

# CG-Project2

---

## 概述

---

本次计算机图形学大作业要求编写一个路径追踪渲染器。

由于本人在之前学习过[Ray Tracing in One Weekend](#), 所以本项目的结构与逻辑多少与其会有类似之处。本项目用到了[tiny obj loader](#)用来载入OBJ与MTL文件, 使用了[stb image](#)用来存储输出结果的图像。

除了路径追踪, 本人还实现了纯粹漫反射的双向路径追踪。

## 路径追踪

---

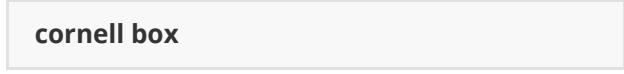
该路径追踪算法大致如下:

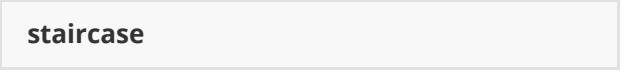
- 1.遍历所有像素以用路径追踪算法计算该像素的颜色值, 每像素采样N次以获得更接近真实的结果。
- 2.计算当前像素在相机前的虚拟位置, 以该位置为终点, 从相机处射出一条光线。若该光线未能与任何物体相交, 则该次采样结果为0; 若与物体相交, 则以相交点为参数递归计算路径追踪结果; 若与光源相交, 则返回该光源的radiance。
- 3.递归计算路径追踪结果。
  - 3.1若该点材质为漫反射材质, 将分别计算直接光照与间接光照。对每个光源进行点的均匀采样, 并将采样点与当前点相连, 若光路可见, 则直接光照的结果加上该光源的radiance乘以BRDF、两点的COS项与光源面积, 除以两点距离的平方。间接光照则以俄罗斯转盘的方式选择是否继续生成新的光线。以该点的法向为y轴构造法向空间, 在半球上随机生成COS分布的立体角方向, 往该方向射出新的光线。若该光线未能与任何物体相交, 则该次采样结果为0; 若与物体相交, 则以递归计算相交点的路径追踪结果, 并乘以BRDF与COS项, 除以随机概率与俄罗斯转盘通过率的乘积; 若与光源相交, 同样返回0。
  - 3.2若该点材质为半透明材质, 则以菲涅尔反射的公式计算出光线反射的概率, 以该概率随机决定光线反射或折射。在决定后, 计算相应新光线的方向, 往该方向射出新的光线。若该光线未能与任何物体相交, 则该次采样结果为0; 若与物体相交, 则以递归计算相交点的路径追踪结果, 并乘以材质的透过率; 若与光源相交, 则以光源的radiance乘以材质的透过率。
- 4.在所有像素计算完毕后, 对整体图像作伽马值为2.2的伽马矫正得到最终结果图像。

光线求交使用BVH来加速。

在生成随机立体角方向时, 本人阅读了[Using the Modied Phong Reflectance Model for Physically-Based Rendering](#), 本打算使用Phong模型的BRDF分布来生成随机方向以获得更好的质量。但是当材质的shiness项过大时, 概率P极易超过float的最小表示精度, 使得P变为0, 造成除0错误, 所以只能使用COS分布来生成方向。

下面为使用路径追踪算法, 每像素采样128次得到的结果:

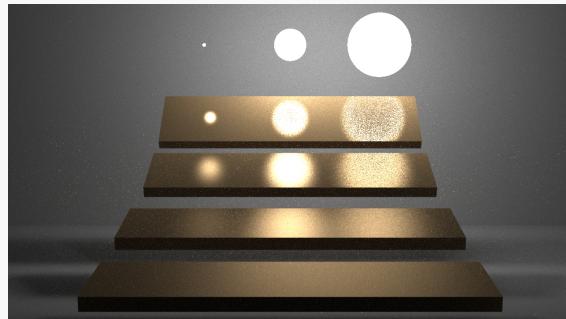
 cornell box

 staircase

**cornell box****staircase**

292mins

352mins

**veach mis****veach bidir**

29mins

153mins

## 双向路径追踪

为了实现透明材质的双向路径追踪，单项路径追踪的光路概率不可或缺。但在将单项路径追踪的概率加入MIS后，整体画面会变得过于明亮，在犄角处又会出现神秘的亮点。



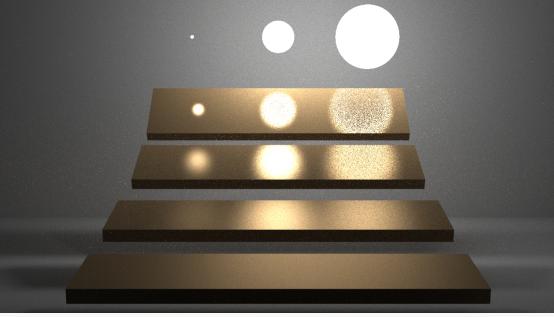
因此在调试许久仍未解决后，本项目放弃了透明材质的双向路径追踪实现。

纯漫反射的双向路径追踪的算法如下：

1. 获得像素光线的方法同路径追踪。
2. 在计算双向路径追踪时，分别从相机与各个光源处以俄罗斯转盘方式，按COS分布随机生成新的光线。在两者停止生成新的光线（未通过俄罗斯转盘或未能与物体相交）后，计算两条光路合并后的结果。
3. 计算光路结果时，将使用路径复用的方法，从两条光路中生成许多完整的光路。
  - 3.1 枚举相机光路与光源光路中的点，每次将两者相连并检验可见性。若光路可见，则计算该光路的BRDF与G项的总乘积。
  - 3.2 为了得到更好的结果并消除路径复用带来的结果偏性，将使用MIS来计算概率。对于一条光路来说，生成该光路的方式不止一种。只要枚举所有生成该光路的概率，以MIS的公式计算得到最终的概率即可。接下来只要将总乘积除以概率便可获得该光路的结果。
4. 在所有像素计算完毕后，对整体图像作伽马值为2.2的伽马矫正得到最终结果图像。

下面为使用双向路径追踪算法，每像素采样128次得到的结果：

<b>cornell box</b>	<b>staircase(无法收敛版本)</b>
--------------------	--------------------------

<b>cornell box</b>	<b>staircase(无法收敛版本)</b>
	
487mins	753mins
<b>veach mis</b>	<b>veach bidir (无法收敛版本)</b>
	
59mins	379mins

## 路径追踪与双向路径追踪的比较

根据上面的结果可以看到，同样采样128次，双向路径追踪虽然有更好的结果，但是计算的总时间远远大于路径追踪。下面来简单比较一下，渲染相同时间，路径追踪与双向路径追踪的质量如何。

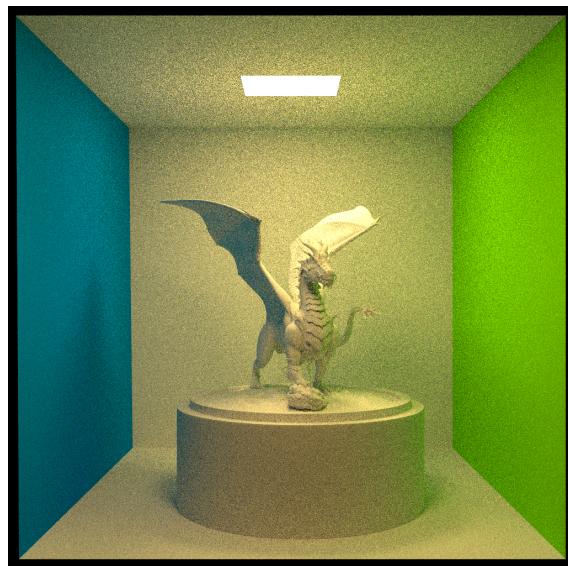
<b>路径追踪</b>	<b>双向路径追踪</b>
-------------	---------------

路径追踪



64mins | 32SPP

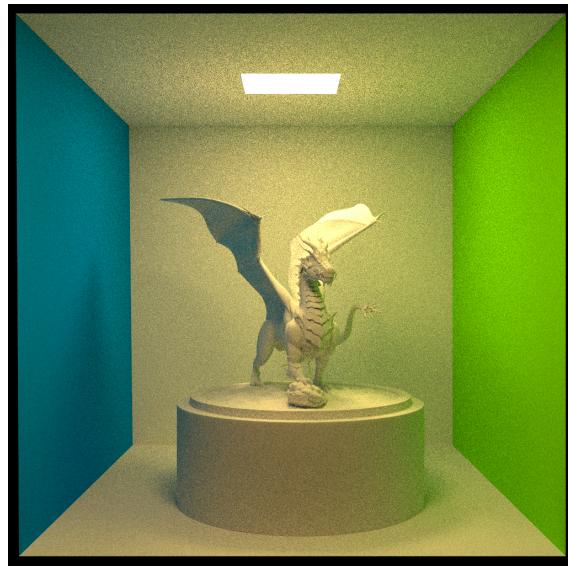
双向路径追踪



62mins | 16SPP



117mins | 60SPP



118mins | 31SPP

可以看到，在相同渲染时间下，双向路径追踪的结果比路径追踪更加明亮，但是由于更少的SPP，产生了更多的噪点。在渲染明亮的场景上，双向路径追踪可能没有多少优势。

另外，路径追踪相比双向路径追踪实现简单，只用一个递归函数即可实现，而双向路径追踪由于其复杂的数学推导调试困难，结构相对没有那么灵活。双向路径追踪唯一的优势可能就是在光路复杂的场景了。

## 总结

由于本人曾经尝试过编写路径追踪渲染器，所以路径追踪编写得相对轻松，完成情况较为理想。但是本人在编写双向路径追踪时遇到了极大的困难，其背后复杂的数学推导以及额外的路径复用和MIS都使得编写的难度加大了很多。

双向路径追踪的调试也变得极其困难：难以定位问题所在；不同SPP下产生的问题各不相同；细节问题倍增。到最后本人也没有完成一个通用的双向路径追踪渲染器。这不仅是因为本人在程序结构设计上的经验不足，还因为没有彻底地搞懂双向路径追踪背后的数学原理。

在本项目结束后，本人将继续学习与编写双向路径追踪，并且学习光子映射等各种真实感渲染的技术。