Basic

1. 实现 Phong 光照模型

- 场景中绘制一个cube
- 自己写shader实现两种shading：Phong Shading 和 Gouraud Shading，并解释两种shading的实现原理

  在写实现两种shading之前首先实现一个Phong光照模型。

  Phong光照模型由三部分组成：

  - Ambient Light 环境光

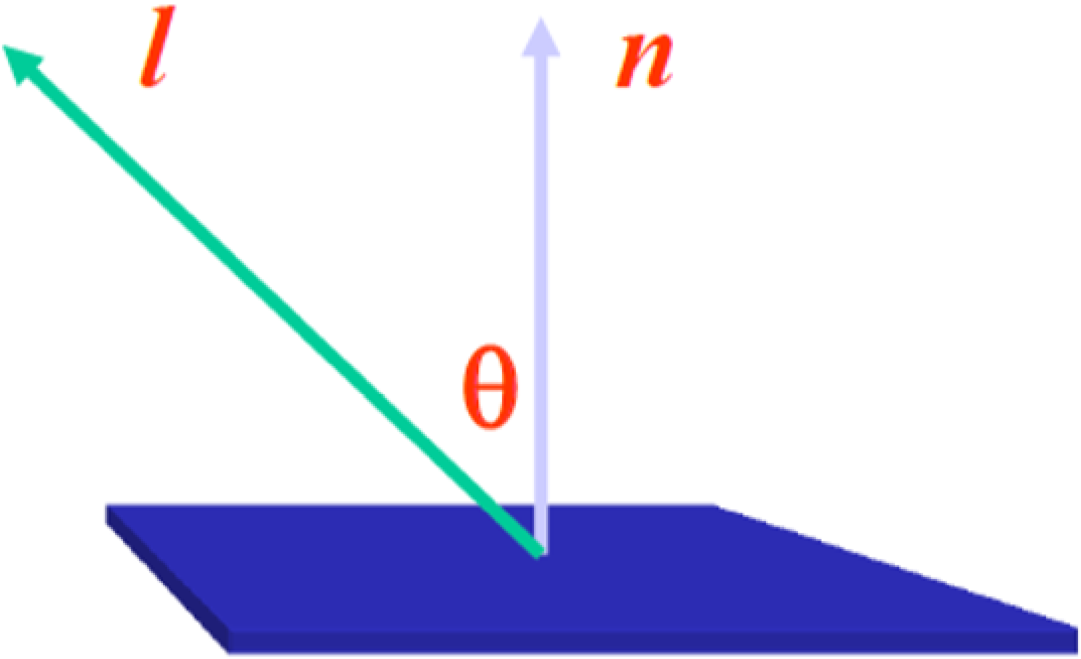
    $I = K_a I_a$

    I_a -> 环境光的强度

    K_a -> 环境光反射系数

    ```
    // in Source.cpp
    unsigned int ambientStrengthLoc =
    glGetUniformLocation(ourShader.Program, "ambientStrength");

    glUniform1f(ambientStrengthLoc, ambientStrength);


    // in shader
    uniform float ambientStrength;
    uniform vec3 lightColor;

    vec3 ambient = ambientStrength * lightColor;
    ```

  - Diffuse 漫反射



$I_d = K_d I_e cos\alpha = K_d I_e (n \cdot l)$

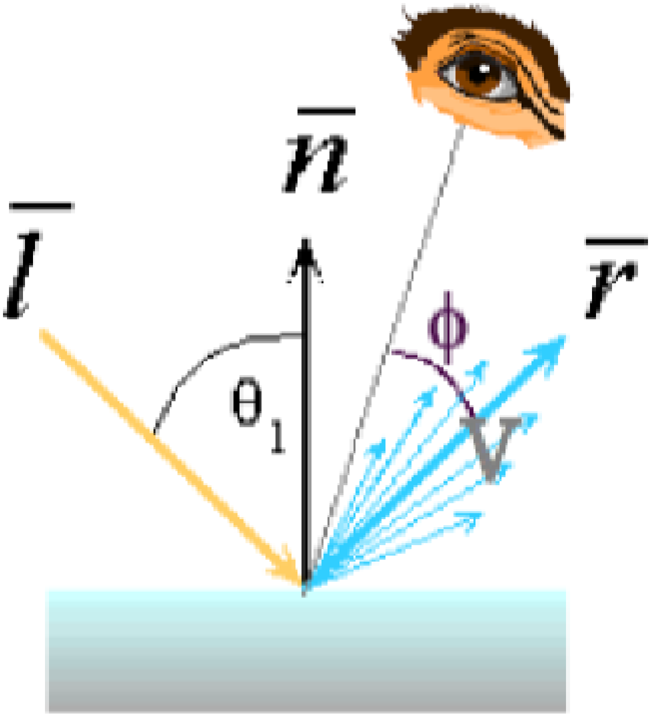
K_d -> 漫反射的反射系数

I_e -> 入射光强度

$cos\alpha$ -> 入射光（方向指向光源）与法向量的夹角

```
// in Source.cpp
unsigned int diffuseStrengthLoc =
glGetUniformLocation(ourShader.Program, "diffuseStrength");
glUniform1f(diffuseStrengthLoc, diffuseStrength);

// in shader
uniform float diffuseStrength;
float diff = max(dot(norm, lightDir), 0.0f);
vec3 diffuse = diffuseStrength * diff * lightColor;
```

- Specular 镜面反射



$I_{specular} = K_s I_e(\bar{v} \cdot \bar{r})^{n_{shiny}}$

K_s -> 镜面反射的反射系数

I_e -> 入射光强度

$\bar{v}$ -> 指向观察者的向量

$\bar{r}$ -> 反射光的向量

n_{shiny} -> 材质发光常数

```
unsigned int specularStrengthLoc =
glGetUniformLocation(ourShader.Program, "specularStrength");
unsigned int nshinessLoc = glGetUniformLocation(ourShader.Program,
"nshiness");
glUniform1f(specularStrengthLoc, specularStrength);
glUniform1i(nshinessLoc, nshiness);


// in shader
uniform float specularStrength;
uniform int nshiness;

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

需要注意代入计算的方向向量都要先正则化

接着利用上面的Phong光照模型来分别实现Phong Shading 和 Gouraud Shading

- Phong Shading

  在物体表面的每一个像素都用Phong光照模型计算出该像素的颜色值。

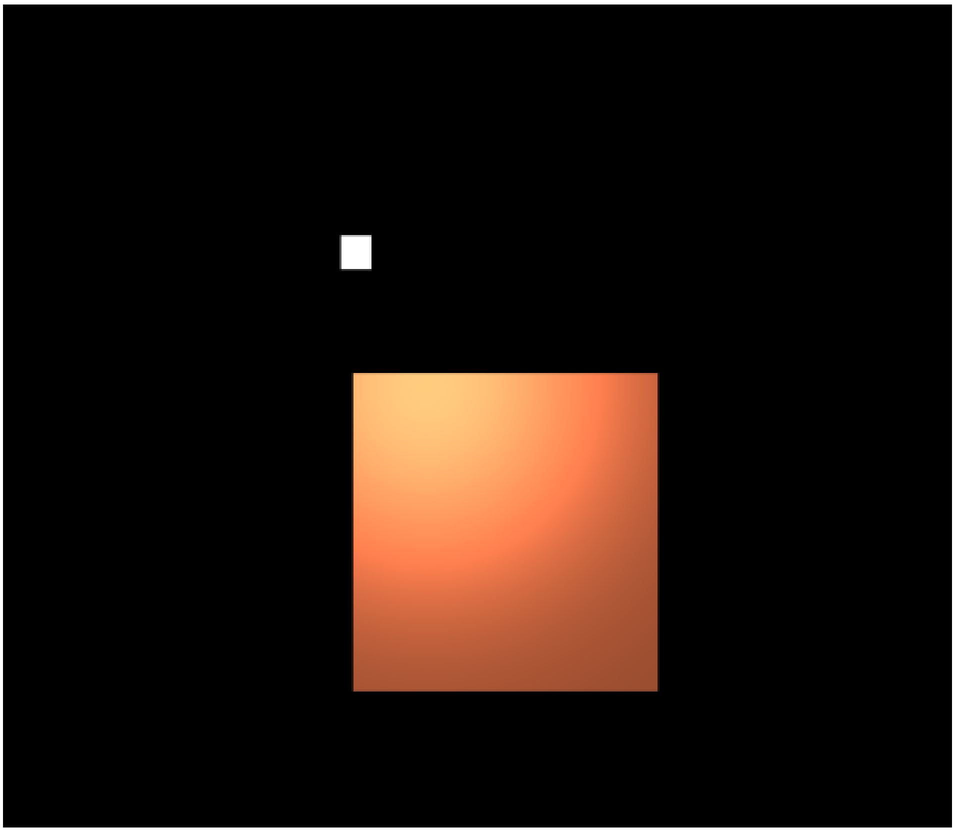
  在OpenGL里面，实现Phong Shading就需要在片段着色器中实现Phong光照模型，原因是片段着色器中计算的颜色值是物体表面每一个像素的颜色值。

```
// shader.vs 顶点着色器
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 normal;

uniform int chooseVs;

uniform mat4 transform;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

out vec3 mycolor;
out vec2 texcoord;

out vec3 FragPos;
out vec3 Normal;

void main(){
    gl_Position = projection * view * model * vec4(position,1.0f);
    FragPos = vec3(model * vec4(position, 1.0f));
    //Normal = normal;
    Normal = mat3(transpose(inverse(model))) * normal;
}
```

```
// shader.frag 片段着色器
#version 330 core

in vec3 Normal;
in vec3 FragPos;

uniform int chooseFrag;

uniform vec3 objectColor;
uniform vec3 lightColor;
uniform vec3 lightPos;
uniform vec3 viewPos;

uniform float ambientStrength;
uniform float diffuseStrength;
```

```
uniform float specularStrength;
uniform int nshiness;

void main(){
    // Ambient
    //float ambientStrength = 0.1f;
    vec3 ambient = ambientStrength * lightColor;

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

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

    vec3 result = (ambient + diffuse + specular) * objectColor;
    FragColor = vec4(result, 1.0f);
    //FragColor = vec4(0.0f, 1.0f, 1.0f, 1.0f);
}
```

效果



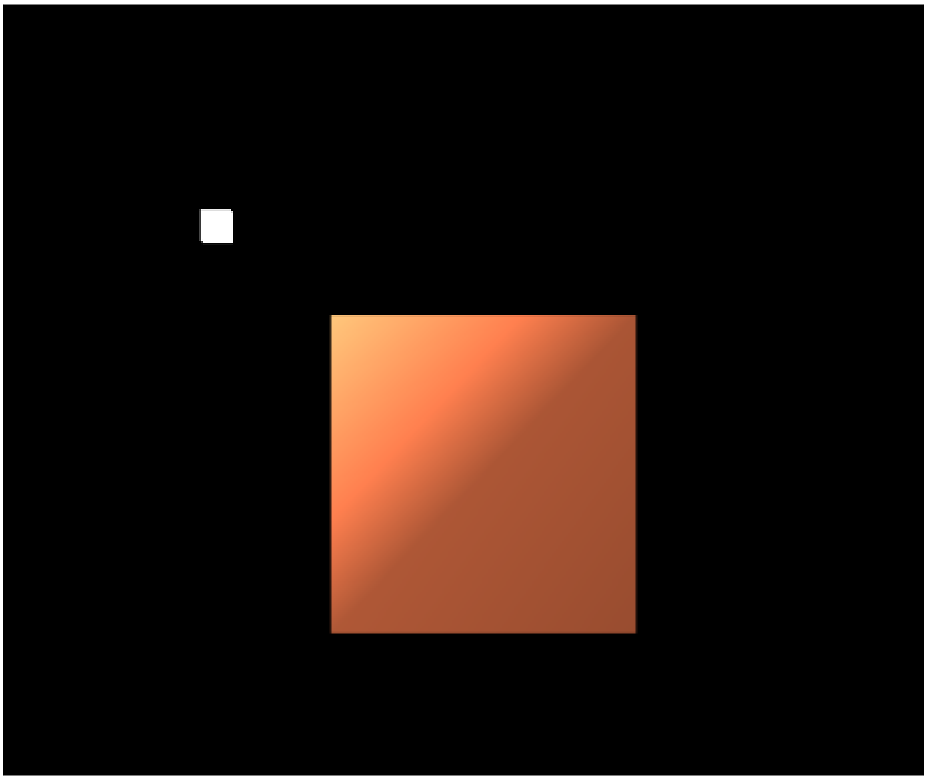
- Gouraud Shading 在物体表面的顶点用Phong光照模型计算出颜色值，其他部分的颜色值是顶点颜色值的插值。

  在OpenGL里面，实现Gouraud Shading就需要在顶点着色器中实现Phong光照模型，计算出顶点的颜色，再将数据传入片段着色器。片段着色器会通过插值产生其他像素的颜色。

```
// shader.vs 顶点着色器
out vec3 LightingColor;

uniform float ambientStrengthG;
uniform float diffuseStrengthG;
uniform float specularStrengthG;
uniform int nshinessG;

uniform vec3 lightPosG;
uniform vec3 viewPosG;
uniform vec3 lightColorG;

void main(){
    gl_Position = projection * view * model * vec4(position, 1.0f);

    vec3 Position = vec3(model * vec4(position, 1.0));
    vec3 Normal = mat3(transpose(inverse(model))) * normal;

    // ambient
    vec3 ambient = ambientStrengthG * lightColorG;

    // diffuse
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPosG - Position);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diffuseStrengthG * diff * lightColorG;

    // specular
    vec3 viewDir = normalize(viewPosG - Position);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), nshinessG);
    vec3 specular = specularStrengthG * spec * lightColorG;

    LightingColor = ambient + diffuse + specular;

}
```

```
out vec4 FragColor;

in vec3 LightingColor;

uniform vec3 objectColor;

void main(){
    else if(chooseFrag == 1){
        FragColor = vec4(LightingColor * objectColor, 1.0);
    }
}
```

效果



可以看到通过顶点插值产生的光照没有每个顶点都使用Phong光照模型计算出来的那么平滑、自然。

2. 使用GUI，使参数可调节，效果实时更改：

- GUI里可以切换两种Shading

```
// homework 6
if (choose_index == 6) {
    // 使光源在场景中来回移动
    lightPos.x = 1.0f + sin(glfwGetTime()) * 2.0f;
    lightPos.y = sin(glfwGetTime() / 2.0f) * 1.0f;
    // 选择Phong光照或者gouraud光照
    ImGui::TextWrapped("Please select a lighting model");
    const char* names2[] = { "Phong", "Gouraud" };
    if (ImGui::BeginPopup("my_select_popup2"))
```

```
    {
        ImGui::Text("Lighting model");
        ImGui::Separator();
        for (int i = 0; i < IM_ARRAYSIZE(names2); i++)
            if (ImGui::Selectable(names2[i])) {
                selected_shading = i;
            }
        ImGui::EndPopup();
    }
    if (ImGui::Button("Select lighting model.."))
        ImGui::OpenPopup("my_select_popup2");
    ImGui::SameLine();
    ImGui::TextUnformatted(selected_shading == -1 ? "<None>" :
names2[selected_shading]);
}

if (selected_shading == 0) { // Phong shading
    // ....
}
else if (selected_shading == 1) { // Gouraud shading
    // ...
}
```

在shader里面，我通过`chooseFrag`和`chooseVs`来判断使用哪种shading

```
// shader.vs
void main() {
    if(chooseVs == 0){
        gl_Position = projection * view * model * vec4(position,1.0f);
        FragPos = vec3(model * vec4(position, 1.0f));
        //Normal = normal;
        Normal = mat3(transpose(inverse(model))) * normal;
    }
    else{
        gl_Position = projection * view * model * vec4(position, 1.0f);

        vec3 Position = vec3(model * vec4(position, 1.0));
        vec3 Normal = mat3(transpose(inverse(model))) * normal;

        // ambient
        vec3 ambient = ambientStrengthG * lightColorG;

        // diffuse
        vec3 norm = normalize(Normal);
        vec3 lightDir = normalize(lightPosG - Position);
        float diff = max(dot(norm, lightDir), 0.0);
        vec3 diffuse = diffuseStrengthG * diff * lightColorG;

        // specular
        vec3 viewDir = normalize(viewPosG - Position);
```

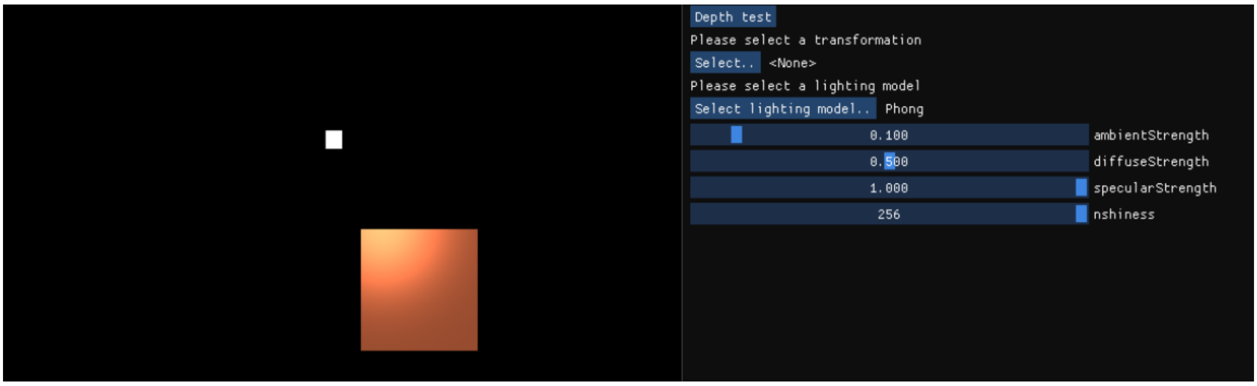
```
        vec3 reflectDir = reflect(-lightDir, norm);
        float spec = pow(max(dot(viewDir, reflectDir), 0.0), nshinessG);
        vec3 specular = specularStrengthG * spec * lightColorG;

        LightingColor = ambient + diffuse + specular;
    }

}
```

```
// shader.frag 片段着色器
void main(){
    if(chooseFrag == 0){
        // Ambient
        //float ambientStrength = 0.1f;
        vec3 ambient = ambientStrength * lightColor;

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

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

        vec3 result = (ambient + diffuse + specular) * objectColor;
        FragColor = vec4(result, 1.0f);
    }
    else if(chooseFrag == 1){
        FragColor = vec4(LightingColor * objectColor, 1.0);
    }

}
```

- 使用如进度条这样的控件，使ambient因子、diffuse因子、specular因子、反光度等参数可调节，光照效 果实时更改

```
// 进度条 更改因子
ImGui::SliderFloat("ambientStrength", &ambientStrength, 0.0f, 1.0f);
ImGui::SliderFloat("diffuseStrength", &diffuseStrength, 0.0f, 1.0f);
ImGui::SliderFloat("specularStrength", &specularStrength, 0.0f, 1.0f);
ImGui::SliderInt("nshiness", &nshiness, 0, 256);
```

效果



Bonus

当前光源为静止状态，尝试使光源在场景中来回移动，光照效果实时更改。

```
// 使光源在场景中来回移动
lightPos.x = 1.0f + sin(glfwGetTime()) * 2.0f;
lightPos.y = sin(glfwGetTime() / 2.0f) * 1.0f;
```

PS

把法向量从物体空间坐标转换到世界空间坐标

法向量不可以直接通过乘上一个Model矩阵来转换到世界空间坐标系，而是需要乘上一个正规矩阵（Normal matrix）来进行变换。

推导如下

在坐标变换之前，假设T是位于图形表面的向量，N是法向量，于是有

$N \cdot T = 0$

$N^T T = 0$

假设T'为T经过坐标变换后得到的向量，N'为N经过坐标变换后得到的向量，有

$T' = MT$

$N' = GN$

要使N'依旧为法向量，则要有

$N' \cdot T' = 0$

$(GN) \cdot (MT) = 0$

$(GN)^T (MT) = 0$

$N^T G^T MT = 0$

而我们已知$N^T T = 0$

所以只需要$G^T M = I$即有$N' \cdot T' = 0$

易得$G = (M^{-1})^T$