浙江大学

本科实验报告

课程名称：计算机图形学

姓　　名：王晨宇

学　　院：计算机科学与技术学院

系：

专　　业：计算机科学与技术

学　　号：3180104919

指导教师：唐敏

目录

浙江大学实验报告

课程名称：	计算机图形学	实验类型：	综合		
实验项目名称：	OpenGL 实例化				
学生姓名：	王晨宇	专业：	计算机科学与技术	学号：	3180104919
同组学生姓名：	None	指导老师：	唐敏		
实验地点：	紫金港机房	实验时间：	2021-05-23		

一、 实验目的和要求

在 OpenGL 观察实验的基础上，通过实现实验内容，掌握 OpenGL 中消隐和光照的设置，并验证课程中消隐和光照的内容。

二、 实验环境

- Visual Studio C++ 2019
- GLFW、GLAD、GLM

三、 实验内容和原理

3.1 材质

对于不同的物体会对光产生不同的反应，根据环境光反射颜色、漫反射光反射颜色、镜面光反射颜色以及反射率来定义一个物体表面的材质：

albedo 环境光反射系数

ka 反射系数

kd 漫反射系数

ks 镜面反射系数

它们格子在对应的光照模型中起到作用。于是我们就可以这么来定义一个物体的材质：

```
"struct Material {\n"
" vec3 albedo;\n"
" float ka;\n"
" vec3 kd;\n"
" vec 3 ks;\n"
"};\n"
```

它们可以在 shader 里一个一个属性地 set 过去。

3.2 光照

有三个光照模型。

环境光照

只是给物体上个底色，假设环境有个光存在，那么根据物体自己的颜色以及光反射率计算一个返回的颜色。

在这次实验的实现中，把环境光照当作 $(1.0, 1.0, 1.0)$，反射的环境光就只是 $ka * albedo$ 而已：

```
const char* fragCode =
  "#version 330 core\n"
  "out vec4 color;\n"
  "struct Material {\n"
  " vec3 albedo;\n"
  " float ka;\n"
  "};\n"
  "uniform Material material;\n"
  "void main() {\n"
  " vec3 result = material.ka * material.albedo;\n"
  " color = vec4(result, 1.0f);\n"
  "}\n";
```

可以看成是物体自己在沿着自己的法线方向发光，每个顶点的光强都是一样的，所以也没有阴影。能看出物体的轮廓。

漫反射光照

漫反射光照是根据入射光线与法线夹角的大小关系来反应反射光的光强大小，从而形成明暗的效果。而这个大小关系大致就是夹角的余弦值。

所以对于一个顶点，我们需要计算光源到这个点的入射向量，并和法向量求一个点积，得到漫反射系数，再和相应的材质信息得到漫反射光线对颜色的影响：

```
"vec3 calcDirectionalLight(vec3 normal) {\n"
" vec3 lightDir = normalize(-directionalLight.direction);\n"
" // diffuse color\n"
" vec3 diffuse = directionalLight.color * max(dot(lightDir, normal), 0.0f) *
    material.kd;\n"
" return directionalLight.intensity * diffuse ;\n"
"}\n"

"vec3 calcSpotLight(vec3 normal) {\n"
" vec3 lightDir = normalize(spotLight.position - FragPos);\n"
" float theta = acos(-dot(lightDir, normalize(spotLight.direction)));\n"
" if (theta > spotLight.angle) {\n"
```

```
12    "   return vec3(0.0f, 0.0f, 0.0f);\n"
13    " }\n"
14    " vec3 diffuse = spotLight.color * max(dot(lightDir, normal), 0.0f) * material.
        kd;\n"
15    " float distance = length(spotLight.position - FragPos);\n"
16    " float attenuation = 1.0f / (spotLight.kc + spotLight.kl * distance +
        spotLight.kq * distance * distance);\n"
17    " return spotLight.intensity * attenuation * diffuse;\n"
18    "}\n"
```

首先是一个平行光源，和点光源的区别是我们不需要考虑不同位置光强的区别——它是从无穷远处射过来的。所以只要简单地方向作点积就好。

对于点光源，我们不仅要计算入射光线的夹角，还要计算距离对光强的影响——它大致是关于距离的一个二次函数（穿过不同球面的光强和是相同的），kc、kl、kq 定义了这个衰减的特性。乘上这一部分的影响就好了。

当然这中间有一个问题是，我们对模型作缩放变换时，它的法向量也会相应地变化——不与那个平面平行了。所以我们要通过现有的 model 矩阵修复这个法向量。推导也非常简单：原模型的切线向量 T 与法线向量 N 满足：

$$dot(T, N) = 0$$

现在对模型关于矩阵 M 做了变换，我们要求出一个法线向量的变换矩阵 A 使得

$$dot(MT, AN) = 0$$

那么

$$(MT)^T AN = 0$$
$$T^T M^T AN = 0$$

因为

$$dots(T, N) = 0$$

所以

$$M^T A = I$$
$$A = (M^{-1})^T$$

这一部分体现在片段着色器中。

镜面反射

镜面反射与观察者的视角有关，通过反射光相对与视线方向角度差确定镜面反射的光强，从而形成高光。所以在计算时需要提供摄像机的坐标。同时根据材质的反光度来计算反光程度。

因为这一部分时自己实现，所以代码详见下一节。

四、 代码实现

4.1 shader 部分

```
"#version 330 core\n"
"in vec3 FragPos;\n"
"in vec3 Normal;\n"
"out vec4 color;\n"
"// material data structure declaration\n"
"struct Material {\n"
" vec3 albedo;\n"
" float ka;\n"
" vec3 kd;\n"
" vec3 ks;\n"
" float ns;\n"
"};\n"

"// directional light data structure declaration\n"
"struct DirectionalLight {\n"
" vec3 direction;\n"
" float intensity;\n"
" vec3 color;\n"
"};\n"

"// spot light data structure declaration\n"
"struct SpotLight {\n"
" vec3 position;\n"
" vec3 direction;\n"
" float intensity;\n"
" vec3 color;\n"
" float angle;\n"
" float kc;\n"
" float kl;\n"
" float kq;\n"
"};\n"

"// uniform variables\n"
"uniform Material material;\n"
"uniform DirectionalLight directionalLight;\n"
"uniform SpotLight spotLight;\n"
"uniform vec3 viewPos;\n"

"vec3 calcDirectionalLight(vec3 normal) {\n"
" vec3 lightDir = normalize(-directionalLight.direction);\n"
" vec3 viewDir = normalize(viewPos - FragPos);\n"
" vec3 reflectDir = reflect(-lightDir, normal);\n"
" float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.ns);\n"
" vec3 specular = 0.5 * spec * directionalLight.color * material.ks;\n"
```

```
" // diffuse color\n"
" vec3 diffuse = directionalLight.color * max(dot(lightDir, normal), 0.0f) *
    material.kd;\n"
" return directionalLight.intensity * (diffuse + specular);\n"
"}\n"

"vec3 calcSpotLight(vec3 normal) {\n"
" vec3 lightDir = normalize(spotLight.position - FragPos);\n"
" float theta = acos(-dot(lightDir, normalize(spotLight.direction)));\n"
" if (theta > spotLight.angle) {\n"
"   return vec3(0.0f, 0.0f, 0.0f);\n"
" }\n"
" vec3 viewDir = normalize(viewPos - FragPos);\n"
" vec3 reflectDir = reflect(-lightDir, normal);\n"
" float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.ns);\n"
" vec3 specular = 0.5 * spec * spotLight.color * material.ks;\n"

" vec3 diffuse = spotLight.color * max(dot(lightDir, normal), 0.0f) * material.kd
    ;\n"
" float distance = length(spotLight.position - FragPos);\n"
" float attenuation = 1.0f / (spotLight.kc + spotLight.kl * distance + spotLight.
    kq * distance * distance);\n"
" return spotLight.intensity * attenuation * (diffuse + specular);\n"
"}\n"


"void main() {\n"
" vec3 normal = normalize(Normal);\n"
" // ambient color\n"
" vec3 ambient = material.ka * material.albedo;\n"
" // diffuse color\n"
" vec3 diffuse = calcDirectionalLight(normal) + calcSpotLight(normal);\n"
" color = vec4(ambient + diffuse, 1.0f);\n"
"}\n";
```

主要工作其实在点光源和平行光都是一样的：

```
" vec3 viewDir = normalize(viewPos - FragPos);\n"
" vec3 reflectDir = reflect(-lightDir, normal);\n"
" float spec = pow(max(dot(viewDir, reflectDir), 0.0), material.ns);\n"
" vec3 specular = 0.5 * spec * directionalLight.color * material.ks;\n"
```

计算视角方向、计算反射光向量，计算镜面反射强度。最后根据向量加法的结合律，一起加起来返回就好了。

4.2 shader 传参部分

因为镜面反射新增了镜面反射率、观察位置以及物体的镜面材质这三个参数，所以需要额外传参，就不贴代码了。

五、 实验结果

5.1 phong 光照模型单独展示

没有怎么调参，感觉暗的时候比较容易看得出镜面的效果，这个材质有点像橡胶或是陶瓷什么的。

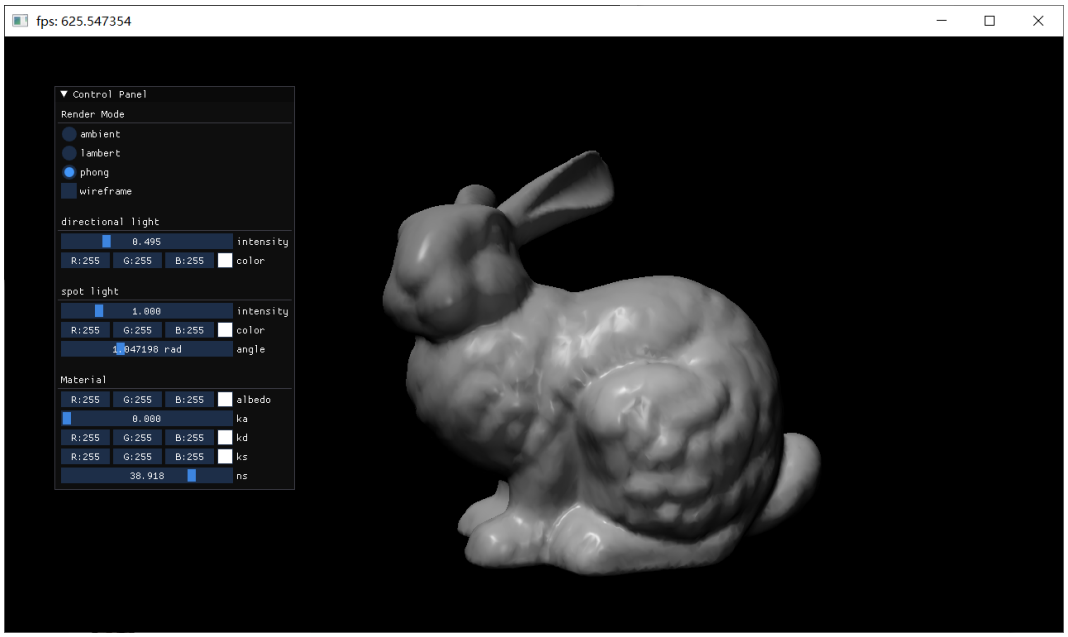

图 1: 正面

顺便把照相机设成了 lookat 的样子来看看背面

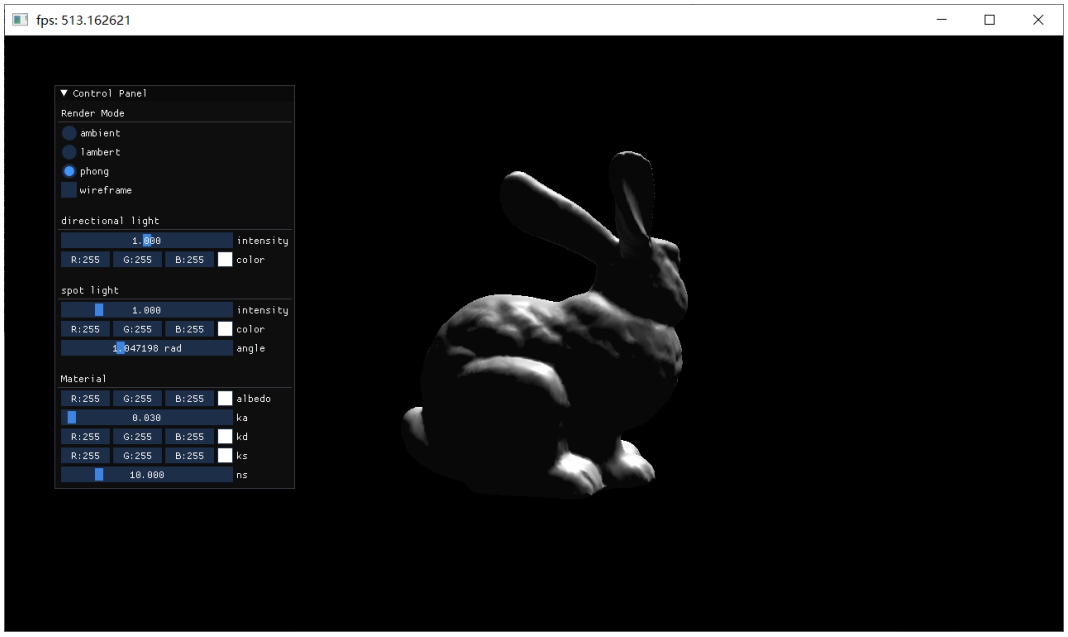

图 2: 背面

然后调整 ks 和 ns，可以发现 ks 大概就是高光部分的颜色，而 ns 越小，反射光强度越高高光的部分越多。

图 3: ns 小

图 4: ns 大

当然这个点光源也非常影响镜面反射，因为点光源力模型较近，当它的光强比较强，高光就不明显。当然这个聚光灯可以调角度，也挺有意思。因为篇幅问题就不截图展示。

5.2 几种光照模型比较

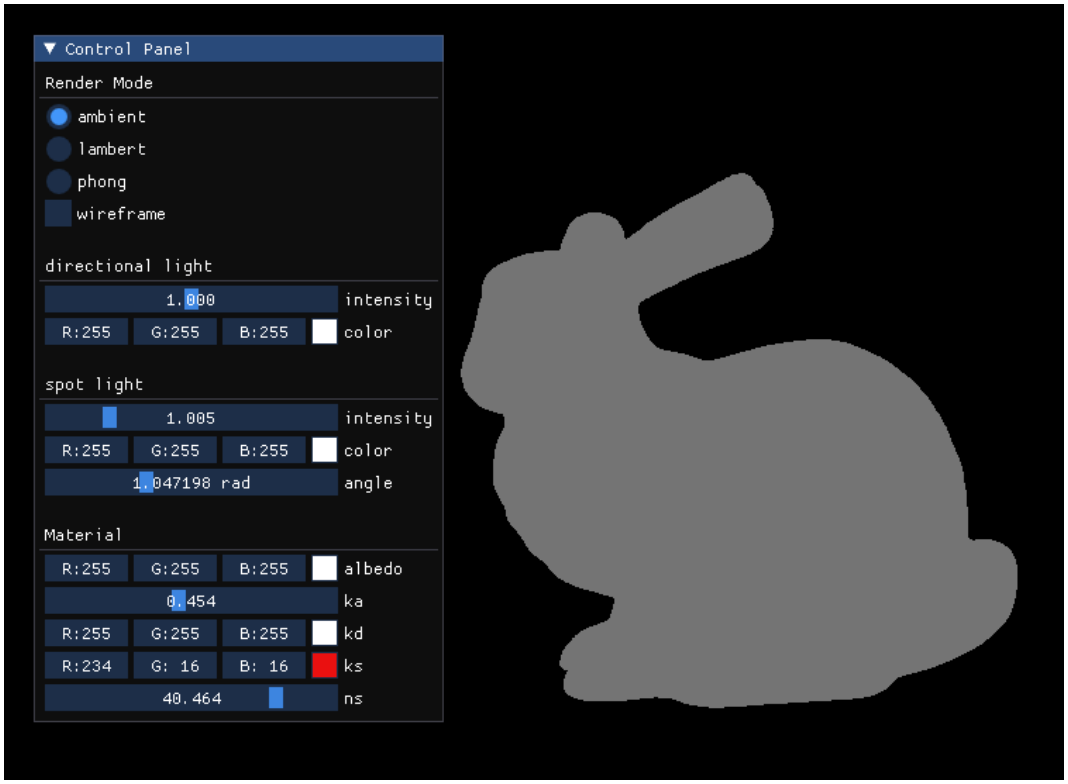

图 5: ambient

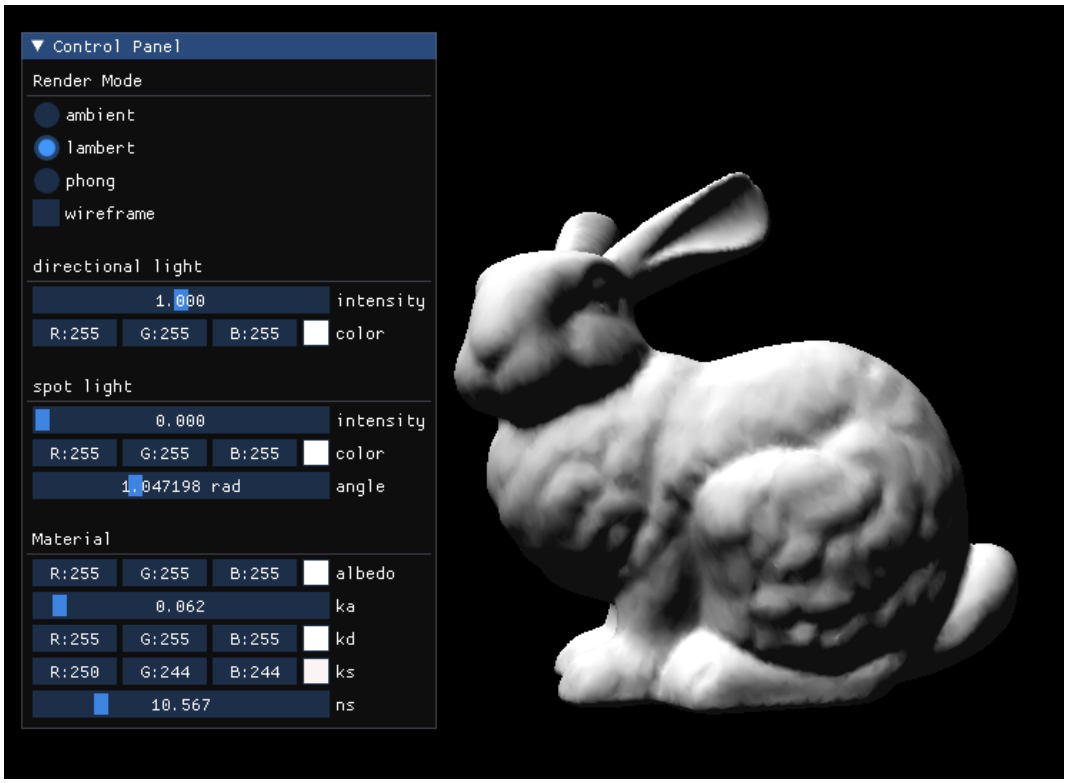
Control Panel
Render Mode
ambient
lambert
phong
wireframe
directional light
1.000
intensity
R:255
G:255
B:255
color
spot light
0.000
intensity
R:255
G:255
B:255
color
1.047198 rad
angle
Material
R:255
G:255
B:255
albedo
0.062
ka
R:255
G:255
B:255
kd
R:250
G:244
B:244
ks
10.567
ns

图 6: lambert

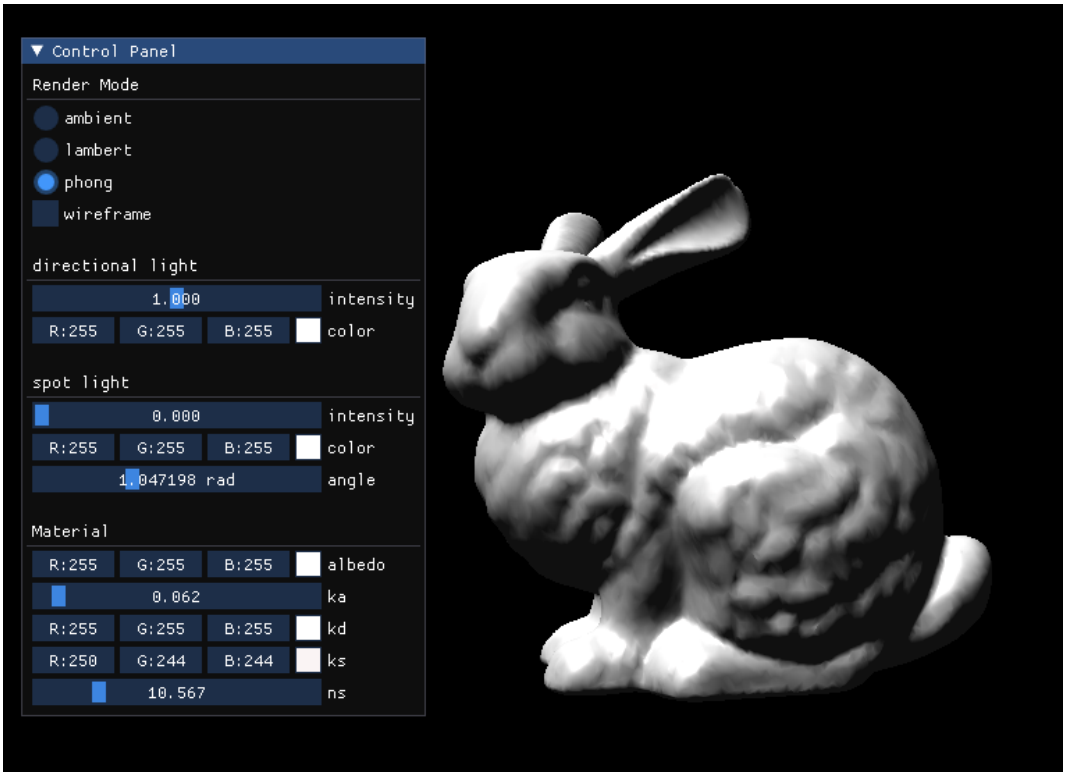

图 7: phong

可以看到综合了各种因素的 phong 光照模型还是非常优秀的。

六、 讨论和心得

在这次实验中我学习了各种光照模型的原理和实现，并对最后的模型做了许多尝试，感觉很有意思。也发现在建模中为了达到最好的效果有那么多的参数需要调整，要是要做一些实体建模达到更好的效果看来也需要许多实验测量。

但是在实验中也发现之前写的一些视角控制好像不是很好用，在接下来会优化 camera 的性能。