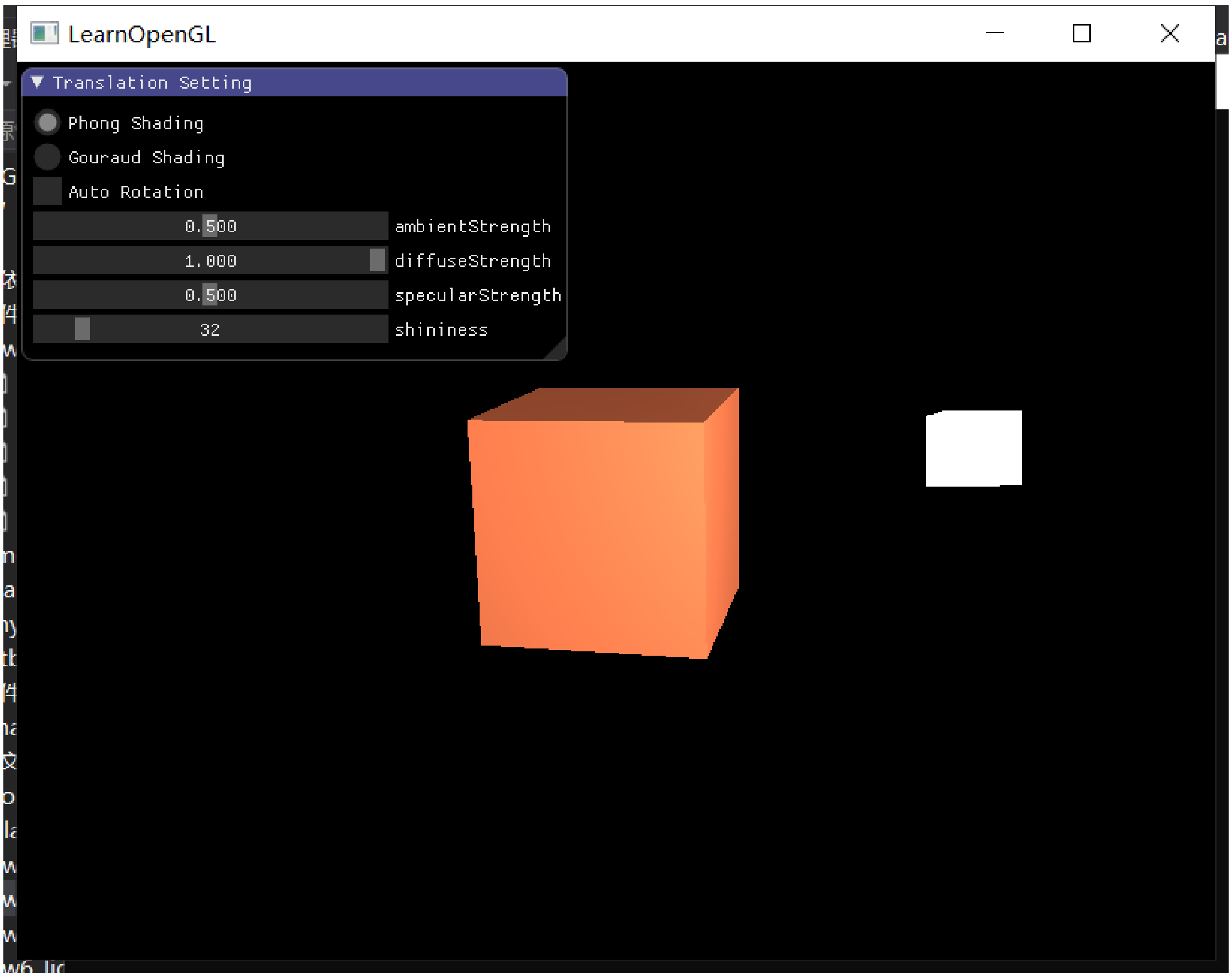
计算机图形学 作业六

一、实验结果

- Basic：实现了 Phong Shading 和 Gouraud Shading 两种模型，设置了合理的视点、光源位置、光源颜色、物体颜色使得光照效果明显，并添加了 GUI 使得可以切换两种 Shading 模型、控制光源静止或移动、调整 ambient 因子、diffffuse 因子、specular 因子、反光度等参数



```
0.500    ambientStrength
1.000    diffuseStrength
0.500    specularStrength
32       shininess
```

- Bonus：我将光源设置成绕Z轴旋转，并实现了光照效果实时改变，具体效果见附件 Demo.gif

二、代码实现

代码上与上次作业的思路一致，主要的差异在于着色器，下面是各个着色器的定义：

- 光源的着色器：

```
// 片段着色器
```

```
#version 330 core
out vec4 FragColor;

void main()
{
    FragColor = vec4(1.0);
}

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

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

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

- Phong Shading 模型的着色器:

```
// 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 shininess;

uniform vec3 lightPos;
uniform vec3 viewPos;

uniform vec3 objectColor;
uniform vec3 lightColor;

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

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

    // specular
    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) * objectColor;
    FragColor = vec4(result, 1.0);
}

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

out vec3 Normal;
out vec3 FragPos;

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 * model * vec4(aPos, 1.0);
}
```

- Gouraud Shading 模型的着色器：

```
// 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);
}

// 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 shininess;
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);

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

    // ambient
    vec3 ambient = ambientStrength * lightColor;

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

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

    LightingColor = ambient + diffuse + specular;
}
```

对比两种 Shading 的着色器可以发现，Phong lighting model 在顶点着色器中实现就称为 Gouraud Shading，在片段着色器中实现就称为 Phong Shading，两者的区别在于，因为 Gouraud Shading 在顶点着色器中实现，相比在片段着色器中实现来说，顶点要少得多，开销较大的光照计算频率会低得多，所以计算效率会高得多，但是因为顶点着色器中计算出来的最终颜色值只是顶点的颜色，片段的颜色值是通过顶点之间插值得到的，得到的效果看起来非常不真实，除非使用了大量的顶点，即 Phong Shading 能产生更平滑的光照效果（在这次实验中因为我整个立方体都用的是相同的颜色所以看不出不平滑的效果，也没有出现马赫带效应）
