Guocheng Qian
KAUSTID: 172525

1. Setup a glsl program: create files for the shaders (at least vertex and fragment shaders) and load them at runtime of the program. Your shaders must include variables for the camera transformation (matrix) and the lighting models (see below).

Solution:

I write a `shader.cpp` and the correspond `shader.h` file to load, complie and link the vertex and fragment shaders. Two main functions of shader: 1) loadShader; 2) setUniform.

Main steps inside loadShader : 1) glCreateShader ; 2) load source: glShaderSource; 3) glCompileShader; 4) check shader error; 5) create shader program; 6) attach shaders to the program; 7) link the vertex and fragment shader glLinkProgram.

The goal of setUniform is to set the values for glsl (shading language);

Shader language files are inside shader folder, for example the phong shading vertex shader glsl `./shader/phong.vs`, and frag shader glsl: `./shader/phong.fs`. The glsl is the shading language to tell the shaders how to process shading. For example, the ModelViewMatrix to control the objects; the NormalMatrix; the shaderModel defining the shading model (like phong lighting) used to shader.

2. Set up camera and object transformations (at least rotation, and zoom for camera and translation for objects) that can be manipulated using the mouse and keyboard.

Solution:

See `camera.h` and `shadermodel.h`;

`camera.h` provides access to the Front, UP and Position vectors of the camera. Those vectors are used to control the rotation, zoom and translation for cameras. The view matrix of the camera is obtained by: glm::lookAt(Position, Position + Front, Up);

`shadermodel.h` provides access to the Translation, Rotation vectors of an object. Those vectors are used to control the rotation and translation for objects. The model direction matrix is obtained by glm::rotate, glm::translation. A modelviewmatrix is generated and passed to the shader:

```
glm::mat4 model = shadermodel.model;

glm::mat4 mv = view * model;

obj_shader.setUniform("ModelViewMatrix", mv);
```

The function of controlling them using mouse and keyboard are implemented inside the ` main_assignment2.cpp` file and are called by the callback funcitons. A simple code example is showed below (see the cpp file for details):

```
   void processInput(GLFWwindow *window)
{ xxx
   // translate camera
   if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
      camera.ProcessKeyboard(RIGHT, deltaTime);
   // translate model
   if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS)
      shadermodel.ProcessKeyboard(Model_FORWARD, deltaTime);
  xxx
   // rotate model
   if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS)
      shadermodel.ProcessKeyboard(Model_ROT_X_PLUS, deltaTime);
}
 // zoom camera
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
   camera.ProcessMouseScroll(yoffset);
}
```

3. Implement Phong lighting+Gouraud shading [1] (the Phong lighting model is evaluated in the vertex shader).

Phong Lighting = ambient lighting + diffuse lighting + specular lighting

ambient lighting = Light.La * Material.Ka;

diffuse lighting = Light.Ld * Material.Kd * max(dot(s,n), 0.0);

specular lighting is the somehow like an advanced diffuse lighting, which takes the view direction into consideration also. Specular lighting is the strongest reflection light on the object surface. Specular lighting = Light.Ls * Material.Ks * pow(max(dot(r,v), 0.0), Material.Shininess)

(Light.L is the light intensity, which defines the light color;

Material.K is material reflectivity, which defines the object color;

n is the product of the object normal; s is the light direction;);

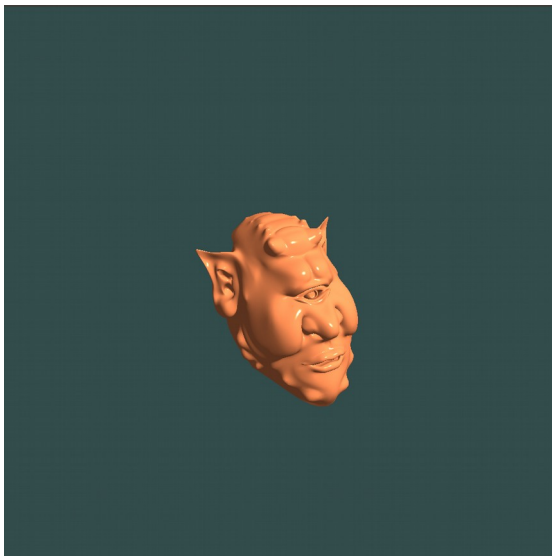
Gouraud shading is the shading when Phong lighting model is evaluated in the vertex shader, and the frag shader is as simple as setting the color to the light intensity calculated vertex color (in the shading process, the color of fragment is set to the interpolation of the corresponding vertex colors).

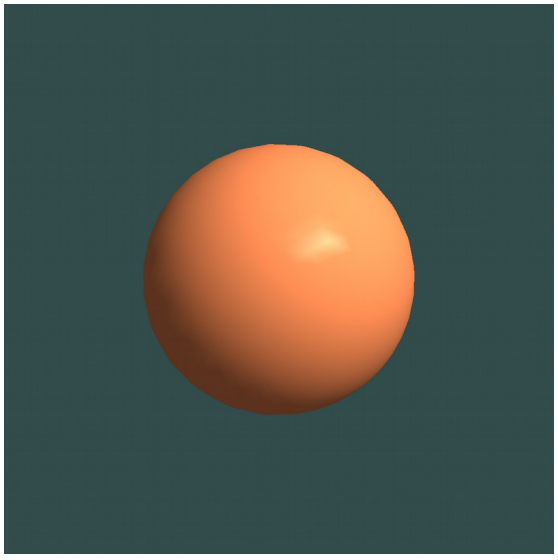
See my `shader/gouraud.vs` and `shader/gouraud.fs` for details.

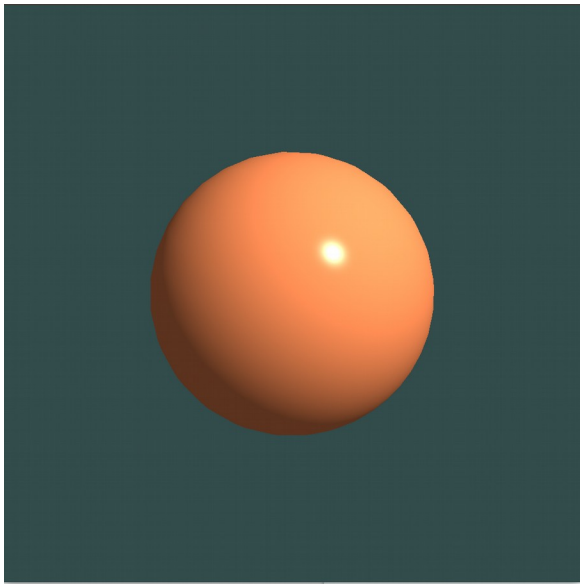
4. Implement Phong lighting+Phong shading [2] (the Phong lighting model is evaluated in the fragment shader, not in the vertex shader as for Gouraud shading).

Apply the same phong lighting model on the frag shader. See `shader/phong.vs` and `shader/phong .fs` for details.

Difference between Gouraud shading (Left) and Phong shading (Right) .







Gouraud

Phong

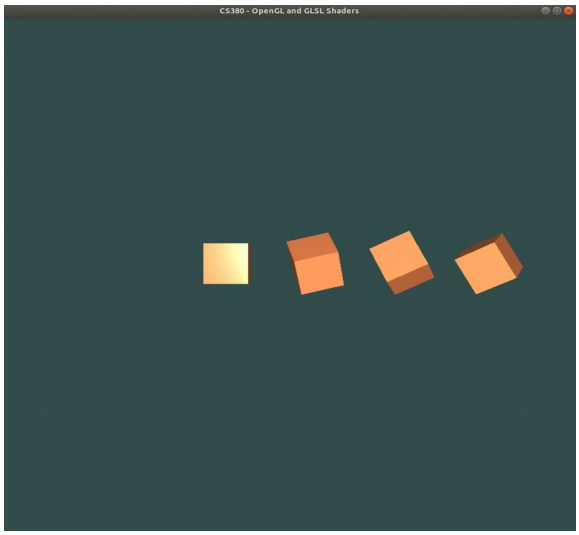
Gouraud shading is per-vertex color computation; and Phong shading is per-frag color computation. The advantage of doing lighting in the vertex shader is that it is a lot more ef?cient since there are generally a lot less vertices compared to fragments, so the (expensive) lighting calculations are done less frequently. However, the resulting color value in the vertex shader is the resulting lighting color of that vertex only and the color values of the surrounding fragments are then the result of interpolated lighting colors. The result was that the lighting was not very realistic unless large amounts of vertices were used. Phong shading givees much smoother lighting results.

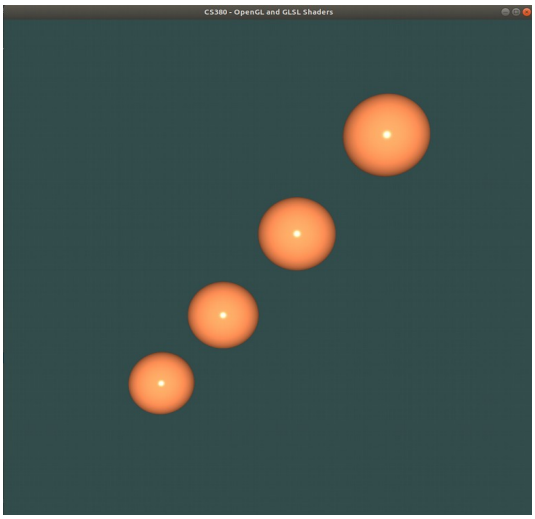
5. Implement a class that generates and stores the mesh geometry for a) a cylinder and b) a sphere.

Cylinder see `cylinder.cpp` and `cylinder .h`.

Sphere see `sphere.cpp` and `sphere.h`.

6. Render multiple instances of an object within one scene. Render the same object multiple times, applying different transformations to each instance. To achieve this you can set a different transformation matrix for each instance as a uniform variable in the vertex shader.





Set `bool multiInstance = true; ` in `main_assignment2.cpp` to activate this multi instance setting.

```
  if (multiInstance){
    for (int i=0; i<=3; i++){
        obj_shader.Use();
        // view/projection transformations
        glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)width / (float)height, 0.3f, 100.0f);
        glm::mat4 view = camera.GetViewMatrix();

        // model translation and rotation
        glm::mat4 model = glm::mat4(1.0f); // initlized by an indentity matrix.
        model = glm::translate(model, vec3(1.5*i,1.0,1.0));
        model = glm::rotate(model, glm::radians(30.0f*i), vec3(1.0f,0.0f,0.0f));
        model = glm::rotate(model, glm::radians(10.0f*i), vec3(0.0f,1.0f,0.0f));
        model = glm::rotate(model, glm::radians(10.0f*i), vec3(0.0f,0.0f,1.0f));
        glm::mat4 mv = view * model;
        obj_shader.setUniform("ModelViewMatrix", mv);
        obj_shader.setUniform("NormalMatrix", glm::mat3( vec3(mv[0]), vec3(mv[1]), vec3(mv[2]) ));
        obj_shader.setUniform("MVP", projection * mv);
        if (objectName=="sphere"){sphere->render();}
        if (objectName=="cube"){m_pCube->render();}
        if (objectName=="cylinder"){cylinder->render();}
        if (objectName=="teapot"){teapot->render();}
        if (objectName=="mesh"){m_pMesh->render();}
    }
}
```

7. Perform different kinds of procedural shading (in the fragment shader):

Implement the following procedural shaders

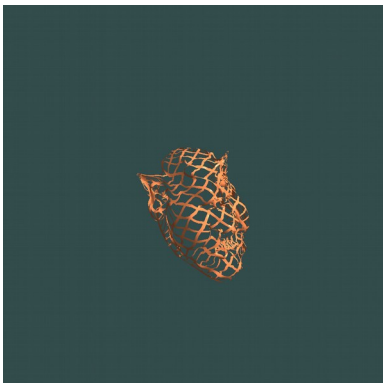
- Stripes described in chapter 11.1 of the 'OpenGL Shading Language' book (this is not the 'OpenGL 4.0 Shading Language Cookbook')

- Lattice described in chapter 11.3 of the 'OpenGL Shading Language' book (this is not the 'OpenGL 4.0 Shading Language Cookbook')

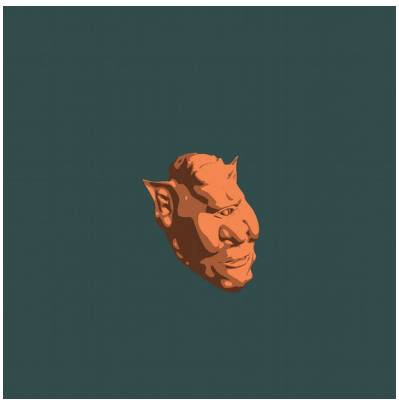
- Toon shading described in chapter 3 section 'Creating a cartoon shading effect' of the 'OpenGL 4.0 Shading Language Cookbook'

- Fog described in chapter 3 section 'Simulating fog' of the 'OpenGL 4.0 Shading Language Cookbook'

Stripes



Lattice



Toon

Fog

See each .vs and .fs inside `shader` folder for implementation details.

8. Provide key mappings to allow the user to switch between different kinds of shading methods and to set parameters for the lighting models.

Press Key 1, 2, 3, 4, 5, 6 to switch the Phong, Gouraud, Stripes, Lattice, Toon, Fog.

```
// Switch shading
if (glfwGetKey(window, GLFW_KEY_1) == GLFW_PRESS){
    shading = "gouraud";
    setupScene();
}
if (glfwGetKey(window, GLFW_KEY_2) == GLFW_PRESS){
    shading = "phong";
    setupScene();
}
if (glfwGetKey(window, GLFW_KEY_3) == GLFW_PRESS){
    shading = "stripe";
    setupScene();
}
if (glfwGetKey(window, GLFW_KEY_4) == GLFW_PRESS){
    shading = "lattice";
    setupScene();
}
if (glfwGetKey(window, GLFW_KEY_5) == GLFW_PRESS){
    shading = "toon";
    setupScene();
}
if (glfwGetKey(window, GLFW_KEY_6) == GLFW_PRESS){
    shading = "fog";
    setupScene();
}
```

Use parameters `string objectName = "mesh";` to change the lighting model.

```
if (objectName=="sphere"){sphere->render();}
if (objectName=="cube"){m_pCube->render();}
if (objectName=="cylinder"){cylinder->render();}
if (objectName=="mesh"){m_pMesh->render();}
```

9. Submit your program and a report including the comparison of Phong and Gouraud shading and results of the different procedural shading methods.

See above. Thank you for reading my report.
Guocheng