计算机图形学作业6

Basic

1. 实现Phong光照模型

首先在场景中放置一个立方体用于观察光照效果，每个面都应用木箱纹理并且各个面的颜色不同。由于立方体的法向量可以直接计算得到，在初始化数据时设置各个面的法向量垂直向外并加载到GPU中。为了模拟真实光照场景，再增加一个白色的立方体作为光源，放置在木箱的左前方。

由于木箱、光源以及不同的shading需要不同的shader，先将这部分封装成`shader`类，然后传入不同的着色器代码就可以实现不同的shader了。声明`Shader`类如下

```
class Shader {

public:
  Shader(const char* vertexShaderSource, const char* fragmentShaderSource);
  ~Shader();

  void use();
  void setInt(const char* name, int value);
  void setFloat(const char* name, float value);
  void setVec3(const char* name, glm::vec3 value);
  void setMat4(const char* name, glm::mat4 value);

private:
  int shaderProgram;

};
```

其中构造函数实现了对顶点着色器和片段着色器的编译和链接，`use`函数为调用着色器程序，其它`set`函数用于传递`uniform`变量。

光源的着色器比较简单，只需要变换坐标并将颜色设置为白色即可。顶点着色器如下

```
#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);
}
```

片段着色器如下

```
#version 330 core
out vec4 FragColor;
void main() {
  FragColor = vec4(1.0);
}
```

接下来实现Phong光照模型。Phong光照模型就是将物体三个通道的颜色分别乘上对应的光照强度I得到最终的像素点颜色，而光照强度由光源和反射方式决定，具体分为环境光照、漫反射光照和镜面光照。环境光照指的是光在不同物体表面上不断反射间接照亮整个空间的物体，为了模拟这种效果，我们取常量K_a作为环境因子，用IK_a表示环境光照I_a。漫反射光照指的是光照射在粗糙表面上向所有方向反射均匀的光照，漫反射光照I_d与光线的入射角度有关，可以表示为$IK_d(n \cdot l)$，其中K_d为漫反射因子，n为物体表面的单位法向量，l为光线的入射单位向量。镜面反射指的是光照射在光滑表面上形成的镜面高光，不但与光线的入射角度有关，还与观察角度有关，镜面光照可以表示为$IK_d(v \cdot r)^n$，其中K_d为镜面因子，v为视线方向的单位向量，r为光线的反射单位向量，n为高光的反光度。Phong光照模型将计算得到的三种光照相加，再乘上光源的光照强度，得到照射在物体上的光照强度，最后乘上物体的颜色得到像素点颜色。需要注意的是，由于Phong光照模型涉及内积运算，计算镜面光照有可能得到负数的结果，这是由于正在计算光照的面在背光面，因此需要对结果与0取较大值。

实现Phong光照模型有两种方式，分别是Phong shading和Gouraud shading。Phong shading就是对物体每个顶点的法向量进行插值，再对每一个点应用光照模型，而Gouraud shading则是对物体每个顶点应用光照模型，再对顶点的颜色进行插值。具体实现就是Phong shading在顶点着色器中传递法向量，在片段着色器中计算光照，而Gouraud shading在顶点着色器中计算光照。Phong shading的顶点着色器如下

```
#version 330 core
layout(location = 0) in vec3 aPos;
layout(location = 1) in vec2 aTexCoord;
layout(location = 2) in vec3 aColor;
layout(location = 3) in vec3 aNormal;
out vec2 TexCoord;
out vec3 objectColor;
out vec3 Normal;
out vec3 FragPos;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
void main() {
  gl_Position = projection * view * model * vec4(aPos, 1.0);
  TexCoord = aTexCoord;
  objectColor = aColor;
  Normal = mat3(transpose(inverse(model))) * aNormal;
  FragPos = vec3(model * vec4(aPos, 1.0));
}
```

片段着色器如下

```
#version 330 core
out vec4 FragColor;
in vec2 TexCoord;
in vec3 objectColor;
in vec3 Normal;
in vec3 FragPos;
uniform sampler2D texture;
```

```
uniform vec3 lightColor;
uniform vec3 lightPos;
uniform float ambientStrength;
uniform float diffuseStrength;
uniform vec3 viewPos;
uniform int shininess;
uniform float specularStrength;
void main() {
  vec3 ambient = ambientStrength * lightColor;
  vec3 norm = normalize(Normal);
  vec3 lightDir = normalize(lightPos - FragPos);
  float diff = max(dot(norm, lightDir), 0.0);
  vec3 diffuse = diffuseStrength * diff * lightColor;
  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;
  FragColor = texture(texture, TexCoord) * vec4((ambient + diffuse + specular) *
objectColor, 1.0f);
}
```

Gouraud shading的顶点着色器如下

```
#version 330 core
layout(location = 0) in vec3 aPos;
layout(location = 1) in vec2 aTexCoord;
layout(location = 2) in vec3 aColor;
layout(location = 3) in vec3 aNormal;
out vec2 TexCoord;
out vec3 objectColor;
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;
uniform vec3 lightColor;
uniform vec3 lightPos;
uniform float ambientStrength;
uniform float diffuseStrength;
uniform vec3 viewPos;
uniform int shininess;
uniform float specularStrength;
void main() {
  gl_Position = projection * view * model * vec4(aPos, 1.0);
  TexCoord = aTexCoord;
  vec3 ambient = ambientStrength * lightColor;
  vec3 norm = normalize(aNormal);
  vec3 FragPos = vec3(model * vec4(aPos, 1.0));
  vec3 lightDir = normalize(lightPos - FragPos);
  float diff = max(dot(norm, lightDir), 0.0);
  vec3 diffuse = diffuseStrength * diff * lightColor;
  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;
```

```
    objectColor = (ambient + diffuse + specular) * aColor;
}\0";
```

对比Phong shading和Gouraud shading可以发现，Gouraud shading几乎不显示镜面光照的高光部分，这是因为Gouraud shading只在顶点应用了光照模型，而高光主要出现在图形内部，所以插值得到的结果中没有高光。而Phong shading在每一个点都应用了光照模型，所以可以显示完整的镜面光照效果。

为了更好地观察光照效果，加入视角移动的功能，可以移动摄像机到不同角度来观察光照效果。在ImGui菜单栏View中，选中Move进行移动，通过键盘按键【w】【s】【a】【d】或方向键来进行前后左右移动，通过鼠标来控制视角移动，使用滚轮可以缩放视角，选中Freeze停止移动，也可以用键盘按键【5】【6】来切换状态。在ImGui菜单栏Shading中，选中Phong为Phong shading，选中Gouraud为Gouraud shading，在进行视角移动时可以用键盘按键【1】【2】进行切换。Shader类见`Shader.h`和`Shader.cpp`，实现Phong shading和Gouraud shading的shader见`P1.cpp`，实现光照模型的完整代码见`P2.cpp`，观察光照效果的视频见`P1.mp4`。

2. 调节参数

将环境因子、漫反射因子、镜面因子和反光度都用uniform变量表示，在渲染循环中更新参数，再传递到shader中，实时修改光照效果。其中环境因子、漫反射因子和镜面因子的取值都在0到1之间，反光度按2的指数从1到1024递增。

经过调节发现，环境因子的大小直接决定了物体背面的亮度，因为光源无法直射物体背面；漫反射因子决定了物体大部分被照射区域的亮度，因为光源离物体较远，到各顶点的入射角度相差不大；镜面因子决定了物体高光部分的亮度，高光区域会随着视角移动而转移；反光度决定了物体高光部分的大小，反光度越高，物体高光部分越小。

在ImGui菜单栏中，使用浮点数滑动输入框调节各参数的大小，实现参数调节的代码见`P2.cpp`，调节参数实时更新光照效果的视频见`P1.mp4`。

Bonus

1. 移动光源

将光源随时间绕Y轴旋转，即使视角不变，光照效果也会发生改变，因为漫反射光照和镜面光照都与入射角有关。在ImGui菜单栏的Light中，选中Move移动光源，选中Freeze停止移动，在进行视角移动时可以通过键盘按键【3】【4】切换状态。实现光源移动的函数见`B1.cpp`，移动光源的视频见`B1.mp4`。