Victor Leach 27/11/2017

Game Engine Assignment

Introduction

I created a component based game engine using OpenGL, C++ and SDL.

The engine can display a given scene filled with game objects. Game objects can be created dynamically and components added to each game object. Components and shaders can easily be subclassed to extend functionality and add features easily.

The engine includes a basic shader for simple meshes with colour or an advanced shader with texturing, lighting and normal mapping. Multiple shaders can be used at once and the engine handles switching between them.

Basic bullet physics wrapper components have been implemented and collision is working. A variety of shape and mesh colliders through rigid body components allow game objects to interact with the physics world.

Structure

Please see attached UML diagram for class structure.

Code Standards

I used a consistent code style across my whole project. I used camel case naming convention and similar logical ordering of function definitions and include statements.

Component Properties

The first version of the component based system had elements such as mesh and shader as separate components but I quickly learnt that this did not make sense and that they should be component properties. Reducing the number of components helps make for a simpler and better developer experience as it’s less confusing.

Generics

Generics are a key part of a component based system as they allow for strongly typed component instances and offer flexibility when adding and accessing components within a game object. I was able to use template classes and dynamic casting to check for existing components and return them.

Although I didn't use this in future, components can be subclassed to provide more specific functionality. Dynamic casting would then allow for the superclass properties and methods to be accessed without having to check each component subclass individually for existence.

Shader First

Ultimately the shader itself is the only class that knows which uniform and buffer inputs it should take. For this reason there was no clean way to implement a MeshRenderer which actually handled the rendering functionality itself. Instead I put the actual rendering code into a Shader wrapper class which receives a mesh renderer component which needs to be displayed. This allows the shader to decide how the object is displayed and can choose which mesh properties to utilise.

The shader class is designed in such a way that new shaders can easily be created by subclassing shader. They can implement their own custom functionality both in the class and glsl code and then simply assigned to a renderer component.

Evaluation

What Went Well

I believe that my shader design allows for advanced shaders to be created easily without disrupting the main game functionality.

The parent child game object system works well and allows for hierarchical objects to be created. The demonstration scene shows how physics can be applied to one object and another follows the parent game object position.

Instead of implementing multiple collider components I created one which can be configured to represent any shape or mesh. This simplifies the component hierarchy and adds flexibility to the use of physics. Rigid body does most of the physics implementation meaning that the collider can have very basic functionality.

Some game engines may report unexpected errors if a component has another component dependancy which does not exist. I check beforehand to make sure the required components exist and output a useful error if this is the case. If I had more time I would extend this system further to allow components to specify their dependancies which are validated at game object creation.

Improvements

Currently I only have support for one light inside the scene as the shader code doesn’t support any more. If I had more time I would research how to handle multiple light sources and integrate this into my project.

I wrote the object loader inside the mesh class myself and so is not perfect currently. It may not be compatible with all forms of face data as the string processing functions are not flexible enough. If I had more time I would test my engine with a larger variety of object files and include more intelligent string parsing. I would also add support for obj files including multiple material definitions from the mtl file.

The physics engine is still quite basic and only deals with collision currently. If I had more time I would extend this to include more wrapper functionality around the bullet library.

I am using shared pointers throughout my entire game engine except for the bullet physics library as I couldn't get it to work properly with shared pointers. This may be because bullet manages it's own memory and caused crashes when attempting to access shared pointers. I've used raw pointers for now and would revisit this issue if given more time.

Currently the movement component interacts directly with the transform position and rotation. This is incorrect if the object is taking part in the physics world as it would not update the physics position when moved. If I was given more time I would create a separate component which passed movement actions to the rigid body instead of the transform

References

Normal, specular and ambient mapping lighting functionality in my advanced shader was referenced from my pervious graphics assignment.