Optimizing workflow for virtual environments using Vulkan Model Viewer and Exporter

The University of Roehampton
Department of Arts and Digital Industries
London, United Kingdom

Author
Mr. Zakariya Oulhadj

Supervisors
Dr. Charles Clarke
Alex Collins

Submitted in partial fulfillment of the requirements for the degree of
Bachelors of Science in Computer Science

May 12, 2023

This report is dedicated to my family, friends and lecturers who have supported me throughout my degree

Abstract

Computer graphics is a rapidly growing field that is vital in many industries that rely on digital graphics including scientific research, simulations, education and training, entertainment and more. The flexibility and increase of computing resources provide real-world benefits in ways not previously observed before the use of computer imagery.

This report presents my final year project an application built using the latest technologies as a learning tool for 3D model rendering. Its goal is to be easy to use, performant and provide a collection of tools for graphics manipulation that users can take advantage of when designing and building virtual environments. It's intended to serve as an important application that bridges the gap between inexperienced users and industry-leading rendering engines.

Declaration

I hereby certify that this report constitutes my work, that where the language of others is used, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions, or writings of others. I declare that this report describes the original work that has not been previously presented for the award of any other degree or institution.

May 12, 2023	z.oulhadj
Date	Signature

Contents

Glossary

Vulkan Low-level GPU application programmable interface.

Acronyms

API Application Programmable Interface.

CPU Central Processing Unit.

GDPR General Data Protection Regulation.

GPU Graphics Processing Unit.

IDE Integrated Development Environment.

LOD Level of Detail.

MSVC Microsoft Visual Compiler.

OS Operating System.

STL C++ Standard Template Library.

SVN Apache Subversion.

UI User Interface.

VCS Version Control System.

VMVE Vulkan Model Viewer and Exporter.

VRAM Video Random Access Memory.

1 Introduction

Computer graphics is a vast area within the field of Computer Science. Since its conception in the early 1960s, it has been used for a wide range of purposes including gaming, film, scientific research, education, architecture, engineering, medicine and more recently within vehicles. This is a testament to how versatile this field has become and consists of numerous benefits that computer graphics provide to the real world.

1.1 Project Problem

Industrial graphics software today are large and highly technical applications that are the culmination of decades worth of work. Examples include Unreal Engine[1], Unity[2] and RenderMan[3]. These applications provide users with innovative technologies such as ray tracing, real-time global illumination, subsurface scattering, geometry compression, streaming and much more that are designed for use in professional environments which results in stringent requirements and complex feature sets.

This results in the adoption of the software being a barrier to entry for graphics developers, artists and businesses which inadvertently introduces a wide range of issues such as requiring training for employees, additional costs for businesses and a decrease in productivity. Other issues include performance, compatibility and storage when using these complex applications.

To address these issues, this report presents my final year project, Vulkan Model Viewer and Exporter (VMVE). A standalone application developed in the domain of computer graphics that provides an easy-to-use, efficient platform for 3D graphics rendering and asset encryption. It serves as an important tool and as a stepping stone that aims to bridge to gap between new users and industry-leading rendering engines.

To achieve this, VMVE provides a small but specific subset of features that are designed from the ground up to be easy to use and understand for rapid graphics prototyping such as constructing virtual environments, simulating the effect of light on 3D geometry and securing digital assets through the use of encryption.

1.2 Stakeholders

Stakeholders are individuals or groups who have an interest or ”stake” in VMVE which can include decisions made throughout the development or the end product. Each stakeholder group may have different goals, priorities, and/or expectations, which can sometimes lead to conflicts and challenges in decision-making and implementation. It is important to identify the different groups to ensure their requirements and concerns are being taken into account so that informed decisions can be made.

1.2.1 Students and trainees

Students and trainees are the key stakeholder groups for the project which the application is designed to accommodate by providing the necessary tools and functionality. This includes providing a suite of easy-to-use tools for graphics rendering that students and trainees could use to learn about basic rendering concepts and to use this knowledge to construct virtual environments for graphics prototyping and rendering.

1.2.2 Businesses

The other type of stakeholder is employers who will want to train new potential employees in the field of computer graphics. The goal is for businesses to use VMVE as a training tool to lower the learning curve and reduce the barrier to entry within the graphics industry.

1.3 Requirements

The design, implementation and evaluation of the project rely on a set of aims and objectives to be fully defined. The first step toward obtaining these requirements requires a "requirements gathering" stage. The purpose of this is to outline and better understand the requirements for each of the stakeholders mentioned in section 1.2.

1.3.1 Aims

VMVE has several aims which outline the general goals that the application wishes to achieve throughout this project. These aims go on to define concrete objectives that provide a clear and concise set of instructions required for the design and implementation stages.

1. **Virtual environment rendering** The first aim of VMVE is to provide an "easy-to-use" platform for creating virtual environments. This means that the application should reduce the learning curve so that users with little to no prior experience with 3D rendering software and

the underlying technology such as students and trainees can use the application to achieve their desired goals.

2. **Lower system requirements** Many industry-leading applications are complex applications that have significant hardware requirements to run. VMVE will aim to address this by focusing on the efficiency of the application and only providing the necessary tools needed which are not computationally expensive.

3. **3D asset encryption** The last of the three main aims is for the application to include a custom model format that makes use of encryption to safely secure a user's digital assets. The motivation for this functionality formed as a result of the increase in theft regarding digital assets. A recent example is the data leak that affected Rockstar Studios and their upcoming game (Grand Theft Auto 6) which saw assets, source code and documentation released to the public [4].

1.3.2 Objectives

Fulfilling these aims requires a set of objectives that need to be implemented to achieve the aims outlined in section 1.3.1. It provides a more concrete definition of what the project must achieve.

1. **Virtual environment rendering** The project will be built to run as a desktop application that renders a user interface editor as well as a virtual environment. The User Interface (UI) must be designed intuitively to reduce the amount of friction between the application and the user's goal. This means common related tasks and functionality must be grouped. Furthermore, the language used throughout the user interface must reduce if not eliminate jargon. Doing so ensures the application is easier to understand.

2. **Lower system requirements** This can be achieved through the use of efficient technologies, algorithms and overall structure related to rendering that will allow for assets to be displayed using as few hardware resources as possible. For example, by using the latest rendering Application Programmable Interfaces (APIs) and packing/compressing crucial data. Efficiency requirements vary based on the component in question which includes the Central Processing Unit (CPU), Graphics Processing Unit (GPU), memory and storage.

3. **3D asset encryption** VMVE would be able to provide a platform to

encrypt these assets and only unlock them using a special key that only the user will know. This ensures that in the event of digital assets begin obtained by third parties, they cannot be used. This requires several key elements to work in tandem to achieve this objective.

The raw model must be able to be encrypted using different types of encryption algorithms. In addition to this, the application will need to include a parser that can read the internal structure of the encrypted file format and successfully load the model into the application.

With these aims and objectives fully defined, the project proposal document was signed off as part of the requirements for milestone 2 which can be seen in appendix section B.

1.4 Considerations

Important deliberations must be taken into account before commencing work on the application which includes legal and ethical considerations.

1.4.1 Legal

The legality of VMVE is one of the most important considerations that must be taken into account. As the application is intended to be distributed to users around the world, local laws and regulations must be adhered to. As a relatively basic application that serves a specific user base, there is a small set of features and functionality that require clarification.

Encryption Firstly, VMVE will include the ability to secure critical assets such as 3D models. This will be achieved using encryption by making use of a secret key and initialisation vector. This information is not stored and is only generated once for a user to save to encrypt and decrypt a model.

Asset modification Secondly, VMVE will not modify any of the raw data being imported and only change the visual appearance based on several factors including rendering settings, lighting, translation, rotation and scaling.

Personal information Finally, the application does not store or process any personal information and also does not contain any networking functionality. This means that all data remains local to the application and the user's device. Therefore, General Data Protection Regulation (GDPR) will not need to be taken into account. In the future, however, if the use of data and/or information changes then this will be revisited to ensure that all laws are adhered to.

The source code for the project will not be publicly available throughout the development. However, upon completion of the project, this may change and thus, will be licensed under the MIT license[5] which states that the source code can be used without restriction allowing for it to be copied, modified as well as distributed.

1.4.2 Ethical

One of the main ethical considerations is the computation that the application will perform internally. The technical implementation details are not accessible to users as the source code is not open source. Therefore, individuals do not know exactly what the software is doing during runtime.

To ensure that VMVE remains ethical, the application will not perform any additional computations that are not strictly required for 3D rendering. Furthermore, the application will not store or send any data as this is not required and would fall outside the scope of its initial aims and objectives.

Another ethical consideration is the use of deceptive design also known as "Dark patterns" that refers to designs/tricks that manipulate users into performing certain actions without their informed consent. VMVE will make use of a user interface, however, during the design and implementation stages extra effort will be spent on ensuring that no such design pattern is introduced. This will greatly benefit users and their overall user experience.

1.5 Background

This project is ambitious and challenging that requires a strong understanding of computer graphics, programming, and software architecture. It is a large undertaking that uses the skills and knowledge gained through the BSc (Bachelor of Science) computer science program from modules such as software development (1, 2, 3), algorithms, data science, cybersecurity and mathematics. Not only does it showcase technical ability but also highlights critical thinking, problem-solving, decision-making and time management skills. As a result, they will be highly valued in the professional industry and necessary for success in the field of computer science and therefore, is a suitable choice for a final-year project.

1.6 Overview

This report is structured such that it presents each aspect of the development in chronological order. There are a total of four key sections in this report:

Technology review (2) Covers the different technologies that could be potentially used throughout the development and outlines the advantages and disadvantages of each. This will lay the foundation for the design section.

Design (3) Focuses on the concept of the project, the design of the requirements obtained during the requirements-gathering stage followed by the design of the application such as the tools going to be used, user interface and engine architecture.

Implementation (4) Takes the proposed designs and consist of the development of the application as well as presenting the technical implementation details and decisions.

Conclusion (5) Reflects on the project to ensure that the application has met the aims and objectives originally set out and discusses the specific features that will need to be implemented going forward.

2 Technology Review

Prior to commencing work, a technological review must be undertaken that analyses and compares different types of potential tools that could be used. Throughout this comparison, the choice of potential tools is heavily influenced by the aims and objectives outlined in sections 1.3.1 and 1.3.2 respectively.

2.1 Version Control

A version control system is a software tool that helps developers manage and track changes to source code or other documents over time and will be used in this project. For a more in-depth explanation of what a Version Control System (VCS) is, visit the appendix section D. There are two main types of systems, which are SVN and Git.

2.1.1 Apache Subversion

Apache Subversion (SVN) was originally developed in 2000 by Apache and is a centralised and open-source VCS which means that the repository is stored on a remote server which developers all connect to. In the past, this was far more popular amongst developers.

2.1.2 Git

On the other hand, Git is the most widely-used, free and open-source VCS developed in 2005. It is a distributed system meaning that developers have copies of a repository that they work off. Git is generally considered to be faster and more efficient than SVN as a result of its distributed structure. Furthermore, Git is more flexible regarding how it can be installed such as onto a server manually or by using existing platforms that are built on top of Git. A few examples include GitHub, GitLab and Bitbucket.

According to the StackOverflow Developer Survey, in 2022 Git was the most widely used VCS with an adoption percentage of 93.9% as seen in figure 1. This shows how popular Git is compared to other similar systems[6].

Git	SVN
93.9%	5.2%

Table 1: Version control usage comparison

2.2 Development Environment

The project will be developed using a dedicated development environment which is a set of tools that provide helpful features. These features can include debugging, formatting, source code analyses, profiling and much more.

Microsoft Visual Studio 2022 This application is a fully-fledged Integrated Development Environment (IDE) that contains many different features that are dedicated to application development. In addition to the application, it comes with the Microsoft Visual Compiler (MSVC) which is Microsoft's own C/C++ compiler and linker. This is a program that parses source code and generates assembly instructions that the CPU will be able to understand and therefore, execute. The IDE together with MSVC create a solid ecosystem for development. The IDE also provides debugging functionality that will be used extensively to fix crashes, and bugs and generally ensure that the application runs as expected[7].

The main downside to this option, however, is that it is only supported on the Windows operating system. Therefore, it's not available on macOS or Linux meaning that if the application will be cross-platform then other tools will be required.

Microsoft Visual Studio Code Is a light and open-source text editor built using Electron which is flexible and has support for many extensions that include debugging, code analysis and many other ide-like features. As it's built on top of Electron, it's cross-platform and can run on different operating systems.

However, the disadvantage of using a tool such as this is that it's not a complete package as it relies on external tools to provide more advanced functionality. Therefore, issues regarding compatibility, functionality and performance may arise which is not ideal when developing a complex application.

JetBrains Clion Developed by JetBrains it's primarily a C/C++ IDE for use with the CMake build system. One of the major downsides to this application is performance. CLion was built using the Java programming language which runs on JVM (Java Virtual Machine). Parsing complex source code requires significant performance requirements and if the IDE starts to slow down then this makes the developer experience worse.

2.3 Programming Language

As a desktop application, there are several options for which programming language should be chosen with each one having its advantages and disadvantages. Due to the nature of the application and its performance requirements, the language must allow VMVE to be compiled as a binary file. Furthermore, direct memory access and manipulation are required. Three main candidates have been proposed.

C Developed in 1972, it is one of the older programming languages and closest to the hardware. It is a simple language in terms of the functionality that it provides. This, however, makes development using C more work as various basic functionality must be implemented from the group up and also could introduce bugs and performance issues if not done correctly.

C++ Developed on top of C, it introduces a wide range of features such as classes, inheritance, type safety and the standard template library. As a result of this, the language is much more popular in the domain of computer graphics. Furthermore, this is the language that I have the most experience with.

Rust Is the newest language out of the three and has been developed from the ground up with memory safety in mind. One of the key drawbacks to using Rust is the lack of adoption and thus, many external libraries or features may not be available as of yet.

2.4 Rendering API

A rendering API is a set of instructions that acts as an intermediary and allows for an application to make use of a systems GPU which is also known as hardware acceleration. There are several different rendering APIs that exist including OpenGL, Vulkan, Direct3D 11 and Direct3D 12. OpenGL and Direct3D 11 are previous generations APIs whereas Vulkan and Direct3D 12 are the latest technologies. Each of these has its advantages and disadvantages. Figure 1 shows a table that compares each of these with specific types of functionality between the APIs.

	Cross Platform	Finer control	Multithreading
OpenGL	✓	✗	✗
Vulkan	✓	✓	✓
Microsoft DirectX 11	✗	✗	✗
DirectX 12	✗	✓	✓

Figure 1: Rendering API comparison

It shows that the Direct3D APIs are not cross-platform and are only created to target Microsoft-backed operating systems such as Windows and Xbox. On the other hand, OpenGL and Vulkan are designed according to their specifications to run on different platforms.

2.5 External Libraries

The use of external libraries is vital in allowing VMVE to be implemented within the short timeframe allocated. Libraries provide functionality that the application can easily incorporate without the need to implement the same solution from scratch. The table shown in figure 2 lists the different libraries based on the feature.

Feature	Libraries
Model Loading	Assimp, tinyGLTF, tinyOBJ
User Interface	ImGui, QT
Encryption	Crypto++
Serialisation	Cereal, Boost

Table 2: Library comparison

2.6 Task Management

Task management refers to the use of tools to aid in managing tasks or issues. Various tools are designed for this purpose however, there are two that are the most well-known which are Trello and GitHub Projects also known as GitHub issues.

The Trello platform was developed before GitHub Projects. It also provides far more features that can be used to better manage the project and the various tasks that will be created throughout the development process. On the other hand, GitHub Projects is much more integrated into the GitHub ecosystem making it far easier to incorporate GitHub issues as tasks and also for the project repository as a whole.

3 Design

The design of the project consists of selecting the specific tools discussed in the technology review 2 and the architectural design decisions for both the source code and front end of the application.

3.1 Tools

3.1.1 Git/GitHub

The VCS that was chosen for this project was Git including the GitHub platform[8]. It's the most popular version control system as mentioned and is the platform that I am most familiar with. Additionally, the performance benefits that it provides over similar systems are ideal for this type of project. Appendix section E provides additional information regarding the technical details of how the repository is hosted on GitHub.

3.1.2 GitHub Projects

After weighing the pros and cons of Trello and GitHub Projects, it was decided to use GitHub for task management and tracking. To track outstanding tasks, "posts" are created on GitHub that includes the task's priority, current progress, and expected deadline. Figure 2 provides a preview of the Kanban board, which consists of three main columns: backlog, in progress, and done, used to categorise issues. When a new feature idea or application bug is discovered, a GitHub issue is created and moved to the "Backlog" board to document the task within a specific timeframe. When ready to address an issue, it is moved to the "In Progress" board. Throughout the issue resolution process, comments are added to the issue to capture key discussion points for documentation purposes and reference in commits or merges. Finally, when the task is complete, the GitHub commit containing the specific fix is referenced in the issue and marked as complete by moving it to the "Done" board, thereby closing the GitHub issue.

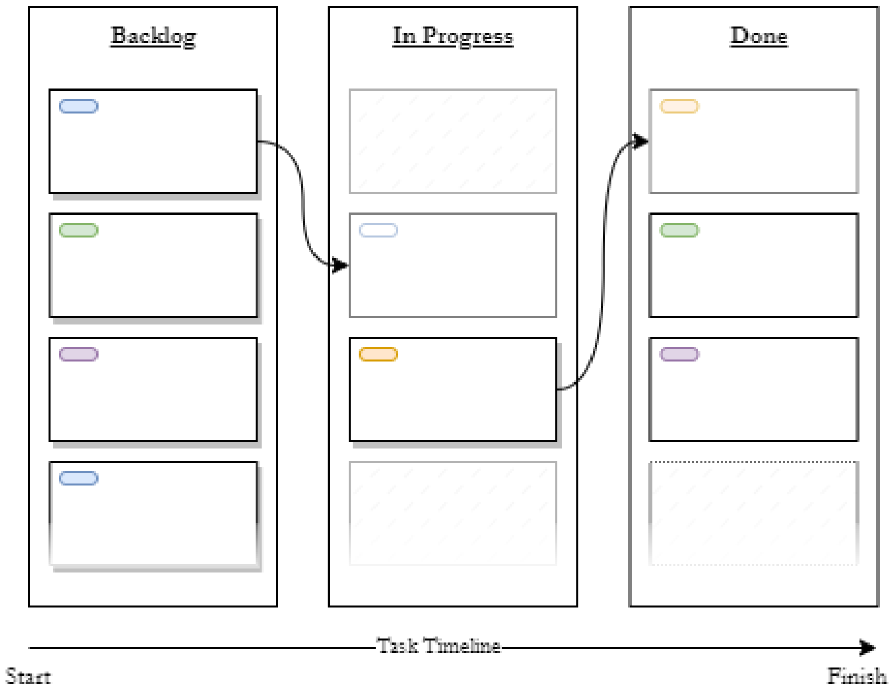


Figure 2: Kanban design

Overall, this organisation and management of tasks allow the project to meet key sprints and deadlines and allow for issues to be resolved within a suitable timeframe.

3.1.3 Microsoft Visual Studio 2022

Out of the proposed development environments, Microsoft Visual Studio 2022 was chosen. This is primarily because of the vast amount of features and functionality that it provides compared to the other tools. Additionally, in conjunction with Visual Studio, Visual Studio Assist X will be used to further improve the ease of development [9]. This is an extension that provides many useful features that the base IDE does not provide such as reliable symbol renaming, file symbol outlining and quick file searching.

3.1.4 RenderDoc

As well as using software for debugging on the CPU, there also exists software that allows for debugging the GPU. More specifically, these tools allow for analysing per-frame data as well as detailed frame synchronisation metrics. RenderDoc developed by Baldur Karlsson can be used to inspect different rendering APIs including individual frames and its entire state [10]. This ensures that you can debug specific GPU related bugs such as model data, textures, rendering commands etc.

3.1.5 AMD Radeon GPU Profiler

Similarly to RenderDoc, AMD has its own GPU profiling tool called "AMD Radeon GPU Profiler" [11]. This tool is only available for AMD-supported GPUs which is ideal as the hardware going to be used throughout development meets this requirement. The tool also analyses per-frame data, however, contains more technical and detailed metrics such as command buffers, fences and semaphores between frames and also validates API usage.

3.1.6 C++ 23

C++ 23 was chosen as the programming language for this project due to two primary reasons over C or Rust. Firstly, as the language I am most experienced with, it significantly facilitates the implementation stage. Secondly, the C++ Standard Template Library (STL) was a key factor. The STL provides prebuilt data structures, containers, and algorithms such as "std::vector", "std::string" and "std::find", saving time by eliminating the need to develop a custom solution within the limited timeframe of the project. Other language features, including function overloading, templates, compile-time expressions, direct memory access, and generally faster performance compared to other languages, confirmed the suitability of the chosen language.

3.1.7 Vulkan

Having evaluated the potential use of the different APIs covered in section 2.4, it was concluded that Vulkan would be the most suitable rendering API for VMVE. Firstly, Vulkan is extremely verbose as it exposes the technicalities related to the GPU and requires the application to implement the basic functionality as opposed to the underlying driver provided by major GPU vendors. This reduces the complexity of the display driver and can potentially increase performance as it no longer needs to make assumptions about the application and the commands being issued as the entire state of the rendering pipeline is predefined. Vulkan was also chosen as the technology allows one to learn and understand the complex inner workings of graphics applications and pipelines as well as GPU computation. Furthermore, Figure 1 shows that the API provides the most amount of functionality including the ability to run on different operating systems, finer rendering control and support for multithreading.

Figure 3: Vulkan logo [12]

3.2 Libraries

The application will make use of different external libraries to help aid and speed up development as a result of the short timeframe. These libraries vary in terms of functionality and will provide a solid foundation on which the application can be built on top of.

3.2.1 ImGui v1.89

Users will need a way of interacting with VMVE and the 3D environment at runtime. This will be achieved through the use of a user interface in which the user can directly manipulate the application. As mentioned earlier, VMVE is an application that uses the GPU for rendering and therefore, the UI will have to interact with the GPU. To reduce the development time of this particular aspect of the application, the decision was made to make use of an existing library. This library is called Dear ImGui [13]. This is an immediate-mode user interface library that provides an API to render UI elements. It's easy to use and popular which is what makes it an ideal choice compared to the QT library.

3.2.2 Crypto++ v8.7

VMVE will include its own model file format. This is a custom format that will be encrypted as standard. Implementing encryption is a very complex area that can be considered an entire project on its own. Instead, the project will make use of a well-known encryption library known as Crypto++[14] and the application will use version 8.7. This is a C++ library that provides many algorithms including AES, Diffie-Hellman, GCM, RSA and much more.

3.2.3 Cereal v1.3.2

In addition to encrypting the model data, the custom file format will need to store additional information such as application version and encryption type. Therefore, VMVE must be able to serialise the data when saving to disk as well as deserialise when loading a ".vmve" model. To achieve this, the cereal C++ library will be used[15]. This library allows for fast and efficient data serialisation into different formats such as compact binary encodings, XML or JSON.

3.2.4 Summary

Several other external libraries were used which are shown in figure 3.

Assimp (Model Loading)	v5.2.5
GLM (Mathematics)	v0.9.9.8
STB (Image Loading)	v2.27
VMA (Vulkan Memory Allocator)	v3.1.0
VOLK (Vulkan Function Loader)	v1.3215
ENTT (Entity Component System)	v3.11

Table 3: Additional library versions

3.3 Branding

The branding which is the look of the application was an important factor during the design as it's marketed towards end users and can directly influence the user experience. Additionally, being recognisable results in an increase in brand recognition that may also increase user traffic as found in a 2013 paper titled *The role of logos in building brand awareness and performance: implications for entrepreneurs*[16].

3.3.1 Project Name

VMVE stands for "Vulkan Model Viewer and Exporter". The name is a combination of two parts. The first is "Vulkan" the other is "Model Viewer and Exporter". The rendering API used to render 3D digital assets is Vulkan and therefore, is a key part of the application. The second part is because the application will include functionality such as importing and exporting models into a custom file format as mentioned in section 3.2.2. The combination of these two allows a user to have a basic understanding of what the application does without having used it before which is also known as a descriptive logo.

3.3.2 Logo

There are two versions of the VMVE logo that was designed and intended to be used in different scenarios. The first is the complete logo as seen below in figure 4 which is referred to as the "large logo" and contains the icon and the name of the application.

Figure 4: VMVE Large Logo

This is to be used on the VMVE website as well as documentation pages providing consistent branding across different mediums.

The second version is designed to be minimal and therefore, only consists of the icon. This is intended to be mainly used throughout the application and in locations where there is a minimal amount of space such as the Windows taskbar.

Figure 5: VMVE Small Logo

3.4 Project Architecture

3.4.1 Development Life Cycle

Agile is the development life cycle of choice throughout the project's development. This is a software development methodology which is an iterative process that occurs for key stages of a project. This allows for continuous designing, implementation and evaluation that ensures requirements are being adhered to.

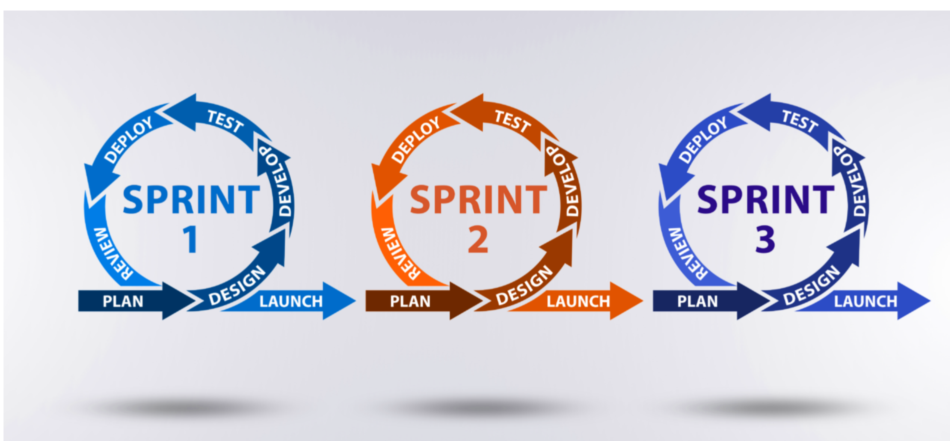


Figure 6: Agile Development Life Cycle[17]

3.4.2 Version Control

As the only developer for this final-year project, the design and architecture of the VCS is kept simple as possible while still providing the core requirements. Figure 7 shows the proposed VCS architecture and includes two branches named "main" and "develop". Develop branch is the primary branch used throughout the implementation stages. In other words, when changes occur throughout the source code, all commits will be published to the development branch.

"Main" is known as a stable branch that is only updated for each official release. This would occur for each project milestone such that for "Sprint 1" a pull request will be made from develop to main and this will be tagged as v0.0.1 and likewise "Sprint 2" will be tagged as v0.0.2.

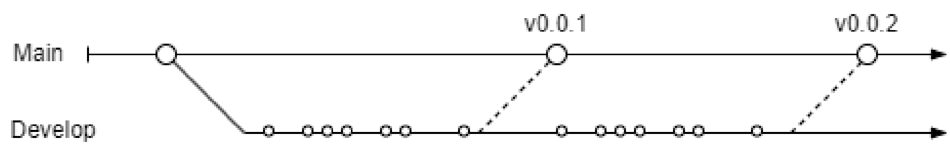


Figure 7: Git branch design

Overall, this architecture ensures that as the sole developer of the project, changes can be made consistently while ensuring that users are provided with stable versions of the application for increased stability and usability.

3.4.3 Source Code

The source code for the application and overall architecture are to adhere to the C++ core guidelines[18] to ensure that the project follows best practices. Section NL.10 of the core guidelines recommends using the snake case style as it follows the standard libraries naming convention. Since the project has no existing code base and therefore, no existing convention to follow, the project will make use of the underscore style for types, functions and variables.

Additionally, section NL.17 states that the use of the "K&R" indentation style[19] should be used as it preserves vertical space whilst maintaining readability. In other words, reducing vertical line height for code blocks such as "if, else, while and for" allowing for more lines of code to be visible at any given point whilst allowing for a more distinct separation of structures and functions.

The combination of these two specific conventions regarding source code style can be seen in figure 8.

```
struct foo
{
    int a;
};

void bar(int a)
{
    if (a) {
        function_call();
    } else {
        function_call();
    }
}
```

Figure 8: Source code convention

VMVE is also developed in a functional style and by using "structs" rather than object-oriented with "classes". This is primarily related to the ease of use in terms of development since data can be more easily accessed and manipulated. Furthermore, this will eliminate the need for getter and setter functions that needlessly take up source code space and is redundant when member variables are accessible.

3.4.4 Architecture

Architecturally, VMVE is a combination of two projects in one. The "Engine" project also known as the core contains the fundamental implementation details. This includes the Window, Renderer, UI and Audio. This project will be distributed as a library file (.lib). Making this distinction between the core and VMVE allows for the same sets of tools and technologies to be used in the future for other similar projects. For this project, VMVE includes the "Engine" project by importing the library (.lib file). A high-level overview of the project architecture can be seen in figure 9.

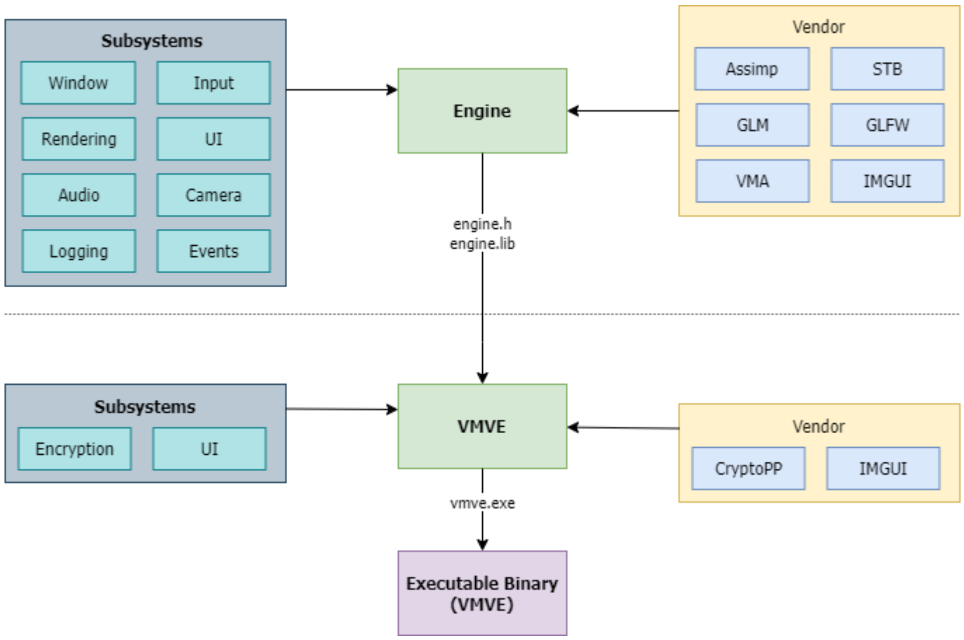


Figure 9: Project Architecture

For a complete list of the directories that are part of the project architecture, visit appendix section C

3.4.5 Renderer Architecture

As mentioned in section 3.1.7. Vulkan will be the rendering API of choice for VMVE. The API is extremely verbose giving the programmer the flexibility to control every aspect of the GPU. When interacting with Vulkan throughout the engine a certain degree of encapsulation is necessary to reduce the amount of effort required to implement functionality as a result of Vulkans verbose API.

Therefore, the renderer must be designed so that flexibility is maintained whilst simplifying the API. Figure 10 shows the proposed renderer architecture which makes distinctive boundaries and encapsulates key sub-systems.

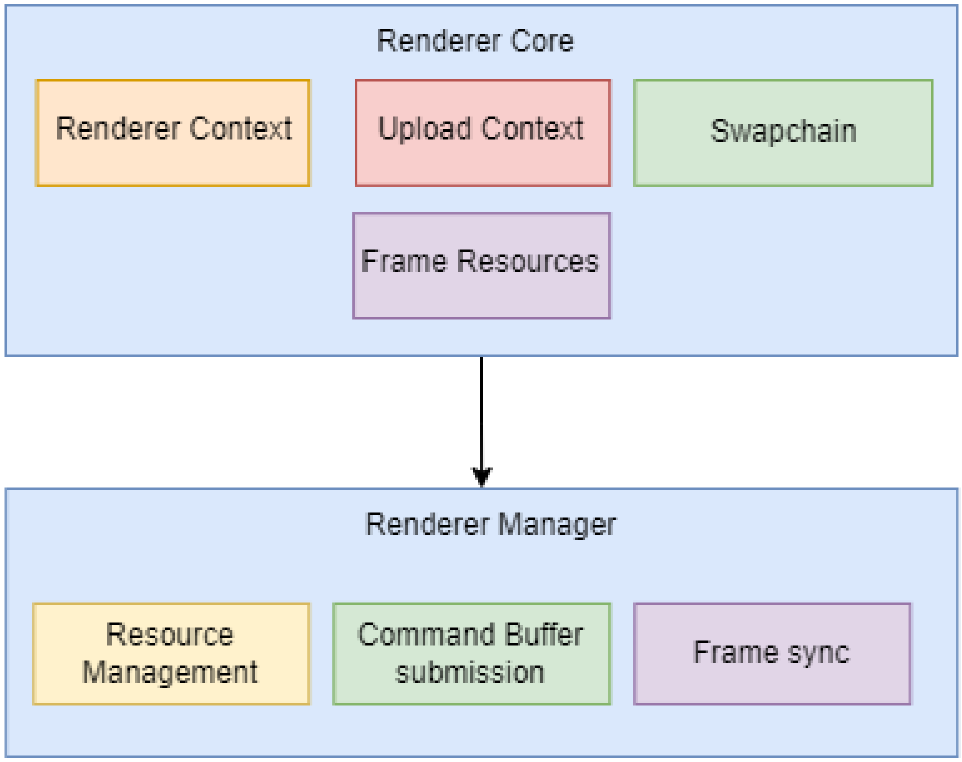


Figure 10: Renderer Architecture

In terms of the main rendering pipeline, two main techniques can be used called forward rendering and deferred rendering. The first is a more traditional method of rendering that renders and performs lighting calculations on all objects individually and finally displays the results on the screen. On the other hand, deferred rendering can significantly improve performance when large numbers of light sources are present compared to forward rendering. This is because deferred rendering defers all lighting calculations to a second pass and only needs to perform those calculations once. It also allows for important data to be accessed within shaders (GPU programs) such as normals, world positions, colours etc. Figure 11 provides a visualisation of the two techniques.

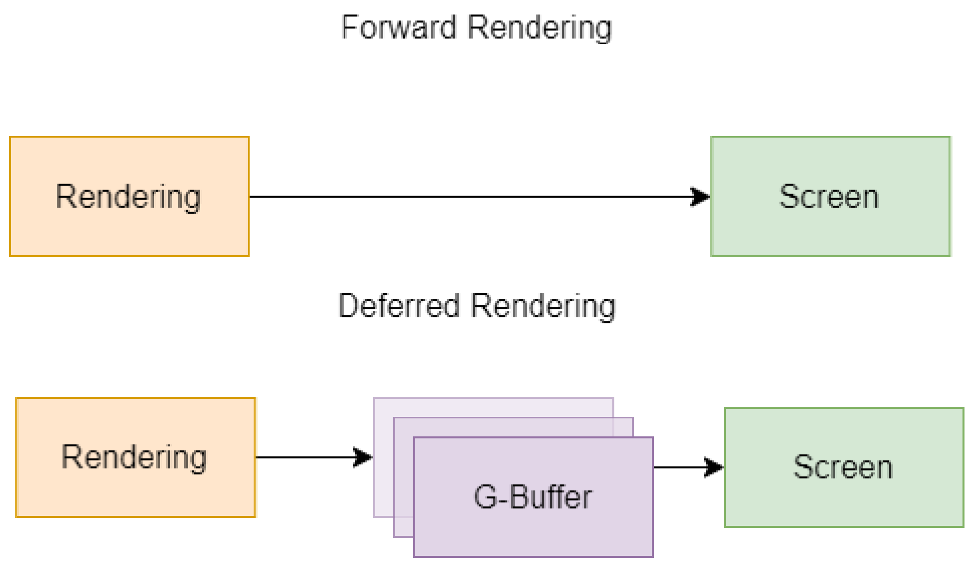


Figure 11: Forward vs Deferred Rendering

VMVE will make use of the deferred rendering pipeline as that is more widely used in the industry and primarily provides increased performance.

3.5 User Interface

Designing the user interface was the next step as part of the design stage of the project. Figure 12 shows the initial user interface wireframe that includes four main elements titled “Global Controls”, “Logs”, “Model Controls” and “Main Viewport”. Each of these UI elements is located in their respective windows and is designed in such a way that common controls are grouped and located appropriately if not within the same panel. The purpose of this is to provide an easy-to-use interface for interacting with VMVE and primarily the main viewport located in the center of the application. Figure 13 and 14 shows the initial proposed design for the windows that will allow for a user to load and export models within the application.

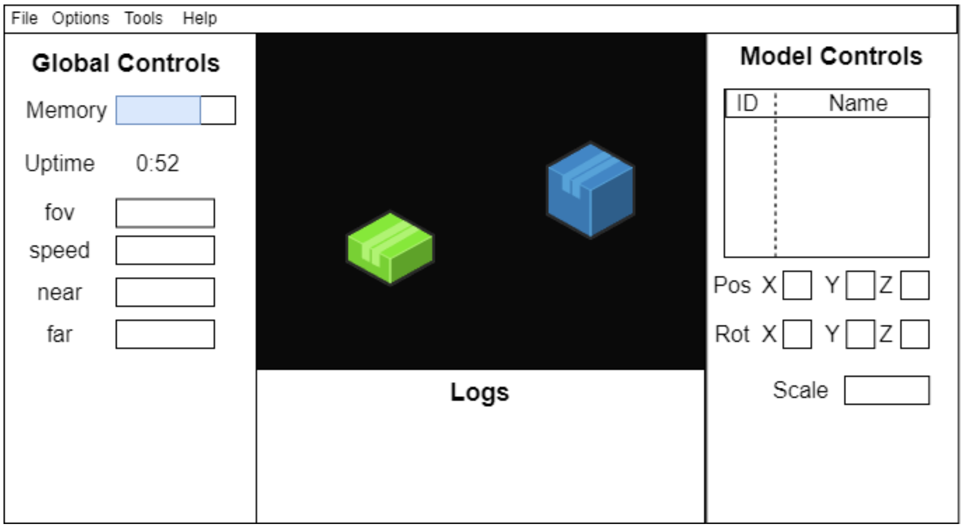


Figure 12: User Interface Wireframe

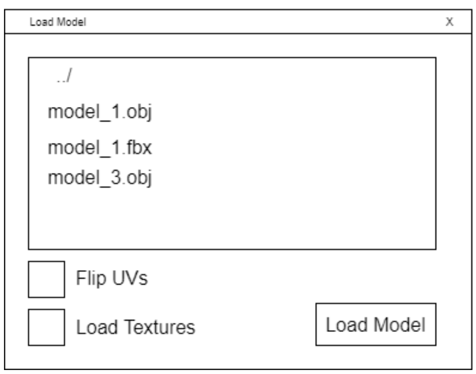


Figure 13: Load Model Wireframe

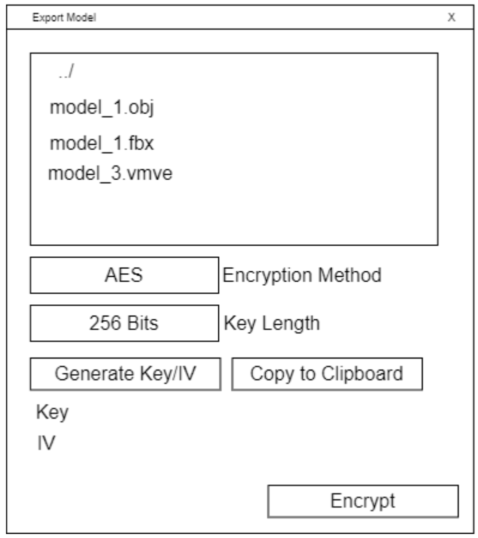


Figure 14: Export Model Wireframe

3.5.1 Main Viewport

The main viewport is located in the center of the window and is the main window of the user interface. The main viewport renders and displays the virtual environment. The size of the window is purposely designed to fill the majority of the screen space as it focuses the user's attention on that window but also increases visibility whilst interacting with the environment.

3.5.2 Global Panel

Located on the left side of the application, the global controls panel contains elements that apply globally throughout the application such as general information and camera controls allowing users to manipulate the view of the environment.

3.5.3 Model/Entity Panel

The right side panel called model controls will display model and entity specific information and will be the primary control surface for interacting with objects in the virtual environment which includes adding and removing entities.

3.5.4 Logs Panel

The logs panel is designed to contain all internal messages that the application prints out. These messages will then be displayed within the logs panel. Each log message is one of three categories such as log, warning or error. Depending on which log type the message is, it will be shown in a different color such as white, orange or red respectively. This is designed so that the user will have a clear understanding of the internal state of the application.

3.6 Custom File Format and Encryption

3.6.1 File format

VMVE will include a custom file format that allows for model data to be encrypted for security purposes. Figure 15 shows the proposed internal structure of the file. The internal structure is split into two main sections. The header contains key pieces of information that help the application determine how and if the main data should be decrypted.

Version indicates which version of VMVE was used to encrypt the file. At the moment, this is to be purely informational however, in future versions, this can be used for reporting compatibility issues and also, allowing for encrypted files to be updated to newer file layouts.

Mode is the encryption algorithm used (e.g AES, DH) and is what will allow the application to know which decryption algorithm needs to be used.

Encrypted Keys is used for validating the input key and initialisation vector. If these match then the application will then decrypt and load the data.

The second element in the VMVE header is the encryption mode. This allows for VMVE to know which algorithm must be used to decrypt the data.

The remaining section of the file consists of the actual encrypted geometry data which must be loaded and displayed in the virtual environment.

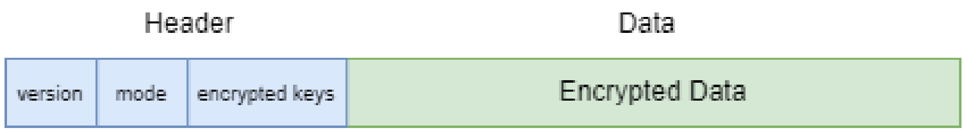


Figure 15: VMVE File internal structure

3.6.2 Encryption and Decryption

When dealing with encryption there are two primary stages, encrypting and decrypting. The term “encryption” means taking some data and running it through an algorithm making the data secure and unreadable. “Decryption” is the opposite of that which takes the unreadable data and turns it back to the original data.

Designing such a system requires careful consideration and extra care to ensure that the data being managed is not negatively affected such as information being lost when converting from one stage to another. The primary encryption algorithm that will be used is AES and the key length can be two different values (128 or 256 bits) that the user will choose based on the level of security they want for their assets.

Figure 16 shows a flowchart describing the general process of how a model will be loaded into VMVE.

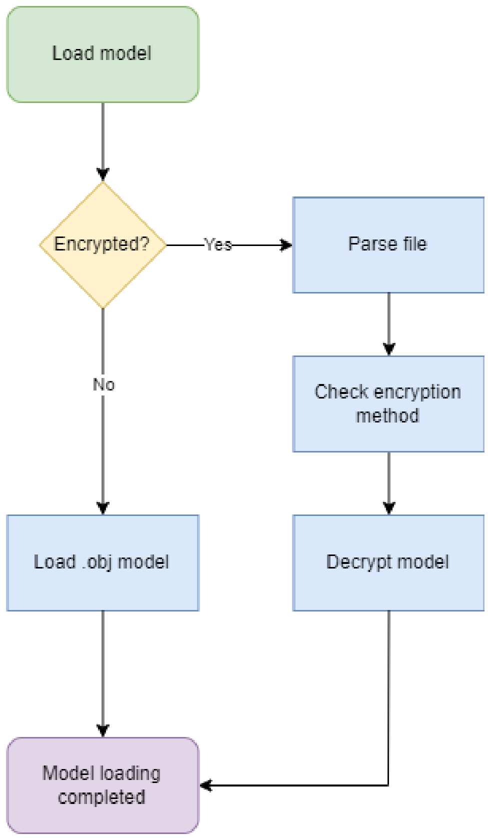


Figure 16: Load model flowchart

Likewise encrypting a model will consist of three main stages which are loading, encrypting and exporting as show in figure 17.

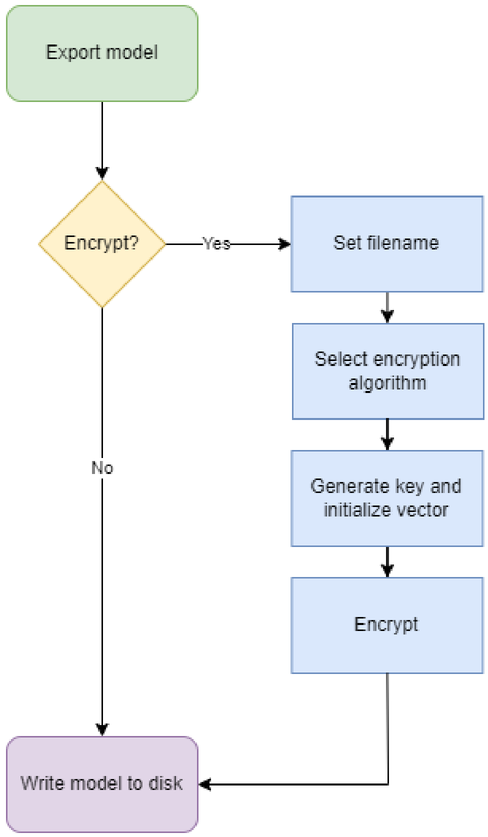


Figure 17: Export model flowchart

4 Implementation

The implementation section of this report presents the technical details of VMVE and is structured in order of initialisation for all subsystems of the application.

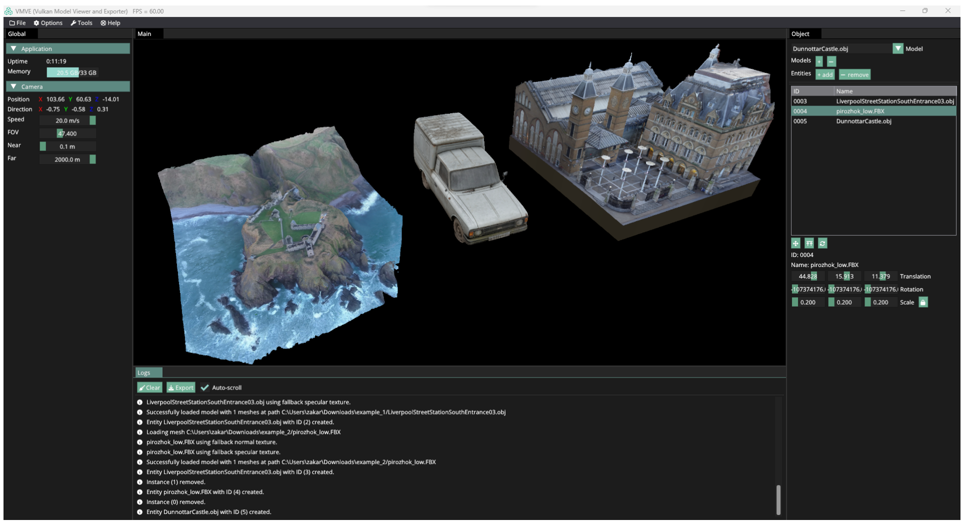


Figure 18: UI implementation

4.1 Overview

Internally, the application consists of several key stages that depend on each other. Figure 19 shows the general pseudo-code overview of the various stages that occur during initialisation, runtime and shutting down.

```
int main()
{
  // begin application initialisation
  create_window(width, height, name);
  create_renderer();
  create_audio();
  create_ui();

  // set default configuration options
  create_camera();

  // begin application rendering
  while (running)
  {
    update_renderer();

    render_geometry();
    render_ui();

    update_window();
  }

  // shutdown application
  destroy_ui();
  destroy_audio();
  destroy_renderer();
  destroy_window();

  return 0;
}
```

Figure 19: Implementation overview pseudo code

4.2 Window System

The first subsystem within VMVE that initialises is the platform window. This system is responsible for creating a desktop window based on a series of configuration options specified at the start of the application. These options include the window width, height and name. Internally, VMVE uses the lightweight GLFW library to handle window creation. Under the hood, on Windows, the library uses the Win32 API provided by Microsoft to construct the window. The purpose of this library is to provide an API that is cross-platform and allows applications to easily create windows on different operating systems. In addition to the window creation, various function callbacks are created to handle specific events such as window resizing, input, cursor position etc.

4.3 Rendering System

The next subsystem that initialises is the Vulkan renderer which is the primary feature that was implemented throughout the project. This system is responsible for communicating with the GPU and rendering geometry data onto the screen via the window created previously.

4.3.1 Renderer Context

The first step in creating the renderer is initialising a structure called "vk_context". This object is designed within the application that groups key Vulkan handles which are responsible for creating GPU resources. The handles within the context structure can be seen in the code snippet below and a diagram can be seen in figure 10.

```
struct vk_context
{
  VkInstance          // Initialises Vulkan library
  VkPhysicalDevice    // Handle to physical hardware
  VkDevice            // Logical handle to physical hardware
  VkQueue             // Graphics queue
  VkQueue             // Presentation queue
  VmaAllocator        // VMA memory allocator
};
```

The initialisation of the renderer context can be broken down into several stages. The first of these states acquires the function pointers into the Vulkan loader which contains the implementation details for the GPU driver. This is achieved by statically linking a "stub" library file into the executable. However, as discussed in a blog post by Arseny Kapoulkine titled *"Reducing Vulkan® API call overhead"* [20], he finds that this method has some overhead due to how function calls are dispatched. To reduce this overhead, a meta-loader called Volk is used which contains pointers to the real implementation and is dynamically loaded at runtime.

The next step is acquiring a GPU with a specific set of requirements that can be used for rendering. This step involves querying all available GPUs their names, hardware capabilities as well as if it supports basic screen rendering.

4.3.2 Swapchain

A swapchain is a series of images also known as framebuffers used by Vulkan that allocates a region of memory that stores frame data i.e. pixel colours. As the name suggests, a "swapchain" is a chain of these images. The most common number of images is two or three which is called "double-buffering" and "triple-buffering" respectively. The renderer within VMVE is configured to use two images, one for the front buffer that is shown on a monitor and a back buffer that is used for rendering.

In addition to buffers, a swapchain is also responsible for handling vertical refresh rate also known as vsync. This is the rate at which new images are rendered and then presented on a user's screen. There are two main types of vsync modes used within Vulkan which are `VK_PRESENT_MODE_FIFO_KHR` and `VK_PRESENT_MODE_IMMEDIATE_KHR`. The "fifo" mode indicates to the swapchain and subsequently, the presentation engine that images are to wait for the monitor's refresh rate before being present whereas "immediate" simply renders images as fast as possible.

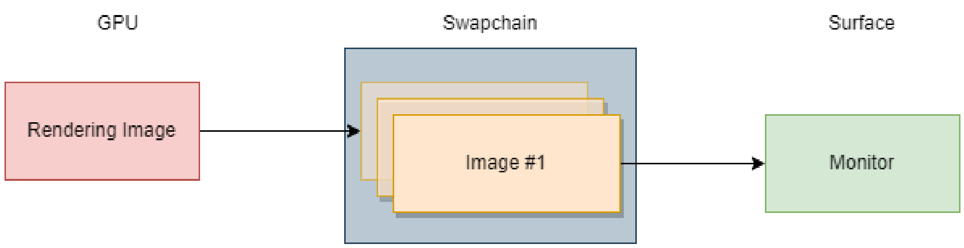


Figure 20: Swapchain

VMVE uses the "fifo" presentation mode as it conserves hardware resources as new images will not be rendered needlessly. Also, this mode ensures that no screen tearing occurs between refresh rates. Figure 22 shows a detailed image regarding the presentation engine and frame synchronisation.

4.3.3 Frame synchronisation

Once the swapchain along with its framebuffers has been fully initialised, the renderer can now start creating the necessary resources required for frame synchronisation. Vulkan has two concepts that relate to synchronisation which are VkFence and VkSemaphore.

```
struct vk_frame
{
    VkFence submit_fence;

    VkSemaphore image_ready;
    VkSemaphore image_complete;
};
```

Figure 21: Frame sync structure

The Vulkan 1.3.246 specification states that "`VkFence` is a synchronisation primitive that can be used between queue submissions and the host" [7.3 Fences] [21]. In other words, a fence can be signaled on the CPU to indicate in this case, when a new image from the presentation engine has been acquired via the `vkAcquireNextImageKHR` function call.

Similarly, "`VkSemaphore` is a synchronisation primitive that can be used between queue operations" [7.4 Semaphores] [21]. For the application, this means that rendering operations can occur simultaneously for multiple frames while ensuring that frames "in-flight" are not interrupted.

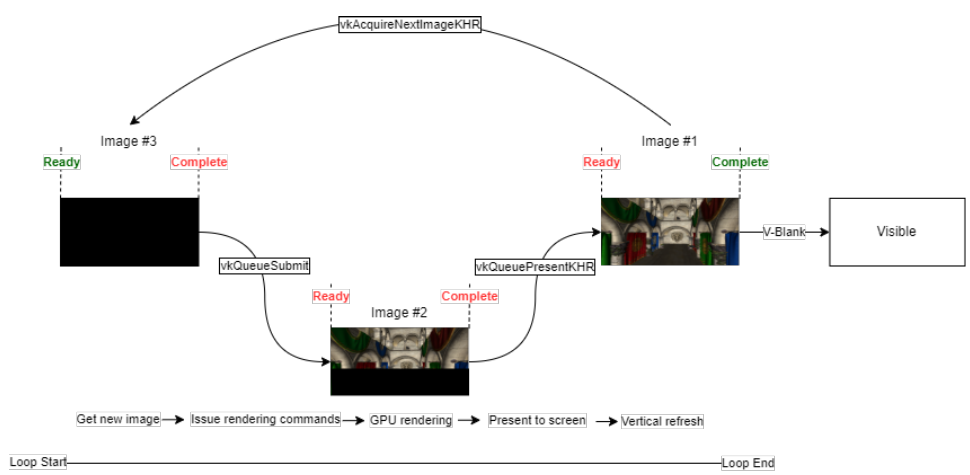


Figure 22: Frame Synchronisation

4.3.4 Deferred Rendering Pipeline

The next rendering feature that was implemented was the deferred rendering pipeline as proposed in section 3.4.5. During rendering, there are two main passes. The first pass renders the raw geometry data and performs MVP translation (section 4.3.9). For each rendered frame a total of five framebuffers are filled with different types of information including colors, world positions, normals, depth and specular. Then the second pass reads the data from the five framebuffers and performs the lighting calculations.

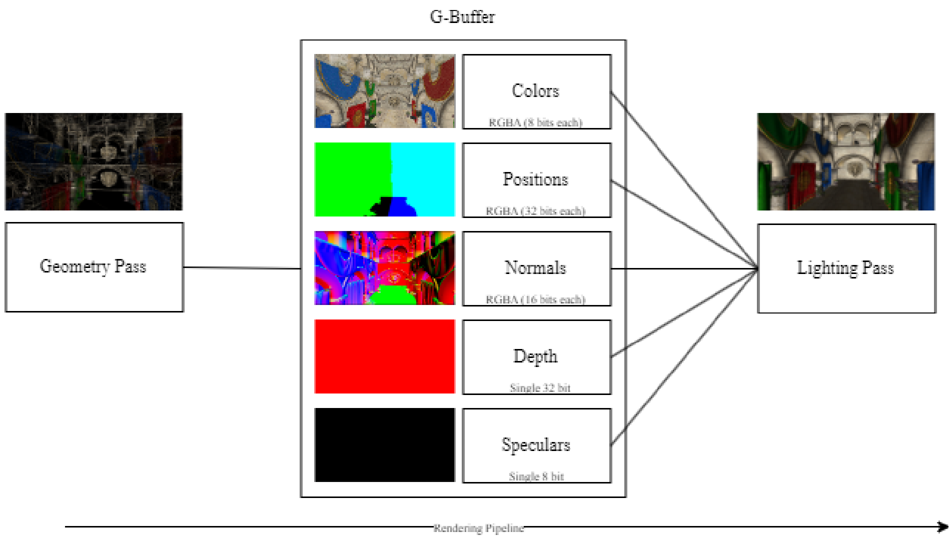


Figure 23: G-Buffer Pipeline

4.3.5 Dynamic Uniform Buffers

Uniform buffers are regions of memory located in Video Random Access Memory (VRAM) on the GPU. This memory can be accessed by shaders when creating a frame. These buffers can contain any sort of data including camera information, light properties, object transformations etc.

VMVE makes use of two dynamic uniform buffers one for the camera projection and the other for scene information such as lights. A dynamic buffer is a buffer that is large enough to store per-frame data and each frame of data can be accessed using an offset into the buffer and the current frame index. Figure 24 shows the internal structure of a dynamic uniform buffer that stores three sub-regions of memory in a triple-buffered frame setup.

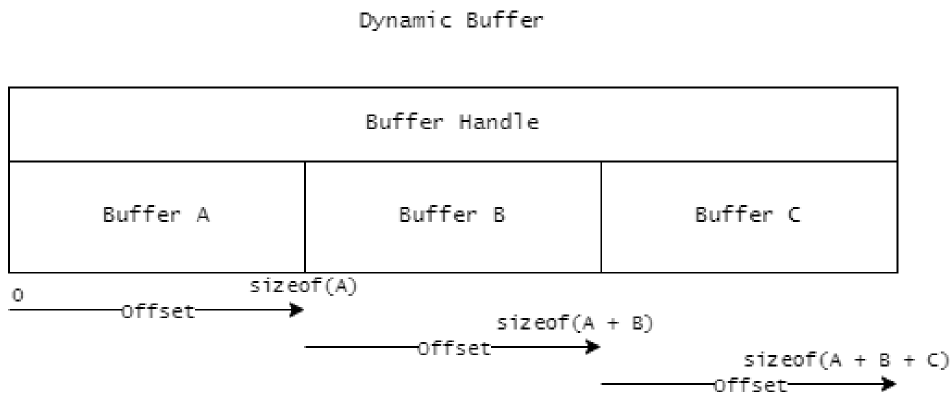


Figure 24: Dynamic Uniform Buffer

The use of dynamic uniform buffers provides several benefits compared to using a unique buffer handle for each frame.

4.3.6 Texture Mipmapping

VMVE makes use of a rendering technique known as "Texture mipmapping". This improves the visual fidelity by reducing if not eliminating Moire patterns which are visual artifacts that appear when identical patterns/textures are transformed on the screen[22].

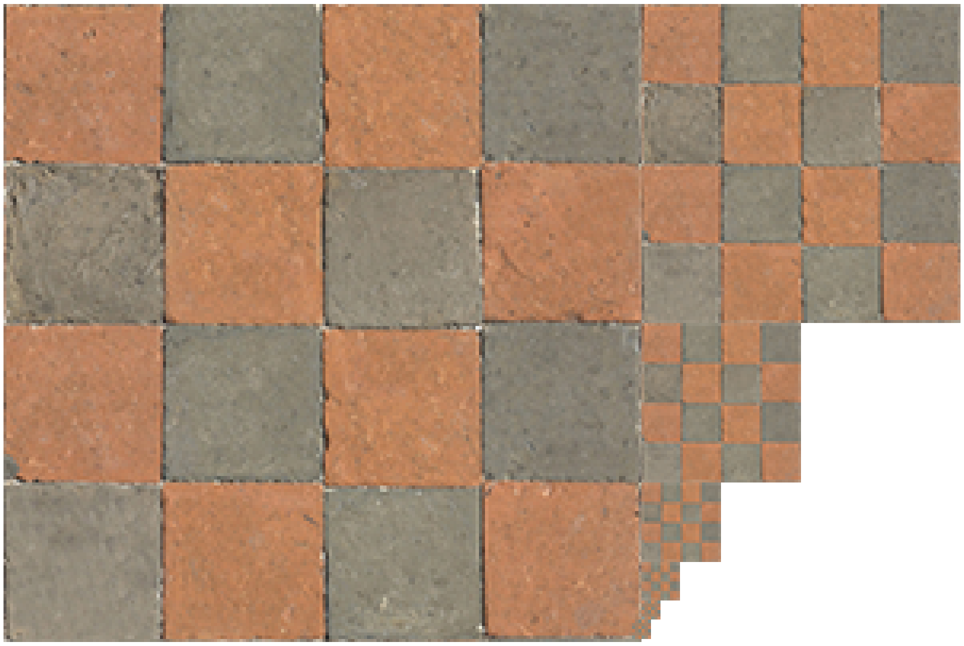


Figure 25: Texture mipmapping
[23]

Using Vulkan, this is implemented by taking the originally loaded texture and continuously halving it on both the width and height so that the subsequent texture is a quarter of the previous size. This is performed using the following equation provided by "Sascha Willems".

$$m = \lfloor \log_2(\max(w, h)) \rfloor + 1 \tag{1}$$

where m is the number of mipmap levels, w and h are the width and height of the source texture respectively. For each texture mipmap created the size of the texture becomes exponentially smaller.

In the source code, it looks like this:

```
const auto mip_levels = static_cast<uint32_t>(std::floor(
std::log2(std::max(width, height)))) + 1;

```

Figure 26: Mipmapping equation

4.3.7 Models

In the context of the rendering system, a "model" is a structure that represents a 3D geometry object. This could be as simple as a cube or as complex as an entire scene. The data structure contains several key pieces of information such as geometry data and textures.

Complex models are not singular pieces of geometry. Instead, artists combine multiple smaller objects to form the final model. These smaller parts are known as "Meshes" within the application. Each mesh has its own vertex and index buffers stored on GPU that describe to Vulkan how the piece of geometry is to be interpreted.

Figure 27 shows the internal structure of an example model which has been loaded into VMVE.

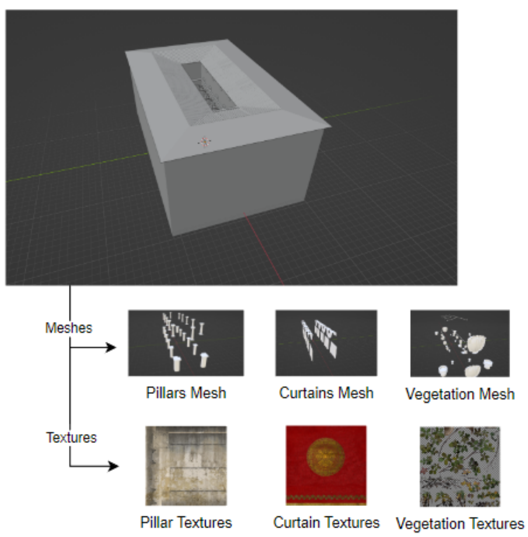


Figure 27: Model Structure

When a user loads a model, the model is appended to a continuous array (`std::vector<Model> models;`) which is iterated over each frame in order to render all the models on the screen.

4.3.8 Entities

An entity represents an instance of the model in the virtual environment. There can be many different entities using the same model. An example of this can be seen in figure 28 which shows four entities using the same backpack model.

This is achieved by storing a model index that indicates which model the entity refers to within the contiguous array of models. Secondly, an entity contains a transformation matrix that positions the entity in the world. A more in-depth explanation of the model matrix can be found in section 4.3.9.

Figure 28: Multiple entities in virtual environment

4.3.9 Camera

3D geometry data must be transformed through a series of mathematical calculations that will take vertex points from a 3D "world" and then convert them onto a 2D image for it to then be displayed on a monitor. This transformation is known as perspective projection[24] and is accomplished by computing several intermediate coordinate spaces as seen in figure 34.

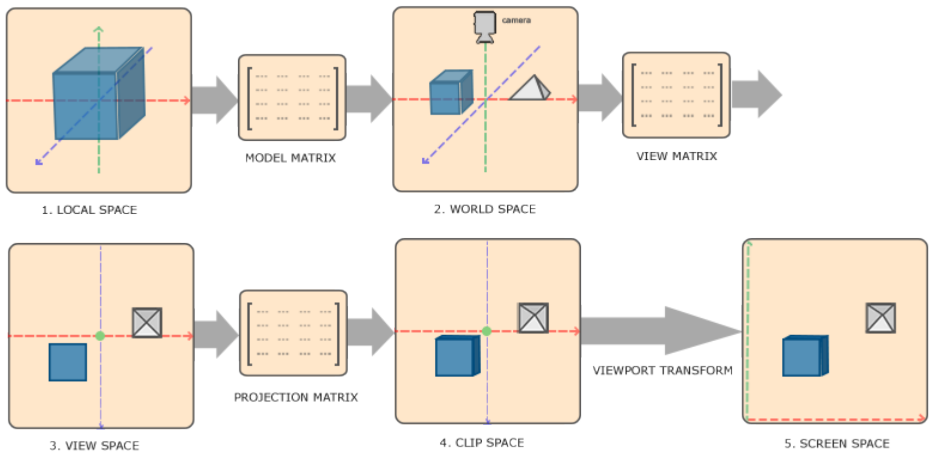


Figure 29: MVP Transformation[25]

Geometric data is first created within a local coordinate space. This "space" is positions relative to the object. In other words, the model data imported by VMVE will be in this local coordinate system often created by a 3D modeling program such as Blender or 3DSMax.

The next step is to convert these local coordinates to "world" space. This is the actual virtual environment that an object will live in. The transformation to this space is done by constructing a 4x4 matrix and performing a combination of translation, rotation and scaling. A pseudo-code example can be seen in 30.

```
mat4 model = mat4(1.0f);         // Identity matrix
model = translate(position);     // Move object
model = rotate(radians, axis);   // Rotate object
model = scale(scale);            // Scale object

```

Figure 30: Model matrix construction

The next step is transforming the world space positions to view space or also known as camera space. This is important as how the virtual scene is displayed depends on the properties of the camera. In the context of graphics rendering the concept of a "camera" does not exist and is only really an illusion. In reality, all points/vertex positions in the world space are transformed relative to the "camera". For example, if the position of the camera along the z-axis increases i.e. we move forward then all objects in the world will be moved towards us. A nice visualisation of this is shown in figure 31 courtesy of a blog post by Jordan Santell https://jsantell.com/model-view-projection/.

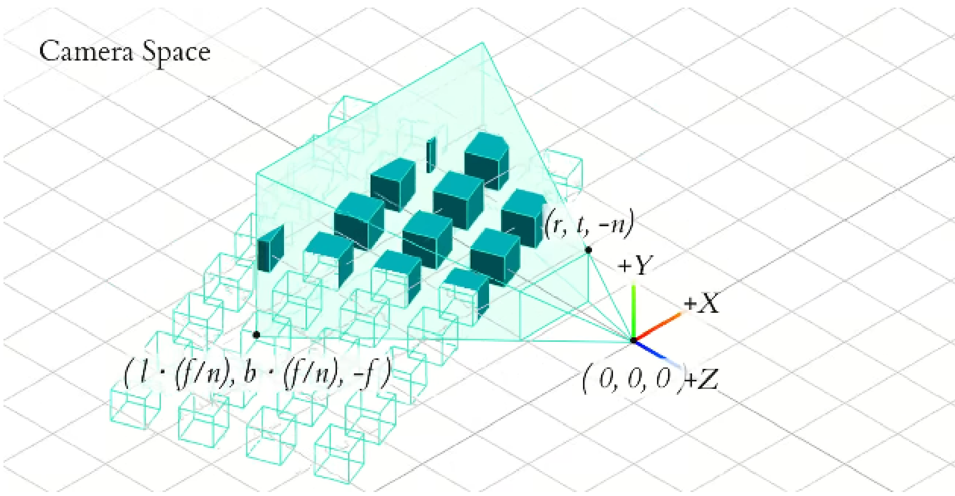


Figure 31: Camera/view projection[26]

VMVE makes use of a quaternion to perform the view projection. This is an alternative to Euler angles and has several benefits including, preventing gimbal lock and easier interpolation between orientations.

```
mat4 view = lookAt(camera_position, view_direction,
camera_up);
```

Figure 32: View matrix construction

The final step in calculating the MVP matrix is the projection which converts points from local space to clip space. This involves projecting the points to either a perspective or orthographic projection.

```
mat4 proj = perspective(fovy, aspect, near, far);
```

Figure 33: Projection matrix construction

With the fully constructed mvp matrix, this can now be multiplied with each vertex position and it will display onto the screen as expected.

$$Output = MVP \times Vertex \tag{2}$$

Figure 34: MVP projection

A more detailed code example can be seen in figure 71 in the appendix.

4.3.10 Lighting

The implementation of lighting was the final rendering system that was implemented. Lighting in Computer Graphics refers to how the virtual environment i.e. the colors of the resulting pixels are manipulated based on certain properties such as light, surface direction, occlusion, emissive materials and more. The use of lighting significantly improvises the visual fidelity of the graphics and increases the realism.

The lighting model that was implemented is called Blinn-Phong which derives from the original Phong model was developed by Bui Tuong Phong in a paper titled *"Illumination for Computer Generated Pictures"* published in 1975[27]. It describes three main elements ambient, diffuse and specular lighting that work together to produce the final result.

Ambient Ambient is a constant value that is used instead of global illumination as it requires more computational resources and more complex algorithms.

$$A = G + P \tag{3}$$

Diffuse The next step is implementing diffuse lighting. This takes into account the light direction L and the normal N for a surface at a particular pixel. The formula below calculates the intensity at which light reflects off of an object based on the angle of a particular surface.

The dot product of the light direction and surface normal returns a value between the ranges of -1 to 1 depending on how parallel the directions are. If this value is equal to or less than 0 then it indicates that the light is behind the surface and thus, is clamped as a result of the max function which returns the largest value for the given two parameters.

$$\mathrm{I} = \max(\vec{\mathrm{L}} \cdot \vec{\mathrm{N}}, 0) \tag{4}$$

$$\tag{5}$$

Having calculated the intensity value we can now simply multiply it with the objects' surface color as shown in the equation below.

$$\mathrm{D} = \mathrm{I} \times \mathrm{C} \tag{6}$$

Figure 35 demonstrates a simple example of diffuse lighting and how light interacts with an object based on the various properties discussed.

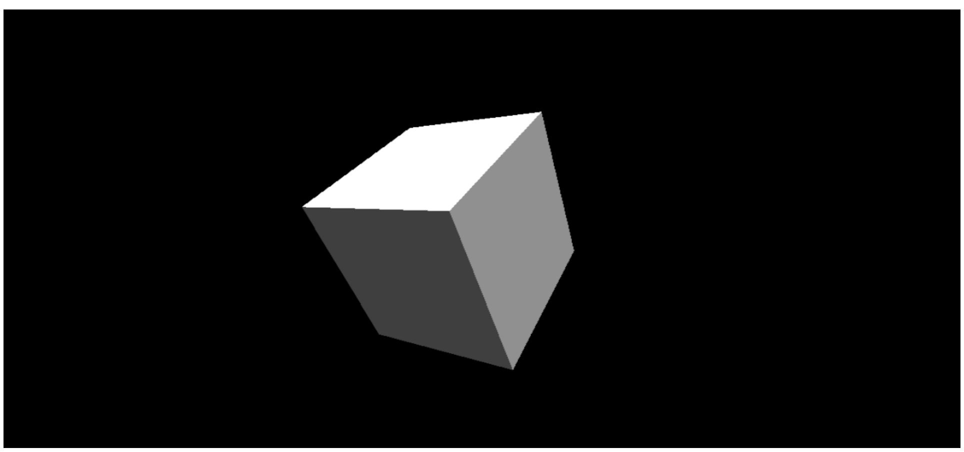


Figure 35: Diffuse Lighting

Specular The final step in the Blinn-Phong lighting model is specular. This effect adds highlights to certain parts of an object and is where the the intensity of light is strongest.

4.4 User Interface

The implementation of the user interface was the next major system. This is the front end and how the users will be able to interact with the application. The user interface is built on top of the Vulkan renderer.

4.4.1 Controls

The term "Controls" refers to a combination of keys also known as "Shortcuts" that are used to perform specific actions within the application. Various built-in controls aim to increase productivity for the user by reducing the amount of clicking and menu interaction for repeating tasks.

Controls are split into two categories which are Global and Viewport. Global controls are controls that can be activated at any point. Viewport controls, however, can only be activated whilst using the viewport i.e the camera.

This distinction for controls between different modes allows for greater context and better usability.

4.4.2 Fonts and Icons

The font used for the user interface is Open Sans which provides a simple and easy to read font.

VMVE also uses icons throughout the user interface and is an important aspect in conveying key information such as the task being performed or for additional information. The icon font used is provided by Font Awesome[28].

Typically, data for fonts and icons are stored in a font file which ends using extensions such as ".ttf" or ".otf". However, one of VMVEs goals is to be distributed as a single executable file. Therefore, we cannot depend on external font files. Instead, data for fonts and icons are stored directly in the application in a continuous array of bytes and are encoded in base85. A small example of what this looks like can be seen in figure 36. The full version of this is over 3000 lines long due to the vast amount of data stored within the font.

```
// 206 out of 201650 characters are shown

const char open_sans_regular[201650 + 1] =
"7])#######2Pc7('/###I),##d-LhLjKI##4%1S:`*]n8)K.v5*8_c)iZ
;99=$$$$c(m]4pKdp/(RdL<snZo'oI,hLNDnx4Uu/>8Q7oo^eFb3hB4JYc'
Tx-3l_wgd2Tf._r+&sAqV,-G"":F8LD=5,n]A&aA+<gXG-<iobW&>$>QJ8Z
.W$jg0Fv-o^(^JJnf4T"
```

Figure 36: Base85 encoded font

4.4.3 Menu Bar

The menu bar is a panel located at the top of the application and provides global and commonly accessed tools and functionality. Each option within the menu bar contains various properties that allow a to toggle key settings.

Figure 37: Menu Bar

File Has the main buttons for importing and exporting models. It also provides convenient "Exit" button to free all resources and terminate the application.

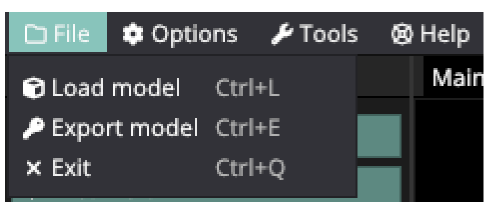


Figure 38: Menu Bar File

Options This is possibly the most frequently used item within the main menu as this has all the key options for the renderer and various visualisation tools.

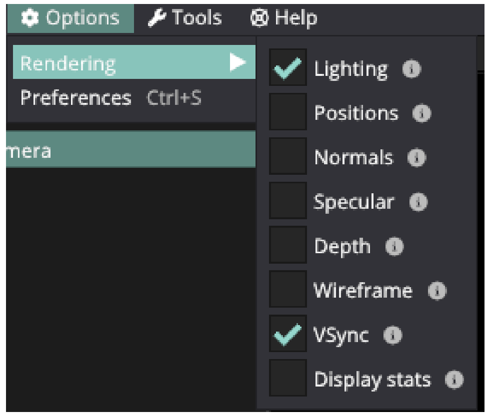


Figure 39: Menu Bar Options

Tools As well as rendering, VMVE will provide many other tools such as audio, a built-in console, an editor etc. To facilitate this, a dedicated tools menu is included. Currently, this acts as a prototype but in the near future will contain a plethora of fully featured supporting tools.

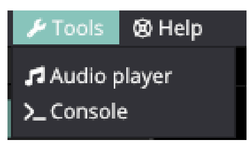


Figure 40: Menu Bar Tools

Help Lastly, the help option is for users who need additional assistance whilst using the application. It provides information about the application but also, a link that redirects users to the documentation website for VMVE

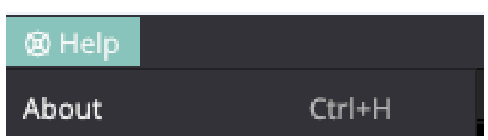


Figure 41: Menu Bar Help

4.4.4 Settings Window

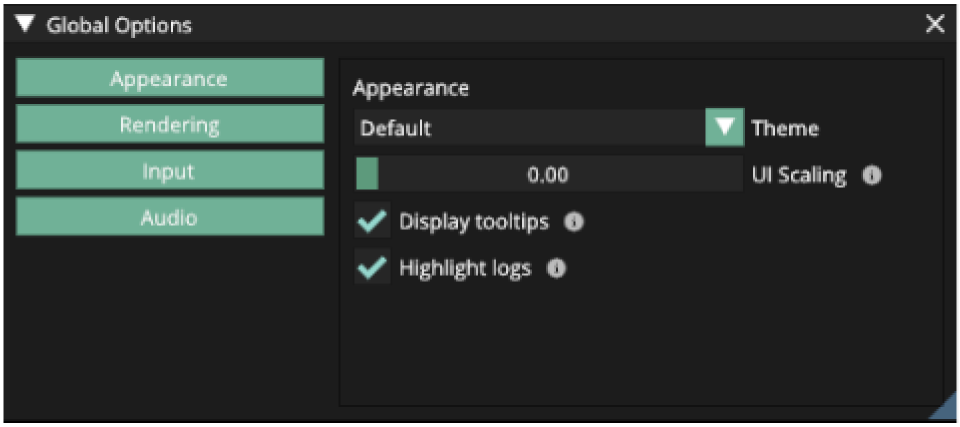


Figure 42: Settings Window

4.4.5 Load Model Window

The load model window is accessed through `File > Load Model` or by pressing the `Ctrl + L` shortcut. The window displays the file system which contains a list of items and directories. The user can interact with this in order to load a 3D model at runtime.

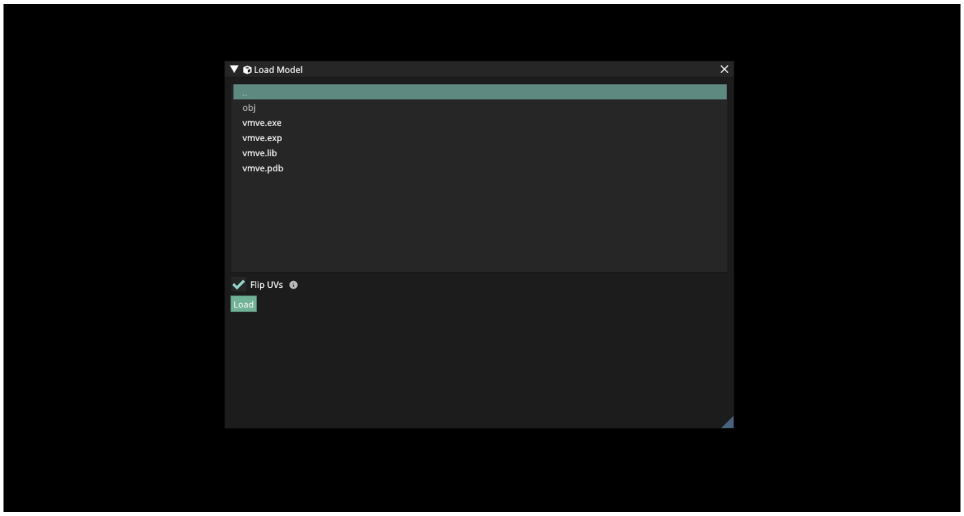


Figure 43: Load Model Window

If the 3D geometry file is detected to be encrypted then a modal window will popup asking the user to enter the files encryption key and initialisation vector.

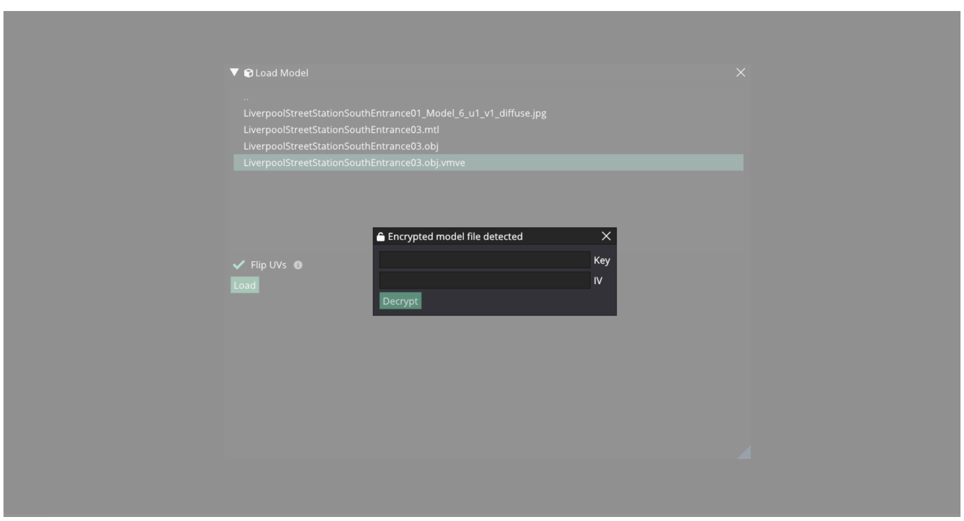


Figure 44: Load Model Encrypted Window

4.4.6 Export Model Window

The export window is responsible for encrypting and exporting models to the file system. There are two key options when encrypting a model which are the encryption mode and the length of the key. Users can change these options based on their security requirements. The next step is to generate the key and initialisation vector which can be performed by clicking on the "Generate key/iv" button. Finally, the model can be encrypted.

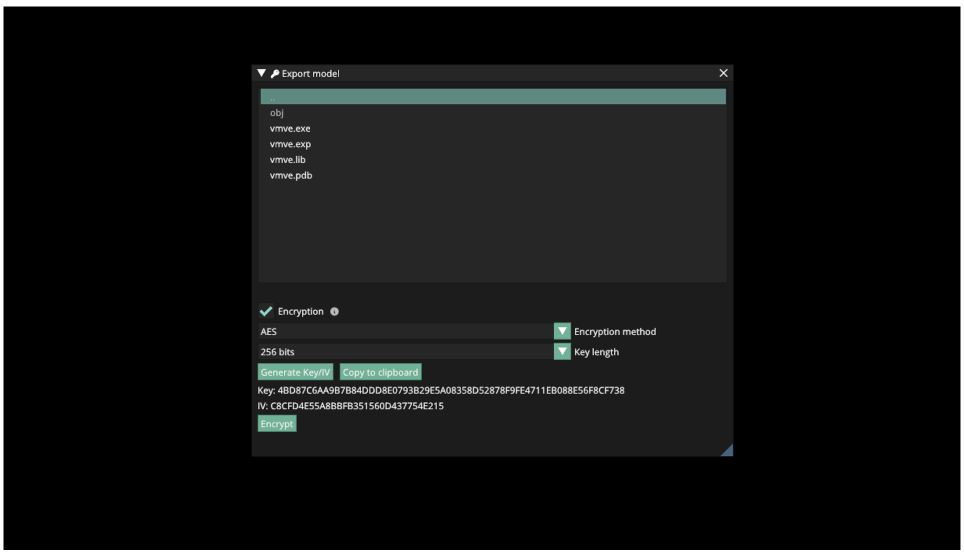


Figure 45: Export Model Window

4.4.7 Global Panel

The global panel contains application-wide information such as uptime and memory usage. Additionally, camera information and properties can be configured here which changes the way the virtual environment is rendered.

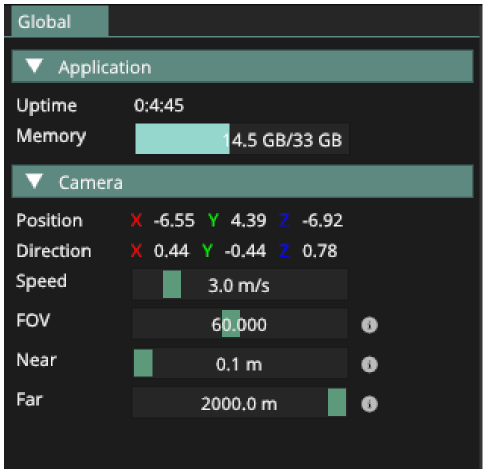


Figure 46: Global Panel

4.4.8 Entity Panel

The main method of interacting with entities in the virtual environment is through the entity panel. It contains a list of all loaded models in application as well as the entities as shown in figure 47. Below the entity list is dedicated controls for a specifically selected entity. This allows the user to manipulate the position, rotation and scale of an object in the world.

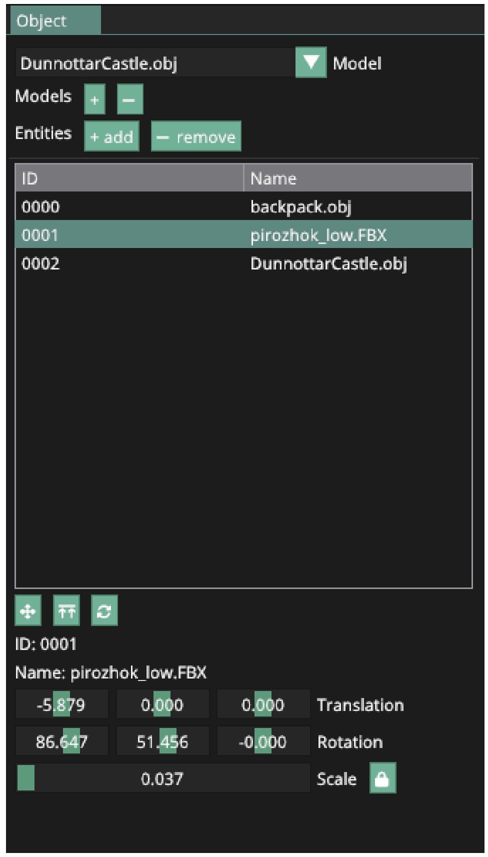


Figure 47: Entity Panel

4.4.9 Object gizmo

An additional feature that is part of the user interface is the gizmo. This is a visualisation of one of three operations is the main method of interaction that a user has with an object and is used to specify the exact location and orientation within the virtual environment. The operations are translation (moving), rotation and scaling and can be seen in figure 48 respectively.

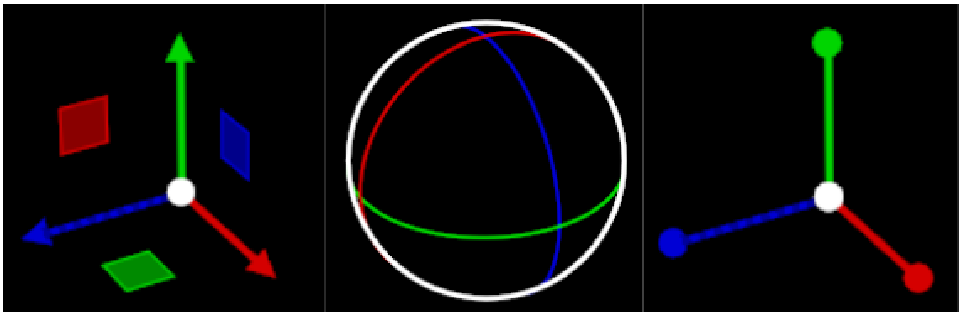


Figure 48: Gizmo operations

This functionality is implemented by taking the matrix transformation of an object and feeding that information to ImGuizmo[29]. It will then perform the necessary mathematical calculations to convert from the objects world space to screen space. In other words, no matter where in the virtual environment the object is located, it will always be shown relative to the user's screen. A small code snippet can be seen below 49 where M represents the model in the MVP projection.

```
const auto& operation = static_cast<ImGuizmo::OPERATION>(
guizmo_operation);
const auto& mode = ImGuizmo::MODE::WORLD;
ImGuizmo::SetDrawlist();
ImGuizmo::SetRect(x, y, viewport_x, viewport_y);
ImGuizmo::Manipulate(view, proj, operation, mode, m);
```

Figure 49: ImGuizmo world to screen space

4.5 Logging System

Due to the relative complexity of the underlying application and the number of moving components a logging system was developed to provide a clear understanding of the current state of VMVE.

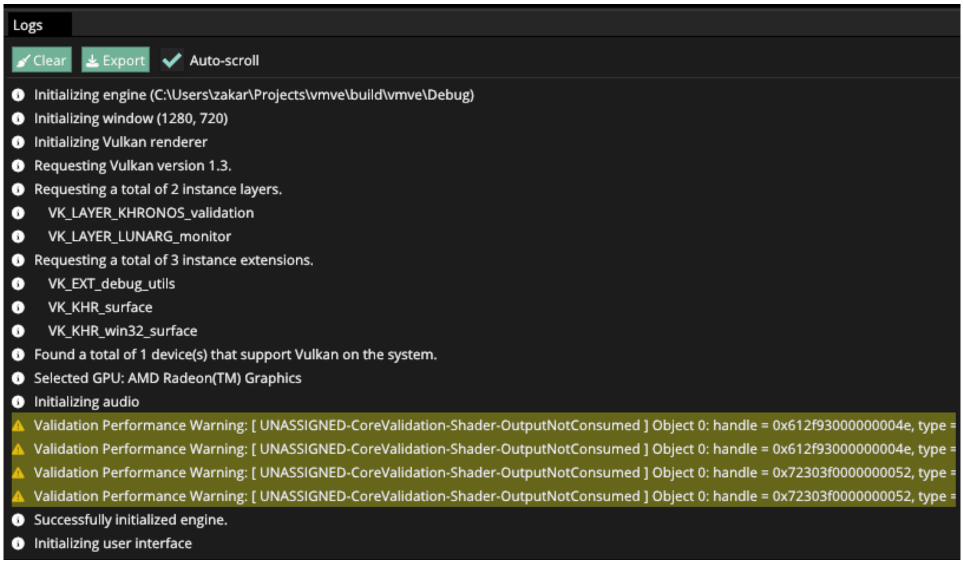


Figure 50: Logging window

Messages are stored in a data structure known as a ring buffer. This is a fixed-size buffer where the start and end of the buffer are connected. In other words, when the buffer becomes full, data stored at the start gets overwritten.

By default VMVE allows for a maximum of 200 log messages before the buffer becomes full. The underlying structure of this system can be seen in figure 51.

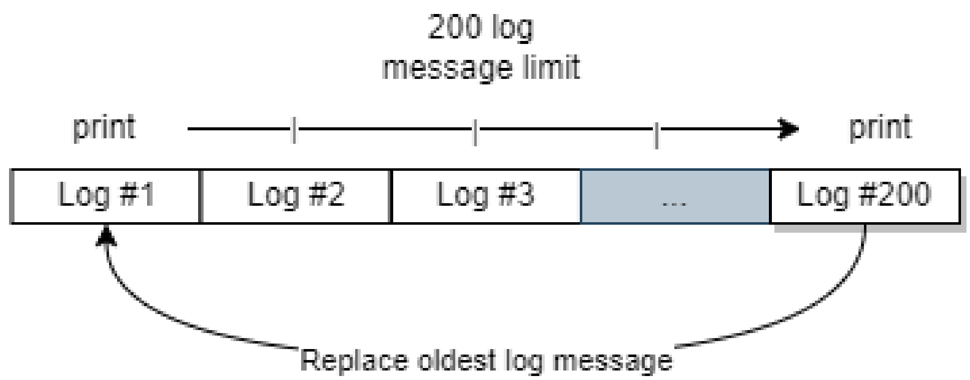


Figure 51: Logging FIFO system

The benefit of retaining log messages in memory as opposed to printing to the console is that they can be used at a later time such as for the UI or being saved to a file on disk.

4.6 Encryption and Custom Model Format

The simplicity of the Crypto++ library results in a relatively straightforward function, shown in Figure 52 which shows how a model is encrypted using the AES algorithm.

```
std::string encrypt_aes(const std::string& text, encryption_keys& keys, int key_size)
{
    std::string encrypted_text;

    // Set key and IV to encryption mode
    CryptoPP::CBC_Mode<CryptoPP::AES>::Encryption encryption;
    encryption.SetKeyWithIV(
        reinterpret_cast<const CryptoPP::byte*>(keys.key.data()),
        key_size,
        reinterpret_cast<const CryptoPP::byte*>(keys.iv.data())
    );

    // Encrypt text
    CryptoPP::StringSource s(text, true, new CryptoPP::StreamTransformationFilter(encryption, new CryptoPP::StringSink(encrypted_text)));

    return encrypted_text;
}

```

Figure 52: AES Encryption

4.6.1 Decryption validation

When a model is being decrypted the system must check and ensure that the key and iv being used match those originally used to encrypt it. Without this check in place, the system will not know what the “correct” key and iv are.

As outlined during the design stage, the file header contains the encrypted key and iv. When the user enters their specific key and iv as input parameters into the decryption dialog box as shown in figure 44 in section 4.4.5, VMVE will first decrypt the header section and check if the input values match. If successful, the application will then proceed to decrypt the data section and then load the file. If however, this check fails, then the application will throw an error and prevent the file from begin decrypted.

4.7 Distribution

The distribution is an important final step during the development of the application which compiles and optimises the source code and makes the application available for end users. Before compiling, the project configuration is switched to “Release” mode as seen in figure 53.

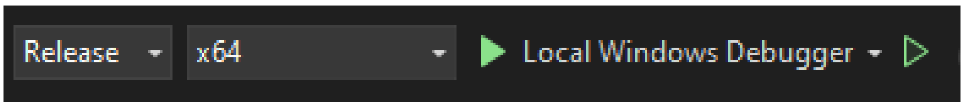


Figure 53: Project configuration

As mentioned in section 3.1.3, the compiler used is MSVC and therefore, specific compiler flags are used. One such flag is `O2` which performs a series of optimisations during the compilation phase which maximises speed such as frame-pointer omission, inline function expansion, string deduplication, function-level linking and much more.

4.7.1 Executable

These optimations ensure that the resulting application has a small executable size and performs efficiently. The resulting executable is named “vmve.exe”.

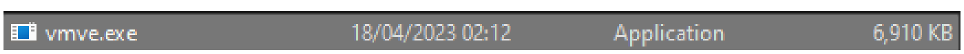


Figure 54: Executable

4.7.2 Versions

VMVE follows the major, minor and patch system of versioning and is used as follows [Major].[Minor].[Patch]. The use of versions is important in differentiating between VMVE releases. Each new version of the application comes with new features as well as bug fixes that aim to improve the software over time. Due to the rapid development of new features and functionality, backward compatibility cannot be guaranteed. The current official release of VMVE is v0.0.3.

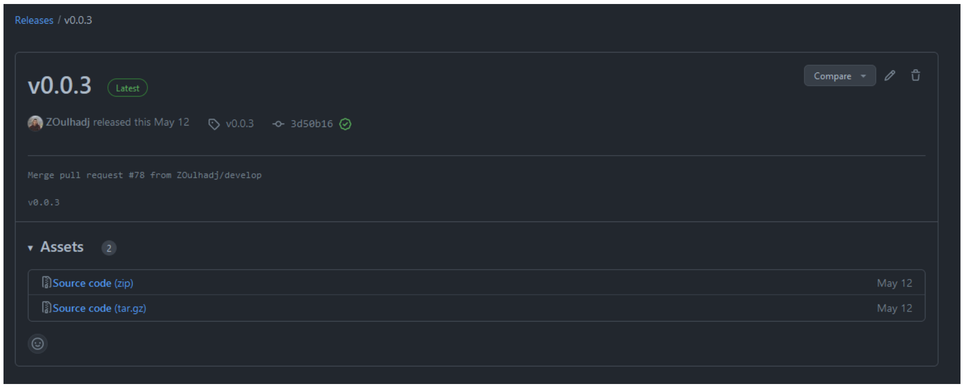


Figure 55: Latest release

4.7.3 Website

Alongside the application, a website was created as a platform for easily distributing the application. It includes a features section that showcases and gives users a preview of the application through the use of images and videos. Furthermore, a download link is provided giving users an easy way of obtaining the executable without needing to understand the technicalities of GitHub as it is designed with programmers in mind.

In regards to downloads, there are two types. The first is an active development version which is regularly updated and acts as a beta release for versions that have not yet been officially released. The second is current and past VMVE versions going all the back to the first official release (v0.0.1).

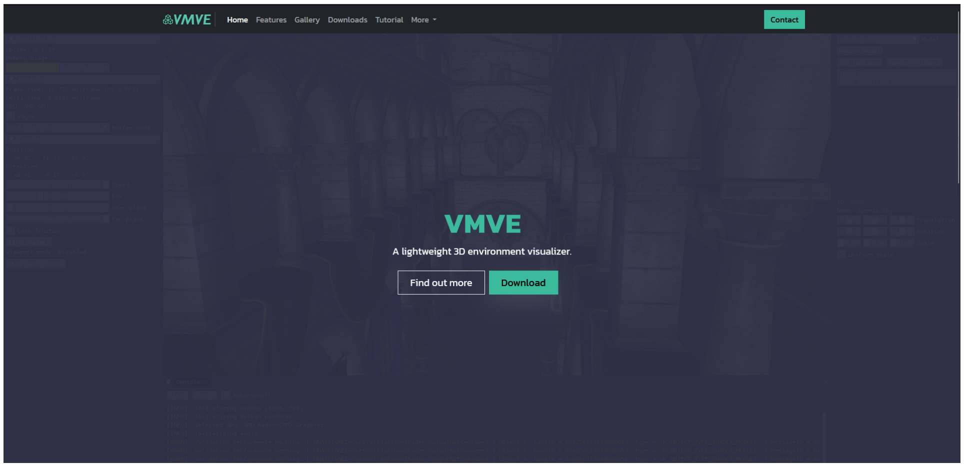
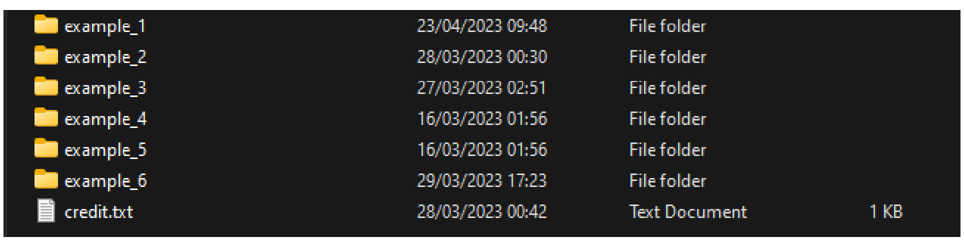


Figure 56: VMVE website

4.7.4 Example models

The VMVE website provides an additional download link to six different example models. These are provided by third parties (with appropriate credits) and are corrected to ensure they can be loaded into VMVE without any issues. The purpose of these example models is to allow users to easily test the application without needing to search the internet or create their own.



example_1	23/04/2023 09:48	File folder	
example_2	28/03/2023 00:30	File folder	
example_3	27/03/2023 02:51	File folder	
example_4	16/03/2023 01:56	File folder	
example_5	16/03/2023 01:56	File folder	
example_6	29/03/2023 17:23	File folder	
credit.txt	28/03/2023 00:42	Text Document	1 KB

Figure 57: Example models

4.7.5 Documentation

In addition to a website, a documentation website was also created courtesy of Read the Docs (https://readthedocs.org). The purpose of this site is to provide additional information about VMVE and also, provide an in-depth tutorial that aims to help users understand how to use the application.

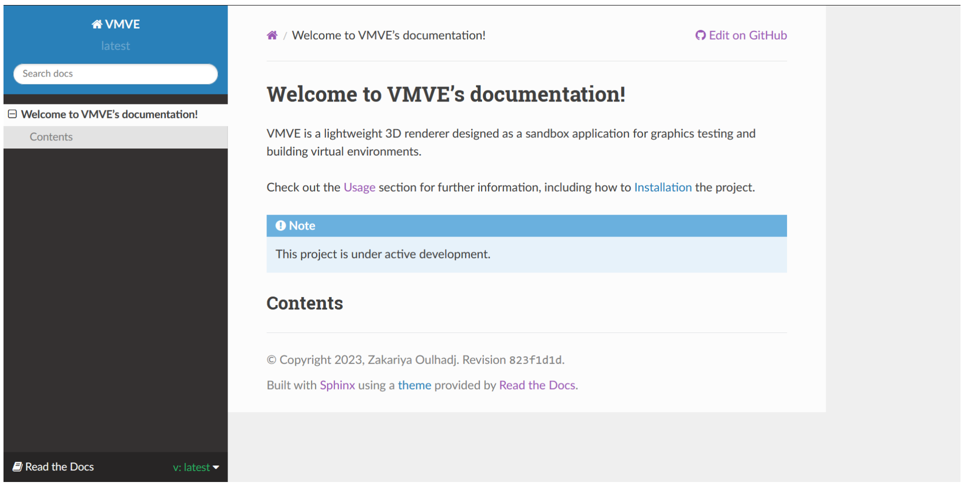


Figure 58: VMVE documentation

4.8 Evaluation

The completion of the project transitions into the evaluation stage where the project is evaluated against the original goals and requirements in order to judge if those requirements have been met. When evaluating the project there are two distinct categories. The first is self-evaluation and the other is user feedback.

4.8.1 Distribution

In hindsight, the most challenging aspect of the entire project was by far, distribution and more specifically ensuring that VMVE was able to run on all supported systems as expected. Throughout the applications development, there were points where VMVE would work on the development machine, however, throw an exception error on other systems. These inconsistencies paired with the lack of reproducibility made these issues quite difficult to debug. The majority of these inconsistencies between different systems occurred during the initialisation stages. This would include creating the window, initialising the renderer, initialising the UI, creating the audio system etc.

Some examples of these inconsistencies include the application crashing when resizing the window, crashing if audio is disabled on a system as well as frame stuttering.

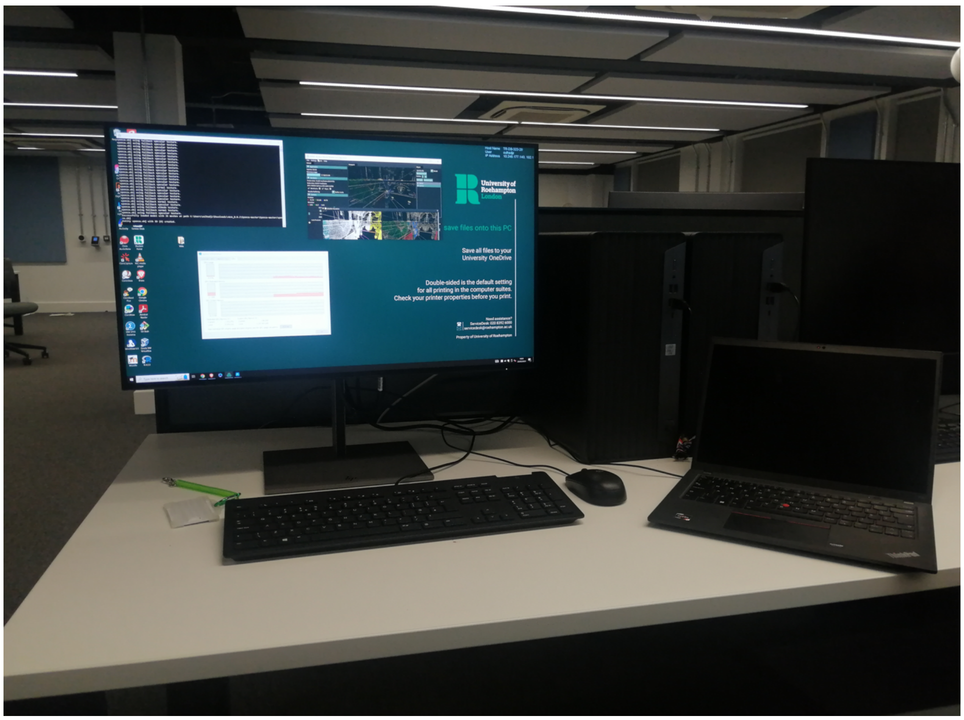


Figure 59: Working on distribution

During this part of development, the decision was made to design the logging system so that all messages in the buffer would be stored and written to disk before exiting. This allowed for unexpected crashes to be better understood as the `vmve_crash.txt` file could be analysed.

4.8.2 Time Management

Given the complexity of this project, it was vital to effectively manage time during development and to ensure that all of the requirements of the project could be met on time. Failure to correctly manage this workload would result in key goals not being achieved and/or features not being implemented.

As mentioned in section 3.1.1, GitHub was chosen and has been a highly effective tool throughout development regarding management and distributing tasks. Over 50 different GitHub issues have been resolved such as small user interface issues, implementation bugs and major structural changes.

Figure 60 shows that a total of 357 commits were made over the course of eight months. This includes the design and implementation stages of the application.

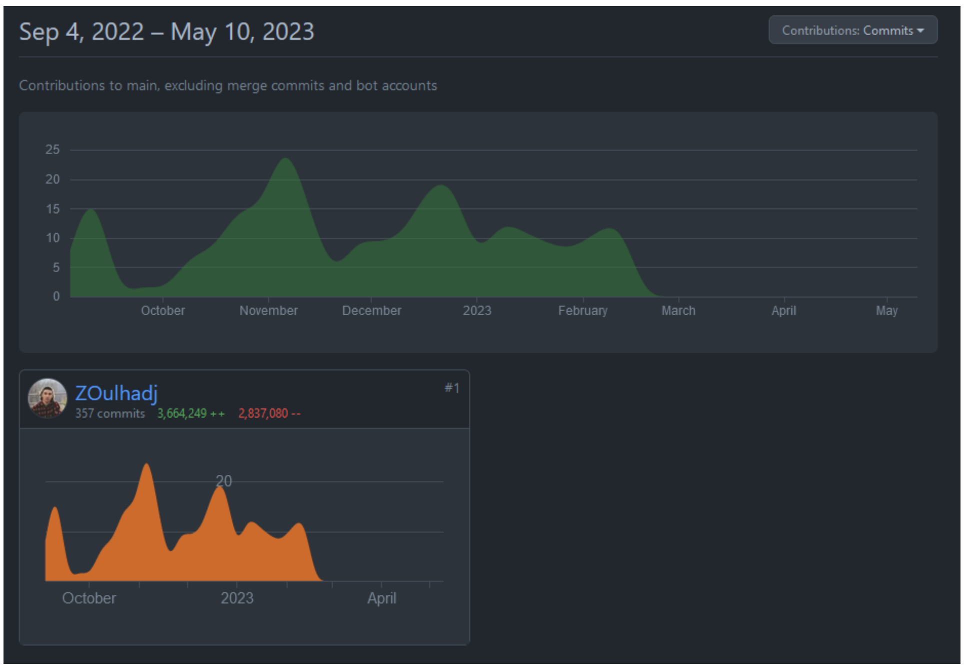


Figure 60: Commit history

VMVE currently has 32 outstanding GitHub issues that have yet to be resolved. These issues are not considered critical and do not hinder the project in terms of meeting its original requirements. However, they are notable and should be resolved soon.

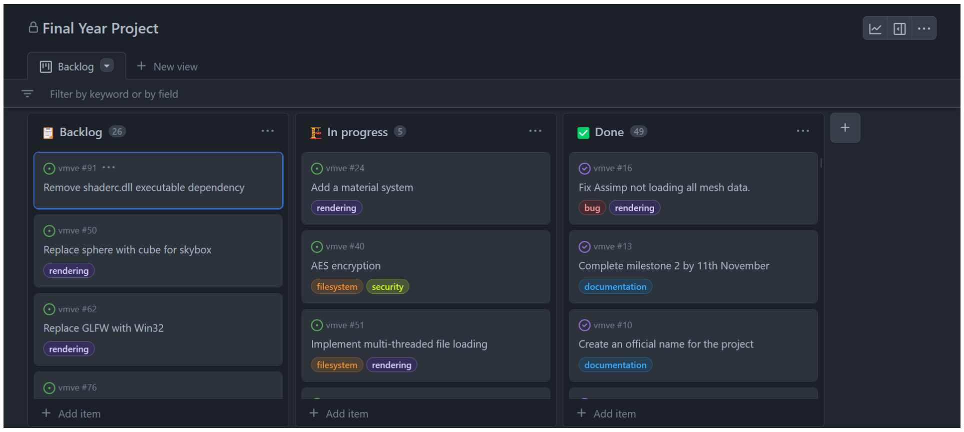


Figure 61: GitHub Issues Kanban

These tests were performed on the main development machine used throughout the project which was a Thinkpad T14s Gen 3. The following list shown in figure 4 shows the laptop's specifications.

Component	Name
CPU	AMD Ryzen 7 Pro 6850U
GPU	Integrated
RAM	32GB DDR6
SSD	1TB

Table 4: Development Machine

4.8.3 Performance

Measuring the performance of VMVE is another key aspect of evaluation that ensures the original goals and requirements have been met. Two main aspects can be measured which are compilation and runtime performance.

Compilation refers to the process of building the application in an offline setting. The speed at which the project is compiled directly affects the developer and subsequently the development of the project.

Runtime refers to the performance of the application while it is running and its effects on the end users.

VMVE takes an average of 53 seconds to build using MSVC. This includes compiling the source code and precompiled headers. To keep compilation times reasonable as the codebase grew in size and complexity, a Precompiled Header (PCH) technique was used. This involves including a header file (pch.h) that itself includes various STL libraries and external dependency headers that don't change often, reducing the compiler's workload by compiling it once and reusing it. The timing is measured in milliseconds using the "chrono" library. The startup time for VMVE is around 300ms in the distribution version, calculated by taking the time difference from the start of initialization to the end. The built-in profiler in Microsoft Visual Studio allows further analysis by using a tool called a "flame graph" (accessed through `Debug > Performance Profiler`) that takes a snapshot of the application's internal state at runtime.

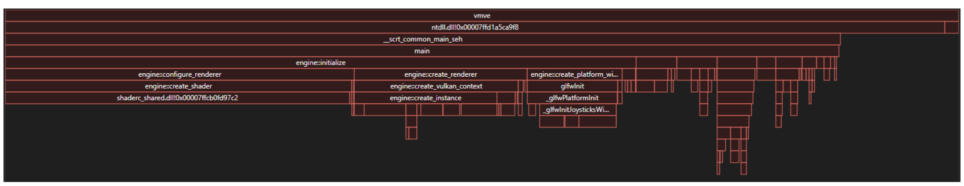


Figure 62: VMVE startup profiling

What this shows us is that a large chunk of the initialisation is taken up by the `engine::configure_renderer` function and more specifically the `engine::create_shader` function which is responsible for compiling the application shaders.

This can be taken a step further by expanding the hot code path which is what Visual Studio refers to as slow code. Figure 63 shows the exact function that takes around 36% of the total initialisation time.

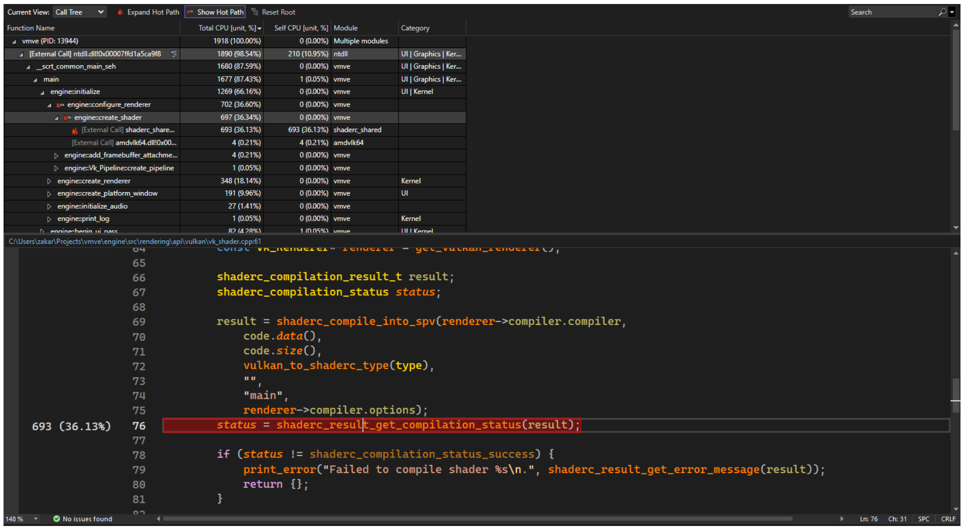


Figure 63: Profiling slow code path

The question one might ask is, "How can we improve this startup time?". Currently, VMVE stores the shader source code as simple strings within a header file called `shader.h`. The main advantage of this is that shaders can be combined with the executable rather than be in separate files.

The downside to this, however, is that the shader source is not in a precompiled binary format, meaning that each time VMVE runs, it must recompile the shader files.

In the future, this will need to be addressed and the system redesigned in such a way that allows for shader files to be precompiled and bundled with the executable at the same time eliminating the need to compile shader files and as a result improving the startup time.

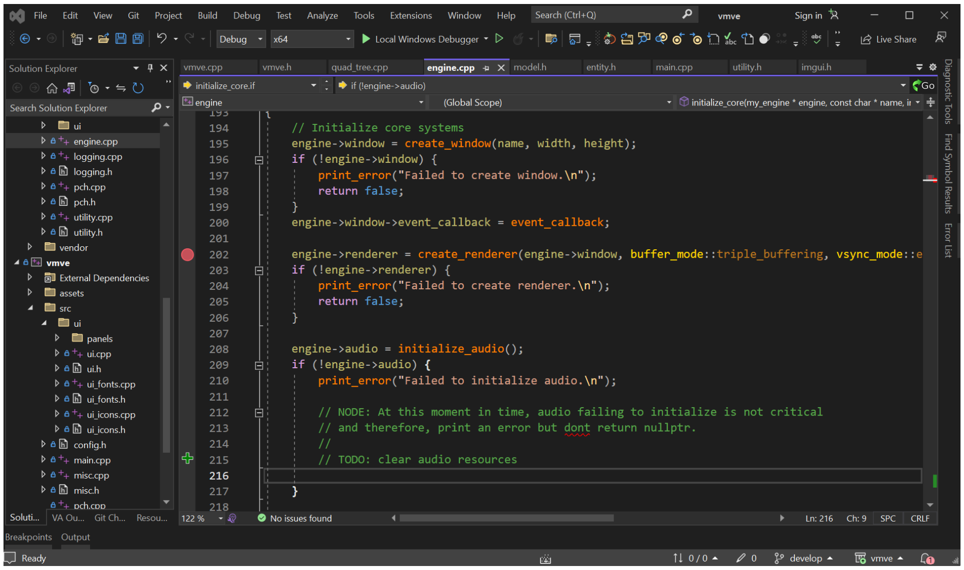


Figure 64: Microsoft Visual Studio

4.8.4 User Feedback

Having conducted the evaluation, it is equally important to obtain feedback from the stakeholders such as users. The users that tested the application were predominantly other students. The process of obtaining this feedback was separated into two stages.

The first was to measure how intuitive the user interface is which was achieved by giving a user a set of instructions as seen in figure 65 file. A score would be given for each task based on how easily the user was able to complete it.

User	Availability	Keys	Load a model	Add an instance	Enter the viewport	Enter instance edit mode	View surface normals	Encrypt a model	Load encrypted model	Change theme	Export logs to file
Tom	Yes	Completed									
Adam	Yes	Required assistance									
Bogdan	Yes	Not completed									
Taylor	Yes										
Danny	Yes										
Trevor	Yes										
Savergery	Yes										

Figure 65: User feedback instructions

The second stage would obtain direct feedback from the users by presenting a series of questions that would assess VMVEs usability, performance and overall user experience. This data was recorded using Google Forms and further analysed to produce different sets of visualisations.

The bar chart below shows the results of an ease-of-use test that was conducted between different users. The average ease of use score was 4.2. This indicates that the majority of the users found the user interface to be suitable for users regarding achieving their desired goals.

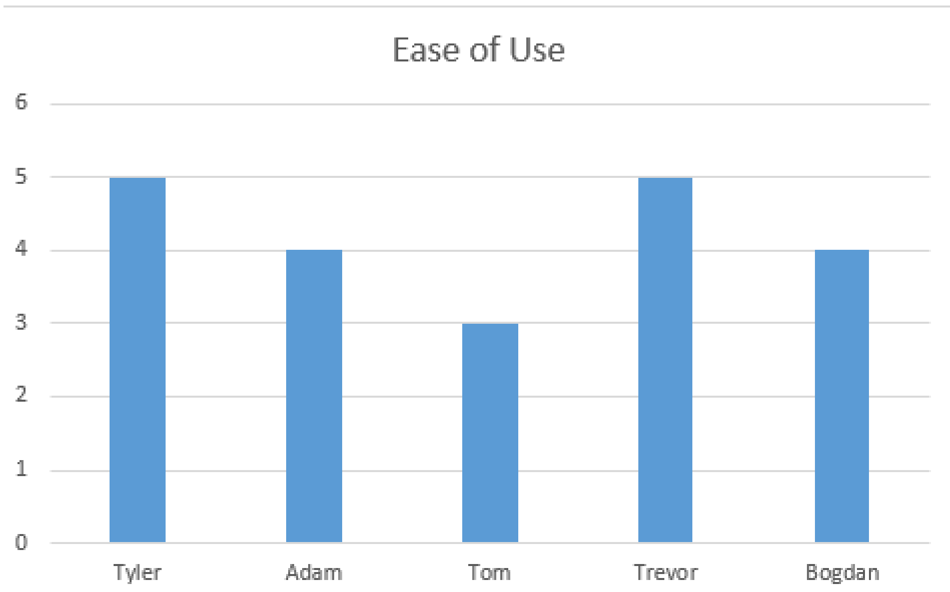


Figure 66: Usability bar chart

Tom who gave the VMVE an ease-of-use score of 3 said the following: "*Overall, the user interface is quite simple, however, I found that the shortcuts used throughout the application were not as easy to use as most required the use of two hands*".

This highlights an important issue in terms of ergonomics between the application and the user and as such will have to be addressed in the future.

Another test was conducted that asked users which of the main features of the application required additional work and overall improvements. The pie chart in figure 67 shows that lighting followed by performance were the two most mentioned features.

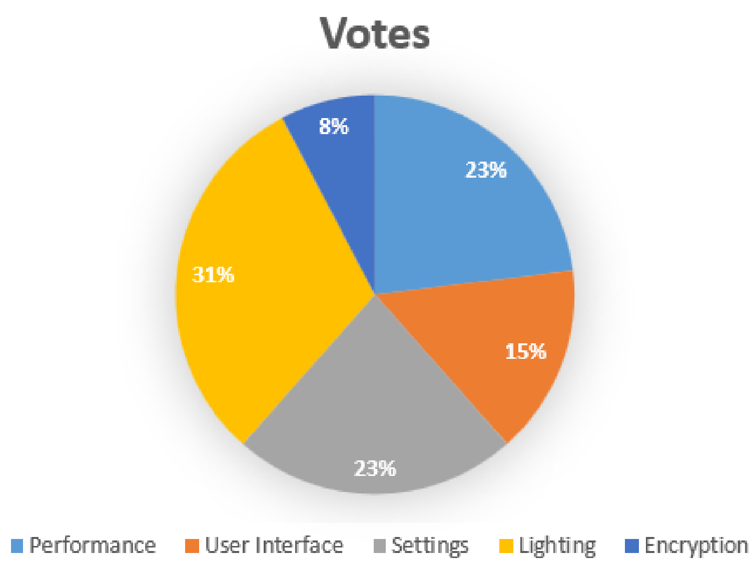


Figure 67: Feature Improvements

Taylor, one of the users that tested the application said the following: *When displaying a single object VMVE can do this reasonably well without any issues. However, when increasing the entity count so that there are many objects in the scene, the performance of the application starts to drastically lower.*

Based on how VMVE has been initially developed, this makes sense as the application does not make use of the many commonly found optimisations that are required for rendering large quantities of objects. This also needs to be addressed in future versions of the application so that usability is maintained, and that VMVE meets the demands and requirements of its users.

5 Conclusion

VMVE has been designed to be a platform in which users are provided with 3D graphics tools without needing to know the complex implementation details of the underlying technology. This allows users to focus on their needs and requirements when developing virtual environments. The project has been developed in line with the original aims and objectives and has been fulfilled with the primary aim of targeting students and trainees as an education tool for the digital graphics industry.

Although many of the original requirements have been met, the project as a whole must be evaluated to better understand the various strengths and weaknesses at key stages including requirements gathering, design, implementation and user feedback.

5.1 Reflection

To fully understand how well the project went, it must be evaluated as part of the reflection stage. This consists analysing the design, implementation and management surrounding the project.

5.1.1 VMVE Model Loading

The completion of the application exposed a requirement that was not fully met based on one of the initial aims. When loading an encrypted ".vmve" model, due to a limitation in the library used for loading model data (Assimp), textures were not being loaded as expected as seen in figure 68

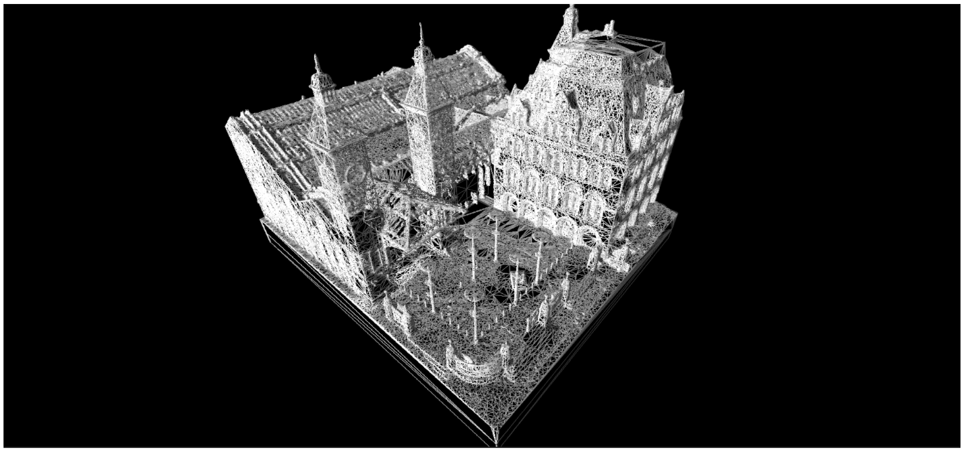


Figure 68: Missing encrypted textures

This will have to be addressed by further evaluating the usage of Assimp to better understand which features it does and does not support. Additionally, VMVE should transition to use a newer asset file format such as glTF which supports packing texture data into a single file.

5.1.2 Design

Having concluded the project, it has shown how important the requirements gathering and design stages are. During the implementation of the rendering architecture, it took much longer than expected as certain aspects were not fully defined and designed during the design stages. This resulted in having to redesign the architecture mid way and thus, delaying other aspects of the project.

However, it's also important to recognise that in large projects such as this, it can be difficult to fully define all the requirements as they may change over time. Therefore, this needs to be taken into consideration for design stages and have alternative solutions in future projects.

5.1.3 Time Management

One of the major downsides that the project suffers from is the projects time constraints. Given the relative complexity of the project, many of the features implemented in VMVE are quite rudimentary and serve as a basic prototype that showcases the underlying technology and the potential future work that can be undertaken.

5.1.4 Alternative Rendering API

Vulkan which as previously discussed is a low-level GPU API. One of the major downsides to having chosen Vulkan as the API of choice for this project was the significant amount of time that had to be spent implementing the various rendering features due to how verbose and technical the API is. The use of the API is only really beneficial if a developer wants to make full use of the additional performance through the use of multithreading, multiple command buffers and very specific memory allocation requirements.

If given the choice, VMVE would be rewritten to use a simpler API such as Direct3D or OpenGL as they provide the same functionality but with significantly less work required. This would allow me to focus and spend more time implementing the many other features required in such an application. In the future, if the renderer's performance becomes an issue then the use of more advanced APIs could be reconsidered.

5.2 Future Work

Looking forward to the future, several features need to be worked on and implemented that aim to greatly increase VMVEs usability and provide a whole host of new features that benefit the key stakeholder groups.

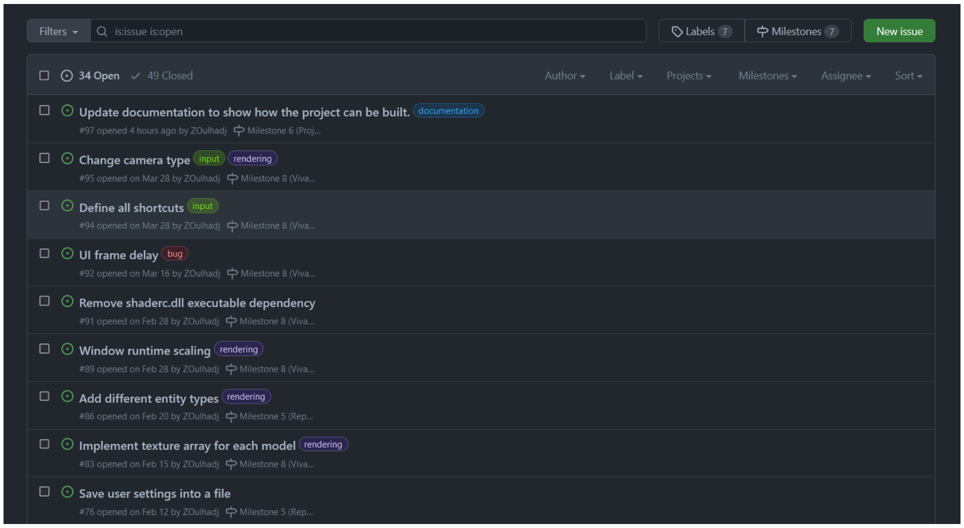


Figure 69: Outstanding GitHub issues

The feature list provides a basic outline of the improvements that can be made. For a detailed and in-depth discussion of the various features that will need to be implemented in the future, see section H of the appendix.

Multiple rendering APIs Currently, Vulkan is the only supported rendering API. The application should be able to support the other APIs such as OpenGL and Direct3D for greater platform support and user reach.

Renderer optimisations How the renderer system has been implemented does not scale appropriately as the complexity of the virtual environment increases. This results in poorer performance. To solve this, various other optimisations can be implemented such as frustum culling and spatial acceleration structures for large-scale terrains.

glTF model loading As a new model format, it has several benefits over current formats such as "obj" and "fbx". Supporting this would decrease model loading times and allow for assets to be packed more efficiently both on disk and in memory.

User interface accessibility The usability of VMVE can be further improved by making the application accessible such as by using text-to-speech, improving the usage of colors and also supporting color blindness.

Version control The other major improvement that can be made is the architecture of the GitHub repository. To ensure that the project can scale better in the future, an additional branch called "Beta" could be implemented so that users can gain early access to new features.

6 References

[1] Epic Games, "Unreal engine." [Online]. Available: https://www.unrealengine.com/en-US/

[2] Unity Technologies, "Unity." [Online]. Available: https://unity.com/

[3] Pixar, "Renderman." [Online]. Available: https://renderman.pixar.com/

[4] Keza MacDonald, Keith Stuart and Alex Hern, 9 2022. [Online]. Available: https://www.theguardian.com/games/2022/sep/19/grand-theft-auto-6-leak-who-hacked-rockstar-and-what-was-stolen

[5] "The mit license." [Online]. Available: https://opensource.org/license/mit/

[6] StackOverflow, "Stackoverflow developer survey," 2022. [Online]. Available: https://survey.stackoverflow.co/2022/#section-version-control-version-control-systems

[7] Microsoft, "Microsoft visual studio 2022." [Online]. Available: https://visualstudio.microsoft.com/

[8] "Git version control system." [Online]. Available: https://git-scm.com/

[9] Embarcadero, "Visual Studio Assist X." [Online]. Available: https://www.wholetomato.com/

[10] B. Karlsson, "Renderdoc." [Online]. Available: https://renderdoc.org/

[11] "Amd radeon gpu profiler." [Online]. Available: https://gpuopen.com/rgp/

[12] The Khronos Vulkan Working Group, "Vulkan." [Online]. Available: https://www.vulkan.org/

[13] A. Cornut, "Imgui," 2018. [Online]. Available: https://github.com/ocornut/imgui/

[14] W. Dai, "Crypto++ library." [Online]. Available: https://www.cryptopp.com/

[15] W. S. Grant and R. Voorhies, "cereal - a c++11 library for serialization," 2017. [Online]. Available: https://uscilab.github.io/cereal/

[16] T. Girard, M. M. Anitsal, and I. Anitsal, "The role of logos in building brand awareness and performance: Implications for entrepreneurs," *The Entrepreneurial Executive*, vol. 18, p. 7, 2013.

[17] J. Adam, "Agile development life cycle," 11 2022. [Online]. Available: https://kruschecompany.com/agile-software-development/

[18] B. Stroustrup, "C++ core guidelines," 4 2023. [Online]. Available: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

[19] D. Straker, *C. Style*. London, England: Prentice-Hall, Nov. 1991.

[20] A. Kapoulkine, "Vulkan loader," 4 2018. [Online]. Available: https://gpuopen.com/learn/reducing-vulkan-api-call-overhead/

[21] The Khronos Vulkan Working Group, "Vulkan 1.3 specification," 5 2023. [Online]. Available: https://registry.khronos.org/vulkan/specs/1.3/html/vkspec.html

[22] G. Oster and Y. Nishijima, "MoirÉ patterns," *Scientific American*, vol. 208, no. 5, pp. 54–63, 1963. [Online]. Available: http://www.jstor.org/stable/24936147

[23] J. de Vries, "Texture mipmapping," 6 2014. [Online]. Available: https://learnopengl.com/Getting-started/Textures

[24] D. Salomon, *Transformations and Projections in Computer Graphics*. Springer London, 2007. [Online]. Available: https://books.google.co.uk/books?id=6CzuydFlMpwC

[25] J. de Vries, "Coordinate systems," 6 2014. [Online]. Available: https://learnopengl.com/Getting-started/Coordinate-Systems

[26] J. Santell, "Model view projection," 4 2019. [Online]. Available: https://jsantell.com/model-view-projection/

[27] Bui Tuong Phong, "Illumination for Computer Generated Pictures," 1975, [accessed 12-December-2022]. [Online]. Available: https://users.cs.northwestern.edu/~ago820/cs395/Papers/Phong_1975.pdf

[28] "Font awesome." [Online]. Available: https://fontawesome.com/

[29] C. Guillemet, "Imguizmo." [Online]. Available: https://github.com/CedricGuillemet/ImGuizmo

[30] George Eckel, Renate Kempf, and Leif Wennerberg, "High-level strategic tools for fast rendering," 1998. [On-

line]. Available: https://techpubs.jurassic.nl/manuals/nt/developer/Optimizer_PG/sgi_html/ch05.html

[31] Mozilla Developer Network contributors, "Mdn web docs, 3d collision detection," 2021. [Online]. Available: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection

[32] The Khronos Group, "gltf runtime 3d asset delivery." [Online]. Available: https://www.khronos.org/gltf/

[33] "Cmake, open source and cross platform build system." [Online]. Available: https://cmake.org/

[34] "Premake - powerfully simple build configuration." [Online]. Available: https://premake.github.io/

[35] D. Byte, "Cheat engine." [Online]. Available: https://www.cheatengine.org/

7 Appendices

The appendices may be referenced throughout the report which provides additional information or access to important documents.

A Supporting Documents

Milestone 1 Project initiation document for milestone 1.

Milestone 2 Project proposal document for milestone 2.

Milestone 3 Mid-term project review document for milestone 3.

Milestone 3 Presentation A PowerPoint presentation as part of the mid-term project review.

Meetings A complete log of all meetings held during the project.

Source The source code to this latex document.

Note that in the event of your PDF viewer not supporting the attached files, the documents can be downloaded directly here

B Project Proposal Sign-Off

Student and First Supervisor Project Sign Off

STUDENT: I agree to completing this project:	X	**Date:**	10 / 11 / 22
Student Name:	Zakariya Oulhadj		
Student Signature:	Z.Oulhadj		
SUPERVISOR: I approve this project proposal:	X	**Date:**	10 / 11 / 2022
Supervisor Name:	Dr Charles Clarke		
Supervisor Signature:			

It is the supervisor's responsibility to approve this project as meeting the requirements for the module. This includes professional body requirements, programme requirements, and module requirements. By signing the form, you are agreeing that you have validated the suitability of the project.

Figure 70: Project proposal sign-off

C Project Directory Structure

```
/
├── engine/ .................. Engine project directory
│   ├── include/ ............. Engine public interface
│   ├── src/ ................. Engine source directory
│   │   ├── core/ ............ Window and Input systems
│   │   ├── events/ .......... Event system
│   │   ├── filesystem/ ...... Virtual file system
│   │   ├── rendering/ ....... Rendering
│   │   │   ├── api/ ......... GPU APIs
│   │   │   ├── shaders/ ..... GPU programs
│   │   │   └── ui/ .......... User Interface implementation
│   │   └── utils/ ........... Helper functionaility
│   └── vendor/ .............. Engine external libraries
└── vmve/ .................... VMVE project directory
    ├── assets/ .............. Model Assets
    ├── src/ ................. VMVE source directory
    │   └── ui/ .............. User Interface
    │       └── panels/ ...... User Interface components
    └── vendor/ .............. VMVE external libraries
```

D Version Control

A VCS is a tool used for backing up and/or collaborating with developers on a project. The use of a VCS was an obvious choice as this would provide a platform on which the project source code could be hosted. Furthermore, if for some reason a local copy of the project is lost or corrupted then another copy is safely hosted on the servers managed by the VCS.

Another key feature of a VCS is project management. These types of systems provide various tools that greatly benefit developers. One such feature is known as a "commit" which records any changes made to a particular repository at that moment in time. Developers use commits to view changes that occur at each stage but also, provide means of reverting to previous versions of particular sections, files or even an entire repository.

E GitHub

The VMVE project is hosted on GitHub as a private repository (accessible to supervisors) which can be found at the following link: https://github.com/ZOulhadj/vmve/. The repository also contains a guide on how to download and build the project.

F Development Logs

At key stages of the project's development, development logs i.e. videos were recorded showcasing the state of development at that specific point in time. All videos were uploaded and are hosted on the Zakariya Oulhadj YouTube channel https://www.youtube.com/@ZOulhadj

G Camera Transformation

```
// Construct M (model)
glm::mat4 m = glm::mat4(1.0f);
m = glm::translate(m, world_position);
m = glm::rotate(m, radians, rotation_axis);
m = glm::scale(m, scale_amount);

// Construct V (view)
glm::mat4 v = glm::lookAt(camera_position, camera_direction
, camera_up);

// Construct P (projection)
glm::mat4 p = glm::perspective(field_of_view, aspect_ratio,
 near, far);

// Construct MVP
glm::mat4 mvp = p * v * m; // Order is important
```

Figure 71: Complete MVP example

H Detailed Future Work

Multiple rendering APIs

VMVE currently only supports one rendering API which is Vulkan. As discussed in the technology review section, Vulkan is officially supported on Windows, Linux and macOS (through MoltenVK). However, to support additional operating systems as well as hardware that does not support Vulkan, more rendering APIs should be supported including DirectX12 and previous generation APIs such as OpenGL and DirectX11 for increased compatibility.

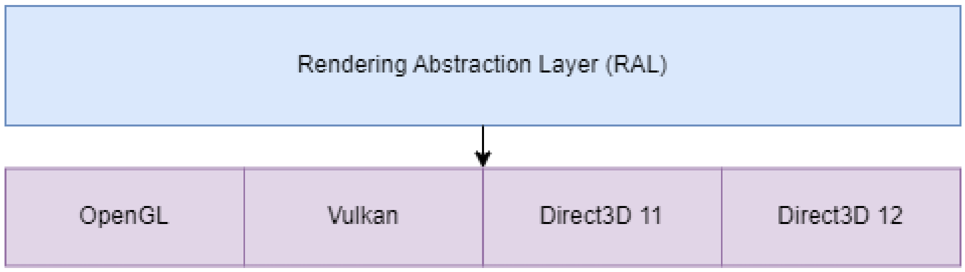


Figure 72: Rendering API Abstraction

Frustum Culling

Currently, VMVE sends all object data to the GPU to be processed and rendered. The GPU then subsequently traverses each vertex to figure out if it needs to be "discarded". This process is part of the graphics pipeline and occurs for each vertex. As the complexity of both the scene and the objects themselves increases, this starts to become a GPU intensive task and results in increased GPU usage and lower performance.

To solve this, frustum culling must be implemented which is a rendering optimisation in which objects not visible from the "cameras point of view" are discarded completely and not sent to the GPU entirely. The term "frustum" refers to the camera projection frustum which can be seen in figure 73 and "Culling" simply means discarding.

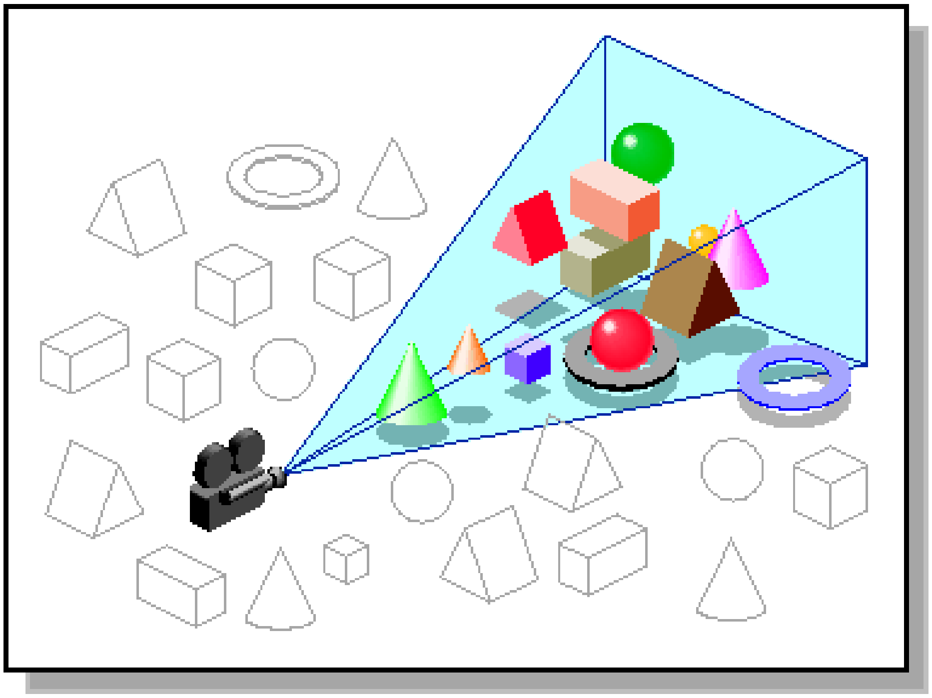


Figure 73: Frustum Culling[30]

This technique significantly improves performance as each object in the world is contained within a “bounding box” which is often a cube or a sphere 74. Then for each object, a check is performed against the bounded box instead of the actual vertices. This allows for in the best case a single check or at its worst 8 checks per object. This is far better than needing to perform thousands of checks for all the vertices of an object.

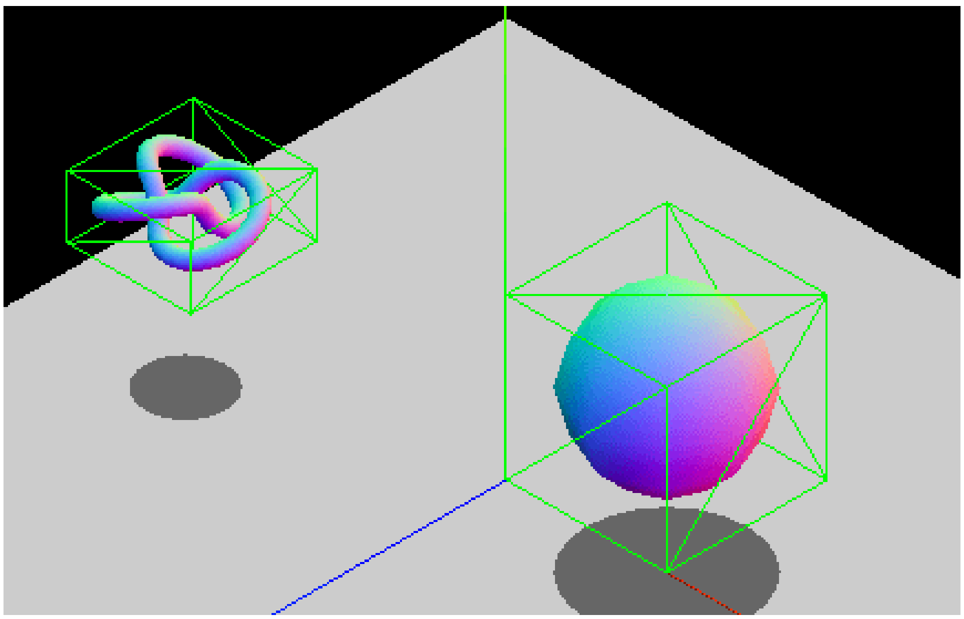


Figure 74: Bounding Boxes[31]

Spatial Acceleration Structures

VMVE only supports loading basic models however, in the future many other features need to be implemented. One such feature is large-scale terrain as seen in figure 75.

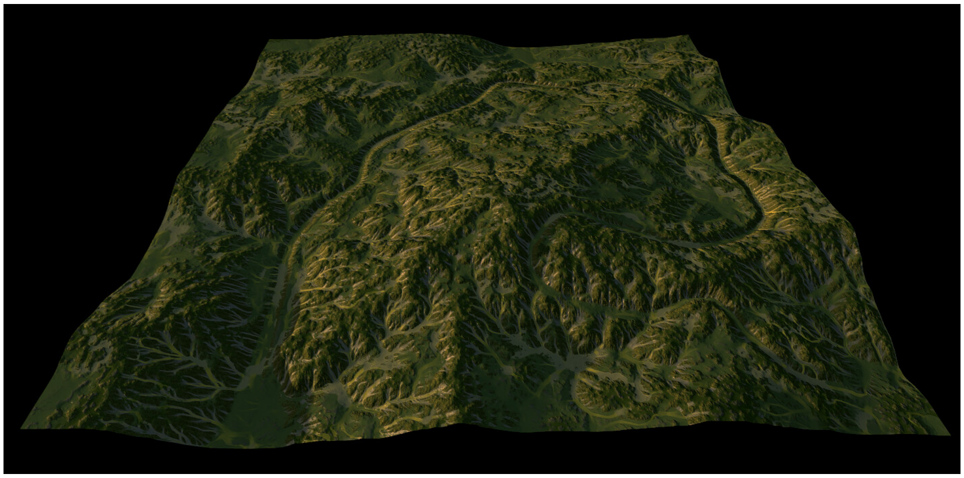


Figure 75: Large scale terrain

Currently, VMVE would struggle to render such objects due to their complexity in terms of the number of vertices required. For example, an area of 20x20km with a resolution of 1 meter would require 400 million vertices. Each vertex, if packed efficiently could be 32 bytes each that includes positions (12 bytes), normals (12 bytes) and texture coordinates (8 bytes). In terms of memory, this would require 12.8GB of data just for the terrain.

A common solution to this is implementing a type of spatial acceleration structure also known as Level of Detail (LOD). These structures are designed as the name suggests, to increase the speed of algorithms in the spatial domain including images and environments.

A quadtree is a type of spatial acceleration structure that stores a hierarchical collection of nodes that each represent a 2D area (and an octree in the case of 3D). Figure 76 visualises a quadtree that shows the resolution of each tile and how there are fewer tiles the further away from the camera they are.

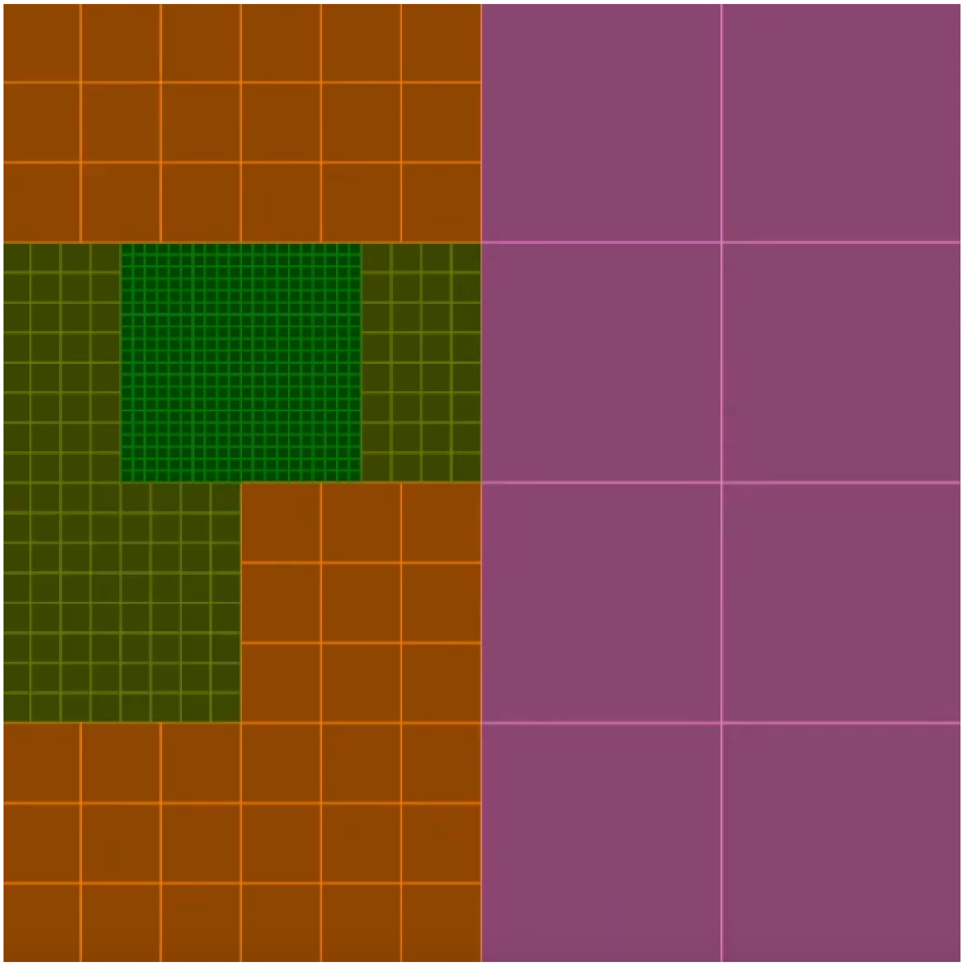


Figure 76: Quad Tree visualisation

The benefit of using this technique is that the number of vertices is significantly reduced saving on performance and memory usage.

glTF Support

Currently, the engine only really supports the VMVE and OBJ model formats for importing. The main issue with the OBJ file format is its size since the data is stored in a text format which subsequently increases the assets file size. glTF [32] is a new model format by The Khronos Group that has many useful features such as compression, binary representation, scene hierarchy and more.

Cross Platform

In contrast to the project's initial proposal which was the support multiple Operating Systems (OSs), Windows is the only OS that VMVE officially supports due to the time constraints for this project.

Going beyond the final deadline, VMVE should add cross-platform support which would greatly benefit the application as it would increase flexibility in terms of which OSs its able to run on and subsequently, increase the potential user base. Implementing such functionality would require significant changes to the underlying architecture of the application such as implementing opaque types for structures that have different implementations depending on the OS being used. Additionally, a different build system would have to be used that is cross-platform such as CMake[33] or Premake[34].

Networking Support

VMVE is a virtual environment editor and as such can be significantly improved by adding support for networking. The idea is that there would be a server running which would keep track of the virtual world and multiple clients via VMVE would be able to connect and interact with the environment simultaneously.

This would allow for greater efficiency and speed up various tasks. For example, multiple users could work together to construct a virtual environment in preparation for scientific research which would otherwise have taken much longer had only one user worked on it.

File Format/Encryption

VMVE supports both the AES encryption standard by default. The application should also include additional encryption algorithms such as Diffie-Helm and RC6. This would provide users with greater flexibility in terms of how their assets are encrypted as well as varying levels of encryption strength/methods based on the user's encryption requirements.

Another feature that will need to be added to the application is related to the custom model format and the ability to store and encrypt other types of data. Currently, the encrypted data stores just the vertices and their corresponding information. The downside to this is that assets such as textures are still required to be located alongside the ".vmve" file since the format does not support textures.

In the future, VMVE should be able to store textures so that users can be left with a single asset file that includes all the information, similar to how the glTF model format supports the ".glb" binary file. This would significantly increase the user's experience since assets files would become easier to manage.

Runtime memory protection

One of the major flaws present is the lack of security that prevents a user from maliciously altering the memory of the application during runtime so that encrypted models can be accessed without the need for a valid key or iv.

This can be achieved using applications such as Cheat Engine which provides users with a set of tools for analysing memory addresses, offsets and values that can also be manipulated [35]. For example, an attacker could alter the boolean statement that checks if the key and iv being inputted are valid. Doing so would force the application to go ahead and decrypt a model.

This will need to be addressed in future versions of VMVE so that digital assets remain secure for all users. A possible alternative implementation would involve the use of symmetric and asymmetric encryption as well as techniques designed specially to prevent runtime memory manipulation.

Version Control

In regards to the VCS architecture, a potential change that can be done is to add a third branch called "Beta". The purpose of this branch would be to release relatively stable yet unfinished features to users. This would allow users to use new features and test them before an official version is released. Overall, this would reduce the number of bugs in final versions and be beneficial to users who are keen on using recently developed features. Figure 77 shows how the VCS architecture could look like with the addition of a "Beta" branch.

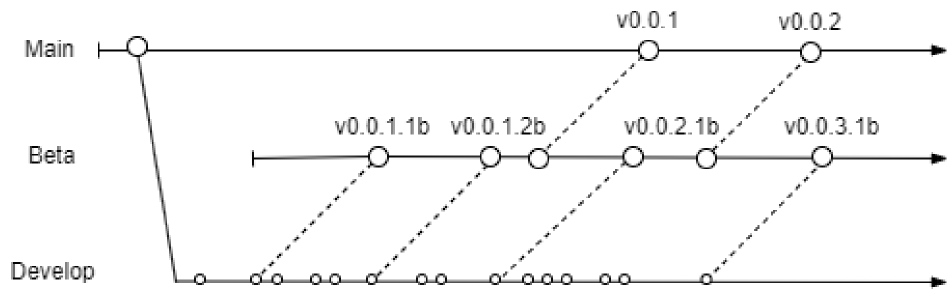


Figure 77: Potential future git branch design

Reduce Library Dependencies

VMVE makes use of several libraries which allows specific functionality to be added quickly. This is ideal given the project's time constraints and requirements. However, going forward the aim will be to reduce the number of external libraries being used.

This will provide various benefits such as less reliance on external code, faster compilation times as only functionality that is required will be implemented and generally finer control of the code that VMVE is built on top of.

Accessibility

Another aspect that needs to be addressed soon is improved accessibility. The application includes some features that address this such as icons, a tutorial, documentation as well as additional information. However, some of the features that VMVE does not include is taking into account color blindness and text-to-speech. The reason for this is that these are more complex topics and require additional time and research to ensure a proper implementation is released.

VMVE should also support localisation. Currently, only the English language is supported with no way of adapting to other languages. Therefore, the internal structure of the application must be redesigned to allow for other languages to be added and thus, improving accessibility.