Game Engine Programming

ASSIGNMENT 1

Richard Hancock | BSc Games Programming | 12/12/16

*More than two pages because it's impossible with UML diagrams that take up a page each by themselves. I'm within the word count.

Table of Contents

Introduction

For this assignment I have implemented my own game engine following the basic design principles of the Mutiny engine. I have tried to focus on the cross platform capabilities as my primary focus, with input and graphics following closely behind,

UML Diagrams

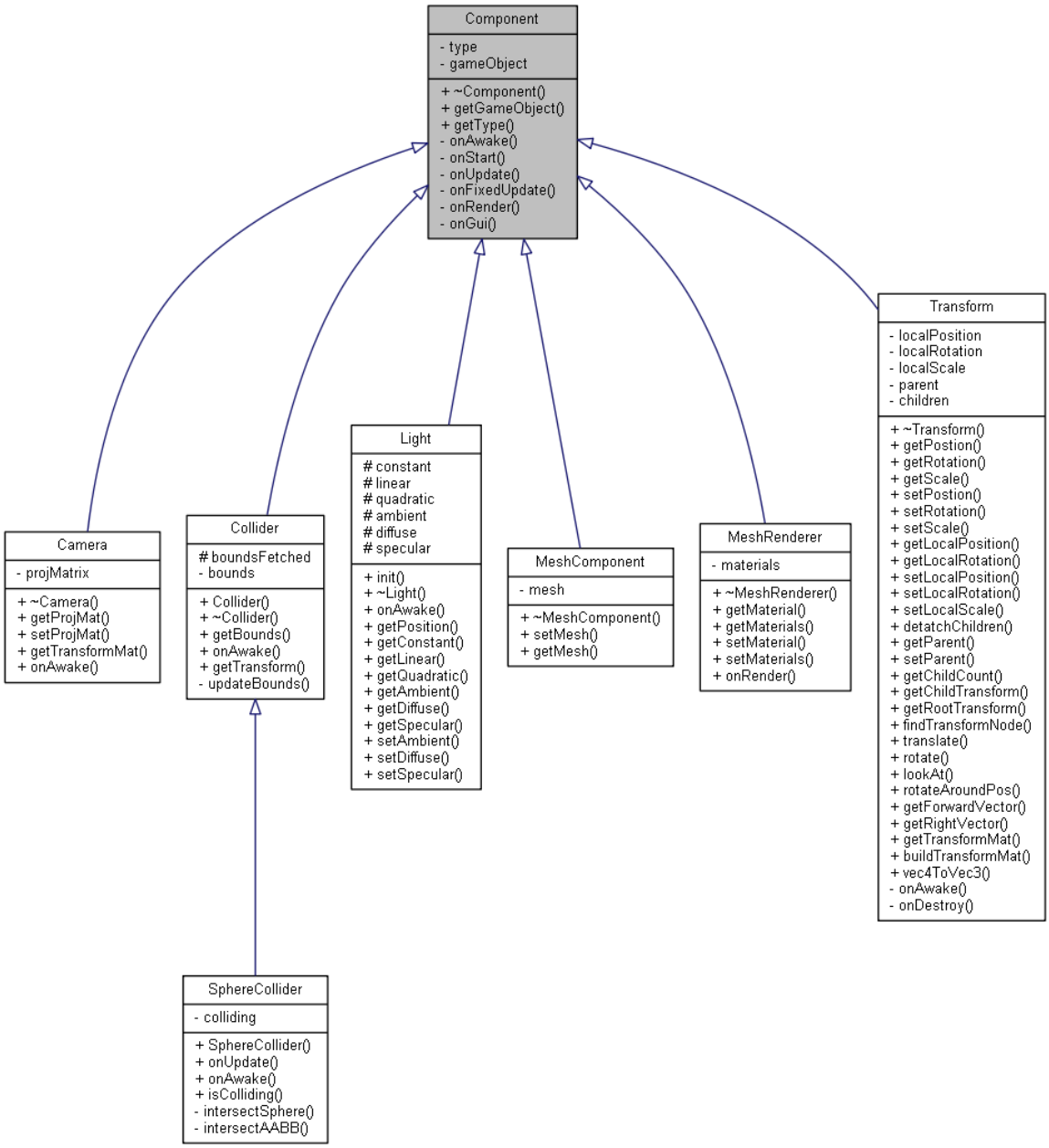

Figure 1 Component UML Diagram

My engine features most of the major components needed for the assignment plus a few extra. Figure 1 shows a UML inheritance diagram showing their basic capabilities.

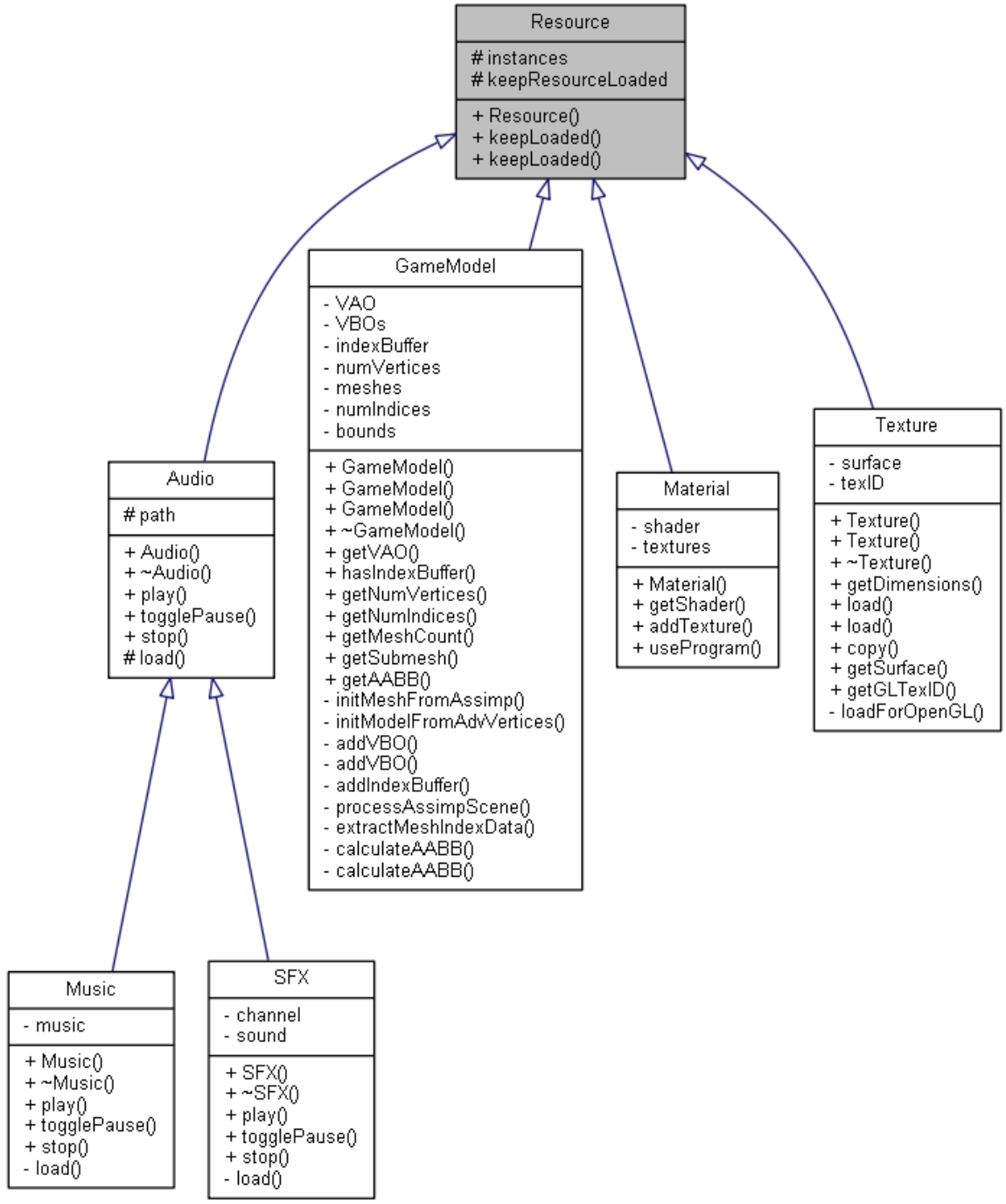

Figure 2 UML of Resource types

Figure 2 shows the different type of resource my engine can load and utilize. They are all loaded into my resource manager as smart pointers to protect from memory leaks and for convenience of access.

Features

- Supports Linux style operating systems (mainly tested on Debian and Ubuntu). Only feature not guaranteed to work is controller support as Linux is not very standardized in this area. I created a python script that generates the Makefile on Linux platforms.
- Static Platform class that handles all initialization, storage and deletion of SDL, OpenGL, GLEW and any other libraries. Also features a settings file loader that loads XML files containing user settings such as resolution, anti-aliasing and Fullscreen/Borderless/Windowed.
- Fully feature input system that supports all keyboard and mouse operations, plus supports controllers such as the Xbox 360/One controller. It can handle up to 4 controllers at a time. Also supports controller haptics such as rumble/vibrations with configurable strength and durations.
- Configurable Shader class that allows uniforms variables to be created and sent to the graphics card easily. Keeps track of what variables have already been passed to the graphics card so that it can overwrite the data rather than allocating new space.
- Supports the loading of complex meshes made up of several sub components and materials. Currently supports: normal maps, specular maps and standard diffuse maps/textures. Utilizes Assimp library for complex models, but I have my own OBJ loader that I use for the simpler meshes like spheres.
- Multiple light support, in the demo I have a directional light and a movable point light to demonstrate this.
- Can use Sphere vs Sphere collision and AABB vs AABB, technically can also do Sphere vs AABB but the sphere collider falls back to its AABB component.

User Guide

Settings can be changed in the settings file which is stored at:
“C://Users/NAME/AppData/Roaming/RH/Engine”

Once the program has loaded, the camera can be controlled using:

- W – Move forward
- S – Move backwards
- A – Move left

- D – Move right
- Q – Move down
- E – Move right
- Arrow keys – Rotate
- K, L – Roll

Alternatively, the Left analog stick of a connected controller can move the camera in some axis.

- 1 Key or A button – Plays a sound to demonstrate audio capability
- C Key or Y button – Switch between controlling camera and the light emitting sphere.
- Space Bar or B button – Resets the camera and sphere back to starting positions.

As mentioned above you can switch to controlling the light emitting sphere by pressing the C Key or Y button, by moving this around you can test that collisions and the lighting are working. On collision with the other sphere the controller should also trigger a haptic event (Rumbles).

Evaluation

I have run my program through Valgrind a memory leak detection tool, the results seemed to show that the only leaks in my program are due to 3rd party libraries such as Assimp and SDL.
The Engine also compiles with almost no warnings on level 4 strictness, only warns about not aligning GLM matrices in memory. I have also tested the Engine on Ubuntu where it runs at nearly the same speed as the Windows version.

Conclusions

One part I would like to improved is the memory management, as some parts of the code still use raw pointers. Most of these are related to Assimp which threw many errors when I attempted to use smart pointers as it manages its own memory internally.
Another aspect I would like to improve is the lighting, as while it has fairly complex elements to it, all of the details are just washed out by the base textures.
I would have also liked to support Emscripten however the amount of changes required to the current code base would be extreme, so I will investigate this possibility for the next assignment.

For the next assignment I hope to further refine the component system to be more convenient to use, as currently declaring one game object and all of its components takes many lines. I would also like to implement even more detailed lighting and shadows.

Appendices

In addition to this report I have generated full DoxyGen documentation in both HTML and PDF formats. I have turned all the UML settings to their highest to provide inheritance, collaboration, usage and file diagrams.