基于Phong模型的光照渲染器

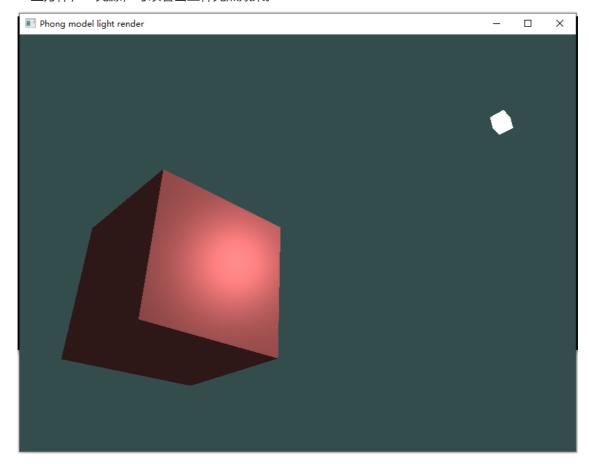
1711368 齐冲

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。

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

```
9 uniform mat4 model;//世界坐标变换矩阵
10
   uniform mat4 view;//视角坐标变换矩阵
   uniform mat4 projection;//投影坐标变换矩阵
11
12
13 | void main()
14 {
15
       FragPos = vec3(model * vec4(aPos, 1.0));//世界坐标中的片段位置
16
       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
17
       gl_Position = projection * view * vec4(FragPos, 1.0);//最终显示在屏幕
18
   上的位置:将它们相乘
19
   }
```

```
1 //物体片段着色器
2
   #version 330 core
  out vec4 FragColor;
 5
   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)
6
7
  uniform vec3 lightPos; //光源位置,固定
8
9
   uniform vec3 viewPos; //视点位置
10 uniform vec3 lightColor;//光的颜色,默认为白光
11 uniform vec3 objectColor;//物体颜色
12
13 void main()
14
15
       //环境光, 计算方式: 环境光系数*光的颜色
       float ambientStrength = 0.3;
16
17
       vec3 ambient = ambientStrength * lightColor;
18
19
       //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
20
       vec3 norm = normalize(Normal);
21
       vec3 lightDir = normalize(lightPos - FragPos);//光线方向向量
       float diff = max(dot(norm, lightDir), 0.0);//漫反射分量
22
23
       vec3 diffuse = diff * lightColor;
24
25
       //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
26
       float specularStrength = 0.8;//镜面强度
27
       vec3 viewDir = normalize(viewPos - FragPos);//视线方向向量
       vec3 reflectDir = reflect(-lightDir, norm);//反射光线方向向量
28
29
       float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);//计算镜面分
    量,32为"反光度"
30
       vec3 specular = specularStrength * spec * lightColor;
31
32
       vec3 result = (ambient + diffuse + specular) * objectColor;//最后三
   种结果相加再乘以物体颜色
33
       FragColor = vec4(result, 1.0);
34
   }
```

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
2
        float vertices[] = {
           -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f,
3
           0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f,
4
           0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f,
5
6
           0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f,
7
           -0.5f, 0.5f, -0.5f, 0.0f, 0.0f, -1.0f,
           -0.5f, -0.5f, -0.5f, 0.0f, 0.0f, -1.0f,
8
9
           -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,
10
           0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,
11
12
           0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,
           0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,
13
           -0.5f, 0.5f, 0.5f, 0.0f, 0.0f, 1.0f,
14
           -0.5f, -0.5f, 0.5f, 0.0f, 0.0f, 1.0f,
15
16
17
           -0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f,
           -0.5f, 0.5f, -0.5f, -1.0f, 0.0f,
18
                                            0.0f.
           -0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f,
19
           -0.5f, -0.5f, -0.5f, -1.0f, 0.0f, 0.0f,
20
           -0.5f, -0.5f, 0.5f, -1.0f, 0.0f, 0.0f,
21
22
           -0.5f, 0.5f, 0.5f, -1.0f, 0.0f, 0.0f,
23
           0.5f, 0.5f, 0.5f, 1.0f, 0.0f, 0.0f,
24
           0.5f, 0.5f, -0.5f, 1.0f, 0.0f,
25
           0.5f, -0.5f, -0.5f, 1.0f, 0.0f, 0.0f,
26
27
           0.5f, -0.5f, -0.5f, 1.0f, 0.0f,
                                            0.0f,
           0.5f, -0.5f, 0.5f, 1.0f, 0.0f,
                                            0.0f,
28
           0.5f, 0.5f, 0.5f, 1.0f, 0.0f,
29
30
           -0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f,
31
           0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f,
32
           0.5f, -0.5f, 0.5f, 0.0f, -1.0f,
33
                                            0.0f.
           0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f,
34
           -0.5f, -0.5f, 0.5f, 0.0f, -1.0f, 0.0f,
35
           -0.5f, -0.5f, -0.5f, 0.0f, -1.0f, 0.0f,
36
37
           -0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
38
           0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f,
39
40
           0.5f, 0.5f, 0.5f, 0.0f, 1.0f,
                                            0.0f,
           0.5f, 0.5f, 0.5f, 0.0f, 1.0f,
41
           -0.5f, 0.5f, 0.5f, 0.0f, 1.0f, 0.0f,
42
           -0.5f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f
43
44
       };
```

循环渲染

```
//渲染
2
       while (!glfwWindowShouldClose(window))//当未按下esc时,循环渲染
3
4
           //计算deltatime,用来计算摄像机位移
5
           float currentFrame = glfwGetTime();
6
           deltaTime = currentFrame - lastFrame;
7
           lastFrame = currentFrame;
8
9
           //检测键盘输入: esc,上,下,左,右
10
           processInput(window);
```

```
11
12
            //设置背景颜色
            glclearColor(0.2f, 0.3f, 0.3f, 1.0f);
13
14
            glclear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
15
16
            //use着色器,设置颜色与位置
            lightingShader.use();
17
            lightingShader.setVec3("objectColor", 0.6f, 0.3f, 0.3f);
18
            lightingShader.setVec3("lightColor", 1.0f, 1.0f, 1.0f);
19
20
            lightingShader.setVec3("lightPos", lightPos);
            lightingShader.setVec3("viewPos", camera.Position);
21
22
23
            //视角空间与投影矩阵
24
            glm::mat4 projection =
    glm::perspective(glm::radians(camera.Zoom), (float)SCR_WIDTH /
    (float)SCR_HEIGHT, 0.1f, 100.0f);
25
            glm::mat4 view = camera.GetViewMatrix();
26
            lightingShader.setMat4("projection", projection);
            lightingShader.setMat4("view", view);
27
28
29
            //世界空间
30
            glm::mat4 model = glm::mat4(1.0f);
31
            lightingShader.setMat4("model", model);
32
33
            //渲染出物体立方体
            glBindVertexArray(cubeVAO);
34
            glDrawArrays(GL_TRIANGLES, 0, 36);
35
36
            //渲染出光源立方体
37
38
            lampShader.use();
            lampShader.setMat4("projection", projection);
39
40
            lampShader.setMat4("view", view);
            model = glm::mat4(1.0f);
41
42
            model = glm::translate(model, lightPos);
43
            model = glm::scale(model, glm::vec3(0.1f));
44
            lampShader.setMat4("model", model);
45
46
            glBindVertexArray(lightVAO);
            glDrawArrays(GL_TRIANGLES, 0, 36);
47
48
49
50
            //交换缓存,检测IO
51
            glfwSwapBuffers(window);
52
            glfwPollEvents();
53
        }
```

• 摄像机移动、视角转动与缩放

```
1
   //键盘输入
2
       void ProcessKeyboard(Camera_Movement direction, float deltaTime)
3
       {
           float velocity = MovementSpeed * deltaTime;//计算位移
4
           if (direction == FORWARD)
               Position += Front * velocity;
6
7
           if (direction == BACKWARD)
8
               Position -= Front * velocity;
9
           if (direction == LEFT)
```

```
10
                Position -= Right * velocity;
11
            if (direction == RIGHT)
12
                Position += Right * velocity;
            if (direction == UPWARD)
13
                Position += Up * velocity;
14
15
            if (direction == DOWNWARD)
16
                Position -= Up * velocity;
17
        }
18
19
        //鼠标输入
        void ProcessMouseMovement(float xoffset, float yoffset, GLboolean
20
    constrainPitch = true)
21
        {
22
            xoffset *= MouseSensitivity;
            yoffset *= MouseSensitivity;
23
24
25
            Yaw += xoffset;//偏航角
26
            Pitch += yoffset;//俯仰角
27
            if (constrainPitch)//修正俯仰角
28
29
30
                if (Pitch > 89.0f)
31
                    Pitch = 89.0f;
32
                if (Pitch < -89.0f)
                    Pitch = -89.0f;
33
34
            }
35
36
            updateCameraVectors();
37
        }
38
        //滚轮缩放
39
40
        void ProcessMouseScroll(float yoffset)
41
        {
            if (Zoom >= 1.0f && Zoom <= 45.0f)
42
43
                Zoom -= yoffset;
            if (Zoom <= 1.0f)
44
45
                Zoom = 1.0f;
            if (Zoom >= 45.0f)
46
47
                Zoom = 45.0f;
48
        }
```