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。除了使用开源库 stb、tinyexr 用于处理图像读写，以及 pugixml 用于解析 XML 文件，其余功能均由本人独立实现。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
  - 粗糙材质：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、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 专业版

编译器：Visual Studio 2022

进入项目根目录，打开 cmd 运行以下指令：

```
mkdir build
cd build
cmake .. -G "Visual Studio 17 2022"
cmake --build . --config Release
```

exe 文件生成于`build/src/Release`，运行 exe：

```
.\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

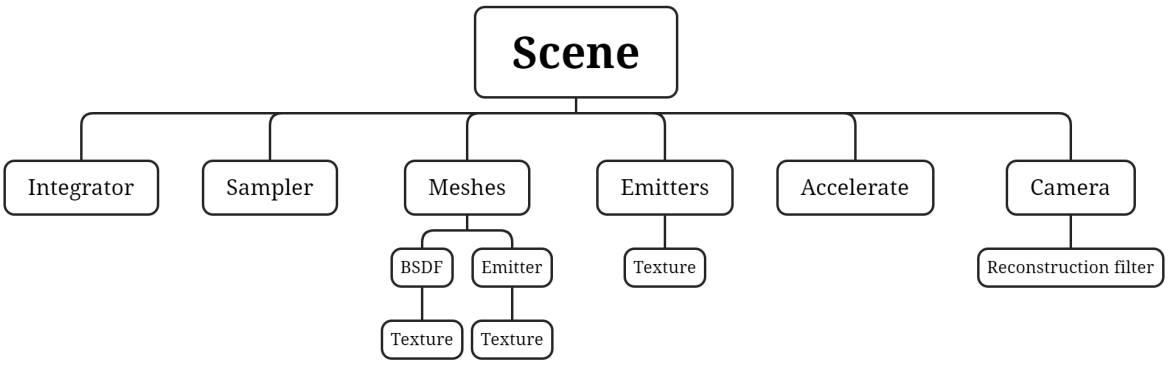
3.1 基本数据类型

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：



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：

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

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$\int f(x)\mathrm{d}\mu(x)$，在积分空间下按照概率分布$X \sim p(x)$采样样本$x_1 \cdots x_n$，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：$\frac{1}{n}\left[\int\frac{f(x)^2}{p(x)}\mathrm{d}\mu(x) - (\int f(x)\mathrm{d}\mu(x))^2\right]$，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：

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

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$\int f(x)\mathrm{d}\mu(x)$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：

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

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

那么对于积分$\int f_r(p,\omega_i,\omega_o)L_e(p',\omega_i)|\cos\theta_i|\mathrm{d}\omega_i$，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。

4.2 实现

```
[[nodiscard]] Color3f li(const std::shared_ptr<Scene> &scene,
std::shared_ptr<Sampler> sampler, const Ray3f &ray_,
                         bool &valid) const override {
    // Configure loop state
    Ray3f ray(ray_);
    Color3f throughput(1.0f);
    Color3f result(0.0f);
    float eta      = 1.0f;
    uint32_t depth = 0;
    bool valid_ray = false;

    // Variables caching information from the previous bounce
    Intersection3f prev_si;
    float prev_bsdf_pdf  = 1.0f;
    bool prev_bsdf_delta = true;

    // Path tracing loop
    for (uint32_t i = 0; i < m_max_depth && valid; i++) {
        // Ray intersect
        SurfaceIntersection3f its;
        bool is_intersect = scene->get_accel()->ray_intersect(ray, its, false);

        // ---------------------- Direct emission ----------------------

        // If intersect an emitter
        if (is_intersect && its.mesh->is_emitter()) {
            DirectionSample3f ds(its, prev_si);
            float em_pdf = 0.0f;

            if (!prev_bsdf_delta) {
                em_pdf = scene->pdf_emitter_direction(prev_si, ds, valid);
            }

            float mis_bsdf = mis_weight(prev_bsdf_pdf, em_pdf);

            result += throughput * ds.emitter->eval(its, valid) * mis_bsdf;
```

```
        }

        // Continue tracing the path at this point?
        bool active_next = depth + 1 < m_max_depth && is_intersect;

        if (!active_next) {
            break;
        }

        std::shared_ptr<BSDF> bsdf = its.mesh->get_bsdf();

        // ---------------------- Emitter sampling ----------------------
        bool active_em = active_next;

        DirectionSample3f ds;
        Color3f em_weight;
        Vector3f wo;

        if (active_em) {
            std::tie(ds, em_weight) = scene->sample_emitter_direction(its, sampler->next2d(), true, active_em);
            active_em &= ds.pdf != 0.0f;
            wo = its.to_local(ds.d);
        }

        // ------ Evaluate BSDF * cos(theta) and sample direction -------
        float sample1   = sampler->next1d();
        Point2f sample2 = sampler->next2d();

        auto bsdf_val                   = bsdf->eval(its, wo, active_next);
        auto bsdf_pdf                   = bsdf->pdf(its, wo, active_next);
        auto [bsdf_sample, bsdf_weight] = bsdf->sample(its, sample1, sample2, active_next);

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

        // ---------------------- BSDF sampling ----------------------
        ray = its.spawn_ray(its.to_world(bsdf_sample.wo));

        // ------ Update loop variables based on current interaction ------
        throughput *= bsdf_weight;
        eta *= bsdf_sample.eta;
        valid_ray |= valid && its.is_valid();
```

```
81
82          // Information about the current vertex needed by the next iteration
83          prev_si         = its;
84          prev_bsdf_pdf   = bsdf_pdf;
85          prev_bsdf_delta = bsdf_sample.delta;
86
87          // -------------------- Stopping criterion --------------------
88          depth += 1;
89          float throughput_max = throughput.max_value();
90          float rr_prob        = M_MIN(throughput_max * eta * eta, 0.95f);
91          bool rr_active       = depth >= m_rr_depth;
92          bool rr_continue     = sampler->next1d() < rr_prob;
93
94          valid = (!rr_active || rr_continue) && throughput_max != 0.0f;
95      }
96
97      return result;
98  }
```

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

1. 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u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

- 如果交点是一个发光体，则直接计算其辐射亮度贡献： Ldirect=Le(x,ωo)L_{direct} = L_e(x, \omega_o)
- 如果当前点不是发光体，需要通过采样周围光源方向来评估直接光照。

1. 间接辐射贡献

- 使用场景的 BRDF 函数从交点采样下一条反射光线方向 ωi\omega_i：Lindirect=fr(x,ωi,ωo)·Li(x,ωi)·∣cosθi∣L_{indirect} = f_r(x, \omega_i, \omega_o) \cdot L_i(x, \omega_i) \cdot |\cos\theta_i|
- 根据蒙特卡洛方法，路径追踪通过递归的方式采样多个光线路径，积累所有路径的光能。

2. 重要性采样与多重重要性采样（MIS）

- 重要性采样：为了减少方差，路径追踪通过对 BRDF 或光源分布进行重要性采样，使得采样的方向更有可能落在高亮区域。
- 多重重要性采样（MIS）：结合 BRDF 采样和光源采样的结果，利用加权函数分配权重以避免偏差： wMIS=pBSDF2pBSDF2+pEmitter2w*{MIS}* = *frac{p*{BSDF}^{2}{p_{BSDF}2 + p*{Emitter}^2} 其中 pBSDFp*{BSDF} 和 pEmitterp_{Emitter} 分别是 BSDF 采样和光源采样的概率密度函数。

3. 路径中止（Russian Roulette）

- 为了避免路径追踪陷入无穷递归，使用俄罗斯轮盘（Russian Roulette）决定是否终止路径。
- 中止的概率与路径的累计通量（throughput）有关，通常在路径达到一定深度后开始应用： Pterminate=1−min(throughput·η2,0.95)P_{terminate} = 1 - \min(throughput \cdot \eta^2, 0.95) 如果路径被终止，则不会再继续计算当前路径的贡献。

4. 累积结果

每条光线路径的贡献通过加权和累积形成最终像素值：

Lo=∑i=1NLpath,ipiL_o = \sum*{i=1}^{N} *frac{L*{path,i}}{p_{i}}

其中 pip_i 是采样的概率密度，NN 是采样的路径数量。

路径追踪通过递归的方式逐步构建光线路径，并通过采样求解积分，使得每条路径的贡献与光能的分布密切相关。MIS 的引入使得路径追踪能够更有效地结合光源采样和 BRDF 采样的优点，从而显著减少图像中的噪声。

5 结果展示