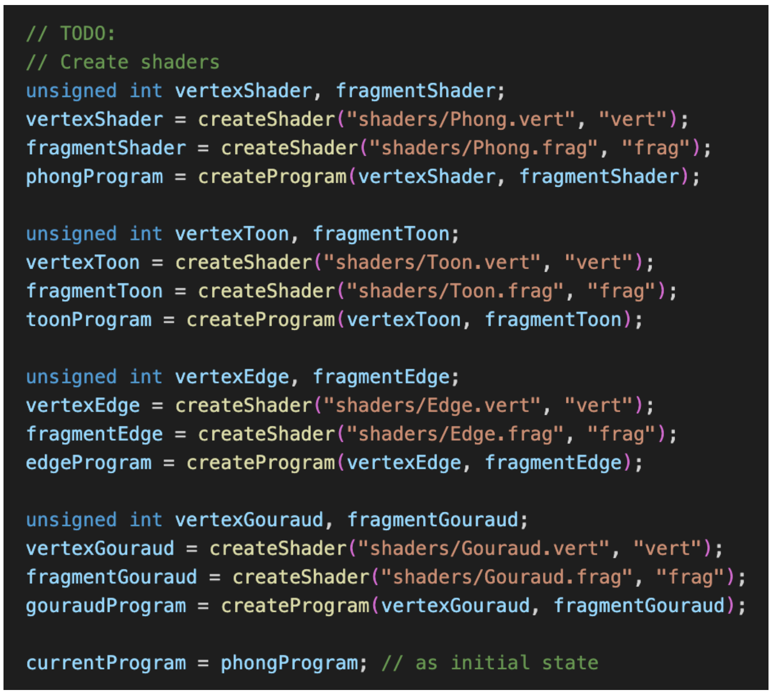
109550069 林宇柔

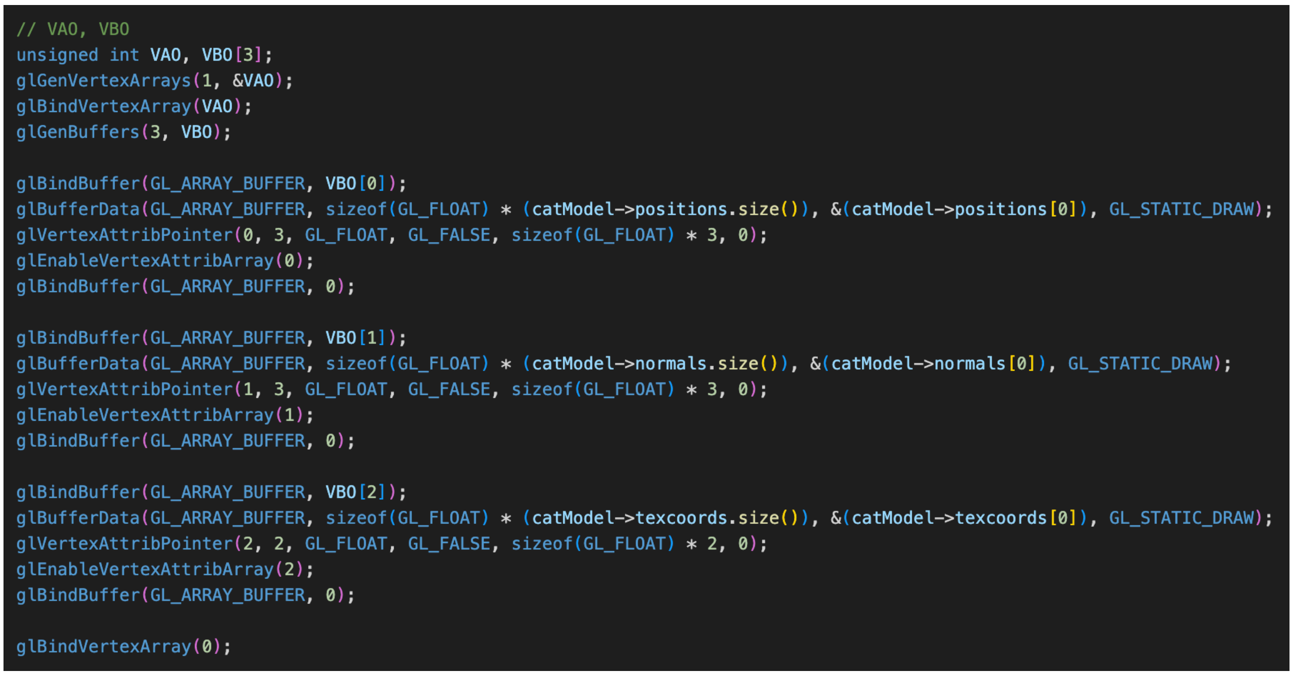
HW3_Report

I. Code

a) main.cpp

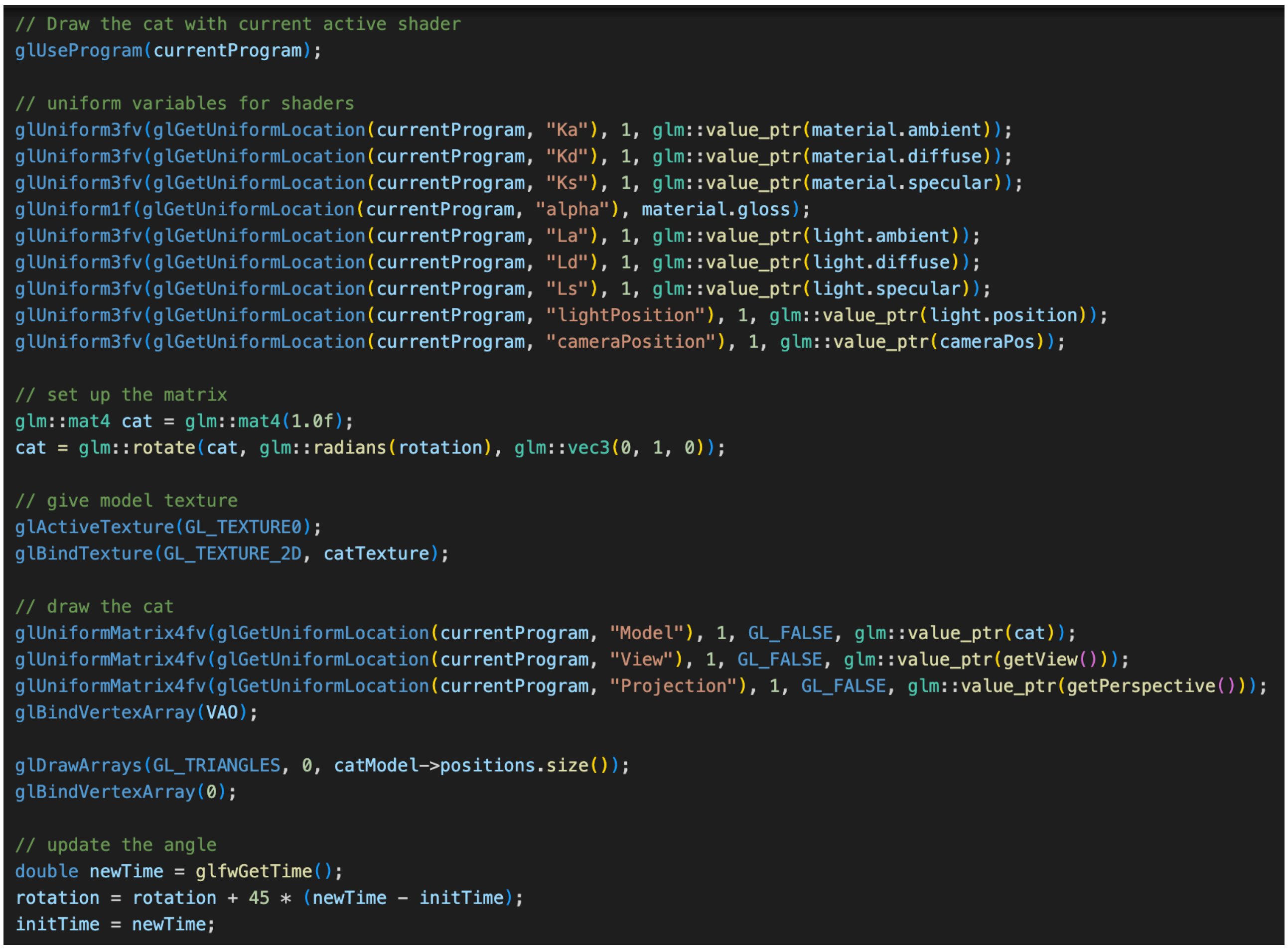


```
// TODO:
// Create shaders
unsigned int vertexShader, fragmentShader;
vertexShader = createShader("shaders/Phong.vert", "vert");
fragmentShader = createShader("shaders/Phong.frag", "frag");
phongProgram = createProgram(vertexShader, fragmentShader);

unsigned int vertexToon, fragmentToon;
vertexToon = createShader("shaders/Toon.vert", "vert");
fragmentToon = createShader("shaders/Toon.frag", "frag");
toonProgram = createProgram(vertexToon, fragmentToon);

unsigned int vertexEdge, fragmentEdge;
vertexEdge = createShader("shaders/Edge.vert", "vert");
fragmentEdge = createShader("shaders/Edge.frag", "frag");
edgeProgram = createProgram(vertexEdge, fragmentEdge);

unsigned int vertexGouraud, fragmentGouraud;
vertexGouraud = createShader("shaders/Gouraud.vert", "vert");
fragmentGouraud = createShader("shaders/Gouraud.frag", "frag");
gouraudProgram = createProgram(vertexGouraud, fragmentGouraud);

currentProgram = phongProgram; // as initial state
```

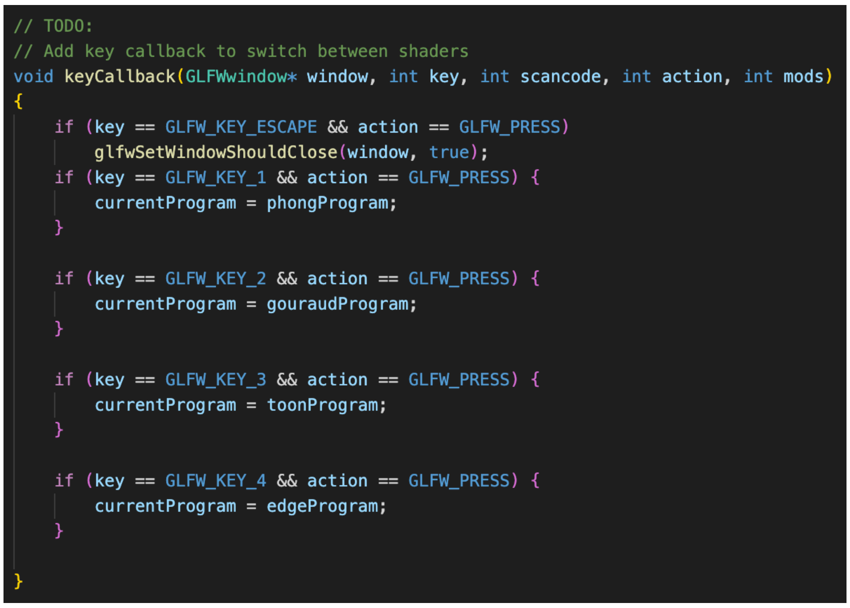


```
// VAO, VBO
unsigned int VAO, VBO[3];
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);
glGenBuffers(3, VBO);

glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * (catModel->positions.size()), &(catModel->positions[0]), GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(GL_FLOAT) * 3, 0);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, 0);

glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * (catModel->normals.size()), &(catModel->normals[0]), GL_STATIC_DRAW);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(GL_FLOAT) * 3, 0);
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, 0);

glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
glBufferData(GL_ARRAY_BUFFER, sizeof(GL_FLOAT) * (catModel->texcoords.size()), &(catModel->texcoords[0]), GL_STATIC_DRAW);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(GL_FLOAT) * 2, 0);
glEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, 0);

glBindVertexArray(0);
```

First of all I created all the shader programs that will be used in the main function, both the vertex shader and fragment shader for each effect, and I set the Phong shading effect as the default program. Then I created the VAO and VBO for the cat model, afterwards bound the vertex array with VAO and the elements positions, normals, texture coordinates with VBO.



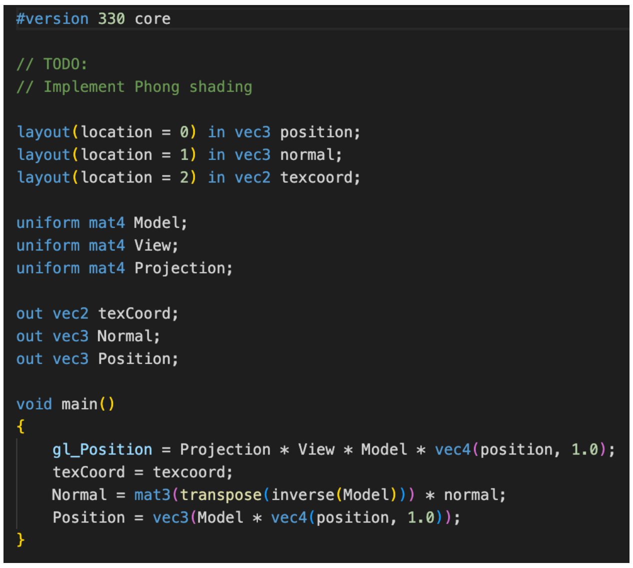
```cpp
// Draw the cat with current active shader
glUseProgram(currentProgram);

// uniform variables for shaders
glUniform3fv(glGetUniformLocation(currentProgram, "Ka"), 1, glm::value_ptr(material.ambient));
glUniform3fv(glGetUniformLocation(currentProgram, "Kd"), 1, glm::value_ptr(material.diffuse));
glUniform3fv(glGetUniformLocation(currentProgram, "Ks"), 1, glm::value_ptr(material.specular));
glUniform1f(glGetUniformLocation(currentProgram, "alpha"), material.gloss);
glUniform3fv(glGetUniformLocation(currentProgram, "La"), 1, glm::value_ptr(light.ambient));
glUniform3fv(glGetUniformLocation(currentProgram, "Ld"), 1, glm::value_ptr(light.diffuse));
glUniform3fv(glGetUniformLocation(currentProgram, "Ls"), 1, glm::value_ptr(light.specular));
glUniform3fv(glGetUniformLocation(currentProgram, "lightPosition"), 1, glm::value_ptr(light.position));
glUniform3fv(glGetUniformLocation(currentProgram, "cameraPosition"), 1, glm::value_ptr(cameraPos));

// set up the matrix
glm::mat4 cat = glm::mat4(1.0f);
cat = glm::rotate(cat, glm::radians(rotation), glm::vec3(0, 1, 0));

// give model texture
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, catTexture);

// draw the cat
glUniformMatrix4fv(glGetUniformLocation(currentProgram, "Model"), 1, GL_FALSE, glm::value_ptr(cat));
glUniformMatrix4fv(glGetUniformLocation(currentProgram, "View"), 1, GL_FALSE, glm::value_ptr(getView()));
glUniformMatrix4fv(glGetUniformLocation(currentProgram, "Projection"), 1, GL_FALSE, glm::value_ptr(getPerspective()));
glBindVertexArray(VAO);

glDrawArrays(GL_TRIANGLES, 0, catModel->positions.size());
glBindVertexArray(0);

// update the angle
double newTime = glfwGetTime();
rotation = rotation + 45 * (newTime - initTime);
initTime = newTime;
```



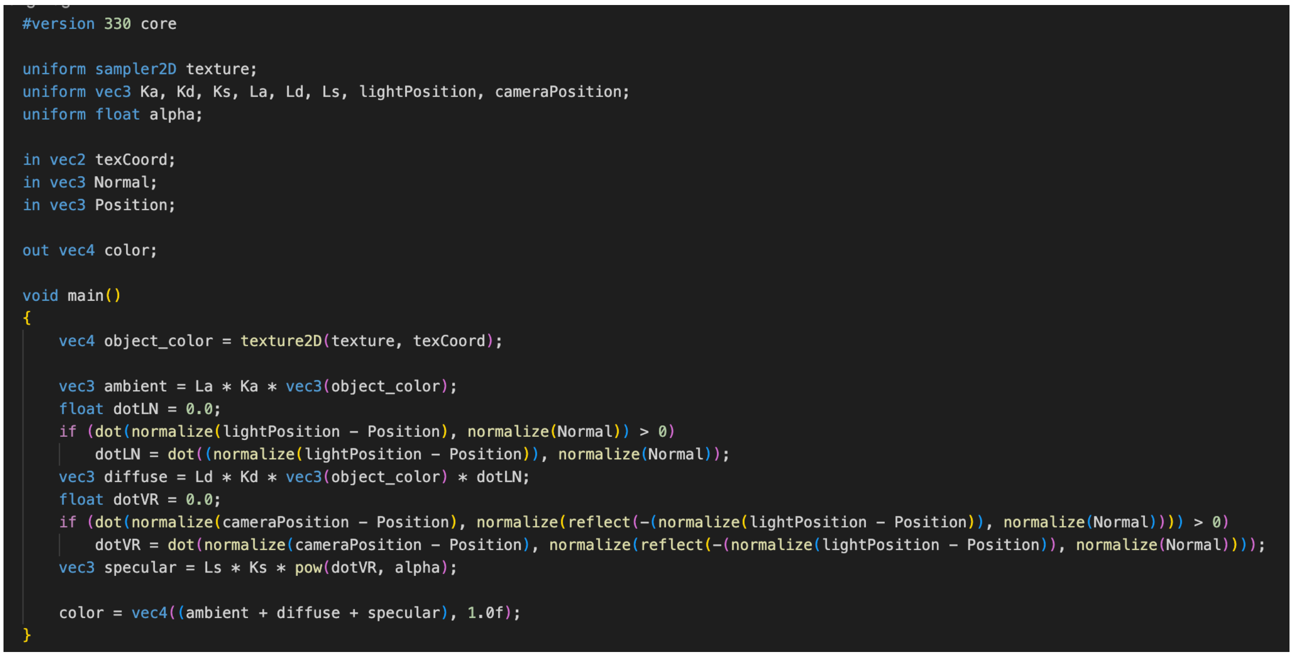
```cpp
// TODO:
// Add key callback to switch between shaders
void keyCallback(GLFWwindow* window, int key, int scancode, int action, int mods)
{
    if (key == GLFW_KEY_ESCAPE && action == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
    if (key == GLFW_KEY_1 && action == GLFW_PRESS) {
        currentProgram = phongProgram;
    }

    if (key == GLFW_KEY_2 && action == GLFW_PRESS) {
        currentProgram = gouraudProgram;
    }

    if (key == GLFW_KEY_3 && action == GLFW_PRESS) {
        currentProgram = toonProgram;
    }

    if (key == GLFW_KEY_4 && action == GLFW_PRESS) {
        currentProgram = edgeProgram;
    }

}
```

Next in the while loop, first use the current program due to possible switches, then I declared all the uniform variables that might be used in the shaders including the light and camera position. Then set up the model matrix for the cat to rotate 45 degrees per second, also give the model texture and the perspective and view matrix.
For switching, Phong, Gouraud, Toon, Effect are represented by key 1, 2, 3, 4, respectively.

b) Phong shading



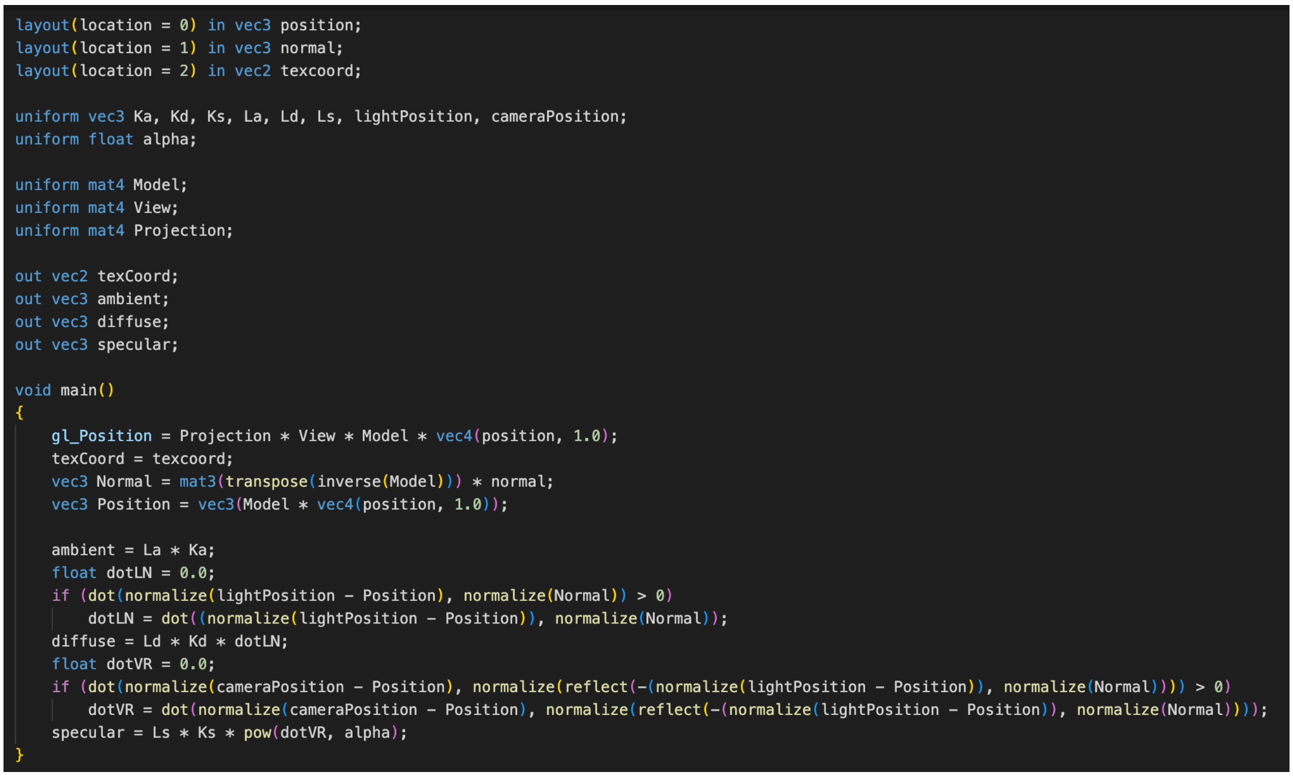
```glsl
#version 330 core

// TODO:
// Implement Phong shading

layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
layout(location = 2) in vec2 texcoord;

uniform mat4 Model;
uniform mat4 View;
uniform mat4 Projection;

out vec2 texCoord;
out vec3 Normal;
out vec3 Position;

void main()
{
    gl_Position = Projection * View * Model * vec4(position, 1.0);
    texCoord = texcoord;
    Normal = mat3(transpose(inverse(Model))) * normal;
    Position = vec3(Model * vec4(position, 1.0));
}
```



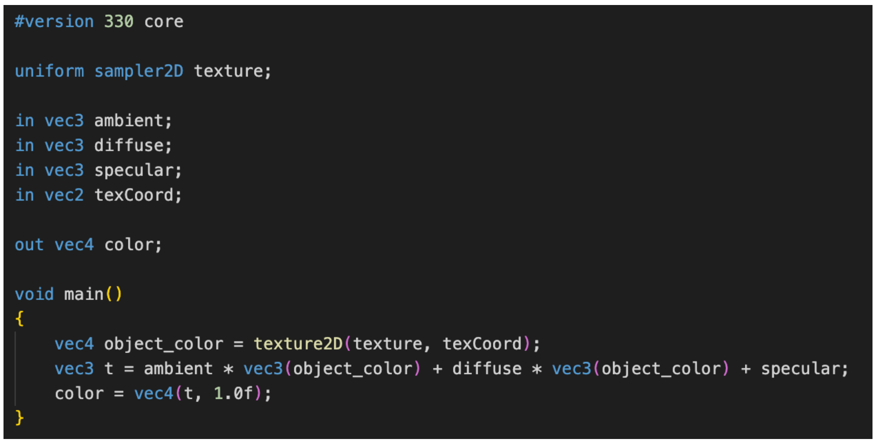
```glsl
#version 330 core

uniform sampler2D texture;
uniform vec3 Ka, Kd, Ks, La, Ld, Ls, lightPosition, cameraPosition;
uniform float alpha;

in vec2 texCoord;
in vec3 Normal;
in vec3 Position;

out vec4 color;

void main()
{
    vec4 object_color = texture2D(texture, texCoord);

    vec3 ambient = La * Ka * vec3(object_color);
    float dotLN = 0.0;
    if (dot(normalize(lightPosition - Position), normalize(Normal)) > 0)
        dotLN = dot((normalize(lightPosition - Position)), normalize(Normal));
    vec3 diffuse = Ld * Kd * vec3(object_color) * dotLN;
    float dotVR = 0.0;
    if (dot(normalize(cameraPosition - Position), normalize(reflect(-(normalize(lightPosition - Position)), normalize(Normal)))) > 0)
        dotVR = dot(normalize(cameraPosition - Position), normalize(reflect(-(normalize(lightPosition - Position)), normalize(Normal))));
    vec3 specular = Ls * Ks * pow(dotVR, alpha);

    color = vec4((ambient + diffuse + specular), 1.0f);
}
```

In its vertex shader, set the gl_Position with the uniform matrices Projection, View, and Model and make it a 4-element vector, then assign the texture coordination, the normal, and the fragment position.
In the fragment shader, use a vector to store the texture of the cat before applying the shading. Then calculate the ambient with the uniform variables passed in. As for diffuse, I declared a float to store dot(L, N) and initialized it to zero, because then it could be set to zero if the value calculated is negative. In my calculation, L stands for the difference of position between the light and fragment normalized, while N is the normal normalized. Next for the specular, I did the same thing declaring a float dotVR, V stands for the difference of position between the camera and fragment normalized, while R is the reflection of the light (which is L). After the three elements are calculated, add them up and store it as the result color.

c) Gouraud shading



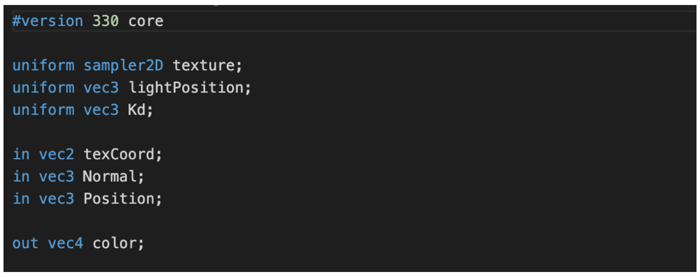
```
layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
layout(location = 2) in vec2 texcoord;

uniform vec3 Ka, Kd, Ks, La, Ld, Ls, lightPosition, cameraPosition;
uniform float alpha;

uniform mat4 Model;
uniform mat4 View;
uniform mat4 Projection;

out vec2 texCoord;
out vec3 ambient;
out vec3 diffuse;
out vec3 specular;

void main()
{
    gl_Position = Projection * View * Model * vec4(position, 1.0);
    texCoord = texcoord;
    vec3 Normal = mat3(transpose(inverse(Model))) * normal;
    vec3 Position = vec3(Model * vec4(position, 1.0));

    ambient = La * Ka;
    float dotLN = 0.0;
    if (dot(normalize(lightPosition - Position), normalize(Normal)) > 0)
        dotLN = dot((normalize(lightPosition - Position)), normalize(Normal));
    diffuse = Ld * Kd * dotLN;
    float dotVR = 0.0;
    if (dot(normalize(cameraPosition - Position), normalize(reflect(-(normalize(lightPosition - Position)), normalize(Normal)))) > 0)
        dotVR = dot(normalize(cameraPosition - Position), normalize(reflect(-(normalize(lightPosition - Position)), normalize(Normal))));
    specular = Ls * Ks * pow(dotVR, alpha);
}
```



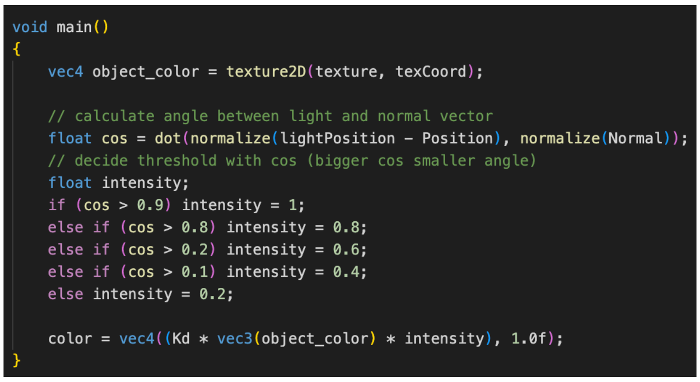
```
#version 330 core

uniform sampler2D texture;

in vec3 ambient;
in vec3 diffuse;
in vec3 specular;
in vec2 texCoord;

out vec4 color;

void main()
{
    vec4 object_color = texture2D(texture, texCoord);
    vec3 t = ambient * vec3(object_color) + diffuse * vec3(object_color) + specular;
    color = vec4(t, 1.0f);
}
```

In Gouraud shading's vertex shader, I did the exact same thing as what I did in the Phong shading's fragment shader, except that I didn't multiply the sum of the three elements with the texture, and only passed the summation to the fragment shader. In its fragment shader, I timed the summation with the texture.

d) Toon shading



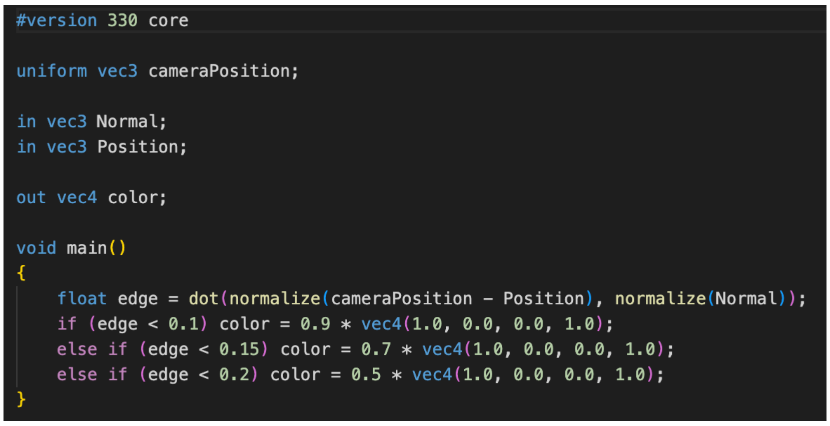
```
#version 330 core

uniform sampler2D texture;
uniform vec3 lightPosition;
uniform vec3 Kd;

in vec2 texCoord;
in vec3 Normal;
in vec3 Position;

out vec4 color;
```



```
void main()
{
    vec4 object_color = texture2D(texture, texCoord);

    // calculate angle between light and normal vector
    float cos = dot(normalize(lightPosition - Position), normalize(Normal));
    // decide threshold with cos (bigger cos smaller angle)
    float intensity;
    if (cos > 0.9) intensity = 1;
    else if (cos > 0.8) intensity = 0.8;
    else if (cos > 0.2) intensity = 0.6;
    else if (cos > 0.1) intensity = 0.4;
    else intensity = 0.2;

    color = vec4((Kd * vec3(object_color) * intensity), 1.0f);
}
```

For Toon shading, the vertex shader is the same as the Phong shading one, so I'll focus on the fragment shader. Still, I declared a variable for storing the texture. Then by comparing the angle from dot(L, N), give the intensity to the part, with bigger cosine value, I assign them with higher intensity, and I decided 5 levels in total.

e) Edge effect



```
#version 330 core

uniform vec3 cameraPosition;

in vec3 Normal;
in vec3 Position;

out vec4 color;

void main()
{
    float edge = dot(normalize(cameraPosition - Position), normalize(Normal));
    if (edge < 0.1) color = 0.9 * vec4(1.0, 0.0, 0.0, 1.0);
    else if (edge < 0.15) color = 0.7 * vec4(1.0, 0.0, 0.0, 1.0);
    else if (edge < 0.2) color = 0.5 * vec4(1.0, 0.0, 0.0, 1.0);
}
```

For edge effect, the difference between its vertex shader and the one of Phong shading's is that this time the texture coordinate not needed. As for its fragment shader, I get the edge by calculating the dot of difference of position between camera and fragment normalized and the normal normalized. There would be color if the value is smaller than 2, the smaller the value the darker the red line will be.

II. Problems Encountered

For the same reason as HW2, I wasn't able to finish the homework on my own computer, so I went for the computer center at the first place, however, the computers there were rebooting, and the TAs there said it would last more than a week, so I wasn't able to do my homework again. At last I borrowed my friend's computer and finished this assignment as soon as possible.
When I was implementing Gouraud shading, the texture seemed blurred at some point, then I figured that I shouldn't implement the texture part in my vertex shader, after I changed that everything went fine.