

可视计算与交互概率 期末大作业报告

一. 选题: C. Rendering 5.Path Tracing

Case1: 实现路径追踪, 一种基于渲染方程的全局光照渲染方法。在界面提供 UI 接口以调整参数和显示进度。使用 bvh 包围盒结构加速。

Case2: 在路径追踪基础上, 通过刻意使用不合理的采样生成艺术纹理效果。

二. 实现思路:

0. 文件结构:

花了比较多的时间理解 lab 框架的博大精深。本作业基于 lab0 的框架（运行请输入 xmake, xmake run lab0), case1 沿用 CaseFixed.cpp 和 CaseFixed.h, 功能实现上添加了 PathTracing.cpp 和 PathTracing.h; case2 沿用 CaseResizable.cpp 和 CaseResizable.h, 功能实现上添加了 Painterly.cpp 和 Painterly.h。

1. 路径追踪:

参考了 smallpt¹和对它的解读²与改进³。原理参考了 GAMES101 Lecture16⁴。

要解渲染方程:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega^+} L_i(p, \omega_i) f_r(p, \omega_i, \omega_o) (n \cdot \omega_i) d\omega_i$$

计算积分使用蒙特卡洛方法。

$$\text{Monte Carlo integration: } \int_a^b f(x) dx \approx \frac{1}{N} \sum_{k=1}^N \frac{f(X_k)}{p(X_k)} \quad X_k \sim p(x)$$

对于每一个着色点, 均匀随机选取一个方向发射一条光, 如果达到光源则为直接光照, 添加光源贡献; 若达到其他物体, 计算间接光照, 递归求交点反射过来的光。

1) 为防止光束数量呈指数级增长, 只随机选取一个方向发射光, 但穿过一个像素

¹ <http://www.kevinbeason.com/smallpt/>

²<https://drive.google.com/file/d/0B8g97JkuSSBwUENiWTJXeGtTOHFmSm51UC01YWtCZw/view?resourcekey=0-BSMES1GmPEtllcv6EgBTjw>

³ <http://www.kevinbeason.com/smallpt/explicit.cpp>

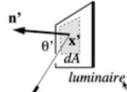
⁴https://www.bilibili.com/video/BV1X7411F744?vd_source=c27f680575ff5da483336a4a7f1602f8&p=16&pm_id_from=333.788.videopod.episodes

可以发射多条光减少噪声。2) 递归终止：使用俄罗斯轮盘赌 RR，随机决定终止。
以一定概率 P 发射光线，得到的结果/P 保证期望不变。3) 提升发射光束的效率
(打到光源)，直接光照从光源采样 (将直接光照和间接光照分开计算)

Sampling the Light

Then we can rewrite the rendering equation as

$$L_o(x, \omega_o) = \int_{\Omega^+} L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \cos \theta d\omega_i \\ = \int_A L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \frac{\cos \theta \cos \theta'}{\|x' - x\|^2} dA$$

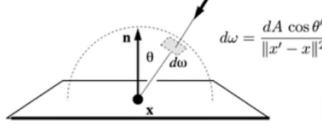


Now an integration on the light!

Monte Carlo integration:

"f(x)": everything inside

Pdf: $1 / A$



路径追踪代码流程如下：

Path Tracing Algorithm

Algorithm 3 Path Tracing Main Loop

```

1: for each pixel (i,j) do
2:   Vec3 C = 0
3:   for (k=0; k < samplesPerPixel; k++) do
4:     Create random ray in pixel:
5:       Choose random point on lens Plens
6:       Choose random point on image plane Pimage
7:       D = normalize(Pimage - Plens)
8:       Ray ray = Ray(Plens, D)
9:       castRay(ray, isect)
10:      if the ray hits something then
11:        C += radiance(ray, isect, 0)
12:      else
13:        C += backgroundColor(D)
14:      end if
15:    end for
16:    image(i,j) = C / samplesPerPixel
17:  end for

```

2. bvh 包围盒结构：

通过构建 bvh 包围盒加速 intersect。具体步骤：首先构建包围盒，由于 aabb 与坐标轴平行，参数只需引入坐标最小的点和最大的点。对当前所有物体先构建大的包围盒，再给物体按照一根坐标轴上的位置排序，在中点进行分割，递归构建子树，直到只剩一个物体。求光线与物体焦点时，先判断光线与包围盒是否相交，如果是叶节点再测试光线是否与球体相交，减少需要求交的物体数。代码编写中特别注意需要创建一个新的索引 indices 记录物体排序结果，不能和物体本身的序号弄错了。

本项目中由于场景比较简单，添加了包围盒结构后速度没有明显提升(甚至觉得更慢了)。因此将带包围盒(finalproject_bvh)和不带包围盒(finalproject)的工程文件都提

交了。

3. 艺术纹理效果

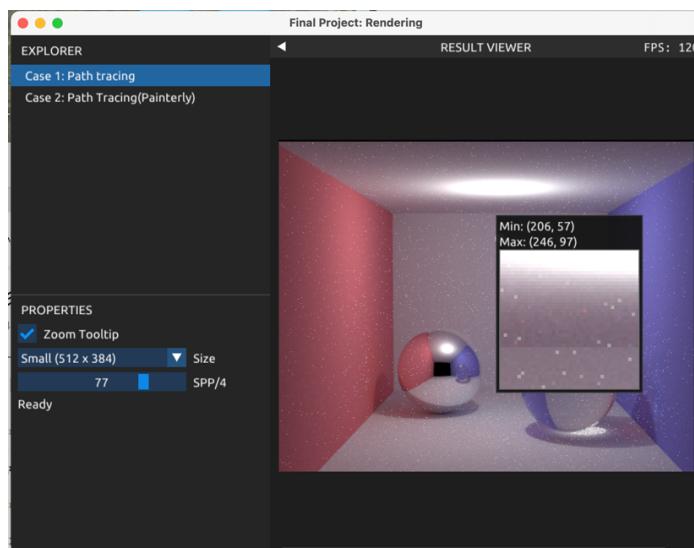
参考 Smallpaint⁵，在路径追踪基础上通过不合理的采样生成艺术纹理效果。研究 smallpaint 的代码可以发现，主要问题在于做路径追踪处理反射的 hemisphere sampling 不合理。hemisphere sampling 用于给定一个表面的法向量，随机采样该法向量上半球中的一个方向。

Smallpt 中的正确做法是：以法线为 z 轴建立局部坐标系，生成两个随机数，使用余弦加权来随机采样半球方向，再转换回世界坐标系。而 smallpaint 中的做法是：以世界坐标系的 z 轴构成标准上半球采样，在直接与法线相加，导致采样并不均匀。case2 中我采用了 smallpaint 中的采样方法，复刻扰动带来的纹理效果。

三 . 效果：

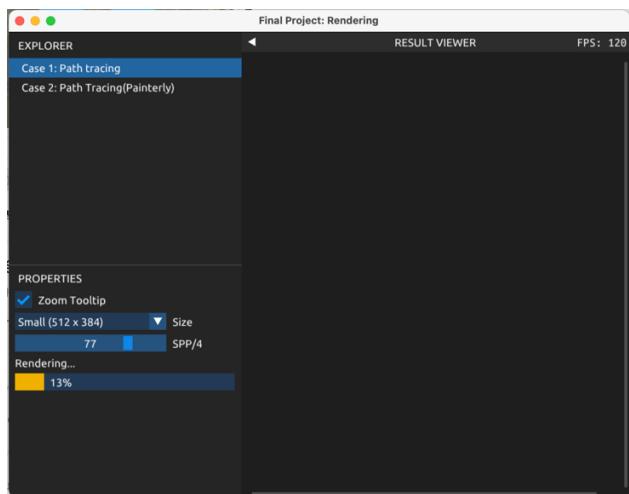
1. UI 交互：

- a) 可以选择图像大小和 SPP，zoom 的功能可以查看细节



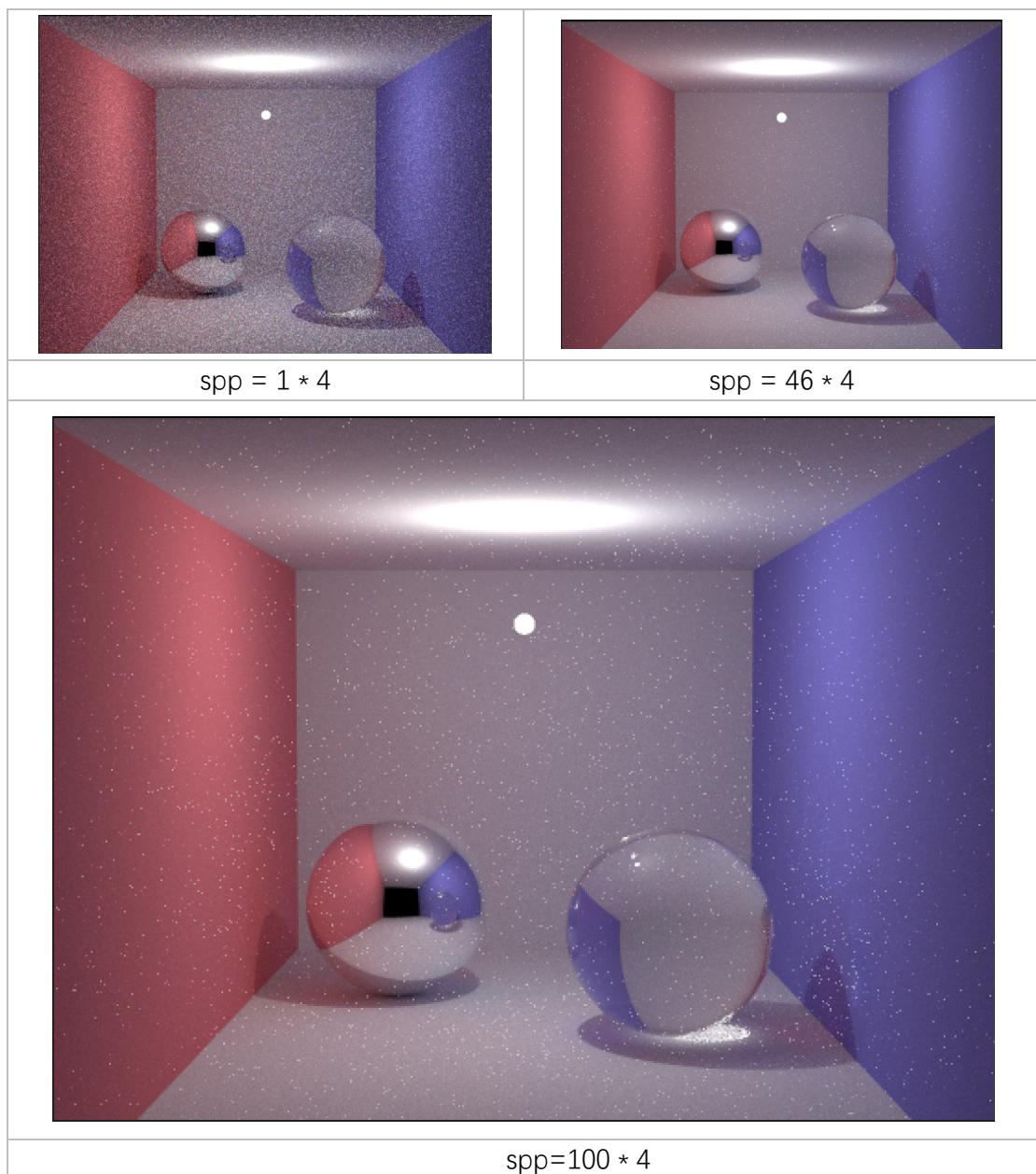
- b) 渲染过程中有进度条提示渲染进度

⁵ <https://users.cg.tuwien.ac.at/zsolnai/gfx/smallpaint/>



2. 渲染结果

a) Case 1: 路径追踪



b) Case 2: 路径追踪 (绘画效果)

