中国科学技术大学

实　验　汇　报

固定相机与方向光条件下的
树木 Billboard 贴图优化

基于可微着色的双贴图材质拟合

姓　　名： 张竞一

学　　号： PB23010356

学　　院： 数学科学学院

课　　程： 2026 夏计算机图形学前沿

指　　导： 微胖（网易）

2026 年 8 月

摘要

Billboard 通过二维平面与少量贴图近似复杂三维对象，是游戏中远距离植被渲染常用的细节层次技术。普通单贴图方案通常直接使用一次离线渲染结果，颜色中混入了生成时的光照，因此当运行时方向光发生变化时，平面外观难以保持与高模一致。针对这一问题，本课题以真实树木模型 `tree.glb` 为输入，在 Blender 中固定一台正交相机，仅改变方向光方向生成 32 张 RGBA 参考图，其中 24 张用于训练、8 张用于测试。优化阶段保持平面与相机不变，将 `albedo_alpha` 和 `normal_roughness` 两张纹理设为可训练参数，并以 PyTorch 张量运算实现包含环境项、漫反射项和粗糙度控制高光项的可微着色模型，金属度固定为 0。训练结果表明，200 个 epoch 后平均训练目标由 0.1254 降至 0.0490；在 8 个未参与训练的方向光上，前景 RGB 平均绝对误差由 0.1145 降至 0.0492。三栏对比图进一步显示，优化后的平面能够恢复树干和叶簇的颜色层次，并对方向光变化产生比初始单贴图方案更合理的响应。实验同时表明，固定观察方向的二维表示仍无法复现高模的遮挡、自阴影和细小几何细节，当前方法应被理解为面向远距离树木渲染的材质近似，而不是完整三维替代。

关键词：Billboard；可微渲染；方向光；纹理优化；树木 LOD

目录

1 引言

1.1 研究背景

实时场景中的树木通常具有大量叶片、枝条和透明材质。近距离渲染需要保留这些几何与材质细节，而当树木远离相机后，继续提交完整网格会造成不必要的顶点处理、光栅化和材质计算开销。细节层次技术通过随屏幕占比降低表示复杂度，在图像质量与运行效率之间取得平衡。Clark 提出的层次几何模型奠定了这一思想的基础 [1]，渐进网格与二次误差度量则代表了后续几何简化路线 [2, 3]。

Billboard 是比几何简化更激进的图像化表示：将复杂对象压缩到一张平面及其材质贴图中，使渲染成本基本不再随原始模型面数增长。对树木、草丛和远景装饰物而言，这一表示具有很强的工程实用性。然而，若只把某一次高模渲染直接作为颜色贴图，贴图会同时包含反照率和当时的明暗结果。运行时改变主光方向时，平面上的已烘焙阴影不会随之改变，容易出现树冠始终偏亮或始终偏暗、树干受光方向错误等问题。

本课题讨论相机固定，方向光改变时，如何让 Billboard 更接近同一条件下的高模渲染。

1.2 研究问题

设固定相机为 C，树木高模为 M，方向光单位向量为 $\boldsymbol{l}$。Blender 参考渲染记为

$$\mathcal{I}^{\mathrm{ref}}(\boldsymbol{l}) = R_{\mathrm{high}}(M, C, \boldsymbol{l}). \tag{1}$$

Billboard 的可训练参数由两张纹理组成：颜色与透明度纹理 $T_{a\alpha}$，以及法线与粗糙度纹理 T_{nr}。其输出记为

$$\mathcal{I}^{\mathrm{bill}}(\boldsymbol{l}; T_{a\alpha}, T_{nr}) = R_{\mathrm{bill}}(C, \boldsymbol{l}, T_{a\alpha}, T_{nr}). \tag{2}$$

目标是在一组训练方向光上最小化两者差异，同时在未参与优化的方向光上保持合理响应：

$$(T^*_{a\alpha}, T^*_{nr}) = \arg\min_{T_{a\alpha}, T_{nr}} \sum_{\boldsymbol{l}\in\mathcal{L}_{\mathrm{train}}} \mathcal{L}\Big(\mathcal{I}^{\mathrm{bill}}(\boldsymbol{l}), \mathcal{I}^{\mathrm{ref}}(\boldsymbol{l})\Big). \tag{3}$$

这里不优化树木几何、相机参数或光源参数，实验变量只有方向光方向。

1.3 相关工作

可微渲染把图像误差反向传播到场景参数，为纹理、几何和光照反演提供了统一工具。OpenDR 通过近似导数建立早期可微渲染框架 [4]；Neural 3D Mesh Renderer 和 Soft Rasterizer 分别通过近似梯度与概率化覆盖处理离散光栅化边界 [5, 6]；nvdiffrast 和 PyTorch3D 进一步提供了模块化、高性能的实现 [7, 8]。

本课题平面在图像中始终正对固定相机，核心是把纹理采样、法线恢复和局部光照写成可求导的 PyTorch 运算。

1.4 主要工作

本课题完成了以下工作：

(1) 以真实的 `tree.glb` 为高模输入，建立固定相机、改变方向光的 Blender 数据生成流程，并保存透明背景参考图与光照元数据；

(2) 将颜色、透明度、切线空间法线和粗糙度组织为两张可训练纹理，构造简化可微着色模型；

(3) 使用当前训练权重和真实测试图生成“优化前、优化后、高模参考”的结果，并结合训练曲线、纹理结果和测试误差分析方法效果与局限。

1.5 论文结构

第 2 章定义固定相机实验与双贴图着色方法；第 3 章说明 Blender 数据生成、训练参数和输出流程；第 4 章基于项目中现有的真实训练结果进行分析；第 5 章总结工作并讨论后续改进方向。

2 问题定义与方法设计

2.1 固定相机实验边界

实验场景包含一棵高模树、透明背景、一台正交相机和一个太阳光源。数据生成脚本首先为树木选择一个轮廓面积较大的观察方向，随后锁定相机位置、朝向、投影类型和正交尺度。训练集与测试集中的每一张图都由这台相机生成，因此图像间不存在相机参数变化。

固定相机使 Billboard 平面可以直接对应输出图像平面。该简化有两点作用：一是排除了相机变化带来的投影、遮挡和轮廓差异；二是让优化重点落在“同一轮廓在不同方向光下应如何着色”这一问题上。

2.2 方向光采样与坐标变换

Blender 脚本通过半球上的 Fibonacci 序列生成 $N_l = 32$ 个单位方向。设世界坐标中的光方向为 $\boldsymbol{l}_w$，固定相机朝向对应的 Billboard 法线为 $\boldsymbol{n}_b$。以世界竖直方向 $\boldsymbol{u}_w = (0, 0, 1)$ 构造平面坐标基：

$$\boldsymbol{r}_b = \frac{\boldsymbol{u}_w \times \boldsymbol{n}_b}{\|\boldsymbol{u}_w \times \boldsymbol{n}_b\|_2}, \tag{4}$$

$$\boldsymbol{u}_b = \boldsymbol{n}_b \times \boldsymbol{r}_b. \tag{5}$$

光方向在平面坐标中的表示为

$$\boldsymbol{l} = \begin{bmatrix} \boldsymbol{r}_b^{\mathsf{T}} \boldsymbol{l}_w & \boldsymbol{u}_b^{\mathsf{T}} \boldsymbol{l}_w & \boldsymbol{n}_b^{\mathsf{T}} \boldsymbol{l}_w \end{bmatrix}^{\mathsf{T}}. \tag{6}$$

训练和推理均使用该三维向量，使着色器中的法线与光线处于同一坐标系。

2.3 两张可训练纹理

2.3.1 颜色与透明度纹理

第一张纹理 $T_{a\alpha} \in \mathbb{R}^{4\times H_t \times W_t}$ 保存 RGB 反照率和 alpha。网络参数本身不受范围约束，前向过程中使用 sigmoid 映射到 $[0, 1]$：

$$[\boldsymbol{a}, \alpha] = \sigma(T_{a\alpha}). \tag{7}$$

训练开始前，代码从训练加载器首个批次的第一张 RGBA 参考图初始化该纹理。由此得到的初始状态可以看作普通单贴图 Billboard：颜色和轮廓来自一张离线参考图，尚未根据多种方向光共同调整。

2.3.2 法线与粗糙度纹理

第二张纹理 $T_{nr} \in \mathbb{R}^{3\times H_t \times W_t}$ 的前两个通道表示切线空间法线的 x, y 分量，第三个通道表示粗糙度。为避免 $x^2 + y^2 > 1$，我使用

$$n_x = 0.95 \tanh(T_{nr}^x), \tag{8}$$

$$n_y = 0.95 \tanh(T_{nr}^y), \tag{9}$$

$$n_z = \sqrt{\max(1 - n_x^2 - n_y^2, \varepsilon)}. \tag{10}$$

粗糙度为

$$\rho = \mathrm{clip}(\sigma(T_{nr}^{r}), 0.08, 1). \tag{11}$$

这种参数化始终产生朝向平面正侧的单位法线近似，并避免高光指数出现数值异常。金属度固定为 0，不作为纹理通道或优化变量。

2.4 简化可微着色模型

纹理先通过双线性插值从 128×128 放大到 256×256。设固定观察方向为 $\boldsymbol{v} = (0, 0, 1)$，归一化方向光为 $\boldsymbol{l}$，半程向量为

$$\boldsymbol{h} = \frac{\boldsymbol{l} + \boldsymbol{v}}{\|\boldsymbol{l} + \boldsymbol{v}\|_2}. \tag{12}$$

漫反射响应为

$$d = \max(\boldsymbol{n} \cdot \boldsymbol{l}, 0), \tag{13}$$

高光响应使用由粗糙度控制的指数：

$$p(\rho) = \frac{2}{\rho^2} - 2, \tag{14}$$

$$s = 0.08 \max(\boldsymbol{n} \cdot \boldsymbol{h}, 0)^{p(\rho)}. \tag{15}$$

最终预乘 alpha 的 RGB 输出为

$$\boldsymbol{c} = [\boldsymbol{a}(k_a + d) + s]\, \alpha, \qquad k_a = 0.15. \tag{16}$$

模型保持环境项为常数，不计算投射阴影、环境遮蔽、透射和多次散射。

2.5 训练目标

设预测 RGB 和 alpha 分别为 $\hat{\mathcal{I}}$、$\hat{A}$，参考图为 $\mathcal{I}$、A。当前 `train.py` 使用三部分损失。首先，只在参考前景区域统计 RGB 绝对误差：

$$\mathcal{L}_{\mathrm{rgb}} = \frac{\sum_{p,c} A_p \left| \hat{\mathcal{I}}_{p,c} - \mathcal{I}_{p,c} \right|}{\sum_{p,c} A_p + \varepsilon}. \tag{17}$$

其次，在整幅图上约束 alpha：

$$\mathcal{L}_{\alpha} = \frac{1}{|\Omega|} \sum_{p} |\hat{A}_p - A_p|. \tag{18}$$

最后，对预测 RGB 的相邻像素差施加平滑项：

$$\mathcal{L}_{\mathrm{smooth}} = \mathrm{mean}\, |\hat{\mathcal{I}}_{i+1,j} - \hat{\mathcal{I}}_{i,j}| + \mathrm{mean}\, |\hat{\mathcal{I}}_{i,j+1} - \hat{\mathcal{I}}_{i,j}|. \tag{19}$$

总目标为

$$\mathcal{L} = \mathcal{L}_{\mathrm{rgb}} + 0.5\mathcal{L}_{\alpha} + 0.02\mathcal{L}_{\mathrm{smooth}}. \tag{20}$$

所有运算均由 PyTorch 自动求导，因此梯度可以直接更新两张纹理。

3 数据生成与工程实现

3.1 高模输入与归一化

本实验使用 `tree.glb` 作为唯一高模输入。Blender 脚本通过 glTF 导入器加载模型，并保留导入层级的父子变换。随后计算所有网格包围盒角点在世界坐标中的范围，将模型中心平移到原点，并按最长边缩放，使整体尺寸约为 2 个世界单位，降低了不同资产尺度对相机和光照参数的影响。

3.2 固定相机的自动选定

`render_blender.py` 在正式生成数据前，沿水平方向检查 12 个候选相机方向。候选相机均使用正交投影、相同距离和相同正交尺度，脚本在 Blender Workbench 中快速渲染 128×128 透明图，并以可见 alpha 像素数作为轮廓面积近似，选择面积最大的方向。该过程只用于确定一次固定相机；选择完成后，所有正式样本均使用同一位置和朝向。

当前数据的固定相机位置记录在 `metadata.json` 中，为

$$(3.9367,\ 0,\ 0.7086), \tag{21}$$

目标点为原点。相机类型为正交相机，正交尺度为 2.6，默认距离参数为 4.0。

3.3 方向光参考图生成

正式渲染使用 Blender Cycles。图像分辨率为 256×256，每像素采样数为 32，开启去噪并使用透明胶片背景。太阳光能量设为 3.0。脚本依次设置 32 个方向光，并输出

```
dataset/tree/light_000.png
...
dataset/tree/light_031.png
```

以及 `dataset/tree/metadata.json`。每条元数据包含图像文件名、光照编号、训练或测试标记、转换到 Billboard 坐标的光方向和固定相机参数。

图 1 给出八个方向光样本。所有图像中的树干底部、树冠外轮廓和枝条投影均保持在相同像素位置，而树冠顶部、左右叶簇和树干表面的亮度随光照编号发生变化。这一现象与脚本中的固定相机设定一致：数据差异由方向光引起，而不是由投影位置或观察方向变化引起。

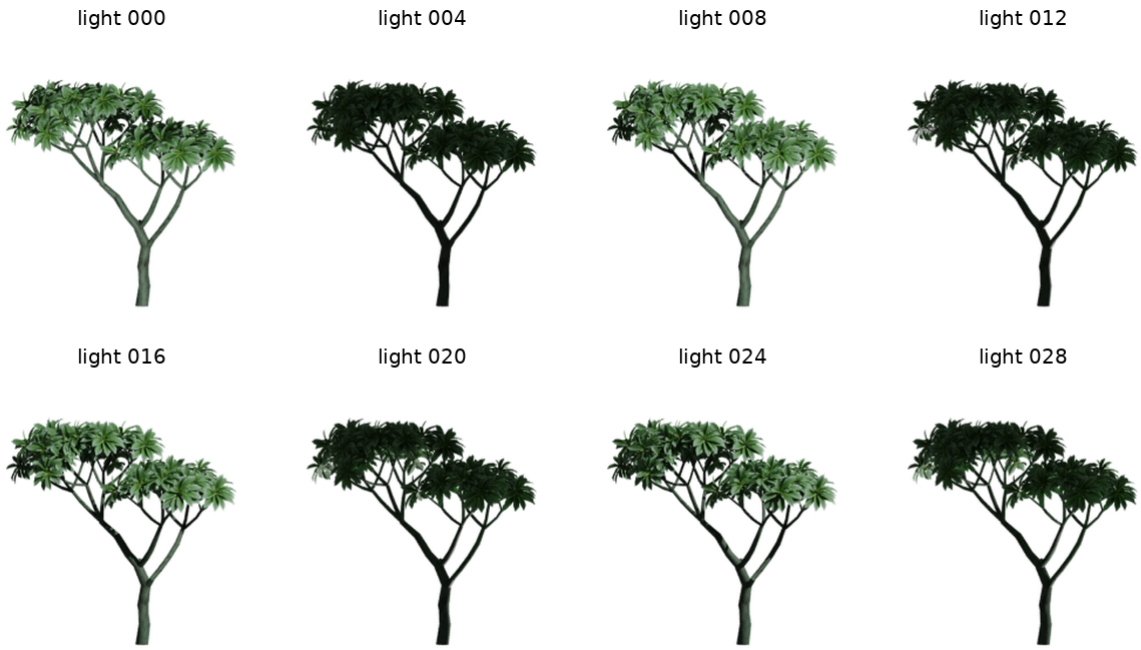

图 1: 固定相机下八个方向光的高模参考图。树木投影与轮廓位置保持一致，明暗分布随方向光变化。

从 light 000、008、016 和 024 可以看到较亮的叶片区域在树冠不同部分迁移；light 004、012、020 和 028 的整体照度较低，但树干与叶簇的轮廓仍严格对齐，这从输出层面验证了数据生成流程对实验变量的控制。

3.4 训练与测试划分

32 个光照样本按编号确定性划分：编号 0–23 为训练集，共 24 张；编号 24–31 为测试集，共 8 张。`dataset.py` 读取 RGBA 图像，将 RGB 和 alpha 分开，并对光方向重新归一化。训练加载器启用随机打乱，测试加载器保持顺序不变。

3.5 训练配置

当前训练由 `train.py` 完成，主要参数列于表 1。

表 1: 当前实际训练配置

项目	数值	代码位置
训练样本数	24 个方向光	`--n_train 24`
测试样本数	8 个方向光	`metadata.json`
参考图分辨率	256×256	`--img_res 256`
纹理分辨率	128×128	`--tex_res 128`
批大小	4	`--batch_size 4`
训练轮数	200	`--epochs 200`
优化器	Adam	`torch.optim.Adam`
学习率	0.01	`--lr 0.01`
环境项	0.15	`shader.py`

训练开始时，`albedo_alpha` 由一张训练参考图初始化，`normal_roughness` 使用常量初值；这一状态保存为 `baseline.pt`。200 个 epoch 后的参数保存为 `optimized.pt`。训练同时导出两张纹理、每轮损失记录和训练曲线。

3.6 评估与输出流程

`eval.py` 逐个读取 8 个测试方向光，同时加载训练前后两个检查点。对每个方向光分别渲染初始 Billboard 和优化后 Billboard，并读取对应的高模参考图。显示时，将 alpha 小于 0.5 的像素设为白色，最后水平拼接为三栏图：

$$\text{初始单贴图 Billboard} \mid \text{优化后双贴图 Billboard} \mid \text{高模参考}. \tag{22}$$

4 实验结果与分析

4.1 结果组织与评价依据

实验输出由四类构成：训练损失曲线用于观察优化过程是否稳定；两张训练纹理及其通道分解用于检查参数是否形成了合理的材质结构；测试方向光上的误差用于衡量未参与训练的光照条件下的重建质量；对比图用于直接观察优化前 Billboard、优化后 Billboard 与高模参考之间的视觉差异。

对比图从左到右表示：优化前的普通单贴图 Billboard、优化后的双贴图 Billboard、相同方向光下的高模 `tree.glb` 参考渲染。由于三者使用相同图像尺寸和相同方向光，树冠明暗、树干亮度与轮廓区域可以直接比较。

4.2 训练收敛情况

图 2 为 200 个 epoch 的平均训练目标。第 1 个 epoch 的损失为 0.1254，第 200 个 epoch 为 0.0490，下降约 60.9%。曲线在前 30 个 epoch 下降较快，此后以较缓速度继续下降，并在训练末段保持稳定。该形态表明颜色、透明度和局部法线首先完成大尺度调整，后续迭代主要修正叶簇边缘、枝干区域与较困难光照下的细节。

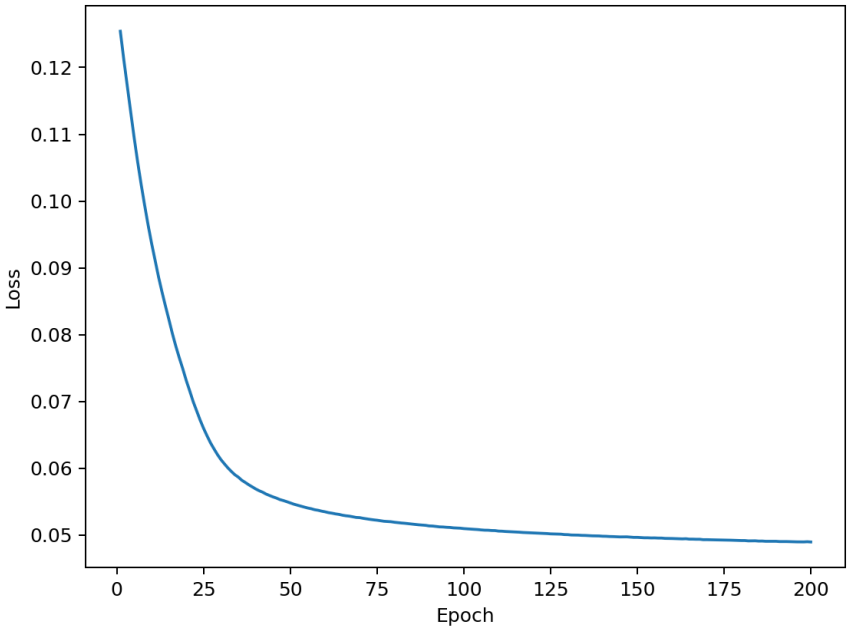

图 2: 200 个 epoch 的训练目标曲线。损失持续下降并逐渐收敛。

训练曲线反映的是 24 个训练方向光上的复合目标。其平稳下降说明可微着色器、损失函数和纹理参数之间的梯度传递能够正常工作。

4.3 双贴图输出与通道分析

训练结束后导出的两张 128×128 纹理如图 3 所示。`albedo_alpha` 的 RGB 通道保存反照率，alpha 通道保存树木轮廓；`normal_roughness` 保存经过输出映射后的法线参数与粗糙度参数。

albedo_alpha　　normal_roughness

图 3: 训练程序直接导出的两张纹理。左图编码 RGB 反照率与 alpha，右图编码切线空间法线参数与粗糙度。

为了更清楚地解释各通道，图 4 进一步分解为四种可视化。第一幅将反照率按 alpha 合成到白色背景，可看到树冠、树干和细枝的颜色结构；第二幅为 alpha 灰度图，白色区域对应保留的树木像素；第三幅根据着色器公式重建单位法线并映射到 RGB；第四幅为粗糙度灰度图。

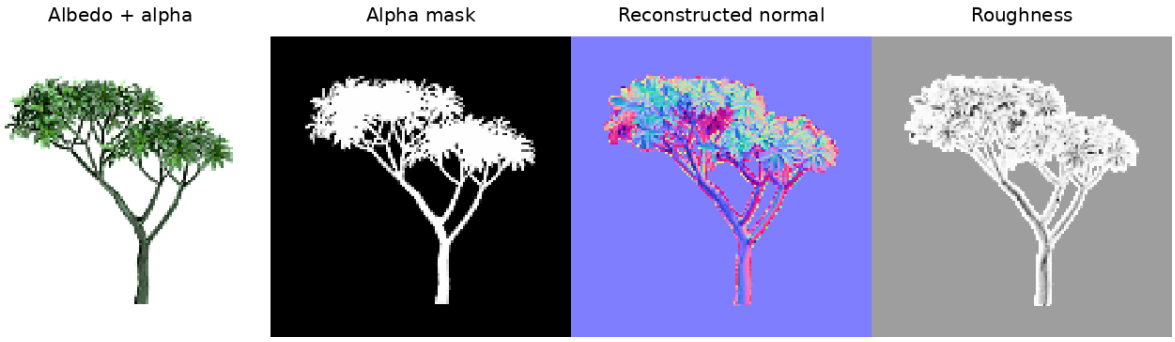

图 4: 训练纹理的通道分解。由左至右为反照率与透明度合成结果、alpha 掩膜、重建的切线空间法线以及粗糙度。

alpha 掩膜与高模投影的树冠及树干轮廓一致，说明透明度优化能够将三维树木压缩为有效的二维覆盖区域。法线图在叶簇边缘、树冠内部和枝条附近具有明显方向变化，而背景区域较为平滑，这正是只在对象区域学习方向相关着色的目标结果。粗糙度图在树冠和枝干之间形成差异，使高光响应不必由单一常数控制。两张纹理共同提供了普通颜色贴图无法表达的方向光响应能力。

4.4 测试方向光的数值结果

本文在编号 24–31 的 8 个测试方向光上计算前景 RGB L1、整图 alpha L1 和前景 PSNR。平均结果见表 2。

表 2: 8 个测试方向光上的平均结果

状态	前景 RGB L1 ↓	Alpha L1 ↓	前景 PSNR/dB ↑
优化前单贴图	0.1145	0.01031	16.05
优化后双贴图	0.0492	0.00536	22.93

优化后前景 RGB L1 降低约 57.0%，alpha L1 降低约 48.0%，前景 PSNR 提高约 6.88 dB。图 5 进一步给出每个测试方向光的结果。8 个样本的 RGB L1 均下降，PSNR 均提高。

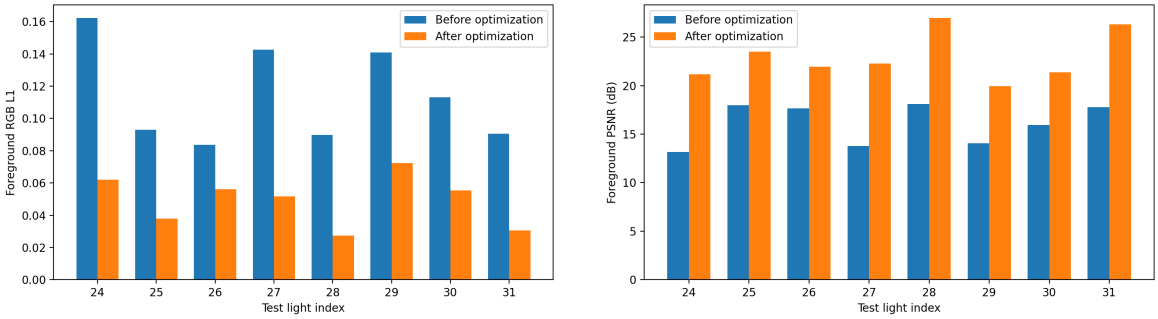

图 5: 各测试方向光的误差比较。左图为前景 RGB L1，右图为前景 PSNR；横轴编号对应 light 024–031。

不同方向光的提升幅度存在差异。light 026 的 RGB L1 降幅约为 32.8%，是 8 个样本中相对较小的一项；light 028 的降幅约为 69.6%，优化后前景 PSNR 达到 26.96 dB。即使在相对困难的 light 026 上，优化后误差仍低于优化前结果，表明两张纹理学到的响应能够覆盖完整测试集合。

4.5 训练前后对比

4.5.1 较强正向照明：light 024

图 6: 测试光照 024。左：优化前的普通单贴图 Billboard；中：优化后的双贴图 Billboard；右：相同方向光下的高模参考。

light 024 中，高模树冠包含明显的绿色叶片层次，左侧普通单贴图结果则大面积接近黑色。优化后的中栏恢复了顶部叶簇和右侧树冠的亮度变化，树干也由近乎纯黑转变为带有纵向明暗的灰绿色。该样本的前景 RGB L1 从 0.1623 降至 0.0621，PSNR 提高约 7.97 dB，数值变化与视觉改善一致。

4.5.2 整体偏暗照明：light 026

图 7: 测试光照 026。左：优化前的普通单贴图 Billboard；中：优化后的双贴图 Billboard；右：相同方向光下的高模参考。

light 026 的高模参考整体较暗，树冠内部阴影较多。优化后的 Billboard 保留了这种暗调，同时恢复了叶片间的绿色差异和树干右侧的受光变化。该方向光的提升幅度小于其他样本，主要差异集中在树冠内部的遮挡暗部和细枝交叉区域。

4.5.3 误差最低的测试样本：light 028

图 8: 测试光照 028。左：优化前的普通单贴图 Billboard；中：优化后的双贴图 Billboard；右：相同方向光下的高模参考。

light 028 的优化后结果在树冠整体明暗、主干亮度和左右叶簇对比上均接近高模参考。其前景 RGB L1 为 0.0273，前景 PSNR 为 26.96 dB，是测试集中误差最低的样本。中栏与右栏仍可观察到叶片边缘锐度和局部投射阴影的差异，但大尺度颜色及受光关系已经得到较好重建。

4.5.4 侧向亮区迁移：light 029

图 9: 测试光照 029。左：优化前的普通单贴图 Billboard；中：优化后的双贴图 Billboard；右：相同方向光下的高模参考。

light 029 中，高模右侧树冠与顶部叶簇形成较明显的亮区。优化后的 Billboard 能够将亮度集中到相近位置，而不是保持初始化图像中的固定明暗。中栏部分叶片的亮度略高，树冠内部阴影也比高模平滑，说明当前法线与粗糙度模型已经学到光照方向变化。

4.5.5 低照度下的结构保持：light 031

图 10: 测试光照 031。左：优化前的普通单贴图 Billboard；中：优化后的双贴图 Billboard；右：相同方向光下的高模参考。

light 031 中，优化后结果保持了暗色树冠的整体基调，同时在顶部叶片、右侧叶簇和主干边缘保留了可辨识的亮度层次。其 RGB L1 从 0.0906 降至 0.0305，PSNR 提高约 8.55 dB。该样本说明优化并非简单地把整张颜色纹理提亮，而是能够根据输入光方向产生不同的局部响应。

4.6 完整测试集可视化

图 11 汇总了全部 8 个测试方向光。各行均保持相同的左、中、右顺序。优化前 Billboard 在多个光照下呈现近似的暗色图案；优化后 Billboard 的树冠亮区、树干明暗和叶簇对比随方向光

变化，并在整体趋势上接近对应高模参考。这组结果与误差曲线是一致的，说明训练得到的材质不只适配某一个展示样本。

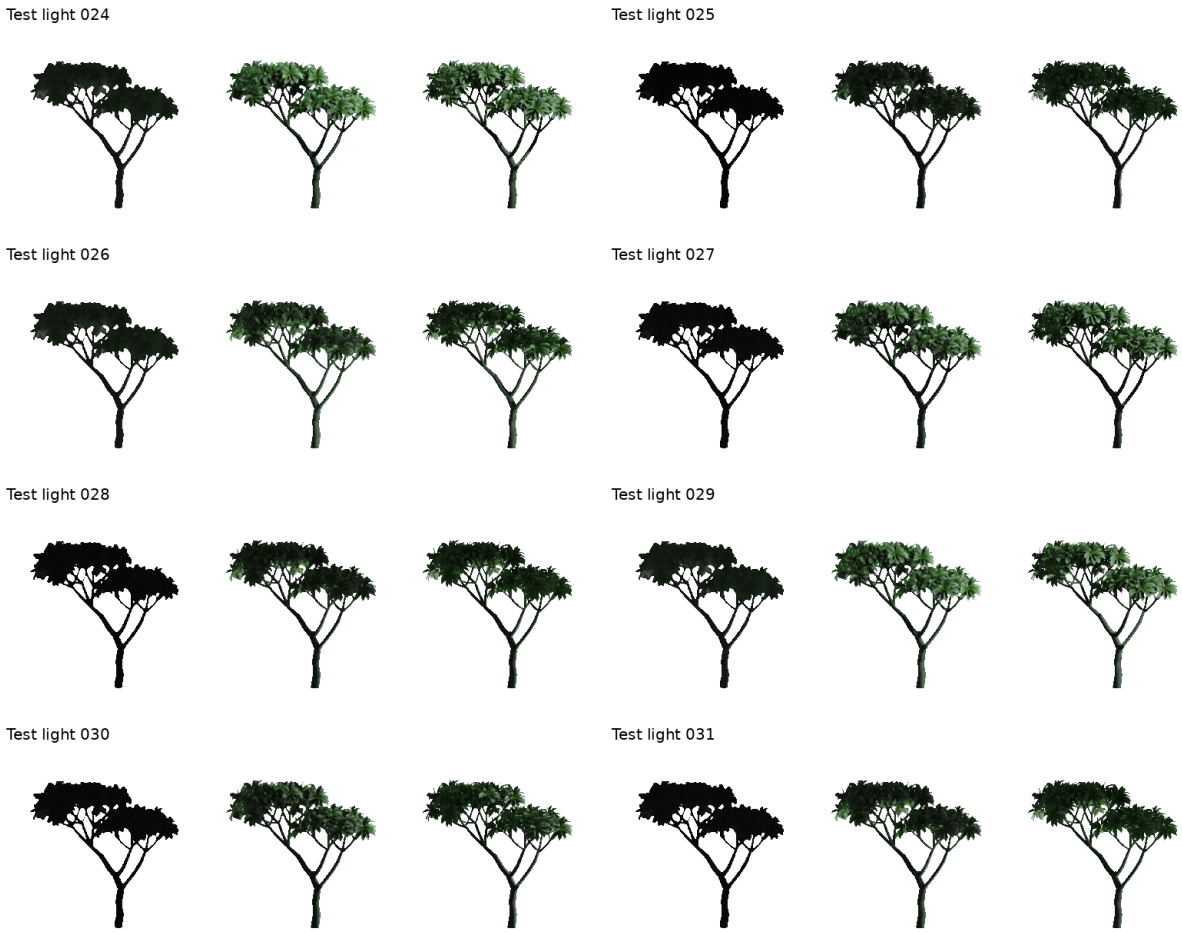

图 11: 8 个测试方向光的完整三栏结果。每个小图从左到右依次为优化前 Billboard、优化后 Billboard 和高模参考。

4.7 结果讨论与局限性

4.7.1 方向光响应的有效性

训练曲线、纹理通道、逐光照误差和三栏图构成了相互一致的证据链。损失下降说明纹理参数能够被梯度更新；法线与粗糙度通道分布产生了结构化，说明第二张纹理参与了材质的优化训练；8 个测试方向光上的误差均改善，说明结果具有基本的光照适应性；三栏图中亮区随光方向迁移，则直接体现了方向相关着色效果。

4.7.2 单平面表示的边界

高模参考由 Cycles 渲染，包含叶片间遮挡、枝条投射阴影和复杂材质效应。Billboard 着色器只计算常量环境项、漫反射和简化高光，每个像素仅保存一个代表法线。因此，树冠内部的多层叶片不能被完全还原，细枝遮挡和轮廓边缘也会比高模更平滑。light 026 和 light 029 中的局部差异主要来自这一表示能力限制。

5 结论与展望

5.1 工作总结

本文针对实时渲染中传统单贴图树木 Billboard 难以响应动态方向光的问题，在固定正交相机的实验条件下，实现了一套基于可微着色与可训练纹理的贴图优化流程。实验使用标准化流程生成仅改变方向光的树木参考数据集，将材质拆分为反照率透明度、法线粗糙度两张可训练纹理，借助 PyTorch 搭建完整可微着色管线，通过包含颜色、轮廓与平滑约束的复合损失完成迭代优化。经过 200 轮训练后模型损失显著收敛，在未参与训练的测试光照上，优化后 Billboard 相比初始单贴图方案误差大幅下降，视觉上能够跟随光源方向还原树冠与树干的明暗层次，有效改善了传统贴图光照固化的缺陷。同时实验也反映出单一平面 Billboard 本身存在表达上限，无法复刻高模多层叶片遮挡与精细自阴影。

5.2 后续改进方向

后续可以在着色模型中加入简易的环境遮蔽近似，进一步提升树冠内部暗部的还原效果；可将整套离线优化工具链对接游戏引擎，完成真实场景下的性能与画质对照测试，完善方案的工程落地能力。

5.3 课程实践认识

本次计算机图形学前沿课程实验完整走完了从数据生成、算法搭建到结果核验的小型研究流程，也让我收获了很多图形学研究层面的实践感悟。想要得到可靠的实验结论，单一变量控制的实验设计是重中之重，本次课题全程锁定相机、模型尺度、渲染采样等无关参数，只保留方向光作为唯一变量，所有视觉与数值差异都能清晰对应到光照响应能力上。

可微渲染是本次项目最核心的技术，传统 Billboard 烘焙只能得到静态混合光照的贴图，而可微着色管线能够把高模图像的视觉误差反向传播至纹理像素，自动分离混杂在一起的反照率、法线与光照信息，用数据驱动得到物理合理的材质参数，这种从高精度三维资产到轻量化实时二维表示的转化思路相比手工调整贴图高效得多，也让我理解了可微渲染在图形轻量化方向的实用价值。

同时我也意识到，图形算法不能只依赖单一的评价方式，仅看训练损失只能证明模型收敛，只靠肉眼对比图又缺少客观标准。本次实验结合了收敛曲线、纹理通道分解、量化误差与可视化对照多重指标来验证结论，从训练过程、材质结构、泛化能力和直观视觉多个角度互相证明，避免片面判断，完成了严谨的图形实验论证。

另外这次实践也让我看清学术原型和工业渲染之间存在距离，离线训练阶段我更侧重指标提升，但真正落地到游戏引擎时，显存、帧率、管线兼容都是必须兼顾的现实问题。今后再做图形相关的算法研究，我会提前把工程性能纳入评价考量，让算法方案更贴合真实开发场景的需求。

A 项目目录与代码

A.1 项目目录

```
Billboard/
|-- main.tex
|-- abstract.tex
|-- ch01_intro.tex
|-- ch02_design.tex
|-- ch03_impl.tex
|-- ch04_result.tex
|-- ch05_conclusion.tex
|-- appendix.tex
|-- refs.bib
|-- code/
| |-- README.md
| |-- data_collection/render_blender.py
| |-- workflow.py
| |-- dataset.py
| |-- shader.py
| |-- train.py
| |-- eval.py
| `-- tests/test_light_only_workflow.py
|-- dataset/tree/
| |-- light_000.png ... light_031.png
| `-- metadata.json
|-- outputs/tree/
| |-- checkpoints/baseline.pt
| |-- checkpoints/optimized.pt
| |-- comparisons/comparison_light_024.png ... 031.png
| |-- textures/albedo_alpha.png
| |-- textures/normal_roughness.png
| |-- metrics.json
| |-- report_metrics.json
| `-- training_curve.png
|-- figures/
| |-- dataset_lighting_gallery.png
| |-- texture_channel_analysis.png
| |-- rgb_l1_per_light.png
| |-- psnr_per_light.png
| `-- test_comparison_gallery.png
|-- references/
`-- main.pdf
```

A.2 核心代码索引

本附录列出论文中的核心文件。

A.2.1 方向光数据生成

Listing 1: 固定相机方向光数据生成脚本

```
"""Render one fixed camera view of a high-poly model under many sun directions.

Run with Blender: blender --background --python render_blender.py -- --model ... --output ...
"""
import argparse
import json
import math
import os
import sys

import bpy
import mathutils
sys.path.insert(0, os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
from workflow import world_light_to_billboard


def arguments():
    argv = sys.argv[sys.argv.index("--") + 1:] if "--" in sys.argv else []
    parser = argparse.ArgumentParser()
    parser.add_argument("--model", default=r"C:\Users\Zhangjingyi\Downloads\tree.glb")
    parser.add_argument("--output", default="../dataset/tree")
    parser.add_argument("--n_lights", type=int, default=32)
    parser.add_argument("--n_train", type=int, default=24)
    parser.add_argument("--res", type=int, default=256)
    parser.add_argument("--samples", type=int, default=32)
    parser.add_argument("--distance", type=float, default=4.0)
    return parser.parse_args(argv)


def fibonacci_hemisphere(count):
    golden = math.pi * (3.0 - math.sqrt(5.0))
    return [(math.cos(i * golden) * math.sqrt(1 - y * y), y, math.sin(i * golden))
            for i in range(count) for y in [0.15 + 0.85 * (i + 0.5) / count]]


def look_at(obj, target):
    obj.rotation_euler = (mathutils.Vector(target) - obj.location).to_track_quat("-Z", "Y").
        to_euler()


def load_and_normalize(path):
    bpy.ops.object.select_all(action="SELECT")
    bpy.ops.object.delete(use_global=False)
    if path.lower().endswith((".glb", ".gltf")):
        bpy.ops.import_scene.gltf(filepath=path)
```

```
    elif path.lower().endswith(".fbx"):
        bpy.ops.import_scene.fbx(filepath=path)
    else:
        bpy.ops.import_scene.obj(filepath=path)
    meshes = [obj for obj in bpy.context.scene.objects if obj.type == "MESH"]
    corners = [obj.matrix_world @ mathutils.Vector(corner) for obj in meshes for corner in obj.
        bound_box]
    minimum = mathutils.Vector(tuple(min(v[i] for v in corners) for i in range(3)))
    maximum = mathutils.Vector(tuple(max(v[i] for v in corners) for i in range(3)))
    center, scale = (minimum + maximum) / 2, 2.0 / max(maximum - minimum)
    # Preserve GLB parent transforms: transform the imported hierarchy as one unit.
    root = bpy.data.objects.new("NormalizedModelRoot", None)
    bpy.context.collection.objects.link(root)
    for obj in [item for item in bpy.context.scene.objects if item != root and item.parent is
        None]:
        obj.parent = root
    root.location = -center * scale
    root.scale = (scale, scale, scale)
    bpy.context.view_layer.update()
    return root


def choose_camera(camera, scene):
    """Select the horizontal view with the largest rendered tree silhouette."""
    original_engine = scene.render.engine
    original_resolution = (scene.render.resolution_x, scene.render.resolution_y)
    camera.data.type, camera.data.ortho_scale = "ORTHO", 2.6
    scene.render.engine = "BLENDER_WORKBENCH"
    scene.display.shading.light = "STUDIO"
    scene.display.shading.color_type = "MATERIAL"
    scene.render.resolution_x = scene.render.resolution_y = 128
    best_direction, best_pixels = None, -1
    for index in range(12):
        angle = 2.0 * math.pi * index / 12
        direction = mathutils.Vector((math.cos(angle), math.sin(angle), 0.18)).normalized()
        camera.location = direction * 4.0
        look_at(camera, (0.0, 0.0, 0.0))
        bpy.ops.render.render()
        pixels = bpy.data.images["Render Result"].pixels[:]
        visible = sum(1 for offset in range(3, len(pixels), 4) if pixels[offset] > 0.01)
        if visible > best_pixels:
            best_direction, best_pixels = direction, visible
    camera.location = best_direction * 4.0
    look_at(camera, (0.0, 0.0, 0.0))
    scene.render.engine = original_engine
    scene.render.resolution_x, scene.render.resolution_y = original_resolution
    return best_direction


```

```
def main():
    args = arguments()
    if not 0 < args.n_train < args.n_lights:
        raise ValueError("n_train must be between 1 and n_lights - 1")
    # Blender resets its process working directory on some Windows installs.
    # Resolve relative paths from `code/`, so `../dataset/tree` is reproducible.
    if not os.path.isabs(args.output):
        args.output = os.path.abspath(os.path.join(os.path.dirname(os.path.dirname(__file__)),
            args.output))
    os.makedirs(args.output, exist_ok=True)
    load_and_normalize(args.model)
    scene = bpy.context.scene
    scene.render.engine = "CYCLES"
    scene.cycles.samples, scene.cycles.use_denoising = args.samples, True
    scene.render.resolution_x = scene.render.resolution_y = args.res
    scene.render.resolution_percentage = 100
    scene.render.image_settings.file_format, scene.render.image_settings.color_mode = "PNG", "
        RGBA"
    scene.render.film_transparent = True
    camera_data = bpy.data.cameras.new("FixedCamera")
    camera = bpy.data.objects.new("FixedCamera", camera_data)
    bpy.context.collection.objects.link(camera)
    scene.camera = camera
    camera_direction = choose_camera(camera, scene)
    camera.location = camera_direction * args.distance
    look_at(camera, (0.0, 0.0, 0.0))
    light_data = bpy.data.lights.new("DirectionalLight", "SUN")
    light_data.energy = 3.0
    light = bpy.data.objects.new("DirectionalLight", light_data)
    bpy.context.collection.objects.link(light)
    records = []
    for index, direction in enumerate(fibonacci_hemisphere(args.n_lights)):
        # `direction` points from the shaded object toward the light source.
        light.location = mathutils.Vector(direction) * 5
        look_at(light, (0.0, 0.0, 0.0))
        filename = f"light_{index:03d}.png"
        scene.render.filepath = os.path.join(args.output, filename)
        bpy.ops.render.render(write_still=True)
        records.append({"image": filename, "light_idx": index,
                        "split": "train" if index < args.n_train else "test",
                        # Billboard coordinates: x=world x, y=world z, z=toward camera (-world y).
                        "light_dir": world_light_to_billboard(direction, camera_direction),
                        "camera": {"position": list(camera.location), "target": [0, 0, 0]}})
    with open(os.path.join(args.output, "metadata.json"), "w", encoding="utf-8") as handle:
        json.dump(records, handle, indent=2)


if __name__ == "__main__":
    main()
```

A.2.2 光方向坐标变换

Listing 2: 元数据划分与光方向坐标变换

```
"""Dependency-free metadata helpers for the fixed-camera experiment."""
import math


def split_metadata_by_light(metadata, train_count: int):
    """Split deterministically by light index; camera/view fields are irrelevant."""
    ordered = sorted(metadata, key=lambda item: item["light_idx"])
    return ordered[:train_count], ordered[train_count:]


def world_light_to_billboard(direction, camera_direction=(0.0, -1.0, 0.0)):
    """Express a world-space light vector in a billboard frame facing the camera."""
    def normalize(vector):
        length = math.sqrt(sum(value * value for value in vector))
        return [value / length for value in vector]
    normal = normalize(camera_direction)
    world_up = [0.0, 0.0, 1.0]
    right = normalize([world_up[1] * normal[2] - world_up[2] * normal[1],
                       world_up[2] * normal[0] - world_up[0] * normal[2],
                       world_up[0] * normal[1] - world_up[1] * normal[0]])
    up = [normal[1] * right[2] - normal[2] * right[1],
          normal[2] * right[0] - normal[0] * right[2],
          normal[0] * right[1] - normal[1] * right[0]]
    return [sum(a * b for a, b in zip(direction, axis)) for axis in (right, up, normal)]
```

A.2.3 数据加载

Listing 3: 固定相机方向光数据集

```
"""Dataset for one fixed camera under varying directional lights."""
import json
import os
from typing import Optional, Tuple

import cv2
import imageio.v2 as imageio
import numpy as np
import torch
from torch.utils.data import DataLoader, Dataset
from workflow import split_metadata_by_light


class BillboardDataset(Dataset):
    def __init__(self, root_dir: str, split: str, train_count: int = 24,
                 image_size: Optional[Tuple[int, int]] = None):
```

```
        if split not in {"train", "test", "all"}:
            raise ValueError("split must be train, test, or all")
        with open(os.path.join(root_dir, "metadata.json"), encoding="utf-8") as handle:
            records = json.load(handle)
        if records and "split" in records[0]:
            selected = [r for r in records if split == "all" or r["split"] == split]
        else:
            train, test = split_metadata_by_light(records, train_count)
            selected = {"train": train, "test": test, "all": train + test}[split]
        self.root_dir, self.records, self.image_size = root_dir, selected, image_size

    def __len__(self):
        return len(self.records)

    def __getitem__(self, index):
        record = self.records[index]
        image = imageio.imread(os.path.join(self.root_dir, record["image"])).astype(np.float32)
             / 255.0
        if image.ndim != 3 or image.shape[-1] not in (3, 4):
            raise ValueError(f"Expected RGB/RGBA image, got {image.shape}")
        if image.shape[-1] == 3:
            image = np.concatenate([image, np.ones_like(image[..., :1])], axis=-1)
        if self.image_size:
            height, width = self.image_size
            image = cv2.resize(image, (width, height), interpolation=cv2.INTER_LINEAR)
        direction = np.asarray(record["light_dir"], dtype=np.float32)
        direction /= max(float(np.linalg.norm(direction)), 1e-8)
        return {"image": torch.from_numpy(image[..., :3]),
                "alpha": torch.from_numpy(image[..., 3:4]),
                "light_dir": torch.from_numpy(direction),
                "light_idx": int(record["light_idx"])}


def make_dataloader(root_dir, batch_size=2, split="train", train_count=24,
                    image_size=None, num_workers=0):
    dataset = BillboardDataset(root_dir, split, train_count, image_size)
    return DataLoader(dataset, batch_size=batch_size, shuffle=split == "train",
                      num_workers=num_workers, drop_last=split == "train")
```

A.2.4 双贴图可微着色器

Listing 4: 固定相机 Billboard 可微着色器

```
"""A fixed-camera billboard with two trainable material textures."""
import torch
import torch.nn as nn
import torch.nn.functional as F


```

```python
class FixedCameraBillboardShader(nn.Module):
    def __init__(self, tex_resolution=128, image_resolution=256, ambient=0.15):
        super().__init__()
        self.image_resolution, self.ambient = image_resolution, ambient
        self.albedo_alpha = nn.Parameter(torch.full((1, 4, tex_resolution, tex_resolution), 0.5))
        normal_roughness = torch.zeros((1, 3, tex_resolution, tex_resolution))
        normal_roughness[:, 2].fill_(0.5)
        self.normal_roughness = nn.Parameter(normal_roughness)

    @torch.no_grad()
    def initialize_from_rgba(self, rgba: torch.Tensor):
        """Initialize the ordinary single-texture billboard from a reference RGBA image."""
        if rgba.ndim == 3:
            rgba = rgba.unsqueeze(0)
        rgba = rgba.permute(0, 3, 1, 2)
        texture = F.interpolate(rgba, self.albedo_alpha.shape[-2:], mode="bilinear", align_corners=False)
        self.albedo_alpha.copy_(torch.logit(texture.clamp(1e-4, 1.0 - 1e-4)))

    def forward(self, light_dir: torch.Tensor):
        batch = light_dir.shape[0]
        material = torch.sigmoid(F.interpolate(self.albedo_alpha, (self.image_resolution, self.image_resolution), mode="bilinear", align_corners=False))
        material = material.expand(batch, -1, -1, -1)
        albedo, alpha = material[:, :3], material[:, 3:4]
        nr = F.interpolate(self.normal_roughness, (self.image_resolution, self.image_resolution), mode="bilinear", align_corners=False).expand(batch, -1, -1, -1)
        xy = torch.tanh(nr[:, :2]) * 0.95
        z = torch.sqrt((1.0 - (xy * xy).sum(1, keepdim=True)).clamp_min(1e-6))
        normal = torch.cat([xy, z], dim=1)
        light = F.normalize(light_dir, dim=-1)[:, :, None, None]
        view = torch.tensor([0.0, 0.0, 1.0], device=light.device).view(1, 3, 1, 1)
        ndotl = (normal * light).sum(1, keepdim=True).clamp_min(0.0)
        half_vector = F.normalize(light + view, dim=1)
        roughness = torch.sigmoid(nr[:, 2:3]).clamp(0.08, 1.0)
        specular = (normal * half_vector).sum(1, keepdim=True).clamp_min(0.0).pow(2.0 / (roughness * roughness) - 2.0) * 0.08
        color = (albedo * (self.ambient + ndotl) + specular) * alpha
        return color.permute(0, 2, 3, 1), alpha.permute(0, 2, 3, 1)


# Backward-compatible name for local imports; its interface is intentionally light-only.
DifferentiableBillboardShader = FixedCameraBillboardShader
```

A.2.5 训练主程序

Listing 5: 双贴图优化训练程序

```
import argparse
import json
import os

import imageio.v2 as imageio
import matplotlib.pyplot as plt
import torch
import torch.nn.functional as F

from dataset import make_dataloader
from shader import FixedCameraBillboardShader


def composite_loss(pred, alpha, target, target_alpha):
    foreground = target_alpha.expand_as(target)
    rgb = (torch.abs(pred - target) * foreground).sum() / foreground.sum().clamp_min(1.0)
    mask = F.l1_loss(alpha, target_alpha)
    smooth = (torch.abs(pred[:, 1:] - pred[:, :-1]).mean() + torch.abs(pred[:, :, 1:] - pred[:, :, :-1]).mean())
    return rgb + 0.5 * mask + 0.02 * smooth, {"rgb_l1": float(rgb), "alpha_l1": float(mask)}


def save_texture(path, tensor):
    image = torch.sigmoid(tensor[0]).permute(1, 2, 0).detach().cpu().numpy()
    imageio.imwrite(path, (image.clip(0, 1) * 255).astype("uint8"))


def main():
    parser = argparse.ArgumentParser(description="Train a fixed-camera, light-varying billboard")
    parser.add_argument("--data_dir", required=True)
    parser.add_argument("--output_dir", default="../outputs/tree")
    parser.add_argument("--epochs", type=int, default=200)
    parser.add_argument("--batch_size", type=int, default=4)
    parser.add_argument("--tex_res", type=int, default=128)
    parser.add_argument("--img_res", type=int, default=256)
    parser.add_argument("--lr", type=float, default=0.01)
    parser.add_argument("--n_train", type=int, default=24)
    args = parser.parse_args()
    os.makedirs(args.output_dir, exist_ok=True)
    for name in ("textures", "checkpoints", "comparisons"):
        os.makedirs(os.path.join(args.output_dir, name), exist_ok=True)
    train_loader = make_dataloader(args.data_dir, args.batch_size, "train", args.n_train, (args.img_res, args.img_res))
    if not len(train_loader.dataset):
        raise RuntimeError("No training images found")
    shader = FixedCameraBillboardShader(args.tex_res, args.img_res)
    first = next(iter(train_loader))
```

```
    shader.initialize_from_rgba(torch.cat([first["image"][0], first["alpha"][0]], dim=-1))
    baseline_state = {key: value.detach().clone() for key, value in shader.state_dict().items()}
    torch.save(baseline_state, os.path.join(args.output_dir, "checkpoints", "baseline.pt"))
    optimizer, history = torch.optim.Adam(shader.parameters(), lr=args.lr), []
    for epoch in range(args.epochs):
        values = []
        for batch in train_loader:
            prediction, alpha = shader(batch["light_dir"])
            loss, metrics = composite_loss(prediction, alpha, batch["image"], batch["alpha"])
            optimizer.zero_grad(); loss.backward(); optimizer.step()
            values.append(float(loss.detach()))
        history.append({"epoch": epoch + 1, "loss": sum(values) / len(values), **metrics})
        print(f"epoch {epoch + 1:03d}/{args.epochs}: loss={history[-1]['loss']:.6f}")
    torch.save(shader.state_dict(), os.path.join(args.output_dir, "checkpoints", "optimized.pt"))
    save_texture(os.path.join(args.output_dir, "textures", "albedo_alpha.png"), shader.albedo_alpha)
    save_texture(os.path.join(args.output_dir, "textures", "normal_roughness.png"), shader.normal_roughness)
    with open(os.path.join(args.output_dir, "metrics.json"), "w", encoding="utf-8") as handle:
        json.dump(history, handle, indent=2)
    plt.plot([item["epoch"] for item in history], [item["loss"] for item in history])
    plt.xlabel("Epoch"); plt.ylabel("Loss"); plt.tight_layout()
    plt.savefig(os.path.join(args.output_dir, "training_curve.png"), dpi=180); plt.close()
    from eval import write_comparisons
    write_comparisons(args.data_dir, args.output_dir, args.tex_res, args.img_res, args.n_train)


if __name__ == "__main__":
    main()
```

A.2.6 对比图结果导出

Listing 6: 测试方向光三栏图导出程序

```
import argparse
import os

import imageio.v2 as imageio
import numpy as np
import torch

from dataset import make_dataloader
from shader import FixedCameraBillboardShader


def display(rgb, alpha):
    image = rgb.detach().cpu().numpy().copy()
```

```
    image[alpha.detach().cpu().numpy()[..., 0] < 0.5] = 1.0
    return image


@torch.no_grad()
def write_comparisons(data_dir, output_dir, tex_res=128, img_res=256, n_train=24):
    loader = make_dataloader(data_dir, 1, "test", n_train, (img_res, img_res))
    baseline = FixedCameraBillboardShader(tex_res, img_res)
    optimized = FixedCameraBillboardShader(tex_res, img_res)
    baseline.load_state_dict(torch.load(os.path.join(output_dir, "checkpoints", "baseline.pt"),
         map_location="cpu"))
    optimized.load_state_dict(torch.load(os.path.join(output_dir, "checkpoints", "optimized.pt"
        ), map_location="cpu"))
    comparison_dir = os.path.join(output_dir, "comparisons"); os.makedirs(comparison_dir,
        exist_ok=True)
    for batch in loader:
        before, before_alpha = baseline(batch["light_dir"])
        after, after_alpha = optimized(batch["light_dir"])
        reference = display(batch["image"][0], batch["alpha"][0])
        panel = np.concatenate([display(before[0], before_alpha[0]), display(after[0],
            after_alpha[0]), reference], axis=1)
        index = int(batch["light_idx"][0])
        imageio.imwrite(os.path.join(comparison_dir, f"comparison_light_{index:03d}.png"), (
            panel.clip(0, 1) * 255).astype("uint8"))


def main():
    parser = argparse.ArgumentParser(description="Export fixed-camera billboard comparisons")
    parser.add_argument("--data_dir", required=True)
    parser.add_argument("--output_dir", default="../outputs/tree")
    parser.add_argument("--tex_res", type=int, default=128)
    parser.add_argument("--img_res", type=int, default=256)
    parser.add_argument("--n_train", type=int, default=24)
    args = parser.parse_args()
    write_comparisons(args.data_dir, args.output_dir, args.tex_res, args.img_res, args.n_train)


if __name__ == "__main__":
    main()
```

B 结果文件说明

B.1 训练记录

`outputs/tree/metrics.json` 含 200 条记录。每条记录中的 `loss` 是该 epoch 所有训练批次损失的平均值。该文件首末值分别为 0.1254189362 和 0.0489781170。

B.2 测试误差复核

`outputs/tree/report_metrics.json` 是基于 `baseline.pt`、`optimized.pt` 和 8 张测试参考图重新计算的复核结果。其定义为：

- `rgb_l1`：仅在参考 alpha 前景区域统计的 RGB 平均绝对误差；
- `alpha_l1`：整幅图的 alpha 平均绝对误差；
- `foreground_psnr_db`：由参考前景区域 RGB 均方误差计算的峰值信噪比。

该文件同时保存每个测试方向光的单独结果，便于复核表 2。

参考文献

[1] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976. doi: 10.1145/360349.360354.

[2] Hugues Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH*, pages 99–108, 1996. doi: 10.1145/237170.237216.

[3] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH*, pages 209–216, 1997. doi: 10.1145/258734.258849.

[4] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *Proceedings of the European Conference on Computer Vision*, pages 154–169, 2014. doi: 10.1007/978-3-319-10584-0_11.

[5] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2018. doi: 10.1109/CVPR.2018.00411.

[6] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3D reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019. doi: 10.1109/ICCV.2019.00780.

[7] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics*, 39(6):1–14, 2020. doi: 10.1145/3414685.3417861.

[8] Nikhila Ravi, Jeremy Reizenstein, David Novotny, Taylor Gordon, Wan-Yen Lo, Justin Johnson, and Georgia Gkioxari. Accelerating 3D deep learning with PyTorch3D. *arXiv preprint arXiv:2007.08501*, 2020.