

Basic:

1. 绘制一个 cube, vertex 的数据格式如下所示:

```
float vertices[] = {
    -2.0f, -2.0f, -2.0f,  0.4f, 0.5f, 0.8f,  0.0f, 0.0f, -1.0f,
    2.0f, -2.0f, -2.0f,  0.4f, 0.5f, 0.8f,  0.0f, 0.0f, -1.0f,
    2.0f,  2.0f, -2.0f,  0.4f, 0.5f, 0.8f,  0.0f, 0.0f, -1.0f,
    2.0f,  2.0f, -2.0f,  0.4f, 0.5f, 0.8f,  0.0f, 0.0f, -1.0f,
    -2.0f,  2.0f, -2.0f,  0.4f, 0.5f, 0.8f,  0.0f, 0.0f, -1.0f,
    -2.0f, -2.0f, -2.0f,  0.4f, 0.5f, 0.8f,  0.0f, 0.0f, -1.0f,
```

前三个为顶点坐标，中间 3 个为面的颜色（本作业中忽略掉），最后为法向量；

- ## 2.分别实现 Phong Shader 和 Gouraud Shader:

```
// shader
Shader phongShader("phong.vs", "phong.fs");
Shader gouraudShader("gouraud.vs", "gouraud.fs");
Shader lightShader("light.vs", "light.fs");
```

Phong Shading:

按照教程中的写法完成 vs 和 fs 文件即可，其中，phongShading 是在 frag 着色器中进行光照颜色的计算的：

```
vs      phong.fs  ✕  main.cpp      camera.hpp

uniform float ambientStrength;
uniform float specularStrength;
uniform float shininess; // 32

void main()
{
    // FragColor = texture(texture1, TexCoord);
    // ambient
    // float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

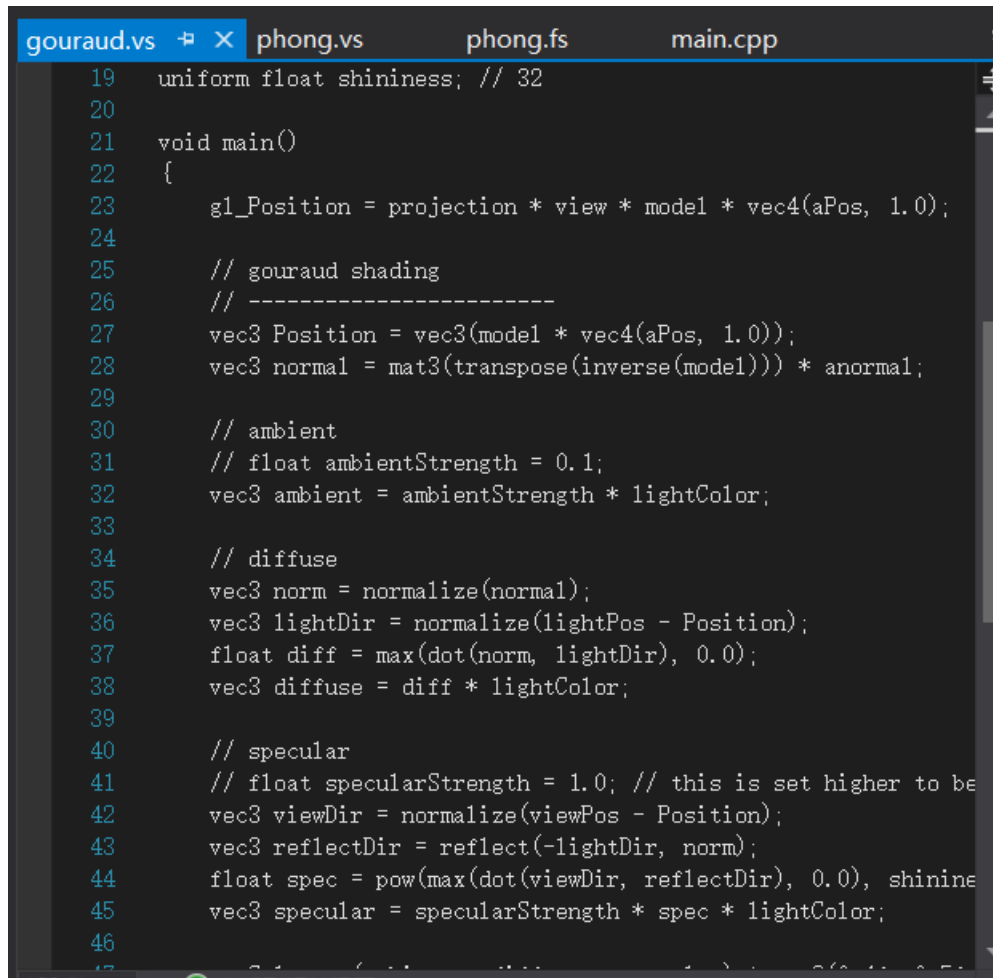
    // diffuse
    vec3 norm = normalize(normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diff * lightColor;

    // specular
    // float specularStrength = 0.5;
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
    vec3 specular = specularStrength * spec * lightColor;

    vec3 result = (ambient + diffuse + specular) * outColor;
    FragColor = vec4(result, 1.0);
}
```

Gouraud Shading:

Gouraud Shading 的颜色计算发生在顶点着色器中:

A screenshot of a code editor with a dark theme. The editor has four tabs at the top: 'gouraud.vs' (active), 'phong.vs', 'phong.fs', and 'main.cpp'. The 'gouraud.vs' tab contains C++ code for a vertex shader. The code includes a uniform 'float shininess' and a 'void main()' function. Inside 'main()', it calculates 'gl_Position' using a projection, view, and model matrix. It then performs Gouraud shading calculations: it calculates a position vector 'Position' and a normal vector 'normal' from the model matrix. It calculates an ambient color 'ambient' based on 'ambientStrength' and 'lightColor'. It calculates a diffuse color 'diffuse' based on the dot product of the normalized normal and light direction. It calculates a specular color 'specular' based on the dot product of the view direction and the reflection of the light direction. Finally, it calculates the output color 'outColor' as the sum of ambient, diffuse, and specular colors, scaled by 'shininess'.

```
19 uniform float shininess; // 32
20
21 void main()
22 {
23     gl_Position = projection * view * model * vec4(aPos, 1.0);
24
25     // gouraud shading
26     // -----
27     vec3 Position = vec3(model * vec4(aPos, 1.0));
28     vec3 normal = mat3(transpose(inverse(model))) * anormal;
29
30     // ambient
31     // float ambientStrength = 0.1;
32     vec3 ambient = ambientStrength * lightColor;
33
34     // diffuse
35     vec3 norm = normalize(normal);
36     vec3 lightDir = normalize(lightPos - Position);
37     float diff = max(dot(norm, lightDir), 0.0);
38     vec3 diffuse = diff * lightColor;
39
40     // specular
41     // float specularStrength = 1.0; // this is set higher to be
42     vec3 viewDir = normalize(viewPos - Position);
43     vec3 reflectDir = reflect(-lightDir, norm);
44     float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
45     vec3 specular = specularStrength * spec * lightColor;
46
47     outColor = (ambient + diffuse + specular) * vec3(0.4f, 0.5f, 0.8f);
```

计算完 color 之后传给 frag 着色器:

```
outColor = (ambient + diffuse + specular) * vec3(0.4f, 0.5f, 0.8f);
```

实现原理:

Gouraud shading 是先计算顶点的颜色, 然后根据顶点信息, 对中间的颜色进行二维的插值。因此其代码直接在顶点着色器中实现。

Phong shading 对每个片段 (fragment) 计算光照, 点的法向量通过顶点的法向量插值得到。因此其代码在片段着色器中实现, 其开销会比 Gouraud 要大。

3. 设置 GUI 以修改参数:

```
ImGui::SliderFloat("ambient", &ambient, 0.0f, 1.0f);
ImGui::SliderFloat("specular", &specular, 0.0f, 2.0f);
ImGui::SliderInt("shininess", &shininess, 10, 50);
```

因为 diffuse 是计算求出来的, 因此无需对其进行改变调节。

用一个引用来切换两种 Shader:

```
Shader& shader = phongShader;
```

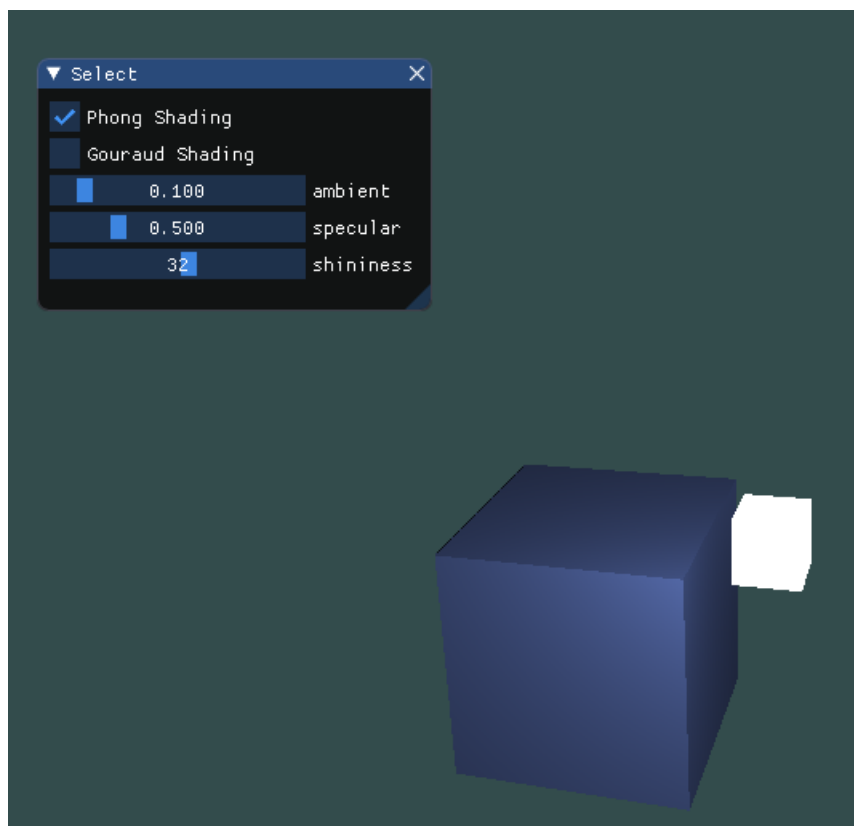
```
if (select1) {  
    shader = phongShader;  
}  
else if (select2) {  
    shader = gouraudShader;  
}
```

Bonus:

更改 lightPos 的值来使光源移动，使其绕中间的 cube 旋转：

```
lightPos.x = 6 * cos(glFWGetTime() * 1.0f);  
lightPos.z = 6 * sin(glFWGetTime() * 1.0f);
```

结果截图：



通过 GUI 可以切换两种 shading，并且调节参数。