2018图形学LAB2——基础光照&&基于BRDF的测量模型

Shi Ruofei
Nanjing University

这篇文档和本次作业的主要关注点在经验模型和测量模型上，加上我时间和水平有限，文中提到的"辐射率"、"光照强度"、"反射/折射"等有物理意义的概念在使用上下文和量纲等方面并不准确，希望大家意会我想表达的抽象概念而不必纠结某些量上的问题
仅通过阅读这篇文档你应该足以完成本次作业，我附上了去年罗浩然学长的关于测量模型的PDF和南大计科《图形绘制技术》课程的一篇关于材质表现的PPT，两者在一定程度上均基于《PBRT》，涉及了很多系统性概念，有兴趣的同学可以看看

我们如何看见这个"世界"

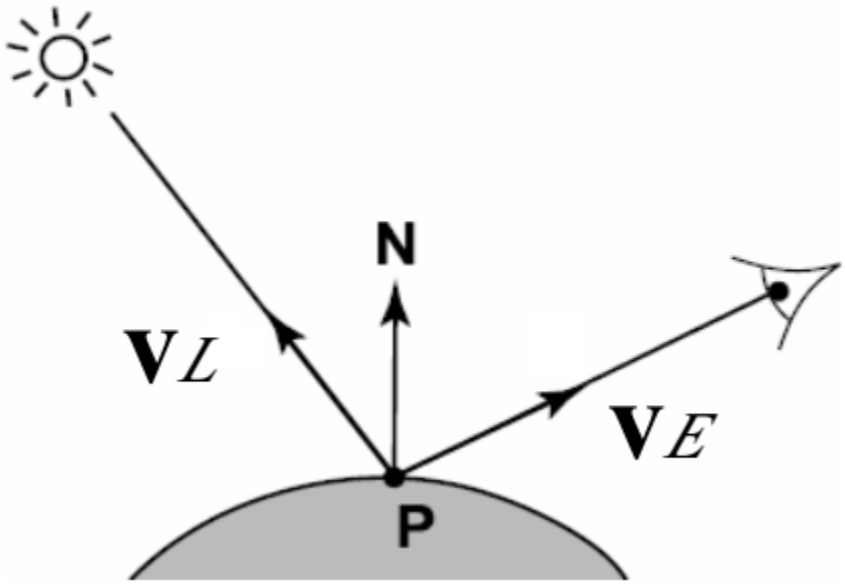
3D模型&&.obj文件

我们已经知道GPU渲染管线需要一系列顶点数据，配合索引，可以表示三角形面片，从而构成几何体的基本单元。同样，一些常见的模型文件的存储数据也采用了罗列顶点和索引的方式，例如在作业/resource/objects内的.obj格式文件。
OBJ是Wavefront科技开发的一种几何体图形文件格式。

> 一个平面在.obj内的表示方式
> v 2.000000 -2.000000 0.0
> v -2.000000 -2.000000 0.0
> v -2.000000 2.000000 0.0
> v 2.000000 2.000000 0.0
> vn 0.0000 0.0000 1.0000
> vt 0.999999 0.000000
> vt 0.000000 0.000000
> vt 0.000000 0.999999
> vt 0.999999 0.999999
> f 1/1/1 2/2/1 3/3/1 4/4/1

法线：代表垂直于顶点所处平面的方向，任意一点的法线向量可以根据其所处的面片的所有顶点的法线向量插值得到（当然，在片段着色器中的法线向量值已经被渲染管线自动插值过啦）

表面着色

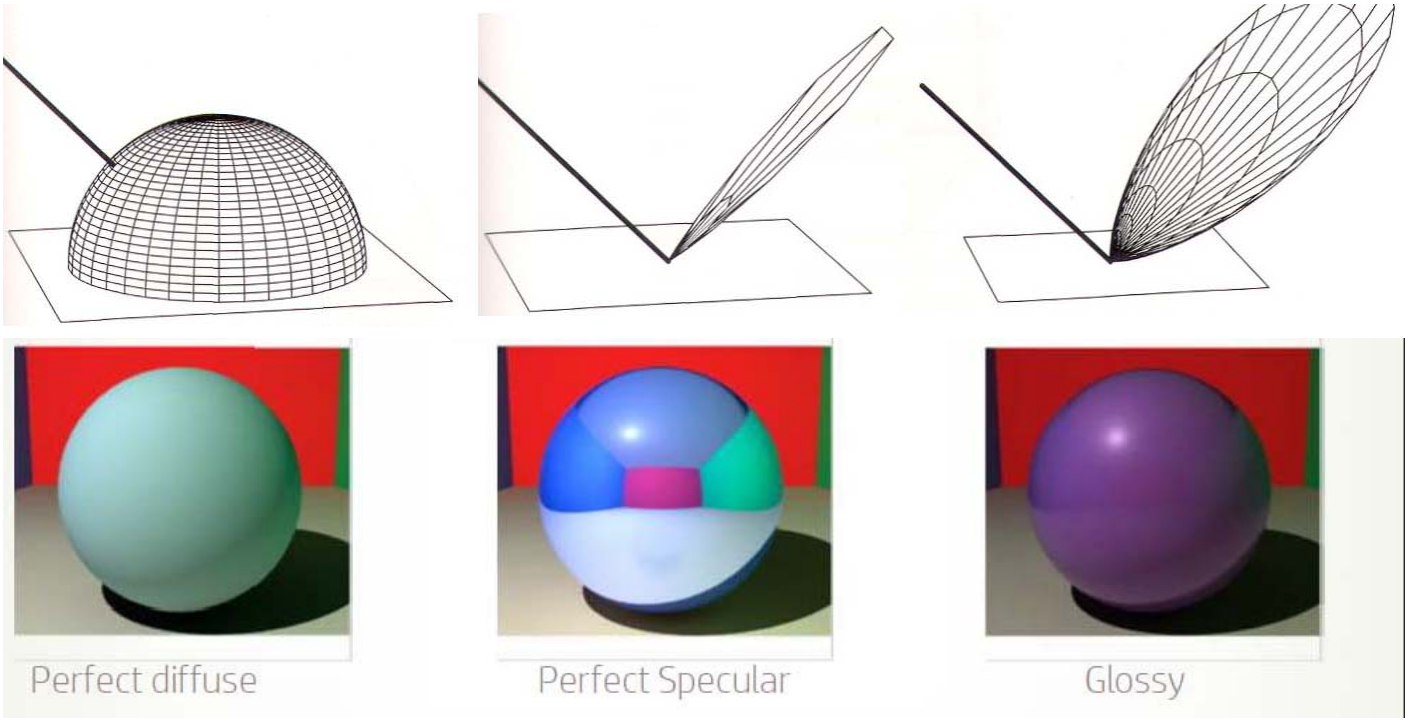


$$L_o(P,\omega_o) = L_e(P,\omega_o) + \int_{H^2(n)f} f_r(P,\omega_i,\omega_o)L_i(P,\omega_i)cos(\theta_i)d\omega_i$$

上述方程一般被称为渲染方程(rendering equation)或着色方程(shading equation)，描述了大部分光学现象所形成的着色效果（磷光、荧光、干涉、次表面散射等无法用此方程描述）
P：表面某点
ω_o：光线出射方向
ω_i：光线入射方向
$H^2(n)$：法线方向所处的半球面
$fr(P,\omega_i,\omega_o)$：P点在ω_o和ω_i为参数时的BRDF值

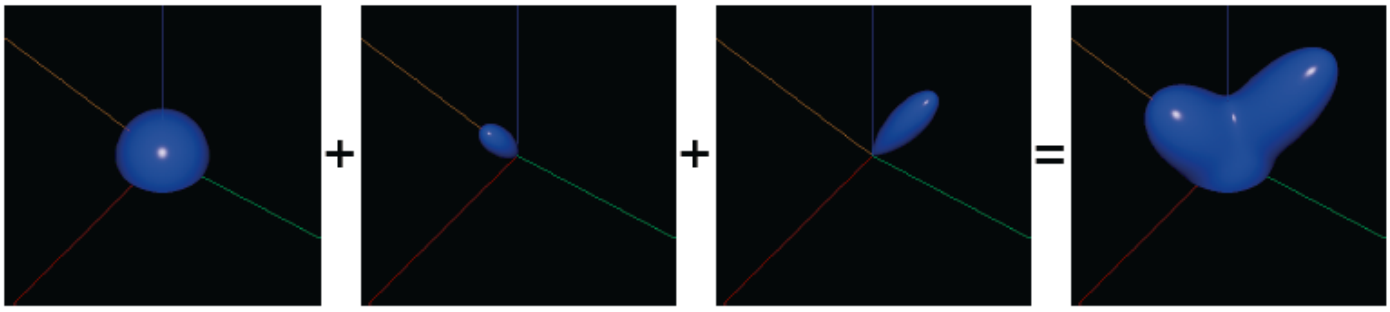
BRDF——双向反射分布方程

描述了物体表面某点在ω_o方向的出射辐射率与在ω_i方向的入射辐射率的比值
BRDF可以代表许多不同类型的反射材质：



从左到右依次是兰伯特漫反射(diffuse)、完美镜面反射、光泽镜面反射(glossy specular)
可以看出漫反射的特点为BRDF值不会随着出射光线的方向而改变，因此无论在何处观察物体都是一样的效果，而对于镜面反射来说，观察角度离入射光线的反射方向越近，光线强度越高，在完全镜面反射中甚至只有在反射方向进行观察才能看见物体表面

然而现实世界的材质极其复杂，基本上无法使用这些简单的模型表示，而且会受到许多因素的影响(难以量化参数)：

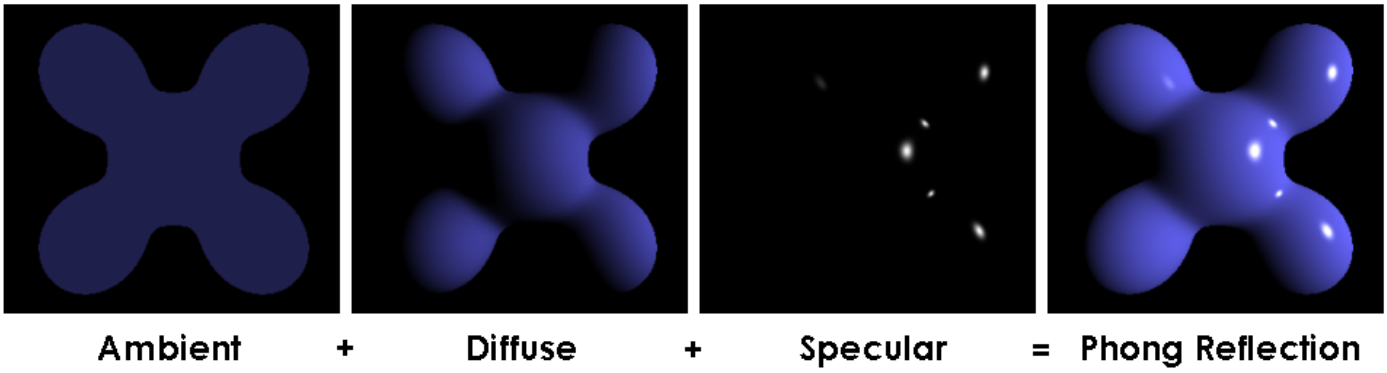


描述表面反射模型的一些方法：

1. 基于现象学的模型：定性方法，通过一些直观的参数调整逼近真实世界的光照现象
2. 基于测量数据：通常以表格形式或一组基函数的系数来描述
3. 基于物理和几何的方法：Physical based rendering，例如微表面模型

基础光照

Phong reflecting model



phong氏光照模型将从表面反射的光分解为3个独立项：

1. 环境(ambient)项为场景中间接反射光的粗略估计，虽然我们主要关注来自光源的直接光照，但间接反射光使物体的阴影部分不会完全黑暗，而是有一些微弱的颜色能看出物体的轮廓
2. 漫反射(diffuse)项模拟直接光源在表面均匀地向各个方向反射，是phong氏光照模型中视觉上最显著的分量(?)，物体表面上的点越正对光源越亮，而效果不因观察角度而改变
3. 镜面发射(specular)项模拟在光滑表面会看到的高亮亮光，镜面高光会出现在光源相对物体表面的直接反射方向

Phong氏模型的输入一般包括：

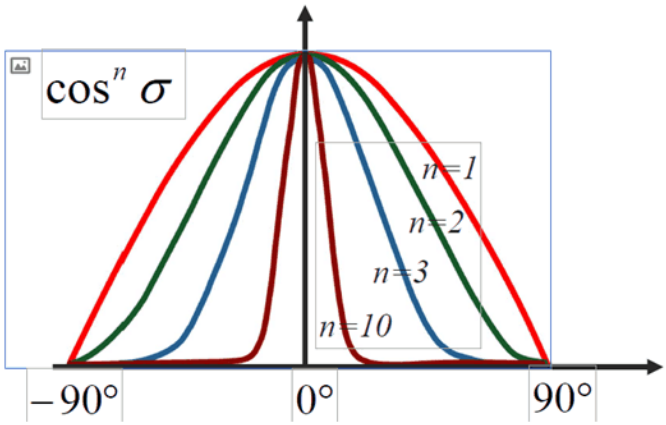
1. 视线方向矢量$\vec{V} = [V_x, V_y, V_z]$是从光线反射点(即物体表面某点)延伸至观察摄像机的方向
2. 3个颜色通道的环境光强度$\vec{A} = [A_R, A_G, A_B]$
3. 光线到达表面上那一点的法线$\vec{N}$
4. 表面的材质反射属性，包括环境反射量k_a，漫反射量k_d，镜面反射量k_s（在实现时这三个值都可以用3维向量表示RGB的不同分量，使得物体在白光下可以呈现不同颜色）和镜面光滑度(glossiness)幂α
5. 每个光源的属性，包括：光源的颜色及强度$\vec{C} = [C_R, C_G, C_B]$，从反射点至光源的方向矢量$\vec{L}$

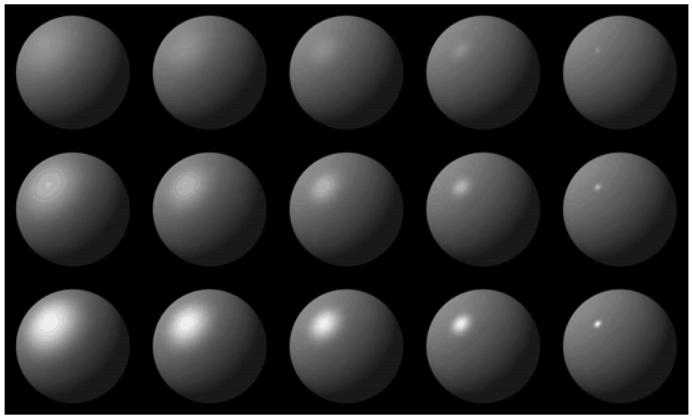
表面上某点反射的光强度I可以表示为以下方程：

$$I = k_a\vec{A} + \sum[k_d(\vec{N}\cdot\vec{L}) + k_s(\vec{R}\cdot\vec{V})^{\alpha}]C$$

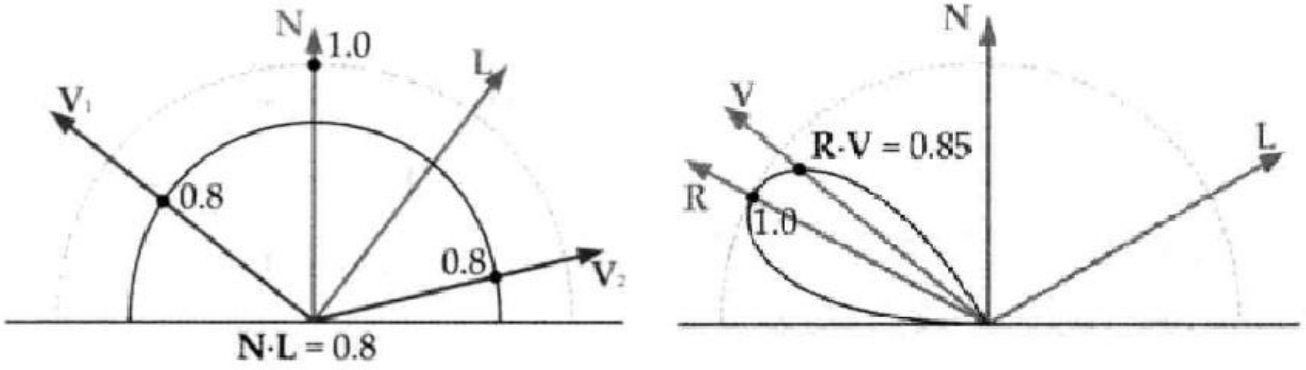
可以看到在漫反射分量中，光线入射方向与法线的夹角越小，对表面颜色的贡献越大(与夹角的余弦值成正比)，因此可以用标准化后光线方向向量和法线方向向量的点乘作为系数

而在镜面反射分量中，光线对表面着色的贡献正比于观察方向与光线反射方向的夹角余弦值的幂，公式里的反射方向$\vec{R}$可以用公式$2(\vec{N}\cdot\vec{L})\vec{N} - \vec{L}$求得，在glsl语言中也有内置的*reflect*函数可以直接获得
其中glossiness代表表面的光滑度，这个值越高，镜面反射所形成的高光越“尖锐”：





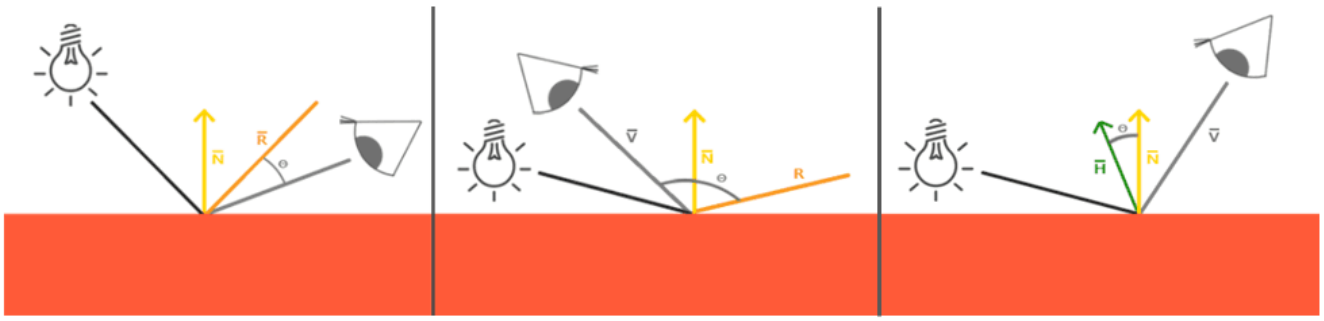
右图从上到下k_s值逐渐变大，从左到右glossiness值逐渐变大



phong氏模型的漫反射项和镜面反射项的BRDF函数在二维空间的情况

Blinn-Phong reflecting model

Blinn-Phong反射模型是Phong反射模型的变种，两者非常相似，只是在计算镜面反射时有些许差别

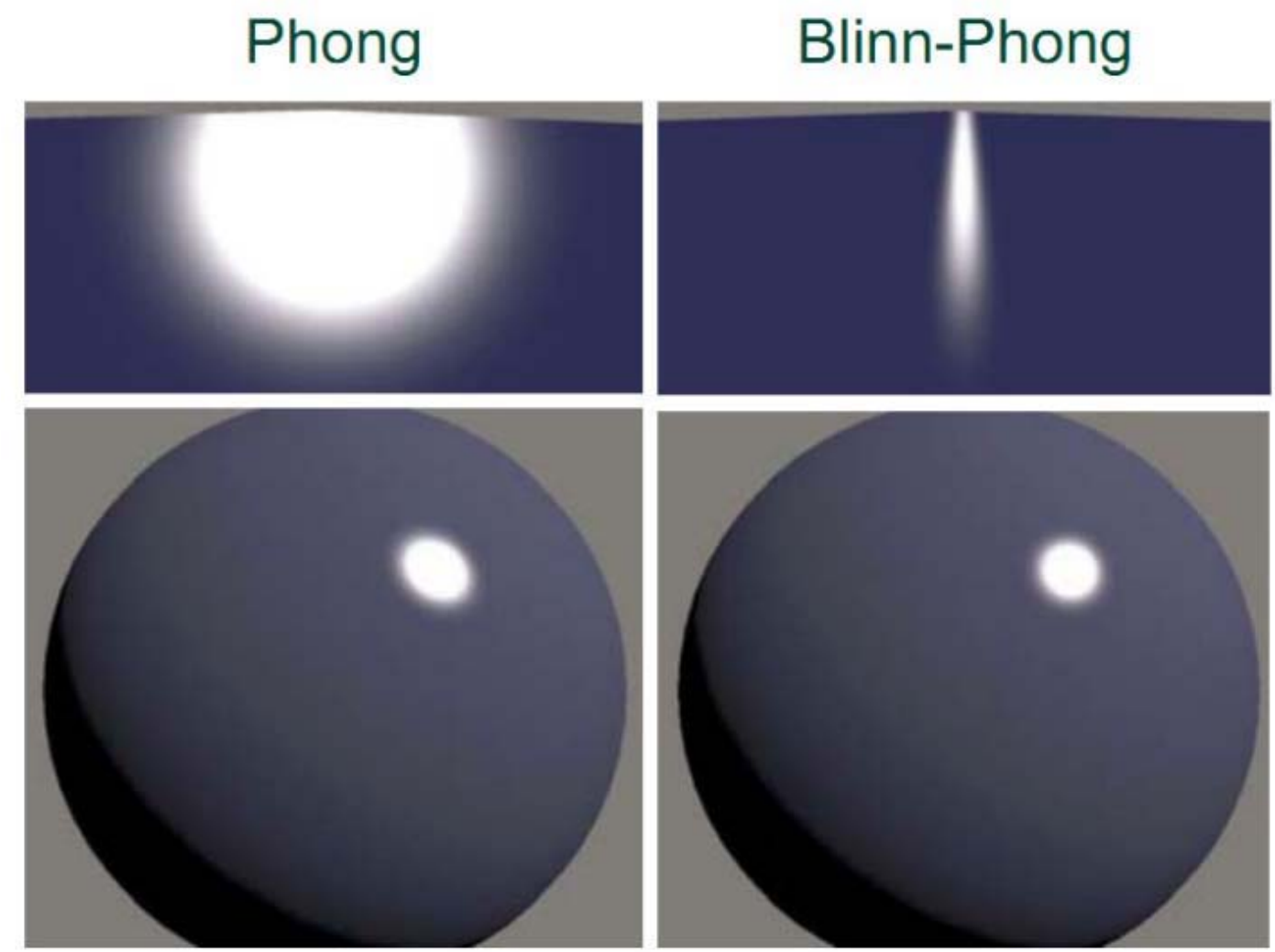


左图是我们熟悉的镜面反射场景，而中图中，视线与反射方向之间的夹角明显大于90度，这种情况下镜面光分量会变为0。这在大多数情况下都不是什么问题，因为观察方向离反射方向非常远。然而，当物体的镜面光分量非常小时，它产生的镜面高光半径足以让这些相反方向的光线对亮度产生足够大的影响。在这种情况下就不能忽略它们对镜面光分量的贡献了。
(这段话出自LearnOpenGL，确实有一定的道理，但不能因此认为phong是错误的模型而blinn-phong是正确的，因为两者都是试图模拟某些材质的经验模型，只能说某些情况下后者更接近实验测量的数据)

blinn-phong模型中引入了半程向量(halfway vector)的概念，$\vec{H}$为视线方向矢量和光线方向矢量的角平分线方向，公式可以简单地表示为

$$\vec{H} = \frac{\vec{L} + \vec{V}}{||\vec{L} + \vec{V}||}$$

然后将phong氏模型中的镜面分量$(\vec{R} \cdot \vec{V})^{\alpha}$替换为半程向量和法线向量之间的关系$(\vec{H} \cdot \vec{N})^{\alpha}$，可以发现半程向量和法线向量的夹角不会在大于90度，从而规避了上文提及的问题。
blinn-phong的glossiness值α和phong氏模型的并不一致，因为blinn-phong所关注的两个向量间的夹角总是比phong的两个要小一些，如果想获得接近的高光强度，应该将glossiness值设得更大，一般是phong的2到4倍



上述光照模型的弊端

诚然这两种光照模型非常的简单高效，也易于理解和实现，但也有相当的局限性：

1. 模拟出来的材质都是接近塑料的粗糙质感(?)
2. 镜面高光的模拟非常简陋，无法体现各向异性、菲涅尔反射等现实中到处可见的现象
3. 电介质的镜面反射分量应该不受物体颜色影响，但金属类的材质则会反射同色的高光（可以通过设置有rgb分量的ks参数进行控制）
4. 依赖不同的漫反射纹理、镜面反射纹理、法线纹理等丰富物体的色泽表现

5. 现在GPU性能进步飞速，所谓的"高性能"并不一定有用武之地

基于BRDF的测量模型

由 MERL（三菱电机研究实验室）测的、近100种材料的表面BRDF，并据此构建了MERL BRDF数据库

http://people.csail.mit.edu/wojciech/BRDFDatabase/

一个表示材质的文件代表一个3维数组，考虑到一个空间内的单位方向矢量需要θ和ϕ表示，而两个方向之间的相对关系则至少需要三个角度值才能唯一确定，所以本质上这就是一个查表过程啦~

在上文链接的/code/BRDFRead.cpp中有完整的如何读取BRDF文件并根据入射方向和出射方向查表的代码，考虑到实现的背景不同（我们需要在片段着色器中查表），下文是罗浩然学长给出的详细解析与查表方法：

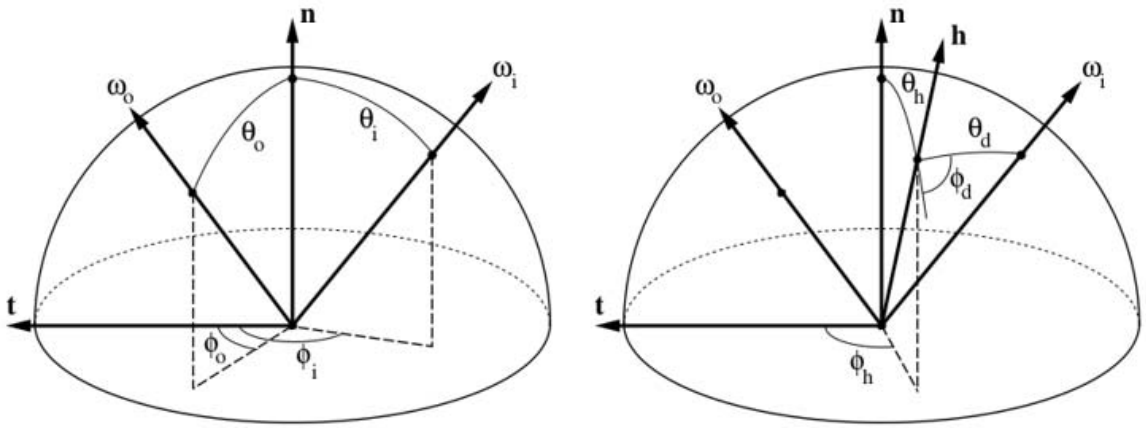


Figure 2: Proposed reparameterization of BRDFs. Instead of treating the BRDF as a function of (θ_i, ϕ_i) and (θ_o, ϕ_o), as shown on the left, we consider it to be a function of the halfangle (θ_h, ϕ_h) and a difference angle (θ_d, ϕ_d), as shown on the right. The vectors marked $\vec{n}$ and $\vec{t}$ are the surface normal and tangent, respectively.

半角：半程向量所对应的立体角，对于各向同性材料，半角甚至可以简化为半向量的天顶角θ_h
差分角：半向量与入射角的差分θ_d和ϕ_d
MERL BRDF数据库的三个维度值分别为

$$\sqrt{\frac{\theta_h}{\pi/2}} \qquad \frac{\theta_d}{\pi/2} \qquad \frac{\phi_d}{\pi}$$

tips：这里三个值都在0-1之间，对应的3个维度的size分别为90、90和180，所以取值应该为

```
// 这只是个伪代码，实际实现中你可能无法直接使用这样的寻址方法
// 并且需要取RGB三个颜色通道的数据(当然方法是一样的)
// 除此之外这里直接用int取整，也可以尝试运用第一次作业的知识做一个三线性插值得到更平滑的效果
merl[int(sqrt(2*theta_h/pi)*90)][int((2*theta_d/pi)*90)][int(180*phi_d/pi)]
```

如何在实时渲染中将入射角和出射角转化为半角和差分角

对三维变换还不熟悉的同学可以先复习一下旋转矩阵和叉乘
注意当使用点乘代表余弦值前一定要标准化向量

设n, i, o, h分别为**标准化后**法线、入射向量、反射向量和半程向量
$\theta_h = arccos(n \cdot h) \quad \theta_d = arccos(i \cdot h)$

如何求ϕ_d?

原计算方法中，需要将整个坐标系先绕z轴旋转$-\omega_h$平面所在的方向，然后绕y轴旋转$-\theta_h$，使得半向量居于z轴所在方向，并求ϕ_d

```
// 伪代码！注意坐标系！
// 求副切线向量，如果法线n代表这个坐标系的z轴，o处于xOz平面
// 那么副切线向量代表坐标系y轴
vec3 bitangent = normalize(cross(normal, o));
// 切线就代表x轴啦
vec3 tangent = normalize(o - dot(o, n)*n);
// 将i绕z轴转-phi_half
// -phi_half = atan(half.y, half.x)
vec3 tmp = rotate_vector(i, normal, -phi_half);
// 将tmp绕y轴转-theta_half
// 同样的旋转如果作用于half向量，那么它现在会与法线对齐
vec3 diff = rotate_vector(tmp, bitangent, -theta_half);
// 在xOy上的投影
// 但这行代码并不对，因为这个坐标系的x轴和y轴并非(1,0,0)和(0,1,0)
phi_diff = atan(diff.y, diff.x);
```

另一种方法——考虑到旋转变换不改变角度之间的关系，直接变换坐标系而不再使用旋转：

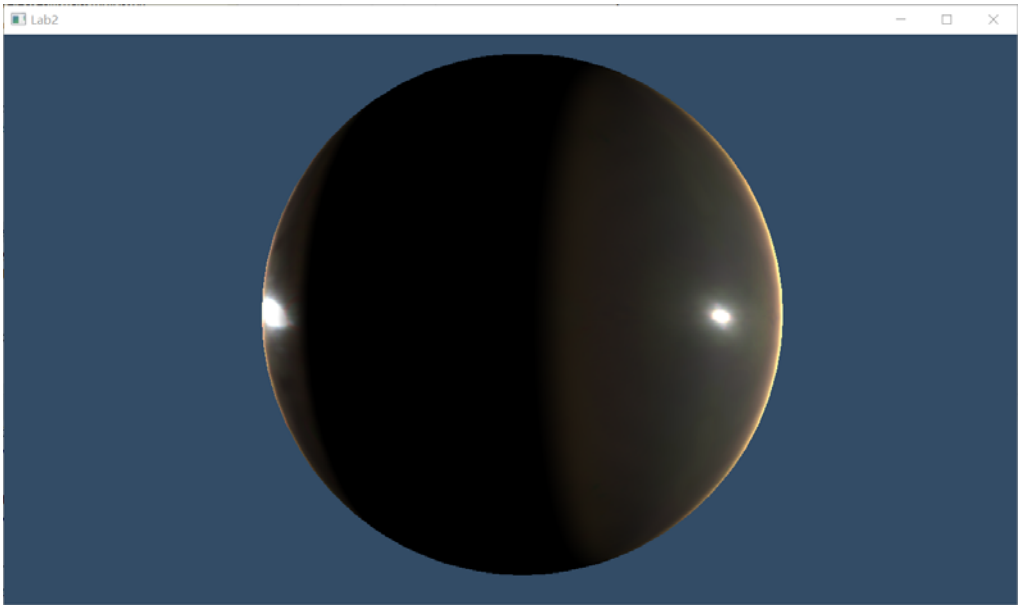
$$h_x = h - \frac{n}{n \cdot h} \quad h_y = h \otimes h_x \quad p = i - (i \cdot h)h$$

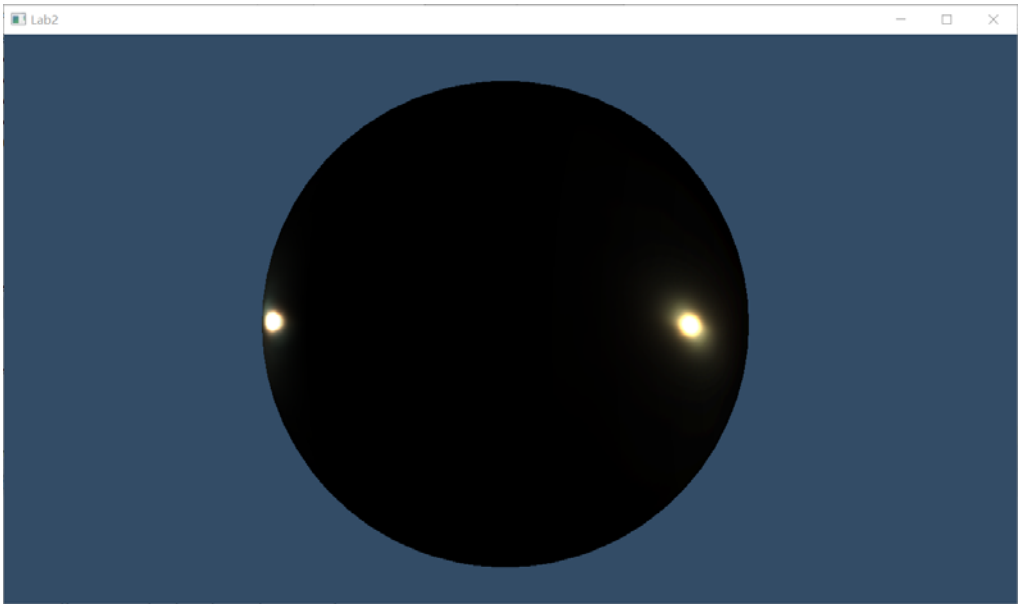
$$h_x = \frac{h_x}{||h_x||} \quad h_y = \frac{h_y}{||h_y||} \quad p = \frac{p}{||p||}$$

$$\phi_d = \begin{cases} arctan(p \cdot h_y, p \cdot h_z) & n \neq h \quad and \quad n \neq i \\ \frac{\pi}{2} & n = h \quad or \quad n = i \end{cases}$$

两种方法的目的都是将半向量h作为新坐标系的z轴，而n和h确定的平面处于新坐标系的x轴。方法二的h_x和h_y确定了新坐标系x轴和y轴，p则是向量i平行于新坐标系xOy平面的分量啦

可以通过在作图/划辅助线等来帮助理解这个坐标系变换的过程





左边是尼龙nylon右边是黄铜brass，可以看到因菲涅尔现象造成的较亮边缘，而对于金属来说，只有镜面反射，而且反射光拥有金属本身的颜色

测量模型的特点

采样点密集，数据量庞大，不适合实时渲染领域(如游戏)
一个测量数据集对应一种固定的材质，无法通过参数调整细节
可以用于离线渲染(如电影)，或在图形学研究中衡量其他模型的真实程度

更"好"的模型与方法……?

作业相关说明与要求

使用框架项目并实现自己的shader

框架使用的外部依赖相比上次作业多了一个只有头文件的rapidjson库，因此原样配置并cmake即可

通过编辑config.json文件搭建自己的渲染场景：

1. 数组可以为空，其他所有没有标注optional的项都必须存在，否则场景初始化会失败
2. 请大家保证json文件内格式正确，例如可以用一些IDE自带的纠错功能，防止因为json无法解析而程序崩溃
3. 一些模型文件可能过大/过小导致场景内看不到，需要调整scale参数，可以打开obj文件看里面顶点数据(v后面的)的大小，剪裁空间的视口大小是[-1, 1]*[-1, 1]，例如earth.obj模型是一个很精细的球体模型，但需要缩小百倍才有合适的大小

```
{
    "scene": {
        "camera": {
            "position": "0.0 0.0 6.0",      //相机初始位置
            "movement_speed": 5.0,          //optional 相机移动速度
            "mouse_sensitivity": 0.1        //optional 鼠标灵敏度
        },
        "point_lights": [
            {
                "position": "0.0 0.0 -10.0",    //点光源位置
                "radiance": "1.0 1.0 1.0",      //点光源三个通道的辐射强度
                "constant": 1.0,                //衰减常数项
                "linear": 0.09,                 //衰减一次项
                "quadratic": 0.032              //衰减二次项
            },
            // other point lights.....
        ],
        "direction_lights": [
            {
                "direction": "-0.2 -1.0 -0.3",  //平行光源方向
                "radiance": "0.1 0.1 0.1"       //平行光源辐射强度
            },
            // other direction lights.....
        ],
        "objects": [
            {
                //obj模型文件路径
                "obj_file_path": "resource/objects/cube.obj",
                //顶点着色器的文件路径
                "vertex_shader_file_path": "shader/vertex.glsl",
                //片段着色器的文件路径
                "fragment_shader_file_path": "shader/phong_fs.glsl",
                //模型位置
                "position": "0.0 0.0 0.0",
                //旋转
                "rotate_x": 0,

                "rotate_y": 0,
                "rotate_z": 0,
                //缩放
                "scale": "1.0 1.0 1.0",
                "material": {
                    //optional 使用的brdf测量文件
                    //如果material里有这一项后面的几个都不需要了
                    "brdf_file_path": "resource/brdfs/nylon.binary",
                    //环境光反射系数
```

```
                    //环境光反射系数
                    "ka": "0.05 0.05 0.05",
                    //漫反射系数
                    "kd": "1.0 1.0 1.0",
                    //镜面反射系数
                    "ks": "1.0 1.0 1.0",
                    //光滑度
                    "glossiness": 8,
                    //optional 使用的纹理贴图
                    //如果没有这一项texture的位置会是一个1*1的白色纹理
                    "texture_file_path": "resource/textures/wood.png"
                }
            },
            // other objects.....
        ]
    }
}
```

自行创建、编写不同算法的片段着色器，在/resource目录下可以看到许多.obj模型文件和纹理图片文件（你也可以从网上下载.obj模型文件和纹理图片），挑选自己喜欢的组合吧
tips：对config.json和shader文件的修改一般不会让IDE自动执行build操作，可能修改后的shader和config文件没有被复制到编译后的执行目录，在调试阶段一个可行的方法是修改执行目录的config.json，而config里的shader路径可以填写项目目录里的绝对路径

Part1 实现phong氏光照模型和blinn-phong模型(50%)

如果你上网查阅这两个老生常谈的模型，可能会发现公式表达上会有一些微妙的差别（主要是在环境光分量的计算和在引入纹理贴图时高光的表现上），而上文所使用的公式参考自《游戏引擎架构》的第十章
甚至上文对模型的一些定性言论都是十分狭隘的(?)
比如这篇知乎专栏金属、塑料，傻傻分不清楚 就直接打了我的脸

但是这些都不重要，当我们谈及这两个模型，我们主要关注的是漫反射项和光线角度与法线夹角的关系，以及镜面反射高光和视线方向与反射方向夹角的关系，即$(\vec{N}\cdot\vec{L})$和$(\vec{R}\cdot\vec{V})^{\alpha}$，因此只要能体现这两点，在视觉上一般都可以得到"正确"的结果

通用的顶点着色器已经给出，一般来说所有物体都可以用这个简单的shader建立顶点：

```
#version 410 core
layout (location = 0) in vec3 vPos;
layout (location = 1) in vec3 vNormal;
layout (location = 2) in vec2 vTexCoord;
layout (std140) uniform Matrices {
    mat4 view;
    mat4 projection;
};
uniform mat4 model;
out VS_OUT {
    vec3 FragPos;
    vec3 Normal;
```

```
    vec2 TexCoord;
} vs_out;
void main() {
    vs_out.FragPos = vec3(model * vec4(vPos, 1.0));
    vs_out.Normal = mat3(transpose(inverse(model))) * vNormal;
    vs_out.TexCoord = vTexCoord;
    gl_Position = projection * view * model * vec4(vPos, 1.0);
}
```

除了多了个法线外其他的和第一次作业的顶点着色器没什么不同，这里由顶点传入的vNormal值需要经过一个"模型矩阵左上角的逆矩阵的转置矩阵"变换，这是为了消除model矩阵里的位移分量并修复不等比缩放的影响，至于如何推导可以参考 http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/the-normal-matrix/

片段着色器便是需要自己实现光照模型的地方啦，了解每个uniform代表的含义和如何使用后实现起来就非常简单了

点光源衰减

一个可以增强实感的做法便是让点光源的强度随距离衰减，直接与距离的平方成反比也许很符合物理理论，但在渲染场景时这么做容易让离光源近的物体过亮，远的物体又过于昏暗，使用一个基于现象的函数计算衰减系数Attenuation可以取得更好的效果：

$$F_att = \frac{1.0}{K_c + K_l * d + K_q * d^2}$$

tipA：因为没有提供环境光强度$\vec{A}$的配置，可以直接用物体材质自己的k_a项乘上物体本身的颜色代替，这个值一般被设置得非常小，除非是在阴影处，不然这一项一般不会对视觉造成太大影响
tipB：现在可以用*texture(tex, TexCoord).rgb*作为物体自己的基础颜色分布啦（如果不设置纹理默认全部白色），一般来说漫反射项和环境光项都应该乘上物体自己的颜色分布来丰富其视觉表现，当然，系数k_d本身也是一个3维向量，你可以用其控制物体的"色调"
tipC：镜面反射项一般不乘物体本身的颜色（纹理颜色）使其效果更明显

Part2 利用查表实现基于MERL BRDF的测量模型(实现40%+三线性插值或实现了两种计算方法10%)

自行下载任意MERL BRDF文件，并在config里写好其路径

在片段着色器里MERL BRDF是一个可采样的三维数组sampler3D，和大家熟悉的sampler2D其实没啥差别，就是需要vec3作为采样目标参数
不幸的是OpenGL里的纹理存储方式的维度顺序是反过来的！

$$x = \sqrt{\frac{\theta_h}{\pi/2}} \qquad y = \frac{\theta_d}{\pi/2} \qquad z = \frac{\phi_d}{\pi}$$

所以你的采样函数需要这么写：

```
uniform sampler3D merlBrdf;
vec3 brdf = texture(merlBrdf, vec3(z, y, x));
```

上文章节已经给出了计算所需的采样位置的每个步骤，建议大家两个方法均进行尝试，阅读并理解BRDFRead.cpp可能可以解答你的疑惑，但需要注意片段着色器中的法线normal值并不是(0, 0, 1)

还记得最开始的渲染方程吗？我们现在求得的只是BRDF值，是出入光线强度的比例，所以还需要乘上$cos(\theta_i)$，配合光源强度和点光源衰减，而这个过程还需要对每个光源迭代后进行累加，渲染才算完成。

有兴趣的同学可以尝试进行三线性插值
用*texelFetch(sampler3D, ivec3)*函数时记得第二个参数是整数向量用来寻址，因此要将之前的几个0-1浮点数乘上每个维度的size

提交方式与思考题

关于项目无法运行、config.json的使用等问题请同学们尽早测试，如因为我的过失造成的问题请务必尽早联系我而不要拖到最后
理论上只需要提交config.json和写的所有shader，当然如果有什么对项目文件或资源文件的添加和改动也欢迎提交整个项目

如何模拟/投射影　？
如何生成一个球体?