基于Phong模型的光照渲染器

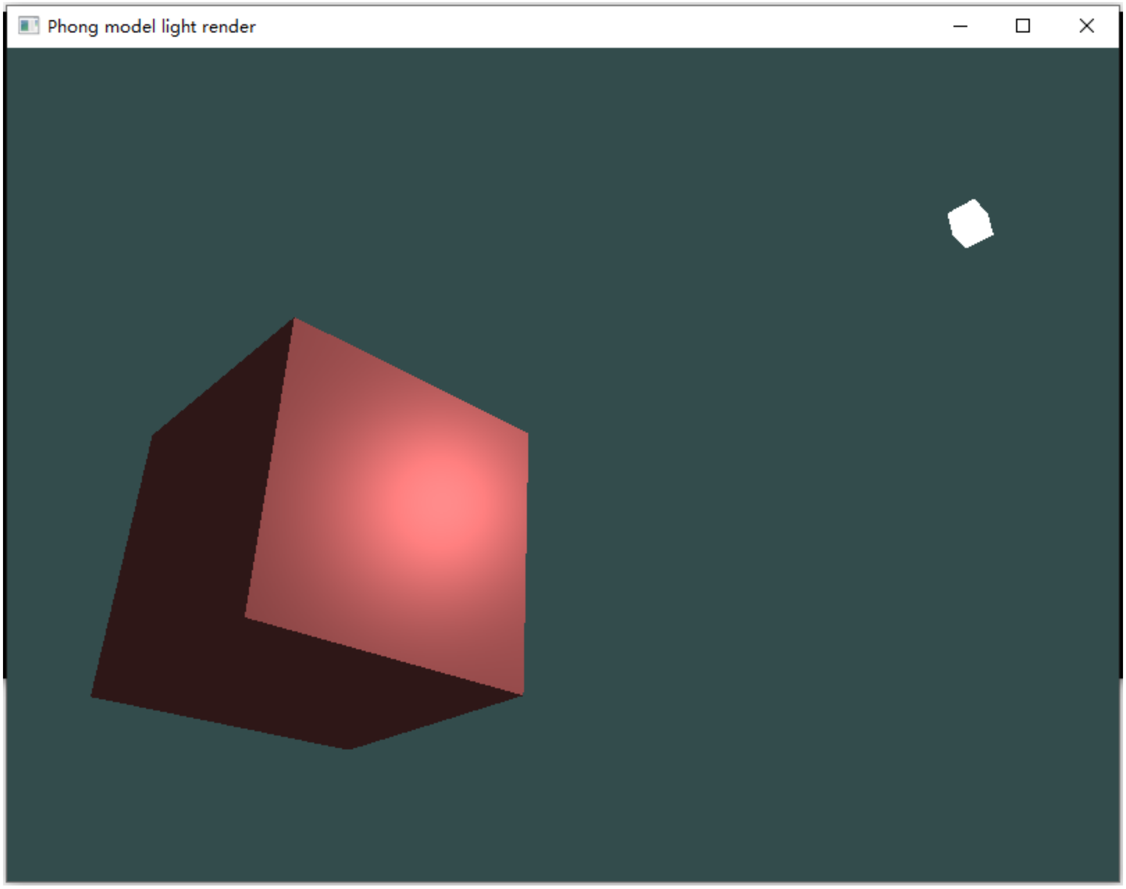
1711368 齐冲

实现功能

- 实现了基于Phong模型的环境光、漫反射光、镜面反射光的渲染；
- 实现了摄像机功能，可以六方向移动与全方向转动和缩放（WSAD+空格+左CTRL+鼠标与滚轮）。

演示截图

- 一立方体，一光源，可以看出三种光照效果。



讲解与部分代码

- 代码中创建了四种着色器：物体的顶点、片段着色器与光源立方体的顶点、片段着色器。其中Phong模型的实现主要依靠物体的两个着色器，先由顶点着色器计算出位置与方向信息，再把结果输入片段着色器计算三种光线，最后结果相加乘以物体颜色即为显示在屏幕上的颜色。

```
//物体顶点着色器
#version 330 core
layout (location = 0) in vec3 aPos;//三分量的位置向量，定属性为0
layout (location = 1) in vec3 aNormal;//三分量的法向量，定属性为1

out vec3 FragPos;
out vec3 Normal;

```

```
uniform mat4 model;//世界坐标变换矩阵
uniform mat4 view;//视角坐标变换矩阵
uniform mat4 projection;//投影坐标变换矩阵

void main()
{
    FragPos = vec3(model * vec4(aPos, 1.0));//世界坐标中的片段位置
    Normal = mat3(transpose(inverse(model))) * aNormal;//使用逆矩阵与转置矩阵来变换法向量，使得在视角移动与缩放时法向量依然保持正确

    gl_Position = projection * view * vec4(FragPos, 1.0);//最终显示在屏幕上的位置：将它们相乘
}
```

```
//物体片段着色器
#version 330 core
out vec4 FragColor;

in vec3 Normal;//顶点着色器传进来的法向量
in vec3 FragPos;//顶点着色器传进来的片段位置（世界坐标）

uniform vec3 lightPos; //光源位置，固定
uniform vec3 viewPos; //视点位置
uniform vec3 lightColor;//光的颜色，默认为白光
uniform vec3 objectColor;//物体颜色

void main()
{
    //环境光，计算方式：环境光系数*光的颜色
    float ambientStrength = 0.3;
    vec3 ambient = ambientStrength * lightColor;

    //漫反射光，先计算光线的方向向量，再点乘法向量得出漫反射分量，再乘以光线颜色
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);//光线方向向量
    float diff = max(dot(norm, lightDir), 0.0);//漫反射分量
    vec3 diffuse = diff * lightColor;

    //镜面反射光，先计算视线方向与反射方向，再点乘计算镜面分量，再乘以镜面强度与光线颜色
    float specularStrength = 0.8;//镜面强度
    vec3 viewDir = normalize(viewPos - FragPos);//视线方向向量
    vec3 reflectDir = reflect(-lightDir, norm);//反射光线方向向量
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);//计算镜面分量，32为“反光度”
    vec3 specular = specularStrength * spec * lightColor;

    vec3 result = (ambient + diffuse + specular) * objectColor;//最后三种结果相加再乘以物体颜色
    FragColor = vec4(result, 1.0);
}
```

- 顶点信息的设置，由于物体与光源均为固定，所以使用一组固定的顶点属性来描述。其中每一个顶点有六个属性，前三个为坐标，后三个为法向量。每一组有六个顶点，因为立方体的每一个面由两个三角形（共六个顶点）拼成。共有六组顶点，因为立方体有六个面。

```
//顶点属性，前三为坐标，后三为法向量，立方体每一面由两个三角形拼成
    float vertices[] = {
        -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        -0.5f,  0.5f, -0.5f,  0.0f,  0.0f, -1.0f,
        -0.5f, -0.5f, -0.5f,  0.0f,  0.0f, -1.0f,

        -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
        0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
        0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
        0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
        -0.5f,  0.5f,  0.5f,  0.0f,  0.0f,  1.0f,
        -0.5f, -0.5f,  0.5f,  0.0f,  0.0f,  1.0f,

        -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f,  0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f, -0.5f, -0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f, -0.5f,  0.5f, -1.0f,  0.0f,  0.0f,
        -0.5f,  0.5f,  0.5f, -1.0f,  0.0f,  0.0f,

        0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
        0.5f,  0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
        0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
        0.5f, -0.5f, -0.5f,  1.0f,  0.0f,  0.0f,
        0.5f, -0.5f,  0.5f,  1.0f,  0.0f,  0.0f,
        0.5f,  0.5f,  0.5f,  1.0f,  0.0f,  0.0f,

        -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
        0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,
        0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
        0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
        -0.5f, -0.5f,  0.5f,  0.0f, -1.0f,  0.0f,
        -0.5f, -0.5f, -0.5f,  0.0f, -1.0f,  0.0f,

        -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
        0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f,
        0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
        0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
        -0.5f,  0.5f,  0.5f,  0.0f,  1.0f,  0.0f,
        -0.5f,  0.5f, -0.5f,  0.0f,  1.0f,  0.0f
    };
```

- 循环渲染

```
//渲染
    while (!glfwWindowShouldClose(window))//当未按下esc时，循环渲染
    {
        //计算deltatime，用来计算摄像机位移
        float currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        //检测键盘输入：esc,上，下，左，右
        processInput(window);

        //设置背景颜色
        glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        //use着色器，设置颜色与位置
        lightingShader.use();
        lightingShader.setVec3("objectColor", 0.6f, 0.3f, 0.3f);
        lightingShader.setVec3("lightColor", 1.0f, 1.0f, 1.0f);
        lightingShader.setVec3("lightPos", lightPos);
        lightingShader.setVec3("viewPos", camera.Position);

        //视角空间与投影矩阵
        glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);
        glm::mat4 view = camera.GetViewMatrix();
        lightingShader.setMat4("projection", projection);
        lightingShader.setMat4("view", view);

        //世界空间
        glm::mat4 model = glm::mat4(1.0f);
        lightingShader.setMat4("model", model);

        //渲染出物体立方体
        glBindVertexArray(cubeVAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);

        //渲染出光源立方体
        lampShader.use();
        lampShader.setMat4("projection", projection);
        lampShader.setMat4("view", view);
        model = glm::mat4(1.0f);
        model = glm::translate(model, lightPos);
        model = glm::scale(model, glm::vec3(0.1f));
        lampShader.setMat4("model", model);

        glBindVertexArray(lightVAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);


        //交换缓存，检测IO
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
```

- 摄像机移动、视角转动与缩放

```
//键盘输入
    void ProcessKeyboard(Camera_Movement direction, float deltaTime)
    {
        float velocity = MovementSpeed * deltaTime;//计算位移
        if (direction == FORWARD)
            Position += Front * velocity;
        if (direction == BACKWARD)
            Position -= Front * velocity;
        if (direction == LEFT)
```

```
            Position -= Right * velocity;
        if (direction == RIGHT)
            Position += Right * velocity;
        if (direction == UPWARD)
            Position += Up * velocity;
        if (direction == DOWNWARD)
            Position -= Up * velocity;
    }

    //鼠标输入
    void ProcessMouseMovement(float xoffset, float yoffset, GLboolean constrainPitch = true)
    {
        xoffset *= MouseSensitivity;
        yoffset *= MouseSensitivity;

        Yaw += xoffset;//偏航角
        Pitch += yoffset;//俯仰角

        if (constrainPitch)//修正俯仰角
        {
            if (Pitch > 89.0f)
                Pitch = 89.0f;
            if (Pitch < -89.0f)
                Pitch = -89.0f;
        }

        updateCameraVectors();
    }

    //滚轮缩放
    void ProcessMouseScroll(float yoffset)
    {
        if (Zoom >= 1.0f && Zoom <= 45.0f)
            Zoom -= yoffset;
        if (Zoom <= 1.0f)
            Zoom = 1.0f;
        if (Zoom >= 45.0f)
            Zoom = 45.0f;
    }
```