

# Ray Tracer Week1 Report

范昊翀

dad9929

## 1 渲染简单的球

这是 RayTracer 的第一份报告，之前因为 Games101 结束晚了，于是这个进度又被连累了。不过还好这几天成功速通了，只拖了一天！现在的状态是边渲染 Final 图片的高画质版本，一边写这份报告，看看谁更快，现在渲染进度是 (1410011/5760000)。

根据书中指示，首先我对每个发射出去的 ray 的 y 方向值映成一个颜色，作为背景色，因为接下来我们要在背景中加入各种各样的球，很多光线的终点都是射到背景上，并返回背景色，这算是我们光线追踪的最基础的部分，忘截图了，姑且看接下来的图片吧！



图 1: 图片中出现奇怪的噪点

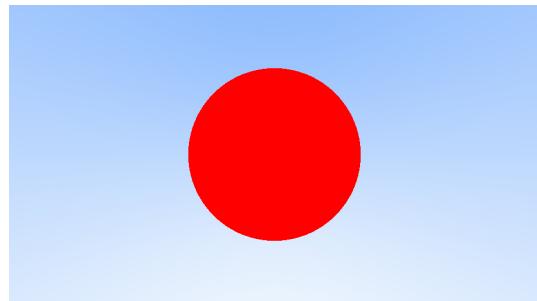


图 2: 噪点消失了

接下来，我加入了一个红色的球，这是通过计算 ray 是否与这个球相交得到的。如果相交，返回球的颜色，否则返回背景色。

在出图的过程中，我发现球的边缘部分总是有奇怪的噪点，而且这些噪点都是以 8\*8 为一团的，感觉很有规律。我检查下来我给像素赋的值要么是白色要么是红色，按理不会出现这样的颜色，已经尝试过在花之前，先把画布全染成白色，但还是会产生产生这样的模糊，说明这些噪点是运行后产生的。在咨询助教后，发现下发程序的导出 jpeg 默认质量是 60，我把它设为 100 之后，噪点便消失了。



图 3: 渐变冷色日本国旗

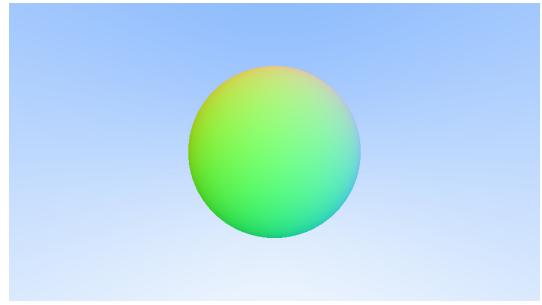


图 4: 渐变黄绿日本国旗

接着，我们不想让这个球整个都是红色的，因此考虑效仿 Games101 中的方法，根据点在球上的位置对颜色赋值，上图是两张效果图，挺好看的。

## 2 使用 Trait 实现继承

实现到这一步时，书中出现了 C++ 的继承和多态，这在 Rust 中并没有简单的类比，我们改用 Trait 来指定带有这一 Trait 的类必须实现的方法。

我们创建一个 hittable 的 Trait，其下有个 scatter 方法，通过引用的方式，“返回”被反射/折射的新 ray。而对于继承 hittable 的 hittable list，我们则遍历其中每个小球，计算离相机最近的一个 scatter，把像素赋成相应的颜色。



图 5: 小球放在大球上



图 6: 加上抗锯齿，Sample = 50

这里我们把一个半径足够大，距离足够远的大球用来模拟地板，我们的程序显然正确地区分了近与远，没有出现透视的状况。随后，我们通过多采样进行抗锯齿，这里我们采用了每像素 50 采样，效果上看，图片变的平滑了许多！

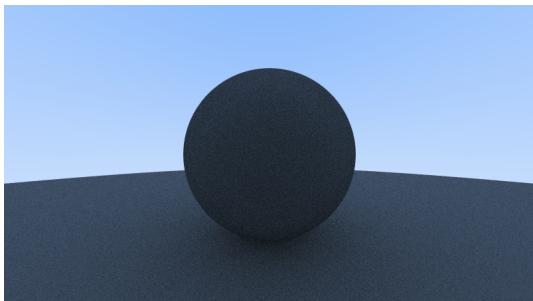


图 7: 无 hit 次数限制

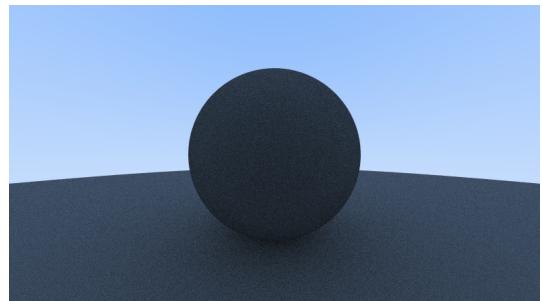


图 8: 限制最大 hit 次数为 10

此前我们的小球的颜色都与环境无关，现在我们不给他们天然地赋一个颜色了，我们要用上背景颜色。我们希望实现的效果是背景光打到小球上，然后射入我们的眼睛里，由于光路可逆，我们可以假装我们发出了光线，经过反射打到背景上获得了颜色。此外，我们希望限制光线反射的次数，因为多次反射之后光强已经很小了，为了节约计算资源，我们把这部分光强忽略不计，由此可以在左图的基础上得到右图。

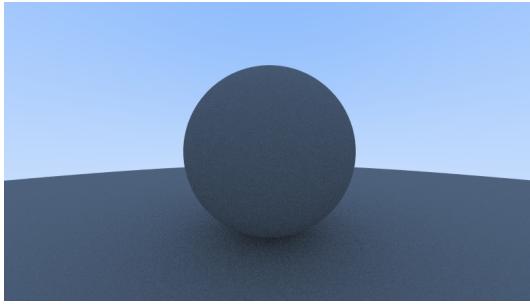


图 9: 修复 Shadow acne

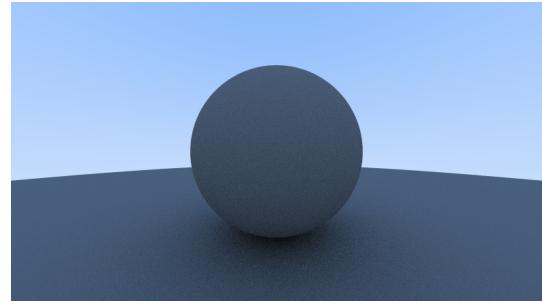


图 10: 修复 Lambertian

接下来是两个小修正：首先，由于浮点数误差，我们 scatter 新得到的 ray 的起始点未必在我们 hit 的点上，比较糟糕的情况是它会跑到球里面去，这样的话会无形中增加反射的次数，为了避免这一点，我们对 ray 的击中时间设了一个下限 0.001，可以发现在修改后，球确实变淡了一些。其次，在真实世界中的光线更倾向于沿着法向 diffuse，因此我们调整随机 diffuse 向量的分布来实现这一点，一个明显的区别是球的影子变深了，因为他们的光线都向正上方 scatter 了，很少会到达相机中。

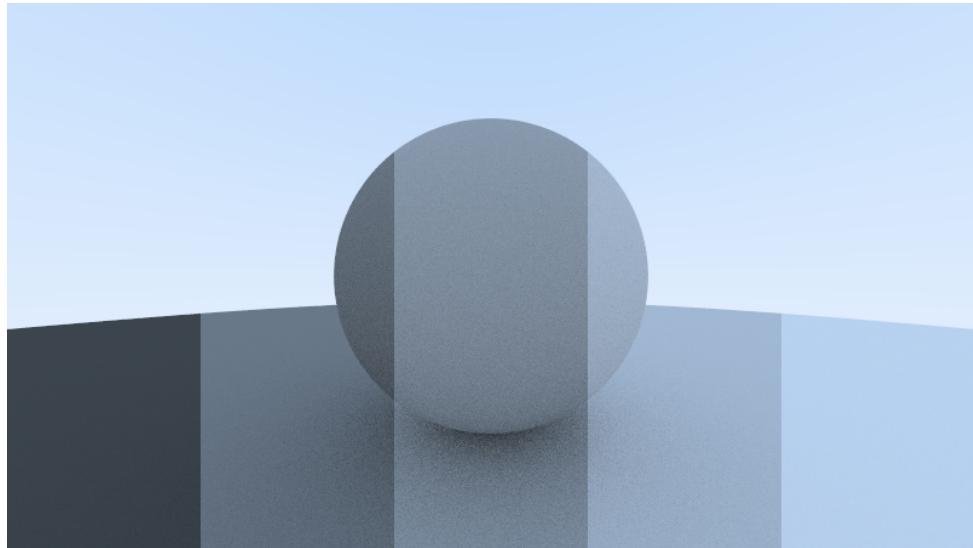


图 11: 设置不同的 gamma 值

最后我们把线性空间中的颜色修正为 Gamma 空间的颜色，从而让我们的电脑可以正确地渲染颜色值！

### 3 使用 $\text{Rc}\langle\text{dyn}\rangle$ 实现基类指针

接下来，我们需要引入更多的球的表面材质，不仅仅限制于 diffusion material。因此，我们需要每个球里面存一个自己的 material 类型，这当然可以通过定义不同的球的类型实现，比如分别定义 metal sphere, glass sphere 等等，但这样太不优雅了。我们考虑把 material 作为一个 Trait，用多态实现各个 material 的方法。然而又有一个问题，我们预先并不知道这个球里面存的 material 种类是什么！在搜索资料后，我通过  $\text{Rc}\langle\text{dyn}\rangle$ ，使用类似存一个基类指针的方法避免了问题。

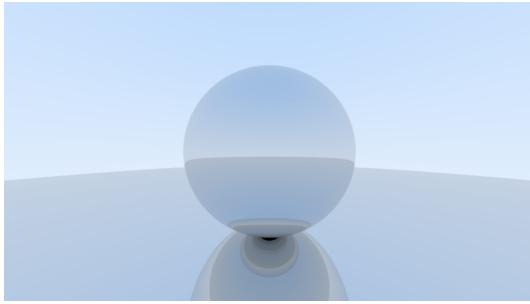


图 12: 两个铁球互相反射

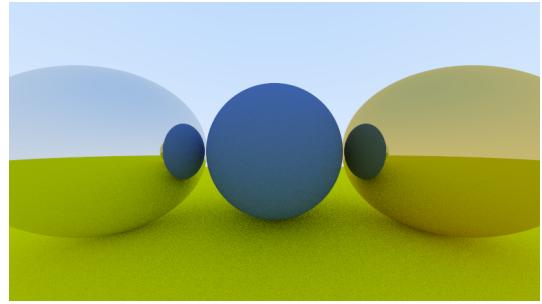


图 13: 加入更多球

在正确地实现了 metal 的 material 类型后，我按照书中例子进行了试验，渲染结果符合预期，如上图所示。

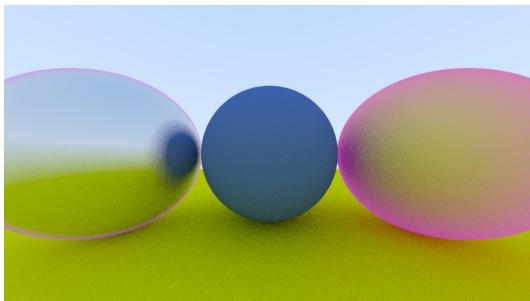


图 14: 长了粉毛的金属球

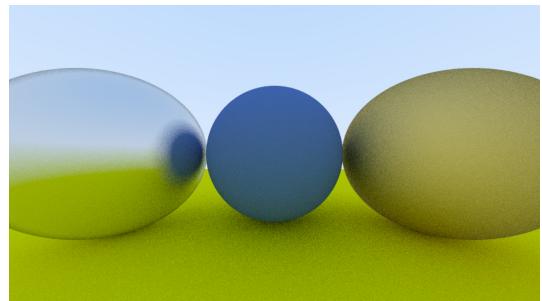


图 15: 粗燥反射的金属球

此外，metal 还额外需要一个表面的毛糙值，左图是一个错误的渲染，我的调试信息忘删了，导致如果没有 scatter 会返回一个品红色的 ray\_color。改掉之后，结果为右图所示，可见金属球的表面反射确实变的模糊了一些。

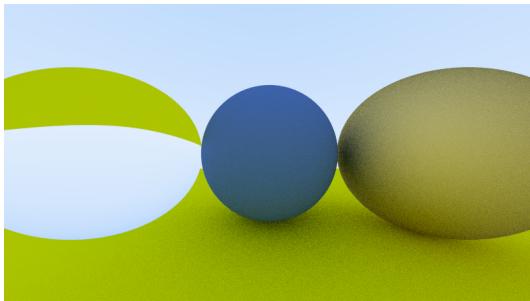


图 16: 朴素的折射

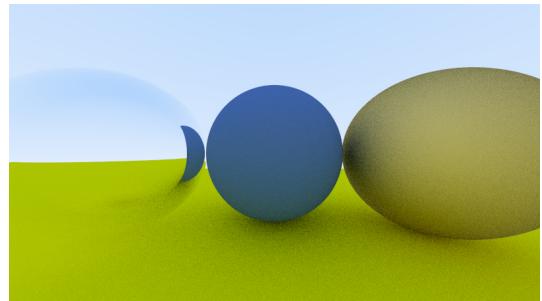


图 17: 考虑全反射

接着我们引入玻璃材质，玻璃的特性比较丰富，首先玻璃会折射光线，因为光线在两个介质间的

折射率不同，实现折射后得到左图。而当光线几乎水平射入时，Snell 定律给出的方程可能无解，此时发生全反射，光线不衰弱，得到右图。

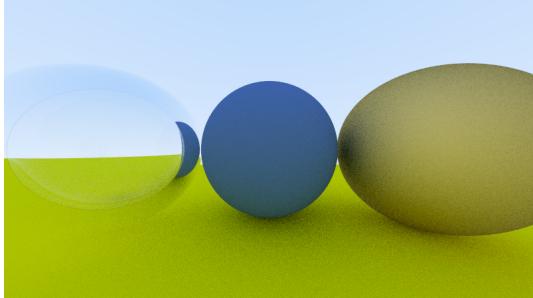


图 18: Schlick 修正

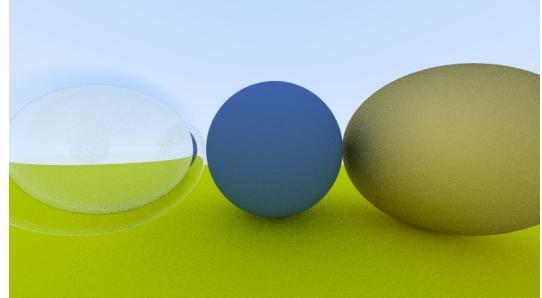


图 19: 中空玻璃球

在现实世界中，玻璃的折射率与入射角度是有关的，我们考虑采用 Schlick 近似来简洁地实现这一点，效果其实不是太明显。接着综合我们学到的有关玻璃球的特性，我们最终在玻璃球中间挖一个洞，测试我们的折射，全反射的成果，渲染图如右图所示。

## 4 移动我们的相机

在这一部分我们解放我们固定的相机，调整他的位置，朝向，以及各个高级的摄影设置，以此使它更符合现实世界的相机。

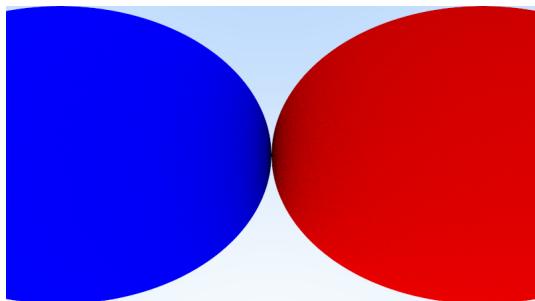


图 20:  $\text{FOV} = 90$

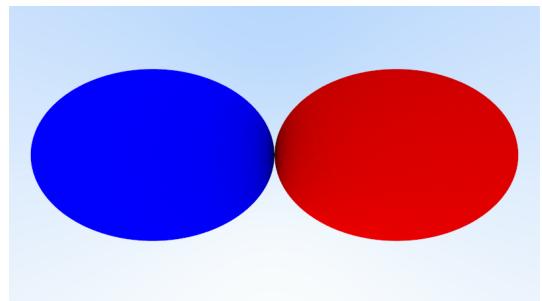


图 21:  $\text{FOV} = 120$

首先保持相机不动，加入相机的视野功能，以此模拟广角与特写功能。

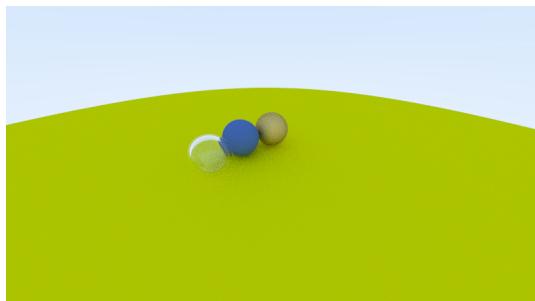


图 22: 远景  $\text{FOV} = 90$

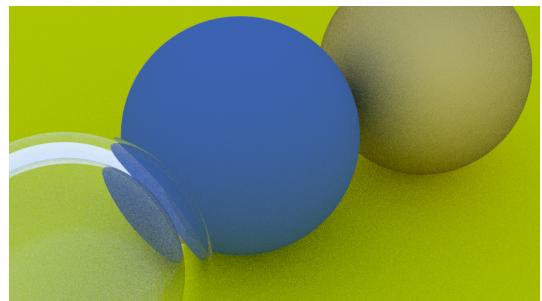


图 23: 近景  $\text{FOV} = 20$

接着我们移动相机的位置，通过缩放给我们刚刚的三个球特写！

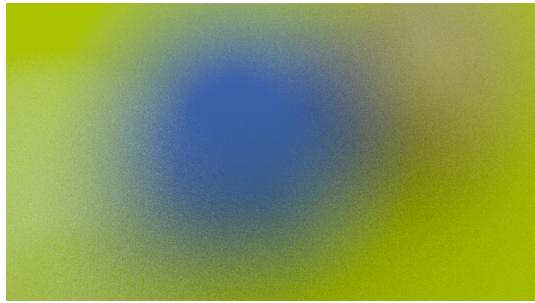


图 24: 写得有问题, 像是有重度散光

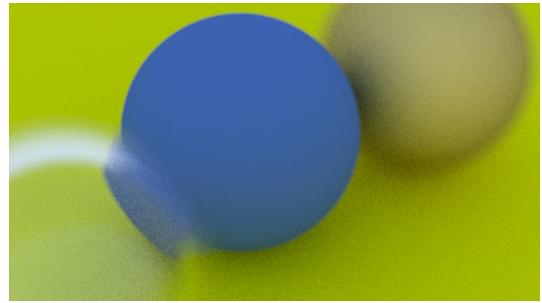


图 25: 调焦的正确实现

最后我们实现调焦和景深 (?)，更符合真实世界相机的行为。

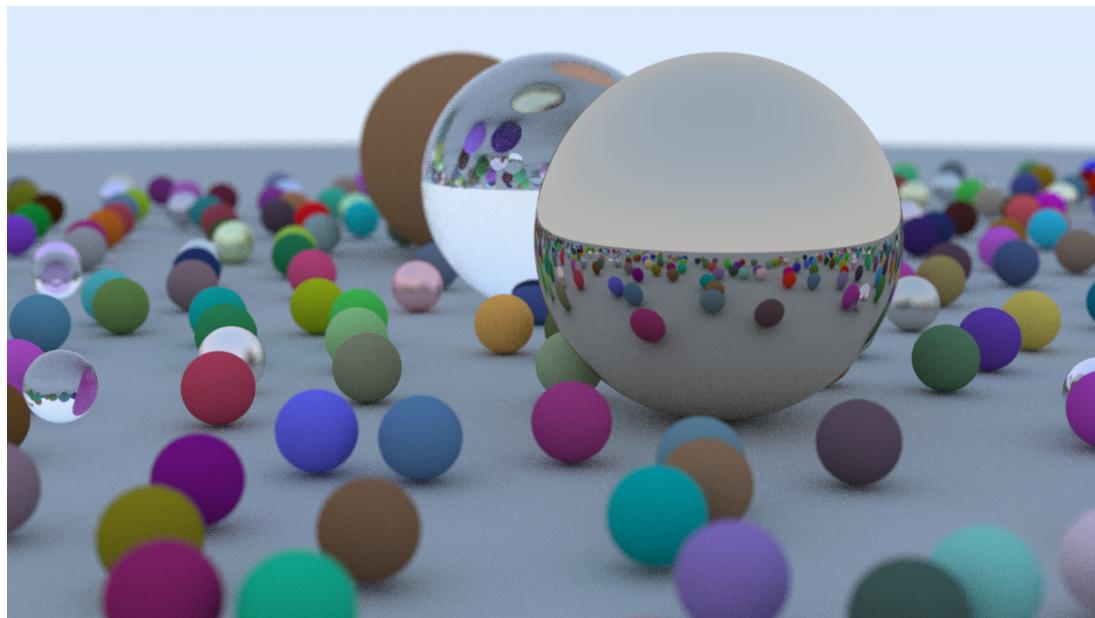


图 26: 比较模糊的最终成品图

至此，我们已经可以进行简单的场景渲染了，当然，我的代码具有很强的可扩展性，因为使用了 Trait 的缘故，要加入新的形状或表面材质是比较简单的。报告先交了，不过我正在尝试学习 Games101 中实现一个 texture 材质，可支持自定义贴图的球！

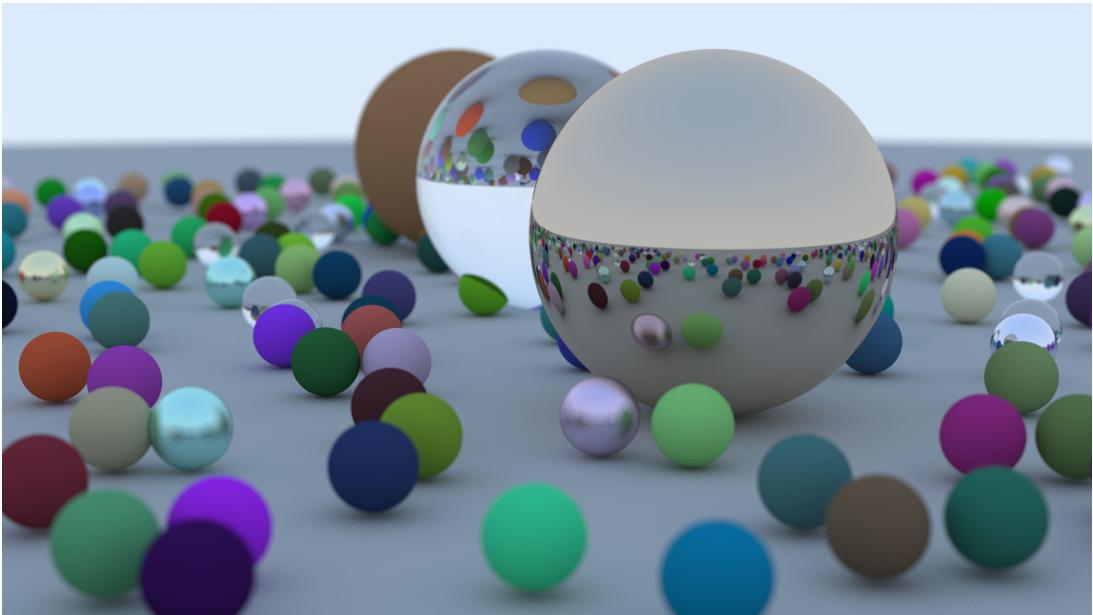


图 27: 高质量的最终成品图

Update: 渲染了一个晚上出了一张高质量的最终图, 每像素取样 100 次, 最大深度设为了 20, 尺寸为 3200\*1800