and dialogue. It may utilize libraries such as SDL_Mixer, OpenAL, or FMOD to play sound files and control audio properties such as volume, pitch, and spatialization.

❏ **Scripting Engine**: The core of the game engine is responsible for managing the game state, updating the game world, and handling user input. It can employ programming languages such as C#, Lua, or Python to implement game mechanics and AI.

❏ **Assets Management**: In charge of managing the game's assets, such as 3D models, textures, fonts, and audio files. This system may include tools for importing, exporting, and organizing assets.

❏ **Scene Editor:** The interface through which developers interact with the engine's features to make their game ideas come true.

1.2.1. High-Level Architecture

In contrast to many other industries, the gaming sector operates without a governing body that prescribes a definitive blueprint for constructing a proper architecture. This duality has both advantages and disadvantages. On the positive side, it grants developers the freedom to experiment with their own innovative ideas. However, on the downside, the absence of standardized guidelines makes it challenging to gauge the effectiveness of one's approach objectively. Naturally, best practices exist and should be considered to avoid redundant mistakes.

Game engines are intentionally designed to be flexible, creating a fertile ground for nurturing creativity and enabling artists to bring their unique visions to life. This diversity aligns with the broad spectrum of purposes, requirements, and objectives that each game serves, underscoring the distinctive character of every development effort.

(Figure 1.2) describes the high-level architecture of what we are aiming at this book. The engine's fundamental capabilities are encapsulated within a shared library in Windows or a shared object in Linux. In computer science, a shared library, is a file designed to be simultaneously employed by multiple programs. During runtime, the library's modules are dynamically loaded into the memory space of an executable, such as **Game.exe** and **Empy.exe**, facilitating access to the engine's defined functionalities. **Game.exe** represents the polished, distributable product, which allows others to enjoy the game made using the engine, while **Empy.exe** serves as a versatile tool for editing game scenes and preparing it for distribution.

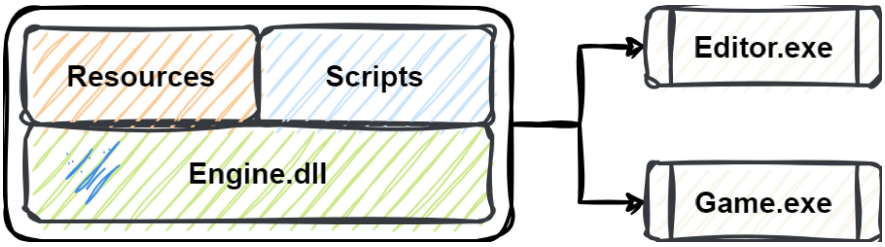

Figure 1.2.: High-Level Architecture

In order to interact with the engine's functionalities, we leverage scripting. This involves crafting supplementary code on top of the existing engine to amplify its capabilities and interface with its features. More insights into this process will be provided in a subsequent chapter.

1.3. Development Tools

A widely accepted consensus is that C++ has firmly established itself as the preferred programming language for game engines, primarily due to its performance optimization capabilities, cross-platform compatibility, and ability to interface with low-level APIs. Its support for object-oriented design aids in structuring complex engine components. The vast C++ ecosystem, coupled with the language's use in established engines like Unreal Engine, has cemented its role in the industry. While C++ dominates, the development of game engines also sees contributions from languages such as C#, Rust, and Python, each chosen based on project-specific needs and goals. This book employs C++ because of the reasons mentioned.

1.3.1. Coding Environment

Navigating the landscape of Integrated Development Environments (IDEs) in the realm of C++ development can be a daunting journey. Developers often encounter a multitude of options, each with its own strengths and complexities. Selecting the right IDE, configuring it, and ensuring compatibility with the C++ development ecosystem can pose significant challenges. However, our choice in

this book is Visual Studio Code (VS-Code), a lightweight and versatile code editor developed by Microsoft. VS-Code boasts a rich ecosystem of extensions and plugins that specifically cater to C++ development needs. It combines a user-friendly interface with powerful functionality, making it an excellent choice for both novice and experienced developers. Its flexibility allows us to tailor our development environment to suit the requirements of game engine development seamlessly.

1.3.2. Build System

A build system is a crucial component that automates the process of compiling source code into executable programs or libraries. It encompasses a set of rules and procedures for transforming source files, such as C++ source code and resource files, into machine-readable binary code that can be run on a computer. The build system's primary objectives are to manage dependencies, determine which source files need to be recompiled based on changes, and ensure that the compilation process is efficient and error-free. **CMake**, which we will utilize in this book, plays a pivotal role in simplifying the often complex and platform-dependent process of building C++ projects, making development more manageable and accessible.

C++ projects often rely on external libraries and packages, each with its own set of dependencies and version requirements. This intricate web of dependencies can lead to compatibility issues, version conflicts, and a significant overhead in configuring and maintaining the development environment. However, **Conan**, a C++ package manager, significantly simplifies this process. Conan simplifies the management of dependencies by providing a centralized repository (*https://conan.io/center*) for C++ packages and ensuring version control and consistency. Automates the retrieval, integration, and building of dependencies, avoiding the tedious and error-prone task of manual dependency management. In this book, we will harness the power of Conan to seamlessly manage and integrate the necessary libraries and packages, allowing us to focus on the core aspects of

game engine development without the complexities of dependency wrangling.

1.3.3. Graphics APIs

Graphics APIs are essential software frameworks that enable developers to create visually captivating and interactive applications, particularly in the realm of computer graphics and game development. These APIs serve as intermediaries between the application and the graphics hardware, allowing programmers to harness the full potential of modern Graphics Processing Units (GPUs). Prominent graphics APIs include OpenGL, DirectX, Vulkan, and Metal, each tailored to specific platforms and offering varying degrees of control and performance optimization. The choice of a graphics API often depends on the target platform, project requirements, and the developer's familiarity with a particular framework. These APIs enable developers to render 2D and 3D graphics, implement shaders, manage textures, and achieve real-time visual effects, forming the foundation of immersive gaming experiences and graphical applications on a multitude of devices.

This book employs **OpenGL** [9] as graphics API for several compelling reasons. OpenGL's cross-platform compatibility ensures that the skills acquired are adaptable to a wide range of operating systems and devices. It is industry standard graphics API widely used in game development, providing readers with information on industry best practices. OpenGL's openness, performance control, and educational value make it an accessible and versatile choice for learning the foundations of modern graphics programming.

1.3.4. Scripting APIs

Scripting in game engines is the art of imbuing games with interactivity, dynamism, and custom behavior through the use of scripted code. It serves as the bridge between the core engine of the game and the dynamic elements that make each gaming experience unique. Scripting empowers developers to define gameplay mechanics, cre-

ate complex AI behaviors, orchestrate in-game events, and shape the player's journey. It offers a flexible and efficient way to iterate on game design without delving into the engine's source code, allowing for rapid prototyping and adjustments.

This book incorporates **Lua** [22] as its scripting language due to several compelling reasons. Lua's lightweight and embeddable nature adds extensibility to the game engine without introducing complexity. Its simplicity, popularity in game development, and versatility make it an ideal choice for scripting game logic and behavior, and it boasts a supportive community with ample resources. Lua's seamless integration with C/C++ allows us to leverage its dynamic capabilities while maintaining performance-critical components.

1.3.5. Graphical User Interface

Game engines such as Unity and Unreal allow you to create your games in a Graphical User Interface (GUI) environment and then export the final result as a runtime for specific platforms such as PC, Smartphone, Console, etc. The engine we will be building in this book also needs an editor. (Figure 1.3) shows a preview.

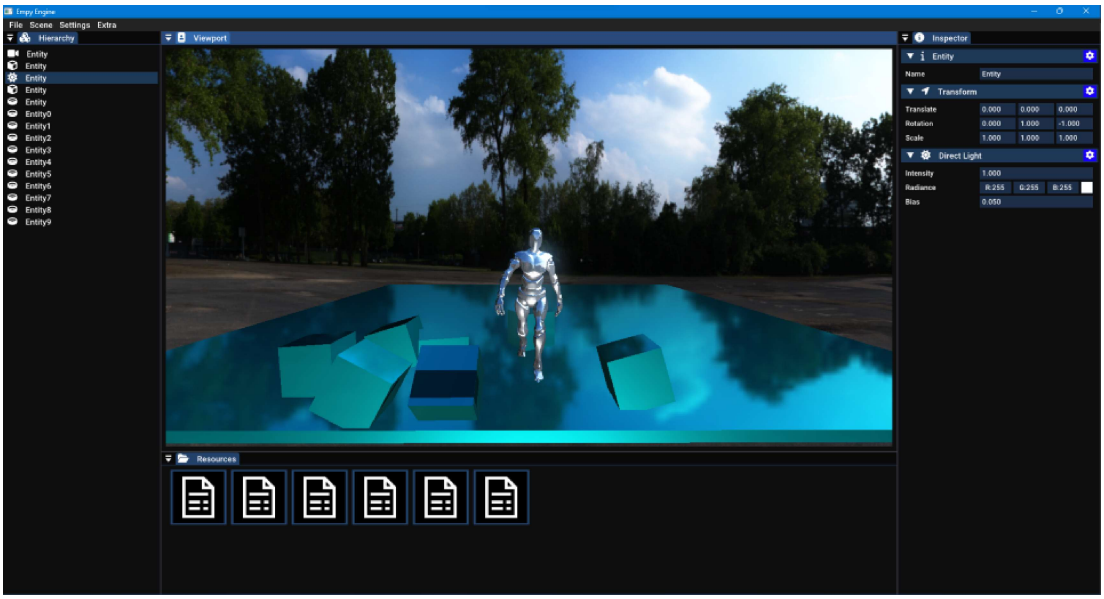

Figure 1.3.: Empy Engine Preview

We will create a game-editor environment using **Dear ImGui** [17]. Dear ImGui is a bloat-free graphical user interface library for C++. It outputs optimized vertex buffers that you can render anytime in your 3D-pipeline-enabled application. It is fast, portable, renderer-agnostic, and self-contained. Dear ImGui is designed to enable fast iterations and empower programmers to create content creation tools and visualization/debug tools. Dear ImGui, it is particularly suited to integration in games engines, full-screen applications, embedded applications, or any applications on console platforms where operating system features are non-standard.

1.4. Summary

Throughout the course of this book, we'll leverage multiple tools and libraries to enhance our engine with features such as sound effects, 3D model loading, serialization, and more. Each new tool will be introduced as needed, seamlessly integrating them into our development process.