ScummVM: a Cross-Platform Virtual Machine

Piero Amendola
Kai Dalen
Sohan Kolla
Stephan Leznikov
Jackson Reid
Harrison Ye

Table of Contents

Abstract

The following report analyzes ScummVM, an open-source emulator that allows users to play classic point-and-click adventure games on modern platforms. The report examines ScummVM's architectural design, system evolution, and developer responsibilities.

Originally created for LucasArts' SCUMM games, ScummVM was developed in 2001. Over time, it has evolved to support multiple game engines and now operates across numerous operating systems and platforms. ScummVM is commonly used to run and preserve old games that would otherwise be incompatible with modern systems.

ScummVM’s architecture follows a layered style, which includes the application layer, core layer, subsystems layer, and hardware layer. These layers interact seamlessly to handle user input, game logic, graphics, audio, and file systems. The core layer manages game engines and coordinates with subsystems like the graphics and audio engines, while the subsystems layer abstracts platform-specific details, enabling ScummVM’s portability across systems.

Developer responsibilities include game engine development, platform-specific support, and quality assurance. This division of responsibilities ensures that ScummVM remains compatible with new systems, provides continuous support for new game engines, and maintains software quality.

In this report, we explore two key use cases. The first use case is Game Initialization, which demonstrates the control and data flow within ScummVM. The second use case focuses on the Frame and Audio Rendering Process, highlighting the system’s use of asynchronous processes and data flow management.

ScummVM’s architecture demonstrates its adaptability and modular design, which allow the system to evolve and support new platforms and games without requiring significant changes to the underlying structure. These design choices ensure the long-term preservation of classic games through modern platforms.

Introduction and Overview

ScummVM is a versatile and open-source application designed to run classic point-and-click adventure games on modern hardware. As a complex software system, it gives users a unique solution for preserving and enhancing the gaming experiences of beloved titles from the past. In this report, we will explore the architecture, control and data flow, system evolution, concurrency, division of developer responsibilities, and two specific use cases that showcase the functionality and user interaction with ScummVM, providing a comprehensive understanding of how the software operates.

Originally developed in 2001 by Ludvig Strigeus, ScummVM began as a personal project aimed at playing *Monkey Island 2* on Linux. Over time, it attracted a dedicated community of developers and gamers, forming itself into a powerful emulator that supports many adventure games across multiple platforms. ScummVM has become an important tool for retro gaming enthusiasts, allowing users to enjoy classic titles on up to date devices without the need for the original hardware.

ScummVM employs a layered architectural style, allowing it to abstract platform-specific details, which is essential for its portability across different operating systems. These layers work collaboratively to form the backbone of ScummVM's functionality, enabling users to select and launch games, manage settings, and experience seamless gameplay.

The control and data flow within ScummVM are closely related. The control flow follows an event driven model, where user interactions trigger events that the core layer processes, resulting in actions across several subsystems. Meanwhile, the data flow involves managing game states, user input, and multimedia resources, making sure that all components work in harmony to deliver a memorable gaming experience.

The division of developer responsibilities is extremely important to ScummVM's success. Contributors are tasked with different responsibilities and roles, including engine development, platform support, quality assurance, and community engagement. This division of labor allows ScummVM to remain current, user-friendly, and adaptable to the evolving scene of retro gaming.

Two use cases will be explored in this report to show the user interaction with ScummVM. The first use case examines the process of selecting and initializing a game, demonstrating the control and data flow during startup. The second use case highlights the frame and audio rendering processes, showcasing how the system efficiently handles multimedia output.

ScummVM solidifies itself as a powerful and effective open-source solution for experiencing classic adventure games, featuring a well-designed architecture, smooth control and data flow, and a variety of roles for developers. This analysis of ScummVM's software architecture allows us to see the complex systems and data flows that work behind the user interface.

Architecture

System Overview

ScummVM is very portable (platform-independent) because of its layered architectural style. Its user interface (UI, Application) layer is the top layer that the user directly interacts with. This layer is composed (components) of a graphical user interface (GUI) system and an input handling system. The GUI allows the user to select and launch games, configure settings for different games, manage game files, and more. The input handling system captures everything the user does and sends it to different layers. The application layer also interacts with the core layer that handles and processes game logic.

The core layer's main responsibilities are managing the game engines, the control flow, and communicating with subsystems that affect gameplay such as graphics, input, and audio. The core layer’s components are the game engine manager, the game logic interpreter, and the control flow. The game engine manager is sent information on which game is chosen by the user, it then selects and initializes the corresponding game engine. The game logic interpreter interprets game scripts and executes game logic accordingly depending on the game. Each game engine has its own game logic interpreter. The control flow manages and loads the in-game state and data. Using this data, it coordinates with the audio, input, and graphics subsystems. The core layer communicates down with the subsystems layer.

The subsystems (abstraction) layer contains many platform-based subsystems that take care of graphics, audio, the filesystem, and user input/output. This is the layer that abstracts away the platform specific details which allows ScummVM to run on many operating systems without any major code changes. The components of this layer include the graphics subsystem, audio subsystem, filesystem subsystem, and input subsystem. The graphics subsystem renders the graphics using APIs that can be used on many different systems (ex: SDL). This subsystem abstracts the user's system's graphics interface. The audio subsystem manages the sound and music. The filesystem subsystem handles game-save states and assets. It also abstracts the differences between different file systems on different platforms. The input subsystem manages user input. This layer communicates downward with the operating system/hardware layer.

The hardware layer represents the OS and hardware the game runs on. Its components platform APIs and hardware devices. The platform APIs are platform interfaces for the subsystems layers components: graphics, audio, and filesystem access. The hardware devices are the physical pieces of the machine necessary to play games: display, sound device, input, storage, etc.

Evolution of the System

The ScummVM development was started in September 2001 by Ludvig Strigeus during his studies at Chalmers University of Technology in Sweden. He was inspired by the desire to play Monkey Island 2 on his Linux machine. Eventually, Strigeus joined forces with Vincent Hamm, who was also interested in building a SCUMM system player. Strigeus and Hamm work on making the platform compatible with both Monkey Island 2 and Indian Jones as the Fate of Atlantis, respectively.

The software started imprinting on the computer science community when it earned a feature on ‘Slashdot’. It led to numerous other developers hopping on board the project. Around November 2001, there was a discussion about changing the name of SCUMM to remove the ‘SCUMM’ part of the name.

Before leaving the project's development, Strigeus implemented support for a sound software called iMUSE, used by LucasArts games. At this point, the project had a team of developers and was taken over by James Brown, who adjusted the project from C to C++ mixed with a GUI.

ScummVM eventually gained enough publicity to lead LucasArts to send the creators a cease & desist letter, which was unexpected since many developers believed that ScummVM would have no legal troubles with the addition of use. Despite the original disagreement between the two companies, they came to a legal agreement allowing for the continuation of development.

The growing demand for Sierra Online games requires the Adventure game interpreter (AGI) and Sierra's Creative Interpreter (SCI). AGI was added in 2006, eventually leading to competition between ScummVM and another project called FreeSCI, which was working to reverse engineer SCI. FreeSCi eventually began to dwindle in popularity, leading to an independent developer integrating FreeSCI into ScummVM and eventually, the two projects agreed to merge. The version, including the support for SCI, was released in 2010.

In 2017, ScummVM saw a new project lead in Eugene Saandulenko, and the project began to slow down its addition of new games. Most support was added for games that port into ScummVM's architecture or that an individual developer independently decided to support due to the difficulty of adding new games. With ScummVM specifically for 2D games at the time, they eventually merged with a sister project called ResidualVM, which was created for 3D games. In August 2021, the project's original developers completed support for Macromedia Director, who was used in games back in the 1990s.

The project is open source, and other developers are encouraged to work on it. Developers can now download the source code through GitHub, compile it, and develop new features, which will be inspected and eventually added to the project if they meet the criteria.

Control and Data flow

The control flow of ScummVM is straightforward. Since the architecture is multilayered, it flows from one layer down to the next smoothly. The four layers were the application layer (GUI), core layer (essential programs), abstraction layer (subsystems), and the hardware layer.

System:

1. User interactions are captured at the application layer (GUI).
2. Input is forwarded to the core layer and processed by the Game Engine Manager.
3. The Game Engine interprets user actions via the Game Logic Interpreter.
4. Commands are sent to Graphics Subsystem and Audio Subsystem to render visuals and play sounds.
5. Feedback provided to the user through updated game state (visuals/audio).

Steps in Control Flow:

1. User selects a game from the GUI.
2. The Game Engine Manager in the Core Layer initializes the appropriate game engine.
3. Game Logic Interpreter runs scripts, interacting with subsystems (graphics, audio).
4. The game's visual and auditory feedback is sent to the user.

The data flow of ScummVM is less direct and moves through quite a few locations.

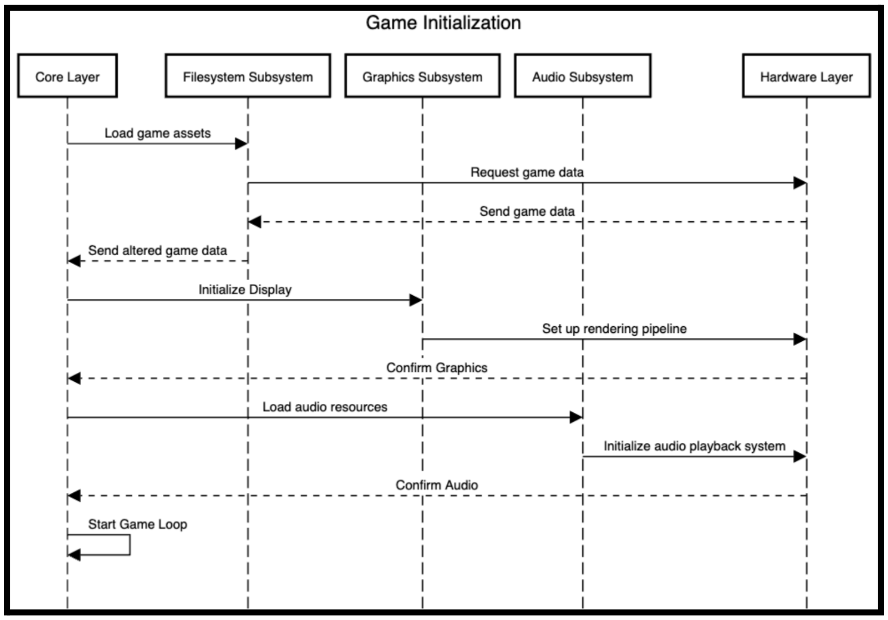
System:

- Game data (e.g., saved states) managed by the File System Subsystem.
- User input captured by the Input Subsystem and processed by the Core Layer.
- Data flows between Game Engine, Graphics, and Audio Subsystems to update game state.
- Output (visuals and sound) sent back to the user via the Application Layer.

Steps in Data Flow:

1. Game data loaded from the File System Subsystem to initialize game state.
2. User inputs captured by Input Subsystem, processed by the Game Engine.
3. Commands sent to Graphics and Audio Subsystems to update visuals/sounds.
4. Updated game state presented back to the user.

Use Case 1: Game Initialization example showing control and data flow



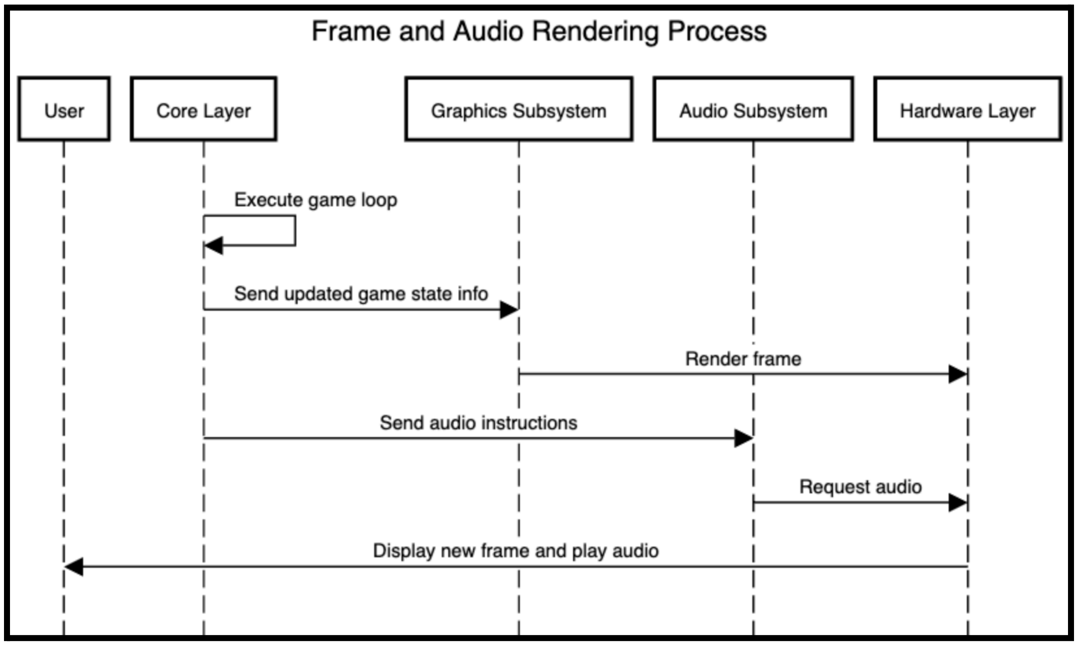
Concurrency

Although not major, there is presence of concurrency in ScummVM's architecture. Concurrency was achieved in the system through usages like asynchronous task handling and multithreading.

In terms of asynchronous task handling, the audio and graphics rendering occur simultaneously (asynchronously). The Audio Subsystem continues to play sounds while Graphics Subsystem renders frames. Another use of asynchronous task handling is to ensure smooth transitions between game states without lag. The key asynchronous processes are audio playback where music and sound effects run concurrently with gameplay logic and graphics rendering where independent rendering of visuals ensures smooth screen updates.

In terms of multithreading, it is used to optimize performance by separating specific tasks (e.g., input/output management) into threads. This is so as to prevent delays in the main game loop by offloading input handling or rendering to separate threads. Examples of usage are present in input handling and I/O management. With input handling, user input is processed on a separate thread to maintain responsiveness of the game. When it comes to I/O management, tasks like saving game states can run in the background without blocking gameplay.

Use Case 2: Frame and Audio Rendering Process example showing data flow and asynchronous processes



Responsibilities Among Developers

The project started with Vincent Hamm, who had done relatively deep research into developing a Scumm system player. Eventually, he was joined by Ludvig Strigeus, who was interested in making his own adventure game. He looked into the architecture of a game called 'Monkey Island 2' and tried to make it work with his Linux. Initially, the project division came from dividing the work into working on making the engine work for two different games, 'Monkey Island 2' for Strigeus and 'Indiana Jones and the Fate of Atlantis' for Hamm. They were uncoordinated but eventually completed the implementation for both games. Overall, the original division of responsibilities was minimal and incohesive as the two developers worked separately on implementing their desired game.

Later in the project, when ScummVM was able to build a team of developers, the project was slowly able to be broken down into subsections. Despite some subsections, if a developer is passionate about bringing a specific game into the software, they can go off on a tangent and reverse engineer the game. The basic structure comprises engine authors, backend authors, and website maintainers, defined on their developer's central page.

Further, the engine authors are divided into more minor, specific responsibilities. They provide new engines, and developers are tasked with adding new ones to ScummVM. They are enhancing ScummVM with advanced features, portability, and reverse engineering. This category of developers is tasked with adding features, optimizing the engine, and adding new software to ScummVM. They are taking up most of the team and most of the tasks. These responsibilities are also available to open-source developers interested in adding new features to the project.

The backend developers are primarily tasked with getting the actual Scumm platform to be ported to other devices or minimizing the backend using loadable modules for compatibility with smaller backends or when no operating system is available.

Lastly, the website maintainers are responsible for updating the website. This part of the project is relatively minimal compared to the other tasks but also allows for open-source editing, where developers can edit files and update data for the website.

Lessons Learned

Exploring the architecture of ScummVM has exemplified the structure of a real-world software application. It strengthened our understanding and provided proper examples of architectural styles and concepts. Seeing how control and data flow through a software project and learning how a multi-layered, component-based system works in action has helped us put together knowledge from course content with real-life examples and applications. As we work on adding new features or implementing our own systems in the future, we will have a better grasp of how these architectures function and how data flows through them, which will help us in making informed architectural decisions.

Conclusions

ScummVM is a testament to the power of community-driven open-source projects and the importance of software preservation. Through its innovative architectural design, ScummVM has successfully made it possible for modern audiences to experience classic adventure games across multiple platforms. The layered approach taken in its development—consisting of the application, core, subsystem, and hardware layers—ensures that the system is both highly modular and adaptable, allowing for easy integration of new game engines and support for different operating systems. This modularity not only extends the lifespan of classic games but also provides a foundation for future developers to contribute to the platform with minimal disruption to its existing functionality. The evolution of ScummVM, from a small project by two developers into a widely supported open-source initiative, showcases how collaboration can drive innovation and overcome technical challenges. Key milestones in its development, such as the merging of the FreeSCI project and the inclusion of support for Sierra Online’s game engines, highlight ScummVM's ability to expand its capabilities over time. Additionally, the project's legal resolution with LucasArts demonstrates how open-source projects can navigate intellectual property challenges while fostering cooperation between developers and content creators. Studying ScummVM has provided valuable lessons in software architecture, particularly in managing control and data flow across a complex, multi-layered system. It also illustrates how concurrency and multithreading can be applied to optimize performance without sacrificing responsiveness. For future developers, ScummVM serves as both an educational tool and a model for designing robust, portable software systems. Ultimately, ScummVM's success is not just in its technical achievements, but in its role in preserving the cultural and historical significance of video games for future generations.

Glossary

API - A set of rules or protocols that enables software applications to communicate with each other to exchange data, features and functionality.

Application Layer - An abstraction layer that specifies the shared communication protocols and interface methods used by hosts in a communications network.

Asynchronous - Events or tasks independent of the main program flow.

Concurrency - The ability of different parts or units of a program to be executed simultaneously, without affecting the outcome.

Architecture - The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

Architecture Style - A reusable set of design decisions and constraints that are applied to an architecture to induce chosen desirable qualities.

Backend - The part of a computer system or application that is not directly accessed by the user, typically responsible for storing and manipulating data.

Core Layer - Involves essential programs that support the management and effective running of the system.

Game Engine - A framework designed specifically for the development of video games.

GUI (Graphical User Interface) - A digital interface in which a user interacts with graphical components such as icons, buttons, and menus.

Hardware Layer - The operating system and hardware the program runs on.

Layered Architecture - A design approach/architecture style where the application is divided into separate layers that can be managed and maintained independently.

Linux - A Unix-like, open source and community-developed operating system (OS).

Multithreading - Allows multiple concurrent tasks to be performed within a single process. Two or more instruction threads can execute independently while sharing the same process resources.

Operating System - A program that manages a computer's resources, especially the allocation of those resources among other programs.

Open Source - Programs with code that is designed to be publicly accessible. Public access to view, modify, and distribute the code as you see fit.

Rendering - The process of generating an image from a 2D or 3D model by means of a computer program.

Subsystems Layer - A layer that contains many platform-based subsystems that take care of graphics, audio, the filesystem, and user input/output.

Slashdot - A social news website popular in the early 2010s focusing on science, technology, and politics.

iMUSE - An interactive music system used to synchronize music with the visual action in a video game.

References

ScummVM. (n.d.). *Features*. Wikipedia. Retrieved October 11, 2024, from https://en.wikipedia.org/wiki/ScummVM#Features

ScummVM. (n.d.). *ScummVM*. Retrieved October 11, 2024, from https://www.scummvm.org/

ScummVM. (n.d.). *ScummVM GitHub repository*. GitHub. Retrieved October 11, 2024, from https://github.com/scummvm/scummvm

ScummVM. (n.d.). *Developer central*. ScummVM Wiki. Retrieved October 11, 2024, from https://wiki.scummvm.org/index.php?title=Developer_Central

ScummVM. (2024). *Documentation v2.8.0*. Retrieved October 11, 2024, from https://docs.scummvm.org/en/v2.8.0/