Olivera Engine

System Requirements:

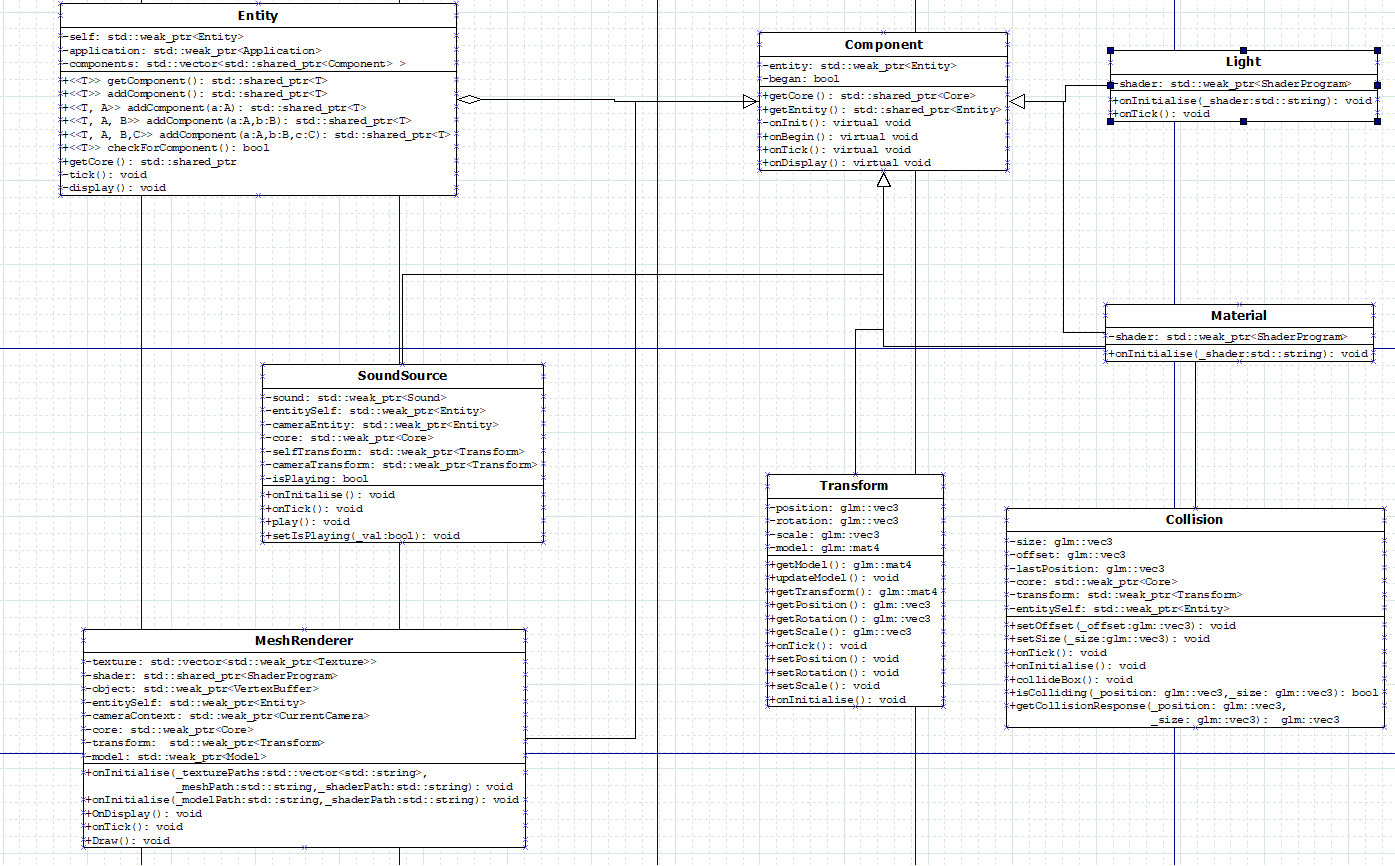
OS: Windows XP or later

RAM: 4GB of RAM or higher

Graphics Card: compatible with openGL 3.3; Nvidia GeForce 8000 or higher

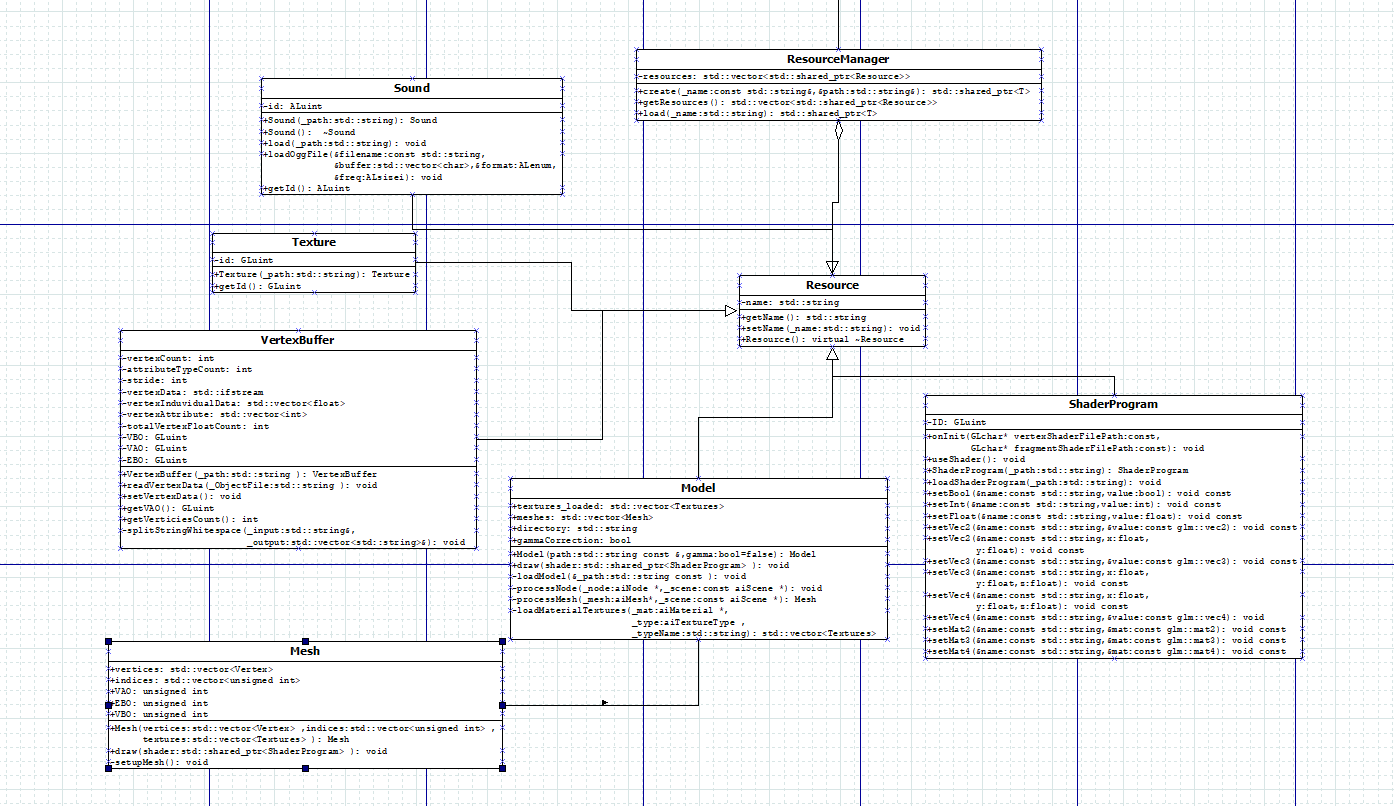
CPU: 1 gigahertz (GHz) or faster 32-bit (x86) or 64-bit (x64) processor

Olivera Engine is a Game Engine using SDL2 for window creation, and OpenGL for rendering. It uses a component-entity system. Whenever an entity gets created the user can attach components(objects derived from the Component class).

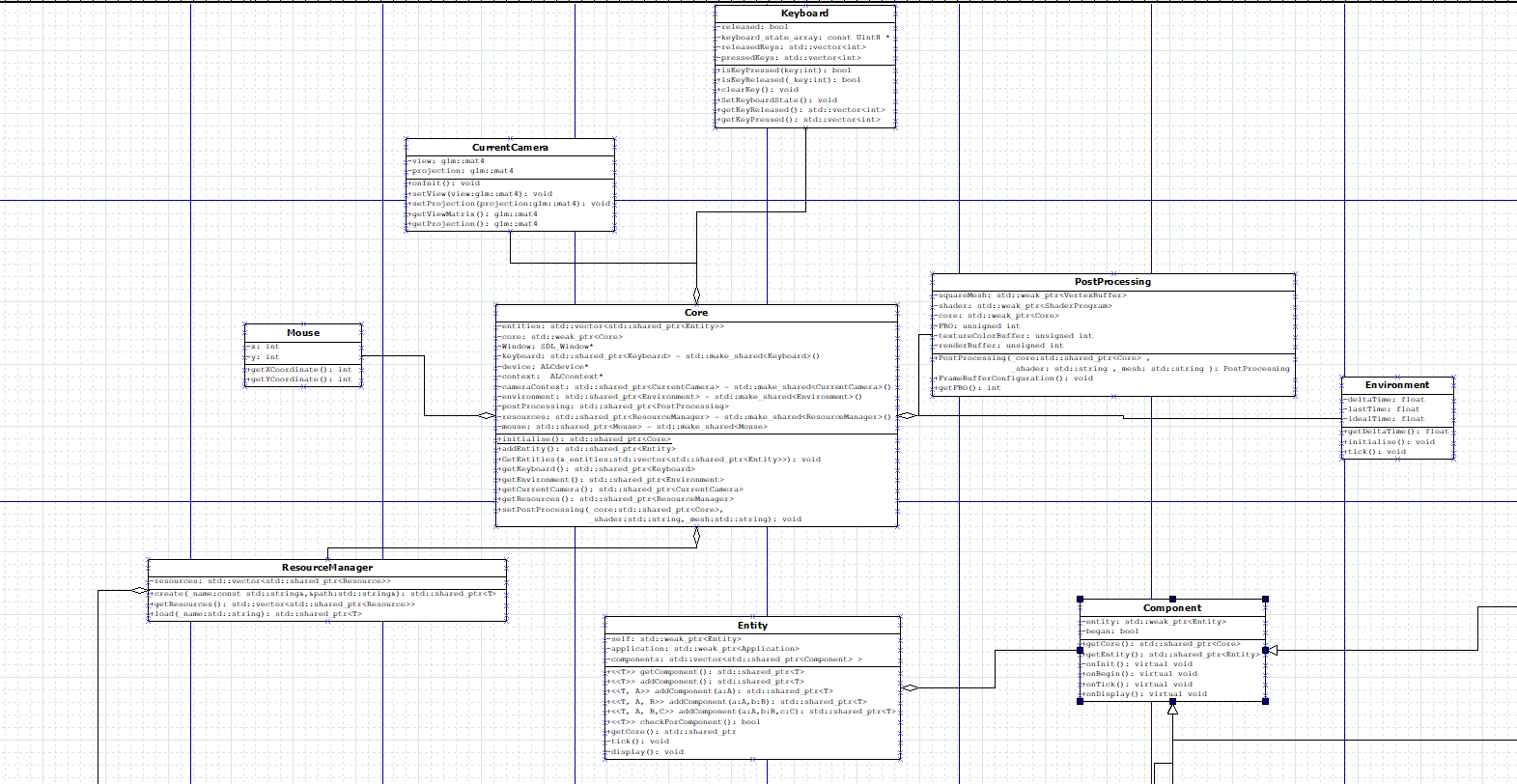


The UML diagram above displays the components within the framework. Mesh Renderer handles rendering meshes, Material sets object material values, Light sets lighting values and sources, SoundSource handles sources of sounds and who can hear them, collision handles simple 3D Box collision and Transform handles an entity's transformation values. Users can make their custom component classes by inheriting from the Component class and using its virtual functions.

The engine also makes use of a resource system:



All resources are stored in a vector contained in the ResourceManager class. Whenever a resource gets created the user needs to assign a name tag so that when they need to load it they can distinguish it by said tag. Resource derived classes that are within the API are: ShaderProgram that handles creation of shader programs by taking in a vertex and fragment file that the user has to create. VertexBuffer which handles creation of simple objects from a text file, which can be useful for testing. Model class which uses the ASSIMP library for loading in 3D objects. Texture class for loading in textures and a sound class that currently only supports loading in .ogg files.

Both systems are contained within the Core class which handles the component-entity system and holds the resource manager as shown in the diagram below:

Core also contains: An Environment class, which handles the framerate and deltaTime; Keyboard class that handles and saves all keyboard inputs; CurrentCamera class that stores a projection and view matrix that users can use setters to modify and implement their own camera systems(FPSCamera in the demo); Mouse class that tracks the mouse on the window; PostProcessing class which users can use to pass their own Post-Processing shaders. The engine provides a decent amount of features so that simple games can be created, while also providing modular code to easily add extra features within the games. However it has a lot of weaknesses:

* It’s windows only compatible, due to program using the windows file path system to locate objects.
* The users have to provide their own Shader Vertex and Fragment Files which would mean the user has to have knowledge of the GLSL shading language
* The window isn’t resizable. This can easily be corrected by making the window width and height values accessible outside of core as they are hardcoded.
* Only keyboard and mouse support
* Limited mouse support to only tracking mouse movement not mouse clicks.

While the engine at its current state is quite limited, due to its modular design extra features would be easily implemented without having to change much if at all of it’s core functionality.