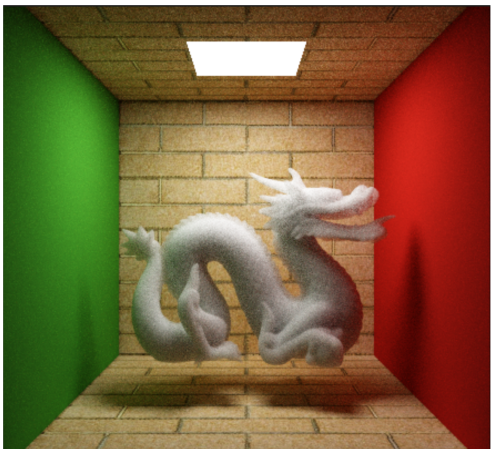
PBRT 系列 15-代码实战-参与介质与体渲染

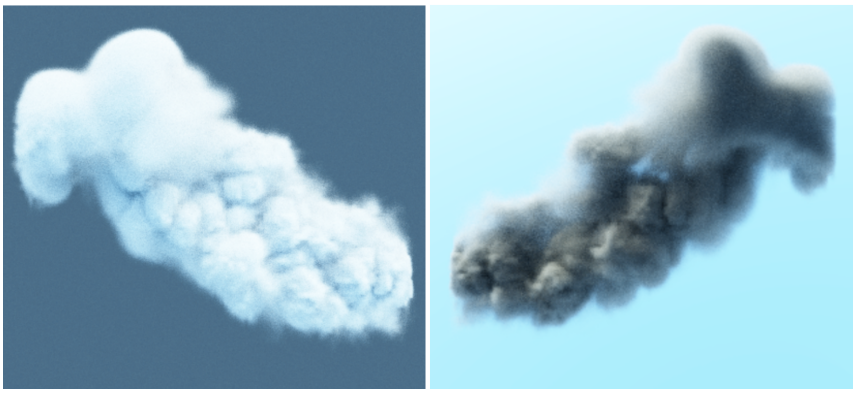
Dezeming Family

2021 年 4 月 2 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 PBRT 中的体渲染原理，实现两种不同的参与介质类型，渲染得到如下效果：





本文于 2022 年 7 月 17 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

参与介质其实并不是很复杂，但将它与系统整合在一起就显得有点麻烦了。我之前主要是做医学影像体渲染的，研究过很多复杂的体渲染可视化方法，但学习和移植 PBRT 的体渲染确实也是一个不小的挑战。

还记得之前我们在各个类中移除的参与介质么，我们现在就要把它们都加回来。还有一些组件也可能会在参与介质中，比如，相机可能在介质中，光源可能也在介质中，这些我们都是需要考虑的。

体渲染也是一个比较大的话题，我之前做了足足一年才把大部分细枝末节都搞定，这里很难讲清楚和明白，所以我只介绍 PBRT 的方法，至于基础原理大家可以专门找一些专著来看。由于需要移植的东西非常多，所以我会对所有移植的内容列一个清单，这样读者就能更容易地看到如果我们想支持渲染体材质，则需要扩充哪些内容和方法。

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　参与介质的基本定义

我们之前所有的场景都是描述的真空下的表面物质。而参与介质则表示的是 3D 空间分布的微小粒子，例如烟尘。光在通过参与介质时会发生散射，能量会被消耗。

我们将会描述影响光线通过参与介质的 radiance 的基本物理过程，介绍 media 基类，它提供了描述空间区域中参与介质的接口。介质实现返回关于在其范围内的点的散射特性的信息，包括相位函数，该函数描述了光在空间中某个点的散射特性。（它是 BSDF 的 volumetric 模拟，BSDF 描述了表面上某一点的散射，而相位函数描述的是体空间任意方向的散射概率）为了确定参与介质对场景中 radiance 分布的影响，需要处理体散射效果的积分器（后面章节再进行叙述）。

在积分器中寻找光路的成本通常与其长度成正比，跟踪具有数百或数千个散射相互作用的路径很快变得不切实际。在这种情况下，最好将基础散射过程的总体效果聚合在一个函数中，该函数将光进入和离开介质的点之间的散射联系起来。因此，最后 PBRT 书 [1] 中的体渲染的内容以 BSSRDF 基类结束，它是一个使实现这种方法成为可能的抽象。BSSRDF 描述了由折射表面限定的介质中的内部散射，我们这本书里不介绍 BSSRDF，放在后面的系列中。

1 1　体散射

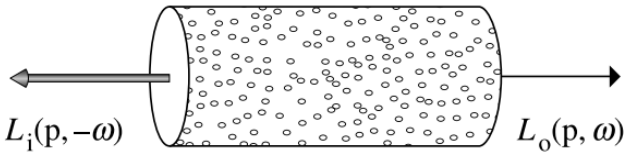
详细的体散射原理和蒙特卡洛方法已在《PBRT 系列 20——渲染概率与采样》进行了详细和完整的解释，这里的内容相对浅显和简单，可以做一个简单的回顾。

参与介质中有三个过程会影响 radiance 的分布：吸收：由于光转换成另一种形式的能量（如热）而引起的 radiance 减少；发射：发光粒子发出 radiance；散射：由于与粒子碰撞而向一个方向散射。

对于参与介质内部的特性，homogeneous 特性表示在空间中体空间中粒子在各处的特性都是相同的，inhomogeneous 表示空间各处粒子特性都是在变化的。

1 1.1　吸收

光以 ω 方向通过单位长度的参与介质以后，携带的能量表示如下：



$$L_o(p,\omega) - L_i(p.-\omega) = dL_o(p,\omega) = -\sigma_a(p,\omega)L_i(p,\omega)dt \tag{一.1}$$

光在 ω 方向通过距离 d 之后剩余的 radiance 占原来的比例为：

$$e^{-\int_0^d \sigma_a(p+t\omega,\omega)dt} \tag{一.2}$$

1 1.2　发射

粒子可能会发光，只考虑发射效应时，表示如下：

$$dL_o(p,\omega) = L_e(p,\omega)dt \tag{一.3}$$

1 1.3　外散射和衰减

光通过参与介质时，会有散射效应。描述有多少 radiance 能够沿着该方向继续前进表述为：

$$dL_o(p,\omega) = -\sigma_s(p,\omega)L_i(p,\omega)dt \tag{一.4}$$

因此总的光衰减系数为：

$$\sigma_t(p,\omega) = \sigma_a(p,\omega) + \sigma_s(p,\omega) \tag{一.5}$$

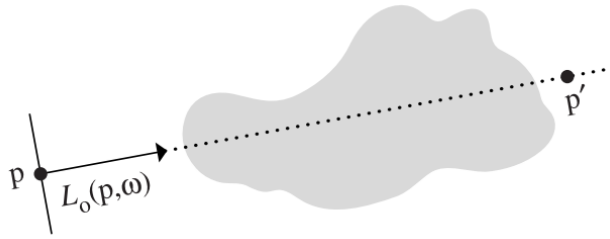
我们定义两个名词，一个是 albedo，直观理解为单位参与介质下，不发光的参与介质的颜色（表示为散射的概率）：

$$\rho = \frac{\sigma_s}{\sigma_t} \tag{一.6}$$

以及 mean free path（平均自由路径长度）：$1/\sigma_t$。

描述总的衰减效应的微分方程表示如下：

$$\frac{dL_o(p,\omega)}{dt} = -\sigma_t(p,\omega)L_i(p,-\omega) \tag{一.7}$$



从 p 点的 radiance，经过参与介质衰减后，到达 p' 点，radiance 变为：

$$T_r(p \to p')L_o(p,\omega) \tag{一.8}$$

$$T_r(p \to p') = e^{-\int_0^d \sigma_t(p+t\omega,\omega)dt} \tag{一.9}$$

要注意的是：

$$T_r(p \to p') = T_r(p' \to p) \tag{一.10}$$

$$T_r(p \to p'') = T_r(p \to p')T_r(p' \to p'') \tag{一.11}$$

T_r 的指数部分用两点间的光学厚度来表示：

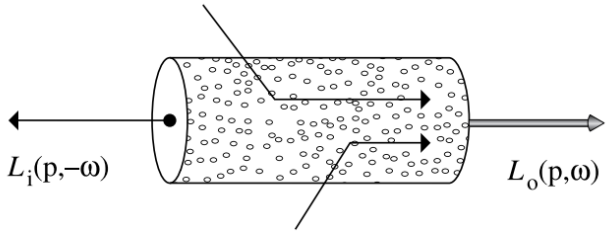
$$\tau(p \to p') = \int_0^d \sigma_t(p + t\omega, -\omega)dt \tag{一.12}$$

因此，当为 homogeneous 介质时，σ_t 是一个常量：

$$T_r(p \to p') = e^{-\sigma_t d} \tag{一.13}$$

1 1.4 内散射

外散射是向外发射 radiance，而内散射表示的是其他方向的 radiance 发射到当前 ω 方向：



关于发射和内散射的 radiance：

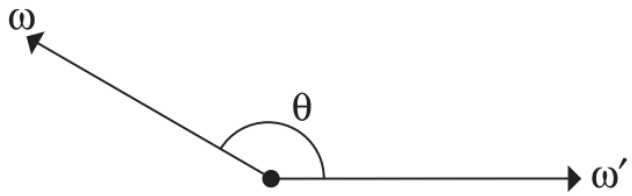
$$L_s(p,\omega) = L_e(p,\omega) + \sigma_s(p,\omega)\int_{S^2} p(p,\omega_i,\omega)L_i(p,\omega_i)d\omega_i \tag{一.14}$$

这里的 $p(p,\omega_i,\omega)$ 表示相位函数，注意它描述的是整个球面上散射到当前方向 ω 的 radiance。

1 2 相位函数

和 BRDF 函数一样，学者们也开发了很多相位函数。

isotropic 相位函数表示散射 radiance 只与夹角有关：



PhaseFunction 类是所有相位函数的基类，Henyey and Greenstein 相位函数模型是我们常用的模型，PhaseHG 就是计算该相位函数模型的函数，HenyeyGreenstein 派生自 PhaseFunction 类。

HenyeyGreenstein 中，p 函数就是返回的 PhaseHG 计算的相位函数值，Sample_p 函数是用来进行采样的（相当于 BSDF 函数中的 sample_f），采样到方向后调用 PhaseHG 计算采样方向的相位函数值，我们后面再描述。

1 3 Media 类

Medium 是所有参与介质的基类，该类有两个函数，一个是 Tr，即计算前面描述的 T_r，返回透射率，另一个是 Sample 函数来进行采样，我们后面再描述。

GeometricPrimitives 类里有 MediumInterface 类，该类有两个成员变量关于 Medium，分别是 inside 和 outside，顾名思义，表示该基元内部的参与介质和外部的参与介质。但是，有可能我们会设置成，比如物体的外部介质和相机所处的介质不同，这样计算出来的渲染值就是不对的。但 PRBT 中并不会进行检查，希望用户能够自己注意。IsMediumTransition 返回内外介质是否是同一种。

Ray 类中有 Medium 指针，指向 Ray 存在的介质中。

SurfaceInteraction 携带的介质信息会首先判断内外介质是否是同一种，如果不是，就说明击中了一个表示参与介质的表面，否则就说明当前的 Ray 在参与介质内部，因此：

```
1 if (mediumInterface.IsMediumTransition())
2     isect->mediumInterface = mediumInterface;
3 else
4     isect->mediumInterface = MediumInterface(r.medium);
```

有时候我们需要参与介质有一个表面，例如湖面的水，表面代表了反射特性。但有时候我们不需要表面，比如云，我们只需要它内部的光衰减效果。

虽然这样一个消失的且不影响光线路径的表面（比如云）可以由 BTDF 完美精确地描述，BTDF 表示两侧具有相同折射率的完美镜面透射，但处理此类表面会给积分器带来额外负担（并非所有积分器都能很好地处理此类镜面光传输）。因此，pbrt 允许这样的表面 Material* 为 nullptr，表示它们不影响通过它们的光；反过来，SurfaceInteraction::bsdf 也将为 nullptr，光传输例程不担心这些表面的光散射，只考虑它们所处当前介质的变化（我们之前也提过很多次了，如果没有计算得到 bsdf，就说明到了参与介质的表面）。

Scene 函数提供了 intersecTr() 方法，它是 Scene::Intersect() 方法的一个推广，该方法返回沿给定 Ray 的光散射表面的第一个交点以及到该点的光束透射比（如果找不到交点，此方法将返回 false，并且不会初始化提供的 SurfaceInteraction）；如果找到了交点，但交点没有材质属性，说明是在体空间表面，则再次产生该方向的 ray，继续求交。该函数会在 EstimateDirect 函数中调用，等到后面用到时它的作用就会一目了然了。

1 4 Medium 交点

Interaction 是交点的基类，这里就包含了一些与参与介质有关的函数，比如 IsMediumInteraction，根据法向量是否是 0 向量来判断是否是参与介质交点；GetMedium 表示在表面边界时，根据法向量和 ray

夹角来判断是返回内部的还是外部的介质；另一个 GetMedium 方法则表示内部交点，需要判断内外介质是否相同。

MediumInteraction 继承自 Interaction，里面只是多了一个相位函数成员。

1 5 Homogeneous Medium 和 3D Grids

HomogeneousMedium 派生自 Medium 基类，表示该空间具有常量 σ_a 和 σ_s 值。

Tr 函数的计算很简单，首先需要注意：

```
//下面表示Ray的实际长度，因为ray.d不一定是单位向量
ray.tMax * ray.d.Length()
```

但是如果出现了 0 乘以无穷，就会得到 NaN 值，因此需要 std::min 函数来返回无穷值。

至于 sample 函数，我们后面再进行讲解。

GridDensityMedium 表示空间场，3D 网格，网格上不同的值会映射为密度，即 Density 函数，该函数从 density 数组里查找值并返回。Tr 函数相对比较复杂，涉及光线在粒子中的自由路径问题，我们后面再介绍。

1 6 代码移植

本章我们先将 PhaseFunction 类，HenyeyGreenstein 类移植好。然后再移植 Medium 类和 MediumInterface 结构。

注意 Interaction 类需要在里面加入介质类，构造函数也得进行修改。同时里面有很多函数都需要修改或移植：

```
SpawnRay
SpawnRayTo
SpawnRayTo
IsMediumInteraction
GetMedium(const Vector3f &w)
GetMedium()
```

同时实现 Interaction 类的派生类 MediumInteraction。注意派生类的相位函数指针要定义成智能指针，因为相位函数是要临时创建的，因此需要及时释放掉，使用智能指针管理会很方便。

```
//相位函数输入参数去掉const
MediumInteraction(const Point3f &p, const Vector3f &wo, float time,
const Medium *medium, PhaseFunction *phase)
//相位函数指针
std::shared_ptr<PhaseFunction> phase = nullptr;
```

但是改过这个以后，编译会出现很多问题，首先是 Light 可能在参与介质里，因此需要修改 Light 的构造函数。Light 基类和各个派生类都需要进行修改。

我们不需要将介质类、介质接口里面用到的指针改为智能指针，因为参与介质是一开始就构建好的，。

Ray 和 RayDifferential 类中也有参与介质成员变量，需要移植。

相机类也要实现一下参与介质接口，在我们目前的构造中，传入参数是 nullptr 就可以了。注意相机类的介质成员变量也不需要用智能指针：

```
const Medium* medium;
```

因为构造时传入的是变量取地址（api.cpp 的 MakeCamera 函数）：

```
MediumInterface mediumInterface = graphicsState.CreateMediumInterface();
camera = CreateOrthographicCamera(paramSet, animatedCam2World, film,
mediumInterface.outside);
```

相机类需要实现产生携带了介质信息的 Ray，因此下面的函数都需要修改：

```
//Transform.h
RayDifferential Transform::operator()(const RayDifferential &r)
Ray Transform::operator()
//Camera派生类
virtual float GenerateRay(const CameraSample &sample, Ray *ray) const {
   return 1; };
    virtual float GenerateRayDifferential(const CameraSample &sample,
           RayDifferential *rd) const;
```

GeometricPrimitive 类里面有 MediumInterface 对象，同时 Intersect 函数也需要进行修改。

移植完以后，能编译就说明移植成功。我们要在移植的时候对每个类中新加入的成员变量有一个初步的印象，比如相机和 Ray 的参与介质成员变量是 Medium 对象，而 GeometricPrimitive 类和 Light 类的成员变量是 MediumInterface 对象，这是因为有的类需要表示内外参与介质类型，而有的类恰好处于参与介质内部，因此不用包含内外两种介质类型。

二　体渲染基本功能

本章描述传输方程和体散射。

2 1　传输方程

使 radiance 增强的组件：

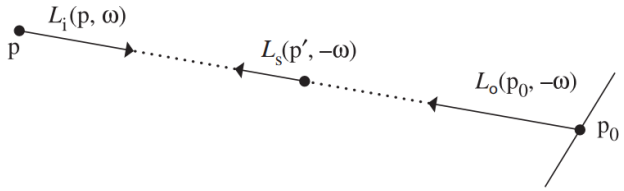
$$L_s(p,\omega) = L_e(p,\omega) + \sigma_s(p,\omega)\int_{S^2} p(p,\omega_i,\omega)L_i(p,\omega_i)d\omega_i \tag{二.1}$$

使 radiance 衰弱的组件：

$$\frac{dL_o(p+t\omega,\omega)}{dt} = -\sigma_t(p,\omega)L_i(p,-\omega) \tag{二.2}$$

将两者组合起来，得到：

$$\frac{dL_o(p+t\omega,\omega)}{dt} = -\sigma_t(p,\omega)L_i(p,-\omega) + L_s(p,\omega) \tag{二.3}$$



$$L_i(p,\omega) = T_r(p_0 \to p)L_o(p_0,-\omega) + \int_0^t T_r(p' \to p)L_s(p',-\omega)dt' \tag{二.4}$$

我们可以将之前得到的渲染方程的几何项 G 写成如下形式：

$$\hat{G}(p \leftrightarrow p') = V(p \leftrightarrow p')T_r(p \to p')\frac{C_p(p,p')C_{p'}(p',p)}{\|p-p'\|^2} \tag{二.5}$$

$$C_p(p,p') = \begin{cases} \left|n_p \cdot \frac{p-p'}{|p-p'|}\right|, & p\ is\ surface\ vertex \\ 1, & otherwise \end{cases} \tag{二.6}$$

注意其实在表面时与前面相同，区别仅在于体空间非表面处。

2 2　PBRT 体渲染功能分析

在介绍功能之前先把所有功能整理一下。

参与介质的类为 Medium，有两个子类，一是 HomogeneousMedium，二是 GridDensityMedium。但是对于场景中的物体，经常它内外属于不同的介质，因此 MediumInterface 就表示内外具有不同介质类型。shape 携带的参与介质是 MediumInterface，而相机类携带的参与介质接口是 Medium

PhaseFunction 描述了物体是如何进行散射的，MediumInteraction 携带有该成员变量。在 Medium 的 Sample 函数中会计算生成相位函数，并创建 MediumInteraction 对象。

那么接下来就分为两个我们需要补充和完善的地方了：第一，当 Ray 与 Medium 相交时，应该如何在 Medium 内部采样；第二，体渲染过程应该如何实现 Ray 与 Medium 的相交。

我建议先从整体上学习，再填充细节，因此我们先研究一下渲染过程。

2 3 体渲染过程

首先，Whitted 积分器里面是不包含与参与介质渲染有关的内容的，顶多会在碰到参与介质边界时 Ray 继续原方向前进：

```
if (!isect.bsdf)
    return Li(isect.SpawnRay(ray.d), scene, sampler, arena, depth);
```

DirectLighting 积分器也是如此，尽管 EstimateDirect 函数（integrator.cpp 文件）会考虑参与介质对光的衰减，但该函数的输入参数有一个 bool handleMedia，默认为 false，在该积分器中调用该函数时该参数都是 false，即不考虑参与介质的影响。

Path 积分器也不会直接处理参与介质，它会处理 BSSRDF，也会使用到 EstimateDirect 函数，但同样调用 EstimateDirect 函数时传入的 handleMedia 值是 false，即不考虑参与介质的影响。

VolPath 积分器主要就是处理参与介质的，在没有参与介质时，它的表现和 Path 积分器没有什么本质区别，但当遇到参与介质后会进行体渲染：

```
if (mi.IsValid()) { }
else { }
```

我们简单了解一下 EstimateDirect 函数（integrator.cpp 文件），采样光源的时候，如果 BRDF 不为 0：

```
if (!f.IsBlack())
```

则需要对光进行采样，则执行：

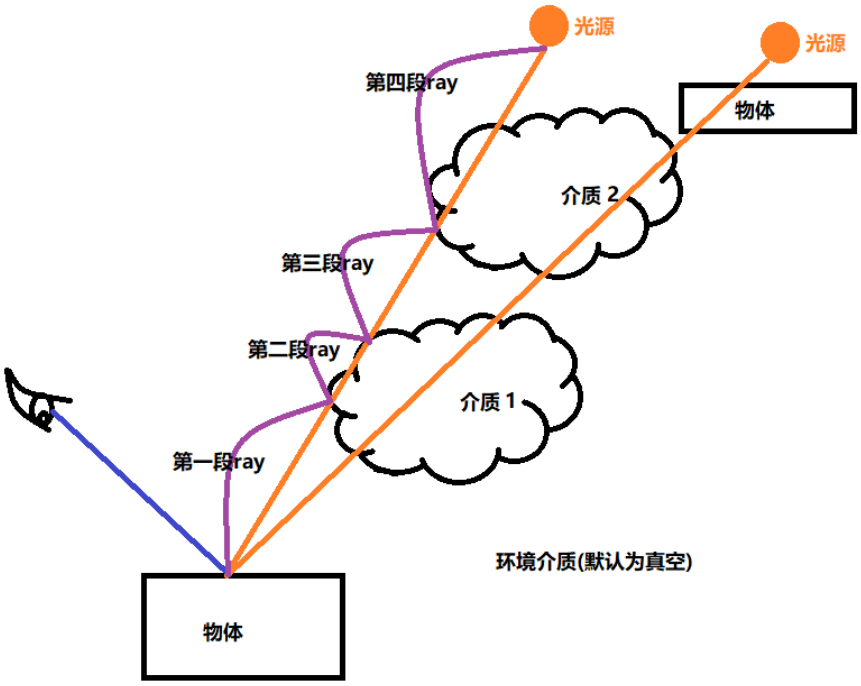
```
Li *= visibility.Tr(scene, sampler);
```

Tr 函数就是判断光路径的光衰减的：

```
//VisibilityTester::Tr
//可能沿途中不止一个参与介质，或者介质不规则，可能与Ray有多次相交。
while(true){
    //求光源采样点与物体之间是否有交点
    //如果有交点，且交点不是参与介质，则说明光被阻挡
    //光没有被阻挡：
    if (ray.medium) Tr *= ray.medium->Tr(ray, sampler);
    //如果没有击中表面，说明沿途中不再有遮挡物或参与介质，跳出循环
}
//返回Tr
```

由此我们再次提一下，Medium 的 Tr 函数计算的是整个 Ray 直线路径上，光被衰减的比例。

我们用图示表示一下，如下图，我们假设外部为真空，即 MediumInterface 的外部参与介质都是 nullptr。在 VisibilityTester::Tr 的 while 循环中，第一次循环时，检测到与介质 1 有交点，且交点属性是介质，但是因为 Ray 此时的 Medium 对象为 nullptr，因此不需要计算光衰减。然后通过 SpawnRayTo 生成新的 Ray，此时 Ray 的 Medium 对象是介质 1。第二次循环中，与介质 1 的另一个表面相交，此时需要计算光衰减。while 不断循环，直到碰到遮挡物或者中间不再有任何表面时，就退出循环。



我们将 VisibilityTester 的 Spectrum Tr 函数移植到自己的系统中。

EstimateDirect 函数不光需要采样光源，还需要采样 BSDF，然后判断采样方向到最近交点之间是否有参与介质：

```
bool foundSurfaceInteraction =
handleMedia ? scene.IntersectTr(ray, sampler, &lightIsect, &Tr)
            : scene.Intersect(ray, &lightIsect);
```

Scene::IntersectTr 我们之前简单介绍过，这里再说一遍。该函数体也是一个 while 循环，非常类似于 Medium 的 Tr 函数，只是它需要在循环中寻找最近的非介质表面，如果没有与非介质表面相交，则返回 false。

我们将 Scene::IntersectTr 函数移植到自己的系统里。EstimateDirect 函数需要修改成包含参与介质的内容，根据上面所述，相信大家很容易就能改会包含参与介质的情况（最好是在自己系统上逐步添加，而不是直接把 PBRT 原函数复制过去替换我们的代码，这样你才能更清楚加入参与介质后对渲染器有哪些影响，尽管可能这样做更容易出错）。

2 4 VolPath 积分器

VolPath 相比于其他 SamplerIntegrator 子类，只是 Li 函数不太一样，我们讲讲该函数以后，就可以直接进行移植了。尽管我们还没有移植 Medium 的子类，也还没有讲解采样相位函数，但至少我们先把整个积分器搞定：

```
for (bounces = 0;; ++bounces){
    SurfaceInteraction isect;
    MediumInteraction mi;
    //分别计算isect与mi。如果当前ray在参与介质中，则mi就会被正确赋值
    if (mi.IsValid()){
        //计算当前介质点的光照值
        //根据采样相位函数来产生新的散射方向
    }
    else{
        //与表面相交，计算表面BSDF特性
        //与路径追踪一致
```

```
12          }
13      }
```

移植也非常容易，只需要删除里面的与 BSSRDF 有关的内容即可，剩下的例如 Float 改为 float，去掉内存管理类等和以前一样，就不再提了。

2 5 体渲染中的采样

体渲染中采样分为两种，一是在参与介质中找下一个散射点（根据当前点的 Density），二是采样散射方向（根据当前点计算出的相位函数）。

找下一个采样点的函数是 Medium::Sample 函数。根据当前点采样散射方向的函数是 PhaseFunction::Sample_p 函数。它们的使用可以从 VolPath 积分器中找到，我们下一章主要就是介绍与采样有关的内容。

三　采样散射点

本章我们完成体渲染最后的内容，即采样。

3 1　Sample() 函数

Sample 函数的意义在于采样到合理的散射点。我们先回顾一下公式：

$$L_i(p,\omega) = T_r(p_0 \to p_1)L_o(p_0,-\omega) + \int_0^t T_r(p+t\omega \to p)L_s(p+t\omega,-\omega)dt \tag{三.1}$$

如果 Sample 没有采样到散射点（比如与介质中的物体表面相交了，或者离开了介质），则直接计算 $T_r(p_0 \to p)L_o(p_0,-\omega)$。

如果 Sample 采样到散射点了，则需要计算衰减项 $\int_0^t T_r(p+t\omega \to p)L_s(p+t\omega,-\omega)dt$，并初始化 MediumInteraction（用来生成下次散射的方向）。

假如 $p_t(t)$ 代表在介质采样中，单位长度下产生一个散射点的概率，注意这个概率不能积分到 1（因为无论何时都有可能不产生散射点），我们定义 $p_{surf} = 1 - \int_0^{t_{max}} p_t(t)dt$ 来表示采样到表面项的概率。

Sample 与以前遇到的散射函数（如 BSDF::Sample_f()）不同，因为它不向调用者提供关于函数值和采样位置处 Pdf 的单独信息。通常我们不需要这些信息，一些介质模型（特别是 heterogeneous 介质）在有可能计算这些量的比率时允许更有效的采样方案。选定 surface 项后，可以计算一个权重：

$$\beta_{surf} = \frac{T_r(p \to p+t\omega)}{p_{surf}} \tag{三.2}$$

它对应于 $T_r(p_0 \to p)L_o(p_0,-\omega)$ 的采样。

在介质中：

$$\beta_{med} = \frac{\sigma_s(p+t\omega)T_r(p \to p+t\omega)}{p_t(t)} \tag{三.3}$$

表示除了内散射光积分介质相关的项 $\int_0^t T_r(p+t\omega \to p)L_s(p+t\omega,-\omega)dt$。

散射系数和透射率允许光谱变化，因此该方法返回光谱值权重因子，以更新路径前进量权重 β 直至表面或介质散射事件。

通常情况对于蒙特卡罗积分，β_{surf} 和 β_{med} 等估计器都承认各种采样技术，它们都产生了期望的分布。异构介质的实现将利用这一事实提供一种比基于反演方法 (inversion method) 的规范采样方法更有效的实现。

通过 PBRT 两种实现的介质可以更好的说明情况。

3 2　均匀介质

HomogeneousMedium::Tr 函数没有什么可说的，套用公式即可。

HomogeneousMedium::Sample 函数相对复杂一点。

使用蒙特卡洛方法采样指数函数时，例如 $f(t) = e^{-\sigma_t t}$，得到采样 t 的计算方法：

$$t = -\frac{ln(1-\xi)}{\sigma_t} \tag{三.4}$$

$$p_t(t) = \sigma_t e^{-\sigma_t t} \tag{三.5}$$

首先采样一个光谱通道 i（因为衰减系数是跟着波长变化的），相关的标量 σ_t^i 来采样一个分布距离。

$$\hat{p}_t^i(t) = \sigma_t^i e^{-\sigma_t^i t} \tag{三.6}$$

因此采样密度是每个采样策略的平均值。

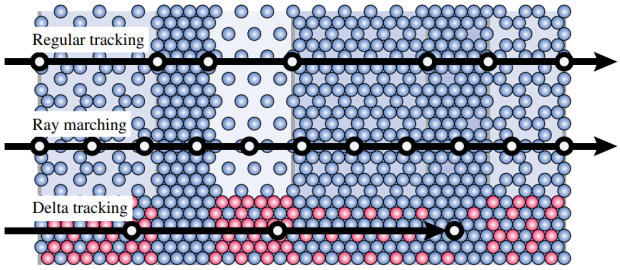
$$\hat{p}_t(t) = \frac{1}{n}\sum_{i=1}^{n}\sigma_t^i e^{-\sigma_t^i t} \tag{三.7}$$

$$p_{surf} = 1 - \int_0^{t_{max}} \hat{p}_t(t)dt = \frac{1}{n}\sum_{i=1}^{n} e^{-\sigma_t^i t_{max}} \tag{三.8}$$

程序的过程和上述是一致的。

3 3 异质介质

异质 heterogeneous 介质用 GridDensityMedium 表示其内部分布不均匀。寻找散射点一般有三种方法：



（顶部）常规跟踪将介质划分为若干同质子区域，并依赖于处理单个同质区域的标准技术。（中间）Ray marching 将光线分割成若干离散的片段，并通过每个片段近似计算透射率。（底部）Delta 跟踪有效地考虑了一种介质，该介质被额外的虚拟粒子（红色）填充，直到达到均匀密度。

其实我们使用随机步长随机位置反射也是可以的，但这里的主要意义在于无偏渲染，因此实现中采用了 woodcock 跟踪的方法。

GridDensityMedium::Tr 的实现过程中使用了俄罗斯轮盘赌的方法。

3 4 采样相位函数与移植

相位函数可以求出对于 $cos\theta$ 的分布，然后我们需要再根据 cos 分布求出 θ 分布来，之后将 θ 与随机的 ϕ 作为球坐标值转换到世界空间坐标系下。

移植的过程相对比较简单，这里就暂且不提了。

3 5 本文全部需要移植的内容清单

列一下本文全部需要移植的内容清单，以便读者能更好地掌握为了支持体渲染，PBRT3 需要实现哪些内容。

内容一： Media 目录下的 Medium 类以及它的派生类 HomogeneousMedium 和 GridDensityMedium 都是要整体移植上来的。

内容二： Ray 类加入参与介质相关功能。

内容三： Camera 类及其派生类要加入参与介质的相关功能。注意 GenerateRay() 和 GenerateRay-Differential() 函数都需要对 Ray 的参与介质进行赋值：

```
1    ray->medium = medium;
```

创建正交投影相机的 CreateOrthographicCamera() 函数与创建透视投影相机的函数也需要修改加入参与介质功能。

内容四： interaction.h 文件里的 Interaction 类需要加上参与介质接口。GetMedium() 两个重载函数。这里有很多函数都用到了参与介质，移植时需要细心。

内容五： Light 类及其派生类都有用到参与介质。VisibilityTester 类中的 Tr() 需要修改。

内容六： GeometricPrimitive 类中需要加上与参与介质有关的功能。

内容七： Scene 类的 IntersectTr() 函数。

内容八：采样光的 EstimateDirect 函数需要计算 Tr()。

内容九：ValPath 积分器要整体移植上来，注意去除 BSSRDF 的部分。

需要移植的内容就是这些。

四　体渲染测试

本章的内容是测试体渲染。

4 1　HomogeneousMedium 测试

注意，参与介质要定义为全局变量。

```
//根据自己的喜好进行调整
HomogeneousMedium homogeneousMedium(2.4, 1.4, 0.5);
//该接口给我们的模型使用
MediumInterface mediumDragon(&homogeneousMedium, nullptr);
MediumInterface noMedium;//不需要参与介质的类使用该变量作为参数
```

得到渲染结果如下：

4 2　GridDensityMedium 测试

我们使用 git Bash：

```
git clone git://git.pbrt.org/pbrt-v3-scenes
```

之后在 C 盘的 Users/用户名下（默认路径）得到各种场景文件。

cloud 文件夹下的文件是我们需要的，里面 cloud.pbrt 或者 somke.pbrt 中可以看到，参与介质的面片文件在 geometry/density_render.ply 里，而网格数据则在 density_render.70.pbrt 文件里。

我将 ply 文件读取到的顶点输出一下：

```
Triangle1
1.99 1.4 0.1
1.99 1.4 0.79
0.1 1.4 0.79
Triangle2
1.99 1.4 0.1
0.1 1.4 0.79
0.1 1.4 0.1
Triangle3
1.99 3.29 0.1
0.1 3.29 0.1
0.1 3.29 0.79
......
```

其实就是一个长方体。

我们可以把 density_render.70.pbrt 文件改一改：

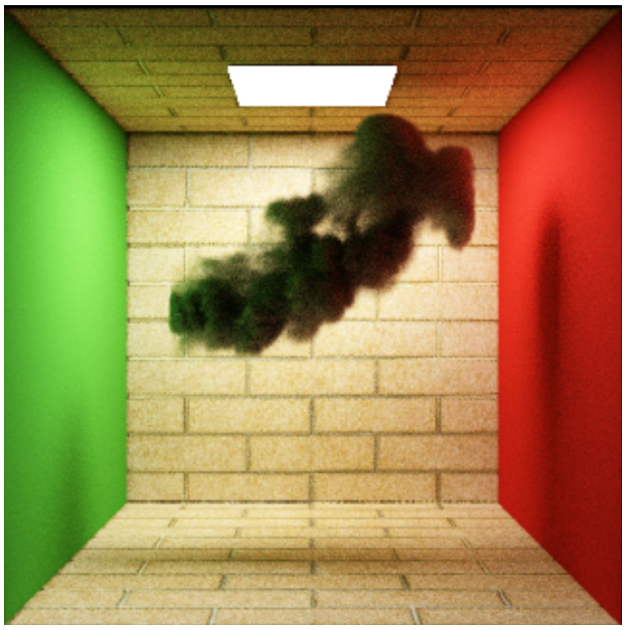
```
nx 100 ny 100 nz 40
p0 0.010000 0.010000 0.010000
p1 1.990000 1.990000 0.790000
//根据自己的需求来设置散射和吸收的比例
sigma_a 10 10 10
sigma_s 90 90 90
```

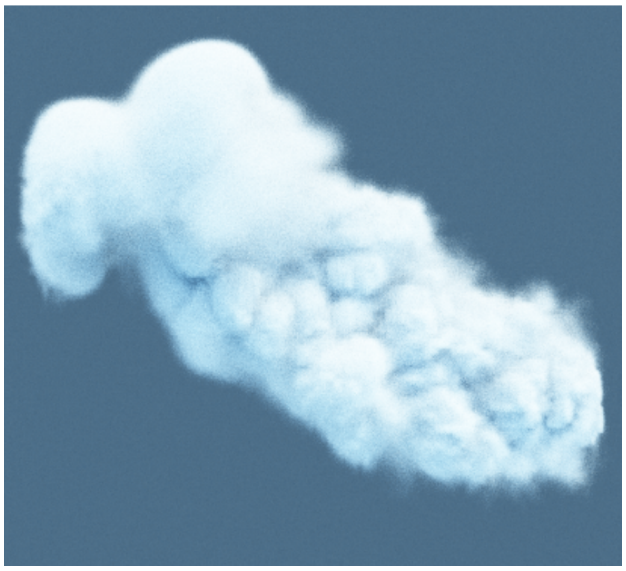
然后构建：

```
std::fstream f("文件路径");
std::string ed;
f >> ed; f >> nx;
f >> ed; f >> ny;
f >> ed; f >> nz;
Point3f p0; Point3f p1;
f >> ed; f >> p0.x; f >> p0.y; f >> p0.z;
f >> ed; f >> p1.x; f >> p1.y; f >> p1.z;
float sig_a_rgb[3] = { .0011f, .0024f, .014f },
    sig_s_rgb[3] = { 2.55f, 3.21f, 3.77f };
f >> ed; f >> sig_a_rgb[0]; f >> sig_a_rgb[1]; f >> sig_a_rgb[2];
f >> ed; f >> sig_s_rgb[0]; f >> sig_s_rgb[1]; f >> sig_s_rgb[2];
Spectrum sig_a = Spectrum::FromRGB(sig_a_rgb), sig_s = Spectrum::FromRGB
   (sig_s_rgb);
float scale = 1.f;
float g = 0.f;
float *data = new float[nx * ny * nz];
for (int i = 0; i < nx*ny*nz; i++)
    f >> data[i];
Transform data2Medium = Translate(Vector3f(p0)) *
```

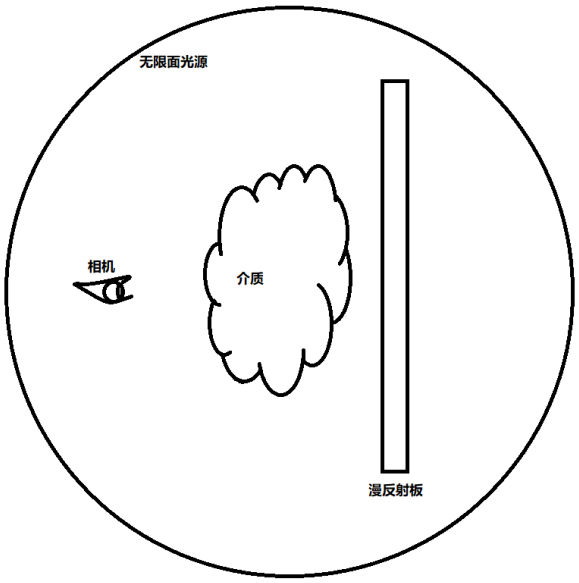
```
20          Scale(p1.x - p0.x, p1.y - p0.y, p1.z - p0.z);
21      //medium2world是介质到世界坐标系的变换
22      med = std::make_shared<GridDensityMedium>(sig_a, sig_s, g, nx, ny, nz,
23          medium2world * data2Medium, data);
```

渲染显示的结果如下：





场景的构建大家可以参考 cloud.pbrt 的实现，我这里把产生比较白的云的场景构建给大家演示一下：



在我们给出的源码中，getMediumBox 中的 pMin 和 pMax 就是体包围盒的最大最小坐标，这是 getBox 函数修改得到的。我们加载异构介质定义的类是自己定义的 MediumLoad 类。getBox 函数以前的立方体坐标设置的有点问题，现在全都改正了过来。

五　总结

本书从 4 月 2 日一直写到了 4 月 15 日，中间因为科研任务重延迟了一段时间。

PBRT 中的参与介质虽然与各个 PBRT 类都联系密切，但其实整体并不复杂。移植的时候我因为漏掉了一些内容，所以调试了很久才找到错误原因。在调试的过程中，几乎能把整个渲染的流程背过了。因此如果您在移植与实现中遇到了一些问题，可以对比源码，一点一点查错（也可以仔细看本书，本书将所有需要移植的地方全都进行了叙述）。

PBRT 高级篇就此结束。目前 PBRT 系列书我一共规划了 27 本，下一个阶段是“PBRT 专业知识理论与代码实战”系列，我会重点讲解基础原理（其实前面的 PBRT 系列主要是动手操作和移植，了解系统架构和基本知识）以及编程实现的细节，因此知识量和内容会非常丰富，同时写作时间也会变得很长（恐怕写完一本需要一个月以上）：

PBRT 系列 16-专业知识理论与代码实战-物理材质

PBRT 系列 17-专业知识理论与代码实战-概率与采样

PBRT 系列 18-专业知识理论与代码实战-次表面散射

PBRT 系列 19-系统功能扩展-复杂模型的读取接口

PBRT 系列 20-专业知识理论与代码实战-切线空间与凹凸贴图、透明贴图

PBRT 系列 21-专业知识理论与代码实战-运动模糊与实例化

PBRT 系列 22-专业知识理论与代码实战-准蒙特卡洛与低差异序列

PBRT 系列 23-专业知识理论与代码实战-图像重建与滤波

还有 PBRT 高级渲染积分器的内容：

PBRT 系列 24-高级积分器-随机渐进式光子映射

PBRT 系列 25-高级积分器-Metropolis 光传输

PBRT 系列 26-高级积分器-双向路径追踪

PBRT 系列 27-高级积分器-球谐光照积分器

我以前曾经在自己设计的渲染器上实现过这些积分器，但显然 PBRT 中的实现更为精准和巧妙，因此在高级积分器系列我会着重研究它们的实现，并详细阐述基本原理。

参考文献

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