Resource Manager

The resource manager is a static class which is the central location which stores all sorts of resources.

**Config Values**

There is a **config** file (assets/config.json) which is loaded during the start function of the resource manager which loads and stores a bunch of configuration values such as batch capacities, maximum number of sub textures per material, etc. The values of these configuration variables can be retrieved with the **getConfigValue** and providing the **ConfigData** enum value of the variable desired.

Current configuration data:

* 3D Batch Capacity (Number of submissions)
* 2D Batch Capacity (Number of Quads)
* Maximum number of 3D vertices
* Maximum number of 3D indices
* Maximum number of a certain component type attachable per entity
* Maximum number of render passes per scene
* Maximum number of layers per scene
* Maximum number of SubTextures per material

See end of document to learn how to **add** new configuration variables to be read from the config value and loaded into the resource manager.

**Resources**

Resources store a **reference counter** which refers to how many external sources reference a particular resource. This is used to determine when to delete the resource, because if the counter reaches 0, it is no longer used by any current object and can be scheduled to be deleted.

Resources are typically loaded by loading a scene, but they can be created and then registered (**registerResource**) with the resource manager who will then manage them.

The following **types** can be managed by the resource manager:

* Vertex Buffer
* Index Buffer
* Vertex Array
* Indirect Buffer
* UniformBuffers
* FrameBuffers
* Shader Program
* Textures (Texture2D and Cubemaps)
* SubTextures
* Model3D
* Materials.

All these resources are stored in a single map stored by names. The function **getResource** is a templated function which requires the name of the resource you want; it is your responsibility to provide the correct resource type. **getResourceAndRef** gets the resource and increases the reference counter of the resource, this function is mostly used when assigning a resource to another object and the resource acknowledges it has another reference to it so the reference counter will be increased. Most objects which hold a resource require the reference counter to be increased **before** it is given to the new object.

Resources can be destroyed with the **destroyResource** function which takes a parameter of the resource name. If no argument is provided or is blank, **all** resources will be deleted.

**resourceExists** will check if the resource name is taken.

**Creating a New Resource Type**

1. Add the resource type in the ResourceType enum in resource.h
2. New resource class must inherit from resource class
   1. Class must call Resource constructor passing the resource type
3. New resource include must be added to resourceManager.h
4. The new resource can now be registered with the resource manager, see previous paragraphs on how to do so
5. If the new type of resource is being loaded from file, update resource loader to load the new resource type

**Creating a New Configuration Variable**

1. Add the new variable name to the ConfigData enum in resourceManager.h
2. Create new static unsigned integer variable in the resource manager to hold the value
   1. Make sure to initialise it in the cpp
3. Update the resource manager’s **start** function to load the value from the config file (json)
4. Update the **getConfigValue** function to return the value when the config data value is passed through as an argument