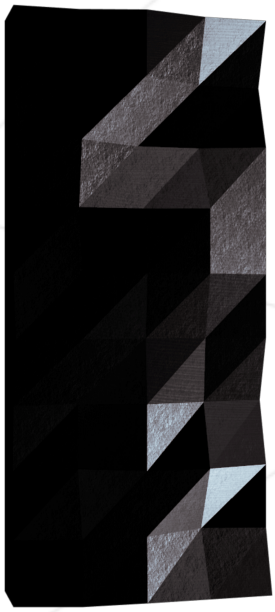
β

This project is currently under beta-test, until it is validated by a fair number of students in the 42 community. Please report any typo, incoherence, inconsistency, error, using the form https://tally.so/r/3lVDJo



Very Real Engine

Complete 3D Vulkan Engine

Summary: Congratulations, you've reach a point where you're able to create your own engine ! Let's make this.

Version: 1.3

Contents

Chapter I

Preamble

Back in 1992, a revolution occurred in the video game industry with the release of `Wolfenstein 3D` by id Software, heralding the start of the era of first-person shooter (FPS) games. The colossal success of `Wolfenstein 3D` paved the way for other iconic titles such as `DOOM` and `Quake`, turning id Software into a household name in the gaming industry. Not only did these games set standards for future FPS titles, but they also introduced technological advancements in the field of real-time graphic rendering.

Over the years, id Software continued to push the boundaries of gaming technology, introducing concepts such as 3D rendering, dynamic lighting, and normal maps to enhance the visual realism of games. With the launch of `Quake III Arena`, id Software introduced a new graphics technology, id Tech 3, which was a massive step forward in terms of graphical capabilities, notably introducing the curved surface rendering technique.

Parallel to id Software's evolution, another revolution was taking place in the game engine world. In 1998, Epic Games launched Unreal Engine, a game engine capable of creating rich and immersive 3D environments with unprecedented flexibility for game developers. Unreal Engine introduced innovative features such as the visual Blueprint scripting system and support for next-generation graphics. This enabled developers to create high-quality games with less time and effort.

Today, both Unreal Engine and id Software continue to be at the forefront of technological innovation in the video game field. They have set the standards for modern gaming and made the creation of increasingly immersive and realistic virtual worlds possible.

In this project, you will embark on the adventure of creating a game engine, drawing inspiration from the pioneers of the video game industry. By tackling this challenge, you will have the opportunity to understand and implement the concepts and techniques that have shaped the history of video games. It's now up to you to write your own chapter in this history. Happy coding!

Chapter II

Objective

The primary goal of this project is to design and implement a library, called `VeryRealEngine`, that serves as a foundation for video game development. This library should encapsulate all the tools necessary to create modern video games, most notably a graphics engine and a physics engine.

The graphics engine is the heart of your game engine and is responsible for rendering the game environment. This includes, but is not limited to, displaying complex 3D models, managing lighting and shadows, and creating realistic and dynamic environments. It will require a deep understanding of computer graphics principles and the Vulkan API to accomplish this.

The physics engine is another critical component of your game engine. This engine should be capable of simulating the laws of physics to create believable interactions between game objects. It will handle tasks like collision detection, rigid body dynamics, and any additional physics-related features you may add.

In addition to these components, you are encouraged to include any additional features that you deem necessary for future game development. This could include systems for handling user input, artificial intelligence, networking, sound, and more.

Remember, the objective here is not just to create a game engine but to create a toolkit that you can use for your future ventures in video game development. It's not just about building a game but building the tools that let you make any game you can imagine.

Chapter III

General Instructions

- This project will be graded by humans only. You are allowed to organize and name your files as you see fit, but you must follow the rules outlined here.
- The final product must be a library named `VeryRealEngine`.
- Your `Makefile` or equivalent (e.g., CMake) must compile the library and must contain the usual rules. It must recompile and re-link the library only when necessary.
- You must handle errors carefully. Under no circumstances should your program terminate unexpectedly (segmentation fault, bus error, double free, etc.).
- Your library must not have memory leaks.
- You are strictly forbidden to use ANY non-system library, with the obvious exception of Vulkan.
  You are authorized to use system libraries such as XCB for Linux or the Windows-API library.
- The core components of your engine, such as the rendering engine, physics engine, and entity-component system, must be implemented by you.
- If you deem it necessary for the bonus part of the project or for any additional features, you may use other functions or even other libraries, as long as you can justify their use during your evaluation. Be smart!
- You must create a small demonstration game or 3D animation to showcase the capabilities of your game engine. This demonstration must be fully functional and utilize various features of your engine.
- Your project submission must include extensive documentation of the library. This can be in the form of code comments, README files, and dedicated documentation files.

Chapter IV

Mandatory Part

The following features are the core objectives for your engine:

IV.1 Engine Basics

- **Basics:**
  Your engine must be able to load a "scene" via a file that describes which objects must be loaded and where they should be placed.

You may create your own custom file type for this, or use JSON, for example.

- **3D Rendering:**
  Your engine must be capable of rendering 3D objects. It should be able to load and display .OBJ files correctly.

We consider that rendering 3D objects refers to both textured and non-textured objects. You should also be able to load multiple OBJ files at the same time.

- **Physics Simulation:**
  Your engine should be capable of handling physics simulations. This includes, but is not limited to, collision detection and response, gravity, and friction.

This also includes the "trigger" type of collision detection, where an object is supposed to detect the collision but not block it.

- **Scene Hierarchy:**
  Develop a system for managing scene hierarchy. This includes handling multiple objects, their transformations, and their parent-child relationships.

During the evaluation, you will be required to "hide" an object and all its children simultaneously by editing only the parent.

- **Optimization:**
  Your engine must be able to efficiently handle large numbers of objects. It should utilize optimization techniques such as Frustum Culling and Occlusion Culling to maintain high frame rates.

You must achieve at least 60 FPS when you run your program in Release mode.

- **Materials and Textures:**
  Implement a system for materials and textures. The engine should support different types of materials, such as wood-like textures, metallic ones, etc. It should also be able to map textures onto these materials.

IV.2 Engine composition

- **Lighting:**
  Your engine must be able to handle multiple light sources. Objects should dynamically cast shadows based on the position and intensity of these lights.

- **Shadow Rendering:**
  Implement a system for both static and dynamic shadow rendering. Shadows should be cast realistically based on the objects' shapes and the light sources in the scene.

- **Scene Loading:**
  Implement a system that can interpret JSON or similar formats for scene loading. This includes loading scenes, changing object properties, and managing light sources, among other functionalities.

Chapter V

Demonstration

The primary objective of this project is to build a game engine, and it is critical to demonstrate its capabilities. Therefore, as a final requirement, you must create a specific 3D demonstration scene: a detailed house environment. This demonstration is an opportunity to showcase your engine's features, such as the rendering engine, the physics engine, and the entity-component system. Your demonstration must take full advantage of your engine's capabilities and showcase its functionality in real-time execution.

V.1 Interactive House Environment

For this demonstration, you are to create an interactive 3D house environment. This environment should be more than a mere static visual model. It should include:

- Various rooms with different lighting conditions: Some rooms should be brightly lit, while others should be darker. This will demonstrate your engine's lighting and shadow rendering capabilities.
- Interactable objects within the house: This might include doors that can open and close, light switches that change the room's lighting when toggled, or other items that can be manipulated in some way. These will demonstrate your physics engine and the interactivity of your engine.
- Particle-emitting objects: For instance, a coffee machine that emits steam, demonstrating your engine's particle system.

Ensure that navigation through the house is smooth, realistic, and intuitive. This environment will act as tangible proof of the effectiveness and capabilities of your game engine. It will allow anyone to step into a world created by your VeryRealEngine.

V.2 Illustrative Examples

To help you better understand what is expected in this demonstration, here are some illustrative examples:

Figure V.1: Architecture Demonstration A

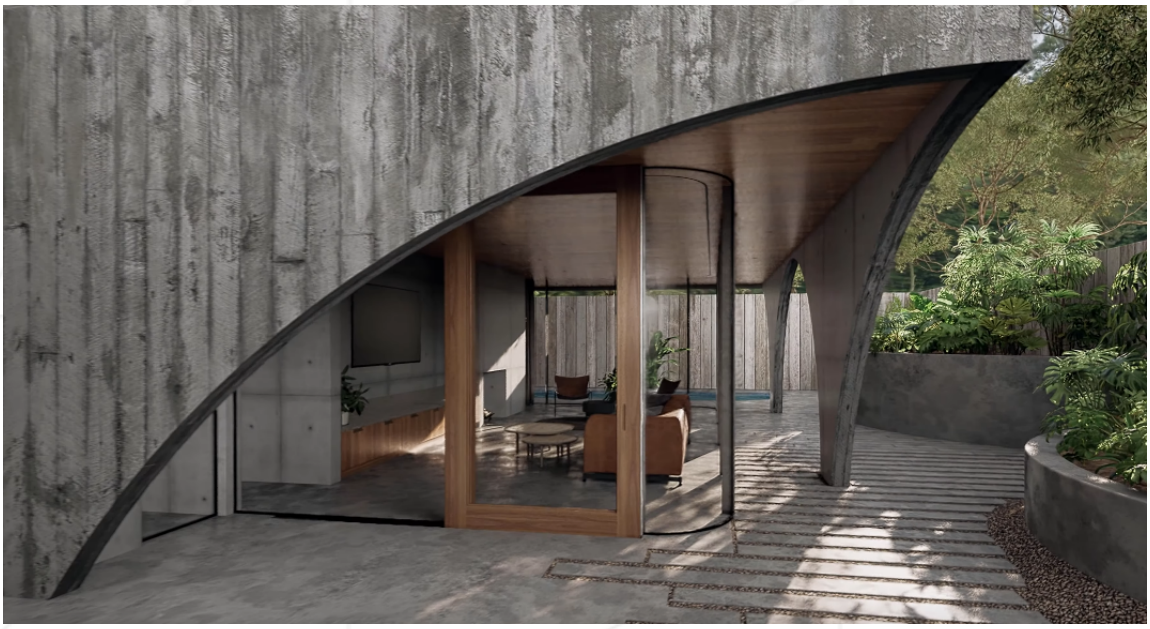


Figure V.2: Architecture Demonstration B

Figure V.3: Architecture Demonstration C

Figure V.4: Architecture Demonstration D

Chapter VI

Documentation

Correctly documenting your code is a crucial aspect of software development. It enables other developers (or even your future self) to understand what the code does and how to use it. This project requires comprehensive code documentation to ensure maintainability and ease of use.

VI.1 Doxygen

For this project, you will use Doxygen, a tool that generates documentation from annotated source code. Doxygen can be used with various programming languages, including C, C++, Java, Python, and many others. It creates documentation in several formats, such as HTML, LaTeX, and RTF.

Doxygen supports Markdown, allowing you to write more detailed and structured documentation. You can also link to related parts of your documentation using Doxygen's special commands.

If you'd like to use a language that isn't supported by Doxygen, you can consider alternative tools, such as Cargo Docs for Rust. Just ensure that you generate complete documentation from the source code!

VI.2 Warnings

Your Doxygen documentation generation must be free of warnings. Be sure to check the output of Doxygen each time you generate your documentation and resolve any warnings that may appear.

This may involve going back and fixing Doxygen comments in your code or ensuring that your code is arranged in a way that allows Doxygen to correctly generate the documentation.

Chapter VII

Bonus Part

Although not mandatory, the following features will enhance the capabilities of your engine and are recommended:

- **Post-processing:**
  Implement post-processing effects such as bloom, motion blur, and depth of field.
- **Sound System:**
  Implement a sound system to play background music and sound effects.
- **Particle Systems:**
  Support particle systems for creating effects like fire, smoke, and magic spells.
- **Skeletal Animation:**
  Support 2D/3D skeletal animation.
- **Networking:**
  Develop a networking module for multiplayer capabilities.

`If your group decides to work on bonuses, you must ensure that they are specifically located within the bonus files.`

Remember that the main goal of this project is to create a functional and optimized 3D game engine.
However, don't forget to have fun in the process.
Happy coding!

Chapter VIII

Submission and Peer Evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double-check the names of your folders and files to ensure they are correct.

VIII.1 Compiling from a Fresh Repository

It's essential that both your library and demonstration can be compiled and executed from a fresh repository clone. This means that all necessary files and dependencies should be included in the repository or be clearly listed in a README file with instructions on how to install them.

Additionally, ensure that your repository includes a build script, makefile, or detailed instructions in the README on how to build your project from source. This enables the reviewer to easily compile and run your code.

VIII.2 Doxygen Documentation Generation

During the evaluation process, your Doxygen documentation will be generated and briefly reviewed for the quality of comments. Ensure that your code is properly commented and that running Doxygen on your repository doesn't produce any warnings.

Remember that good documentation comments not only describe what the code is doing, but also explain why it's doing it that way, what the inputs and outputs are, and any other context that would help a new reader understand the code. Doxygen comments should be present for all classes, functions, methods, and variables in your code.

You should include a script or detailed instructions in your repository on how to generate the Doxygen documentation. This allows reviewers to easily generate the latest version of the documentation based on your code.

VIII.3 Peer Review

Remember that your work will also be subject to peer review. Ensure that your code is readable, well-commented, and follows the agreed-upon coding standards for your project. This will not only aid in understanding your code but also assist in the review process. Your peers will appreciate clear, concise code and thorough documentation.

Submitting your work with these points in mind will increase the quality of your code, the readability of your documentation, and the efficiency of the review process.