颜色

- 物体的颜色, 是由物体反射光源后得到的

```
glm::vec3 lightColor(1.0f, 1.0f, 1.0f);     //  光源颜色为白色
glm::vec3 toyColor(1.0f, 0.5f, 0.31f);      //  物体颜色
glm::vec3 result = lightColor * toyColor;   //  反射后的颜色
```

创建光照场景

创建物体

```
1. 创建物体VAO
    //  cubeVAO
    unsigned int cubeVAO;
    glGenVertexArrays(1, &cubeVAO);
    glBindVertexArray(cubeVAO);

    //  VBO
    unsigned int VBO;
    glGenBuffers(1, &VBO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    //  设置位置属性
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);

    //  法向量属性
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 *
sizeof(float)));
    glEnableVertexAttribArray(1);



    when(...) {
        2. 激活物体着色器并传递参数
        Shader cubeShader("cubeShader.vs", "cubeShader.fs");
```

```
        cubeShader.use();
        cubeShader.setVec3(...)
        cubeShader.setMat4("model", model);
        cubeShader.setMat4("projection", projection);
        cubeShader("view", view);


        3. 绘制物体
        glBindVertexArray(cubeVAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);
    }
```

创建光源

```
1. 创建光源VAO
    //  lightVAO
    unsigned int lightVAO;
    glGenVertexArrays(1, &lightVAO);
    glBindVertexArray(lightVAO);

    //  VBO
    glBindBuffer(GL_ARRAY_BUFFER, VBO);

    //  设置位置属性
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
    glEnableVertexAttribArray(0);


    when(...) {
        2. 激活光源着色器并传递参数
        Shader lightShader("lightShader.vs", "lightShader.fs");
        lightShader.use();
        lightShader.setVec3(...)
        lightShader.setMat4("model", model);
        lightShader.setMat4("projection", projection);
        lightShader("view", view);


        3. 绘制光源
        glBindVertexArray(lightVAO);
        glDrawArrays(GL_TRIANGLES, 0, 36);
    }
```

冯氏光照模型

- Phong Shading : 在片段着色器实现冯氏光照模型
- Gouraud Shading: 在顶点着色器实现的冯氏光照模型

Phong Shading

环境光照Ambient

- 即使在黑暗的情况下，世界上通常也仍然有一些光亮（月亮、远处的光），所以物体几乎永远不会是完全黑暗的。为了模拟这个，我们会使用一个环境光照常量，它永远会给物体一些颜色

```
// 用光的颜色乘以一个很小的常量环境因子，再乘以物体的颜色

    // 片段着色器
    float ambientStrength = 0.1;
    vec3 ambient = ambientStrength * lightColor;

    vec3 result = ambient * objectColor;
    FragColor = vec4(result, 1.0);
```

漫反射光照Diffuse

- 模拟光源对物体的方向性影响。物体的某一部分越是正对着光源，它就会越亮。

```
// 用光源发出的光线与物体表面法向量形成的夹角，计算物体表面的漫反射光照

    // 顶点着色器
    Normal = mat3(transpose(inverse(model))) * aNormal;


    // 片段着色器
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - FragPos);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diffuseStrength * diff * lightColor;
```

镜面光照Specular

- 模拟有光泽物体上面出现的亮点。镜面光照的颜色相比于物体的颜色会更倾向于光的颜色

```
//  需要设置一个观察者的位置变量，然后进行镜面光照计算

    //  片段着色器
    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), ShininessStrength);
    vec3 specular = specularStrength * spec * lightColor;
```

总结

```
//  顶点着色器
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 FragPos;
out vec3 Normal;

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

void main() {
    FragPos = vec3(model * vec4(aPos, 1.0));
    Normal = mat3(transpose(inverse(model))) * aNormal;

    gl_Position = projection * view * vec4(FragPos, 1.0);
}




//  片段着色器
#version 330 core

out vec4 FragColor;

in vec3 Normal;
in vec3 FragPos;

uniform float ambientStrength;
uniform float diffuseStrength;
uniform float specularStrength;
uniform int ShininessStrength;
```

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

void main() {
    //  环境光
    vec3 ambient = ambientStrength * lightColor;

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

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

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

Gouraud Shading

- 将Phong Shading在片段着色器中的计算移动到顶点着色器中.

- 相对Phong Shading, 开销较小; 但是由于片段的颜色是由插值决定的, 因此拟真度较低

总结

```
//  顶点着色器
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 LightingColor;

uniform float ambientStrength;
uniform float diffuseStrength;
uniform float specularStrength;
uniform int ShininessStrength;
```

```
uniform vec3 lightPos;
uniform vec3 viewPos;
uniform vec3 lightColor;

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

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

    // gouraud shading
    vec3 Position = vec3(model * vec4(aPos, 1.0));
    vec3 Normal = mat3(transpose(inverse(model))) * aNormal;

    // 环境光
    vec3 ambient = ambientStrength * lightColor;

    // 漫反射
    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(lightPos - Position);
    float diff = max(dot(norm, lightDir), 0.0);
    vec3 diffuse = diffuseStrength * diff * lightColor;

    // 镜面光
    vec3 viewDir = normalize(viewPos - Position);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0), ShininessStrength);
    vec3 specular = specularStrength * spec * lightColor;

    LightingColor = ambient + diffuse + specular;
}



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

in vec3 LightingColor;

uniform vec3 objectColor;

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

光源来回移动

```
1. 定义全局光源坐标

        glm::vec3 lightPos(20.0f, 8.0f, 20.0f);

2. 光源随时间移动

        when(...) {
        if (enable_translate) {
                        lightPos.x = sin(glfwGetTime()) * 8.0f;
                        lightPos.y = sin(glfwGetTime()) * 8.0f;
                }
        }
```