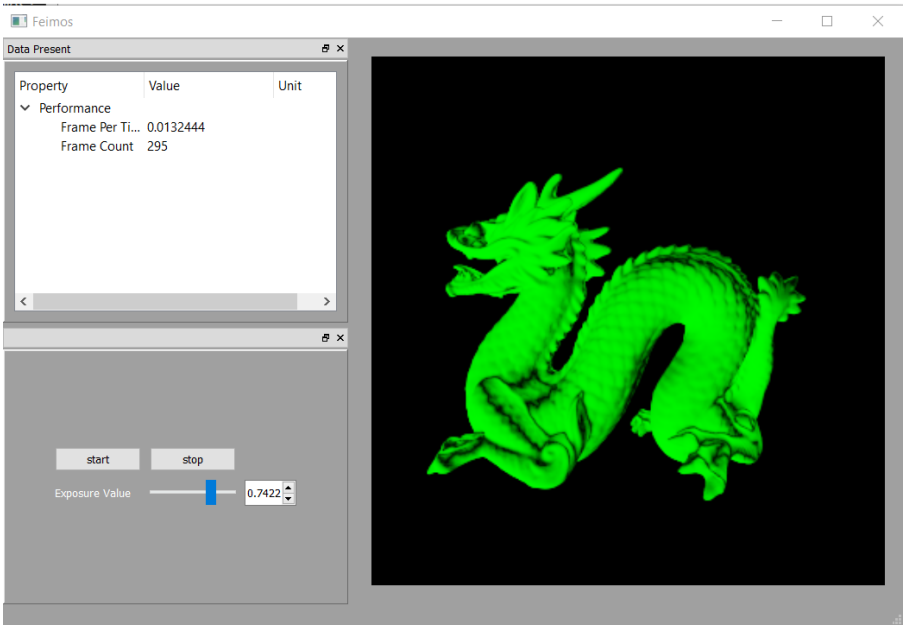
PBRT 系列 4-代码实战-颜色与光谱

Dezeming Family

2021 年 2 月 26 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，最好从官方网站：https://dezeming.top/ 下载新的版本（用户注册后即可免费下载）。

本书目标：学习 PBRT 的光谱类，移植到自己的渲染系统上。渲染出下面的图像效果：



本文于 2022 年 7 月 5 日进行再版，提供了源码。注意源码中的图形 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

没想到这么快，我就写到第四本书了。

本来第四本书是要写光线微分技术，以及误差界定和内存管理等内容的，但是考虑到它们确实比较难，其实系列 3 已经有点难了，而且内容量也很大，虽然我计划读者用三天可能就能学完，但也许系列 3 要学一周左右。因此系列 4 我打算写点简单的内容，也就是颜色与光谱。我们的系列书的写作顺序就是读者的阅读顺序。

注意颜色与光谱虽然我们在这本书就定义和实现了，但是 RGB 与 XYZ 互转用来计算的内容（包括 tone mapping 等），直到我们开始实现真正的 PBRT 渲染器时才开始使用。我们当前只是把上一本书临时使用的 Vector3f 类代替的颜色类替换为一个独立表示颜色的类——Spectrum 类。

本书的售价是 3 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　本书的基本介绍

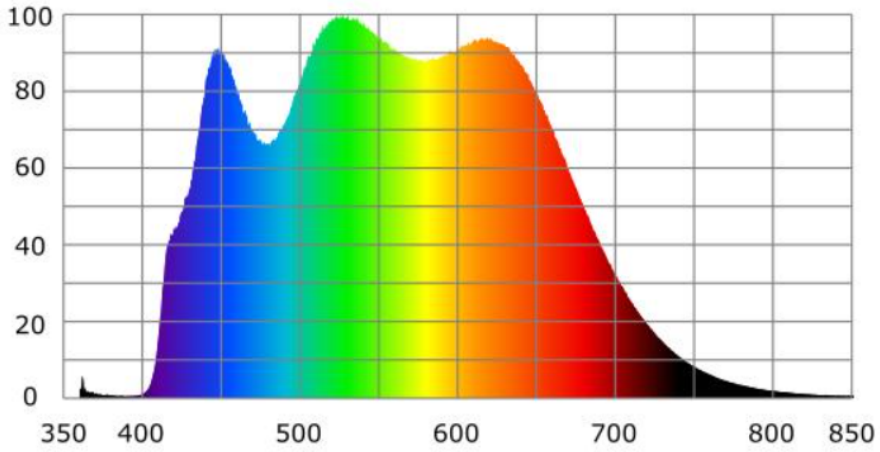
因为有了光，我们才能看到物体。

究竟什么是光，我们知道，光是一种辐射电磁波，既然是一种波，它就有波长和频率。一般可见光的波长都在 380nm 到 780nm 之间，我们看到的光一般都是各种不同波长的光混合而成的。光在图形学中的表示一般称作辐射度量学（radiometry）。我们本书并不涉及与 BRDF、辐射通量等有关的内容，这会在材料与纹理书中进行讲解。

在学习颜色与光谱之前，我打算插入一些前提知识，希望对颜色与光并不熟悉的读者能够能耐心看完。

光速 = 波长乘以频率，光在不同介质中的速度是不一样的，例如在水中的速度要小于在真空中的速度。我们当前假设所有的描述都是在真空中，也就是说光速相等。

当波长为 700nm 左右时的单波长光我们感受到的是红色，当波长为 450nm 左右时的波长我们感受到的是蓝靛色。那什么是白光呢？白光的定义其实没这么简单，有人说，白光就是包含了各种波长的光的集合，这种说法并不正确，因为当一束光中，如果波长 700nm 左右的光占绝大部分，那么人们感受到的就是偏红色的光。那么白光需要所有的波长在整束光中的比例相等吗？这也未必。我们一般认为阳光属于白色光，蓝天下阳光的光谱如下，可以看出，阳光的光谱中，各个波长的比重都有不同（阳光直射的光谱是不一样的）。



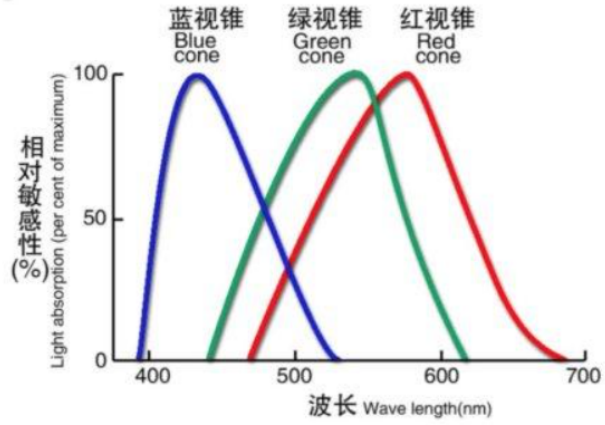
在计算机中，光经常会使用 RGB 等表示方法，也就是说标准的红绿蓝经过组合来得到其他颜色。而什么样的光谱要被表示为标准的红色呢？这里面牵扯到很多问题。除了 RGB，还有 CMY、HSV 等颜色模型，为什么我们有不同的颜色模型，这其中也有很多门道，更详细的介绍可以看 DezemingFamily 的《颜色空间理论》。

由上所知，颜色系统虽然在 PBRT 中只是一个工具类，但也包含了不少的知识。我们会首先讲解颜色和光谱，然后再讲解 PBRT 系统的描述，最终将 PBRT 的颜色系统应用到我们自己的系统上。

二　颜色与光基础

谱功率密度（SPD）表示光线中不同波长的光占的比例。SPD 是线性的，也就是说两种光可以线性叠加，比如开两个手电筒会比一个手电筒亮。颜色是人感知的 SPD，并不是真正的物理现象，我们一般定义颜色是符合大多数人感知的色彩，也有少数人对颜色的感知与一般人不太一样，比如红绿色盲患者。

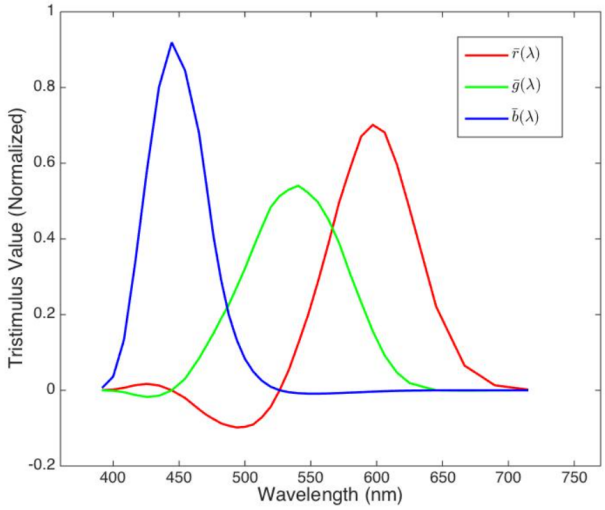
人眼内部有三种锥形细胞来感知颜色，如下图，不同的视锥细胞对光的感知响应是不同的。对于波长 500 的光，绿视锥响应最强烈，红蓝视锥响应很相近；对于波长 650 的光，红视锥响应最强烈，蓝绿视锥几乎不响应。神奇的是这三种细胞的分布在显微镜下不同人之间差异非常明显。



基于此，我们感受到的颜色值就是实际光谱与我们的视锥细胞对光响应的面积的积分。我们标记三种细胞分别为 S，M，L，对于不同的光谱，积分以后就得到了不同的值，也就是一个向量（s,m,l）。之后，我们的大脑就会将（s,m,l）翻译成我们感知到的颜色。

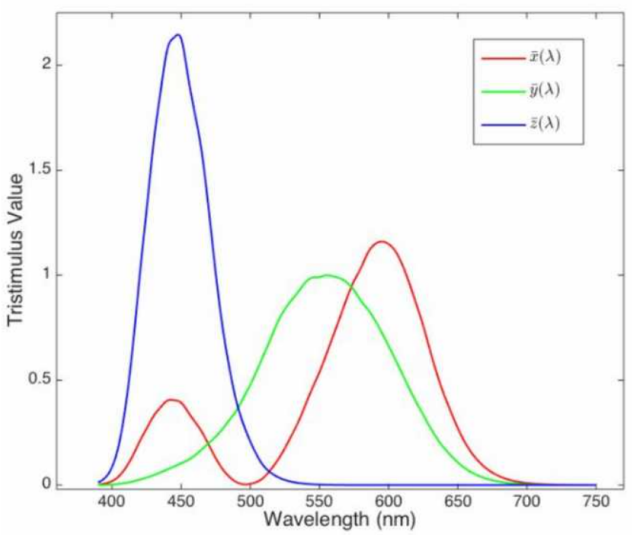
我们可以想象，（s,m,l）不一定是仅有一种光谱得到的，很有可能有不同的光谱积分后得到同样的（s,m,l）值，即同色异谱。

颜色之间是可以混合的（谱之间相加）。自然界都是加色系统，简单来说多个光谱相加就会变亮。计算机图像一般也是使用加色系统，即 RGB 三原色。CIE 组织定义了当前主流的 RGB 系统。R 的波长是 700nm 左右，G 是 546nm 左右，B 是 435.8nm 左右，CIE-RGB 的目标是使用这三种基础三原色混合出与任何单一波长的光感知到的颜色一样的颜色，经过实验和匹配，得到如下结果：



大家可能会不理解图上负值是什么意思，难道颜色还能表示成负的？简单来说，这是因为 RGB 三原色饱和度太高，导致低饱和度的颜色混合不出来。比如我们要混合某种颜色 $C(\lambda)$，但是我们混合不出来，不过我们发现:$C(\lambda)+0.15R=0.73G+0.28B$，因此 $C(\lambda)=-0.15R+0.73G+0.28B$，也就出现了负值（更详细的解释可以参考《颜色空间理论》）。

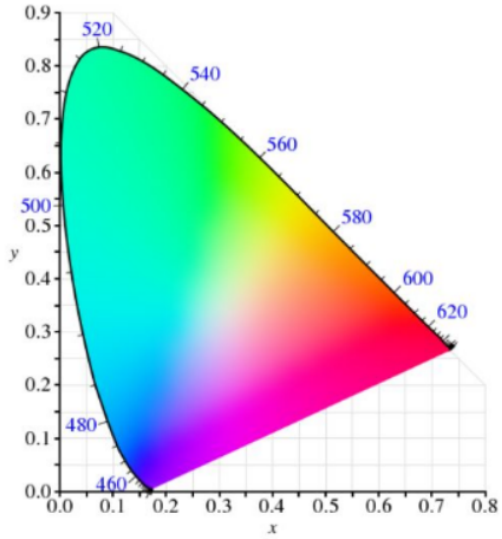
另外一个颜色系统叫 CIE-XYZ 系统，该系统在匹配不同波长光时不会有负数：



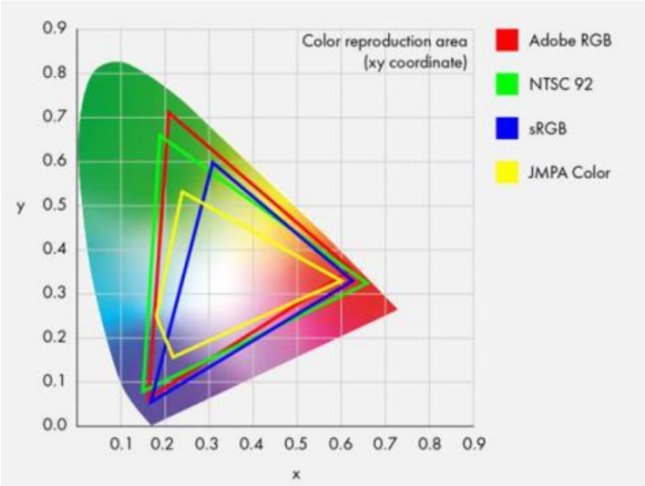
通过 CIE-XYZ 曲线可以合成颜色值（X,Y,Z），由于人眼对绿色的敏感性，我们可以认为这里的 Y 是表示亮度（注意曲线 y 横跨了整个光谱）。通过系数归一化，即:

$$(\frac{X}{X+Y+Z}, \frac{Y}{X+Y+Z}, \frac{Z}{X+Y+Z}) \tag{二.1}$$

我们就能得到色彩空间可视化结果（因为 z=1-x-y，所以我们只需要 x 和 y 坐标，但是因为 Y 表示亮度，所以我们通常显示的色域图是固定 Y 的。Y 值不同，下面显示的色域图其实是不同的。虽然下图的横纵坐标显示的是 x 和 y，但这里的 y 轴其实表示的是 Z 归一化以后的值）：



上图的颜色空间就叫色域。不同的颜色空间表示的色域范围都不一样，比如 sRGB 只能表示里面三角形区域内的颜色值：



在 Photo Shop 软件常使用 HSV 系统（Hue-Saturation-Value），即色调-饱和度-亮度三个属性来定义颜色。HSV 和 sRGB（计算机常用的 RGB 颜色空间我们一般称为 sRGB）可以互相转化。

三　PBRT 的光谱表示

实际的光谱分布 SPD 是一个连续的函数，但是计算机没法储存和计算连续函数，因此一般使用一些近似。比如使用基函数（笼统来讲，用来构成目标函数的子函数一般都称为基函数）混合，例如使用 RGB 三原色混合。

3 1　PBRT 的光谱 SPD 表示

PBRT 的光谱类在文件夹 core 里的 spectrum.h 和 spectrum.cpp 文件中实现。

光谱类的基类名为 CoefficientSpectrum，有两个派生类：RGBSpectrum 和 SampledSpectrum。值得注意的是，在 phrt.h 文件中进行了声明：typedef RGBSpectrum Spectrum; 也就是说，当你使用 Spectrum 命名时，默认使用的是 RGBSpectrum。提一句，这里提前声明一个弊端，即如果您将 Spectrum 作为 SampledSpectrum 类型的别名，那么您将需要重新编译整个 PBRT 系统。但同时两有个好处：第一个好处是，您直接使用的子类，因此运行时不会因为动态选择运行子类虚函数而浪费时间，而是直接使用派生类的 inline 函数；第二个好处是编译时我们就知道了光谱类型，而不是得等到运行时动态分配。

3 2　CoefficientSpectrum 光谱类

CoefficientSpectrum 类是一个模板类：

```
1    template <int nSpectrumSamples> class CoefficientSpectrum;
```

nSpectrumSamples 表示该类内部基或者采样点的数量，如果是 RGBSpectrum 类，nSpectrumSamples 就是 3。

CoefficientSpectrum 类包含了加减运算，以及判断是否为黑色的 IsBlack() 函数。还有一些函数，例如开平方，Lerp 等，我们都可以移植到自己的系统里。

值得注意的是光谱可以被表示为一系列基函数的和，比如 XYZ 三种曲线，但是我们需要提前定义这些曲线（以后如果需要实现我们再说）。

3 3　RGBSpectrum 类

RGBSpectrum 类就是使用 RGB 三个基来混合出目标颜色。但不同的电脑显示器可能 RGB 显示都不同，所以很有可能同样的 RGB 值显示出不同的颜色。FromRGB() 方法将三维向量转换为 RGBSpectrum 对象。

RGBSpectrum 提供了 RGB 与 XYZ 系统相互转换的机制：RGBSpectrum::ToXYZ() 和 RGBSpectrum::FromXYZ()。FromSampled() 方法将光谱 SPD（其实就是个 float 数组，大小取决于光谱采样率）转换为 RGBSpectrum 类。

3 4　SampledSpectrum 类

我们在第一本书《文件加载和设定》时就说到过，光谱类型有两种：

```
1    enum class SpectrumType { Reflectance, Illuminant };
```

一般在 PRBT 中，非照明类物体都会表示为 Reflectance，照明物会被表示为 Illuminant。

SampledSpectrum 类表示 SPD，nSpectrumSamples 表示整个光谱离散化分为几段，一般 60 段就够精确了，即 PBRT 中的 SPD 表示为线性分段函数。

```
1    //光谱采样开始位置：400nm
2    static const int sampledLambdaStart = 400;
3    //光谱采样结束位置：700nm
4    static const int sampledLambdaEnd = 700;
```

```
5     //SPD分段数
6     static const int nSpectralSamples = 60;
```

FromSampled() 函数可以将数组转换为 SampledSpectrum。SampledSpectrum 类可以转换为 XYZ 颜色系统值或者 RGB 颜色系统值。

SampledSpectrum 类更多用于照明物，比如日光灯，我们可以提前测试出其光谱。鉴于我们当前基本上用不到 SampledSpectrum 类，里面的方法我就不再介绍了。人生苦短，及时行乐，我们赶紧把颜色系统实现，然后显示点好玩的东西。

四　移植和测试

本章首先移植光谱类，然后基于光谱类实现一章有趣的图像。

4 1　移植

首先是 CoefficientSpectrum 类，可以直接拷贝到自己的系统中，基本上不需要任何修改。
枚举类 SpectrumType 可以拷贝上来。公共函数 RGBToXYZ() 和 XYZToRGB() 也都是我们需要的。
RGBSpectrum 可以直接复制过来。成员函数 FromSampled() 我们暂时也不需要，直接删除。
之后再在自己的公共头文件声明（PBRT 中是 pbrt.h 的头文件）：

```
template <int nSpectrumSamples>
class CoefficientSpectrum;
class RGBSpectrum;
typedef RGBSpectrum Spectrum;
```

编译没有问题，就算移植完了。

4 2　测试

把之前用 Vector3f 表示的颜色改为如下表示：

```
Spectrum colObj(0.0f); colObj[0] = 1.0f; colObj[1] = 1.0f;
if (agg->Intersect(r, isect)){
    colObj[1] = 0.0f;
}
//显示像素
......
```

编译运行正确，说明光谱类移植成功。
其实到现在，这本书已经结束了。
但每本书我们都应该搞点新东西，对我来说也是这样，因为我想换个新的封面。如果您的模型自带法向量数组，那么就可以直接用于渲染。很可惜的是我的模型没有法向量，所以需要运行的时候计算法向量。
我们在 Triangle::Intersect 的返回语句前面加上：

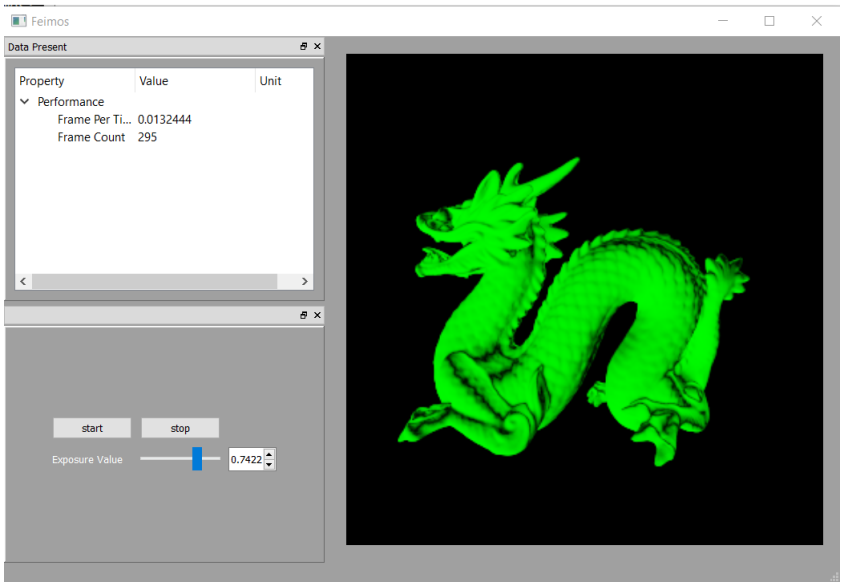
```
//叉积法计算法向量，赋值给交点类Interaction成员变量n
isect->n = Normal3f(Normalize(Cross(p1-p0,p2-p0)));
```

我们并不需要确定求得的法向量是朝模型外的还是朝模型内的（其实正规的模型文件，三角形中三个点的顺序是有要求的，因此按照固定方法求出的法向量就是向外的），渲染的时候要取绝对值。并修改一下渲染的函数：

```
        Vector3f lower_left_corner(-1.5, -1.5, -2.0);
        Vector3f horizontal(3.0, 0.0, 0.0);
        Vector3f vertical(0.0, 3.0, 0.0);
        Point3f origin(0.0, 0.0, -3.0);
        Vector3f Light(1.0,1.0,1.0);
        Light = Normalize(Light);
        for (int i = 0; i < ThreadNum; i++) {
                for (int j = 0; j < ThreadNum; j++) {
                        int offset = (i + ThreadNum * j);
                        float u = float(i + random()) / float(ThreadNum);
```

```
11                          float v = float(j + random()) / float(ThreadNum);
12                          Ray r(origin, (lower_left_corner + u*horizontal + v*
                                vertical) - Vector3f(origin));
13                          float tHit;
14                          SurfaceInteraction isect;
15                          Spectrum colObj(0.0f);
16                          if (agg->Intersect(r, &isect)){
17                                  float Li = Dot(Light, isect.n);
18                                  colObj[1] = std::abs(Li); //取绝对值，防止出
                                        现负值
19                          }
20                          //更新到当前渲染缓冲区或保存
21              ......
22                  }
23          }
```

渲染得到如下结果:



现在我们终于得到比较好看的图片了，不再是一团看不清内部结构的结果了。

五　本书结语

本文内容较为简单，我写作也只用了一天时间。

现在你已经可以自己随便渲染一点有趣的东西了，在我们已有引擎的基础上进行各种修改，我们就可以更加熟悉 PBRT 的各种类和函数。下一节我们介绍相机系统，主要是针孔相机模型，包括透视相机和正交投影相机。全景相机很简单，暂时用不到所以我就不讲了。真实感相机过于复杂，且 [1] 书中写得比较详细，所以我也不再赘述。

最后，感谢读者能够坚持读完这本小书，谢谢！

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.