Game Engine Programming
Lab 5: Resources

Karsten Pedersen
Department of Creative Technology

October 6, 2019

In this lab we are going to focus implementing a flexible resources system. There are a small number of things to initially consider.

- Portability of code
- Efficiency of handling data
- Ease of use of API
- Tools to generate archives

Lets begin by drafting up a potential API that we might like to use:

```
// Load texture at given directory.
shared<Texture> t = getResources()->load<Texture>("images/player");

// Create MeshRenderer component and assign the new texture to it.
shared<MeshRenderer> mr = getEntity()->addComponent<MeshRenderer>();
mr->setTexture(t);
```

This code is quite simple. It basically would load a texture from a given path and the resulting texture is then assigned to the created MeshRenderer.

Note:
Avoiding a static Resources class (as seen in Unity) has a benefit in that you can control the lifespan of the Resources class and all resources contained within it. We cannot guarantee the order of destruction of static classes (or more specifically static objects within static classes). However the fact we need to follow pointers to obtain the Resources class does have a minor performance overhead. Keep these tradeoffs in mind when designing your own solution.

Inside our resources class, we want to track all the loaded resources. This means that in future load attempts, if we already have a resource from that path, we simply return it instead of loading a new copy.

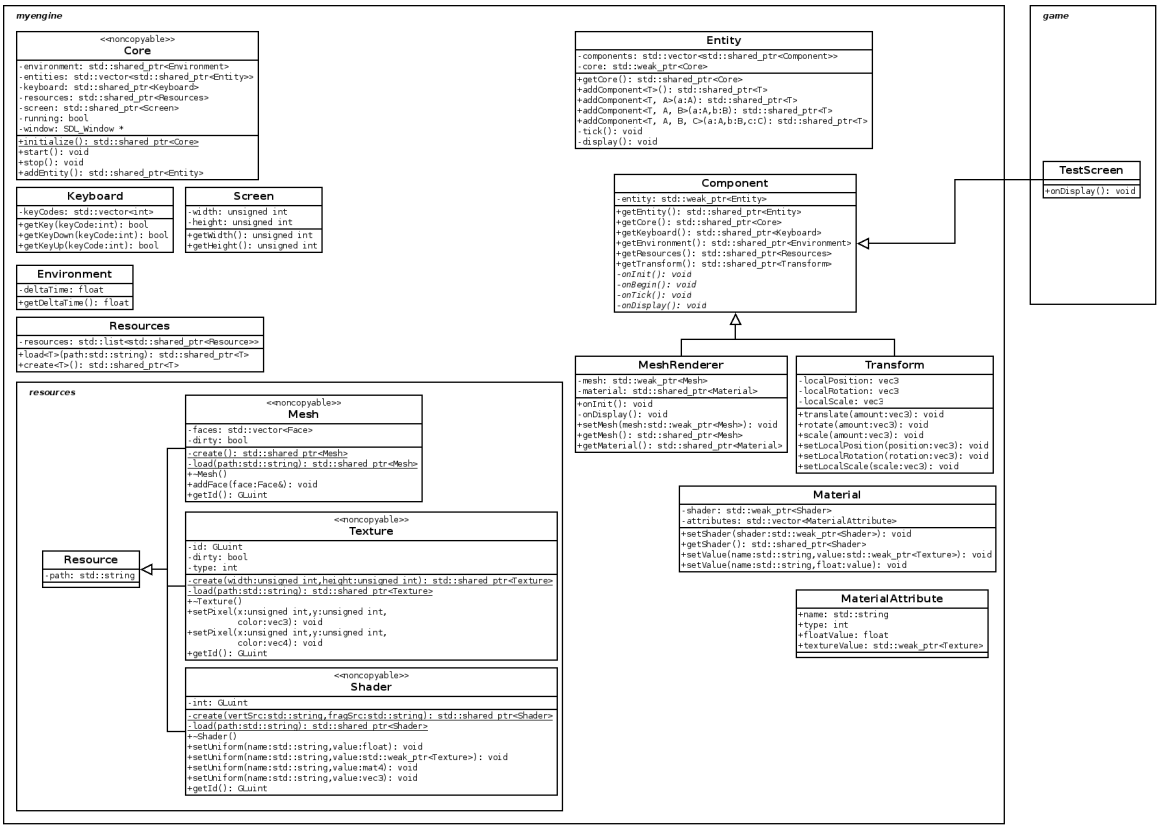

Figure 1: *The structure of an initial Resources system*

Figure ?? demonstrates that the Resources class itself has a vector of all loaded resources. This is what allows us to keep this record of what we have already loaded. This means in order to get each of our resource types (Texture, Sound, Model, etc) into the vector, we need them to all inherit from a single base type. In this example, the Resource class is used for that.

Note:
Resource and Resources are different classes. Resource is the base type that all resources inherit from whereas Resources is a container holding all the resources and related functions.

In this lab, try to implement these concepts into your very own resource loader. Your design may differ from what is covered here if you have good reason for it. Try to relate the texture loading knowledge from the lecture into creating the Texture resource class.