

Tiny Renderer

1 介绍

这是一个用 C++ 实现的路径追踪渲染器，设计遵循 [PBRT](#) 标准，并支持多线程渲染。部分设计参考了 [Mitsuba 3](#)。

该渲染器接受描述场景的 XML 文件作为输入，生成图像并保存至指定路径。

- 加速结构

- 支持层次包围盒（BVH）、KD树（KDTree）和八叉树（Octree）用于光线求交加速。

- 材质支持

- 基本材质：漫反射、导体、介电体、塑料、超薄介电体、粗糙导体、粗糙介电体、粗糙塑料。
 - 特殊材质：掩膜、凹凸贴图、双面材质。
 - 纹理支持：各向异性纹理（exr、hdr、png、jpeg等格式）、常量纹理、棋盘格纹理。

- 光照模型

- 支持任意多个面光源以及天空盒光照。

- 几何形状

- 支持 OBJ 文件定义的网格
 - 内置几何体：立方体、长方形、球体

- 路径追踪

- 使用多重重要性采样，支持对材质以及场景中的光源进行重要性采样。

2 编译运行

测试平台：Windows 11

编译器：Visual Studio 2022

进入项目根目录，打开 cmd 运行以下指令：

```
1 mkdir build
2 cd build
3 cmake .. -G "Visual Studio 17 2022"
4 cmake --build . --config Release
```

exe 文件生成于 [build/src/Release](#)，运行 exe：

```
1 | .\tiny-renderer.exe 'xml relative path of the root directory' -t 'thread count'
```

即可在 xml 同级目录下生成渲染图（png）。

3 代码结构

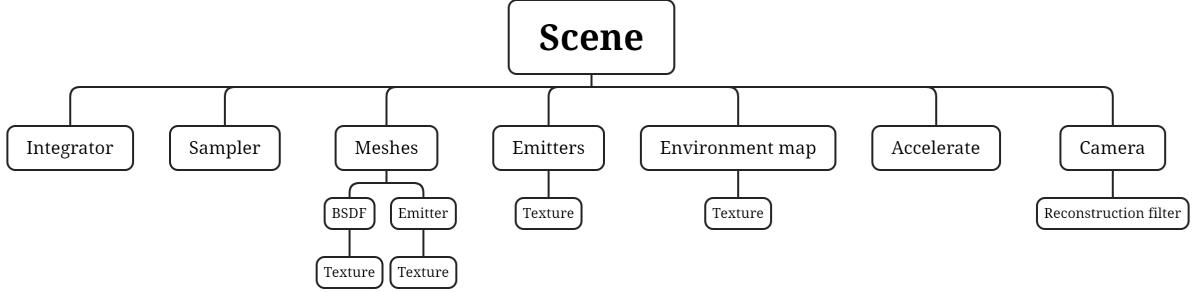
3.1 基本数据类型

基本数据类型位于 `include/core`，主要是模板类，用于定义基本的数据结构以及实现基础的数学库，包括：

- `pcg32`: 随机数生成。
- `array`: 数组，包括向量、点、法向，这三者在进行线性变换时需要区别。
- `matrix`: 矩阵类，包含矩阵的乘法、求逆、转置等常见操作。
- `bounding box`: 包围盒，包含检测、扩展、合并、光线求交等常见操作。
- `distribution`: 对一维概率密度以及二维概率密度进行建模，实现采样、计算概率、重参数化等操作。
- `frame`: 用于世界坐标和局部坐标的变换。
- `fresnel`: 包含导体、介电体的菲涅尔项计算。
- `microfacet`: 微表面模型建模，包含法线分布项、几何遮挡项等。
- `intersection`: 光线与表面交点的数据结构。
- `quad`: 勒让德积分的实现，用于塑料材质次表面散射的建模。
- `ray`: 光线类。
- `record`: 采样数据存储类。
- `spectrum`: 颜色类，目前只实现了 RGB，没有实现光谱。
- `tensor`: 大型张量，存储动态分配，用于材质的存储。
- `timer`: 计时类。
- `transform`: 变换类，包含对向量、点、法向的变换以及透视投影、旋转平移的矩阵构建。
- `warp`: 一系列随机数分布变换函数。

3.2 场景组成

可以由 XML 文件指定实例化的类被称之为组件，代码位于 `include/components`，用于描述场景组件以及包装路径追踪需要用到的工具。所有类均继承基类 `Object`，并通过 `ObjectFactory` 类注册构造函数，以便于读取 XML 文件时实例化组件类并组成包含关系。下面是组件类的包含关系图：



除了环境光照 **Environment map** 外，其余光照 **Emitter** 均依附于几何体 **Mesh**，但是场景 **Scene** 会保留所有光照的共享指针以便于管理光源并做多光源的重要性采样。

4 多重重要性采样算法

4.1 介绍

路径追踪主要依赖递归采样来计算渲染方程。渲染方程可以表示为：

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega} f_r(p, \omega_i, \omega_o) |L_i(p, \omega_i)| \cos \theta_i d\omega_i$$

路径追踪使用蒙特卡洛积分来采样光路并计算积

分。即对于积分 $\int f(x) d\mu(x)$ ，在积分空间下按照概率分布 $X \sim p(x)$ 采样样本 $x_1 \dots x_n$ ，并通过公式 $\frac{1}{n} \sum_{i=1}^n \frac{f(x_i)}{p(x_i)}$ 来近似积分结果。这样计算积分的方差为：

$$\frac{1}{n} \left[\int \frac{f(x)^2}{p(x)} d\mu(x) - \left(\int f(x) d\mu(x) \right)^2 \right]$$
，因此 $p(x)$ 越接近归一化的 $f(x)$ ， $p(x)$ 方差越小。

然而我们很难得到 $f(x)$ 的解析解，此时有两种采样策略：一种是根据 BSDF 采样，一种是根据光源分布采样，一种常见的做法是将漫反射和光源采样分开：

$$\begin{aligned} L_o(p, \omega_o) &= L_e(p, \omega_o) + \int_{\Omega} f_r(p, \omega_i, \omega_o) |L_i(p, \omega_i)| \cos \theta_i d\omega_i \\ &= L_e(p, \omega_o) + \int_{\Omega} f_r(p, \omega_i, \omega_o) (L_e(p', \omega_i) + L_o(p'', \omega_i)) |\cos \theta_i| d\omega_i \\ &= L_e(p, \omega_o) + \int_{\Omega} f_r(p, \omega_i, \omega_o) L_e(p', \omega_i) |\cos \theta_i| d\omega_i + \\ &\quad \int_{\Omega} f_r(p, \omega_i, \omega_o) L_o(p'', \omega_i) |\cos \theta_i| d\omega_i \end{aligned}$$

然而这样做会导致镜面材质无法采样到光源，究其原因在于镜面反射分布是一个 δ 分布，连带着积分是一个 δ 分布的积分，而光源存在一定的面积，很难正好采样在 δ 分布的点上，从而造成采样光源得到的 BSDF 概率密度为 0 的情况。

多重重要性采样（Multiple Importance Sampling, MIS）的思路是结合多种不同的采样方法，分别采样被积函数的不同部分，并将这些采样点结合起来，以达到接近于最优采样的结果。在 MIS 中，为了拟合积分 $\int f(x) d\mu(x)$ 的结果，我们采用 m 种采样策略，每种采样策略采样 n_i 次，MIS 公式可以表示为：

$$\sum_{i=1}^m \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(x_{i,j}) \frac{f(x_{i,j})}{p_i(x_{i,j})}$$

where $w_i(x_{i,j}) = \frac{(n_i p_i(x_{i,j}))^\beta}{\sum_j^m (n_i p_i(x_{i,j}))^\beta}$

那么对于积分 $\int f_r(p, \omega_i, \omega_o) L_e(p', \omega_i) |\cos \theta_i| d\omega_i$, 可以以 p 的概率用 BSDF 采样, 用 $1 - p$ 的概率按照光源采样, 这样可以很大程度上缓解这个问题。多种重要性采样可以表示为:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \hat{L}_s(p, \omega_o)$$

$$\hat{L}_s(p, \omega_o) = \begin{cases} \hat{L}_{s1}(p, \omega_o) = \frac{f_r(p, \omega_i, \omega_o) L_e(p', \omega_i) |\cos \theta_i|}{p(\omega_i)} & \text{if } f_r \text{ is specular at p} \\ \hat{L}_{s2}(p, \omega_o) = \hat{E}(p, \omega_o) + \hat{S}(p, \omega_o) & \text{otherwise} \end{cases}$$

$$\hat{E}(p, \omega_o) = \frac{\hat{E}_I(p, \omega_o)}{p_I(I)} \text{ for } I \in \{1, 2\} \text{ sampled by pdf } p_I$$

$$\hat{E}_1(p, \omega_o) = \frac{f_r(p, \omega_i, \omega_o) L_e(p', \omega_i) |\cos \theta_i|}{p_{emitter}(\omega_i)}$$

$$\hat{E}_2(p, \omega_o) = \frac{f_r(p, \omega_i, \omega_o) L_e(p', \omega_i) |\cos \theta_i|}{p_{bsdf}(\omega_i)}$$

$$\hat{S}(p, \omega_o) = \hat{E}_2(p, \omega_o) = \frac{f_r(p, \omega_i, \omega_o) L_o(p'', \omega_i) |\cos \theta_i|}{p_{bsdf}(\omega_i)}$$

4.2 实现

实现位于 `src/integrator/path.cpp` 的 `li` 函数, 流程如下:

- 光线初始化: 从相机发射一条光线, 沿着视图方向出发, 与场景进行第一次相交计算, 记录交点信息。

```
1 | SurfaceIntersection3f its;
2 | bool is_intersect = scene->get_accel()->ray_intersect(ray, its, false);
```

`its` 记录了交点的各种信息, 包括:

```
1 | PointType p; // 交点位置
2 | Scalar t; // 光线传播距离
3 | NormalType n; // 法向, 如果有法向那就根据法向和重心坐标插值, 没有就三角形两条边叉乘
4 | PointType2 uv; // uv坐标, 有uv就根据uv插值, 没有那就是三角形的重心坐标
5 | FrameType shading_frame; // 法向插值得到的法向
6 | FrameType geometric_frame; // 三角形法向
7 | VectorType wi; // 局部坐标系下的入射光线
8 | VectorType dp_du; // p随u变化率, 用于凹凸贴图
9 | VectorType dp_dv; // p随v变化率, 用于凹凸贴图
10 | uint32_t primitive_index; // 三角形id
11 | std::shared_ptr<Mesh> mesh; // 指向对应mesh的共享指针
```

2. 直接辐射贡献:

```
1 // ----- Direct emission -----
2
3 // If intersect an emitter
4 if (is_intersect && (!its.mesh || its.mesh->is_emitter())) {
5     DirectionSample3f ds(its, prev_si);
6     if (!its.mesh) {
7         ds.emitter = scene->get_environment();
8     }
9     float em_pdf = 0.0f;
10
11    if (!prev_bsdf_delta) {
12        em_pdf = scene->pdf_emitter_direction(prev_si, ds, valid);
13    }
14
15    float mis_bsdf = mis_weight(prev_bsdf_pdf, em_pdf);
16
17    result += throughput * ds.emitter->eval(its, valid) * mis_bsdf;
18 }
```

这里对应公式中的 \hat{E}_2 以及 $L_{s1}(p, \omega_o)$ 中的光源部分。

3. 判断是否要继续弹射:

```
1 bool active_next = depth + 1 < m_max_depth && is_intersect && its.mesh;
2
3 if (!active_next) {
4     break;
5 }
```

如果达到最大深度或者没有相交点时退出路径追踪过程。

4. 光源采样:

```
1 std::shared_ptr<BSDF> bsdf = its.mesh->get_bsdf();
2
3 // -----
4 bool active_em = bsdf->has_flag(ESmooth);
5
6 DirectionSample3f ds;
7 Color3f em_weight;
8 Vector3f wo;
9
10 if (active_em) {
11     std::tie(ds, em_weight) = scene->sample_emitter_direction(its,
12     sampler->next2d(), true, active_em);
13     active_em &= ds.pdf != 0.0f;
```

```

13     wo = its.to_local(ds.d);
14 }

```

在交点处，根据场景所有发光体的分布，选取一个方向并得到采样权重。然后将世界坐标的方向向量转换到交点局部坐标系下，以配合接下来的 BSDF 评估。

5. BSDF 评估与 BSDF 采样

```

1 // ----- Evaluate BSDF * cos(theta) and sample direction -----
2 float sample1 = sampler->next1d();
3 Point2f sample2 = sampler->next2d();
4
5 auto bsdf_val           = bsdf->eval(its, wo, active_next);
6 auto bsdf_pdf           = bsdf->pdf(its, wo, active_next);
7 auto [bsdf_sample, bsdf_weight] = bsdf->sample(its, sample1, sample2,
active_next);

```

前两者评估直接采样光源的 BSDF 值以及对应的概率密度，后者为根据 BSDF 分布重要性采样散射光线的方向、BSDF 值以及概率密度。

6. 多重重要性采样 (MIS) 整合

```

1 // ----- Emitter sampling contribution -----
2 if (active_em) {
3     float mis_em = ds.delta ? 1.0f : mis_weight(ds.pdf, bsdf_pdf);
4     result += throughput * bsdf_val * em_weight * mis_em;
5 }

```

这里对应公式中的 \hat{E}_1 中的光源部分。

7. BSDF 采样得到新方向并更新吞吐量

```

1 // ----- BSDF sampling -----
2 ray = its.spawn_ray(its.to_world(bsdf_sample.wo));
3
4 // ----- Update loop variables based on current interaction -----
5 throughput *= bsdf_weight;
6 eta *= bsdf_sample.eta;
7 valid_ray |= valid && its.is_valid();
8
9 // Information about the current vertex needed by the next iteration
10 prev_si      = its;
11 prev_bsdf_pdf = bsdf_sample.pdf;
12 prev_bsdf_delta = bsdf_sample.delta;

```

这里对应公式中的 \hat{S} 以及 $\hat{L}_{s1}(p, \omega_o)$ 中的散射部分，通过循环采样的方式累加。

8. 俄罗斯轮盘赌 (Russian Roulette)

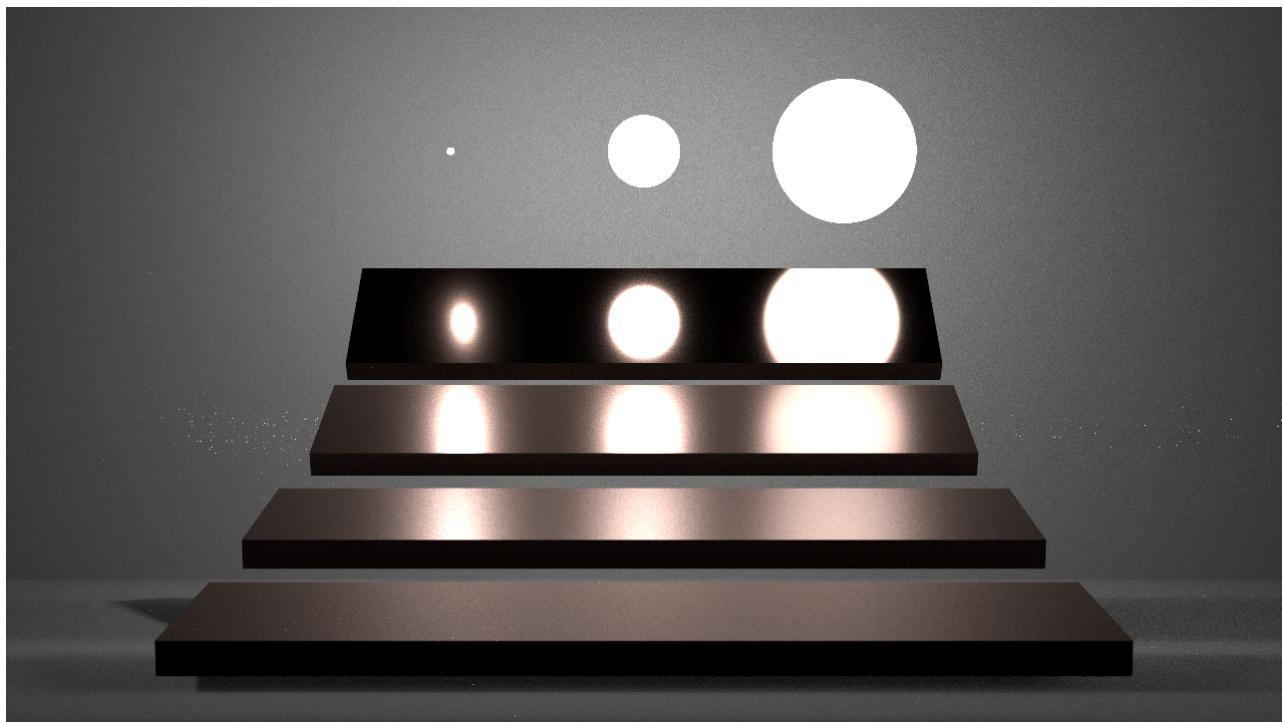
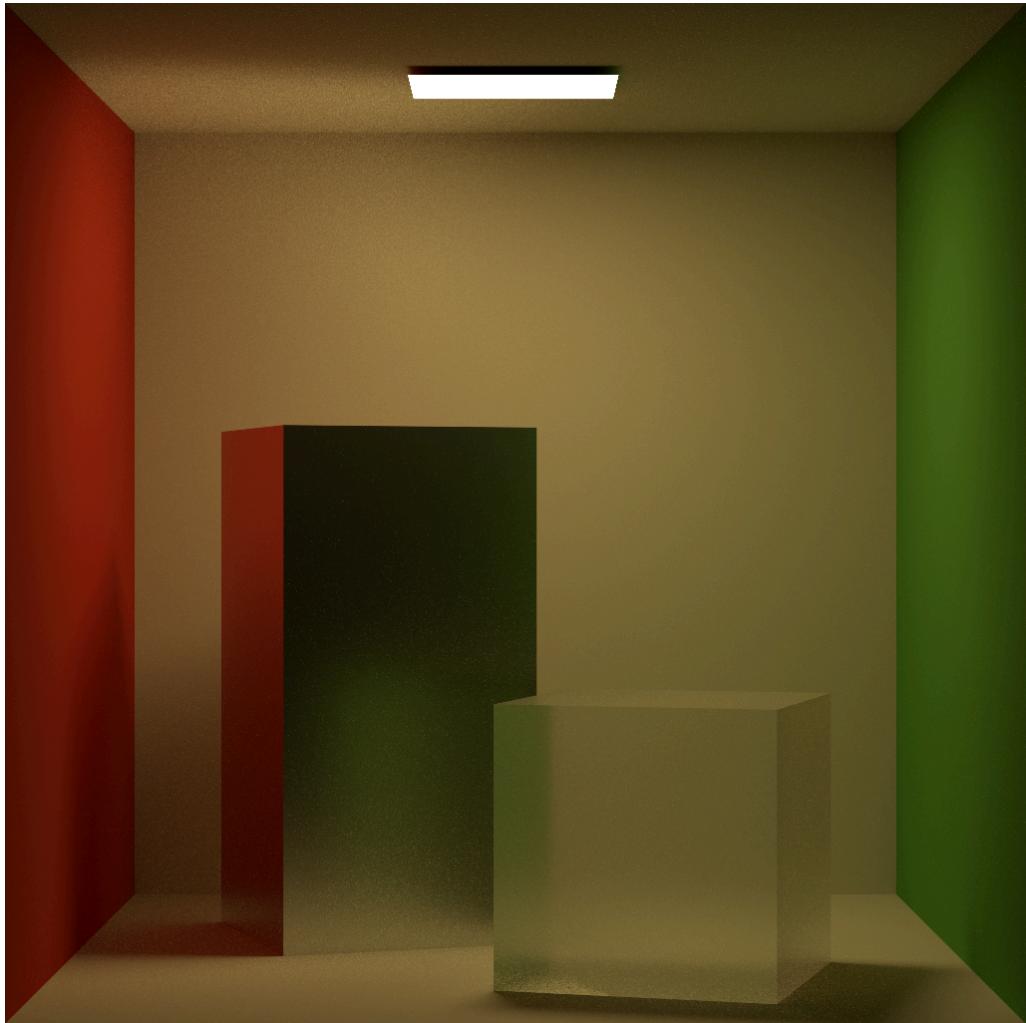
```

1 // ----- Stopping criterion -----
2 depth += 1;
3 float throughput_max = throughput.max_value();
4 float rr_prob        = M_MIN(throughput_max * eta * eta, 0.95f);
5 bool rr_active       = depth >= m_rr_depth;
6 bool rr_continue     = sampler->next1d() < rr_prob;
7
8 if (rr_active) {
9     throughput *= 1.0f / rr_prob;
10}
11
12 valid = (!rr_active || rr_continue) && throughput_max != 0.0f;

```

它的作用是在路径足够长时，随机截断一些能量很弱的路径，从而避免无限递归和无意义计算，同时保证结果无偏。具体而言，其在 `depth >= m_rr_depth`（比如第 5、6 次弹射以后）启用俄罗斯轮盘赌；计算一个继续概率 `rr_prob = min(throughput_max * eta^2, 0.95)`，如果随机数 `sampler->next1d()` 小于这个 `rr_prob`，才继续弹射，否则就终止。若继续了，则要把 `throughput` 再乘上 `1 / rr_prob`，补偿这一随机过程带来的期望下降，保证最终结果不带偏差。

5 结果展示







6 未来更新

1. CUDA并行以及Optix、Embree加速
2. 体渲染：散射介质、相函数以及对应的路径追踪器
3. 更多积分器：双向路径追踪、Metropolis Light Transport
4. 更多材质：法向贴图、BSSRDF（[Position-Free Monte Carlo Simulation for Arbitrary Layered BSDFs](#)）、毛发等
5. 更多类型的几何体，曲面曲线的实现
6. 可交互GUI以及一个简易光栅器作为实时渲染界面
7. 基于GBuffer的光线追踪降噪

7 未修复Bug

1. 图像分辨率不是 2^x 时保存图像会出错。
2. 双层材质 `Smooth` 标签的识别。
3. Clang 编译器下报错。