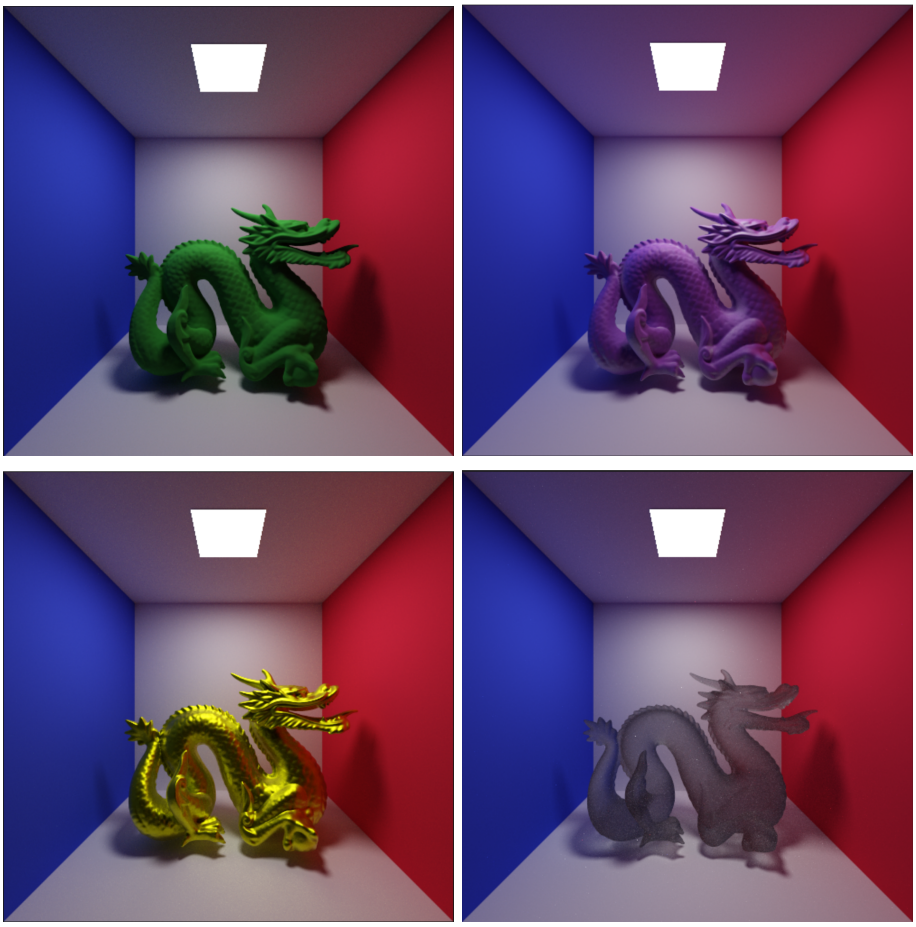
PBRT 系列 12-代码实战-微表面材质

Dezeming Family

2021 年 3 月 17 日

因为本书是电子书，所以会不断进行更新和再版（更新频率会很高）。如果您从其他地方得到了这本书，可以从官方网站：https://dezeming.top/ 下载新的版本（免费下载）。

本书目标：学习多种材质的基本概念，微表面模型的基本概念。移植和测试成功微表面模型。跟踪内存。检查内存释放问题，保证没有内存泄漏。



本文于 2022 年 7 月 15 日进行再版，提供了源码。注意图形 GUI 界面和本文中展示的有点区别，但并不影响学习。源码见网址 [https://github.com/feimos32/PBRT3-DezemingFamily]。

前言

本书其实是一个可选项，没有什么理论知识的讲解，纯粹为了好玩（为了渲染出好看的场景）。在本书我们研究介于完美镜面和纯漫反射表面之间的材质。

有了路径追踪之后，理论上可以渲染出任何场景了，为了更好看的效果，因此本节我们研究材质。

微表面的理论其实比较复杂，但一般来说我们只要移植别人的材质实现，就能渲染出来，因此我只介绍基本原理和实现过程，至于里面的一些分布和细节，大家可以参考 [5]，等我以后有时间会专门写一个关于材质和采样的书（目前规划可能该文章得很长，就放在专业知识理论部分了）。

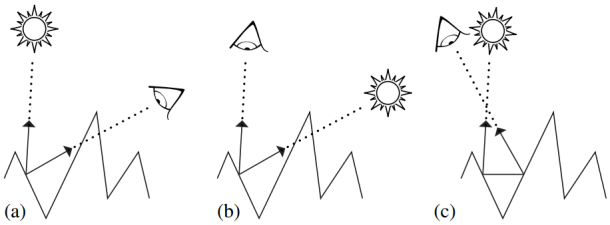
我之前主要是做基于物理的医学体数据渲染的，有时候物理材质没有那么吹毛求疵，甚至用混合了 Phong 模型和一切其他物理模型的渲染都可以调试出非常逼真的图像。我对于初学者的建议就是，有公式就用公式。但是基本概念需要学会，比如在 BxDF 类中，f 函数，Sample_f 函数，Pdf 函数，Sample_L 函数等的意义，这些函数如果明白了是什么意义，表面和材质其实无非就是那么点事儿，无非就是不同的基于能量守恒的模型罢了。先把渲染流程玩熟练，之后剩下的知识体系慢慢补充即可。

本书的售价是 4 元（电子版），但是并不直接收取费用。如果您免费得到了这本书的电子版，在学习和实现时觉得有用，可以往我们的支付宝账户（17853140351，可备注：PBRT）进行支持，您的赞助将是我们 Dezeming Family 继续创作各种图形学、机器学习、以及数学原理小册子的动力！

目录

一　Oren-Nayar 漫反射材质

微表面材质模拟了微小区域内的表面的朝向的分布。微表面中可能会发生三种相互关系：



如上图，其中 (a) 表示的是遮挡关系，有的微表面看不到；(b) 表示的是阴影关系，光无法到达微表面；(c) 表示的是互反射，光经过多次反弹到达人眼。

基于微表面的 BRDF 模型通过对大量微表面的光散射进行统计建模来工作。如果我们假设被照亮的微分面积 dA 与单个微表面的大小相比相对较大（比如 dA 中包含了成千上万个微表面），那么在单位区域中就会有大量的微表面被照亮，单位区域所有的微表面的散射效果之和决定了观察到的散射。

微表面模型的两个主要组成部分是表面分布的表示和描述光如何从单个微表面散射的 BRDF。考虑到这些，微表面模型需要导出一个闭式的表达式，给出描述这种表面散射的 BRDF。完美镜面反射最常用于微表面 BRDF，尽管镜面透射对于许多半透明材料的建模非常有用，而 Oren-Nayar 模型将微表面视为朗伯反射器。

1 1　Oren-Nayar 模型

Oren 和 Nayar 观察到真实世界中的模型并不是完美的 lambertian 反射，例如当照明方向接近视角方向时，粗糙表面会更亮。他们开发了一个反射模型，用 V 形微表面描述粗糙表面，V 形微表面由一个单一参数 σ（微表面方位角的标准差）的球形高斯分布描述。

在 V 形假设下，仅考虑相邻微表面即可解释互反射（即前面图中描述中的 (c)）。Oren 和 Nayar 利用这一点推导了 BRDF，该模型模拟了凹槽集合的总反射。该模型考虑了微表面之间的阴影、掩蔽和相互反射，没有一个闭式的解决方案，因此他们发现一个近似值非常适合，可在书 [1] 的第 8 章第三节查询。

1 2　Oren-Nayar 模型的移植和测试

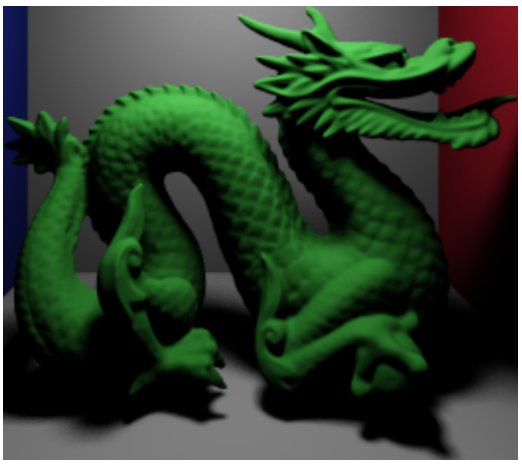
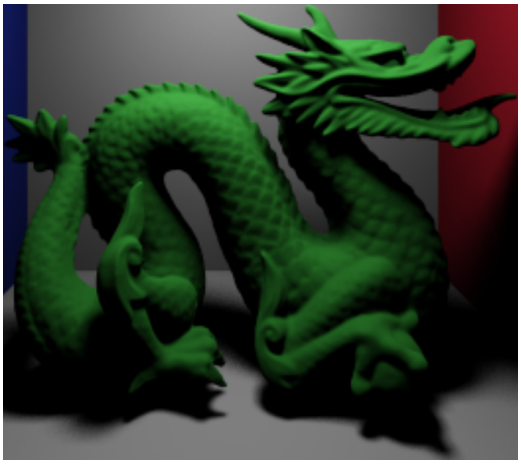
我们首先移植函数的主体，没有什么需要修改的地方。构造函数中，sigma 首先要转化为 $[0, 2\pi]$ 的弧度。

我们采样 BRDF 的 Sample_f 函数继承自父类，即先根据 Cos 比例产生 wi 方向，然后调用 f 函数返回光在 wi 方向的分布。而 f 函数完全是根据公式来实现的，返回通过 BRDF 计算的光分布。

在 MatteMaterial 类的 ComputeScatteringFunctions 函数中：

```
1 if (sig == 0)
2     si->bsdf->Add(new LambertianReflection(r));
3 else
4     si->bsdf->Add(new OrenNayar(r, sig));
```

测试结果如下，左边是 lambertian 模型，右边是 Oren-Nayar 模型，σ 设置为了 60.0f，除此之外其他参数完全相同。

还是能看出一点区别来的。至于哪个更真实反正我是看不出来。

二　微表面模型理论

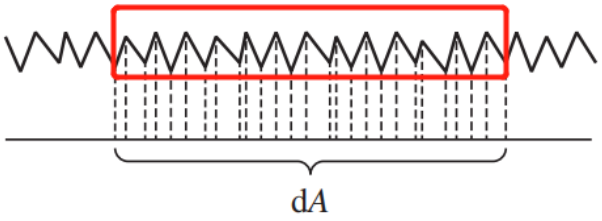
微表面主要用来模拟光泽材料（glossy materials），例如金属，塑料或者磨砂玻璃。

所有微表面分布都定义在 core 文件夹下的 microfacet.h 和 microfacet.cpp 文件中。MicrofacetDistribution 是所有微表面材质的基类。之后，在 reflection.h 文件中定义的 MicrofacetReflection 类则会使用这些分布进行后续计算。

我们先把 microfacet 文件里的内容移植到自己的系统中，几乎不需要什么修改。

2 1　微表面法向量分布

微表面的分布函数 $D(\omega_h)$ 表示微区域中表面法向量 ω_h 的分布。对于完美的粗糙表面来说，微区域的法向量分布都在表面法向量的位置（着色空间中的 z 轴）。为了符合物理，表面分布函数必须要单位化，即半球上分布积分为 1。直观地说，如果我们考虑沿法线方向 n 入射到微表面上的光线，那么每条光线必须恰好与微表面相交一次。更正式地说，给定微表面的微分面积 dA，则该面积上方的微表面投影面积必须等于 dA：



用公式表示为：

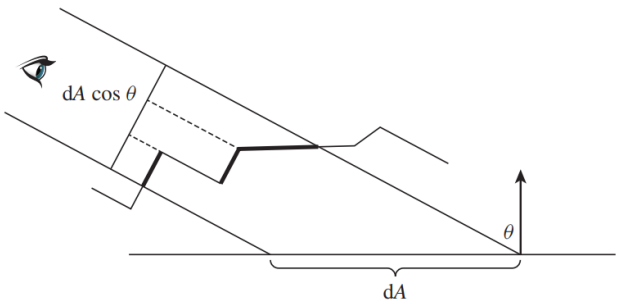
$$\int_{\Omega} D(\omega_h) cos\theta_h d\omega_h = 1 \tag{二.1}$$

注意单位化微表面分布的一个常见错误是在立体角上进行积分，而不是在投影立体角上进行积分（即省略 $cos\theta_h$ 项）。

D 函数的作用就是给个方向，求法向量在该方向的分布概率值。不同的微表面模型使用了不同的 D 分布，有时候在要求实时的游戏领域中就直接使用高斯分布即可。我们不赘述这些不同的 D 分布和计算，基本原理并不困难，直接移植和使用即可。

2 2　遮挡与阴影

单靠微表面法线的分布不足以完全描述微表面特性。同样重要的是要考虑到这样一个事实，即一些微表面在给定的观察或照明方向上是不可见的（其他的微表面挡住了它们）。这种相互遮挡的效果，被称为 masking-shadowing function，用符号 G_1 表示当微表面法向量方向在 ω_h 时，从 ω 方向看时微表面被遮挡的比率。



从观察者或光源看，表面上的微分区域具有面积 $dAcos\theta$，其中 $cos\theta$ 是入射方向与表面法线的角度。可见微表面（粗线）的投影表面积也必须等于 $dAcos\theta$；遮蔽阴影函数 G_1 给出了在给定方向上可见的总微表面面积在 dA 上的比例分数。我们将表面分布与遮挡函数归一到渲染方程的 cos 项中：

$$cos\theta = \int_{\Omega} G_1(\omega, \omega_h) max(0, \omega \cdot \omega_h) D(\omega_h) d\omega_h \tag{二.2}$$

换句话说，给定方向 ω 的可见微表面投影面积必须等于 $cos\theta$ 乘以 dA 的微分面积。因为微表面都在一定的高度场（意思是说所有的微表面都是基于给定表面变化的，投影也是投影在同一个表面上，比如三角形上的微表面都是投影在该三角形所在的平面），因此在单位微表面 dA 上，前向投影和背向投影的面积完全一样。

具体的实现过程可以参考 [5]，其实大多数图形学教程都只是告诉你怎么定义和使用 G，而微表面模型的遮挡函数推导确实很麻烦。

在实际实现中使用 Λ 函数来构成 G_1 函数。同时对于光的入射方向，也要考虑遮挡阴影，因此我们得到：

$$G(\omega_o, \omega_i) = G_1(\omega_o) G_1(\omega_i) \tag{二.3}$$

其中：

$$G_1(\omega) = \frac{1}{1 + \Lambda(\omega)} \tag{二.4}$$

对于不同的 Λ 函数我们不做介绍，Lambda 函数就是为了实现它的。

2 3 Torrance-Sparrow 模型

该模型将法向量分布与遮挡相互结合，再加上菲涅尔项描述被反射的光线对比光线被折射的部分所占的比率，得到用来模拟金属反光的微表面模型：

$$L(\omega_o) = \frac{F_r(\omega_o) L_i(\omega_i) D(\omega_h) d\omega_i}{4 cos\theta_o} \tag{二.5}$$

这里的 ω_h 表示 $Normalize(\omega_i + \omega_o)$。

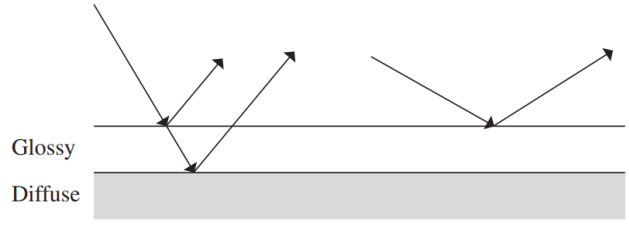
该反射模型即 MicrofacetReflection 类。

我们有了 BSDF，因此 f 函数就很容易了。但是 Sample_f 函数应该怎么采样和计算呢？其实是对 D 分布进行采样的（对法向量分布来采样）法向量，根据法向量生成反射方向，然后求其概率。Pdf 就是求该分布的概率。（具体的计算过程可以参考书 [1]）。

MicrofacetTransmission 同理，只是生成了折射方向而已。这里面都涉及判断是否在同一个半球范围，大家可以参考源码理解。

2 4 Fresnel 入射效应

许多 BRDF 模型没有考虑到菲涅耳反射会减少到达分层对象底部的光量。考虑一张抛光的木桌子或一堵涂有光泽油漆的墙：如果你正面看它们的表面，你主要看到的是木头或油漆的颜色。当您将视点移向一个扫掠角度时，由于菲涅尔效果导致的光泽反射增加，您看到的基础颜色则会较少。



由 Ashikhmin 和 Shirley 开发的 BRDF 模型，该模型分两层，下层面为在漫反射，上层具有光泽的镜面反射面。在考虑菲涅耳效应后，漫反射表面的反射效应被剩余的能量所调节。如上图，当入射方向接

近法线时，大部分光被传输到漫反射层，漫反射项占主导地位。当入射方向接近掠射时，光泽反射是主要的反射模式。

FresnelBlend 类就是这种反射模型，该类的输入也需要一种微表面分布。

该模型在采样 Sample_f 时也是如此，根据菲涅尔项判断是漫反射还是镜面反射，决定对 $cos\theta$ 采样还是对微表面分布进行采样。

2 5 移植

其实实现过程中有很多细节，比如微表面采样会使用 sampleVisibleArea 来判断是对整个分布进行采样还是根据可见微表面进行采样。鉴于展开讲会过于繁琐，我们就不再细讲了。微表面类继承自 BxDF，自然需要实现其方法，具体的原理可见 [1] 和 [5]。

把下面的类及该类的函数都移植成功，我们本章的任务就算完成了。

```
1 MicrofacetDistribution
2 BeckmannDistribution
3 TrowbridgeReitzDistribution
4 MicrofacetReflection
5 MicrofacetTransmission
6 FresnelBlend
```

三　不同材质的实现和测试

我们已经知道，不同的材质其实就是不同的 BxDF 进行组合。本章的任务就是实现多种材质，然后用路径追踪技术渲染出来。

以下的每个材质移植都非常简单，因此不再赘述。

3 1　塑料材质

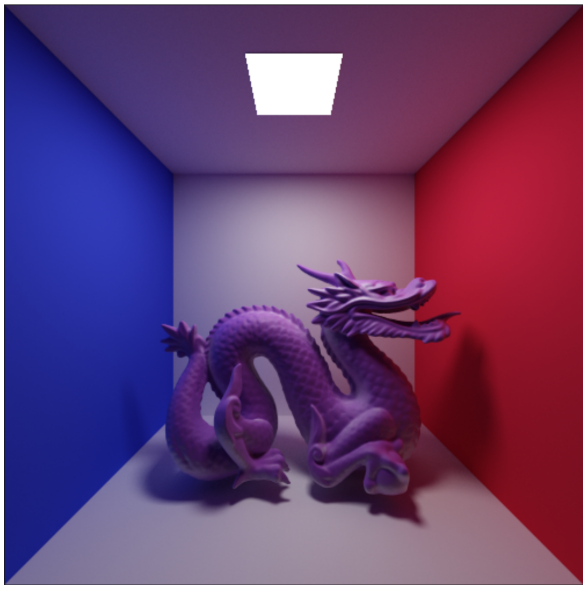
塑料材质是漫反射模型 LambertianReflection 和微表面模型 MicrofacetReflection 的组合体。

该类中粗糙度参数用于确定镜面反射高光的大小。它可以用两种方式指定：如果 remapRoughness 参数为真，那么给定的粗糙度应该在 0 到 1 之间变化。或者，如果参数为假，则粗糙度用于直接初始化微平面分布的参数。

我们实现塑料材质的参数如下：

```
inline std::shared_ptr<Material> getPurplePlasticMaterial() {
    Spectrum purple; purple[0] = 0.35; purple[1] = 0.12; purple[2] = 0.48;
    std::shared_ptr<Texture<Spectrum>> plasticKd = std::make_shared<
        ConstantTexture<Spectrum>>(purple);
    std::shared_ptr<Texture<Spectrum>> plasticKr = std::make_shared<
        ConstantTexture<Spectrum>>(Spectrum(1.f) - purple);
    std::shared_ptr<Texture<float>> plasticRoughness = std::make_shared<
        ConstantTexture<float>>(0.1f);
    std::shared_ptr<Texture<float>> bumpMap = std::make_shared<
        ConstantTexture<float>>(0.0f);
    return std::make_shared<PlasticMaterial>(plasticKd, plasticKr,
        plasticRoughness, bumpMap, true);
}
```

渲染得到如下结果：



3 2　金属材质

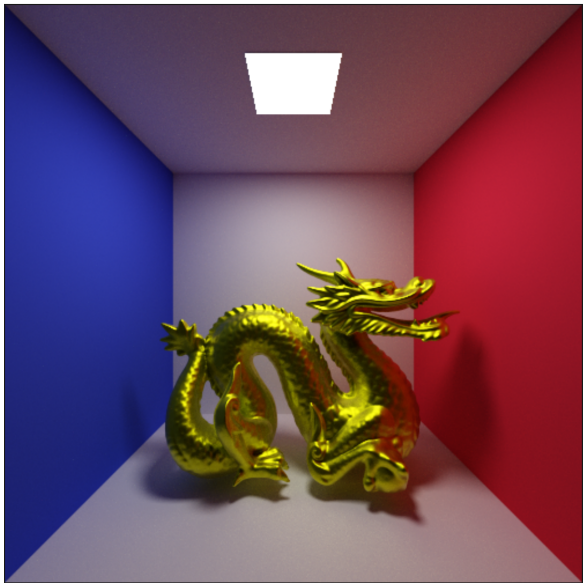
金属材质 MetalMaterial 是直接建模的微表面模型，使用 FresnelConductor 作为菲涅尔项，它是专门计算金属的菲涅尔表示的。

MetalMaterial 的构造函数中，eta 表示，k 表示吸收系数，用于构造 FresnelConductor。

我们实现金属材质的参数如下：

```
inline std::shared_ptr<Material> getYelloMetalMaterial() {
    Spectrum eta; eta[0] = 0.18f; eta[1] = 0.15f; eta[2] = 0.81f;
    std::shared_ptr<Texture<Spectrum>> etaM = std::make_shared<
        ConstantTexture<Spectrum>>(eta);
    Spectrum k; k[0] = 0.11f; k[1] = 0.11f; k[2] = 0.11f;
    std::shared_ptr<Texture<Spectrum>> kM = std::make_shared<ConstantTexture
        <Spectrum>>(k);
    std::shared_ptr<Texture<float>> Roughness = std::make_shared<
        ConstantTexture<float>>(0.2f);
    std::shared_ptr<Texture<float>> RoughnessU = std::make_shared<
        ConstantTexture<float>>(0.2f);
    std::shared_ptr<Texture<float>> RoughnessV = std::make_shared<
        ConstantTexture<float>>(0.2f);
    std::shared_ptr<Texture<float>> bumpMap = std::make_shared<
        ConstantTexture<float>>(0.0f);
    return       std::make_shared<MetalMaterial>(etaM, kM,
         Roughness, RoughnessU, RoughnessV, bumpMap, false);
}
```

渲染得到如下结果：



3 3 玻璃材质

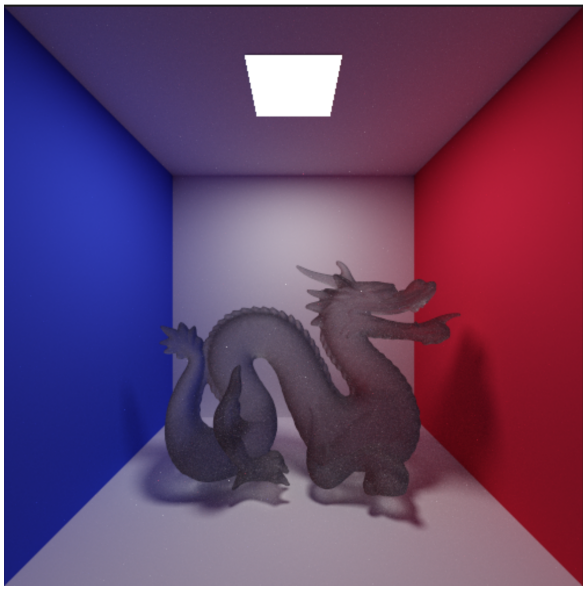
玻璃材质使用的是 MicrofacetTransmission 反射模型或完美镜面反射 SpecularTransmission。

我们实现的磨砂玻璃参数如下：

```
inline std::shared_ptr<Material> getWhiteGlassMaterial() {
    Spectrum c1; c1[0] = 0.98f; c1[1] = 0.98f; c1[2] = 0.98f;
    std::shared_ptr<Texture<Spectrum>> Kr = std::make_shared<ConstantTexture
        <Spectrum>>(c1);
    Spectrum c2; c2[0] = 0.98f; c2[1] = 0.98f; c2[2] = 0.98f;
    std::shared_ptr<Texture<Spectrum>> Kt = std::make_shared<ConstantTexture
        <Spectrum>>(c2);
    std::shared_ptr<Texture<float>> index = std::make_shared<ConstantTexture
```

```
        <float>>(1.5f);
    std::shared_ptr<Texture<float>> RoughnessU = std::make_shared<
        ConstantTexture<float>>(0.5f);
    std::shared_ptr<Texture<float>> RoughnessV = std::make_shared<
        ConstantTexture<float>>(0.5f);
    std::shared_ptr<Texture<float>> bumpMap = std::make_shared<
        ConstantTexture<float>>(0.0f);
    return      std::make_shared<GlassMaterial>(Kr, Kt,
            RoughnessU, RoughnessV, index, bumpMap, false);
}
```

渲染得到如下结果：



四　内存管理

对于比较光滑的材质而言，渲染需要多帧才能得到比较好的结果。我上面的图几乎都渲染了上千帧。如果您运行了几百帧就程序崩溃了，那说明您的内存没有释放，每帧都在进行积累。我们本章的内容就是把内存释放给处理好。

4 1　内存调试

我使用系统自带的内存检测程序，注意使用 psapi.h 文件之前要包含 windows.h。

```
#include <windows.h>
#include <psapi.h>
#pragma comment(lib, "psapi.lib")
void showMemoryInfo(void) {
    //  SIZE_T PeakWorkingSetSize; //峰值内存使用
    //  SIZE_T WorkingSetSize; //内存使用
    //  SIZE_T PagefileUsage; //虚拟内存使用
    //  SIZE_T PeakPagefileUsage; //峰值虚拟内存使用
    EmptyWorkingSet(GetCurrentProcess());

    HANDLE handle = GetCurrentProcess();
    PROCESS_MEMORY_COUNTERS pmc;
    GetProcessMemoryInfo(handle, &pmc, sizeof(pmc));
    //打印这些值，单位都是MB
    pmc.WorkingSetSize / 1000.f / 1000.f
    pmc.PeakWorkingSetSize / 1000.f / 1000.f
    pmc.PagefileUsage / 1000.f / 1000.f
    pmc.PeakPagefileUsage / 1000.f / 1000.f
}
```

把 showMemoryInfo 放在渲染的线程里，观察每次渲染的输出值，如果 PeakPagefileUsage 不会一直上升（物理内存会随时释放，但是没有释放的会被保存到虚拟内存中，因此峰值虚拟内存会不断上升），就说明我们的系统在渲染中没有发生内存泄漏。

4 2　内存管理建议

PBRT 使用的 MemoryArena 用来管理和统一释放局部内存区，我们为了方便管理内存，可以设置指针型数据为智能指针。

BSDF 类：

```
//成员方法
void Add(BxDF *b) {
            bxdfs.push_back(std::shared_ptr<BxDF>(b));
    //注意别忘了加1
            nBxDFs++;
    }
//成员变量
std::vector<std::shared_ptr<BxDF>> bxdfs;
```

MicrofacetReflection 等需要微表面分布和菲涅尔项的类：

```
1     //成员变量
2         const std::shared_ptr<MicrofacetDistribution> distribution;
3         const std::shared_ptr<Fresnel> fresnel;
```

将所有与 new 有关的内容都考虑使用智能指针，虽然效率会变慢，但是可以保证内存全都释放干净。如果峰值虚拟内存没有一直上升，那么你的系统就没有内存泄漏啦！

4 3 本书结语

本书总共写了 1 天。本想讲讲微表面材质，但是感觉各种分布的计算着实比较难讲，而且 PBRT 的实现考虑到性能和稳定性，用的方法也比较复杂，因此不再细讲。我们的目标只是移植和测试成功就可以了。

下本书把无限面光源的重要性采样和纹理都搞定，我们就能渲染出更好看的图像了。

参考文献

[1] Pharr M, Jakob W, Humphreys G. Physically based rendering: From theory to implementation[M]. Morgan Kaufmann, 2016.

[2] Shirley P. Ray Tracing in One Weekend[J]. 2016.

[3] Shirley P. Ray Tracing The Next Week[J]. 2016.

[4] Shirley P. Ray Tracing The Rest Of Your Life[J]. 2016.

[5] Heitz, E. . (2014). Understanding the Masking-Shadowing Function in Microfacet-Based BRDFs. journal of computer graphics techniques.