Moer 渲染器使用手册

Moer: Research Oriented Physically Based Renderer

组织：NJUMeta

版本：1.0

目录

第一章 介绍

Moer 是一个基于物理的光线追踪渲染器，由南京大学Meta Graphics & 3D Vision Lab 开发和维护，我们的项目旨在构建一个易于学习和使用的科研导向渲染平台。

Moer 项目目前已经在 github 上开源，同时还有面向教学使用的 Moer-lite版本。Moer 平台已经用于 Meta Graphics & 3D Vision Lab 的科研及教学。

本文档主要介绍 Moer 渲染器的架构以及使用。

第二章 下载与编译

2.1 环境需要

为了成功的从源码构建 Moer，以下系统环境必须满足：

- git：用于项目源代码的获取
- CMake：版本 3.19 及以上，跨平台（Windows/Linux/MacOS）项目构建
- C++ 编译器：需要支持 C++17（g++/MSVC/clang++）

2.2 下载

源代码拉取

Moer 渲染器代码托管在 github 上，项目地址：https://github.com/NJUCG/Moer。
使用以下命令拉取 Moer 源代码及其第三方依赖

```
git clone --recursive https://github.com/NJUCG/Moer.git
```

成功拉取后，项目结构如下

```
Moer
|--ext
  |--FastMath
  |--json
  |--pcg
  |--stb
|--src
  |--CoreLayer
  |--FunctionLayer
  |--ResourceLayer
  |--main.cpp
  |--CMakeLists.txt
|--CMakeLists.txt
|--README.md
|--...
```

submodule 的拉取和更新

Moer 项目使用 git submodule 功能将部分第三方依赖引入项目，如果 submodule 拉取失败或需要进行更新，请执行以下命令

```
git submodule update --init --recursive
```

2.3 项目编译

在项目中新建 build 目录，在 build 目录中调用 cmake 生成当前系统支持的项目构建文件

```
mkdir build
cd build
cmake ..
```

由于部分第三方依赖的下载与配置在该阶段完成，因此该过程需要一定时间。

项目配置文件生成后，在 Linux/MacOS 系统中使用 make 对项目进行编译，在 Windows 系统中使用 Visual Studio 打开生成的.sln 文件进行编译。

编译成功后，项目结构应当如下：

```
Moer
|--ext
|--src
|--CMakeLists.txt
|--README.md
|--target                  //项目编译结果目录
  |--bin                   //可执行二进制文件
    |--Moer_r(Moer_d)      //后缀与构建类型一致
  |--lib                   //二进制第三方库
    |--embree3
    |--Moer-ext_r(Moer-ext_d) //后缀与构建类型一致
    |--stb_r(stb_d)        //后缀与构建类型一致
    |--tinyobjloader(assimp)
    |--...
```

2.4 可配置编译选项

CMAKE_BUILD_TYPE

Release 或 Debug，控制构建类型。

MESH_LOADER

- Tinyobjloader : 使用 tinyobjloader 解析 obj 文件
- Assimp : 使用 assimp 解析 obj 文件

第三章 渲染器功能与接口

Moer 渲染器的可拓展功能模块主要位于 src / FunctionLayer 中，我们为每个功能模块定义了基类，拓展渲染器功能只需实现基类中定义的接口并重新编译项目即可。

几何体

FunctionLayer / Shape

该模块定义了渲染器可以渲染的所有几何体，实现新的几何体时需要继承并实现 **Entity** 基类中定义的接口，其需要实现的虚函数接口如下

```
//负责光线与几何体求交的逻辑，当发生相交时，返回包含交点局部几何信息的Intersection结构体
virtual std::optional<Intersection> intersect(const Ray &r) const = 0;

//返回几何体的表面面积
virtual double area() const = 0;

//在几何体表面等概率地随机采样一个点，返回包含该点局部几何信息的Intersection结构体
virtual Intersection sample(const Point2d &positionSample) const = 0;

//返回当前几何体在世界坐标系内的包围盒
virtual BoundingBox3f WorldBound() const = 0;
```

目前支持以下几何体的渲染

- Cube 长方体
- Curve 曲线
- Mesh 三角形网格
- Quad 平行四边形
- Sphere 球

加速结构

FunctionLayer / Acceleration

该模块定义了渲染器在进行光线——几何求交时使用的加速结构，实现新的加速结构需要继承并实现 **Accel** 基类中定义的接口，其需要实现的虚函数接口如下

```
//负责光线与加速结构求交的逻辑，当发生相交时，返回包含交点局部几何信息的Intersection结构体
virtual std::optional<Intersection> Intersect(const Ray &r) const = 0;

//返回整个加速结构在世界坐标系内的包围盒
virtual BoundingBox3f getGlobalBoundingBox() const = 0;
```

目前支持以下加速结构

- bvh
- EmbreeAccel 集成Intel 的高性能求交库

相机模型

FunctionLayer / Camera

该模块定义了渲染器可以使用的相机模型，实现新的相机模型需要继承并实现 Camera 基类中定义的接口，其需要实现的虚函数接口如下

```
//根据当前像素的位置及输出图片的分辨率生成一条光线用于进行追踪
virtual Ray generateRay(const Point2i &filmResolution,
                        const Point2i &pixelPosition,
                        const CameraSample &sample) const = 0;
```

目前支持以下相机模型

- Pinhole 针孔相机
- Thinlens 薄透镜相机

随机数采样器

FunctionLayer / Sampler

该模块定义了渲染中所使用的随机数生成器，实现新的采样器需要继承并实现 Sampler 基类中定义的接口，其需要实现的虚函数接口如下

```
//开始某个像素的随机数序列
virtual void startPixel(const Point2i &pixelPosition) = 0;

//切换至该像素的下一个采样的随机数序列
virtual void nextSample() = 0;

//获取一个[0-1]内均匀分布的1维的随机数
virtual double sample1D() = 0;

//获取一个[0-1]^2内均匀分布的2维随机数
virtual Point2d sample2D() = 0;
```

目前支持一下随机数采样器

- IndependentSampler 独立采样器
- StratifiedSampler Stratified 采样器
- HaltonSampler Halton low-discrepancy 采样器

光源

FunctionLayer / Light

该模块定义了渲染器支持的光源模型，实现新的光源模型需要继承并实现 Light 基类中定义的接口，其需要实现的虚函数接口如下

```
//环境光需要实现该方法，计算给定光线方向上的radiance值
virtual LightSampleResult evalEnvironment(const Ray &ray) = 0;

//可以被击中的光源需要实现该方法，根据击中点和入射方向计算给定光线方向上radiance值
virtual LightSampleResult eval(const Ray &ray, const Intersection &its, const Vec3d &d) = 0;

//在光源上采样一个发光点和光线传播方向，双向方法会使用到
virtual LightSampleResult sampleEmit(const Point2d &positionSample, const Point2d &directionSample)
    = 0;

//对给定表面击中点its采样直接光照
virtual LightSampleResult sampleDirect(const Intersection &its, const Point2d &sample) = 0;

//对给定介质散射点mRec采样直接光照
virtual LightSampleResult sampleDirect(const MediumSampleRecord &mRec, Point2d sample) = 0;
```

目前支持以下相机模型

- DiffuseAreaLight 漫反射面光源
- InfiniteSphereLight 环境光源
- PointLight 点光源

积分器

FunctionLayer / Integrator

该模块定义了渲染器求解渲染方程所使用的算法，实现新的积分器需要继承并实现 Integrator 基类中定义的接口。对于基于蒙特卡洛的渲染算法，只需继承并实现 MonteCarloIntegrator 基类中定义的接口，其需要实现的虚函数接口如下

```
//给定一条光线ray，求解当前场景下该光线携带的能量
virtual Spectrum Li(const Ray &ray, std::shared_ptr<Scene> scene) = 0;
```

目前支持以下积分器

- NormalIntegrator 法线可视化
- PathIntegrator 路径追踪积分器
- VolPathIntegrator 介质路径追踪积分器
- GuidedPathIntegrator path guiding 路径追踪

BxDF 散射模型

FunctionLayer / Material / BxDF

该模块定义了物体的表面散射模型，实现新的散射模型需要继承并实现 BxDF 基类中定义的接口，其需要实现的虚函数接口如下

```
//该BxDF模型在给定出射方向out的前提下，对入射方向进行采样
virtual BxDFSampleResult sample(const Vec3d &out, const Point2d& sample) const = 0;

//该BxDF模型在给定入射、出射方向上的双向散射分布函数的值
virtual Spectrum f(const Vec3d &out, const Vec3d &in) const = 0;

//该BxDF模型在给定出射方向out的前提下，采样到入射方向in的pdf值
virtual double pdf(const Vec3d &out, const Vec3d &in) const = 0;
```

目前支持以下 BxDF 模型

- LambertainBxDF Lambert 漫反射模型
- MirrorBxDF 镜面反射模型
- ConductorBxDF 光滑金属散射模型
- RoughConductorBxDF 粗糙金属散射模型
- DielectricBxDF 光滑电介质散射模型
- RoughDirlectricBxDF 粗糙电介质散射模型
- PlasticBxDF 塑料散射模型

纹理

FunctionLayer / Texture

该模块定义了二维平面上的纹理模型，实现新的纹理模型需要继承并实现 Texture 基类中定义的接口，其需要实现的虚函数接口如下

```
//根据交点处的局部几何信息求出交点处对应的纹理值（类型由Tvalue决定）
virtual Tvalue eval(const Intersection &intersection) const = 0;
```

目前支持的纹理模型如下

- ImageTexture 图像纹理
- Checkboard2D 二维棋盘纹理
- ConstantTexture 常量纹理

介质

FunctionLayer / Medium 该模块定义了渲染器支持渲染的介质模型，实现新的介质模型需要继承并实现 Medium 基类中定义的接口，其需要实现的虚函数接口如下

```
//给定光线，采样光线在介质中的传播距离，如果采样光线直接离开介质则返回false
virtual bool sampleDistance(MediumSampleRecord *mRec,
  const Ray &ray,
  const Intersection &itsOpt,
  Point2d sample) const = 0;

//给定起点和终点，计算起点到终点线段上的光学透过率
virtual Spectrum evalTransmittance (Point3d from, Point3d dest) const = 0;
```

目前支持的介质模型如下

- BeerslawMedium 均质介质（只吸收不散射）

- HomogeneousMedium 均质介质

第四章 场景文件格式

Moer 目前支持以 json 文件作为场景的表述方式。

介质的配置

场景中的所有介质在 json 文件中均声明在 mediums 列表中，若场景中不需要介质则 mediums 列表为空即可

```
"mediums" : [...],    //mediums列表
```

Beerslaw 介质

光线在 beerslaw 介质中传播时只会被吸收而不会被散射，其配置如下

```
{
  "type" : "beers_law", //介质类型
  "name" : "medium_name", //介质名称，使用时按名引用
  "intensity" : [5, 5, 5] //介质密度，默认值为[1.0, 1.0, 1.0]
}
```

Homogeneous 介质

光线在 homogeneous 介质中传播时既会被吸收，也会散射，其配置如下

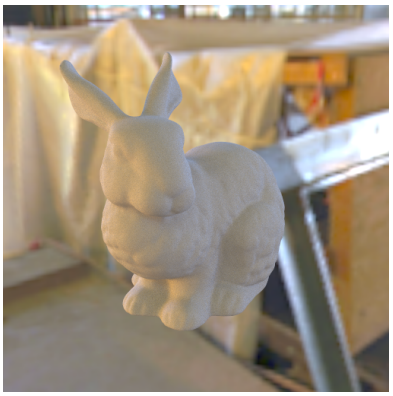
```
{
  "type" : "homogeneous", //介质类型
  "name" : "medium_name", //介质名称，使用时按名引用
  "sigmaT" : [5, 5, 5], //介质的衰减系数，默认值为[0.1, 0.1, 0.1]
  "albedo" : [0.9, 0.9, 0.9]//介质的反射率，默认值为0.8
}
```

材质的配置

介质中所有材质在 json 文件中均声明在 materials 列表中

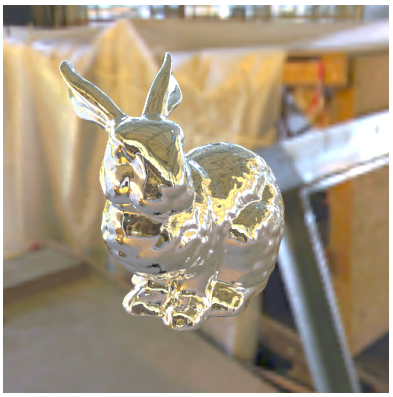
Lambert 漫反射材质

```
{
  "type" : "lambert",      //材质类型
  "name" : "material_name", //材质名称，使用时按名引用
  "albedo" : [0.5, 0.5, 0.5] //漫反射反射率，默认为[1.0, 1.0, 1.0]
}
```



Mirror 镜面材质

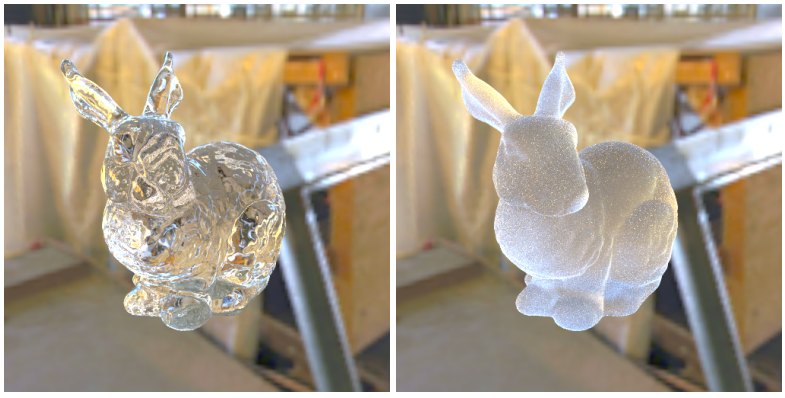
```
{
  "type" : "mirror",       //材质类型
  "name" : "material_name" //材质名称，使用时按名引用
}
```



Dielectric 电介质材质

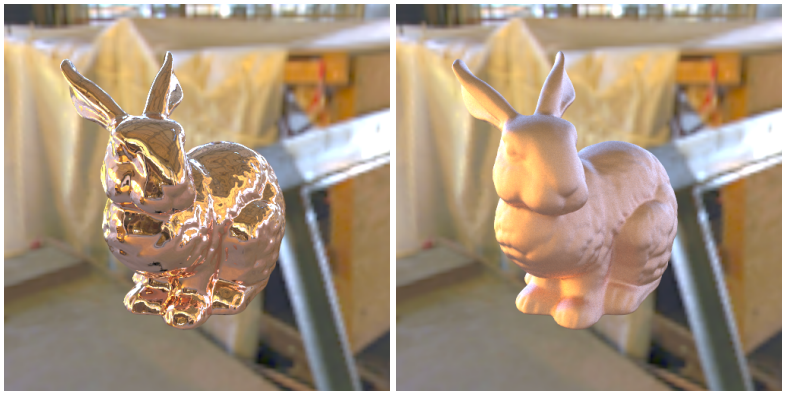
```
{
  "type" : "dielectric",          //材质类型
  "name" : "material_name",       //材质名称，使用时按名引用
  "ior" : 1.33,                   //折射率，默认为1.33
```

```
  "albedo_reflection" : [1, 1, 1], //反射时能量的衰减比例，默认为[1.0, 1.0, 1.0]
  "albedo_transmission" : [1, 1, 1], //透射时能量的衰减比例，默认为[1.0, 1.0, 1.0]
  "roughness" : 0.5                //各向同性的粗糙程度，默认为0
}
```



Conductor 金属材质

```
{
  "type" : "conductor",            //材质类型
  "name" : "material_name",        //材质名称，使用时按名引用
  "albedo" : [1, 1, 1],            //反射时能量的衰减比例
  "conductor" : "Cu",              //金属类型，具体见ComplexIor.cpp文件
  "roughness" : 0.5                //各向同性的粗糙程度，默认为0
}
```



Plastic 塑料材质

```
{
  "type" : "plastic",              //材质类型
  "name" : "material_name",        //材质名称
  "ior" : 1.33,                    //折射率，默认为1.5
  "albedo" : [0.7, 0.7, 0.7],      //漫反射反照率，默认为[0.5, 0.5, 0.5]
  "roughness" : 0.1                //各向同性粗糙程度
}
```

Hair 毛发材质

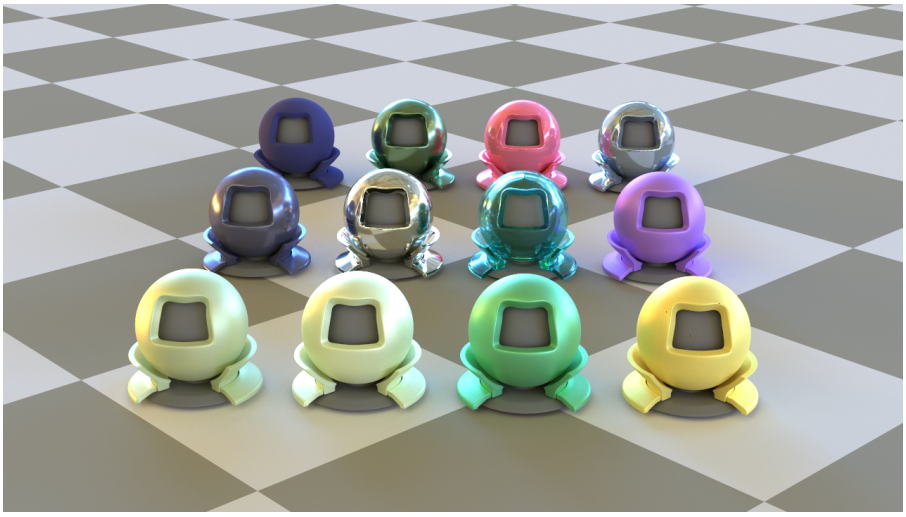
```
{
  "type": "hair",           //材质类型
  "scale_angle": 2.5,       //纵向散射倾斜角
  "melanin_ratio": 1,       //黑色素比例
  "melanin_concentration": 1.3, //黑色素纠正因子
  "beta_m": 0.2,            //纵向散射粗糙度
  "beta_n":0.2              //横向散射粗糙度
}
```

Disney 材质

Disney 材质的配置如下

```
{
  "type": "disney"              //材质类型
  "name": "material_name",      //材质名称
  "base_color" : [0.82, 0.67, 0.16], //材质基色, 默认为[1.0, 1.0, 1.0]
  "specular_transmission": 0.0,  //默认为[0, 0, 0]
  "metallic": 0.0,              //材质的金属程度, 默认为0
  "subsurface": 0.0,            //材质的次表面散射程度, 默认为0
  "specular": 0.5,              //材质的光滑程度, 默认为0
  "roughness": 0.5,             //材质的粗糙程度, 默认额外0
  "anisotropic": 0.0,           //材质的各向异性程度
  "sheen": 0.0,                 //纺织物边缘的明亮效果 该项代表权重
```

```
  "sheen_tint": 0.5,              //sheen分量的颜色向基本颜色靠拢的程度
  "clear_coat": 0.0,              //清漆权重
  "clear_coat_gloss": 0.5,        //清漆光滑度
  "eta": 1.5,                     //折射率
}
```



几何体的配置

场景中的所有几何体在 json 文件中均配置在 entities 列表中

变换矩阵

变换矩阵用于对几何体进行平移、旋转和缩放操作，其配置如下

```
"transform" : {
  "rotation" : [0, 45, 90], //物体分别沿x、y、z坐标轴旋转的角度
  "position" : [0, 0, 1],   //物体分别沿x、y、z坐标轴平移的距离
  "scale" : [1, 1, 1]       //物体分别沿x、y、z坐标轴缩放的比例
}
```

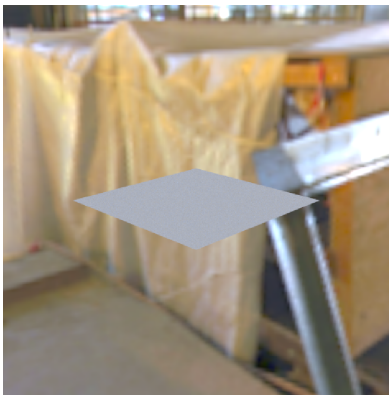
物体材质

物体的材质用于决定几何体的表面光学属性，配置时按名引用 materials 列表中已经声明的材质

```
{
  "materials" : [
    { "name" : "material_1",...}
  ],
  "entities" : [
    { "material" : "material_1",...}
  ]
}
```

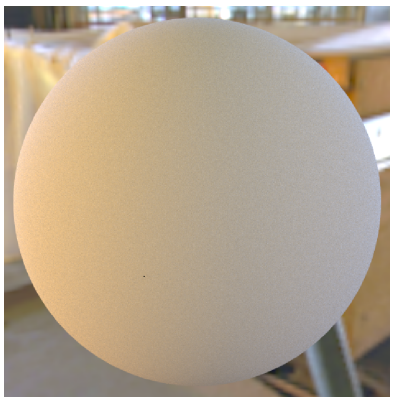
Quad 平行四边形

```
{
  "type" : "quad",     //几何体的类型
  "base" : [0, 0, 0], //平行四边形的一个顶点，默认为[0, 0, 0]
  "edge0" : [1, 0, 0], //平行四边形的一条边，默认为[1, 0, 0]
  "edge1" : [0, 0, 1], //平行四边形的另一条边，默认为[0, 0, 1]
  "transform" : ...,   //变换矩阵
  "material" : ...     //该几何体的材质
}
```



Sphere 球

```
{
  "type" : "sphere",   //几何体的类型
  "center" : [0, 0, 0], //球心，默认为[0, 0, 0]
  "radius" : 1.0       //球的半径，默认为1.0
  "transform" : ...,   //变换矩阵
  "material" : ...     //该几何体的材质
}
```



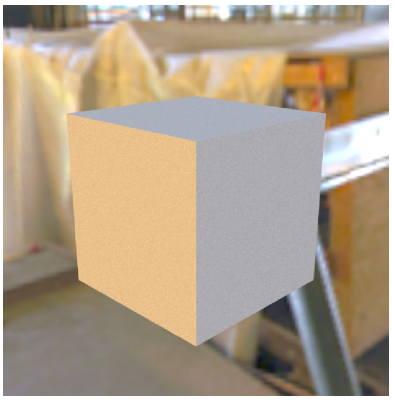
Mesh 三角形网格

```
  {
    "type" : "mesh",   //几何体的类型
    "file" : "xxx.obj", //三角形网格文件的相对路径（对于scene.json文件所在文件夹）
    "transform" : ..., //变换矩阵
    "material" : ...   //该几何体的材质
  }
```

Cube 正方体

```
{
  "type" : "cube",     //几何体的类型，默认是中心在原点，棱长为1的正方体
  "transform" : ...,   //变换矩阵
  "material" : ...     //该几何体的材质
```

```
}
```



Curves 曲线

毛发物体可以用曲线进行描述，其配置如下

```
{
  "type": "curves",          //几何体类型
  "file": "xxx.fiber",       //曲线文件的相对路径（对于scene.json文件所在文件夹）
  "curve_thickness": 0.05,   //曲线宽度
  "curve_taper": true        //是否让曲线逐步变细
}
```

光源的配置

场景中的所有光源同样配置在 entities 列表中，在几何体的配置相中增加 emission 一项

环境光

Moer 将环境光建模为映射到半径为无穷大的球面上的贴图，其配置如下

```
{
  "type": "infinite_sphere", //几何体的类型，无穷大的球面
  "emission": "xxx.hdr",    //环境光贴图的相对路径（对于scene.json文件所在文件夹）
}
```

漫反射面光源

面光源作用在一个几何体的表面，其配置如下

```
{
  "type" : "xxx",          //几何体的类型
  "emission" : [10, 10, 10], //面光源的强度
  ...                      //几何体的其他配置
}
```

点光源

```
{
  "type" : "point_light", //光源的类型
  "position" : [1, 0, 1], //点光源的位置
  "intensity" : [10, 10, 10] //点光源的强度
}
```

相机的配置

针孔相机

```
"camera" : {
  "type" : "pinhole",     //相机的类型
  "resolution" : [512, 512], //输出图像的分辨率
  "fov" : 39,             //相机的视角
  "transform" : {
    "position" : [0, 1, 4],
    "up" : [0, 1, 0],
    "look_at" : [0, 1, 0]
  }                        //相机的look_at矩阵
}
```

渲染配置

在 renderer 中配置采样率和输出图像的名称

```
"renderer" : {
  "output_file" : "xxx", //输出图像为xxx.hdr
  "spp" : 64             //图像的采样率
  }
```