RED ENGINE REPORT

Daniel Hance s5406182

Contents

Introduction 2
Specification 2
Research 3
Engine Design 4
Core Architecture 5
ECS 6
Game Loop 7
Engine Components 8
Resource Caching 9
Analysis and Evaluation 10
Conclusion 11
References 11

Introduction

The goal of this project is to develop a lightweight 3D game engine that is capable of rendering a scene, handling user inputs, and using components to implement basic game logic. By completion the engine should be able to be use as a platform to develop basic 3D games for both windows and web-based platforms. As part of this project, I will also develop a simple 3D game using the engine to demonstrate its features.

Specification

This is the specification for the project:

1. Graphics:
    - Use the rend OpenGL wrapper library for basic shaders and rendering model meshes with texture mapping.
    - Implement a camera system for viewpoint manipulation.
    - Use SDL to create the window for graphics to be displayed on.
2. User Input:
    - Use SDL to get input events.
    - Develop a system for handling keyboard and mouse inputs.
3. Physics and Collision:
    - Implement a basic collision system that can handle collisions between different types of colliders including both box and mesh.
    - Use the Pellet (Bullet) physics library to handle collision response and basic game physics.
4. Entity-Component system (ECS):
    - Utilize a ECS system for the engine's architecture.
    - Implement basic components into the engine such as a transform, model renderer and rigid body.
5. Audio:
    - Use OpenAL API for audio
    - Have both audio sources and listeners within engine
6. GUI
    - Implement an Immediate Mode GUI system.
    - GUI elements should include buttons with logic
7. Asset Management:
    - Implement engine resources to separate the engine from the other libraries.
    - Include a caching system for resources.
    - Support common file types.
8. Platform:
    - Have the engine support building to both windows and web-based platforms.
9. Version control:
    - Use git for version control to track changes made to the engine in development.
    - Push commits with clear names.

10. Documentation:
    - Add clear comments to code base to make it easy to understand.
    - Use doxygen to generate supporting documentation.

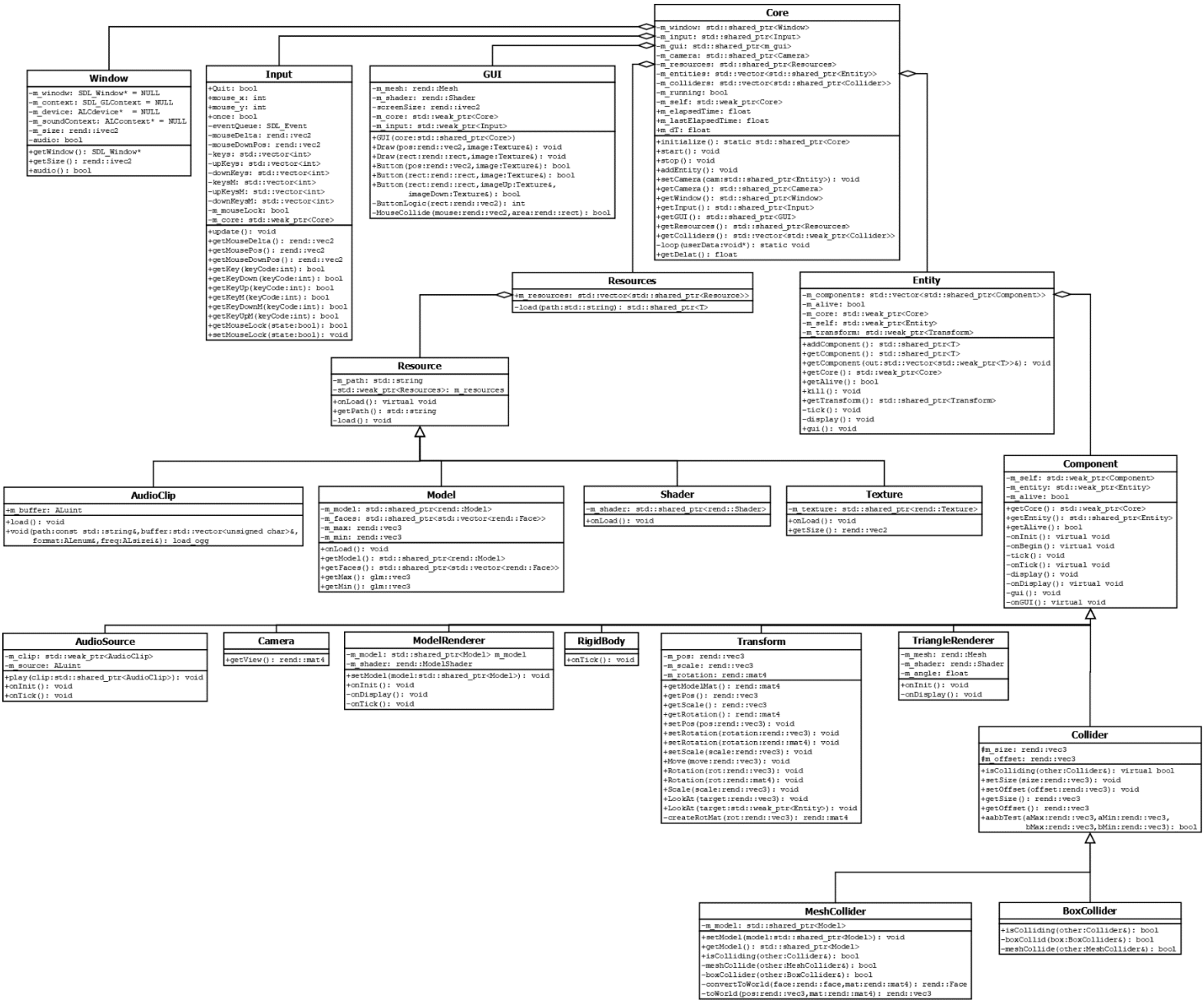
Research

One of the features researched in the development of the engine was the tri-tri collision detection that would be needed within the mesh collider component. As tri-tri collision is complex and can be slow I decided to use an existing c++ implementation[1] by Tomas Moller, this code is very efficient at calculating the collision check as it uses a large number of macros to optimise performance.

I also needed to do a large amount of research on OpenAL as this was my first time using the library, this led to me reading a large portion of the OpenAL programmers guide[2] to learn how to use the library and look up what parameters the functions required.

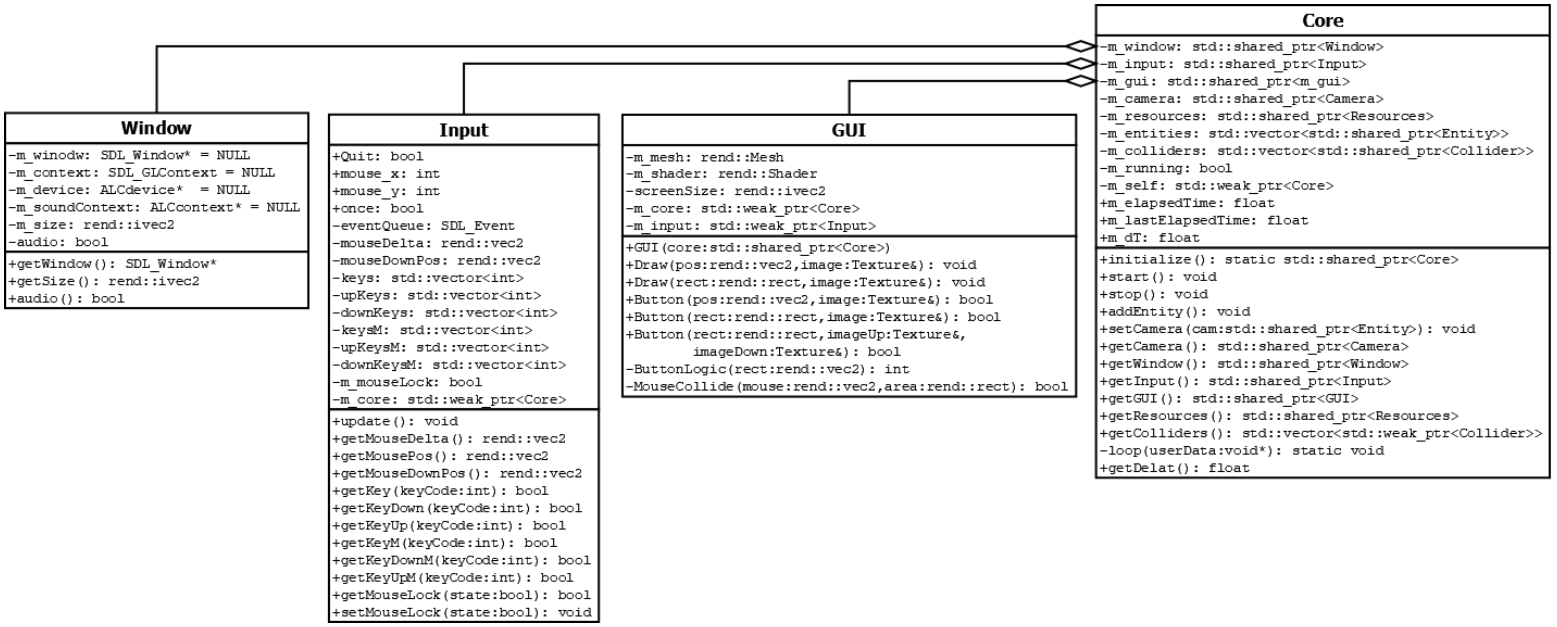
Engine Design

Red Engine was developed in c++ using CMake, allowing it to build games for both Windows and web platforms. The engine uses a modular and hierarchical structure to split the objects into groups making it easy to add more elements and giving it a simple architecture. A modified version of the rend OpenGL wrapper library handles the graphics while SDL is used to create the window and get input events. OpenAL manages the audio and Pellet is implemented but unused, below is a UML diagram for the whole engine:



Core Architecture

The centre of the engine is the core object, core's primary purpose is to store smart pointers to most major systems within Red Engine to allow data to be directed between objects in a memory safe manner, it also houses the game loop controlling what is happening within the engine. Core also has three attached objects that act as an extension of itself for handling specific engine functionalities, Window holds all the SDL, OpenGL and OpenAL initialization code as well as data relating to the window. Input handles all inputs in the engine, storing them in vectors for other engine elements to utilize, 'GUI' has all the GUI related functions including button logic as Red Engine uses immediate mode GUI.

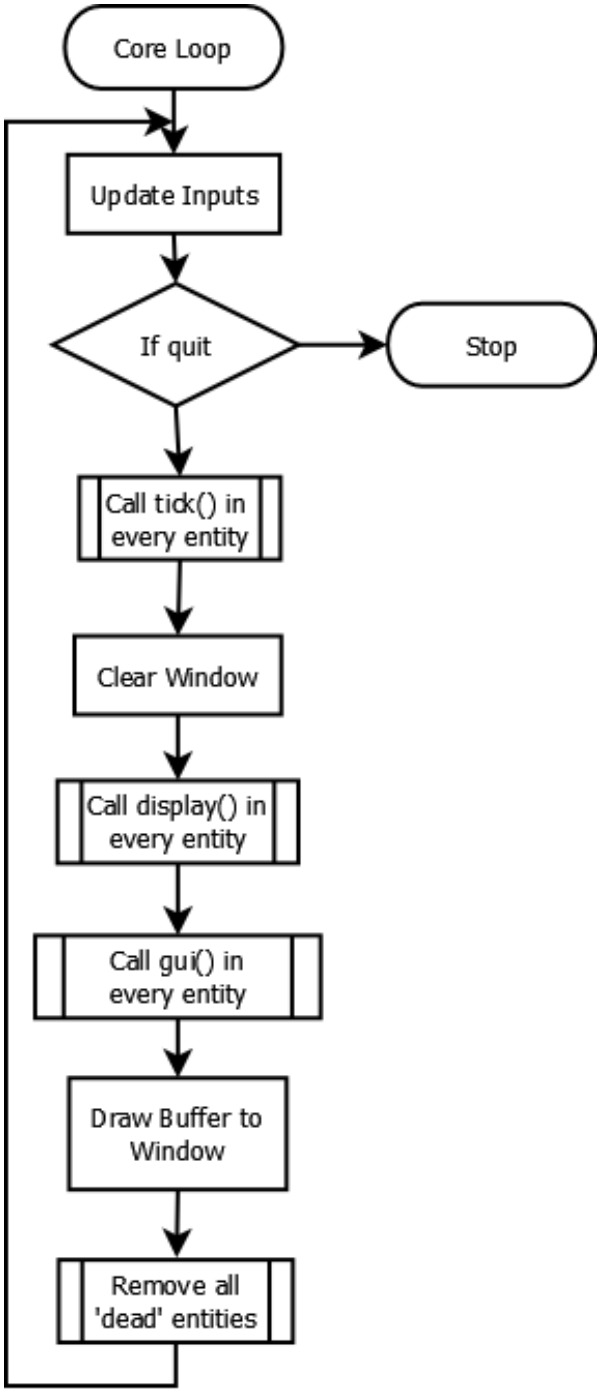


ECS

Red Engine adopts an entity component system to split up the game data and functionality to make it easy for the user to develop games. components are the bottom level, here is where all the game functionality is held, each component fulfils a specific role such as transforms or collision responses. One level higher are entities, these are what would be considered game objects (e.g. Player, Car), entities themselves have no functionality but attach appropriate components to obtain the functionality required. The entity simply holds pointers to all its components so it can call their functions. The top level is the core, as stated already it holds the game loop and will tell each entity when to run their component's functions, core also stores vectors of certain components like colliders to streamline data flow.

Game Loop

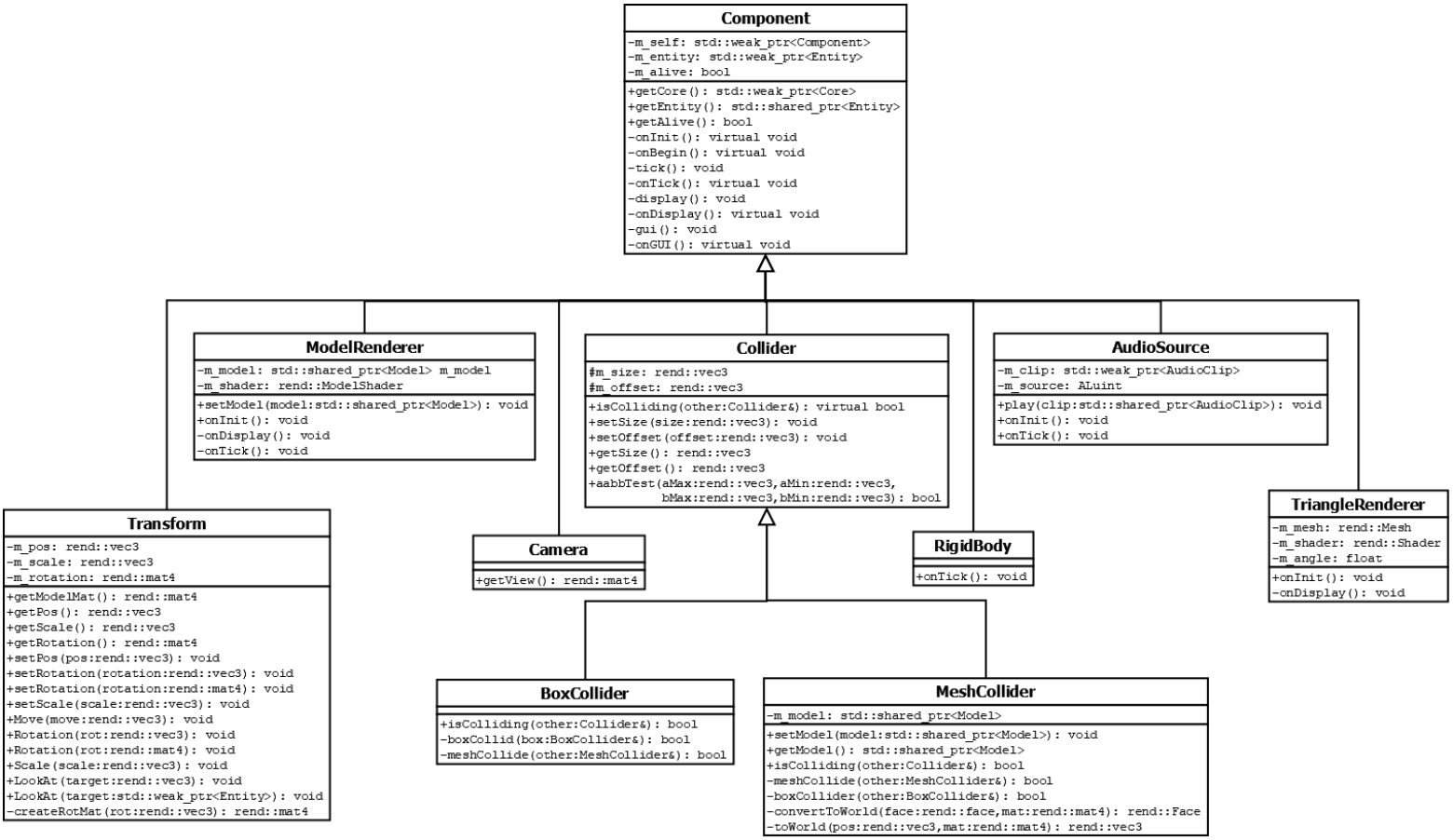
The game loop within core is relatively straightforward, it mainly runs through all the entities attached to core and calls their functions, alongside this both the window and the input vectors are updated. The game loop ends when a control quit flag is set to false, this can be triggered by calling stop() in core or the Input object receiving the SDL quit event.



Engine Components

All components within Red Engine inherit from the component class, it houses multiple public virtual functions, these functions are called by the entity a component is attached to. This design enables the user to make their own components with unique functionality allowing for a large range of games to be developed with the engine. Red Engine also contains some pre-made components for commonly used game mechanics such as renderers, rigid bodies and sounds.

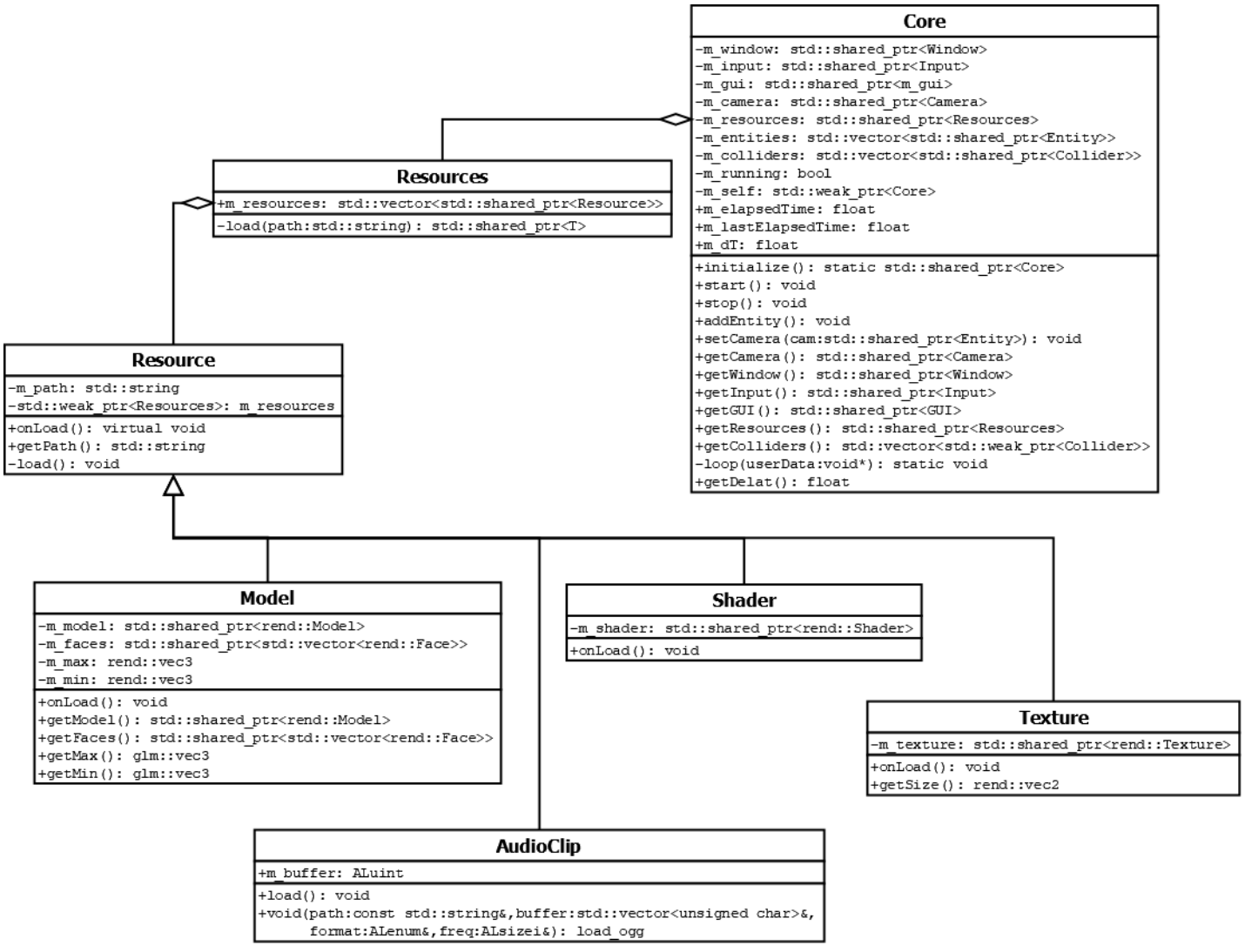
Among the pre-made components two are special, transform handles an objects movement, as this is so widely used it is automatically added to every entity and there are multiple shortcut functions within the engine to simplify accessing transform data. Collider is the second special component as it functions as a parent class for different types of collider components such as mesh and box, it also allows the user to create their own collider components by using virtual collider functions. All colliders are added to a vector within core to make iterating through them simpler.



Resource Caching

Red Engine uses an advanced resource management system designed to minimize the interaction between the engine and the rend library while optimizing memory usage, resulting in faster load times. All resources inherit from a resource class and use its virtual load function enabling them all to be called the same way. The individual resources access data from the rend OpenGL wrapper library or handle raw OpenAL, converting them into a Red Engine resource type. This abstraction of lower-level concepts both simplifies development and ensures memory safety by providing access to all data through Red Engine using smart pointers.

In addition to the individual resources, the engine also has a Resources class which acts as a type of cache memory. When a resource is first loaded a smart pointer to it is stored within a vector in resources, if core requests that resource a second time resources will pass the smart pointer rather than load it again. There are multiple advantages to using this method, primarily being the lower memory usage as if a game needs a large amount of a single resource (e.g. a coin model) only a single instant of it needs to be stored in memory. Additionally, it shortens load times as if a resource is already cached a new instance of it can be loaded instantly.



Analysis and Evaluation

Overall, the finished engine has many of the features listed in the specification with a small number missing or falling slightly short, however there are also additional features implemented within the engine that were not initially specified. This is how the developed engine matches up to the specification:

1. Graphics:

The engine is able to handle basic 3D graphics rendering including texture mapping on models and can choose to use multiple shaders for rendering meting the requirements in the specification. Multi camera functionality has been implemented which allows the user to pick which camera in a scene should be used as the viewMatrix, this is a bonus feature that can let the engine be used to develop more advanced games.

2. User Input:

The user input system developed for the engine is one of its major strengths, the input system can handle both keyboard and mouse inputs as well as adding a system which turns the mouse movement into an input. This can then be used to rotate a camera while still letting the mouse be used for GUI when a menu is open providing more option to the user when developing games.

3. Physics:

The engine falls somewhat short of the planned physics implementation, while the Pellet library is included within the engine's project file it is not currently being used, this effects the collision response as a much simpler algorithm is used instead. On the other hand, the colliders themselves are fully functional but are a key limitation in the engine's overall performance, the problem is that the mesh collider will always run through every face every time. This limitation can be solved by adding spatial partitioning as well as running a box-box collision check on mesh colliders before running through all the faces.

4. Entity – Component System (ECS):

The entity component system has been implemented as planned.

5. Audio:

The audio has been implemented as planned.

6. GUI

GUI is a strong element of the engine as not only is the Immediate Mode system been successfully implemented but the buttons have been given added functionality. Rather than just be clickable the buttons can also change their texture when held giving some visual feedback to the user and making the GUI system fill more responsive overall.

7. Asset Management:

Asset management is one of the engines key strengths as the resource caching system massively improves memory usage and load times.

8. Platform:

Support for both windows and web platforms is fully functioning, however sometimes web versions lose model textures but this is rare so it is not a major priority.

9. Version Control:

Version control has been used throughout development, commits have been given descriptive names.

10. Documentation

Documentation is currently the weakest element of the engine as doxygen was never implemented, while the code still has a large number of manual comments the lack of supporting documents to go with this make the engine difficult to use for people that are new to development within it. This problem can be fix by implementing doxygen and generating the needed documentation.

Conclusion

In summary, Red Engine is in a healthy state and has met a large amount of the specification while adding additional elements in some areas. In its current state it can support the development of basic 3D games with graphics, sound, collision, GUI and basic physics, which was the initial objective for its development.

Going forward the main engine element that needs working on is the documentation as there is very little right now, this can be done by implementing doxygen like was stated in the initial specification. After this improved physics can be developed, well this is a complex task it has been made somewhat simpler by Pellet already being included within the engine's CMake list. Finaly spatial partitioning can be added as the collision checking is currently the largest engine performance bottleneck.

References

- [1] - Muller, T. (no date) *Home page of Tomas Akenine-MuLler*. Available at: https://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-Moller/code/ (Accessed: 12 January 2024).
- [2] - *Documentation* (no date) *OpenAL*. Available at: https://www.openal.org/documentation/ (Accessed: 12 January 2024).
- Game Map - MaxDeaconVRFollow (1966) *Low poly game level - download free 3D model by maxdeaconvr*, *Sketchfab*. Available at: https://sketchfab.com/3d-models/low-poly-game-level-82b7a937ae504cfa9f277d9bf6874ad2 (Accessed: 12 January 2024).
- Game Models – From Half Life, models download from Brightspace
- Rend Library – Downloaded from Brightspace
- Horn Sound Effect – Download from Brightspace