Computer Graphics HW3 Report

0716234 蕭彧

```
glm::vec3 WorldLightPos = glm::vec3(2, 5, 5);
glm::vec3 WorldCamPos = glm::vec3(7.5, 5.0, 7.5);
glm::vec3 Ka = glm::vec3(1, 1, 1);
glm::vec3 Kd = glm::vec3(1, 1, 1);
glm::vec3 Ks = glm::vec3(1, 1, 1);
glm::vec3 La = glm::vec3(0.2, 0.2, 0.2);
glm::vec3 Ld = glm::vec3(0.8, 0.8, 0.8);
glm::vec3 Ls = glm::vec3(0.5, 0.5, 0.5);
float gloss = 25.0;
```

First of all, I declare all the needed parameters such as world coordinate, Ka and gloss and so on.

```
ModelMatrixID = glGetUniformLocation(program, "WorldLightPos");
glUniform3fv(ModelMatrixID, 1, &WorldLightPos[0]);

ModelMatrixID = glGetUniformLocation(program, "WorldCamPos");
glUniform3fv(ModelMatrixID, 1, &WorldCamPos[0]);

ModelMatrixID = glGetUniformLocation(program, "Ka");
glUniform3fv(ModelMatrixID, 1, &Ka[0]);
ModelMatrixID = glGetUniformLocation(program, "Kd");
glUniform3fv(ModelMatrixID, 1, &Kd[0]);
ModelMatrixID = glGetUniformLocation(program, "Ks");
glUniform3fv(ModelMatrixID, 1, &Ks[0]);

ModelMatrixID = glGetUniformLocation(program, "La");
glUniform3fv(ModelMatrixID, 1, &La[0]);
ModelMatrixID = glGetUniformLocation(program, "Ld");
glUniform3fv(ModelMatrixID, 1, &Ld[0]);
ModelMatrixID = glGetUniformLocation(program, "Ls");
glUniform3fv(ModelMatrixID, 1, &Ls[0]);

ModelMatrixID = glGetUniformLocation(program, "gloss");
glUniform1f(ModelMatrixID, gloss);
```

Next, I use uniform to pass all the needed parameters to the vertex/fragment shader.

```
#version 430

layout(location = 0) in vec3 in_position;
layout(location = 1) in vec3 normal_in;
layout(location = 2) in vec2 texcoord;

uniform mat4 M, V, P;
out vec2 Texcoord;
out vec3 FragNormal;
out vec3 FragPos;

void main() {
  gl_Position = P * V * M * vec4(in_position, 1.0);
  FragPos = vec3(M * vec4(in_position, 1.0));
  Texcoord = texcoord;
  mat3 normalMatrix = mat3(transpose(inverse(M)));
  FragNormal = normalMatrix * normal_in;

}
```

I will first introduce the concept of phone shading. Here is the vertex shader of it. Let's focus on main function. Because object's positions are change by model matrix, I need to multiple model matrix with object's positions and then transfer to the fragment shader. For normal, I can't multiple model matrix with it directly because normal is a vector. I need to do some math transformation on model matrix and then multiple with normal. So far, I have transformed model's positions and normal and then passed them to fragment shader properly. The

rest part will be the job of fragment shader.

```
#version 430

uniform sampler2D texture;
uniform vec3 WorldLightPos, WorldCamPos;
uniform vec3 Ka, Kd, Ks, La, Ld, Ls;
uniform float gloss;

in vec2 Texcoord;
in vec3 FragNormal;
in vec3 FragPos;

out vec4 color;
void main()
{
    vec4 object_color = texture2D(texture, Texcoord);

    vec3 ambient = Ka * La * vec3(object_color);

    vec3 lightDir = normalize(WorldLightPos - FragPos);
    vec3 normal = normalize(FragNormal);
    float diffFactor = max(dot(lightDir, normal), 0.0);
    vec3 diffuse =  Kd * Ld * diffFactor * vec3(object_color);

    vec3    reflectDir = normalize(reflect(-lightDir, normal));
    vec3    viewDir = normalize(WorldCamPos - FragPos);
    float   specFactor = pow(max(dot(reflectDir, viewDir), 0.0), gloss);
    vec3    specular = Ks * Ls * specFactor * vec3(object_color);

    vec3 result = ambient + diffuse + specular;

    color = vec4(result, 1.0f);
}
```

Here is the fragment shader of phone shading. On top of the program I get the parameters transferred by uniform. In the main function, I first bind the texture and the rest part is to calculate the ambient, diffuse and specular of phone shading. The ambient is pretty simple. Just multiple Ka, La and object's color then the result is the ambient light. Next, the formula of

diffuse is Kd*Ld*(l*n). l is light vector and n is the normal of the position. I use dot function to get the dot value of them. The effect of max is that if your dot value is negative, it will be 0 because the light can not be negative. Last, the formula of specular light is Ks*Ls*(v*r)^gloss. V is the vector of camera position, r is the reflect vector of light and gloss is the specular coefficient of the object. Same as diffuse, I use dot function to calculate v and r and then power them with gloss. I finish calculating ambient, diffuse and specular. The last thing to do is to add them together and then output the result. Be caution that the calculation above is vec3 so I need to transform result to vec4.

```
#version 430

uniform sampler2D texture;
uniform vec3 WorldLightPos;
uniform vec3 Kd;

in vec2 Texcoord;
in vec3 FragNormal;
in vec3 FragPos;

float intensity;

out vec4 color;
void main()
{
    vec4 object_color = texture2D(texture, Texcoord);

    vec3 lightDir = normalize(WorldLightPos - FragPos);
    vec3 normal = normalize(FragNormal);
    float level = dot(lightDir, normal);

    if (level > 0.95)
        intensity = 1;
    else if (level > 0.75)
        intensity = 0.8;
    else if (level > 0.5)
        intensity = 0.6;
    else if (level > 0.25)
        intensity = 0.4;
    else
        intensity = 0.2;

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

Next, I will talk about toon shading. The vertex shader of it is as same as phone shading, so here is the fragment shader of toon shading. Toon shading is to calculate the angle of light vector and normal vector and then according the value of angle to give different intensity to color. Here I use five level of

intensity. The value of intensity is depended on the cos of the angle. The bigger the cos is, the smaller the angle of the two vector is. Last, I multiple the intensity with the color and Kd and this is the result of the color.

```
#version 430

uniform vec3 WorldCamPos;

in vec3 FragNormal;
in vec3 FragPos;
float show;

vec4 red = vec4(0.0, 0.0, 1.0, 1.0);
out vec4 color;

void main()
{
    vec3 normal = normalize(FragNormal);

    vec3 CameraDir = normalize(WorldCamPos - FragPos);
    float edge = dot(CameraDir, normal);

    if (edge < 0.2)
        show = 1.0;
    else
        show = 0.0;

    color = show * red;
}
```

Last thing I'll talk about is the edge effect of the model. This is the simple one. To only draw the edge, you need to calculate dot value of camera vector and normal vector. If the value is close to 0, then it is an edge of this camera position. So what I do is I calculate the dot value of the two vectors first and

then use “if” statement to verify that isn’t the dot value smaller than 0.2. If the answer is yes, I’ll give a color on that point. By the check, I will only draw the edge of the object.

The problem I met and the solution

The problem I met is how to implement the edge effect. I first think that the cos value of the angle between camera and normal need to be 0 and that is where the edge is. So I check that whether the dot value of two two vectors is 0, and then it shows nothing on the screen :). After that I check if I have some typo or what because I believe my opinion is right. After a while, I find that if the dot value is 0 it can not be seen on screen because it is edge right but you can’t see it from the front. You can only see the edge on the side. So I modify my program from checking the dot value is 0 to check isn’t the dot value is small than a constant value. I then see the edge on the screen. I finally set my constant value to 0.2 because I think it is the best value to see the edge clearly.