Cerberus Engine

Conn Finnegan

Bournemouth University

Author Note

GEP – Conn Finnegan – s5403979

Cerberus Engine GitHub

Abstract

The aim of this project was to create a portable and lightweight game engine to demonstrate my capability in creating a secure and functional game engine.

The project contains the following features:

- **ECS (Entity Component System)**
- **Resource Management**
- **Audio Functionality**
- **Collision system**
- **Rendering System**
- **Secure Engine Core**
- **GitHub Version Control**
- **Doxygen Documentation**
- **CMake Build System**

Instructions for project prerequisites and build configuration can be found on the GitHub page within the README file. This project employs multiple external libraries and tools to provide standard functionality. These include:

- OpenGL

- GLFW

- SDL2

- SDL2_mixer

- GLAD

Entity Component System:

The main feature of this program is to use an entity component system. An ECS if commonly used in the games industry as a way of separating data from behaviour. This allows for code reusability and follows the composition over inheritance principle.

An ECS like this one consists of the following components:

- *Unique id's called entities*
- *Plain datatypes with no behaviour called components.*
- *Systems defined as functions, paired with entities that in themselves have a set of components.*
- *Entities don't have to have components*
- *Entities can change components during runtime*

Advantages:

- *Creates shorter less complicated code*
- *Design that employs decoupling, encapsulation, modularisation and reusability*
- *Allows people with less experience script based of behaviour*
- *Separates data from functions*

Disadvantages:

-It is a fluid system and therefore has no clearly defined structure so moving between projects may become confusing

-Requires the writing of many small systems. This creates the risk of writing inefficient code

Resource Management System:

This project incorporates a resource management system. This is used for efficiently loading, organising, using and unloading resources such as shaders and audio files. In ensures that resources are reused efficiently to save memory and CPU processing time.

Advantages:

- Memory Efficiency, avoids loading duplicate data therefore saving memory.
- Improved performance, resources loaded only when needed and unloaded when no longer required.
- Easy development, devs can focus of high-level logic and design without worrying about memory management.
- Scaling, as the program grows and features more complex elements the resource management system scales with it ensure proper memory management.

Disadvantages:

-Bottlenecks, the system may create performance bottlenecks, such as delays in resource loading or cache misses.

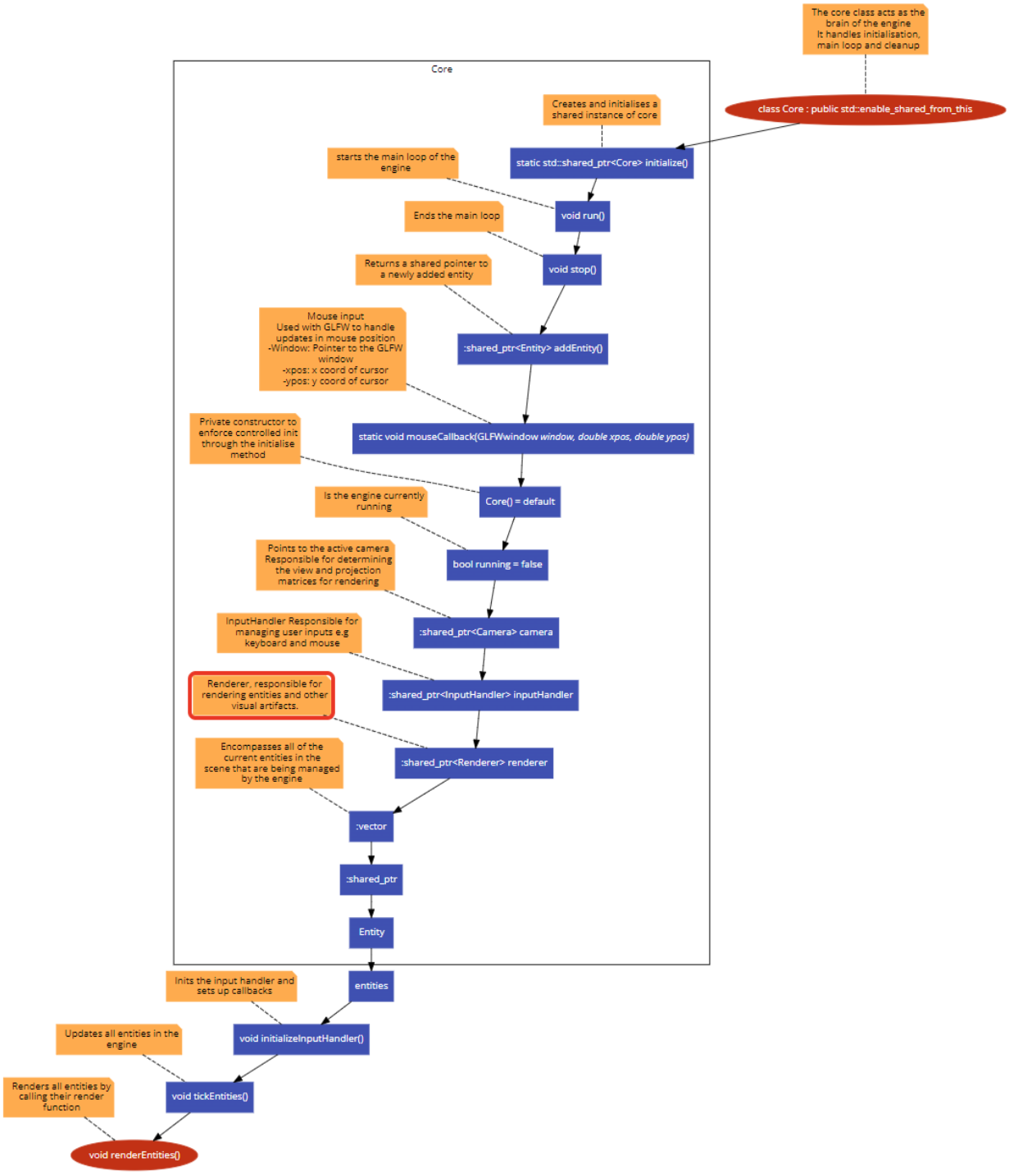
-Debugging, problems in the resource manager may lead to issues that are difficult to trace as data is being passed through the resource manager

-Initial load time, a large project may have an increased load time due to the indexing and preloading of assets.

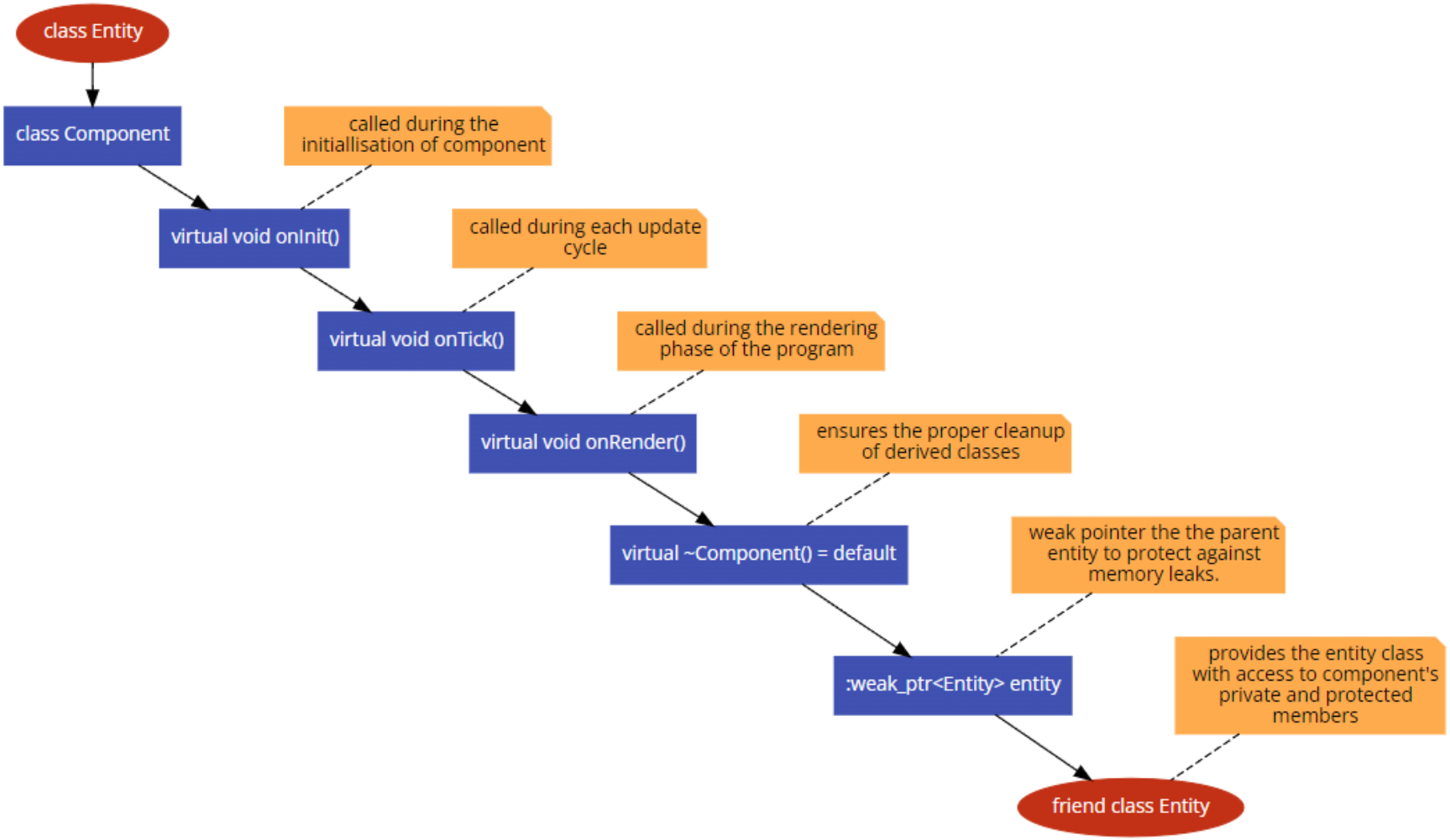
Diagrams

Information Flow Diagrams:

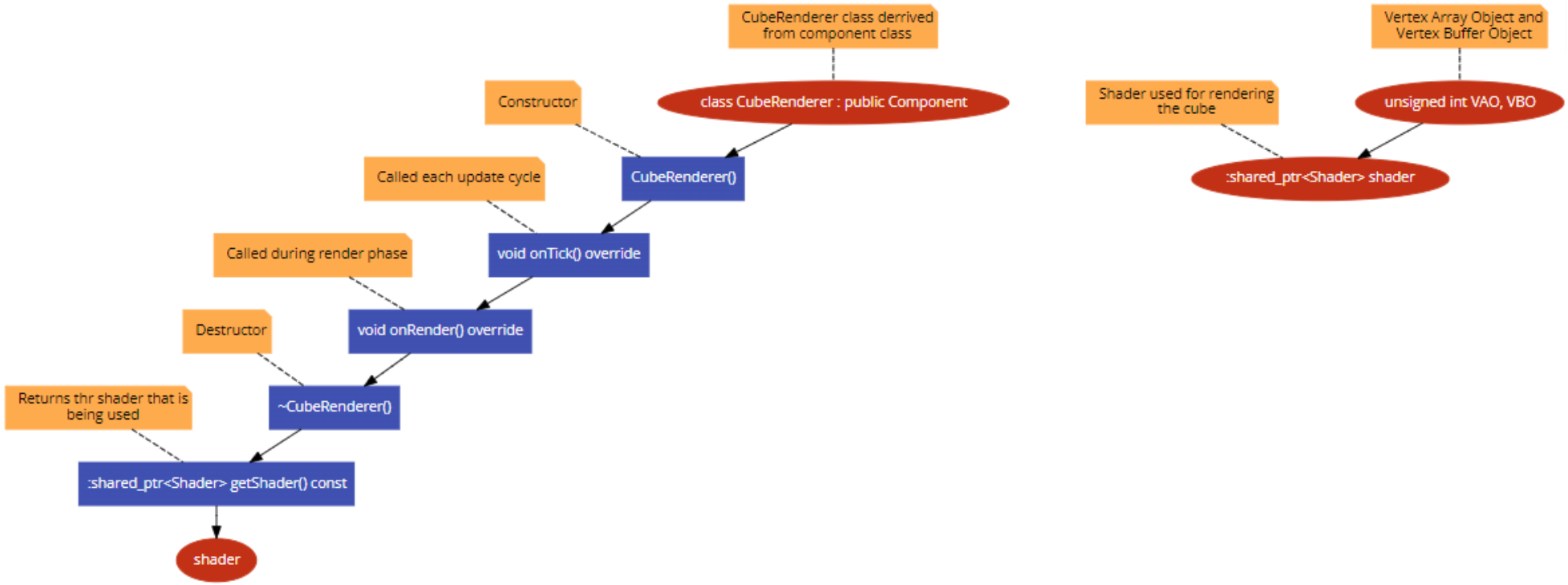
Core.h:



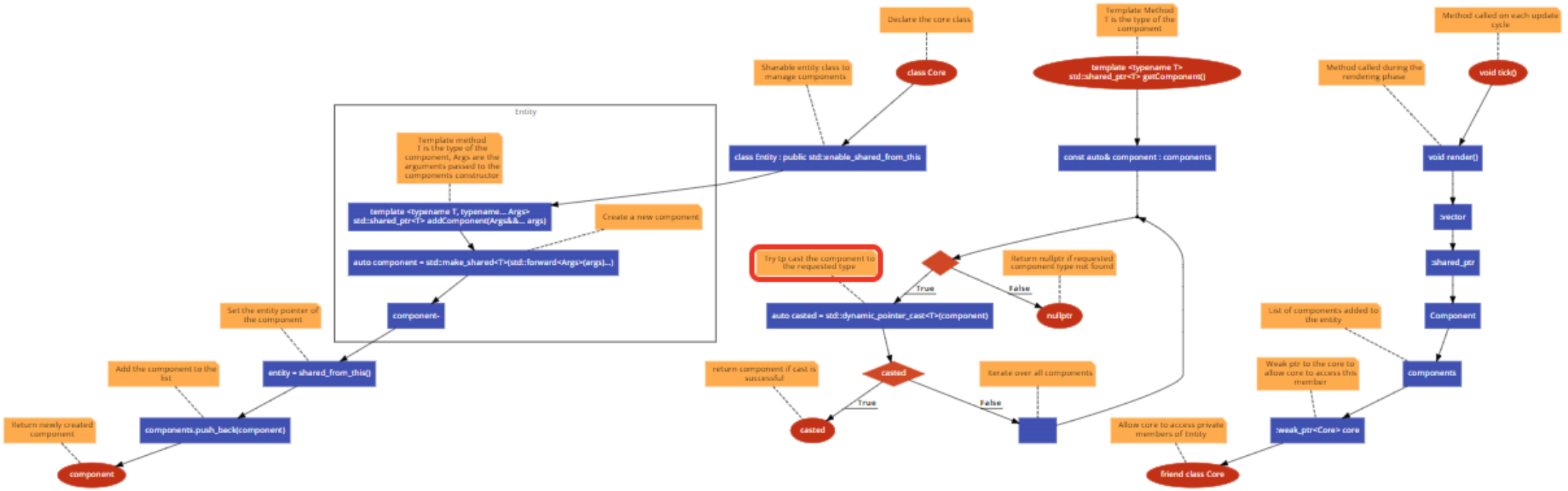
Component.h:


class Entity
class Component
called during the initiallisation of component
virtual void onInit()
called during each update cycle
virtual void onTick()
called during the rendering phase of the program
virtual void onRender()
ensures the proper cleanup of derived classes
virtual ~Component() = default
weak pointer the the parent entity to protect against memory leaks.
:weak_ptr<Entity> entity
provides the entity class with access to component's private and protected members
friend class Entity

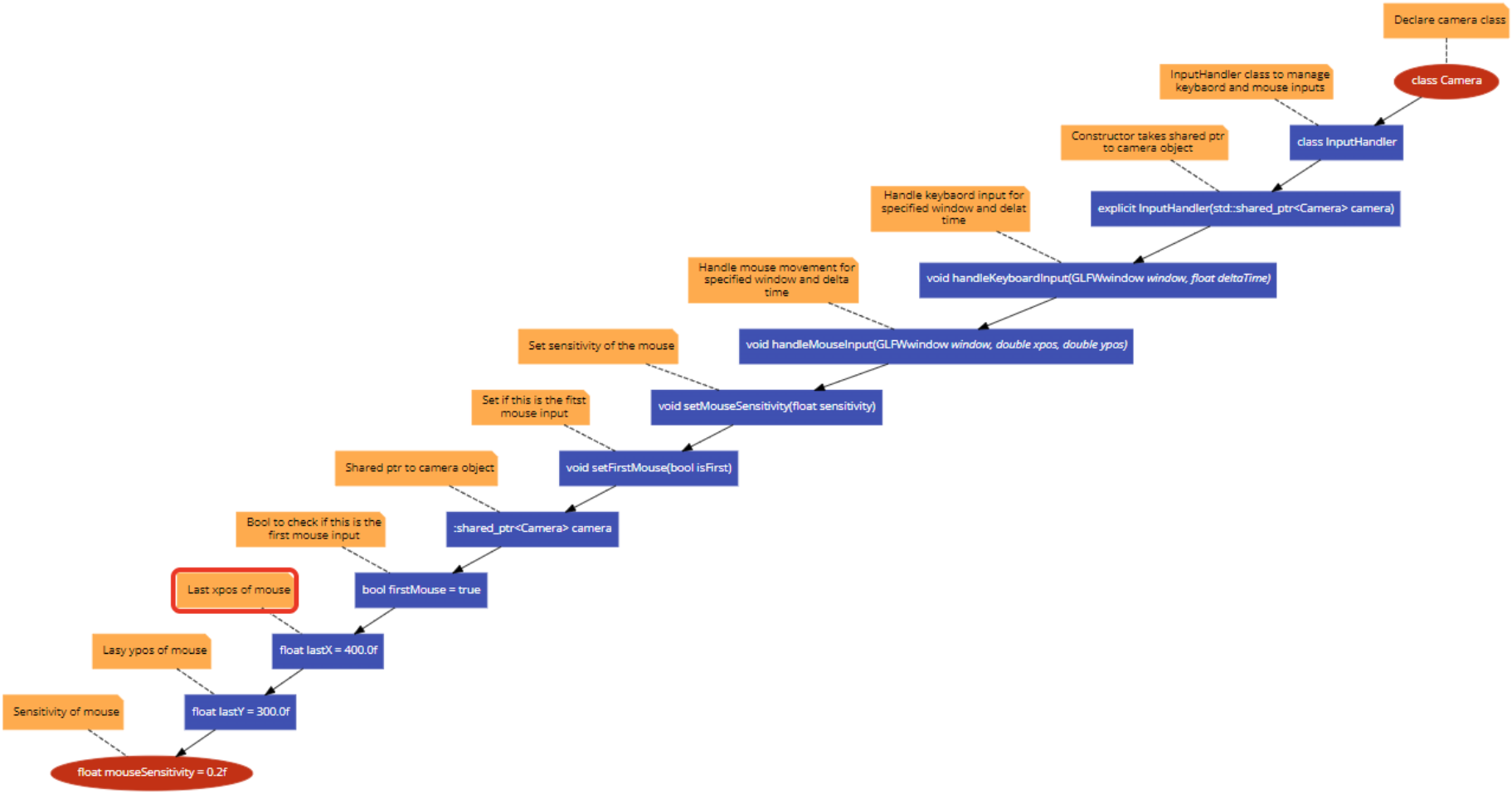
CubeRenderer.h:



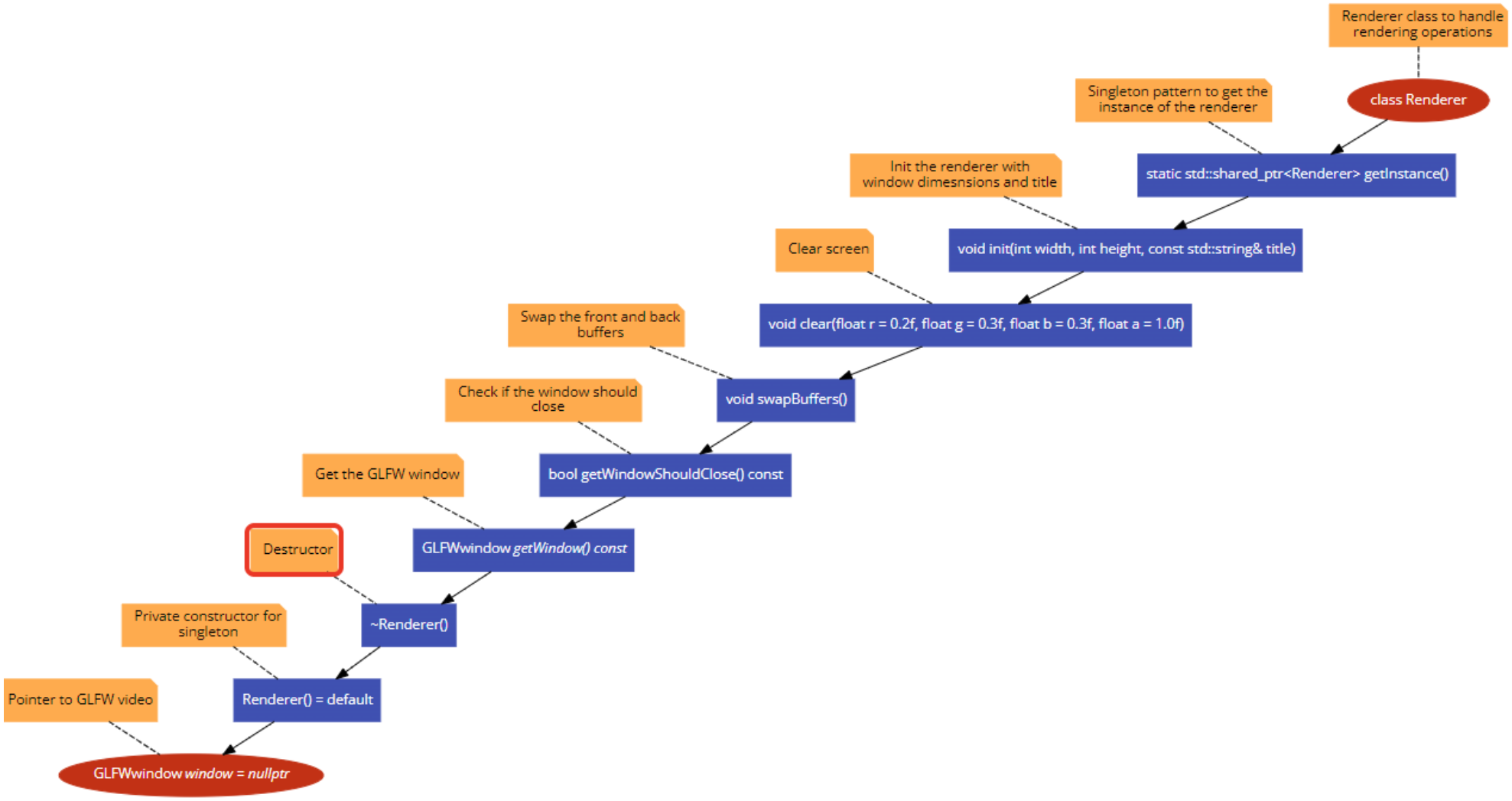
Entity.h:


Entity
Template method
T is the type of the component, Args are the arguments passed to the components constructor
template <typename T, typename... Args>
std::shared_ptr<T> addComponent(Args&&... args)
Create a new component
auto component = std::make_shared<T>(std::forward<Args>(args)...)
component-
Set the entity pointer of the component
entity = shared_from_this()
Add the component to the list
components.push_back(component)
Return newly created component
component
Declare the core class
class Core
Sharable entity class to manage components
class Entity : public std::enable_shared_from_this
Template Method
T is the type of the component
template <typename T>
std::shared_ptr<T> getComponent()
const auto& component : components
Try tp cast the component to the requested type
True
False
Return nullptr if requested component type not found
auto casted = std::dynamic_pointer_cast<T>(component)
nullptr
return component if cast is successful
casted
True
casted
False
Iterate over all components
Allow core to access private members of Entity
friend class Core
Method called on each update cycle
void tick()
Method called during the rendering phase
void render()
:vector
:shared_ptr
Component
List of components added to the entity
components
Weak ptr to the core to allow core to access this member
:weak_ptr<Core> core

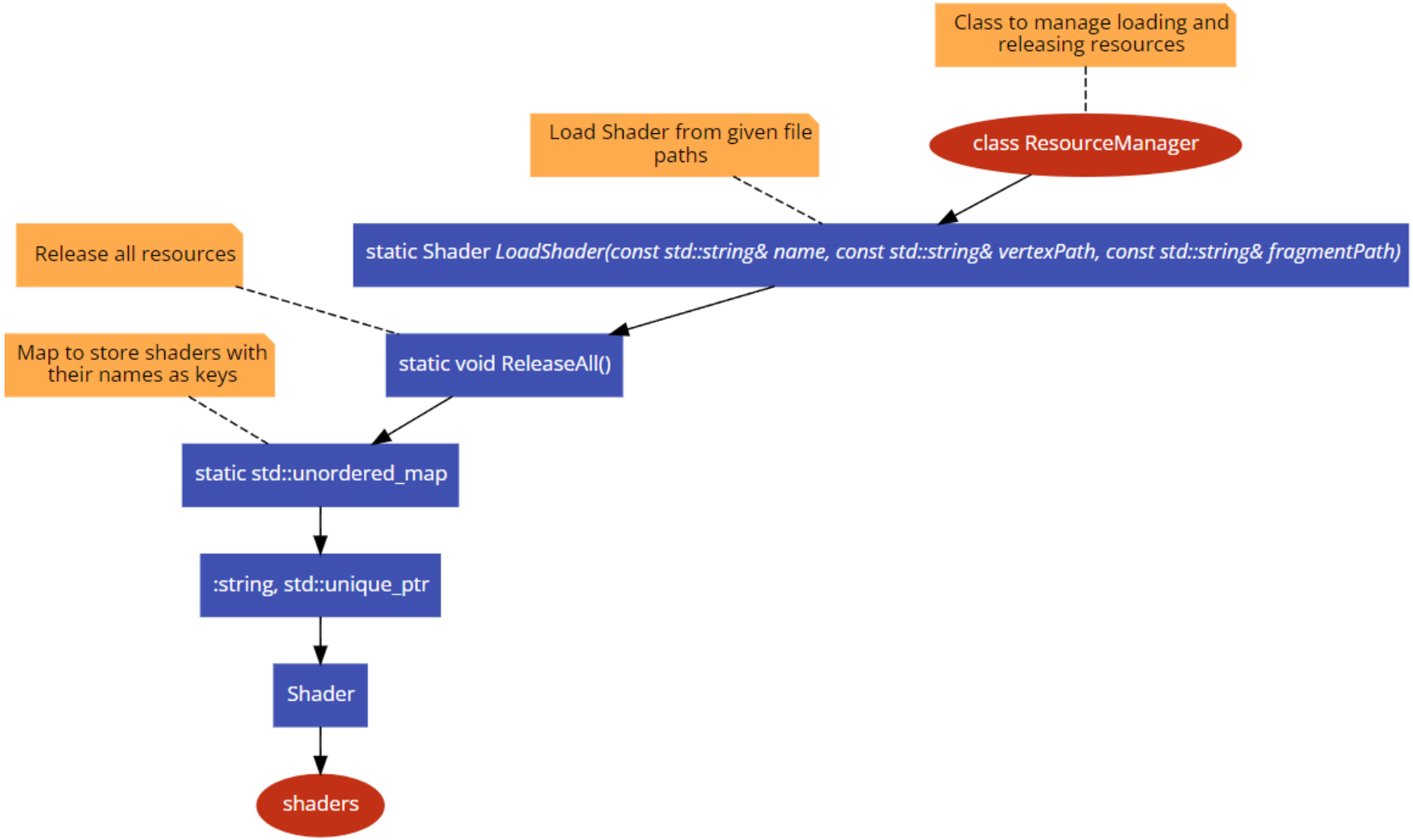
InputHandler.h:



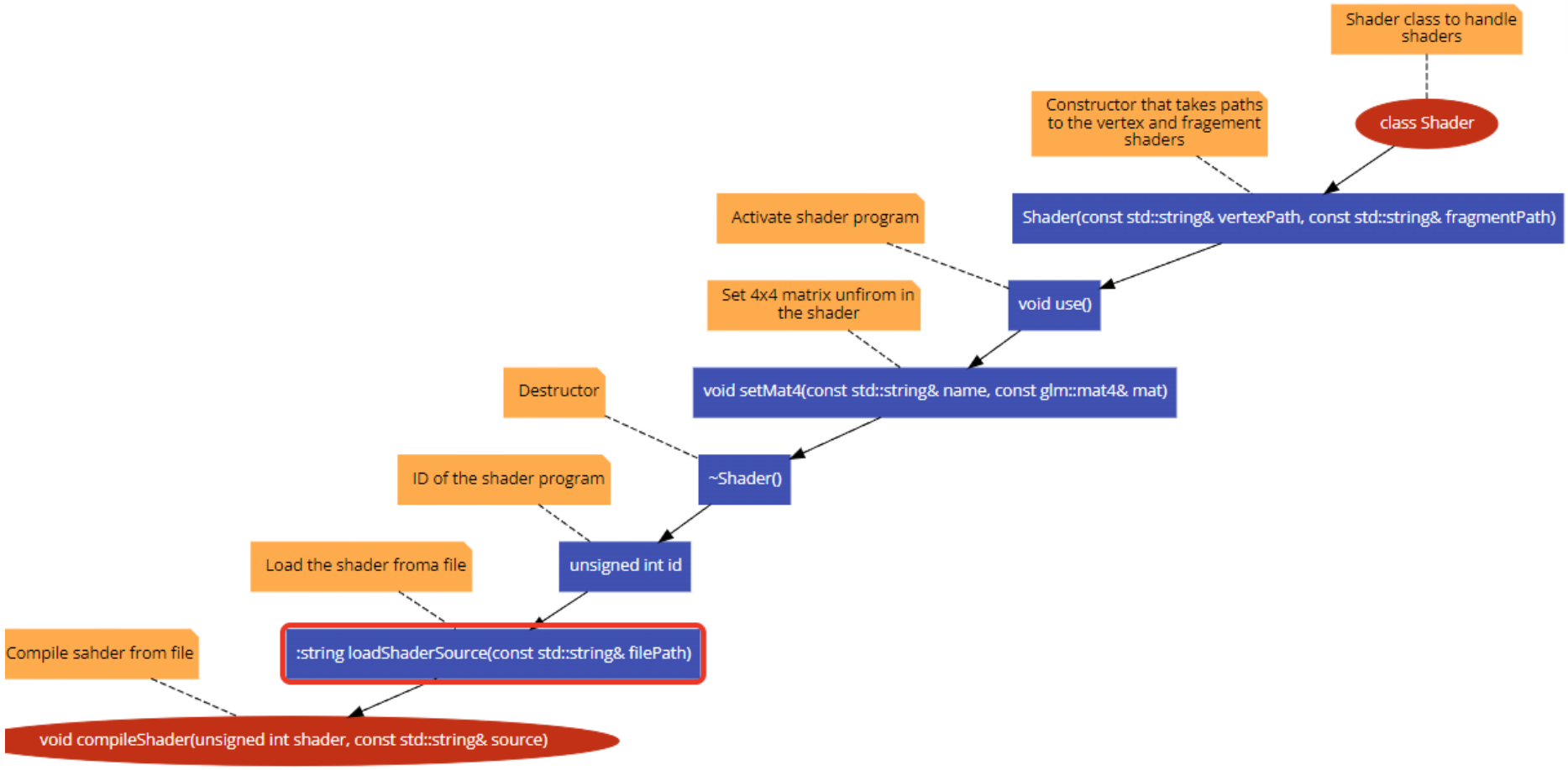
Renderer.h:


Renderer class to handle rendering operations
class Renderer
Singleton pattern to get the instance of the renderer
static std::shared_ptr<Renderer> getInstance()
Init the renderer with window dimesnsions and title
void init(int width, int height, const std::string& title)
Clear screen
void clear(float r = 0.2f, float g = 0.3f, float b = 0.3f, float a = 1.0f)
Swap the front and back buffers
void swapBuffers()
Check if the window should close
bool getWindowShouldClose() const
Get the GLFW window
GLFWwindow getWindow() const
Destructor
~Renderer()
Private constructor for singleton
Renderer() = default
Pointer to GLFW video
GLFWwindow window = nullptr

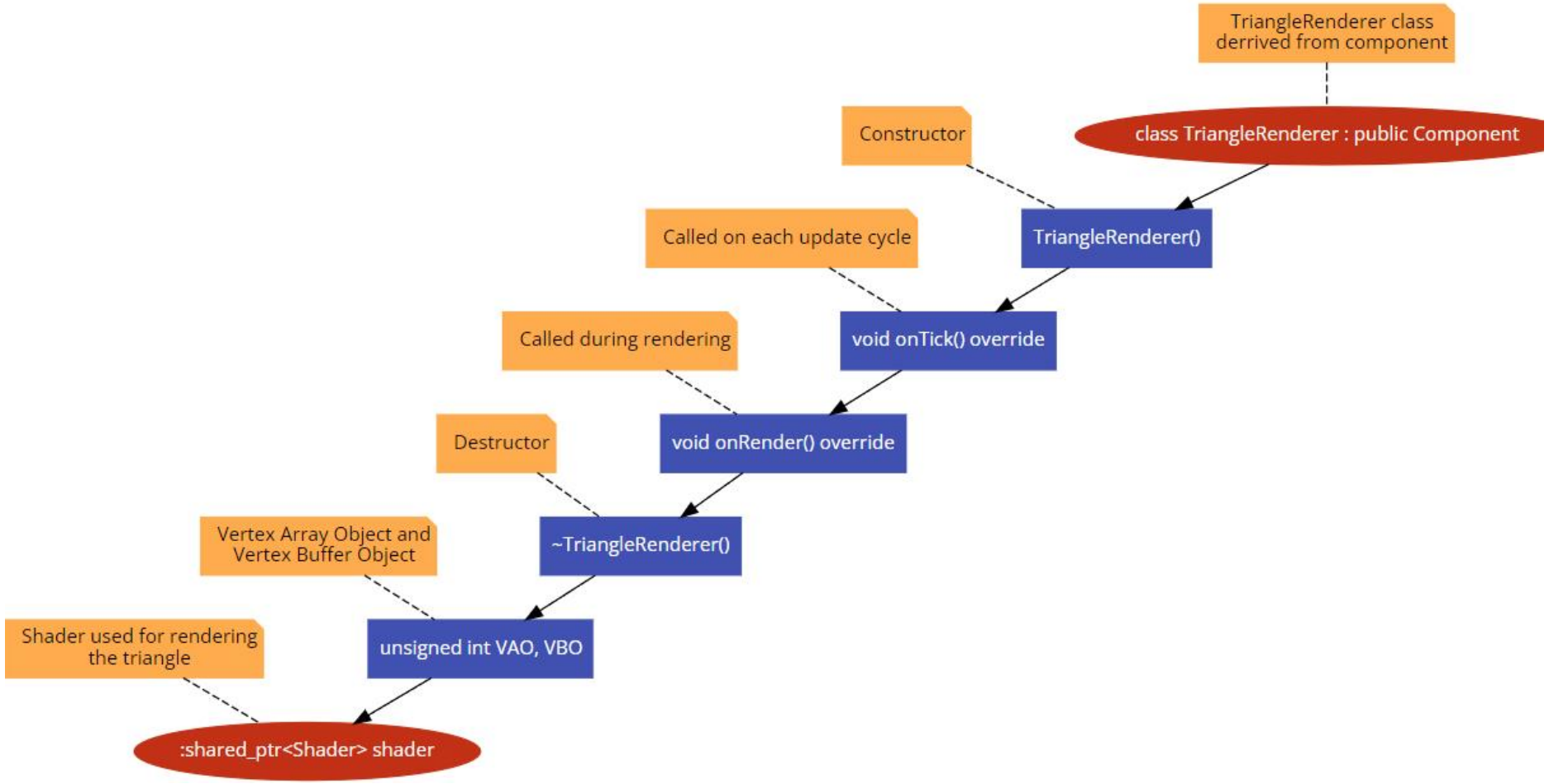
Resource Mnaager.h:



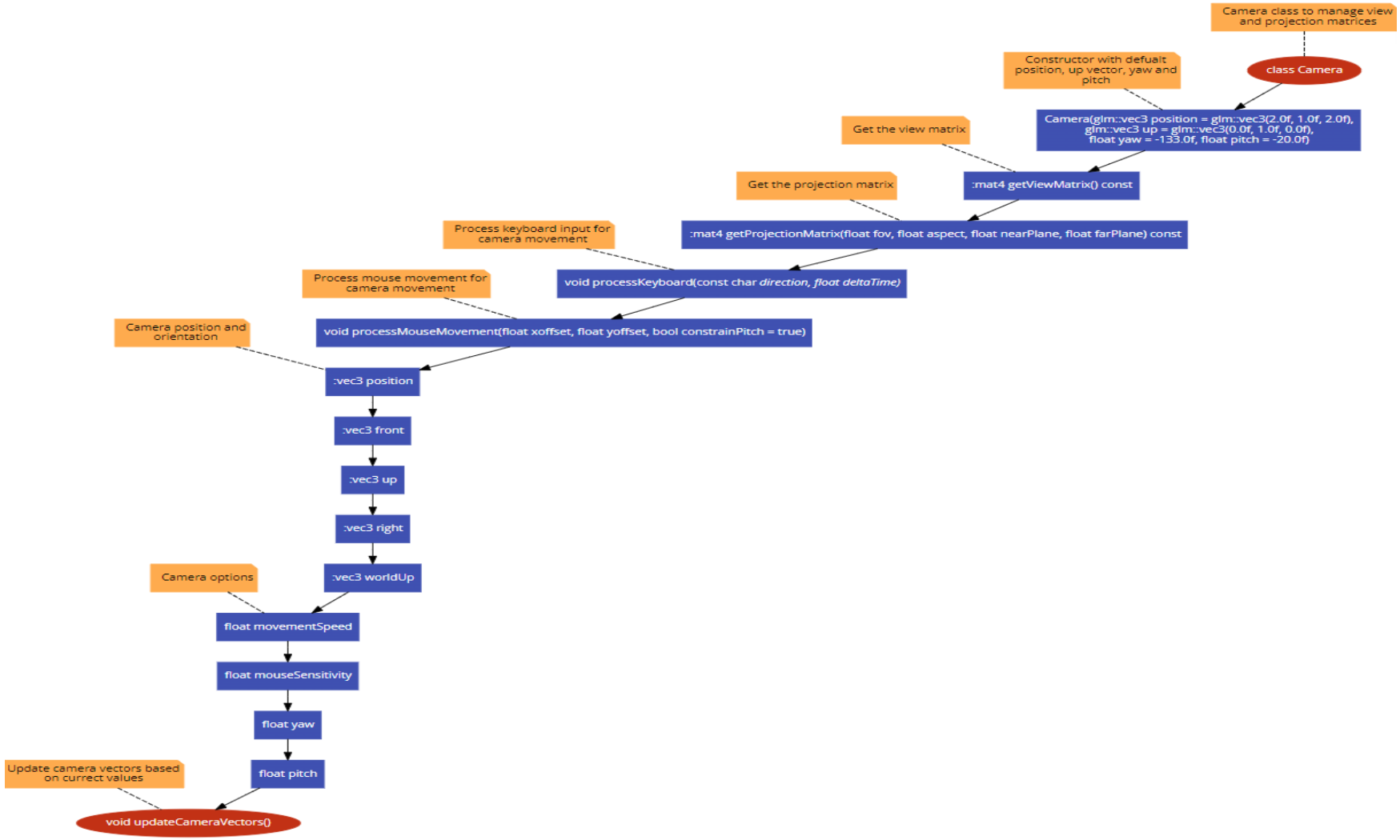
Shader.h:


Shader class to handle shaders
class Shader
Constructor that takes paths to the vertex and fragement shaders
Shader(const std::string& vertexPath, const std::string& fragmentPath)
Activate shader program
void use()
Set 4x4 matrix unfirom in the shader
void setMat4(const std::string& name, const glm::mat4& mat)
Destructor
~Shader()
ID of the shader program
unsigned int id
Load the shader froma file
:string loadShaderSource(const std::string& filePath)
Compile sahder from file
void compileShader(unsigned int shader, const std::string& source)

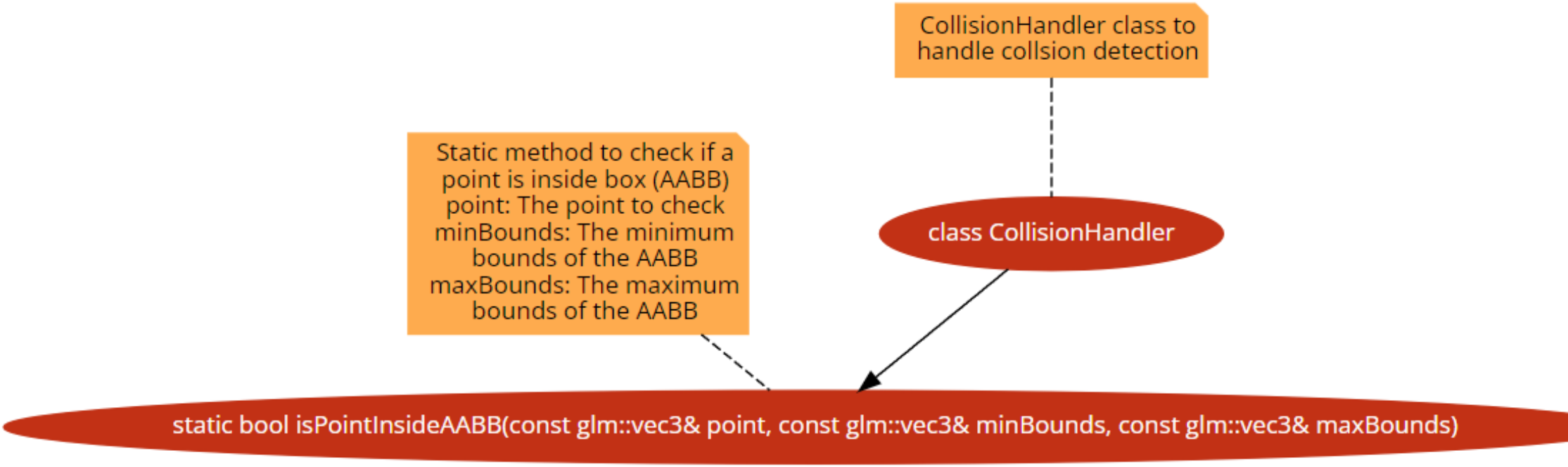
Triangle Renderer.h:



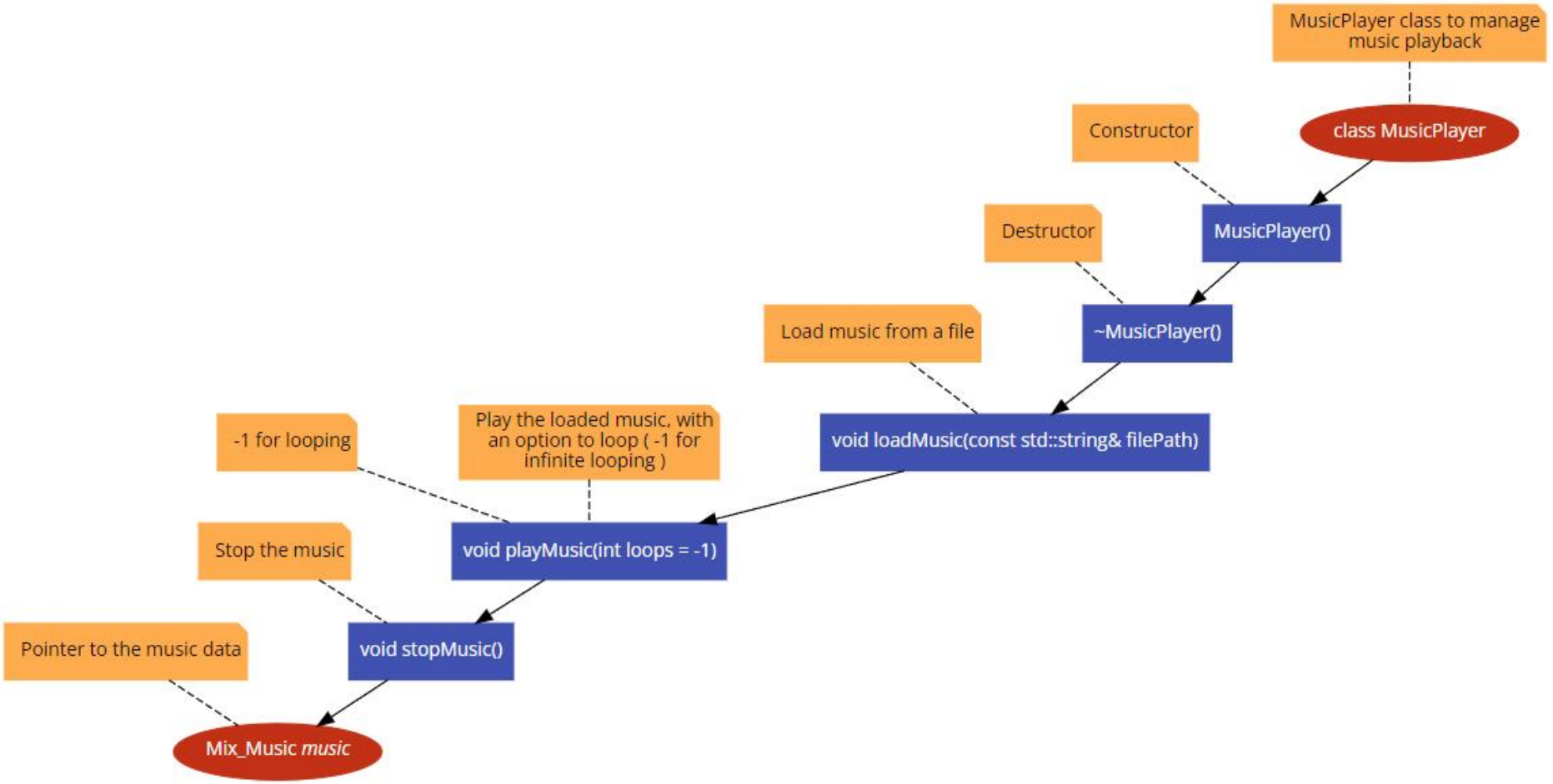
Camera.h:


Camera class to manage view and projection matrices
class Camera
Constructor with defualt position, up vector, yaw and pitch
Camera(glm::vec3 position = glm::vec3(2.0f, 1.0f, 2.0f), glm::vec3 up = glm::vec3(0.0f, 1.0f, 0.0f), float yaw = -133.0f, float pitch = -20.0f)
Get the view matrix
:mat4 getViewMatrix() const
Get the projection matrix
:mat4 getProjectionMatrix(float fov, float aspect, float nearPlane, float farPlane) const
Process keyboard input for camera movement
void processKeyboard(const char direction, float deltaTime)
Process mouse movement for camera movement
void processMouseMovement(float xoffset, float yoffset, bool constrainPitch = true)
Camera position and orientation
:vec3 position
:vec3 front
:vec3 up
:vec3 right
:vec3 worldUp
Camera options
float movementSpeed
float mouseSensitivity
float yaw
float pitch
Update camera vectors based on currect values
void updateCameraVectors()

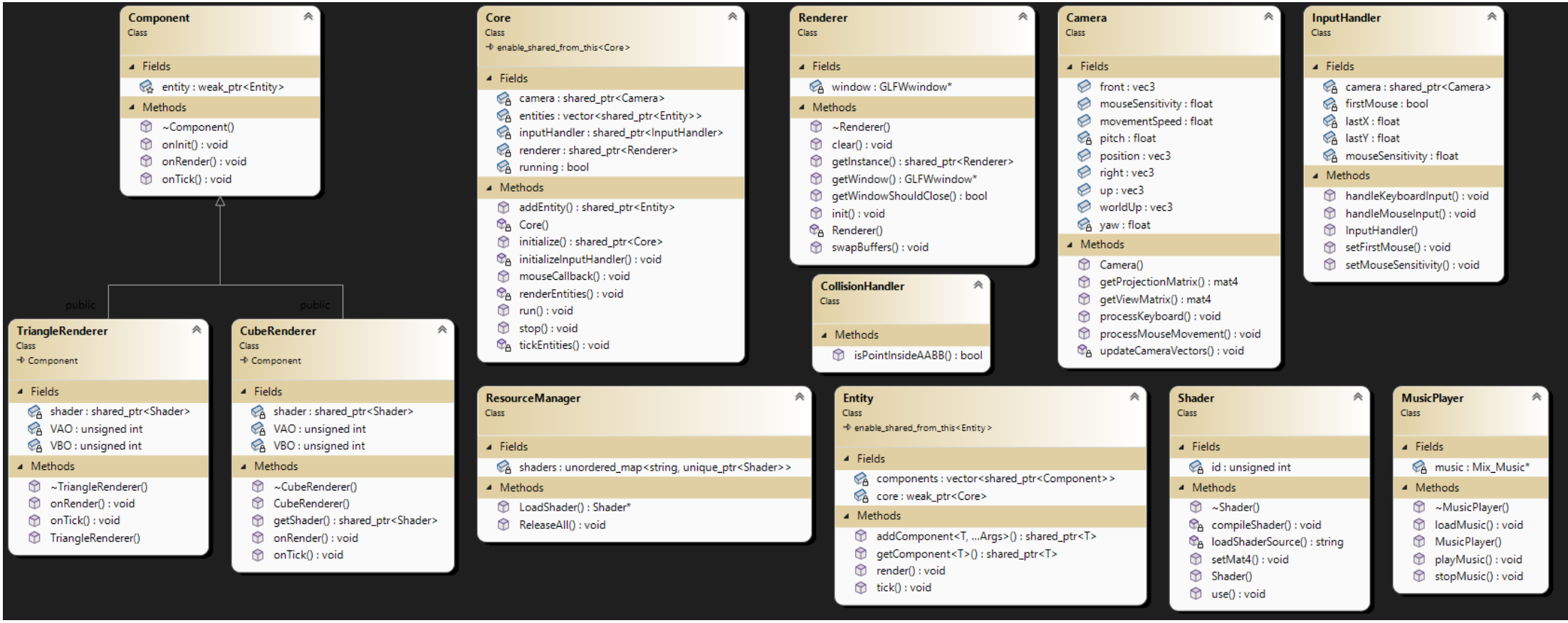
CollisionHandler.h:



MusicPlayer.h:



UML Diagram:


Component
Class
Fields
entity : weak_ptr<Entity>
Methods
~Component()
onInit() : void
onRender() : void
onTick() : void
public
public
TriangleRenderer
Class
Component
Fields
shader : shared_ptr<Shader>
VAO : unsigned int
VBO : unsigned int
Methods
~TriangleRenderer()
onRender() : void
onTick() : void
TriangleRenderer()
CubeRenderer
Class
Component
Fields
shader : shared_ptr<Shader>
VAO : unsigned int
VBO : unsigned int
Methods
~CubeRenderer()
CubeRenderer()
getShader() : shared_ptr<Shader>
onRender() : void
onTick() : void
Core
Class
enable_shared_from_this<Core>
Fields
camera : shared_ptr<Camera>
entities : vector<shared_ptr<Entity>>
inputHandler : shared_ptr<InputHandler>
renderer : shared_ptr<Renderer>
running : bool
Methods
addEntity() : shared_ptr<Entity>
Core()
initialize() : shared_ptr<Core>
initializeInputHandler() : void
mouseCallback() : void
renderEntities() : void
run() : void
stop() : void
tickEntities() : void
Renderer
Class
Fields
window : GLFWwindow*
Methods
~Renderer()
clear() : void
getInstance() : shared_ptr<Renderer>
getWindow() : GLFWwindow*
getWindowShouldClose() : bool
init() : void
Renderer()
swapBuffers() : void
CollisionHandler
Class
Methods
isPointInsideAABB() : bool
Camera
Class
Fields
front : vec3
mouseSensitivity : float
movementSpeed : float
pitch : float
position : vec3
right : vec3
up : vec3
worldUp : vec3
yaw : float
Methods
Camera()
getProjectionMatrix() : mat4
getViewMatrix() : mat4
processKeyboard() : void
processMouseMovement() : void
updateCameraVectors() : void
InputHandler
Class
Fields
camera : shared_ptr<Camera>
firstMouse : bool
lastX : float
lastY : float
mouseSensitivity : float
Methods
handleKeyboardInput() : void
handleMouseInput() : void
InputHandler()
setFirstMouse() : void
setMouseSensitivity() : void
ResourceManager
Class
Fields
shaders : unordered_map<string, unique_ptr<Shader>>
Methods
LoadShader() : Shader*
ReleaseAll() : void
Entity
Class
enable_shared_from_this<Entity>
Fields
components : vector<shared_ptr<Component>>
core : weak_ptr<Core>
Methods
addComponent<T, ...Args>() : shared_ptr<T>
getComponent<T>() : shared_ptr<T>
render() : void
tick() : void
Shader
Class
Fields
id : unsigned int
Methods
~Shader()
compileShader() : void
loadShaderSource() : string
setMat4() : void
Shader()
use() : void
MusicPlayer
Class
Fields
music : Mix_Music*
Methods
~MusicPlayer()
loadMusic() : void
MusicPlayer()
playMusic() : void
stopMusic() : void

References

In creating this project, I conducted a large amount of research into best practices, industry examples and tool usage. When creating a game engine in 2024 utilising resources available to me was essential as so many games and their parent engines run of open-source software that makes the process understandable and compatible even if they do not make the overall task easier. Below I will list my references and how they assisted me with the project.

Initial Research:

Overview of game engine structure and usage:

Glaiel, T. (2021). *How to make your own game engine (and why)*. [online] Geek Culture. Available at: https://medium.com/geekculture/how-to-make-your-own-game-engine-and-why-ddf0acbc5f3.

lisyarus blog. (2023). *So, you want to make a game engine*. [online] Available at: https://lisyarus.github.io/blog/posts/so-you-want-to-make-a-game-engine.html [Accessed 16 Jan. 2025].

Anon, (2019). *How to Make a Game Engine (Step by Step)*. [online] Available at: https://www.gamedesigning.org/learn/make-a-game-engine/.

www.gamedeveloper.com. (n.d.). *How to Make Your Own C++ Game Engine*. [online] Available at: https://www.gamedeveloper.com/game-platforms/how-to-make-your-own-c-game-engine.

Game Engine structure:

Lambert, J. (2014). *Making a Game Engine: Core Design Principles*. [online] GameDev.net. Available at: https://www.gamedev.net/tutorials/programming/general-and-gameplay-programming/making-a-game-engine-core-design-principles-r3210/ [Accessed 16 Jan. 2025].

isetta.io. (n.d.). *Engine Architecture - Isetta Engine*. [online] Available at: https://isetta.io/blogs/engine-architecture/.

Gregory, J. (2019). *Game engine architecture*. Boca Raton ; London ; New York: Crc Press, Taylor & Francis Group.

Ullmann, G.C., Guéhéneuc, Y.-G., Petrillo, F., Anquetil, N. and Politowski, C. (2024). SyDRA: An approach to understand game engine architecture. *Entertainment Computing*, [online] 52, p.100832. doi:https://doi.org/10.1016/j.entcom.2024.100832.

Simplilearn.com. (2022). *Entity Component System: An Introductory Guide | Simplilearn*. [online] Available at: https://www.simplilearn.com/entity-component-system-introductory-guide-article.

Externals:

Dav1d.de. (2020). Available at: https://glad.dav1d.de/.

GLFW. (n.d.). *An OpenGL library*. [online] Available at: https://www.glfw.org/.

Libsdl.org. (2019). *Simple DirectMedia Layer - Homepage*. [online] Available at: https://www.libsdl.org/.

libsdl-org (2024). *GitHub - libsdl-org/SDL_mixer: An audio mixer that supports various file formats for Simple Directmedia Layer.* [online] GitHub. Available at: https://github.com/libsdl-org/SDL_mixer.

GitHub. (2020). *g-truc/glm*. [online] Available at: https://github.com/g-truc/glm.

www.doxygen.nl. (n.d.). *My Project: Overview*. [online] Available at: https://www.doxygen.nl/manual/.

cmake.org. (n.d.). *CMake Tutorial — CMake 3.21.3 Documentation*. [online] Available at: https://cmake.org/cmake/help/latest/guide/tutorial/index.html.

Git-scm.com. (2019). *Git - gittutorial Documentation*. [online] Available at: https://git-scm.com/docs/gittutorial.

Guides:

Harold Serrano (2021). *How to implement the Rendering System | Game Engine Development*. [online] YouTube. Available at: https://www.youtube.com/watch?v=oyXRE809geY [Accessed 16 Jan. 2025].

Kea Sigma Delta (2024). *Building a Game Engine Resource Manager - RayLib 2D Challenge Part 12*. [online] YouTube. Available at: https://www.youtube.com/watch?v=TlR4HOO4A5U [Accessed 16 Jan. 2025].

Conclusion:

The Cerberus Engine is successful in creating a lightweight game engine for simple projects. It features ECS, Resource Management, Audio functionality, Rendering, Collision system and a Secure Engine Core. It is well documented using comments, Doxygen and a README file.

It does however have lots of room for improvement. Model loading would be very beneficial for users moving forward. The core engine is secure but could be bypassed by someone with a high level of C++ knowledge, therefore it would be a good extension to increase its security and have it tested by users.

Resource management could be extended to include capability for Multithreaded resource loading, especially for larger projects. A visual editor for creating and managing scenes like most industry standard game engines would increase the usability for a wider range of skill groups. Finally cross platform support is something that I would like to add in the future so that my engine is platform agnostic.