作业 2: Precomputed Radiance Transfer
GAMES202, 2021 年春季

教授: 闫令琪

计算机图形学与混合现实研讨会

GAMES: Graphics And Mixed Environment Seminar

发布日期为北京时间 2021 年 4 月 22 日 (星期四) 上午 10:00

截止时间为北京时间 2021 年 5 月 4 日 (星期二) 下午 8:00

注意:

- 任何更新或更正都将发布在论坛上，因此请偶尔检查一下。
- 论坛链接 `http://games-cn.org/forums/forum/games202/`
- 你必须独立完成自己的作业。
- 你可以在论坛上发布帖子求助，但是发布问题之前，请仔细阅读本文档。
- 在截止时间之前将你的作业提交到 SmartChair 上。

---

1 总览

物体在不同光照下的表现不同，PRT(Precomputed Radiance Transfer) 是一个计算物体在不同光照下表现的方法。光线在一个环境中，会经历反射，折射，散射，甚至还会物体的内部进行散射。为了模拟具有真实感的渲染结果，传统的 Path Tracing 方法需要考虑来自各个方向的光线、所有可能的传播形式并且收敛速度极慢。PRT 通过一种预计算方法，该方法在离线渲染的 Path Tracing 工具链中预计算 lighting 以及 light transport 并将它们用球谐函数拟合后储存，这样就将时间开销转移到了离线中。最后通过使用这些预计算好的数据，我们可以轻松达到实时渲染严苛的时间要求，同时渲染结果可以呈现出全局光照的效果。

预计算部分是 PRT 算法的核心，也是其局限性的根源。因为在预计算 light transfer 时包含了 visibility 以及 cos 项，这代表着实时渲染使用的这些几何信息已经完全固定了下来（P.S. 我们可以基于球谐函数的旋转性质让光源旋转起来，本轮作业的提高部分中将简单的涉及到这一点）。所以 PRT 方法存在的限制包括：

- 不能计算随机动态场景的全局光照
- 场景中物体不可变动

本次作业的工作主要分为两个部分：cpp 端的离线预计算部分以及在 WebGL 框架上使用预计算数据部分

2 开发说明

2.1 编译运行

预计算部分我们以 nori[`https://github.com/wjakob/nori`] 作为光线追踪代码框架

基础代码只依赖 CMake 与 C++17，下载预计算代码后，执行下列命令，就可以编译这个项目

```
$ cd path-to-your-code
$ mkdir build
```

```
$ cd build
$ cmake ..
$ make
```

在此之后，你可以通过执行 build 目录下的./nori.exe来运行程序，参数为指定的 **xml 格式**场景文件

```
# Win
$ ./nori.exe path/to/scene.xml
# OSX / Linux
$ ./nori ../path/to/scene.xml
```

关于 nori 的更多信息，以及编译指南，可以访问`https://wjakob.github.io/nori/`

2.2 预计算球谐系数

本小节你将实现预计算环境光贴图的球谐系数。本次作业只支持 Cubemap 形式的环境贴图。

2.2.1 环境光照

你需要实现 **prt.cpp** 中的 **ProjEnv::PrecomputeCubemapSH()** 函数。

我们想要使用球谐函数来表示环境光，就需要将环境光投影到球谐函数上来得到对应的系数，即

$$SH_{coeff} = \int_S L_{env}(\omega_i) SH(\omega_i) d\omega_i$$

这里我们使用黎曼积分的方法来计算，可得

$$\widehat{SH}_{coeff} = \sum_i L_{env}(\omega_i) SH(\omega_o) \Delta\omega_i$$

我们将 cubemap 的 6 张贴图上每个像素对应的**单位**方向向量保存在 **cubemapDirs** 中。对于第 i 张贴图上坐标为 (x,y) 的方向向量你可以通过 **cubemapDirs[i * width * height + y * width + x]** 来访问，其中 **width** 和 **height** 分别表示贴图的宽和高。

我们提供了函数 **CalcArea(u,v,width,height)** 来计算 cubemap 上每个像素所代表的矩形区域投影到单位球面的面积，其中 **u,v** 分别代表图片坐标系中的横纵坐标，**width** 和 **height** 分别表示贴图的宽和高。

对于 cubemap 的 6 张贴图，我们将其保存在 **images** 中。保存的结果是在线性空间中的，不需要再进行 gamma 矫正了。

提示: 这里你可以假设环境光是一个常数，然后验证这种情况下你计算的球谐系数是否正确。

2.2.2 Diffuse Unshadowed

对于漫反射传输项来说，分为 **unshadowed**, **shadowed**, **interreflection** 三种情况，我们将分别计算这三种情况的漫反射传输球谐系数。

首先我们有光路传输方程 $L(x,\omega_o) = \int_S f_r(x,\omega_i,\omega_o)L_i(x,\omega_i)H(x,\omega_i)\mathrm{d}\omega_i$，对于表面处处相等的漫反射表面，我们可以简化得到 Unshadowed 光照方程

$$L_{DU} = \frac{\rho}{\pi}\int_S L_i(x,\omega_i)\max(N_x\cdot\omega_i,0)\mathrm{d}\omega_i$$

由球谐函数的性质，我们可以将积分项内的光照辐射度和传输函数项分离开了，光照函数的球谐系数由前一节计算出来的球谐系数提供，而本节将要计算每个顶点的传输项球谐系数，对于最简单的 Unshadowed 情况，我们需要的仅仅是将几何项 $M^{DU} = \max(N_x\cdot\omega_i,0)$ 投影到球谐系数里

伪代码可以表示如下:

```
//对于所有的pre-calculated的球面均匀分布的随机方向射线
for(int i=0;i<n_samples;++i){
    //计算当前射线sample的cosine
    double H = Dot(射线采样的vec3, 法线);
    //如果射线在上半球...
    if(H > 0.0) {
        //那么就投影到SH space
        // SH project over all bands into the sum vector
        for(int j=0; j<n_coeff; ++j) {
         //把传输函数项H投影到SH Space (先\sum H(s) * Y_i(s))
            value = H * preGenVector[i].SH_value[j];
            result[j + red_offset ] +=  value;
            result[j + green_offset] +=  value;
```

```
            result[j + blue_offset ] +=  value;
        }
    }
}
// 把上面求的和乘以(权重/采样数), 求出BRDF的球谐投影的系数向量
double factor = area / n_samples;
for(i=0; i<3*n_coeff; ++i) {
    coeff[i] = result[i] * factor;
}
```

2.2.3 Diffuse Shadowed

对于有自阴影的 Shadowed 漫反射传输，预计算方程多加了一项可见性（Visibility term）：

$$L_{DS} = \frac{\rho}{\pi} \int_S L_i(x, \omega_i) V(\omega_i) \max(N_x \cdot \omega_i, 0) \mathrm{d}\omega_i$$

因此我们要投影的传输项就变为 $M^{DU} = V(\omega_i) \max(N_x \cdot \omega_i, 0)$ 相应的伪代码如下：

```
//对于所有的pre-calculated的球面均匀分布的随机方向射线
for(int i=0;i<n_samples;++i){
    //计算当前射线sample的cosine
    double H = Dot(射线采样的vec3, 法线);
    //如果射线在上半球...
    if(H > 0.0) {
        //如果当前pos发出的射线没有跟其他东西相交
        if(!self_shadow(pos,sample[i].vec)) {
            //没有hit到其他地方的射线就对最终光照信息有贡献
            for(int j=0; j<n_coeff; ++j) {
             //把传输函数项H投影到SH Space (先\sum H(s) * Y_i(s))
                value = Hs * preGenVector[i].SH_value[j];
                result[j + red_offset ] += albedo_red * value;
                result[j + green_offset] += albedo_green * value;
                result[j + blue_offset ] += albedo_blue * value;
            }
        }
    }
}
// 把上面求的和乘以(权重/采样数), 求出BRDF的球谐投影的系数向量
double factor = area / n_samples;
for(i=0; i<3*n_coeff; ++i) {
    coeff[i] = result[i] * factor;
}
```

2.2.4 Diffuse Inter-reflection(bonus)

对于具有相互反射的传输项情况，我们需要考虑光线的多次弹射，对于 Inter-reflection 来说，我们的光照传输方程这时候变为

$$L_{DI} = L_{DS} + \frac{\rho}{\pi}\int_S \hat{L}(x', \omega_i)(1 - V(\omega_i))\max(N_x \cdot \omega_i, 0)\mathrm{d}\omega_i$$

对于间接光，我们需要迭代求解：

1. 对于每个顶点，计算 L_{DS}
2. 从当前顶点发射光线，如果当前光线与其他三角形相交，则在交点处求出重心坐标插值后的球谐系数，这个系数就表示间接光照的球谐系数
3. 对于这个反射回来的间接光，乘以几何项 $N_x \cdot \omega_i$
4. 以当前射线为交点，从**第 2 步**计算以当前交点为初始顶点的 L_{DS}

2.2.5 实现

你需要在 **prt.cpp** 中的 **PRTIntegrator::preprocess(const Scene *scene)** 函数中实现预计算过程，预计算得到的环境光系数保存在 **m_LightCoeffs** 中，这是一个 $3\times(SHOrder+1)^2$ 的矩阵，每一列存储了环境光球谐系数的 **RGB** 值；预计算得到的传输项系数保存在 **m_TransportSHCoeffs** 中，这是一个 $(SHOrder+1)^2 \times VertexCount$ 的矩阵，每一列 j 存储了第 j 个顶点的所有球谐系数

2.3 说明

运行如下命令

```
# Win
$ ./nori.exe scenes/prt.xml
# OSX / Linux
$ ./nori ../scenes/prt.xml
```

主程序会自动识别 xml 里的参数并传给我们的程序，场景文件里的主要参数含义如下：**type** 表示漫反射传输项类型，有 **unshadowed**,**shadowed**,**interreflection**

三个值，分别对应三种情况；**bounce** 代表 inter-reflection 时的反弹次数；**PRT-SampleCount** 代表计算球谐系数时需要的蒙特卡洛的采样数量；**cubemap** 代表计算环境光系数对应的 cubemap 的目录，cubemap 的六张贴图固定名字为 **posx.jpg,negx.jpg,posy.jpg,negy.jpg,posz.jpg,negz.jpg**。

PRTIntegrator::Li(const Scene *scene, Sampler *sampler, const Ray3f ray 实现了可视化球谐系数结果的方法，如果在 **PRTIntegrator::preprocess(const Scene *scene)** 中正确预计算得到环境光球谐系数和传输项球谐系数，那么这个函数将会在 GUI 中绘制出一个着色后的模型，如图1所示，你可以在预计算阶段通过显示的图形来 debug 你的预计算代码。

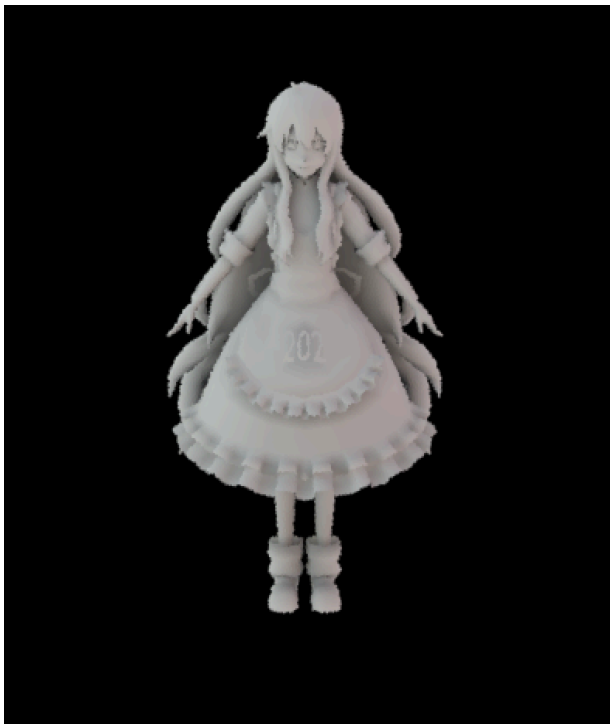


图 1: 预计算 Debug 图片

正确计算的环境光球谐函数系数保存在 cubemap 目录下的 **light.txt** 里，传输项球谐函数系数保存在 cubemap 目录下的 **transport.txt** 里。

你可能会用到以下库或者是函数:

- **sh::ProjectFunction(int order, const SphericalFunction func, int sample_count)** 该函数用来计算给定球谐阶数、投影函数、采样数下的球谐

系数，并返回一个 **std::unique_ptr** 存储的表示为 **std::vector<double>** 的球谐系数,投影函数为用户自定义的 lambda 函数,签名为 **double(double, double)**

- **Mesh::getVertexCount()** 获取当前 mesh 的顶点数量
- **Mesh::getVertexPositions().col(i)** 获取当前 mesh 的第 i 个顶点，顶点类型为 **Point3f**
- **Mesh::getVertexNormals().col(i)** 获取当前 mesh 的第 i 个法线，法线类型为 **Normal3f**
- **Scene::rayIntersect(const Ray3f ray, Intersection its)** 指定射线与场景求交，并返回是否求交与类型为 Intersection 的交点信息
- **Scene::rayIntersect(const Ray3f ray)** 指定射线与场景求交，返回是否相交
- **Intersection::tri_index** 记录了当前交点的三个三角形的序号
- **Intersection::bary** 记录了当前交点的重心坐标

2.4 实时球谐光照计算

如果你顺利完成了本节之前提到的预计算工作，是时候到课程作业框架中使用预计算好的数据进行检验了。本节工作会更深入作业框架的使用，尤其是材质系统，因此主要工作会集中在：

- 创建并调用一种使用预计算数据的材质，可以参照作业 1 创建的 phongMaterial 材质流程，命名随意（代码量：20 行左右）
- 编写该材质对应的 Shader，我们推荐在 **vertexShader** 中通过对应 SH 系数之间点积并累加来计算 **vColor**，接着把 **vColor** 传递到 **fragmentShader** 中插值后着色（代码量：主体代码 10 行左右）

在正式开始本节工作之前有两点需要注意：

第一，之前工作完成的预计算 **lighting** 以及 **light transport** 部分是按照 txt 格式保存，需要在 js 中进行读取分割并保存为 Array。为了使作业更专注于 **PRT** 算法本身的实现，我们将这些工作已经在 **engin.js** 中的 88-114 行实现完毕。将预计算的 light.txt 以及 transport.txt 放入对应的 **assets/cubemap/贴图文件夹/**下后，解开上述几行代码的注释预计算数据就可以自动解析完成了。

解析后的数据储存在了两个全局变量 **precomputeL**、**precomputeLT** 中，方便大家使用。其中 **precomputeL** 将预计算环境光的 $SH\ vector$ 保存为一个长度为 SH 系数数量的 Array，该 Array 中每个元素都是代表着 RGB 的三维数组；**precomputeLT** 将预计算 $light\ transfer$ 的 $SH\ vector$ 保存为一个长度为 $VertexCount * SHCoefficientCount$ 的 Array，这样可以方便后续绑定到 **attribute mat3** 类型的变量中。

第二，作业中的基础部分不会涉及过多的 WebGL 内容，因此对于将 **precomputeLT** 传入给 **PRTVertexShader**，我们要求在 shader 中如此声明：

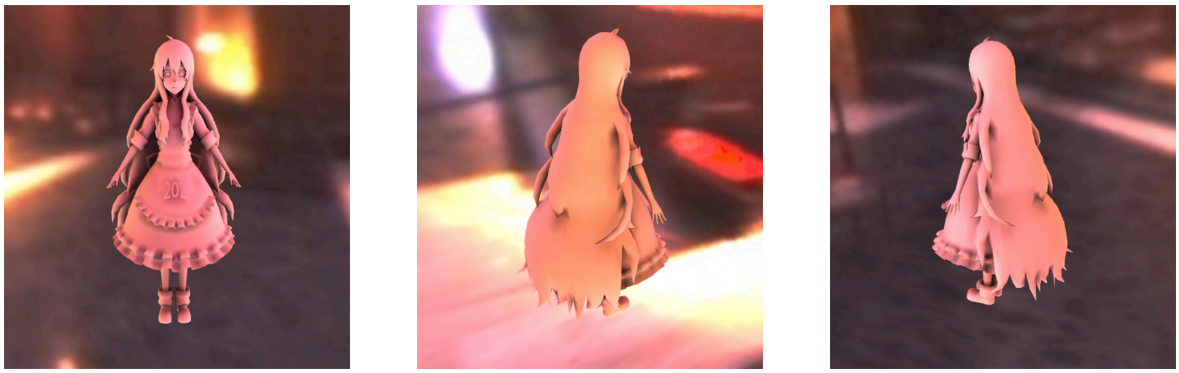
```
attribute mat3 aPrecomputeLT;
```

然后在声明 PRT 材质的构造函数中这样使用：

```
super({ 你的uniform变量 }
}, [
    'aPrecomputeLT'
], vertexShader, fragmentShader, null);
```

后续 **precomputeLT** 到 **aPrecomputeLT** 的绑定工作我们已经帮你完成，每个 vertex 都会在 shader 中绑定传入自己的 **attribute mat3** 类型的变量。

如果一切顺利，此时你应该可以看到呈现出环境光照效果的渲染结果：



(a) (b) (c)

图 2: 环境光照-Mary

2.5 环境光球谐旋转 (bonus)

本节提高部分，要求完成低阶球谐旋转的相关工作——能够在环境光旋转的同时旋转 **precompute lighting** 的$\boldsymbol{SH\ vector}$，从而使得旋转后的环境光与我们的模型依旧可以保持不错的交互效果。

基于我们使用的三阶球谐拟合，在这里我们推荐一种实现起来非常简单的**低阶** SH **快速旋转方法**，当然也很欢迎和期待使用其他能够得到不错渲染结果的球谐旋转算法。为了理解该 SH 快速旋转算法，我们需要对 SH 的旋转性质有所简要了解，这些性质的证明文档中不会有所涉及。首先值得关注的第一个性质是，球谐函数具有旋转不变形性，通俗的讲就是：

假设有一个旋转$\boldsymbol{R}$，对原函数$\boldsymbol{f}(\boldsymbol{x})$ 的旋转$\boldsymbol{R}(\boldsymbol{f}(\boldsymbol{x}))$ 与直接旋转$\boldsymbol{f}(\boldsymbol{x})$ 的自变量是一样的，即

$$\boldsymbol{R}(\boldsymbol{f}(\boldsymbol{x})) = \boldsymbol{f}(\boldsymbol{R}(\boldsymbol{x}))$$

第二个性质是，对每层 $band$ 上的 $SH\ coefficient$，可以分别在上面进行旋转，并且这个旋转是线性变化。

有了这两条性质就可以简单的推导一下快速球谐旋转方法。

对于球谐函数上某层 $band\ l$，该 $band$ 上有一个 $2l+1$ 维的、由 $SH\ coefficient$ 构成的向量。假设我们有 $2l+1$ 个任意的$\boldsymbol{3D\ normal\ vector\ n}$，旋转函数$\boldsymbol{R}$，

能够返回该 $band$ 上球谐投影系数的函数$\boldsymbol{P}$，根据上述两条性质存在一个该 $band$ 上 $SH\ coefficient\ vector$ 的旋转矩阵$\boldsymbol{M}$ 使得：

$$\boldsymbol{MP(n_i) = P(R(n_i))\ , i \in [-l, l]}$$

整理一下得到：

$$\boldsymbol{M[P(n_{-l}), ..., P(n_l)] = [P(R(n_{-l})), ..., P(R(n_l))]}$$

记 $\boldsymbol{A = [P(n_{-l}), ..., P(n_l)]}$ 。如果矩阵$\boldsymbol{A}$ 是可逆矩阵，此时 $SH\ coefficient\ vector$ 的旋转矩阵$\boldsymbol{M}$ 易得，即：

$$\boldsymbol{M = [P(R(n_{-l})), ..., P(R(n_l))]A^{-1}}$$

接下来可以讨论具体算法的实现了，针对本次作业我们需要实现的是三阶球谐的快速旋转，即需要求出 $band1, band2$ 两层上各自 $3*3, 5*5$ 的旋转矩阵 M_1, M_2，注意 $band0$ 只有一个投影系数，故不需要处理。具体实现流程如下所示：

1. 首先，对于第 l 层 $band$，选取 $2l+1$ 个 $normal\ vector\ n$，对所有的 n 将 n_i 在球谐上投影并取该 $band$ 上的系数，即得到$\boldsymbol{P(n_i)}$。总计 $2l+1$ 个 $2l+1$ 维向量$\boldsymbol{P(n_i)}$ 构成了我们需要的矩阵$\boldsymbol{A}$，并求出$\boldsymbol{A^{-1}}$

2. 其次，给定旋转$\boldsymbol{R}$，对所有 n 依次做旋转$\boldsymbol{R(n_i)}$、SH 投影取系数$\boldsymbol{P(R(n_i))}$。此时我们应该得到 $2l+1$ 个 $2l+1$ 维向量$\boldsymbol{P(R(n_i))}$，这些向量构成了我们需要的矩阵$\boldsymbol{S}$

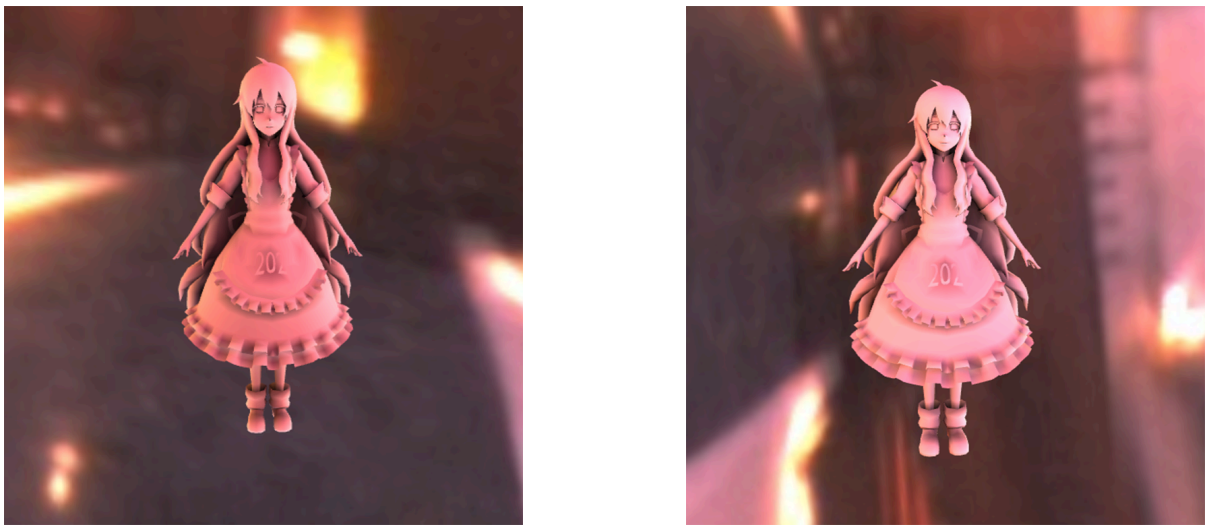
3. 接着，需求的方阵$\boldsymbol{M = SA^{-1}}$，用 $\boldsymbol{M}$ 乘以原 $band$ 上的 $SH\ coefficient\ vector$ 就可以得到旋转后该层 $band$ 上的 SH 系数了

4. 最后，重复上述过程至待求的每一层 $band$ 上，将结果重新拼接起来即可得到整个 SH 系数旋转后的结果

本轮工作的代码量总计应该在 200 行以内。有两点信息可能会在你实现 SH 快速旋转算法时帮上忙：

- lib 文件夹下导入的 **sh.js** 中提供了 **SHEval()** 函数，可以帮助你轻松的获得一个三维向量在指定阶数球谐函数上的所有投影系数
- 本次提高部分的编写需要用到高维矩阵运算以及矩阵求逆等操作，我们已经导入了 **Math.js** 库来进行辅助计算，该库对应的文档地址在`https://mathjs.org/docs/datatypes/matrices.html`

解开 **WebGLRenderer.js** 中的第 53 行代码注释，开启环境光旋转，并将旋转后的 **precomputeL** 传入 Shader。如果一切顺利，你应该可以看到渲染出来的模型与旋转的环境光产生了不错的交互：



(a) (b)

图 3: 环境光旋转-Mary

3 评分与提交

提交时需要删除 /lib, /assets, /build, /ext, /include, /scenes 等所有与你实现代码无关的文件夹，并在建立的 /images 文件夹中保存运行截图。截图需要展示 PRT 环境光照效果，以 PRT_Unshadowed_xxx.png、PRT_Shadowed_xxx.png 格式保存，xxx 的内容可自行决定。如果包含提高部分则任务一命名为 PRT_InterRef_xxx.pn 任务二提交一个 gif 旋转动图命名为 PRT_Rotation_xxx.gif。

同时在 README.md 中简要描述本轮工作，尤其是完成了作业的哪些部分。如果完成的任务中包含了提高部分请注明改动了哪些代码、文件以方便助教同学批改。

3.1 评分

- [5 分] 提交的格式正确，包含所有必须的文件。代码可以编译和运行。
- [10 分] 预计算环境光照：
  正确实现了预计算环境光照算法
- [5 分] 预计算 Diffuse Unshadowed LT：
  正确实现预计算 Diffuse unshadowed LT 算法
- [15 分] 预计算 Diffuse Shadowed LT：
  正确实现预计算 Diffuse Shadowed LT 算法。
- [5 分] 预计算数据使用：
  在作业框架中正确实现 PRT 材质，并能使用预计算数据正确展示出环境光照下的模型
- [10 分] Bonus 1：
  正确实现预计算 Diffuse Inter-reflection。
- [10 分] Bonus 2：
  正确实现 SH 旋转。
- [-3 分] 惩罚分数：
  未删除 /lib，/assets，assignment2.pdf 等与代码无关的文件夹。
  未按格式建立 /images，缺少结果图片。

3.2 提交

作业提交使用的平台为 **Smartchair** 平台，地址为http://www.smartchair.org/GAMES202/。