Programming Assignment 3: Shader programming
CMSC427: Computer Graphics, Fall 2020

Submission deadline: Tuesday, October 29th, 11:59pm

This project is about adding shading and texturing functionality to your rendering engine using GLSL shader programs. The major work will be in writing shaders, and then integrating them into the base code.

The deadline to submit your solution via ELMS/Canvas is Tuesday, October 29th, 11:59pm. Please submit only the *Java and shader files that you modified*. Grading will take place in a meeting with the teaching assistant, for which you need to register via the list on ELMS/Canvas.

Before you start

Examine the different shaders already contained in the base code. You can find them in the project `jrtr` in the directory `shaders`. The most relevant shader is the shader `diffuse.vert/.frag`, which implements diffuse lighting with one light source and a texture. A short explanation for the special variable types in the shaders:

- Type `uniform`: Basically, `uniform` variables in the vertex or fragment shader are passed from the Java code using the function `glUniform`. All variables that remain constant for a certain group vertices and/or pixels are passed this way. Typically, such variables include transformation matrices, information about light sources or textures. The call of `glUniform` needs to be performed after the shader has been activated but before the vertex data is drawn via `glDrawArrays` (see below).

- Type `in` in the vertex shader: The `in` variables in the vertex shader are passed from Java to the shader via so-called "Vertex Buffer Objects". The corresponding Java code is already implemented in `GLRenderContext.draw`. Rendering the data is triggered by the call to the function `glDrawArrays`. "Vertex Buffer Objects" contain all data assigned to the triangle vertices such as "positions", "normals" or "texture coordinates". It is important to note that none of this data is interpreted by OpenGL – your shader will make sense of the data. Index buffers contain indices into the list of vertices to form the actual triangles.

- Type `in` in the fragment shader, or type `out` in the vertex shader: The `in` variables of the fragment shader correspond to the `out` variables of the vertex shader. These

variables are forwarded by OpenGL from the vertex- to the fragment shader and are also automatically interpolated at each pixel.

- Type `out` in the fragment shader: These variables are written to the frame buffer. Typically, this is the color of the pixels, but many advanced techniques write out all sorts of data to different buffers. The shader does not need to write the depth values explicitly to the frame buffer, they are automatically managed. It can be done manually if desired though.

In the Java code of `jrtr` each 3D object of the type `Shape` has a reference to a `Material`, which contains the properties of the material and a reference to a shader program, with which the object is to be rendered. Examine the method `setMaterial` in the class `glRenderContext`, where the shader is activated and the material properties are passed to the `uniform` variables of the shader, as explained above. In this method, the parameters of the light sources are also passed to the shader. In the Java code, light sources are saved as objects of the class `Light`, and a list of light sources is managed in `SimpleSceneManager`.

A good summary for GLSL: GLSL Quick Reference Card. Use the Piazza forum if you have any questions or problems!

1 Diffuse Shading with Multiple Light Sources (3 points)

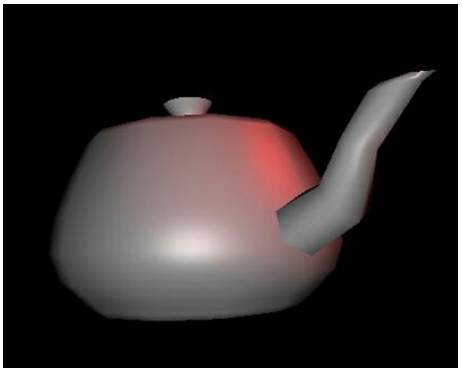
Develop a shader that computes diffuse lighting with multiple point light sources by extending (that means copying and modifying, i.e. starting from) the already existing shader `diffuse.vert/.frag`.

Demonstrate your shader by constructing a scene with various objects with different material properties (different diffuse reflection coefficients and/or textures). Extend the cylinder from the first assignment by assigning texture coordinates to it and show it in the scene, too. The texture coordinates can be assigned similarily to the texture coordinates of the cube in the class `simple` of the basecode.

Notes: The existing shader uses directional lights, but your shader is supposed to use point lights. The parameters of the light sources are saved in `uniform` variables in the shader. Use arrays to accept the data of multiple light sources. You may set the length of the arrays to a fixed number, so the maximum number of lights the shader can handle is limited. These parameters need to be passed to the shader from the host program using the correct variants of the function `glUniform*`. Code for passing light source parameters to the shader already exists in `setMaterial` of `glRenderContext`. Do note, however, that the existing code only passes the direction of directional light sources. For point light sources, you will need to pass the position of the source.

2 Per-Pixel Blinn Shading (4 points)

Develop a vertex- and fragment-shader program which additionally implements specular reflection using per-pixel Blinn shading with multiple point light sources. This shader simply adds (sums together) the contributions of the diffuse and the glossy reflections.



Teapot model with specular reflection of 2 light sources with different colors

Demonstrate your shader by constructing a scene with two light sources of different colors as shown in the figure above. You can best distinguish the two light sources if you choose very shiny material properties.

3 Texturing with Blinn Shading (3 points)

Extend your Blinn shader to support texturing and lighting in the same shader. In order to do that, simply use the colors of the texture as diffuse and ambient reflection coefficients. Also implement a "gloss map", which means using texture values to determine the specular reflection coefficient k_s. You can for example use the brightness (the sum of red, green and blue) of the texture as specular coefficient.

Demonstrate this functionality using a scene with at least two objects (or surfaces) with different textures. You can find many interesting textures by performing an image search with the queries "grass texture", "wood texture" etc. For testing, there is a teapot model with texture coordinates in the folder `obj`, which you can use.