Computer Graphics

Exercises 5

1) Having multiple separate render pipelines may often be the case, but it should be reduced to the minimum number possible. In our case, we have two separate pipelines for meshes with **vertex colors** and **texture colors**. The goal of these mini-tasks is to merge both render pipelines into a single one that can handle both types of meshes.
   a) Create a material.hpp file containing a **Material** struct that holds the following:
      1) **specular**: a float that controls the total specular light
      2) **specular_shininess**: another float that controls the surface shininess
      3) **texture_contribution**: interpolates between **texture** and **vertex colors**
   b) Pass both the **uv coordinates** and **vertex colors** from the vertex to fragment shader. Use the **mix** function to mix the vertex and texture colors (named them **color_vert** and **color_tex** in the example) together, using the **texture_contribution** as the interpolation value like so:

```
vec4 color = mix(color_vert, color_tex, texture_contribution);
```

   c) Create a **Material** object for each of the cubes, one of which should have texture_contribution of 0, the other of 1. Render this using only a **single pipeline**. The Material struct will need a **bind()** function.
2) Create a **model.hpp** file containing the **Model** struct. This should be the main object for holding all the info for renderable objects, such as **Transform**, **Mesh**, **Texture**, **Material**. A single model struct should therefore also have these as member variables, instead of all being inside our **engine.hpp**. Give it the standard functions that are present on similar structures:
   void init() {} *// initialize without texture*
   void init(const std::string& tex_path) {} *// initalize with texture*
   void destroy() {} *// destroy all the members*
   void draw() {} *// bind all the members and then draw the mesh*
3) Currently, we only have a **cube** mesh with no other options within the **Mesh**
   a) Another frequently used primitive shape is a **sphere**. There are many ways to create one with different types of properties; see https://www.songho.ca/opengl/gl_sphere.html for some available options. Pick one and implement it for our mesh. Don't worry about uv-coordinates, as we do not have a texture for spheres.
   b) Since we want to keep the cube, create some sort of option that allows a choice between which mesh should be created; e.g. separated **init_sphere()** and **init_cube()** functions or an enum parameter like **init(eType mesh_type);**.