南京大学

计算机科学与技术系

图形绘制技术

Lab2 实验报告

实验名称： 材质模型

学　　号： 201830204

姓　　名： 顾秋涵

实验时间： 2023.4

目　　录

一、 经验模型

1. Phong 光照模型

1.1 补全代码

（1）根据计算公式

$$
\begin{aligned}
L &= L_a + L_d + L_s \\
&= k_a I_a + k_d \frac{I}{r^2} \boldsymbol{n} \cdot \boldsymbol{l} + k_s \frac{I}{r^2} (\boldsymbol{v} \cdot \boldsymbol{l_r})^p
\end{aligned}
$$

将各个变量反映到代码中：

- Phong 模型 BRDF 实现不包括环境光项
- 由于 Phong 模型中 I/r^2 项包含在光源采样步骤中，因此 BRDF 中不包含 I/r^2。
- 平衡系数 k_d、k_s、高光衰减系数 p、法线方向 n 已知
- 光照方向（从物体表面看向光源）l 即为入射光的反方向
- 光照反射方向 l_r 和视点观察方向 v 均为出射光方向

（2）确定步骤

- 转换坐标系到局部坐标
- 根据公式计算漫反射的贡献 K_d 和环境光的贡献 K_s
- 返回 K_d+K_s

（3）代码

```
virtual Spectrum f(const Vector3f &wo, const Vector3f &wi) const override {
  Spectrum diffuse{0.f};
  Spectrum specular{0.f};
  // TODO
  // 1. 转换坐标系到局部坐标
  Vector3f win=toLocal(wi);
  Vector3f wout=toLocal(wo);

  // 2. 根据公式计算 K_d, K_s
  Vector3f l=win*(-1.0);//光照方向（入射光的反方向）
  diffuse=kd*dot(normal,win);
  specular=ks*pow(dot(wout,wout),p);//RGB的赋值：可以用另一个RGB来赋值，也可以用float来赋值

  // 3. return K_d + K_s
  return diffuse + specular;
  // tips:
  // Phong模型brdf实现不包括环境光项；其I/r^2项包含在光源采样步骤中，因此brdf中不包含I/r^2。

}
```

1.2 结果验证

耗时 6.90s 得到渲染结果：

```
pro@prodeMacBook-Pro-2 bin % ./Moer /Users/pro/Desktop/Rendering_labs/Moer-lite/examples/lab2-test/phong
Using acceleration type embree
100% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
Rendering costs 6.90s
```

1.3 分析不同参数的影响

- kd 和 ks 分别是漫反射系数和镜面反射系数，取值范围在[0,1]之间，p 是高光衰减系数，取值范围为[1,100]

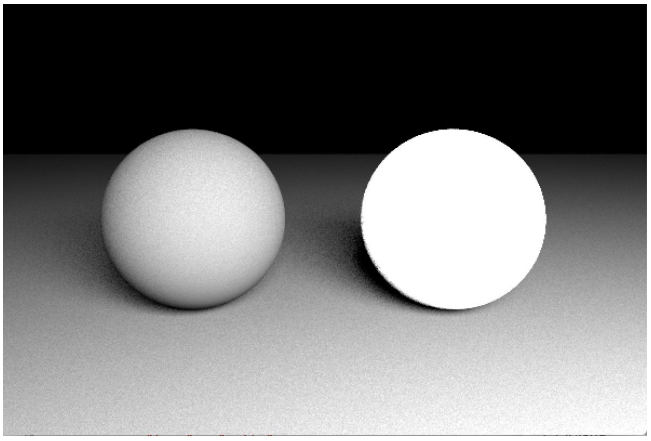
• 采用控制变量法分别探究三个参数对渲染结果的影响

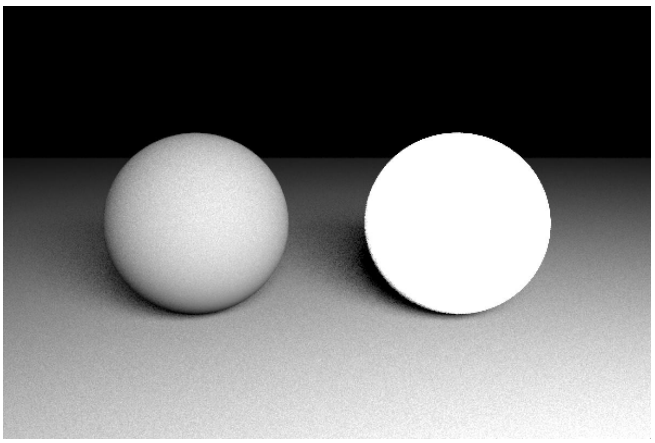
（1）kd 参数的影响

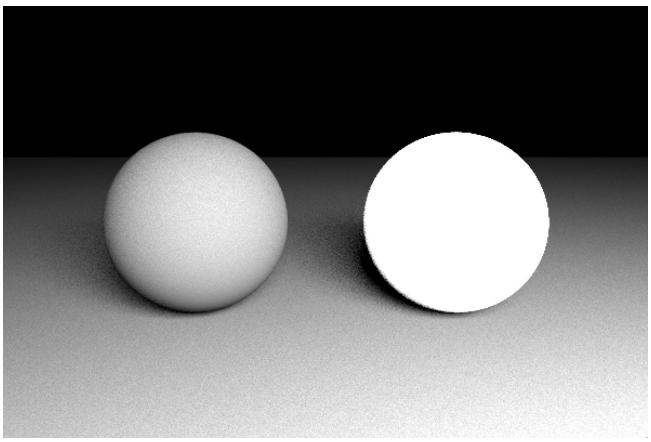
修改模型参数分别为：

```
"kd" : [0, 0, 0],"ks" : [1.0, 1.0, 1.0],"p" : 64
"kd" : [0.5, 0.5, 0.5],"ks" : [1.0, 1.0, 1.0],"p" : 64
"kd" : [1.0, 1.0, 1.0],"ks" : [1.0, 1.0, 1.0],"p" : 64
```

得到以下三张渲染结果：







对比可以得出结论：

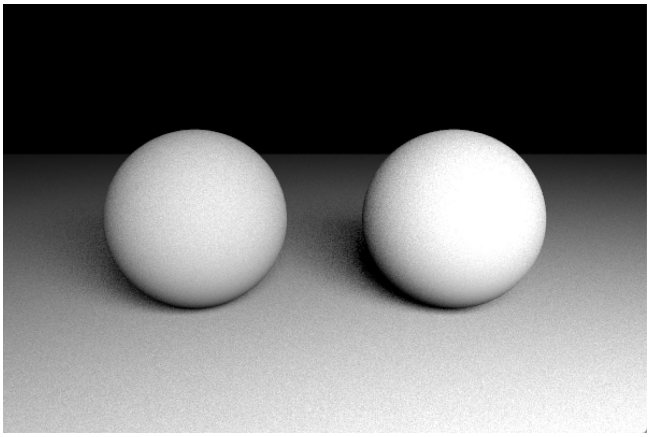
kd 越大，物体表面对漫反射光的反射越强，物体表面颜色越明亮。

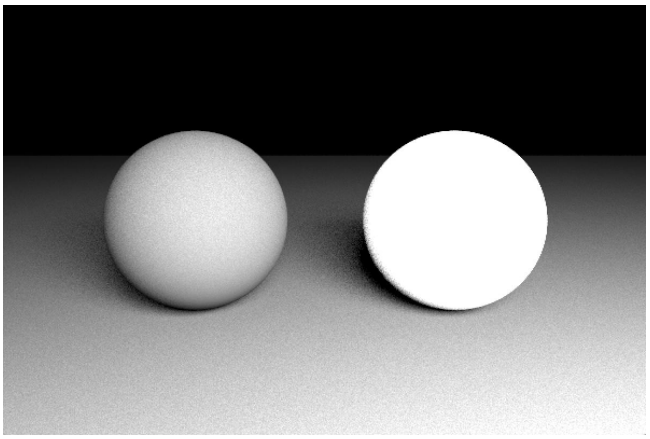
（2）ks 参数的影响

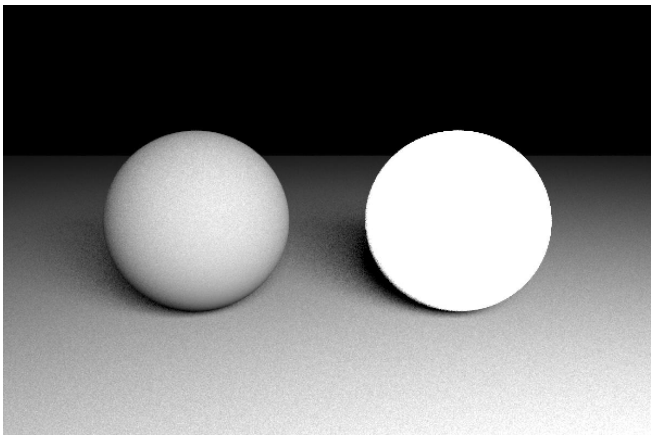
修改模型参数分别为：

```
"kd" : [0.3, 0.3, 0.3],"ks" : [0, 0, 0],"p" : 64
"kd" : [0.3, 0.3, 0.3],"ks" : [0.5, 0.5, 0.5],"p" : 64
"kd" : [0.3, 0.3, 0.3],"ks" : [1.0, 1.0, 1.0],"p" : 64
```

得到以下三张渲染结果：







对比可以得出结论：

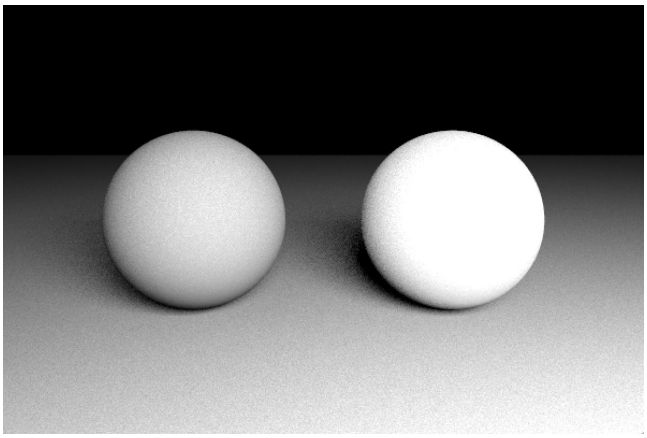
ks 越大，物体表面对镜面反射光的反射越强，物体表面的高光越明显.

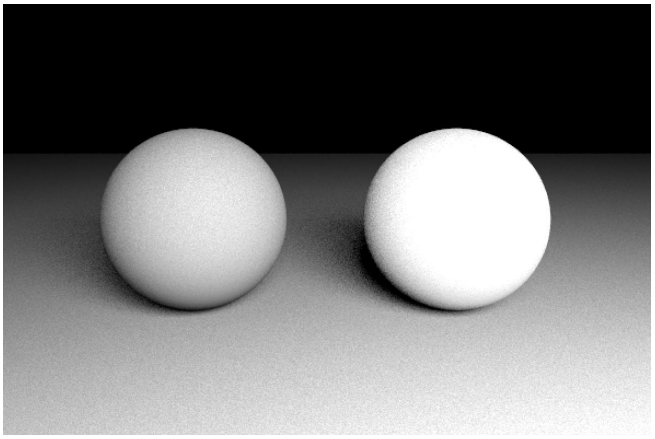
(3) p 参数的影响

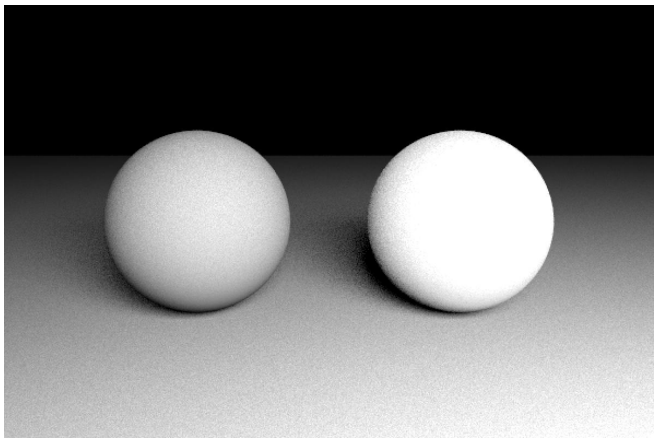
修改模型参数分别为：

```
"kd" : [0.3, 0.3, 0.3],"ks" : [0.1, 0.1, 0.1],"p" : 1
"kd" : [0.3, 0.3, 0.3],"ks" : [0.1, 0.1, 0.1],"p" : 50
"kd" : [0.3, 0.3, 0.3],"ks" : [0.1, 0.1, 0.1],"p" : 100
```

得到以下三张渲染结果：







对比可以得出结论：

p 越大，物体表面的高光越尖锐，反之，高光越模糊。

（4）总结

漫反射系数 kd：kd 越大，物体表面对漫反射光的反射越强，物体表面颜色越明亮。

镜面反射系数 ks：ks 越大，物体表面对镜面反射光的反射越强，物体表面的高光越明显。

反射高光系数 p：p 越大，物体表面的高光越尖锐，反之，高光越模糊。

1.4 思考题

请回答：什么情况下，原始的 Phong 模型（公式 1）会产生错误的结果？

由于 phong 模型是基于经验公式而不是严格的物理模型得出的，因此原始的 Phong 模型在处理较大粗糙度（即高光区域模糊）的物体表面时会产生错误的结果。同时，Phong 模型也不能很好地处理镜面反射和折射，因为它只考虑了漫反射和镜面高光，而没有考虑折射和次表面散射等其他现象。

二、 基于物理的模型

1. Oran-Nayar 模型

1.1 补全代码

（1）根据计算公式

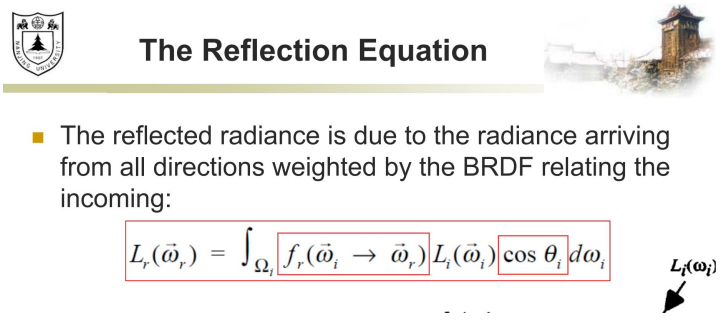
$$f_r(\boldsymbol{\omega}_o, \boldsymbol{\omega}_i) = \frac{\rho}{\pi}(A + B\max(0, \cos(\phi_i - \phi_o))\sin\alpha\tan\beta) \tag{2}$$

$$A = 1 - \frac{\sigma^2}{2(\sigma^2 + 0.33)}, B = \frac{0.45\sigma^2}{\sigma^2 + 0.09}, \alpha = \max(\theta_o, \theta_i), \beta = \min(\theta_o, \theta_i) \tag{3}$$

将各个变量反映到代码中：

- ρ即代码中的 albedo
- ϕ, θ 均为向量在球坐标系下的坐标，在代码中，球坐标系里 y 轴向上
- σ 为粗糙度系数，代码中已给出

- 代码中的光线方向均默认由物体向外
- 代码中的局部法线方向均默认为 y 轴方向
- f()函数最后返回的是 brdf*cos



（4）确定步骤

- 转换坐标系到局部坐标
- 根据公式计算 A, B, \alpha, \beta（可以直接求\sin\alpha,\tan\beta）, \cos(\phi_i-\phi_o)
- brdf*cos

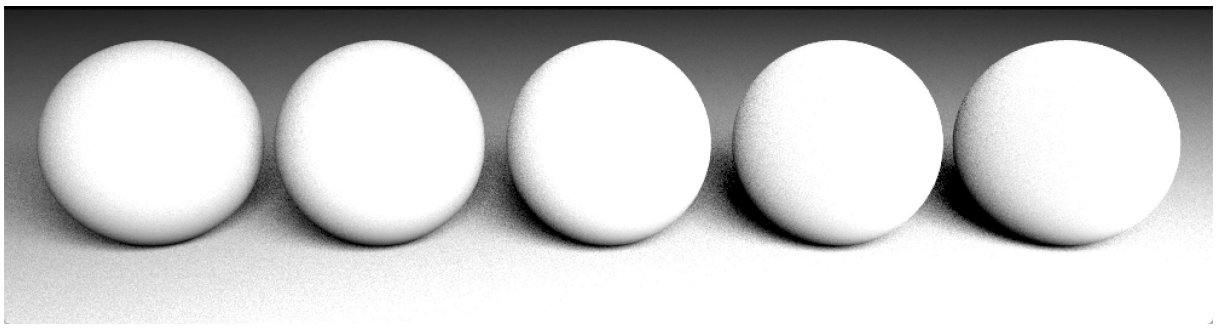
（5）代码

```cpp
class OrenNayarBSDF : public BSDF {
public:
  OrenNayarBSDF(const Vector3f &_normal, const Vector3f &_tangent,
                const Vector3f &_bitangent, Spectrum _albedo, float _sigma)
      : BSDF(_normal, _tangent, _bitangent), albedo(_albedo), sigma(_sigma) {}

  virtual Spectrum f(const Vector3f &wo, const Vector3f &wi) const override {
    Spectrum diffuse{0.f};
    // TODO
    // 1. 转换坐标系到局部坐标
    Vector3f win=toLocal(wi);//这里表示光照方向，是入射方向的反方向
    Vector3f wout=toLocal(wo);

    // 2. 计算 A, B, \alpha, \beta (可以直接求\sin\alpha,\tan\beta) , \cos(\phi_i-\phi_o)
    float A=1-(pow(sigma,2)/(2*(pow(sigma,2)+0.33)));
    float B=(0.45*pow(sigma,2))/(pow(sigma,2)+0.09);
    float cosine=0.f,sinAlpha=0.f,tanBeta=0.f;

    float thetaI=acos(dot(normalize(win),Vector3f(0,1,0)));
    float thetaO=acos(dot(normalize(wout),Vector3f(0,1,0)));
    if(thetaI>thetaO) {sinAlpha=sin(thetaI); tanBeta=tan(thetaO);}
    else {sinAlpha=sin(thetaO); tanBeta=tan(thetaI);}

    float phiI=atan2(win[2],win[0]);//球坐标，y轴向上
    float phiO=atan2(wout[2],wout[0]);
    cosine=cos(phiI-phiO);
    if(cosine<0) cosine=0;


    // 3. return Oren-Nayar brdf
    diffuse=albedo*INV_PI*(A+B*cosine*sinAlpha*tanBeta);
    return diffuse*cos(thetaI);//注意最后要乘以cos
  }
```

1.2 结果验证

耗时 11.15s 得到渲染结果：

```
pro@prodeMacBook-Pro-2 bin % ./Moer /Users/pro/Desktop/Rendering_labs/Moer-lite/
examples/lab2-test/oren-nayar
Using acceleration type embree
100% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
Rendering costs 11.15s
```



2. Torrance-Sparrow 模型

2.1 补全代码

（1）根据计算公式

$$f_r(p,\boldsymbol{\omega}_o,\boldsymbol{\omega}_i)=\frac{D(\boldsymbol{\omega}_h)G(\boldsymbol{\omega}_o,\boldsymbol{\omega}_i)F_r(\boldsymbol{\omega}_o)}{4(\boldsymbol{n}\cdot\boldsymbol{\omega}_o)(\boldsymbol{n}\cdot\boldsymbol{\omega}_i)} \tag{4}$$

将各个变量反映到代码中：

• n 为该表面的宏观法线方向,

• ωh 或者写为 m 为微观表面法线(也是光线入射出射方向的半程向量)

• D 为材质表面的法线分布函数，G 为材质表面的几何衰减项，Fr 为菲涅尔反射项

(6) 确定步骤

• 转换坐标系到局部坐标

• 计算 D(ωh)

• Beckmann-Spizzichino 模型

$$D_b(\boldsymbol{\omega}_h) = \frac{e^{\frac{-\tan^2\theta}{\alpha^2}}}{\pi\alpha^2\cos^4\theta}$$

参数 α 决定了函数分布的幅度，由表面的粗糙度决定；θ 为微观法线与宏观法线的夹角。

• GGX 模型

$$D_{tr}(\boldsymbol{\omega}_h) = \frac{\alpha^2}{\pi\cos^4\theta(\alpha^2+\tan^2\theta)^2}$$

• 计算 G(ωo, ωi)

• Beckmann-Spizzichino 模型

$$G_1^b(\boldsymbol{\omega}) \approx \begin{cases} \frac{3.535a+2.181a^2}{1+2.276a+2.577a^2}, & a < 1.6 \\ 1, & otherwise \end{cases}$$

• GGX 模型

$$G_1^{tr}(\boldsymbol{\omega}) = \frac{2}{1+\sqrt{1+\alpha^2\tan^2\theta_v}}$$

其中，$a = \frac{1}{\alpha\tan\theta_v}$，θv 是光线方向与宏观法线的夹角。

• 计算 Fr(ωo)

导体（Conductor）和电介质（Dielectric）材质具有不同的菲涅尔效应，其对应的菲涅尔方程也不同

• 导体材质：直接调用框架给出的菲涅尔函数

• 电介质材质：调用框架给出的菲涅尔函数，不考虑多重介质，如果光线从真空射入介质，其 eta 即配置中填写的 eta；如果光线从介质射出，则 eta = 1/eta

(7) 代码

• Conductor 材质各向同性的微表面模型

```cpp
class RoughConductorBSDF : public BSDF {
public:
  RoughConductorBSDF(const Vector3f &_normal, const Vector3f &_tangent,
                     const Vector3f &_bitangent, Spectrum _albedo,
                     Vector2f _alpha, Vector3f _eta, Vector3f _k, NDF *_ndf)
      : BSDF(_normal, _tangent, _bitangent), albedo(_albedo), alpha(_alpha),
        eta(_eta), k(_k), ndf(_ndf) {}

  virtual Spectrum f(const Vector3f &wo, const Vector3f &wi) const override {
    Spectrum specular{0.f};
    // TODO
    // 1. 转换坐标系到局部坐标
    Vector3f wout=toLocal(wo);
    Vector3f win=toLocal(wi);//入射光改为从外向交点


    // 2. 根据公式计算 Fr, D, G
    float D,G;
    Vector3f wh=(wout+win)/(wout+win).length();//微观表面法线
    D=ndf->getD(wh,alpha);
    G=ndf->getG(wout,win,alpha);

    //计算Fr
    float thetaI=acos(dot(normalize(win),Vector3f(0,1,0)));
    float thetaO=acos(dot(normalize(wout),Vector3f(0,1,0)));
    Vector3f Fr=getFr(thetaO);
    // 3. return albedo * D * G * Fr / (4 * \cos\theta_o);
    // tips: brdf
    // 中分母的\cos\theta_i项被渲染方程中的cos项消去，因此分母只有4*\cos\theta_o
    specular=albedo*D*G*Fr/(4*cos(thetaO));
    return specular;
  }
```

• Dieletric 材质各向同性的微表面模型

```
class RoughDielectricBSDF : public BSDF {
public:
  RoughDielectricBSDF(const Vector3f &_normal, const Vector3f &_tangent,
                      const Vector3f &_bitangent, Spectrum _albedo,
                      Vector2f _alpha, float _eta, NDF *_ndf)
      : BSDF(_normal, _tangent, _bitangent), albedo(_albedo), alpha(_alpha),
        eta(_eta), ndf(_ndf) {}

  virtual Spectrum f(const Vector3f &wo, const Vector3f &wi) const override {
    Spectrum specular{0.f};
    // TODO
    // 1. 转换坐标系到局部坐标
    Vector3f wout=toLocal(wo);
    Vector3f win=toLocal(wi);

    // 2. 根据公式计算 Fr, D, G
    float D,G;
    Vector3f wh=(wout+win)/(wout+win).length();//微观表面法线
    D=ndf->getD(wh,alpha);
    G=ndf->getG(wout,win,alpha);

    //计算Fr
    float thetaI=acos(dot(normalize(win),Vector3f(0,1,0)));
    float thetaO=acos(dot(normalize(wout),Vector3f(0,1,0)));
    Vector3f Fr;
    if(eta>1) Fr=getFr(eta,thetaO);//从真空射入介质//? 判断标准存疑
    else Fr=getFr(1/eta,thetaO);
    // 3. return albedo * D * G * Fr / (4 * \cos\theta_o);
    // tips:
    // 不考虑多重介质，如果光线从真空射入介质，其eta即配置中填写的eta;
    // 如果光线从介质射出，则eta = 1/eta
    specular=albedo*D*G*Fr/(4*cos(thetaO));
    return specular;
  }
```

• Beckmann 模型

```
class BeckmannDistribution : public NDF {
public:
  BeckmannDistribution() noexcept = default;
  virtual ~BeckmannDistribution() noexcept = default;
  virtual float getD(const Vector3f &whLocal,
                     const Vector2f &alpha) const noexcept override {
    // TODO
    // 根据公式即可
    //宏观法线就是(0,1,0)
    float theta=acos(dot(normalize(whLocal),Vector3f(0,1,0)));
    return INV_PI*exp((-1)*pow(tan(theta),2)/dot(alpha,alpha))/(dot(alpha,alpha)*pow(cos(the
    //return 0.f;
  }
  // tips:
  // float getG1(...) {}
  virtual float getG(const Vector3f &woLocal, const Vector3f &wiLocal,
                     const Vector2f &alpha) const noexcept override {
    // TODO
    // 根据公式即可
    // tips: return getG1(wo) * getG1(wi);
    float gi=1,go=1;
    //Vector3f wh=(woLocal+wiLocal)/(woLocal+wiLocal).length();//微观表面法线
    float thetaI=acos(dot(Vector3f(0,1,0),normalize(wiLocal)));
    float thetaO=acos(dot(Vector3f(0,1,0),normalize(woLocal)));
    float ai=1/(alpha.len()*tan(thetaI));//? 这里alpha.len()存疑
    float ao=1/(alpha.len()*tan(thetaO));
    if(ai<1.6) gi=(3.535*ai+2.181*ai*ai)/(1+2.276*ai+2.577*ai*ai);
    if(ao<1.6) go=(3.535*ao+2.181*ao*ao)/(1+2.276*ao+2.577*ao*ao);

    return gi*go;
  }
```

• GXX 模型

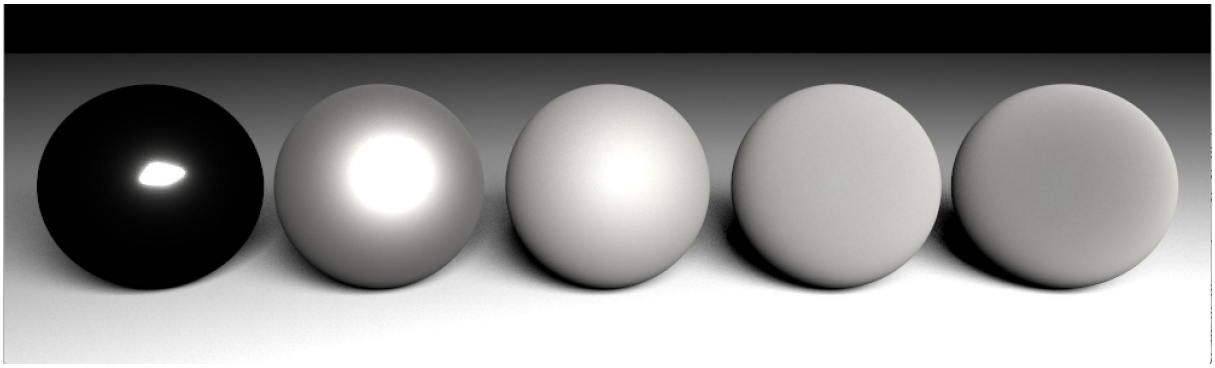
```
class GGXDistribution : public NDF {
public:
  GGXDistribution() noexcept = default;
  virtual ~GGXDistribution() noexcept = default;
  virtual float getD(const Vector3f &whLocal,
                     const Vector2f &alpha) const noexcept override {
    // TODO
    // 根据公式即可
    float theta=acos(dot(normalize(whLocal),Vector3f(0,1,0)));
    return INV_PI*dot(alpha,alpha)/(pow(cos(theta),4)*pow(dot(alpha,alpha)+tan(theta)*tan(th

    //return 0.f;
  }
  // tips:
  // float getG1(...) {}
  virtual float getG(const Vector3f &woLocal, const Vector3f &wiLocal,
                     const Vector2f &alpha) const noexcept override {
    // TODO
    // 根据公式即可
    // tips: return getG1(wo) * getG1(wi);
    float gi=1,go=1;
    //Vector3f wh=(woLocal+wiLocal)/(woLocal+wiLocal).length();//微观表面法线
    float thetaI=acos(dot(Vector3f(0,1,0),normalize(wiLocal)));
    float thetaO=acos(dot(Vector3f(0,1,0),normalize(woLocal)));

    gi=2/(1+sqrt(1+dot(alpha,alpha)*pow(tan(thetaI),2)));
    go=2/(1+sqrt(1+dot(alpha,alpha)*pow(tan(thetaO),2)));

    return gi*go;
    //return 0.f;
  }
```

2.2 结果验证

• conductor-gxx 模型

耗时 190.95s 得到渲染结果：

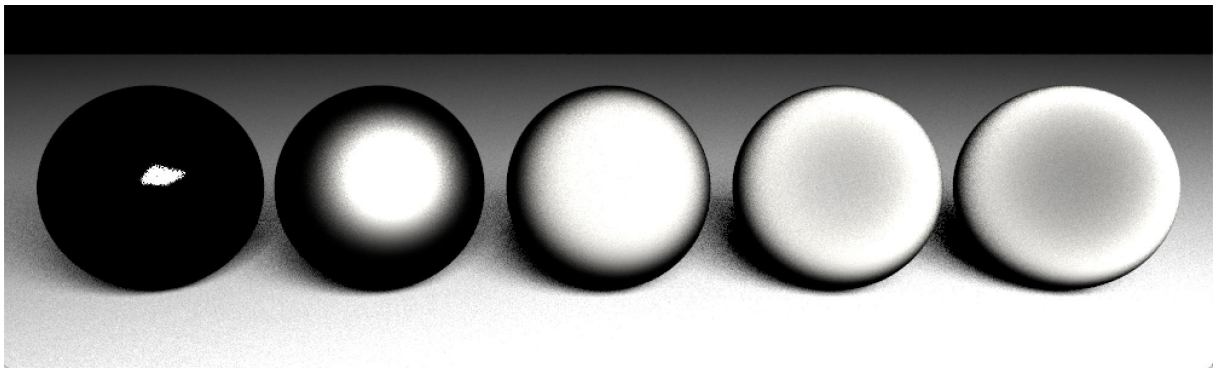
```
pro@prodeMacBook-Pro-2 bin % ./Moer /Users/pro/Desktop/Rendering_labs/Moer-lite/examples/lab2-test/conductor-ggx
Using acceleration type embree
[100% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
Rendering costs 190.95s
```



• conductor-beckmann 模型

耗时 12.43s 得到渲染结果：

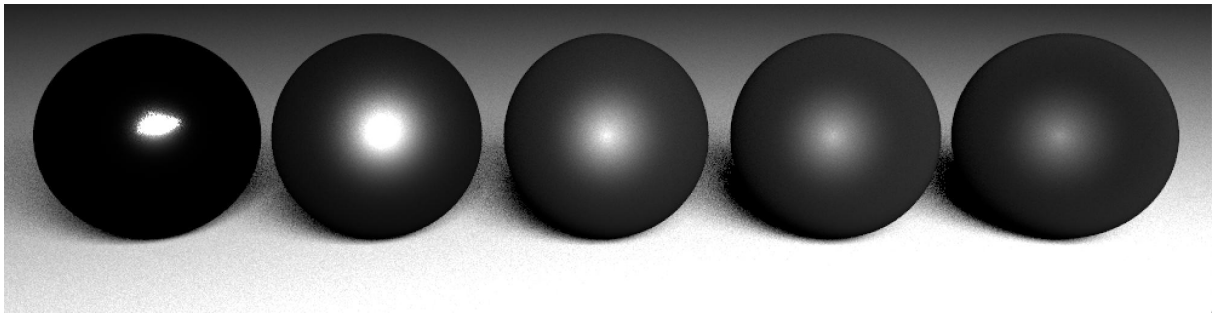
```
pro@prodeMacBook-Pro-2 bin % ./Moer /Users/pro/Desktop/Rendering_labs/Moer-lite/examples/lab2-test/conductor-beckmann
Using acceleration type embree
100% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
Rendering costs 12.43s
```



• dieletric-gxx 模型

耗时 12.05s 得到渲染结果：

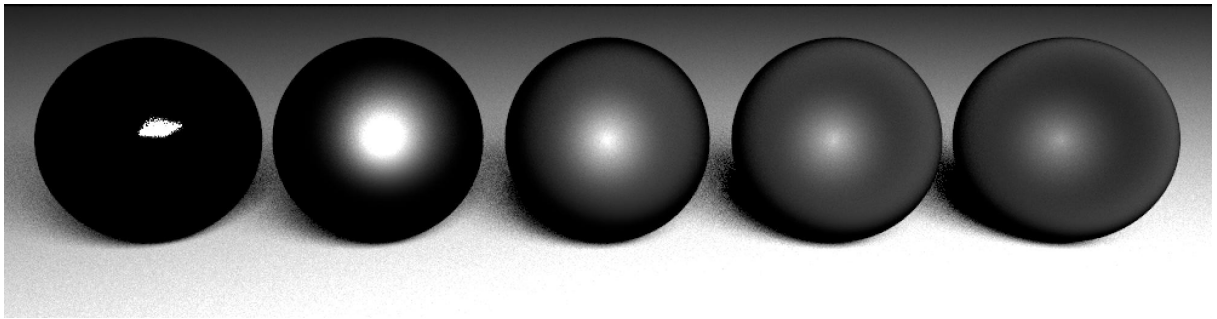
```
pro@prodeMacBook-Pro-2 bin % ./Moer /Users/pro/Desktop/Rendering_labs/Moer-lite/examples/lab2-test/dielectric-ggx
Using acceleration type embree
100% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
Rendering costs 12.05s
```



• dieletric-beckmann 模型

耗时 11.96s 得到渲染结果：

```
pro@prodeMacBook-Pro-2 bin % ./Moer /Users/pro/Desktop/Rendering_labs/Moer-lite/examples/lab2-test/dielectric-beckmann
Using acceleration type embree
100% [|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||]
Rendering costs 11.96s
```



2.3 分析选用不同模型的影响

• 使用 Beckmann 模型会产生非常锐利的高光和阴影。而在 GGX 模型中，产生的高光和阴影比较平滑。

• 原因：在 Beckmann 模型中，NDF 的尾部非常重，因此会产生非常锐利的高光和阴影。而在 GGX 模型中，尾部被削弱，因此产生的高光和阴影比较平滑。

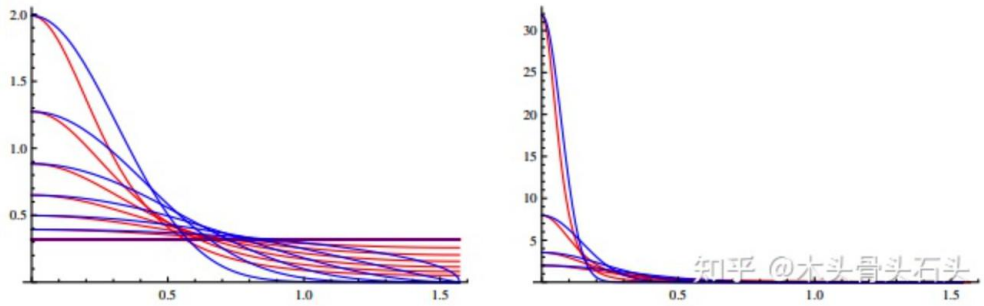
【参考图形渲染基础：微表面材质模型】

1.3 Trowbridge-Reitz 分布

Trowbridge-Reitz 分布又称为 GGX 分布，其形式为：

$$D_{tr} = \frac{\alpha_{tr}^2}{\pi((\vec{n}\cdot\vec{h})^2(\alpha_{tr}^2-1)+1)^2}$$

参数 α_{tr} 与 α_b 功能类似，随着 α_{tr} 的增加，表面越粗糙。Trowbridge-Reitz 与 Beckmann 分布类似，也能表示超级粗糙的表面，但相比于 Beckmann 和 Phong，GGX 分布有一大特点，如下图：



其中，蓝色曲线表示 Phong，红色曲线表示 GGX。两者形状，大致相同，但 GGX 的峰值更窄，同时在 $\vec{h}$ 与 $\vec{n}$ 夹角接近 90 度时，函数值不为 0，这意味着 GGX 的高光有更长的“拖尾”。

上图中，左图来自金属铬的真实图片，中间是 GGX，右边是 Beckmann。GGX 因为这一特性，在工业界得到广泛应用。

三、 总结

1. 遇到的问题

（1）渲染时间过长

确保是 release 版本。debug 版本性能较差，使用 ccmake 修改生成版本，确保使用 cmake 生成的是 release 版本。

（2）Oren-Nayar 模型渲染出的结果全是白色没有细节

函数 f()的返回值其实是 brdf*cos

四、 参考博客

[1] 基于物理的 BSDF：Microfacet Model（微表面模型）
[2] 图形渲染基础：微表面材质模型