Lab-1

图形绘制技术

Introduction and Prerequisite

Lab1-复杂光源的实现

2024 春季学期

1. 恒定功率光源

1.1 现实世界中的光源功率

在渲染器中，我们使用了“某些颜色”来代指光源发出的光。这些颜色的 RGB 往往是大于 1 的，这就导致它们在图片中看起来很亮。然而，在现实生活中，我们并不知道任意光源的“某些颜色”。我们可以知道的是，光源的功率越大，光源的发光效率越高，光源就越亮，其可以照亮更远处的东西。

在现实世界中，光源一般是通电的，电力为其提供的功率有一部分转移至内能，剩余部分作为光线的能量在空间中扩散。对于一些类型的光源，比如白炽灯，其在电功率很高的情况下依然难以提供很高的亮度，因为这一类光源的发光效率不高，绝大部分能量都转化为了内能；而对于 LED，其发光效率极高，几乎所有的能量都变成了光线。我们可以通过查询光源的功率以及光源的发光效率来得到光源的发光功率。

在之前的课程中，你应该已经了解了如何推导碟形均匀光源的发光功率。这就为渲染世界中的光源与现实世界中的光源筑起了桥梁。在这次实验中，你需要实现一类光源，这类光源指定了发光功率，与现实世界中一致。

1.2 需要完成的部分

在 Moer-lite 现在的代码中，由光源获取的“某种东西”使用了一个含混不清的词，energy。实际上，在对光源进行采样时，我们获取的“某种东西”均为 radiance。然而，在现实世界中，我们往往难以得知某种光源的 radiance，但是可以很方便的获取光源的功率。因此，在这一部分的作业中，你将要实现一个“拥有着某种功率”的光源。

在 areaLight.cpp 中，你可以看到如下的初始化面光源的代码：

```
AreaLight::AreaLight(const Json &json) : Light(json) {
    type = LightType::AreaLight;
    shape = Factory::construct_class<Shape>(json["shape"]);
    energy = fetchRequired<Spectrum>(json, "energy");
}
```

你需要将其中的实现改为类似这种形式：

```
AreaLight::AreaLight(const Json &json) : Light(json) {
    type = LightType::AreaLight;
    shape = Factory::construct_class<Shape>(json["shape"]);

    // 保证energy与power不同时存在，且必定存在一个
    energy = fetchOptional<Spectrum>(json, "energy", 0.0f);
    power = fetchOptional<Spectrum>(json, "power", 0.0f);

    if (!energy.isZero())
    {
      // do nothing
    }
    else // if (!power.isZero())
    {
```

```
        // 将power转化为energy
        // write your code here ...
        // energy = ...
    }
}
```

其中，fetchRequired 接口会从 json 文件中获取一个必定存在的条目，而 fetchOptional 接口会从 json 文件中获取一个可选的条目。如果这一条目不存在，则 fetchOptional 会返回其输入的第三个参数，也就是默认值。

你需要完成上述代码中将 power 转化为 energy 的部分。如果你依然对这部分内容存在疑惑的话，请参考上课时讲过的，关于如何计算 disk 光源发光功率的内容。

为了完成这部分代码，你可能需要在 shape.h 中添加额外的接口获取某种形状的表面积：

```
class Shape : public Transformable {
public:
    ...
    virtual float getArea() const {return 0.0f;}
    ...
};
```

在添加了 getArea 接口后，你需要实现各种不同形状的 getArea 方法。但是，**为了完成这一次作业，你只需要实现平行四边形的 getArea 方法即可**。

1.3 测试

我们会在压缩包中提供额外的场景 area-lights-power，同学们也可以自行搭建场景进行渲染。

在完成了上述实现之后，你应该能够得到一张如下图片：

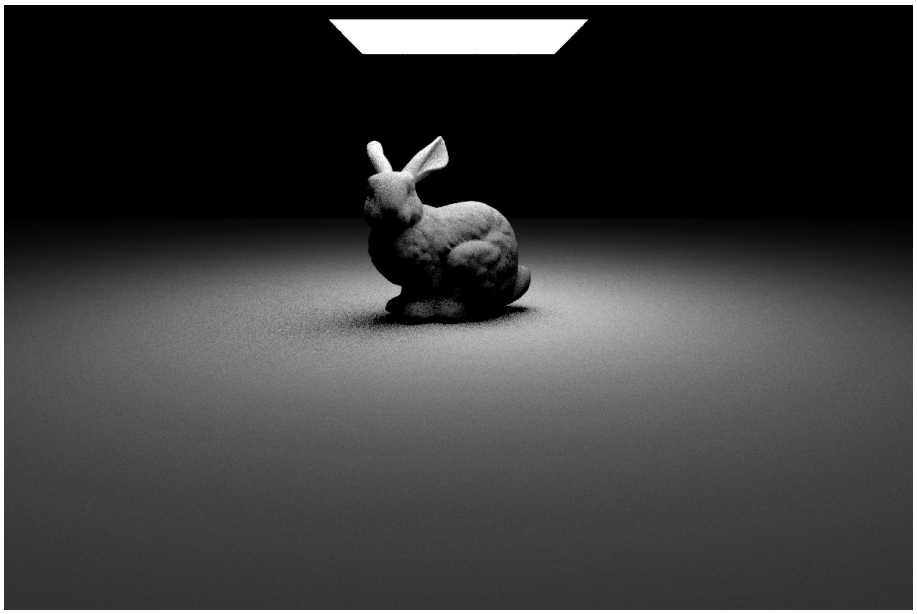

图 1: area-lights-power 场景的正确结果

请把你得到的这一图片不经修改地贴进你的实验报告中。

你可以尝试通过 energy 设置光源和通过 power 设置光源来感受两者的区别。在通过 energy 设置光源时，光源越大，其功率越高，场景就会越亮；在通过 power 设置光源时，更大的光源不会导致场景变得更亮。

2. 球面光源

2.1 球面的均匀采样

对于任意一种面积光源，渲染器中都需要描述如何在其表面进行采样。这一采样一般是均匀的，表面上所有的点的概率密度函数 pdf 完全一致，同时也可以是非均匀的。在这一节，为了实现球面光源，你需要完成一个在球面上均匀采样的接口。

一个球由一个中心位置和一个半径决定。在实际实现中，我们可以先忽略球的中心位置和半径：只要能够在单位球上进行均匀采样，一个简单的变换就可以将这一样本转移到任意的球面上。

单位球上的一点由两个角度构成，方位角 $\theta \in [0, 2\pi)$ 和俯仰角 $\phi \in [-\frac{1}{2}\pi, \frac{1}{2}\pi]$。我们仅需对这两个角度进行采样，即可得到单位球上的随机点。

然而，如果我们在有效范围内均匀地随机生成 θ 和 ϕ，我们会发现生成的样本靠近球的两极。实际上，俯仰角 ϕ 无法通过在角度上随机生成得到均匀的分布。

一个直观的理解是，所有俯仰角为 89 度的点（北极附近）构成了一个集合，这一集合构成一个圆；所有俯仰角为 0 度的点（赤道）构成了另一个集合，这一集合同样也是一个圆。如果俯仰角是均匀采样得到的，那么，采样到前一个圆环的概率比采样到后一个圆环的概率相同。但是，由于后一个圆环比前一个圆环长很多，采样到它的概率也应该大很多。

2.2 需要完成的部分

为了完成这一部分的作业，你需要在 Sphere.h 中实现如下接口，这一接口应当已经提供了签名。这一接口根据 sample 在球面上均匀地产生某一点以及对应的法向量，以及表面上的一些其余附加信息。

```
virtual void uniformSampleOnSurface(Vector2f sample,
                                     Intersection *intersection,
                                     float *pdf) const override {
    /*
    sample: 两个[0,1]之间的随机数;
    intersection: 作为返回值使用，至少需要填写position和normal两个域;
    pdf: 作为返回值使用，表示在这一物体表面上采样的概率密度函数，以面积计。
    */
    // write your code here ...
    return;
}
```

注意，intersection 中的 position 需要考虑球的中心位置和半径。

2.3 测试

我们在压缩包中额外提供了场景 cornell-box-sphere，同学们也可自行搭建场景进行渲染。

在完成了上述实验后，运行这一场景，你应该可以得到下面的图片：

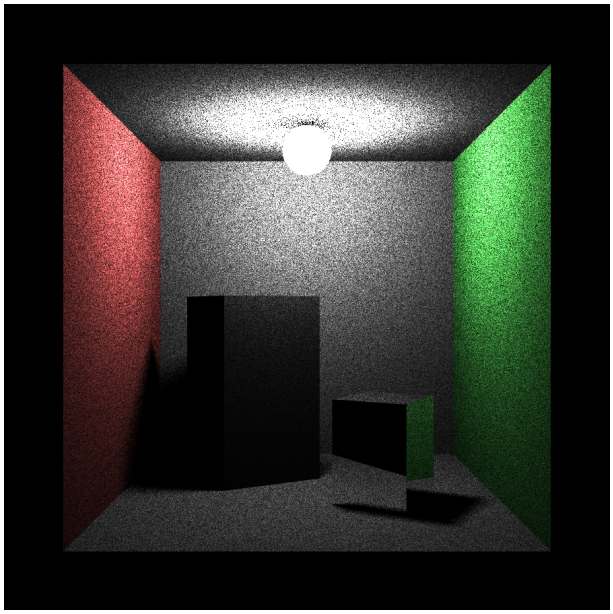

图 2: cornell-box-sphere 场景的正确结果

请把你得到的这一图片不经修改地贴进你的实验报告中。你可以适当地增加 spp 来得到更好的视觉效果。

3. 三角形网格光源

3.1 三角形网格均匀采样

与球面光源类似，为了实现三角形网格光源，你需要完成在三角形网格上进行均匀采样的方法。

三角网格（Mesh）是计算机图形学的基础，通常情况下，场景中的几乎所有物体都是通过三角网格表示的。在计算机中，一个三角网格可以通过如下数据结构进行表示：

- 三角网格中所有顶点位置的列表；
- 三角网格中所有顶点法线的列表；
- 三角网格中所有顶点纹理坐标的列表，本次实验不需要；
- 三角网格中每个面的三个顶点所对应的索引，索引可以用于上述三个列表。由于不同的三角形面之间可以共用顶点，因此这一索引列表是必须的。

3.2 需要完成的部分

与球面光源类似，你需要在 Triangle.h 中完成 TriangleMesh::uniformSampleOnSurface 接口，注意并不是 Triangle::uniformSampleOnSurface 接口。这一接口应当已经提供了签名。

```
virtual void uniformSampleOnSurface(Vector2f sample,
                                     Intersection *intersection,
                                     float *pdf) const override {
    /*
    sample: 两个[0,1]之间的随机数;
    intersection: 作为返回值使用，至少需要填写position和normal两个域;
    pdf: 作为返回值使用，表示在这一物体表面上采样的概率密度函数，以面积计。
    */
    // TODO finish this
    return;
}
```

3.3 实用信息

三角网格的数据结构。

在 TriangleMesh 结构体中提供了指向 MeshData 的智能指针。MeshData 的结构如下所示：

```
struct MeshData {
public:
  int faceCount;                      // 该Mesh三角形面的个数
  int vertexCount;                    // 该Mesh顶点的个数
  std::vector<Point3f> vertexBuffer;  // 存储该Mesh的所有顶点
  std::vector<Vector3f> normalBuffer; // 存储该Mesh的所有法线
  std::vector<Vector2f> texcodBuffer; // 存储该Mesh每个顶点的纹理坐标
  std::vector<std::array<DataIndex, 3>>
      faceBuffer; // 存储该Mesh每个面（即三角形）的三个顶点对应数据的索引

  //* 将一个.obj文件加载到内存中
  //* 请确保加载的.obj文件中只包含一个Mesh
  static std::shared_ptr<MeshData> loadFromFile(std::string filepath);
};
```

三角形采样方法。

你可能需要计算一个三角形的面积。在先前的实验中，你可能已经添加了平行四边形的 getArea 方法。此时你需要实现 TriangleMesh 的 getArea 方法，然后调用。

在已知顶点的情况下，一个三角形的面积可以通过很多方法计算。Moer-lite 已经提供了向量库，你可以考虑如何使用向量计算三角形的面积。

每一个小三角形被采样到的概率为其面积除以物体总面积。假设物体有三个三角形 A、B 和 C，其采样概率分别为 0.1，0.6 和 0.3，则通过一个 0-1 之间的随机数对它们进行采样的方法如下：

- 如果随机数的值小于 0.1，则采样到第一个三角形；
- 如果随机数的值大于等于 0.1 且小于 0.7，则采样到第二个三角形；
- 如果随机数的值大于等于 0.7，则采样到第三个三角形。

更多三角形的情况可以依次类推。这一采样算法可以进一步抽象：我们其实是在一个**单调增的前缀和序列**中**查找**特定元素。是不是存在可以降低查找复杂度的方法？

在三角形内部采样的方法可以参考平行四边形内部的均匀采样方法。三角形是平行四边形的一半，因此你需要对 u 和 v 做出一些额外的处理。否则采样得到的点会超过三角形的范围。

TriangleMesh::fillIntersection 方法已经完成，可以让你的实现方便很多。

3.4 测试

我们在压缩包中额外提供了场景 bunny-light，同学们也可自行搭建场景进行渲染。

在完成了上述实验后，运行这一场景，你应该可以得到下面的图片：

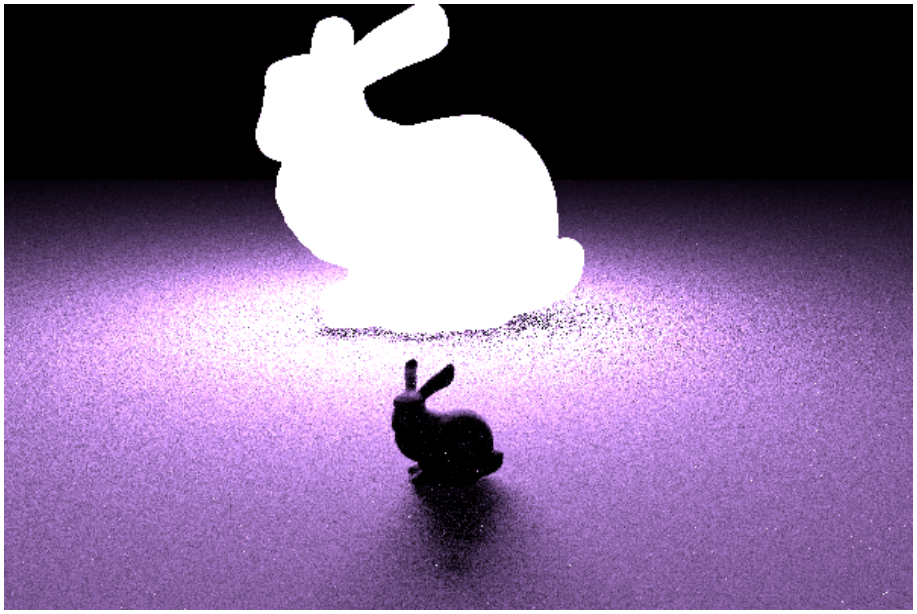

图 3: bunny-light 场景的正确结果

请把你得到的这一图片不经修改地贴进你的实验报告中。你可以适当地增加 spp 来得到更好的视觉效果。

如果你的程序运行时间过慢，尤其是进度条在某个地方突然卡住，请考虑你的三角形表面均匀采样算法是否复杂度过高。

4. Non-Diffuse 光源

4.1 简介

在之前的实验中，我们假设光源是 diffuse 的。换句话说，光源上的每一个点在不同方向上不具有各项异性，朝着任意方向发出的 radiance 都是一致的。

然而，这一假设并不是在所有情况下都成立。举一个极端的例子，在现实世界中，激光笔发出的光线仅朝着正前方的一小块面积，其他地方都无法接收到；一个生活中更为常见的例子是，在侧方向观察你的手机屏幕时，其亮度会变暗；关注数码的同学可能比较熟悉，手机屏幕的方向均匀程度是评测手机屏幕质量的一个重要指标。

在这一节，你需要在 Moer-lite 中模拟出在各个方向上有着不同强度的光源。我们已经在课程中提到如何处理一个高阶余弦分布的光源，你需要实现课程中提到的这类分布。

4.1.1 需要完成的部分

这一部分的完成与否是可选的，如果你完成了这一部分的内容，将会获得额外的加分。

在 AreaLight.cpp 中，你需要完成一些额外的初始化工作，其中部分代码与实验 1 耦合：

```
AreaLight::AreaLight(const Json &json) : Light(json) {
    type = LightType::AreaLight;
    shape = Factory::construct_class<Shape>(json["shape"]);

    // 保证energy与power不同时存在，且必定存在一个
    energy = fetchOptional<Spectrum>(json, "energy", 0.0f);
    power = fetchOptional<Spectrum>(json, "power", 0.0f);

    if (!energy.isZero())
    {
        // write your code here ...
    }
    else // if (!power.isZero())
    {
        // write your code here ...
    }

    // non-diffuse

    // rank与useRank是额外添加的变量，其中rank为cosine分布的指数
    rank = fetchOptional<float>(json, "rank", 0.0f);
    if (rank != 0.0f) useRank = true;
    if (useRank)
    {
        // 添加一些你需要使用的变量
        // write your code here ...
    }
}
```

与此同时，你需要修改 AreaLight 的 evaluateEmission 和 sample 方法，从而返回不同的光源 radiance：

```
Spectrum AreaLight::evaluateEmission(const Intersection &intersection,
                                     const Vector3f &wo) const {

    if(!useRank)
        return energy;
    else
    {
        // write your code here ...
    }
}
```

```
11
12 LightSampleResult AreaLight::sample(const Intersection &shadingPoint,
13                                       const Vector2f &sample) const {
14   Intersection sampleResult;
15   float pdf;
16   shape->uniformSampleOnSurface(sample, &sampleResult, &pdf);
17   Vector3f shadingPoint2sample = sampleResult.position - shadingPoint.position;
18
19   Spectrum retRadiance;
20   if (!useRank)
21   {
22       retRadiance = energy;
23   }
24   else {
25       // write your code here ...
26       // retRadiance = ...
27   }
28
29   return {retRadiance,
30           normalize(shadingPoint2sample),
31           shadingPoint2sample.length() - EPSILON,
32           sampleResult.normal,
33           pdf,
34           false,
35           type};
36 }
```

4.2 测试

通过修改场景文件光源属性中的 rank，你可以测试不同 rank 的光源在统一场景下的视觉效果。

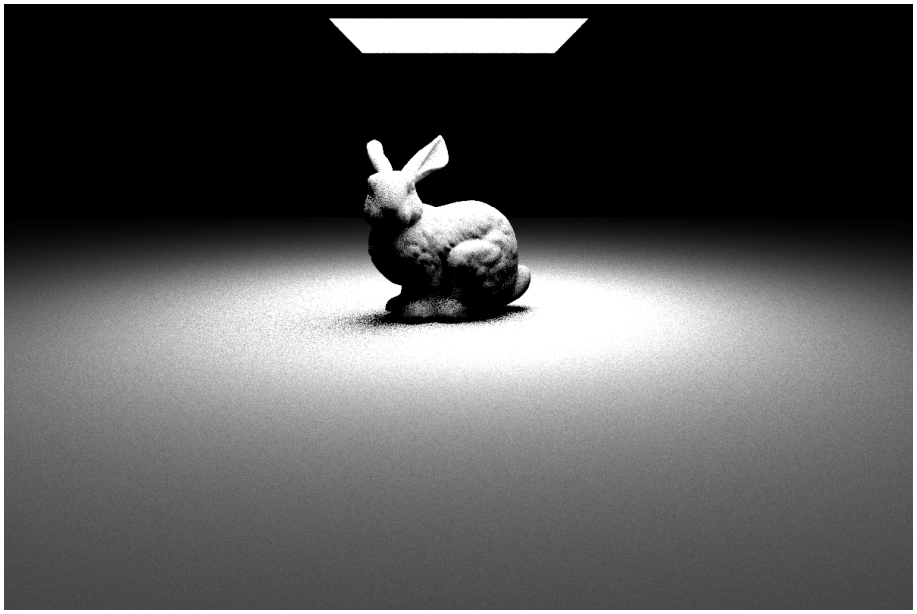

图 4: area-lights-non-diffuse 场景在 rank=1 时的结果

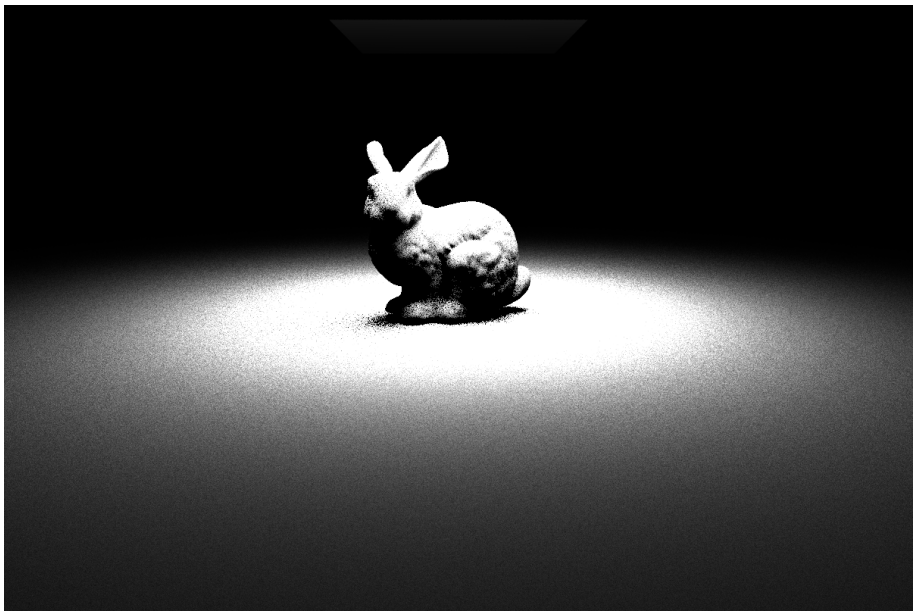

图 5: area-lights-non-diffuse 场景在 rank=5 时的结果

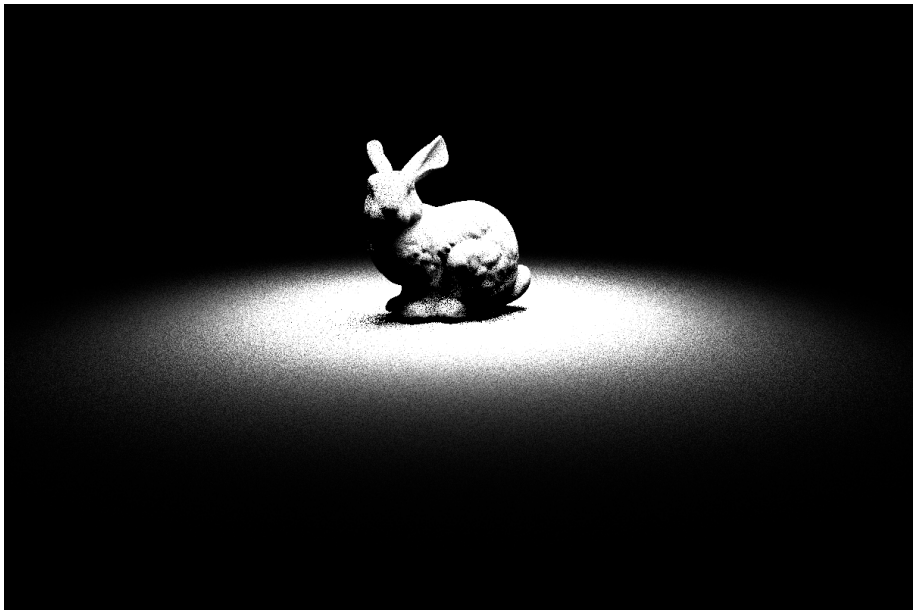

图 6: area-lights-non-diffuse 场景在 rank=15 时的结果

请把你得到的这一图片不经修改地贴进你的实验报告中。

你也可以使得光源在方向上遵循其他的分布。在这个时候，你可以忽略 rank 的值，并构建你自己的分布。如果你尝试了其他的分布，请也把它贴进实验报告中。

5. 杂项与错误修正

5.1 杂项

关于图片格式。

渲染得到的图片可能是 png 格式的，也有可能时 hdr 或者 exr 格式的，这取决于场景 json 的设置。你可以使用 tev 软件将 hdr 或 exr 格式的图片转换为标准动态范围下的 png 图片，进而粘贴进你的实验报告中；如果你觉得这件事情很麻烦，你甚至可以截图。注意，一些全局滤镜（Windows 自动 HDR，NVIDIA RTX HDR）会导致截图得到的图片颜色错误。请在截图之前确认这些全局滤镜都已经被关闭。

关于作弊与抄袭。

本次实验的结果在实验材料中都被包括。因为我们假设所有同学都是诚信的，所以我们**不会**鉴别你的图片是否直接截取自本实验材料。因此理论上如果你想要获得分数又不想花费时间，你可以通过从本实验材料中截取图片的方式获得分数。然而，在这一过程中你没有得到任何除了分数的收益。我们不推荐任何同学这样做。同样地，所有同学的实验报告**不会**经过任何机器查重。

5.2 错误修正

正如我们在第一次作业中提到的，Moer-lite 实验框架依然在不断完善中。为了完成这次作业，你需要在你使用的 Moer-lite 中修复一些错误。

如果你使用 github 上的 Moer-lite，则你只需要拉取最新的主干分支代码即可。如果你使用压缩包中的 Moer-lite，则你需要按照以下方式对代码进行修改。**这些修改是你完成作业所必须的，否则你的渲染图片会产生问题。**

在 Lambert.h 中：

```
virtual Spectrum f(const Vector3f &wo, const Vector3f &wi) const override {
    Vector3f woLocal = toLocal(wo), wiLocal = toLocal(wi);
    return albedo * INV_PI * std::max(wiLocal[1], 0.0f);
}
```

6. 作业

6.1 你需要做的事情

- 根据上述“错误修正”章节，修改你本地的 Moer-lite 源码，或者从 github 仓库拉取最新主干分支；
- 按照要求完成“恒定功率光源”；
- 按照要求完成“球面光源”；

- 按照要求完成“三角形网格光源”；
- 如果你有足够的精力，你可以试着完成“Non-Diffuse 光源”。这一部分是可选的；
- 撰写实验报告，提供实验结果，提交作业。

6.2 提交的文件

你需要提交一份压缩包，在这个压缩包中包含了你的 **pdf 格式的实验报告**。实验报告的撰写形式不限，可以使用 LaTeX，也可以使用更为简单的 markdown 或者 word。**实验报告的形式不会影响分数**，所以我们不推荐你在工具上花费很多精力。我们不鼓励在实验报告的长度上“卷”，过于长的实验报告**不会**获得更高的分数。

对于每个小实验，你需要在实验报告中包含以下内容：

- 你对于这个问题的理解；
- 你的实现思路；
- 你的实现代码，仅需简单表示即可，可以使用 C++ 或者伪代码；
- 你的实验结果；
- 你为这个实验做出的额外工作（新的场景，新的方法，你觉得没有人做过的东西）。

请注意，**实验报告中应避免大段的公式推导**。你需要用文字讲出你对特定问题的理解。

如果你觉得你不小心对实验报告中的图片进行了修改，压缩包中也可以包含你的渲染图片的原图。**此次实验中压缩包中不必包含任何代码文件，你应该在实验报告中将你的实现描述清楚**。你也可以在压缩包中包含你自己搭建的新场景。

6.3 提交截止时间

本次作业放出的时间即为开始时间。

本次作业的提交截止日期为 **2024 年 4 月 15 日晚 23:59**，以邮件日期为准。逾期提交的作业将会按照逾期的时间将分数乘以对应比例。

6.4 提交方式

请发送邮件至助教邮箱 **chenzy@smail.nju.edu.cn**。邮件的标题使用如下格式：

姓名 + 学号 + 图形绘制技术 Lab1

邮件的正文可以包含一些关于你对此次提交的备注，比如已经提交的次数。邮件应包含一个压缩包作为附件，见“提交的文件”章节。**压缩包的标题与邮件标题一致**。