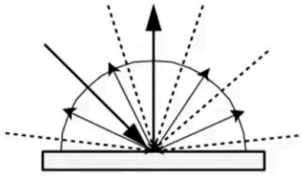
BRDF describe the Materials

diffuse

Diffuse / Lambertian Material

Light is equally reflected in each output direction



$f_r = c$

Suppose the incident lighting is uniform:

$$L_o(\omega_o) = \int_{H^2} f_r L_i(\omega_i) \cos\theta_i \, \mathrm{d}\omega_i$$
$$= f_r L_i \int_{H^2} \cancel{(\omega_i)} \cos\theta_i \, \mathrm{d}\omega_i$$
$$= \pi f_r L_i$$

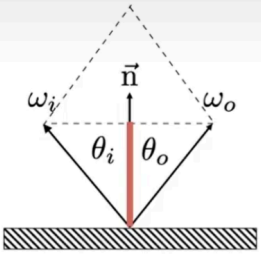
$$f_r = \frac{\rho}{\pi}$$ — albedo (color)

漫反射材质假设入射光是均匀的。

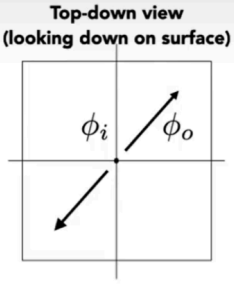
提到了一些其它材质，如 Glossy materials（不怎么光滑的抛光金属），refactive materials（有色玻璃）

reflect

Perfect Specular Reflection



$\theta = \theta_o = \theta_i$



$\phi_o = (\phi_i + \pi) \bmod 2\pi$

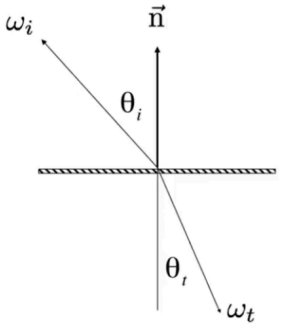
$$\omega_o + \omega_i = 2\cos\theta\,\vec{n} = 2(\omega_i \cdot \vec{n})\vec{n}$$

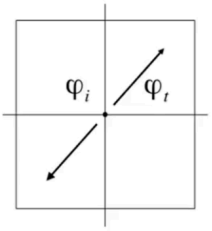
$$\omega_o = -\omega_i + 2(\omega_i \cdot \vec{n})\vec{n}$$

refract

Snell's Law

Transmitted angle depends on
- index of refraction (IOR) for incident ray
- index of refraction (IOR) for exiting ray





$\varphi_t = \varphi_i \pm \pi$

Medium	η^*
Vacuum	1.0
Air (sea level)	1.00029
Water (20°C)	1.333
Glass	1.5-1.6
Diamond	2.42

* index of refraction is wavelength dependent (these are averages)

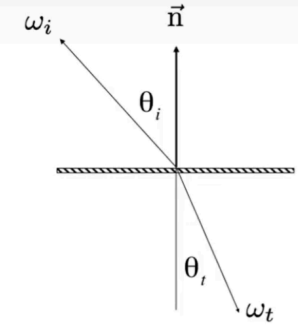
$\eta_i \sin\theta_i = \eta_t \sin\theta_t$

全反射现象：

Law of Refraction

全反射
全内反射
并且入射角比较大的情况——全反射临界角

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t$$



$$\cos\theta_t = \sqrt{1-\sin^2\theta_t}$$
$$= \sqrt{1-\left(\frac{\eta_i}{\eta_t}\right)^2 \sin^2\theta_i}$$
$$= \sqrt{1-\left(\frac{\eta_i}{\eta_t}\right)^2 (1-\cos^2\theta_i)}$$

Total internal reflection:

$$1-\left(\frac{\eta_i}{\eta_t}\right)^2 (1-\cos^2\theta_i) < 0$$

When light is moving from a more optically dense medium to a less optically dense medium: $\frac{\eta_i}{\eta_t} > 1$

Light incident on boundary from large enough angle will not exit medium.

Fresnel Term

解释入射角度对反射折射率的影响。

应用更广的是一种近似方法。

Accurate: need to consider polarization

$$R_{\mathrm{s}} = \left| \frac{n_1 \cos\theta_{\mathrm{i}} - n_2 \cos\theta_{\mathrm{t}}}{n_1 \cos\theta_{\mathrm{i}} + n_2 \cos\theta_{\mathrm{t}}} \right|^2 = \left| \frac{n_1 \cos\theta_{\mathrm{i}} - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin\theta_{\mathrm{i}}\right)^2}}{n_1 \cos\theta_{\mathrm{i}} + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin\theta_{\mathrm{i}}\right)^2}} \right|^2,$$

$$R_{\mathrm{p}} = \left| \frac{n_1 \cos\theta_{\mathrm{t}} - n_2 \cos\theta_{\mathrm{i}}}{n_1 \cos\theta_{\mathrm{t}} + n_2 \cos\theta_{\mathrm{i}}} \right|^2 = \left| \frac{n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin\theta_{\mathrm{i}}\right)^2} - n_2 \cos\theta_{\mathrm{i}}}{n_1 \sqrt{1 - \left(\frac{n_1}{n_2} \sin\theta_{\mathrm{i}}\right)^2} + n_2 \cos\theta_{\mathrm{i}}} \right|^2.$$

$$R_{\mathrm{eff}} = \frac{1}{2}\left(R_{\mathrm{s}} + R_{\mathrm{p}}\right).$$

Approximate: Schlick's approximation

$$R(\theta) = R_0 + (1 - R_0)(1 - \cos\theta)^5$$

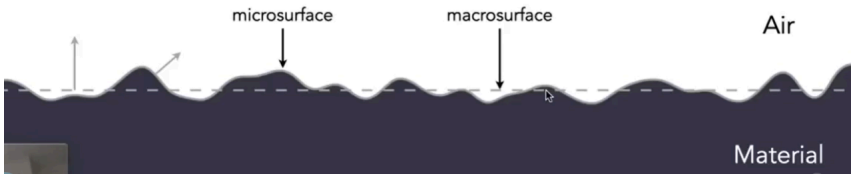
$$R_0 = \left(\frac{n_1 - n_2}{n_1 + n_2}\right)^2$$

Microfacet Materials 微表面材质

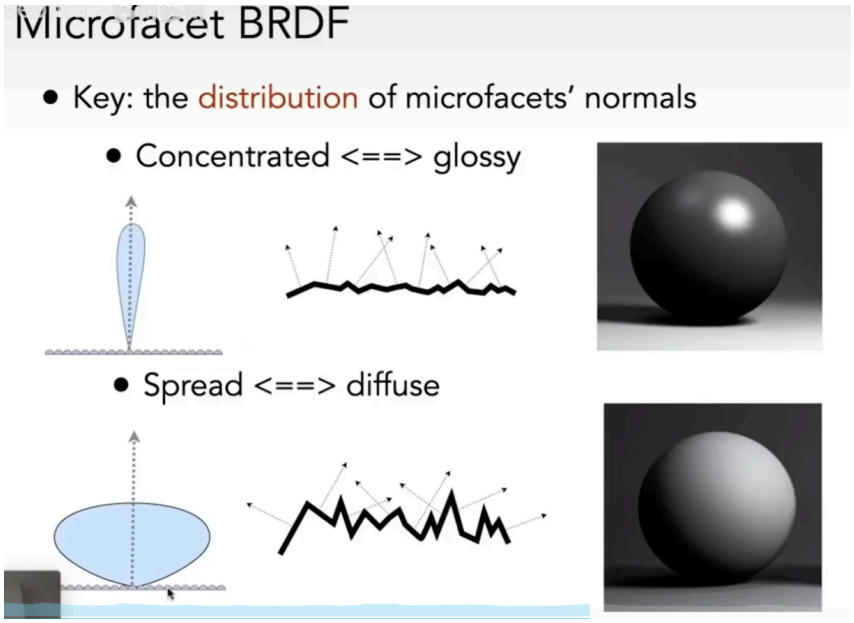
Microfacet Theory

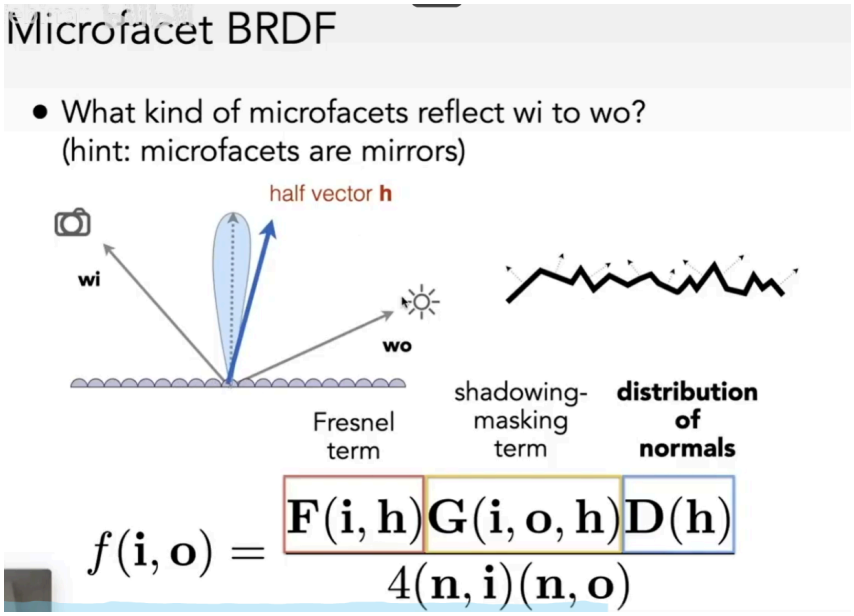
Rough surface

- Macroscale: flat & rough
- Microscale: bumpy & specular



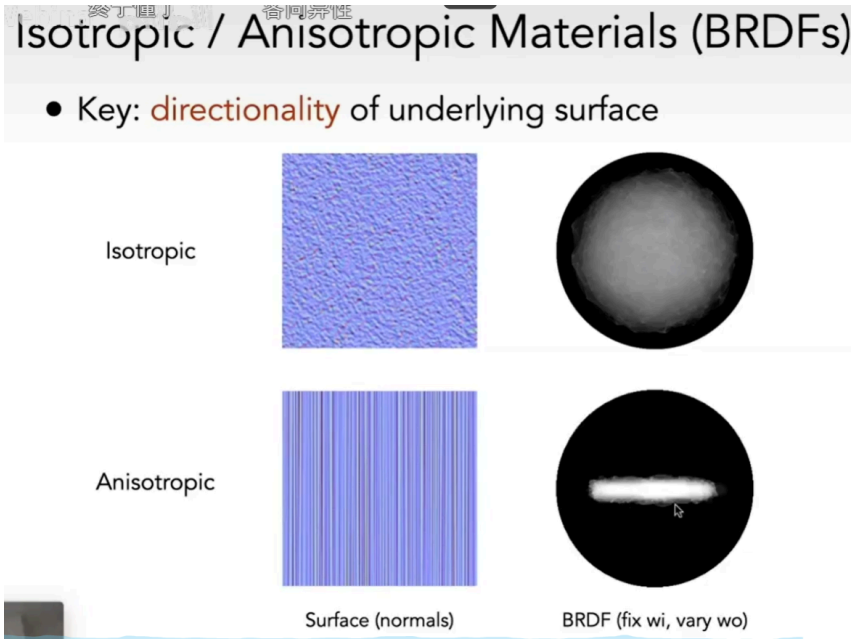
具体研究时，是以统计的视角研究微镜面曲面的法线在宏观上的分布





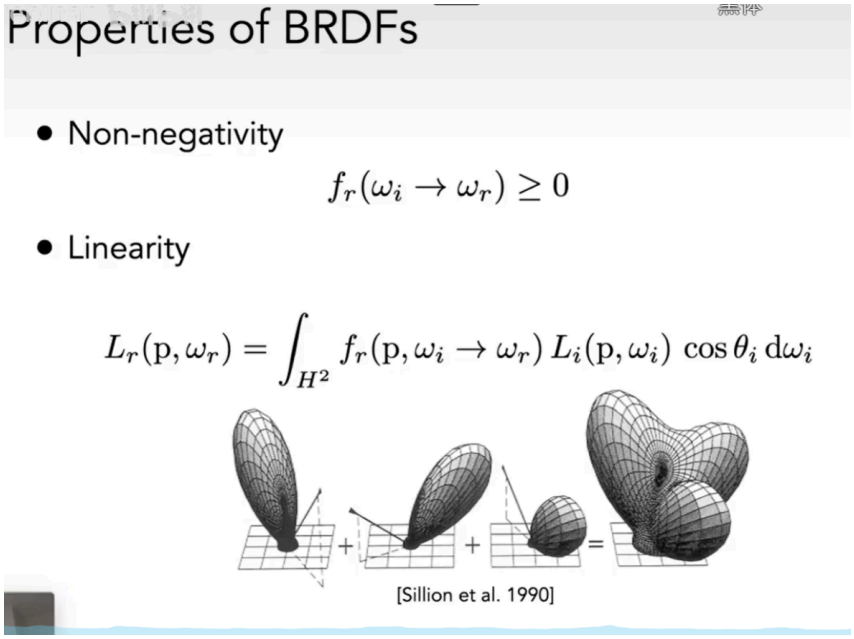
`shadow-masking term` 是考虑微表面之间的相互遮挡，所造成的光线吸收。

实际上，不管是 $\overrightarrow{w_i}$ 还是 $\overrightarrow{w_o}$，只要有其一与宏观表面之间的夹角太小，就很有可能被微表面之间的互相遮挡影响。



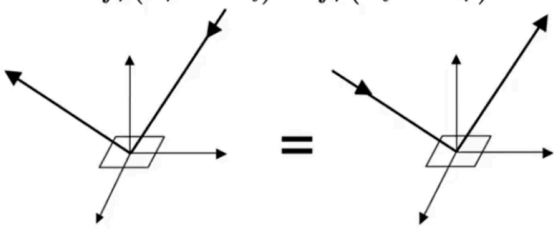
另外，以这种法线分布的形式，微表面模型也可以解释一些各向同性和各向异性的现象。

BRDF 的性质



- Reciprocity principle

$$f_r(\omega_r \to \omega_i) = f_r(\omega_i \to \omega_r)$$

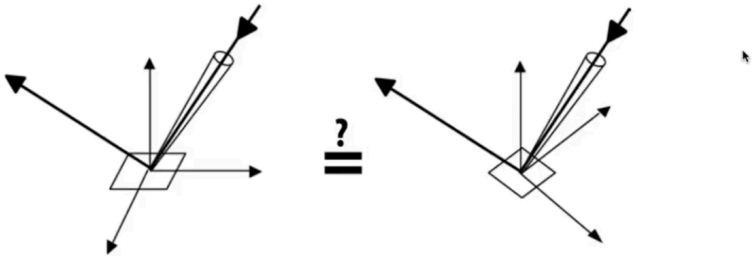


- Energy conservation

$$\forall \omega_r \int_{H^2} f_r(\omega_i \to \omega_r) \cos\theta_i \, \mathrm{d}\omega_i \leq 1$$

- Isotropic vs. anisotropic
  - If isotropic, $f_r(\theta_i, \phi_i; \theta_r, \phi_r) = f_r(\theta_i, \theta_r, \phi_r - \phi_i)$
  - Then, from reciprocity,

$$f_r(\theta_i, \theta_r, \phi_r - \phi_i) = f_r(\theta_r, \theta_i, \phi_i - \phi_r) = f_r(\theta_i, \theta_r, |\phi_r - \phi_i|)$$



下一节 Advanced Topics 基本就是晚上躺板板听听了，直接做作业 7。

光栅化的入口仍然是在 Render.cpp，修改 spp 参数能减少噪点。

对于每层 path tracing 的递归，都对所有光源采样一次。

话说在 scene.hpp 函数里看到了 fresnel 函数，不过我就不实现微表面材质了。

然后看一下三角形.hpp 里实现的采样，这个函数直接影响我们调用 sampleLight() 的方式

```
void Sample(Intersection &pos, float &pdf){
    float x = std::sqrt(get_random_float()), y = get_random_float();
    pos.coords = v0 * (1.0f - x) + v1 * (x * (1.0f - y)) + v2 * (x * y);
    pos.normal = this->normal;
    pdf = 1.0f / area;
}
```

hasemit 就是是否是光源的意思。

所以调用 sampleLight 的方式其实很简单：

```
Intersection inter;
float pdf_light;
sampleLight(inter, pdf_light);
```

这里采样出的其实是某个光源的某片三角形上的某个点。

下面这个函数没有用到，但实际上应该添加进 sampleLight 里去。

```
Vector3f Material::getEmission() {return m_emission;}
```

不过 Intersection 里也有 material 指针了，所以直接在三角形的 Sample 函数里添一句 `pos.m = m;` 就行了。嗯，不过我还是直接用的 `pos.emit = m->getEmission()`，这样似乎可能大概会快一点，毕竟 Material 类看上去还是挺大的。

注意他写的 eval 函数，只用到了 wo，没用 wi，毕竟就是这两次课提到的 diffuse 的 BRDF，这种 BRDF 的出射光也是均匀的，所以只与入射光有关。(我还为这个烦恼了一阵子，一看函数，呵呵)

编不下去了。

好吧我得承认作业的制定者性格十分恶劣，伪代码是用来误导人的。除非你更改 castRay() 函数的结构，或者就得换一个视角来实现它。

伪代码的视角是一个出射光的视角（这里的光默认都是电眼逼人哈)，但 castRay 是入射光的视角，这里的区别在于，伪代码有出射光的法线，而 castRay 没有入射光的法线。

又得重构了，狗东西。

从我们的视角思考，sampleLight 是得放在 sample 之后的。

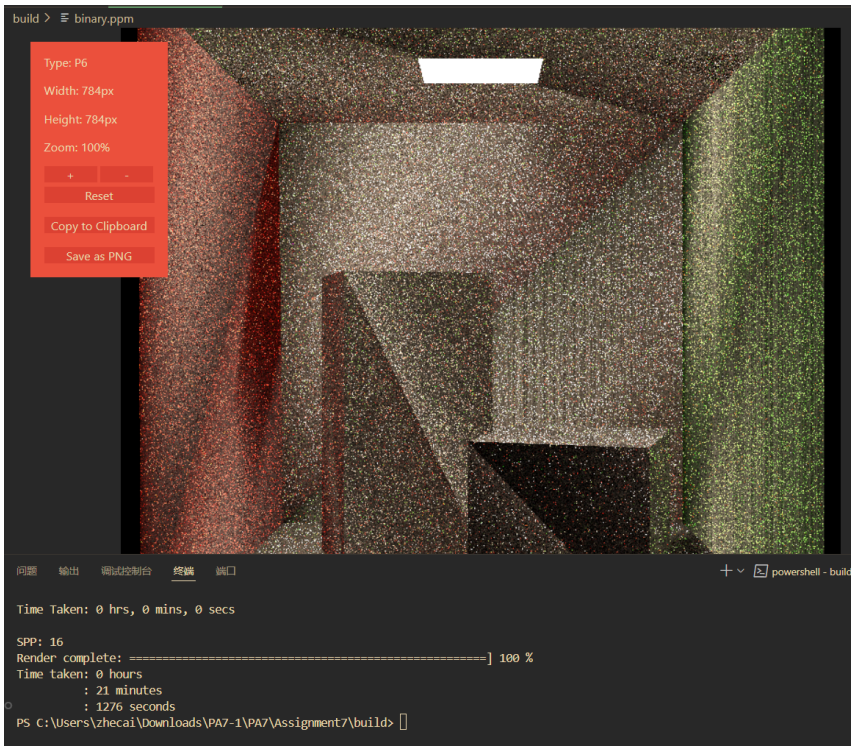
(这里 Material::pdf 在 diffuse 材质的实现也只是与 wo 有关)

不过看 Material 里面函数的调用方式，还是得在三角形.sample 里加上 `pos.m = m;`

```
Vector3f Scene::castRay(const Ray &ray, int depth) const
{
    Vector3f Ldir, Lindir;
    Intersection scene_inter = intersect(ray);
    if(scene_inter.happened == false) return Lindir;// 0

    if(scene_inter.m->hasEmission())
        return scene_inter.m->getEmission();
    else
    {
        Intersection inter_light;
        float pdf_light;
        sampleLight(inter_light, pdf_light);

        Vector3f w_light = normalize(inter_light.coords - scene_inter.coords);
        Ray to_light(scene_inter.coords, w_light);
        if(intersect(to_light).distance >= (inter_light.coords -
scene_inter.coords).norm()) {
            float div = (inter_light.coords - scene_inter.coords).norm();
            Ldir = inter_light.emit * inter_light.m->eval(w_light, w_light,
scene_inter.normal) * dotProduct(w_light, scene_inter.normal)
                * dotProduct(-w_light, inter_light.normal) / div / div / pdf_light;
        }

        // Russia
        std::random_device rd;
        // 使用 Mersenne Twister 引擎
        std::mt19937 gen(rd());
        // 定义分布 - 0.0到1.0之间的均匀分布
        std::uniform_real_distribution<> dis(0.0, 1.0);
```

```
        // 生成随机数并比较
        if(dis(gen) < RussianRoulette)
        {
            Vector3f wo = normalize(scene_inter.m->sample(ray.direction,
scene_inter.normal));
            float pdf = scene_inter.m->pdf(ray.direction, wo, scene_inter.normal);
            Lindir = castRay(Ray(scene_inter.coords, wo), depth + 1) * scene_inter.m-
>eval(wo, wo, scene_inter.normal)
                * dotProduct(wo, scene_inter.normal) / pdf / RussianRoulette;
        }


        return Ldir + Lindir;
    }

    // TO DO Implement Path Tracing Algorithm here
    // Intersection inter;
    // float pdf_light;
    // Vector3f emission;
    // sampleLight(inter, pdf_light);
    // //Vector3f dir = normalize(Vector3f(-x, y, 1));
    // Vector3f w = normalize(inter.coords - ray.origin);
    // if(scene_inter.distance >= (inter.coords - ray.origin).norm()) {
    //     Ldir = inter.emit * Material::eval(w, w, )
    // }
}
```

21 分钟，我勒个。不过结果挺怪的?



于是我把 SPP 减小到 2，然后把俄罗斯轮盘赌的概率减小（具体地，只有 0.2 的概率继续下一层递归），以求快速查错，这个数据下一分钟左右可以出结果。

改各种边界条件根本没啥用，我得先观察。、算了，我放弃了。以后再寻机缘。